HOW 2026
Hello Open-source World

Logical Replication from Other Databases to PostgreSQL

Made Possible with SynchDB

Cary Huang

About Cary

HOW 2026
Hello Open-source World

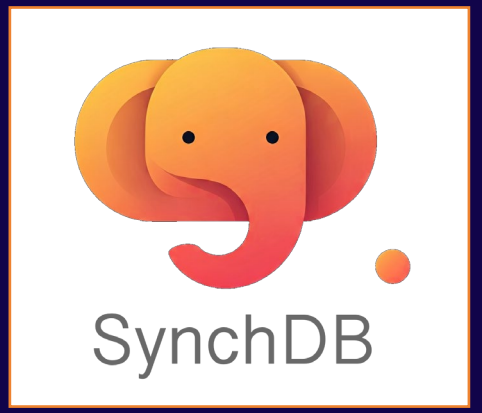
- Electrical Engineering graduate from University of British Columbia (UBC) in 2012
- Worked in the smart metering and energy sector right out of school.
- Joined Hornetlabs Technology in 2019 to start my PostgreSQL journey.
- Post-graduate instructor at Peking University to promote PostgreSQL open-source ecosystem in 2021.
- Worked on several aspects of PostgreSQL including sharding enhancements, distributed database, HA, shared storage, serverless, security..etc.
- Nominated as PostgreSQL Significant Contributor in 2025

Heterogeneous Database Replication with SynchDB

HOW 2026
Hello Open-source World

What is SynchDB?

An open-sourced PostgreSQL extension that enables logical replication from heterogeneous database systems like MySQL, SQLs server, Oracle and OpenLog Replicator directly to PostgreSQL!



Why Heterogeneous Database Replication is Hard?

We must deal with different:

- Data formats
- Semantics
- Toolings
- Protocols…etc

Real-time Change Data Capture (CDC) could be tricky:

- Complexity
- Drift
- Error handling strategies

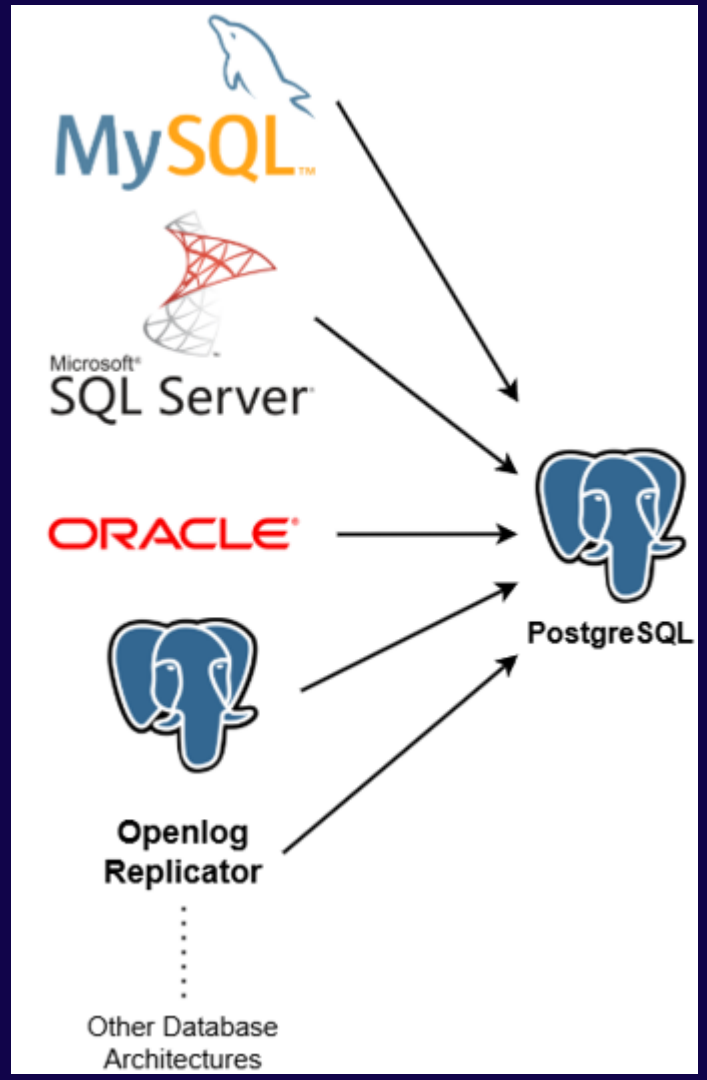
Schema evolution and type mapping headaches:

- Transaction boundaries and consistencies.
- Composite types.
- Changing table schemas.

Let's dive in!

Why PostgreSQL?

HOW 2026
Hello Open-source World



Analytics
PostgreSQL is a desirable and cost-effective choice for data consolidation for analytics, reporting, and data warehousing.

Cost Reduction
PostgreSQL is a powerful open-source database to migrate data to from legacy proprietary databases like SQLServer and Oracle.

Rich Ecosystem
PostgreSQL's extensive feature set, along with JSONB, GIS, geospatial, vector and full-text search, FDWs that other databases may lack.

Cloud Portable
PostgreSQL runs everywhere—AWS, Azure, GCP, on-prem—so your replicated data stays movable, consistent, and free from proprietary lock-in.

Other Options for HeterogeneouS Replication to PostgreSQL?

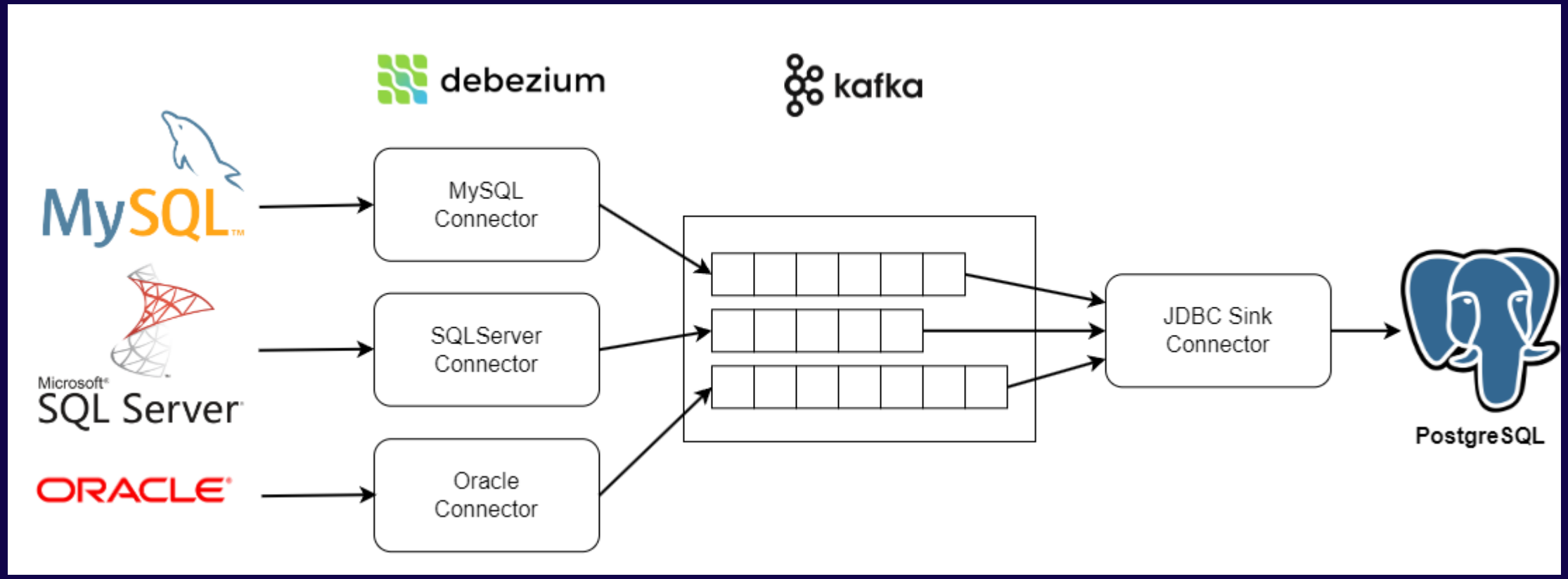
Advantages

- Real-time change data capture (CDC).
- Broad database support.
- Transactional consistency.
- Fault tolerance and scalability with Kafka.
- Decoupled architecture with Kafka in the middle.
- Open-source.

Disadvantages

- Complex setup (Kafka, Kafka Connect, Zookeeper, Kafka Sink)
- Resource intensive.
- Latency from multiple stages (Source -> Kafka topic -> Kafka Sink -> PostgreSQL)
- Schema management and data transform limitations.

Heterogeneous Replication via Debezium and Kafka



https://debezium.io/documentation/reference/stable/architecture.html

Other Options for Heterogeneous Database Replication to PostgreSQL?

HOW 2026
Hello Open-source World

Advantages

- Real-time Change Data Capture (CDC).
- Broad database support.
- Transactional consistency.
- Granular control and customization.
- Robust monitoring and management tools
- High availability topologies (active-active, active-passive).

Disadvantages

- High licensing and operational costs.
- Complex setup, configurations.
- High learning curve.
- Limited flexibility with non-Oracle databases.

Other Options for Heterogeneous Database Replication to PostgreSQL?

HOW 2026
Hello Open-source World

debezium

ORACLE GoldenGate

- Both can do real-time Change Data Capture (CDC).
- Both can ensure transactional consistency.
- Both can support broad database types.

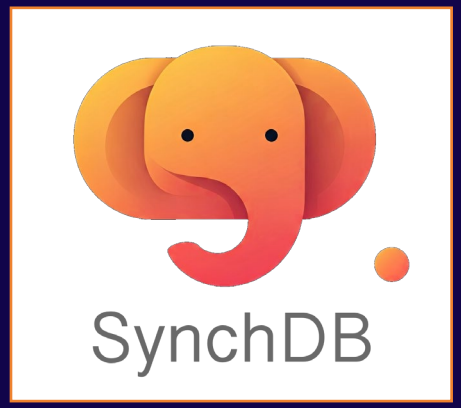
- They are not tailored to PostgreSQL (though it is supported).
- They require complex setups and intensive resource requirements.
- They have limited data transform flexibilities for PostgreSQL.

Is there an option more tailored to PostgreSQL?

Introducing ‘SynchDB’ - A PostgreSQL Extension

HOW 2026
Hello Open-source World

- SynchDB enables PostgreSQL to logically replicate data from heterogeneous database sources!
- Controlled and managed all within PostgreSQL.



Support MySQL, SQL Server, Oracle, PG Openlog Replicator.

Simple Setup and Installation

Flexible Data Transform Rules

Prometheus & Grafana Monitoring

Simple State and Stats Provisioning

High Performance

DDL and DML Replications

Flexible Error Strategies

Real-time Change Data Capture

Open-source and Community Support

Multi-Connector Architecture

Initial Data Sync & Migration

https://github.com/Hornetlabs/synchdb

How does SynchDB Work?

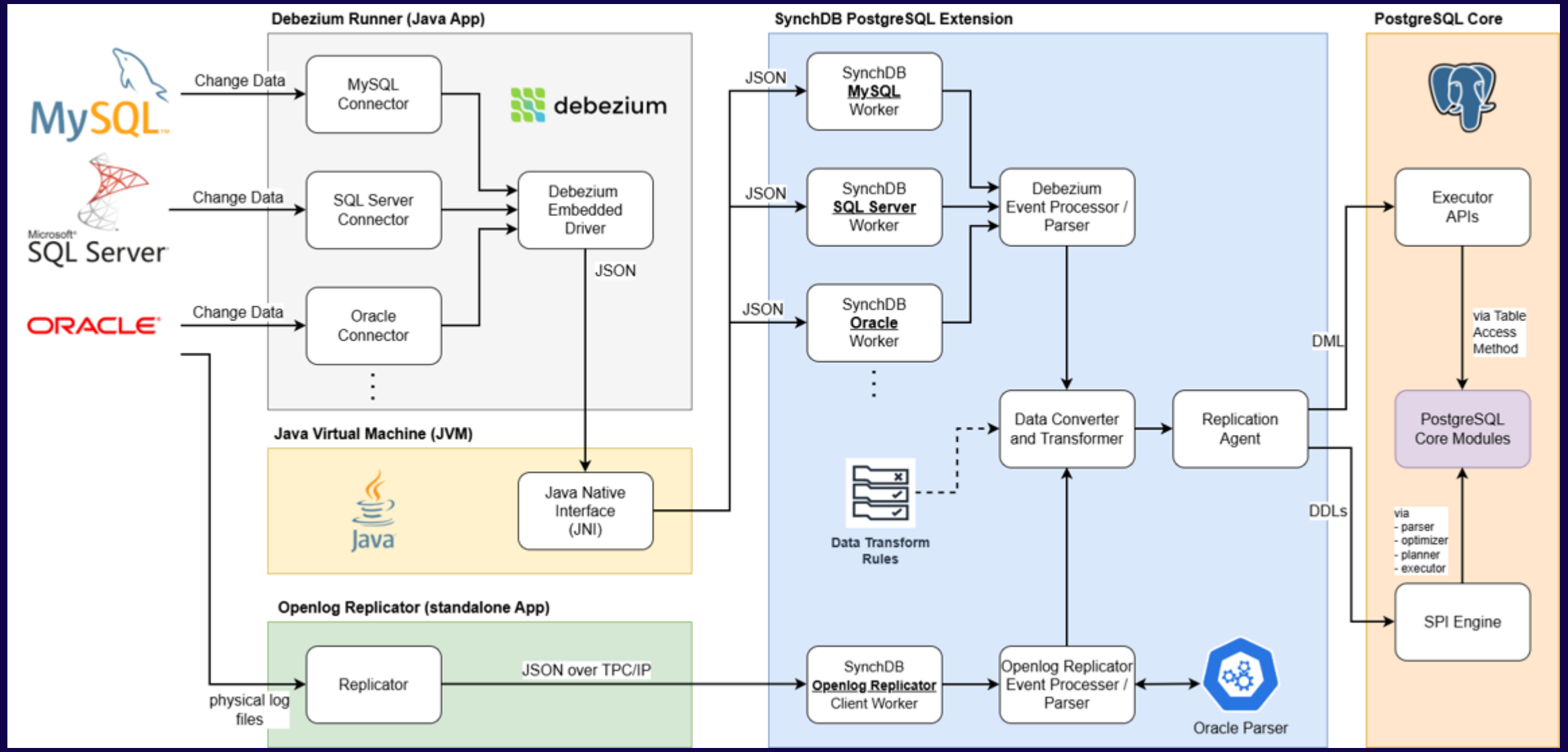
HOW 2026
Hello Open-source World

- Installed as PostgreSQL extension (C) with SQL-like controls.
- 3 ingestion paths:

Embedded Debezium Connector (DBZ)	Openlog Replicator (OLR)	Foreign Data Wrapper Data Snapshot (FDW)
• Initial Snapshot + CDC • Schema tracking + offset management **Java**	• Only CDC • Schema tracking + offset management **Native, Faster**	• Only Initial Snapshot **Native, Faster**

There is one problem though… Debezium is written in Java…

SynchDB Architecture



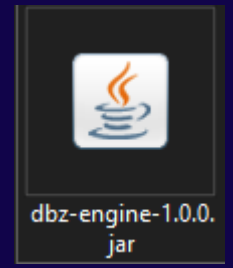
SynchDB Worker with JVM

- A PostgreSQL background worker and JVM do not run as 2 separate processes.
- JVM consists of several running threads inside the process (PostgreSQL background worker) that instantiates it.
- New threads may be spawned to run methods or services inside specified Java Archive (.jar) file.

This is a SynchDB background worker

```
ubuntu@synchdb:~/synchdb/postgres/contrib/synchdb$ ps -ef | grep post
ubuntu     441653       1  0 12:10 ?        00:00:00 /usr/local/pgsql/bin/postgres -D ../../synchdbtest17
ubuntu     441654  441653  0 12:10 ?        00:00:00 postgres: checkpointer
ubuntu     441655  441653  0 12:10 ?        00:00:00 postgres: background writer
ubuntu     441657  441653  0 12:10 ?        00:00:00 postgres: walwriter
ubuntu     441658  441653  0 12:10 ?        00:00:00 postgres: autovacuum launcher
ubuntu     441659  441653  0 12:10 ?        00:00:00 postgres: logical replication launcher
ubuntu     442474  441653 99 14:40 ?        00:01:30 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
ubuntu     442605  439158  0 14:42 pts/0    00:00:00 grep --color=auto post
```

Running this jar file




```
  0[||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||95.4%]   Tasks: 42, 66 thr; 2 running
  1[|||||||||||||||||                                                       22.7%]   Load average: 0.89 0.25 0.08
Mem[||||||||||||||||||||||||||||||                                      602M/7.75G]   Uptime: 4 days, 22:55:56
Swp[                                                                      0K/0K]

    PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%V  TIME+  Command
442474 ubuntu     20   0 3782M  355M 60536 R 117.  4.5  0:49.83 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442475 ubuntu     20   0 3782M  355M 60536 S 11.9  4.5  0:04.48 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442476 ubuntu     20   0 3782M  355M 60536 S  0.0  4.5  0:00.00 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442477 ubuntu     20   0 3782M  355M 60536 S  0.0  4.5  0:00.30 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442478 ubuntu     20   0 3782M  355M 60536 S  0.0  4.5  0:00.09 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442479 ubuntu     20   0 3782M  355M 60536 S  0.0  4.5  0:00.00 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442480 ubuntu     20   0 3782M  355M 60536 S  0.0  4.5  0:00.00 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442481 ubuntu     20   0 3782M  355M 60536 S  0.7  4.5  0:00.21 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442482 ubuntu     20   0 3782M  355M 60536 S  0.0  4.5  0:00.00 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442483 ubuntu     20   0 3782M  355M 60536 S  0.0  4.5  0:00.00 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442484 ubuntu     20   0 3782M  355M 60536 S  0.0  4.5  0:00.00 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442485 ubuntu     20   0 3782M  355M 60536 S  0.0  4.5  0:00.00 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442486 ubuntu     20   0 3782M  355M 60536 S  0.0  4.5  0:00.00 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442487 ubuntu     20   0 3782M  355M 60536 S  0.0  4.5  0:04.81 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442488 ubuntu     20   0 3782M  355M 60536 S  0.0  4.5  0:00.96 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442489 ubuntu     20   0 3782M  355M 60536 S  0.0  4.5  0:00.00 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442490 ubuntu     20   0 3782M  355M 60536 S  0.0  4.5  0:00.00 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442491 ubuntu     20   0 3782M  355M 60536 S 13.2  4.5  0:04.50 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442492 ubuntu     20   0 3782M  355M 60536 S  6.0  4.5  0:02.98 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442493 ubuntu     20   0 3782M  355M 60536 S  0.0  4.5  0:00.00 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442495 ubuntu     20   0 3782M  355M 60536 S  0.0  4.5  0:00.00 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442497 ubuntu     20   0 3782M  355M 60536 S  0.0  4.5  0:01.31 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442498 ubuntu     20   0 3782M  355M 60536 S  2.6  4.5  0:01.84 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
442499 ubuntu     20   0 3782M  355M 60536 S  0.0  4.5  0:00.01 postgres: synchdb engine: mysql@192.168.1.86:3306 -> postgres
```

JVM threads running inside a SynchDB background worker process

Java Native Interface (JNI)

HOW 2026
Hello Open-source World

- JNI is a C-based API exposed by the libjvm.so that bridges C and Java applications.
- It can create an instance of Java Virtual Machine (JVM) and run Java methods and constructs from C application.
- Comes with performance trade-offs comparing with native Java calls.
- Higher the call frequency, slower it becomes.

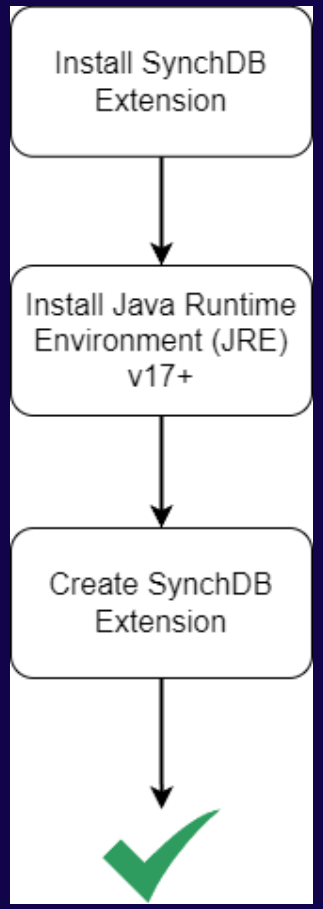
Operation	Native Java	1000 JNI Calls	1 million JNI Calls	1 billion JNI Calls
Simple arithmetic	1x	~3x – 5x slower	~10x – 20x slower	~15000x – 20000x slower
Empty method call	1x	~1x – 3x slower	~3x – 5x slower	~10000x – 15000x slower

⚠ Offload heavy work to native codes and reduce call frequency with batching.

SynchDB Setup – We Want it Simple

HOW 2026
Hello Open-source World

- We envision SynchDB to require minimum efforts to:
  - Install SynchDB to PostgreSQL.
  - Get started quickly.
- SynchDB requires JRE version 17+ to run.




```
$ sudo dpkg -i postgresql-16-synchdb_
    1.0-1.pgdg22.04_amd64.deb
```

```
$ sudo apt install openjdk-17-jre-headless

$ echo "/usr/lib/jvm/java-17-openjdk-amd64/lib/server/" |
    sudo tee /etc/ld.so.conf.d/java.conf

$sudo ldconfig
```

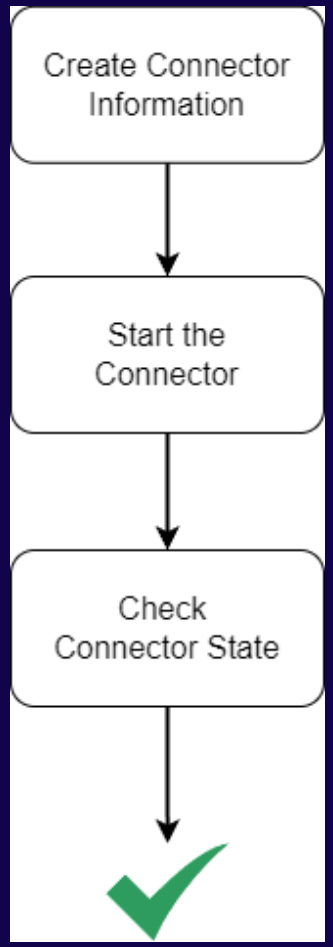
```
postgres=# CREATE EXTENSION SYNCHDB CASCADE;
```

Ready to go

Get Started with SynchDB – We Want it Simple Too!

HOW 2026
Hello Open-source World

```
SELECT synchdb_start_engine_bgw('mysqlconn');
```




```
SELECT synchdb_add_conninfo(
        'mysqlconn',    ← unique connector name
        '127.0.0.1',    ← remote hostname
        3306,           ← remote port
        'mysqluser',    ← remote username
        'mysqlpwd',     ← remote password
        'inventory',    ← remote source database
        'postgres',     ← destination database (PG)
        'inventory.orders',    ← table to replicate
        'inventory.orders',    ← table to snapshot
        'mysql');       ← connector type
```

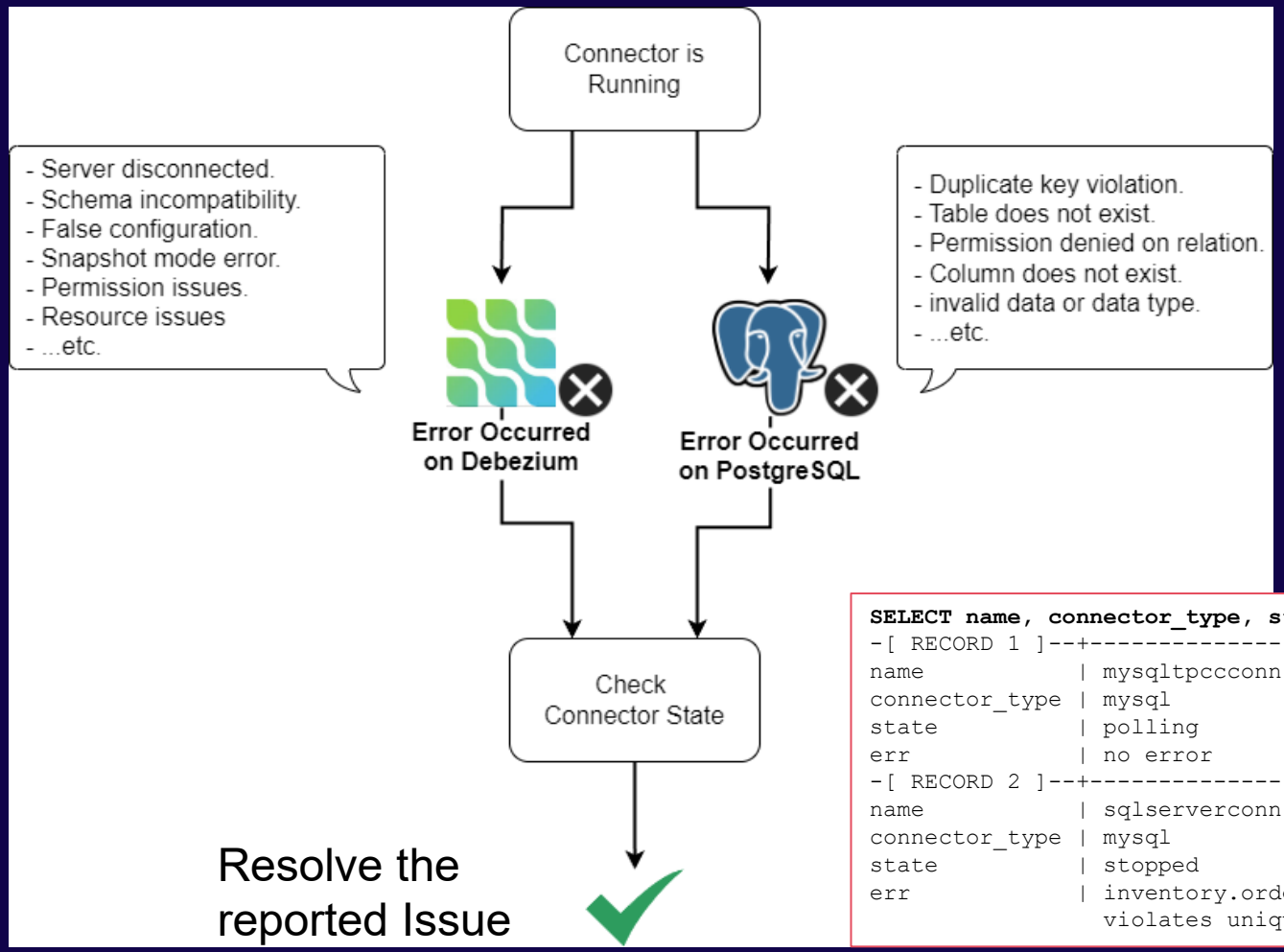
```
SELECT * from synchdb_state_view;
```

Change data incoming…

Provision Connector State, Stats and Error We Want it Straightforward!

HOW 2026
Hello Open-source World

```
SELECT * from synchdb_stats_view;
-[ RECORD 1 ]--+-------------
name           | mysqltpccconn
ddls           | 22
dmls           | 3887111
reads          | 3263746
creates        | 208684
updates        | 400241
deletes        | 14440
bad_events     | 14444
total_events   | 3901573
batches_done   | 2441
avg_batch_size | 1598
-[ RECORD 2 ]--+-------------
name           | sqlserverconn
ddls           | 17
dmls           | 278456
reads          | 376485
creates        | 173945
updates        | 64453
deletes        | 30047
bad_events     | 10011
total_events   | 278456
batches_done   | 3829
avg_batch_size | 465
```

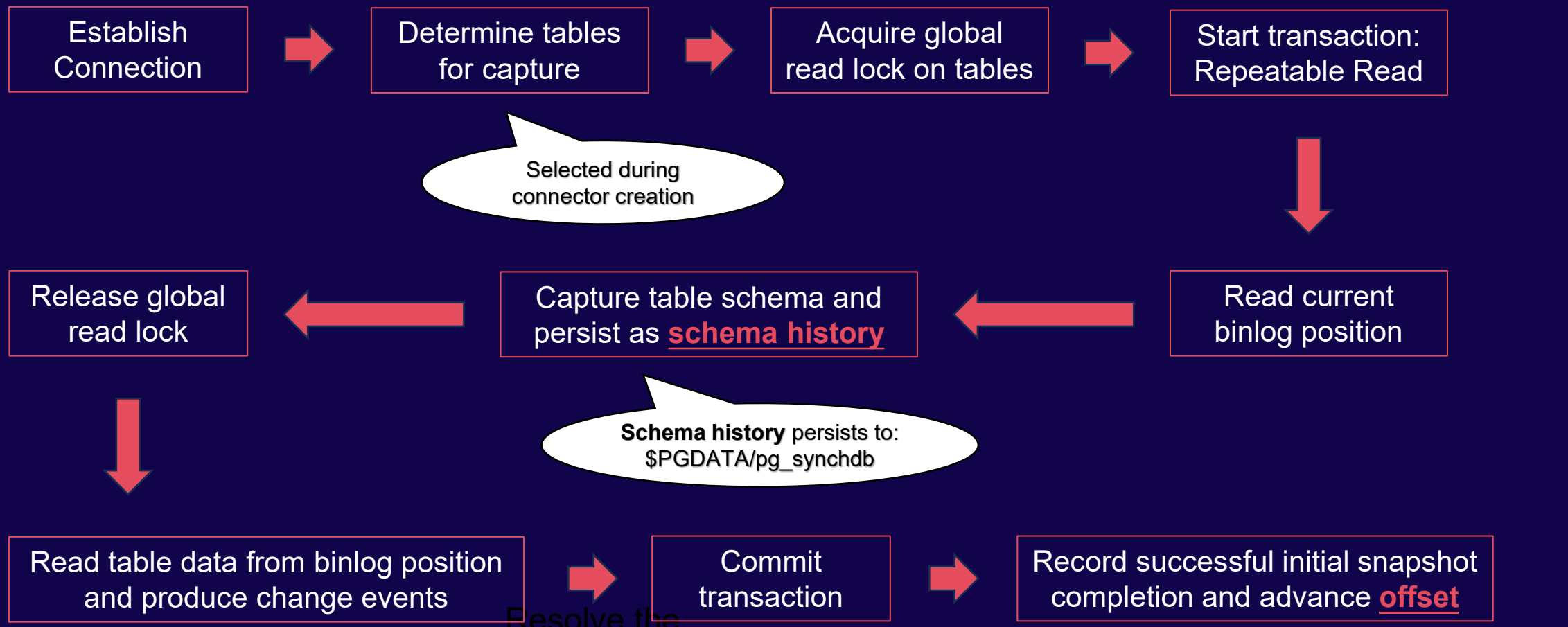



```
SELECT name, connector_type, state, err from synchdb_state_view;
-[ RECORD 1 ]--+-------------
name           | mysqltpccconn
connector_type | mysql
state          | polling
err            | no error
-[ RECORD 2 ]--+-------------
name           | sqlserverconn
connector_type | mysql
state          | stopped
err            | inventory.orders: duplicate key value
               | violates uniqueconstraint "orders_pkey"
```

Error propagation and coordination between C and Java space is important!

Initial Snapshot (TableSync) with Debezium

HOW 2026
Hello Open-source World



Initial Snapshot is performed when connector is started for the very first time

CDC follows immediately after that.

Initial Snapshot (TableSync) with FDW

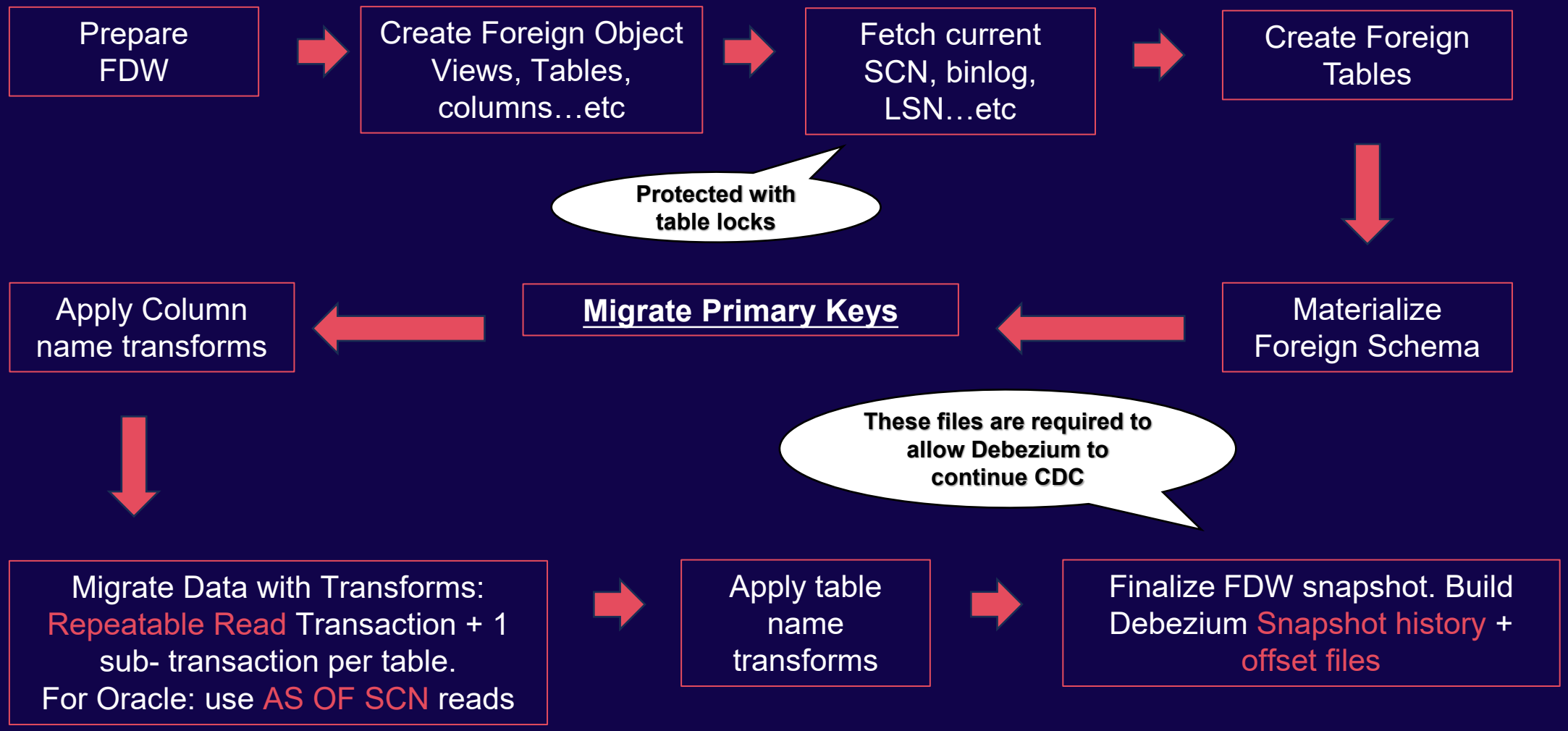
HOW 2026
Hello Open-source World

- Debezium Initial Snapshot works, but could be very slow when there are a lot of tables (thousands) to replicate.
- We could achieve similar snapshot using just FDWs with significant performance boost!
  - oracle_fdw
  - mysql_fdw
  - postgres_fdw

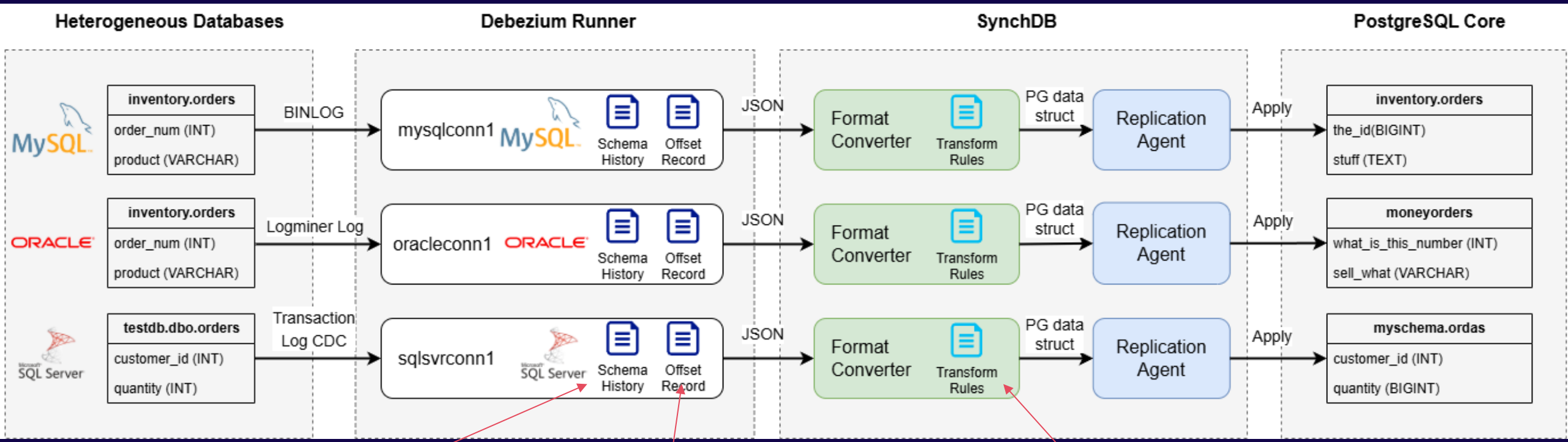
Challenge: How to ensure the system **“SELECT”** from foreign tables up to a certain SCN, LSN, binlog pos…etc

Initial Snapshot (TableSync) with FDW

HOW 2026
Hello Open-source World



SynchDB Workflow – Change Events



Schema history
contains table structure that Debezium requires to produce a data change event

Offset Record
contains the offset (similar to LSN in PG) that Debezium should start to read change records

Transform Rule
Tells SynchDB format converter how to transform incoming table, column, schema names and even the data before applying to PostgreSQL

Data Type Translations

HOW 2026
Hello Open-source World

```
postgres=# select * from synchdb_att_view where name='mysqlconn';
    name    | type  | attnum |         ext_tbname          |         pg_tbname           | ext_attname  |  pg_attname  | ext_atttypename |  pg_atttypename   | transform
------------+-------+--------+-----------------------------+-----------------------------+--------------+--------------+-----------------+-------------------+-----------
 mysqlconn  | mysql |      1 | inventory.addresses         | inventory.addresses         | id           | id           | int             | integer           |
 mysqlconn  | mysql |      2 | inventory.addresses         | inventory.addresses         | customer_id  | customer_id  | int             | integer           |
 mysqlconn  | mysql |      3 | inventory.addresses         | inventory.addresses         | street       | street       | varchar         | character varying |
 mysqlconn  | mysql |      4 | inventory.addresses         | inventory.addresses         | city         | city         | varchar         | character varying |
 mysqlconn  | mysql |      5 | inventory.addresses         | inventory.addresses         | state        | state        | varchar         | character varying |
 mysqlconn  | mysql |      6 | inventory.addresses         | inventory.addresses         | zip          | zip          | varchar         | character varying |
 mysqlconn  | mysql |      7 | inventory.addresses         | inventory.addresses         | type         | type         | enum            | text              |
 mysqlconn  | mysql |      1 | inventory.customers         | inventory.customers         | id           | id           | int             | integer           |
 mysqlconn  | mysql |      2 | inventory.customers         | inventory.customers         | first_name   | first_name   | varchar         | character varying |
 mysqlconn  | mysql |      3 | inventory.customers         | inventory.customers         | last_name    | last_name    | varchar         | character varying |
 mysqlconn  | mysql |      4 | inventory.customers         | inventory.customers         | email        | contact      | varchar         | character varying |
 mysqlconn  | mysql |      1 | inventory.geom              | inventory.geom              | id           | id           | int             | integer           |
 mysqlconn  | mysql |      2 | inventory.geom              | inventory.geom              | g            | g            | geometry        | geometry          |
 mysqlconn  | mysql |      3 | inventory.geom              | inventory.geom              | h            | h            | geometry        | text              |
 mysqlconn  | mysql |      1 | inventory.orders            | inventory.orders            | order_number | order_number | int             | integer           |
 mysqlconn  | mysql |      2 | inventory.orders            | inventory.orders            | order_date   | order_date   | date            | date              |
 mysqlconn  | mysql |      3 | inventory.orders            | inventory.orders            | purchaser    | purchaser    | int             | integer           |
 mysqlconn  | mysql |      4 | inventory.orders            | inventory.orders            | quantity     | quantity     | int             | integer           |
 mysqlconn  | mysql |      5 | inventory.orders            | inventory.orders            | product_id   | product_id   | int             | integer           |
 mysqlconn  | mysql |      1 | inventory.products          | public.stuff                | id           | id           | int             | integer           | %d + 1000
 mysqlconn  | mysql |      2 | inventory.products          | public.stuff                | name         | name         | varchar         | character varying |
 mysqlconn  | mysql |      3 | inventory.products          | public.stuff                | description  | description  | varchar         | character varying |
 mysqlconn  | mysql |      4 | inventory.products          | public.stuff                | weight       | weight       | float           | real              |
 mysqlconn  | mysql |      1 | inventory.products_on_hand  | inventory.products_on_hand  | product_id   | product_id   | int             | integer           |
 mysqlconn  | mysql |      2 | inventory.products_on_hand  | inventory.products_on_hand  | quantity     | quantity     | int             | integer           |
(25 rows)
```

SynchDB displays exactly how it “translates” a table, column or data type and allows an expression to be run before applying

SynchDB Workflow – Data Transform

HOW 2026
Hello Open-source World

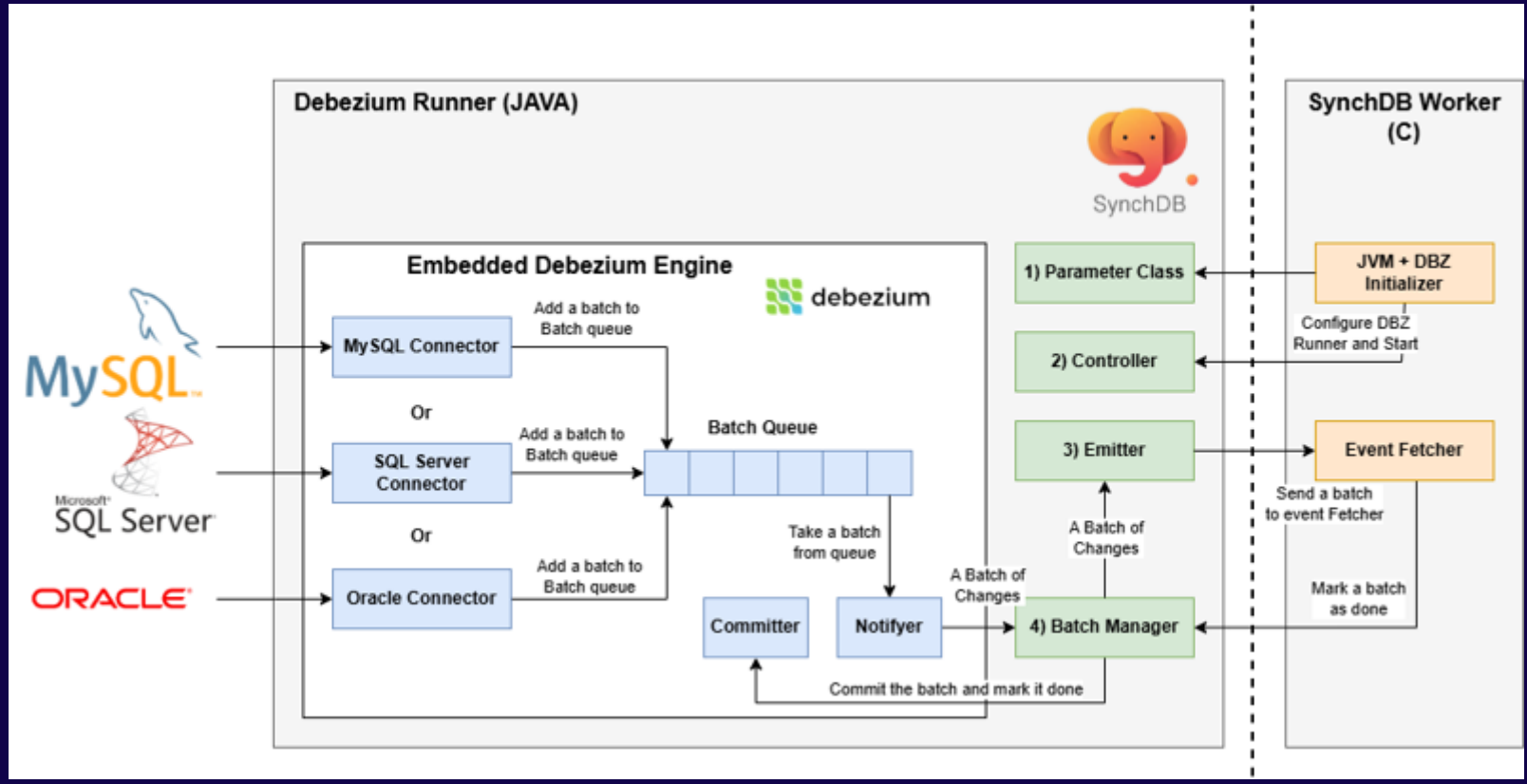
- The transform rules not only transforms between data types, table names and column names.
- It can do data expression transform as well.
- Useful when the replicated data needs further processing before applying to PostgreSQL.
- Not possible, or not flexible enough if we were built as a middleware outside of PostgreSQL.

Received column data is fed through user-defined expression before applying to PostgreSQL:

- %d replaced with data value
- %w replaced with WKB value (geometry type)
- %s replaced with SRID value (geometry type)

SynchDB Workflow – Fetch Changes With Batch Management

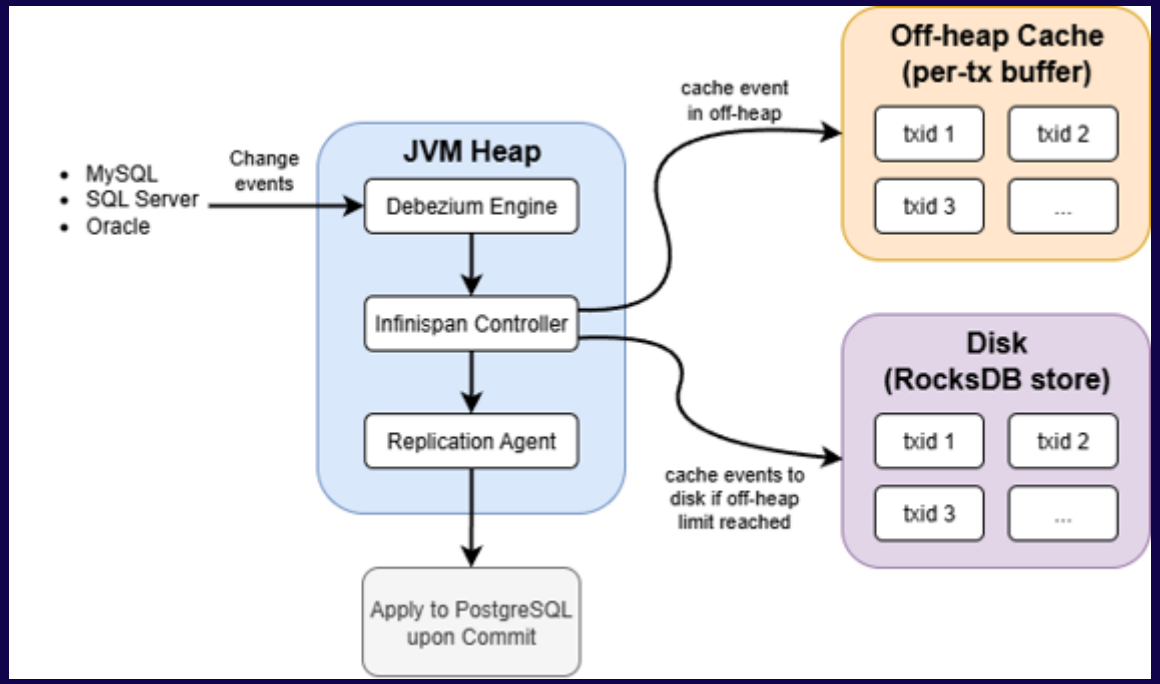
- A change record inside a batch is considered completed when it is successfully applied to PostgreSQL.
- A completed change record inside a batch allows Debezium to “advance the offset”.



Bigger batch sizes -> Less JNI Invocation -> Better performance

Large Transaction – Off Heap Buffering with Infinispan

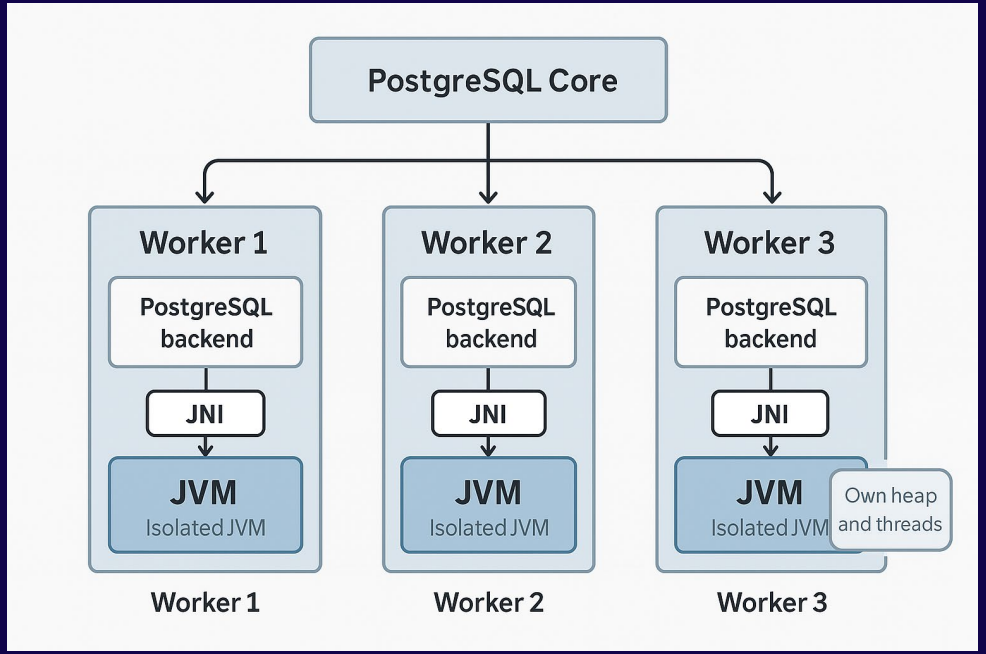
- Large transactions from MySQL, SQL server and Oracle must buffer until COMMIT.
- This can potentially blow JVM heap limit and cause exceptions.
- SynchDB can be configured with Infinispan to utilize off-heap memory (system heap) or disk to cache potentially large transactions before it commits.



Concurrency Concerns

HOW 2026
Hello Open-source World

- JVM is a single process runtime running in one OS process (java –jar) or in a PostgreSQL background worker (with JNI).
- You cannot start multiple JVM instances Inside one process.
- JVM can be shared by multiple threads within the same running process..
- PostgreSQL is a multi-process database system, and each process needs to create its own JVM instance to access Java in parallel and independently.



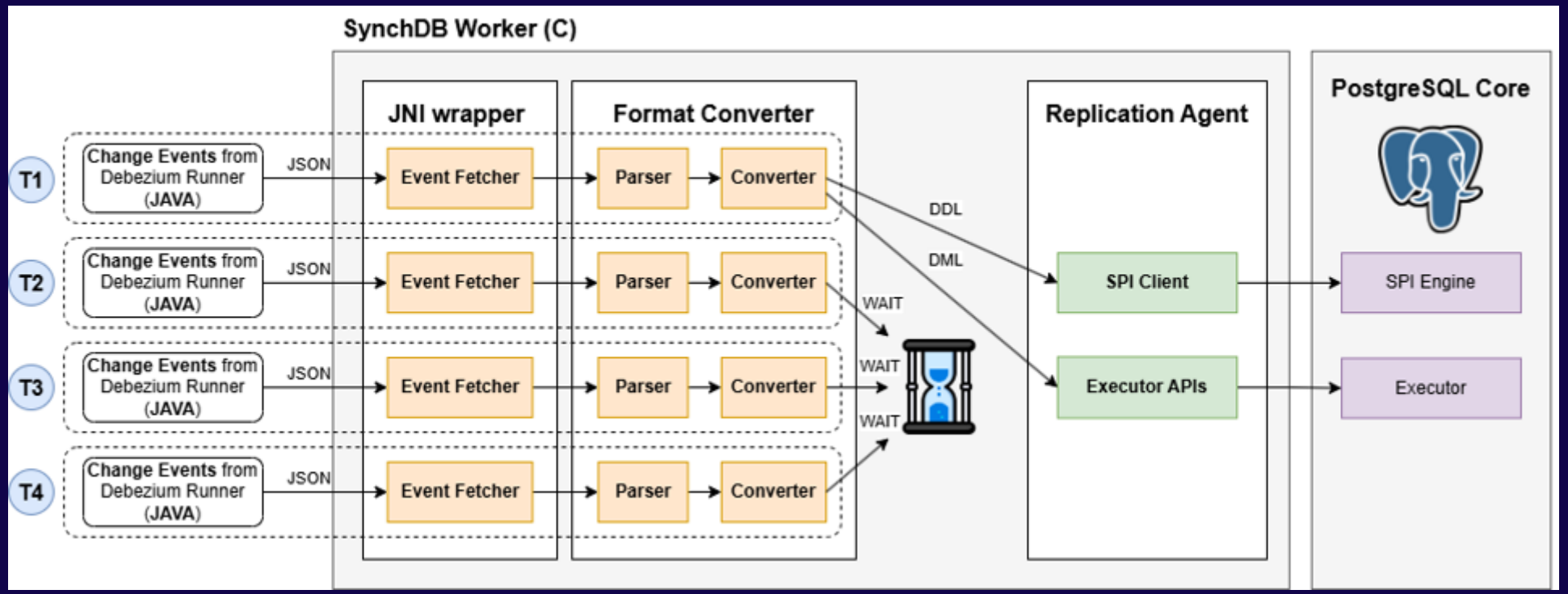
⚠ Multiple JVMs require careful memory planning to prevent resource exhaustion

Does Concurrency Help?

HOW 2026
Hello Open-source World

It depends….

- Threads should only parse/process JVM data — not access PostgreSQL directly!
- Threads cannot safely run transactions, access PostgreSQL catalogs or apply changes to PostgreSQL!



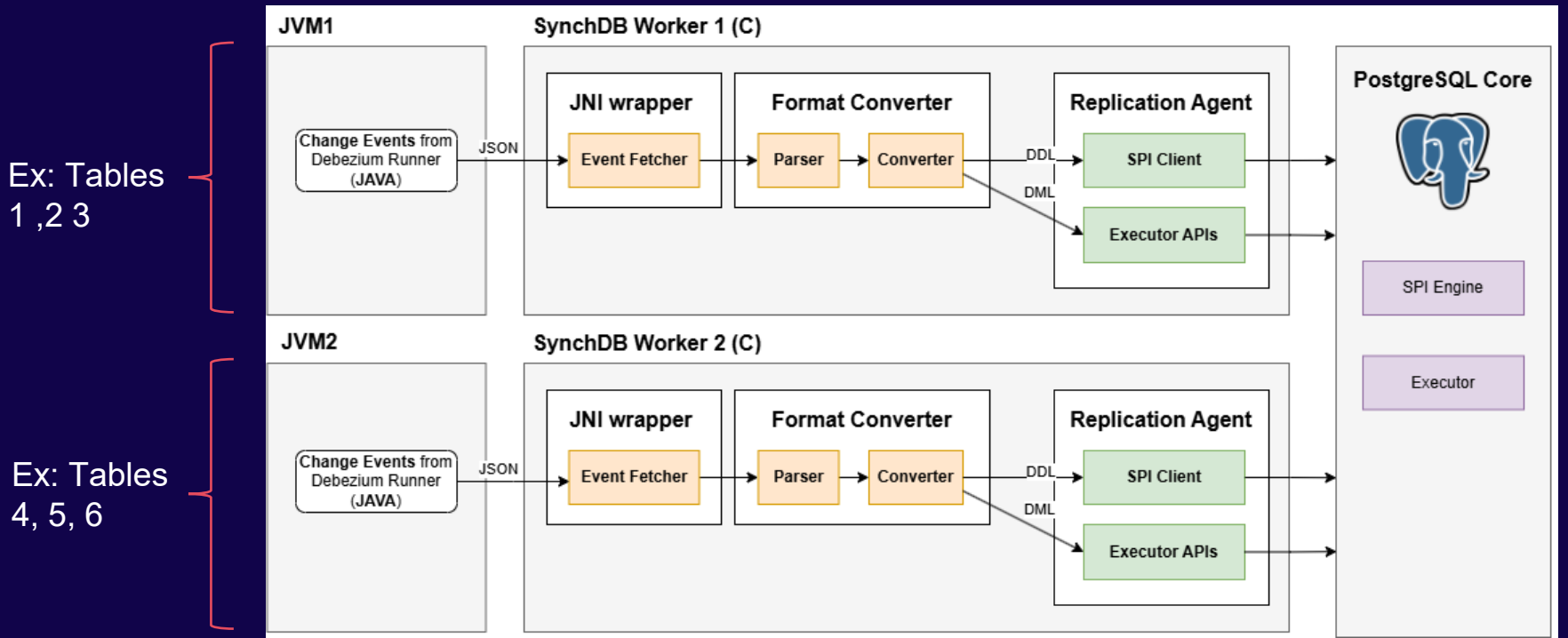
Would threads improve performance under these restrictions? Unlikely

Concurrency Concerns – Cont.

HOW 2026
Hello Open-source World

Safer concurrency?

- Spawn multiple SynchDB workers, each running a JVM to access Java resources and applying changes concurrently.
- Let each worker be responsible for different list of tables to prevent conflicts or deadlocks.
- Same table concurrency? More strict transaction boundary and ordering needs to be enforced = more work.



Performance could increase at a cost of running more JVMs

SynchDB v1.3

HOW 2026
Hello Open-source World

- SynchDB v1.3 is released on Nov 25, 2025.
- Github repository here: **https://github.com/Hornetlabs/synchdb.**
- Documentation site here: **https://docs.synchdb.com/.**
- New Contributors welcome!

Heterogeneous Databases:
- MySQL
- SQLServer
- Oracle
- Openlog Replicator

Transforms:
- Data types
- Table names
- Column names
- Data values

DDLs:
- CREATE/DROP TABLE
- ALTER TABLE ADD/DROP COLUMN
- ALTER TABLE ALTER COLUMN
- ALTER TABLE ADD/DROP CONSTRAINT

DMLs:
- INSERT
- UPDATE
- DELETE
- TRUNCATE

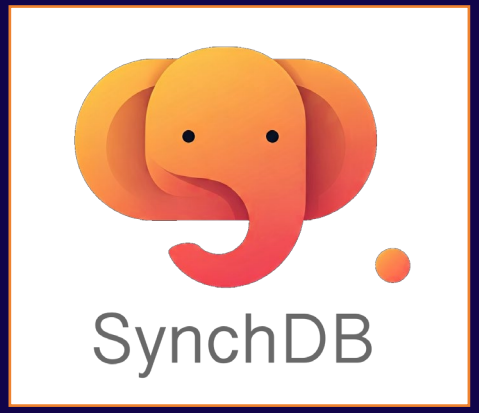
Apply Modes:
- SPI
- Executor APIs
- FDW initial snapshots

Replication Modes:
- Initial Snapshot
- Change Data Capture (CDC)

Snapshot Modes:
- Initial → schema + initial data + CDC
- Initial Only → schema + initial data
- Always → schema + initial data + CDC (always)
- No data → schema + CDC
- Never → CDC only

Monitoring:
- JMX Mbeans
- Prometheus and Grafana

Quick Start:
- Ezdeploy utility




HOW 2026
Hello Open-source World
Thanks