HOW 2026
Hello Open-source World

Logical Replication from Other Databases to PostgreSQL

Made Possible with SynchDB

Cary Huang

关于Cary

HOW 2026
Hello Open-source World

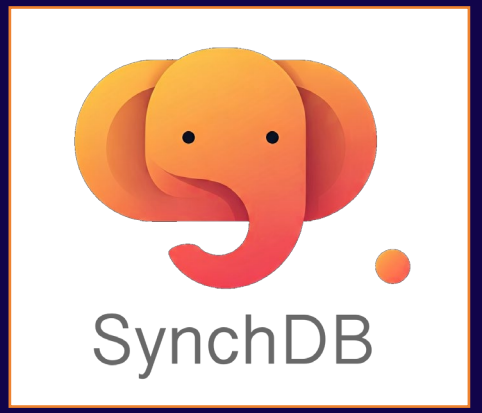
- 2012年畢業於不列顛哥倫比亞大學（UBC），獲電機工程學位。
- 畢業後直接進入只能电表和能源產業工作。
- 2019 年加入Hornetlabs Technology，開啟我的PostgreSQL之旅。
- 2021 年在北京大學擔任研究生導師，致力於推廣PostgreSQL開源生態系統。
- 參與過 PostgreSQL 的多個面向的研究，包括分片增強、分布式数据、高可用性、共享儲存、無服务器架構、安全性等等。
- 2025 年被列为 PostgreSQL Contributor

使用 SynchDB 进行异构数据库复制

HOW 2026
Hello Open-source World

什么是 SynchDB?

SynchDB 是一款开源的 PostgreSQL 扩展，它能够把 MySQL、 SQL Server、Oracle 和 OpenLog Replicator 等异构数据库系统逻辑数据直接复制到 PostgreSQL！



为什么异构数据库复制如此困难?

我们必须处理不同的：

- 数据格式
- 语义
- 工具
- 协议……等等。

Let’s dive in!

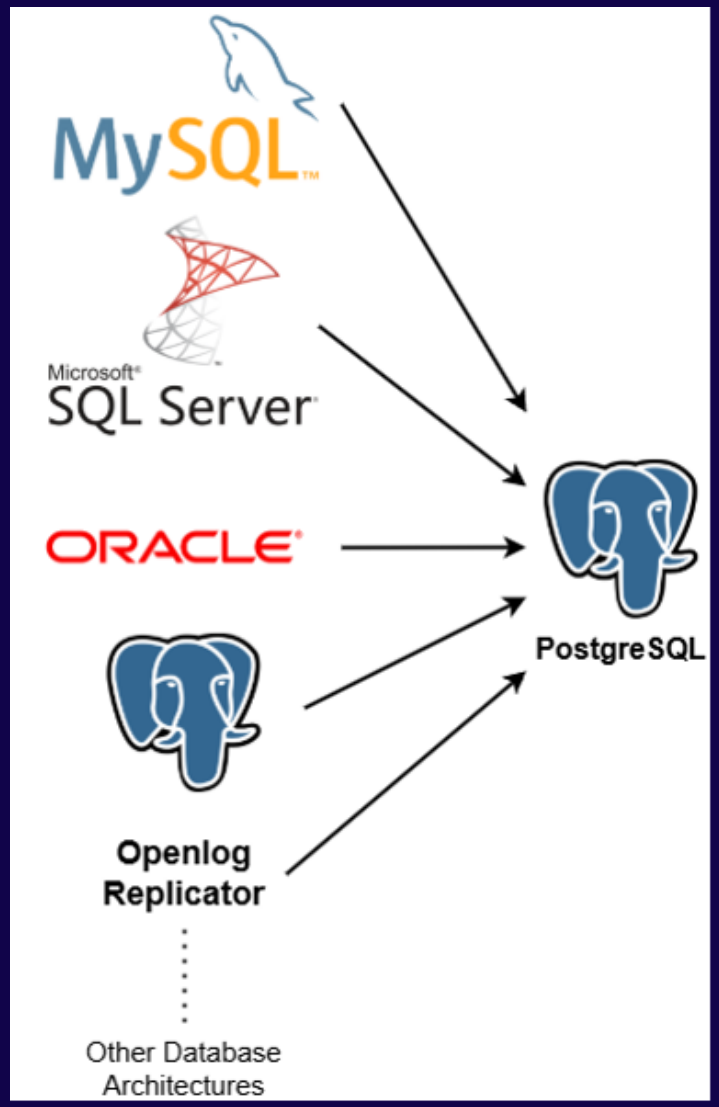
实时变更数据采集 (CDC) 可能面临诸多挑战：

- 复杂性
- 漂移
- 错误处理策略

Schema 变化和类型映射难题：

- 事务边界和一致性。
- 复合类型。
- 表schema变更。

为什么选择PostgreSQL？



分析
PostgreSQL 是数据整合的理想选择，兼具成本效益，适用于分析、报告和数据仓库。

降低成本
PostgreSQL 是一个功能强大的开源数据库，可用于将数据从 SQL Server 和 Oracle 等传统专有数据库迁移到该数据库。

丰富的生态系统
PostgreSQL 拥有其他数据库可能缺乏的丰富功能集，包括 JSONB、GIS、地理空间、矢量和全文搜索等功能。

云端便携性
PostgreSQL 可在任何地方运行——AWS、Azure、GCP、本地部署——因此您的复制数据保持可移动性、一致性，并且不受专有软件锁定限制。

还有其他异构复制到 PostgreSQL 的方案吗?

HOW 2026
Hello Open-source World

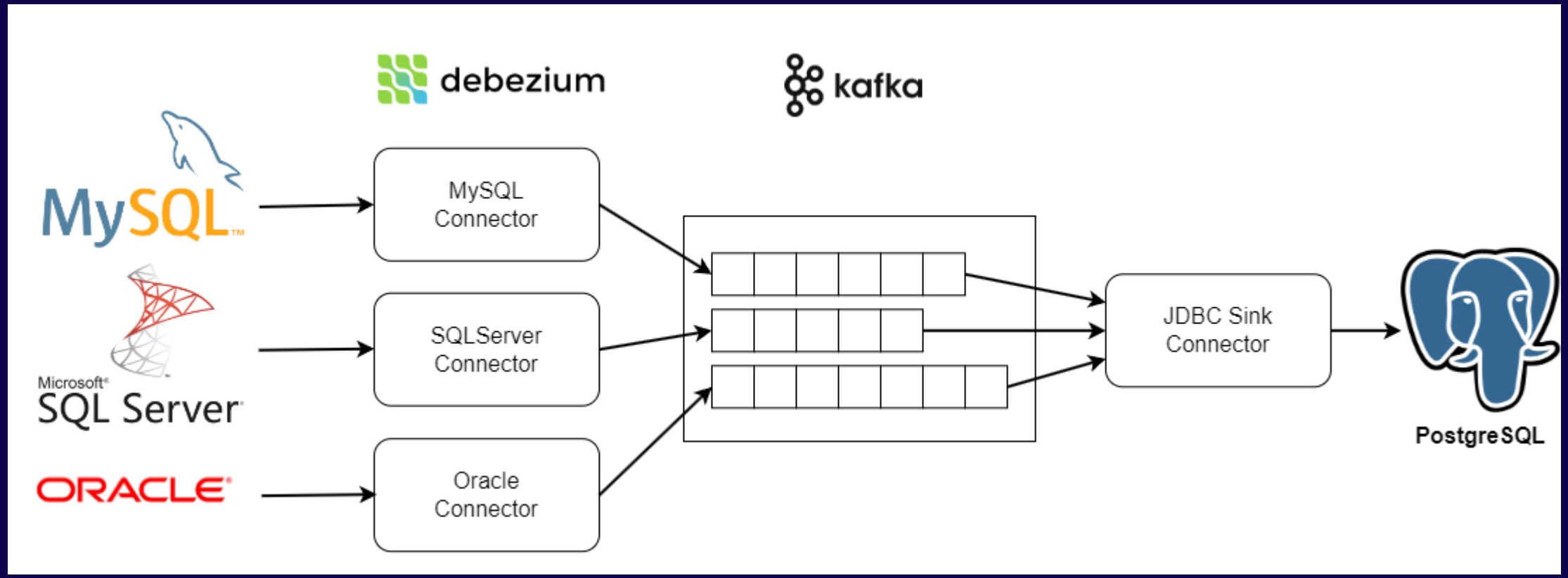
优势

- 实时变更数据捕获 (CDC)。
- 广泛的数据库支持。
- 事务一致性。
- 借助 Kafka 实现容错和可扩展性。
- 以 Kafka 为中间层的解耦架构。
- 开源。

劣势

- 复杂的配置（Kafka、 Kafka Connect、 Zookeeper、 Kafka Sink）。
- 资源需求高。
- 多阶段延迟（源 -> Kafka topic -> Kafka Sink -> PostgreSQL）。
- Schema管理和数据转换方面的限制。

通过 Debezium 和 Kafka 实现异构复制



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

还有其他异构数据库复方案吗?

优势

- 实时变更数据捕获 (CDC)。
- 广泛的数据库支持。
- 事务一致性。
- 精细的控制和自定义。
- 强大的监控和管理工具。
- 高可用性拓扑结构（双活、双活）。

劣势

- 高昂的 license 和运营成本。
- 复杂的设置和配置。
- 学习需要花点时间 – high learning curve。
- 对非 Oracle 数据库的兼容性有限。

还有其他异构数据库复方案吗?

HOW 2026
Hello Open-source World

- 两者都支持实时变更数据捕获 (CDC)。
- 两者都能确保事务一致性。
- 两者都支持多种数据库类型。

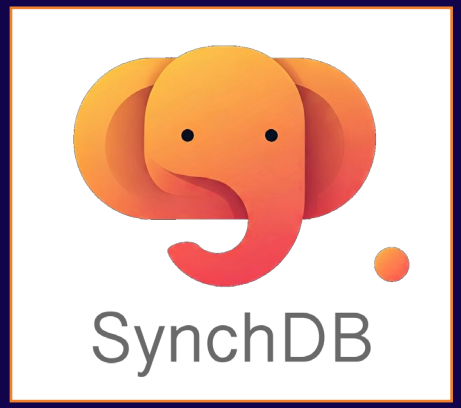
- 它们并非专为 PostgreSQL 定制（尽管支持 PostgreSQL）。
- 它们需要复杂的配置和大量的资源投入。
- 它们对 PostgreSQL 的数据转换灵活性有限。

是否有更适合 PostgreSQL 的选项?

隆重推出“SynchDB”，一个 PostgreSQL 扩展

HOW 2026
Hello Open-source World

- SynchDB 使 PostgreSQL 能够从异构数据库源进行逻辑数据复制！
- 所有操作均在 PostgreSQL 内部控制和管理。



支持 MySQL、SQL Server、Oracle、PG Openlog Replicator

安装简单

灵活的数据转换规则

Prometheus & Grafana 监控

简单明了的状态和统计信息

高性能

DDL 和 DML 复制

灵活的错误应对策略

实时 Change Data Capture （CDC）

开源和社区支持

多连接器架构架构

初始数据同步与迁移

https://github.com/Hornetlabs/synchdb

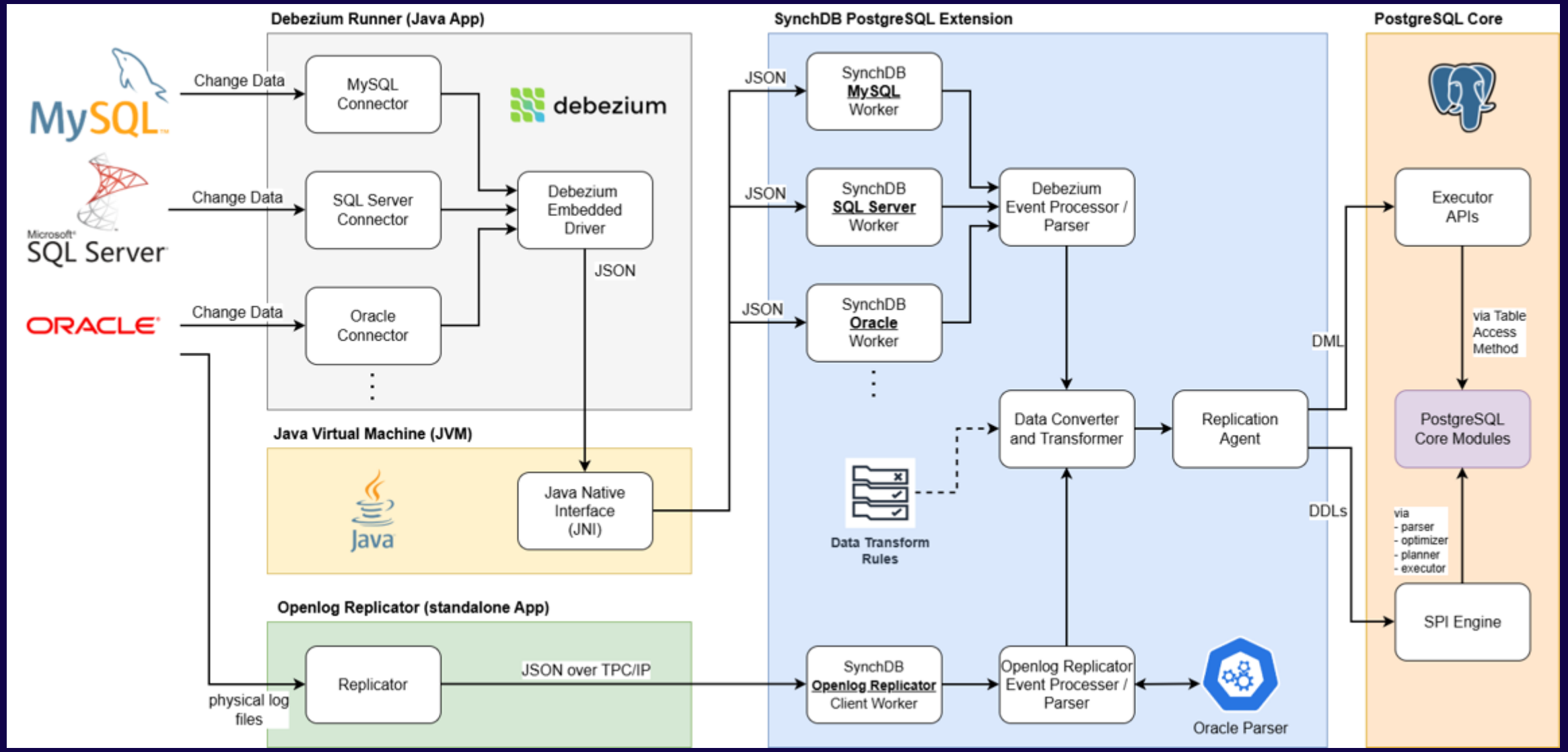
SynchDB 怎么工作的?

- 以 PostgreSQL 扩展 (C) 的形式安装，使用 SQL 函数做功能控制。
- 3 个数据导入路径：

嵌入式 Debezium 连接器 (DBZ)	Openlog Replicator (OLR)	Foreign Data Wrapper (FDW)
• 初始快照 + CDC • shcema跟踪 + 偏移量管理 **Java**	• 仅限 CDC • Schema 跟踪 + 偏移量管理 **原生，速度更快**	• 仅初始快照 **原生，速度更快**

这里有个问题……Debezium 是用 Java 编写的……

SynchDB 架构


Debezium Runner (Java App)
SynchDB PostgreSQL Extension
PostgreSQL Core
MySQL
Change Data
MySQL Connector
debezium
Microsoft SQL Server
Change Data
SQL Server Connector
Debezium Embedded Driver
JSON
ORACLE
Change Data
Oracle Connector
Java Virtual Machine (JVM)
Java
Java Native Interface (JNI)
JSON
SynchDB MySQL Worker
JSON
SynchDB SQL Server Worker
JSON
SynchDB Oracle Worker
Debezium Event Processor / Parser
Data Converter and Transformer
Replication Agent
Data Transform Rules
DML
DDLs
Executor APIs
via Table Access Method
PostgreSQL Core Modules
via
- parser
- optimizer
- planner
- executor
SPI Engine
Openlog Replicator (standalone App)
physical log files
Replicator
JSON over TPC/IP
SynchDB Openlog Replicator Client Worker
Openlog Replicator Event Processer / Parser
Oracle Parser

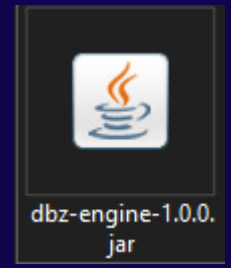
SynchDB Worker 与 JVM

- PostgreSQL 后台工作进程和 JVM 并非作为两个独立的进程运行。
- JVM 由多个运行线程组成，这些线程位于实例化的进程（PostgreSQL 后台工作进程）内部。
- 可以创建新线程来运行指定 Java 归档文件（.jar）中的方法或服务。

这是一个 SynchDB 后台工作进程

```
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
```

运行此 jar 文件




```
  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
```

SynchDB 后台工作进程中运行的 JVM 线程

Java Native Interface (JNI)

- JNI 是由 libjvm.so 库提供的基于 C 语言的 API，用于连接 C 和 Java 应用程序。
- 它可以创建 Java 虚拟机 (JVM) 实例，并从 C 应用程序运行 Java 方法和结构。
- 与原生 Java 调用相比，JNI 的性能有所妥协。
- 调用频率越高，速度越慢。

Operation	Native Java	1000 JNI Calls	1 million JNI Calls	1 billion JNI Calls
简单的算术	1x	~3x – 5x slower	~10x – 20x slower	~15000x – 20000x slower
空方法调用	1x	~1x – 3x slower	~3x – 5x slower	~10000x – 15000x slower

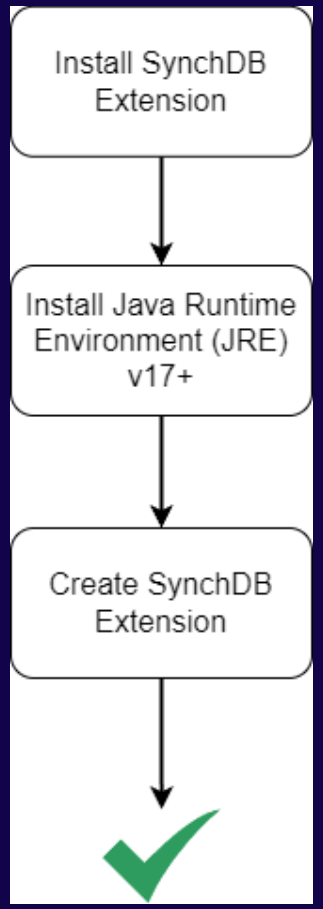
⚠ 将繁重的工作卸载到本地代码，并通过批处理减少调用频率。

SynchDB 设置 – 我们希望它简单易用

HOW 2026
Hello Open-source World

- 我们设想 SynchDB 只需极少的操作即可完成：
  - 将 SynchDB 安装到 PostgreSQL。
  - 快速上手。

- SynchDB 需要 JRE 版本 17 或更高版本才能运行。




```
$ 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
```

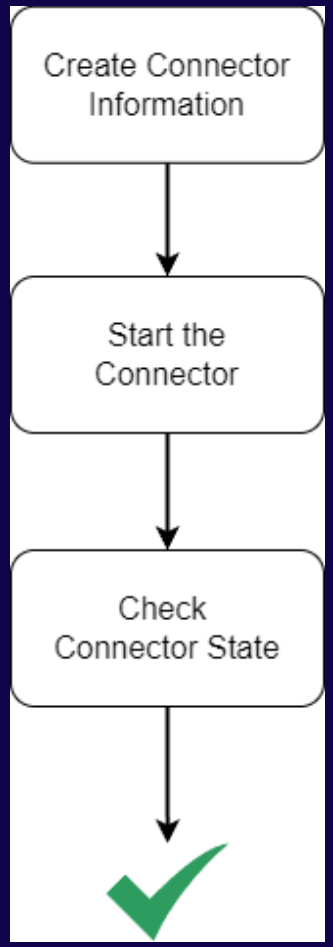
```
postgres=# CREATE EXTENSION SYNCHDB CASCADE;
```

Ready to go

立即开始使用 SynchDB – 我们也希望它简单易用!

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
```

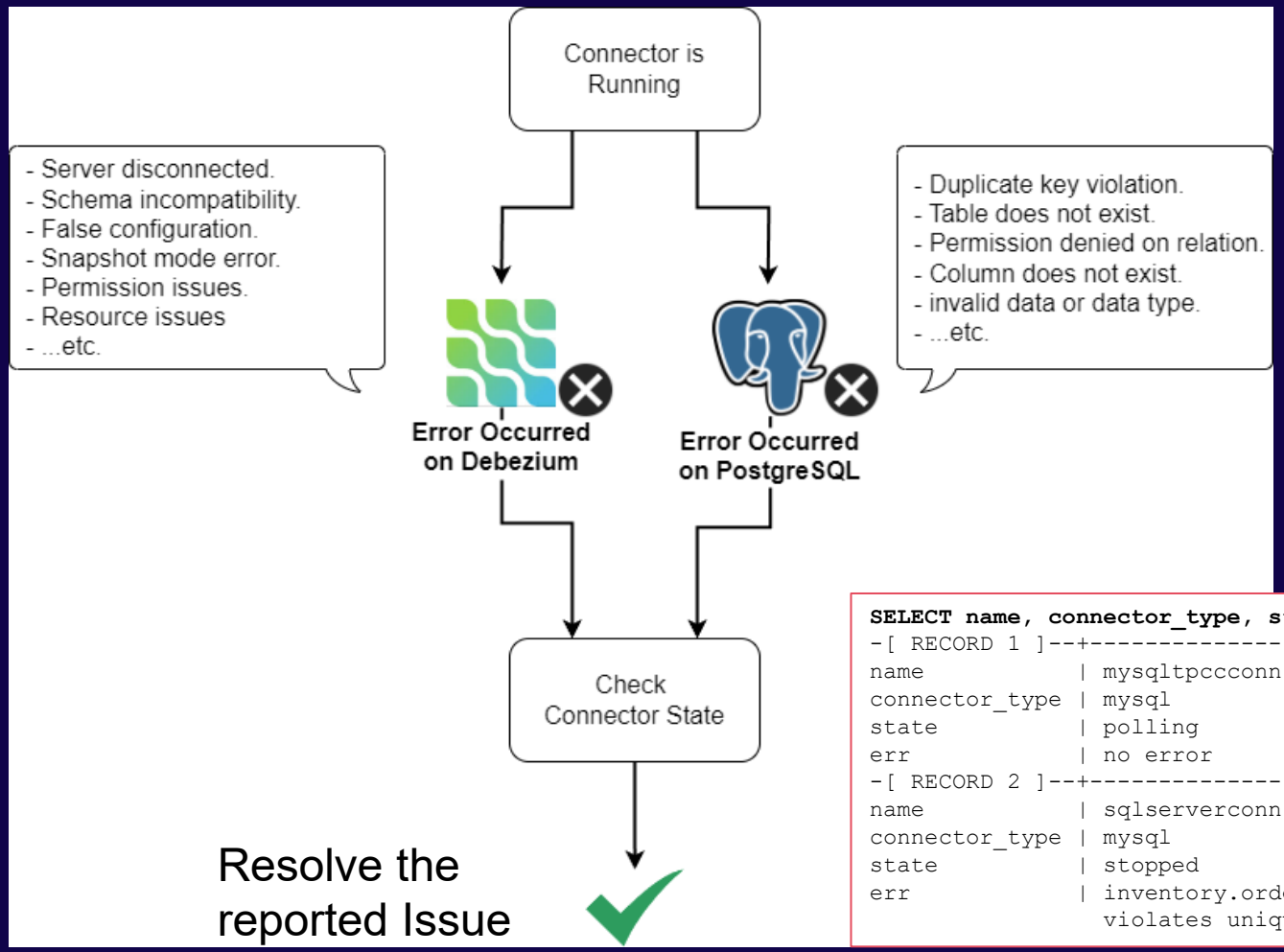
```
SELECT * from synchdb_state_view;
```

变更数据即
将到来……

配置连接器状态、统计信息和错误信息，我们希望它简单明了！

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"
```

C 和 Java 空间的错误传递和协调非常重要！

使用 Debezium 进行初始快照 (TableSync)

建立链接 → 确定要捕获的Table → 获取表上的全局读锁 → 开始事务 Repeatable Read

在连接器创建过程中选择

读取当前 binlog position → 捕获表结构并储存 schema history 文件 → 释放全局读锁 → 从 binlog 位置读取表数据并生成变更事件 → 提交事务 → 记录初始快照成功完成和 advance offset

Schema history persists to: $PGDATA/pg_synchdb

Resolve the

首次启动连接器时会执行初始快照。

CDC 紧随其后

Offset persists to: $PGDATA/pg_synchdb

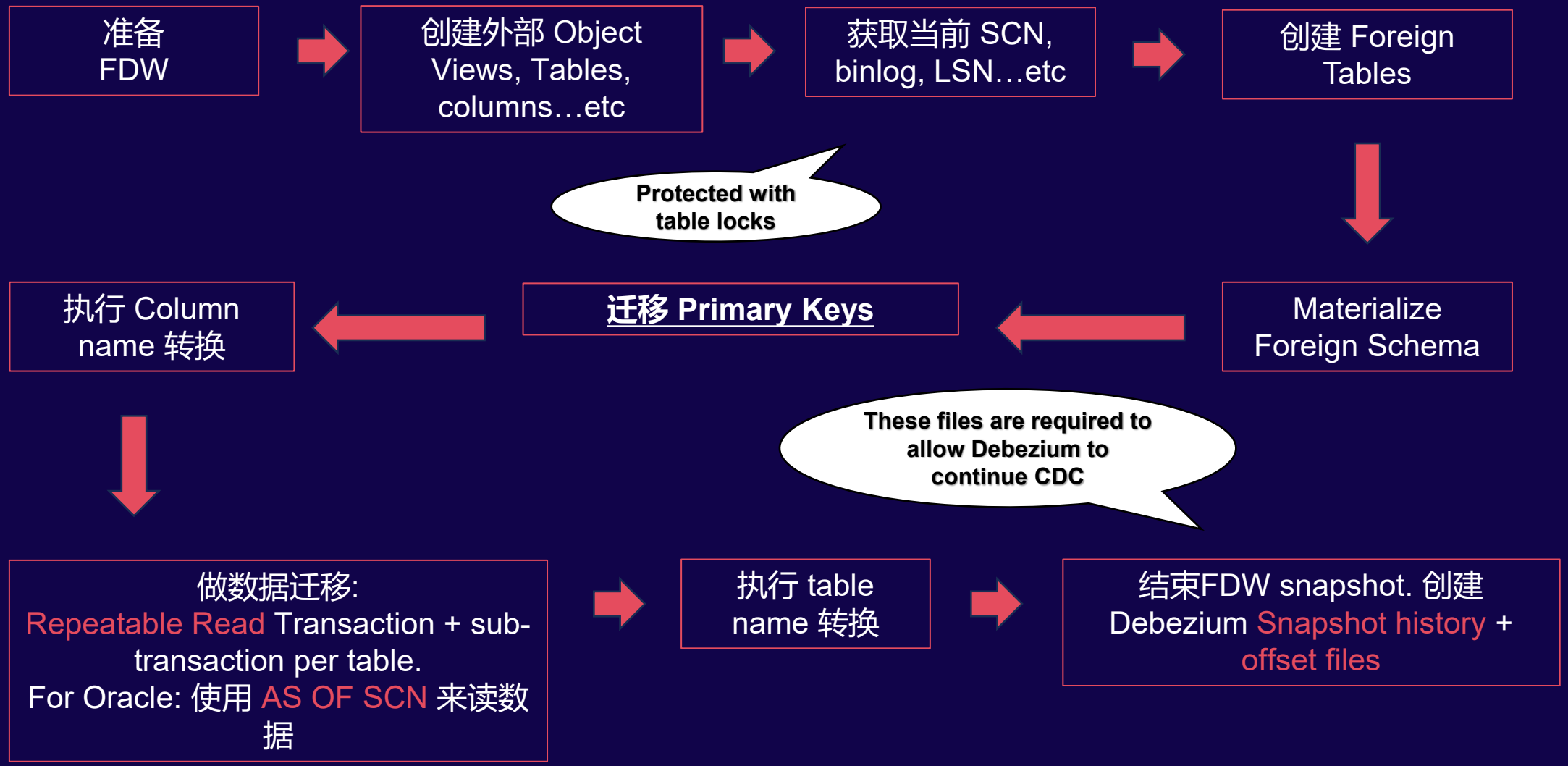
使用 FDW 进行初始快照 (TableSync)

HOW 2026
Hello Open-source World

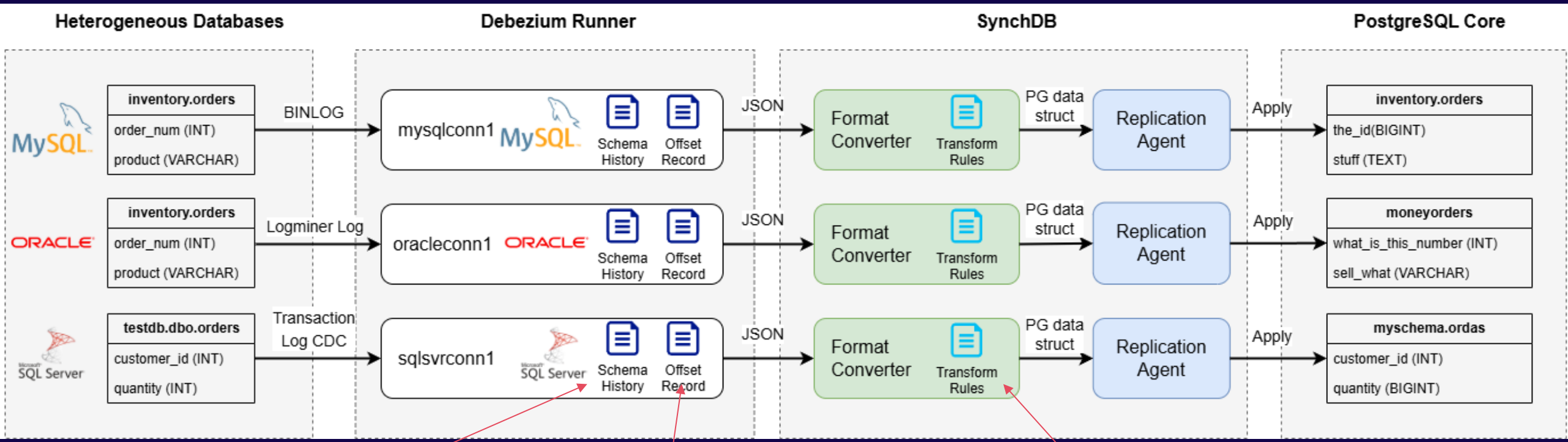
- Debezium 初始快照功能虽然可行，但当需要复制大量表（数千张）时速度会非常慢。
- 我们仅使用 FDW 即可实现类似的快照功能，并且性能显著提升
  - oracle_fdw
  - mysql_fdw
  - postgres_fdw

挑战：如何确保系统能够从外部表中 “SELECT” 特定 SCN、LSN、binlog pos 等范围内的数据?

使用 FDW 进行初始快照 (TableSync)



SynchDB 工作流 – 变更事件



Schema 历史记录

包含 Debezium 生成数据更改事件所需的表结构

Offset 偏移记录

包含 Debezium 开始读取变更记录的偏移量（类似于 PG 中的 LSN）。

转换规则

告诉 SynchDB 格式转换器如何在将数据应用到 PostgreSQL 之前转换传入的表名、列名、模式名甚至数据。

数据类型转换

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 会精确地显示它如何“转换”表、列或数据类型，并允许在 connector 启动前运行表达式。

SynchDB 工作流程 – 数据转换

- 转换规则不仅可以转换数据类型、表名和列名
- 还可以进行数据表达式 （expression）转换。
- 当复制的数据在应用到 PostgreSQL 之前需要进一步处理时，此功能非常有用。
- 如果我们在 PostgreSQL 之外构建中间件，则无法实现此功能，或者不够灵活。

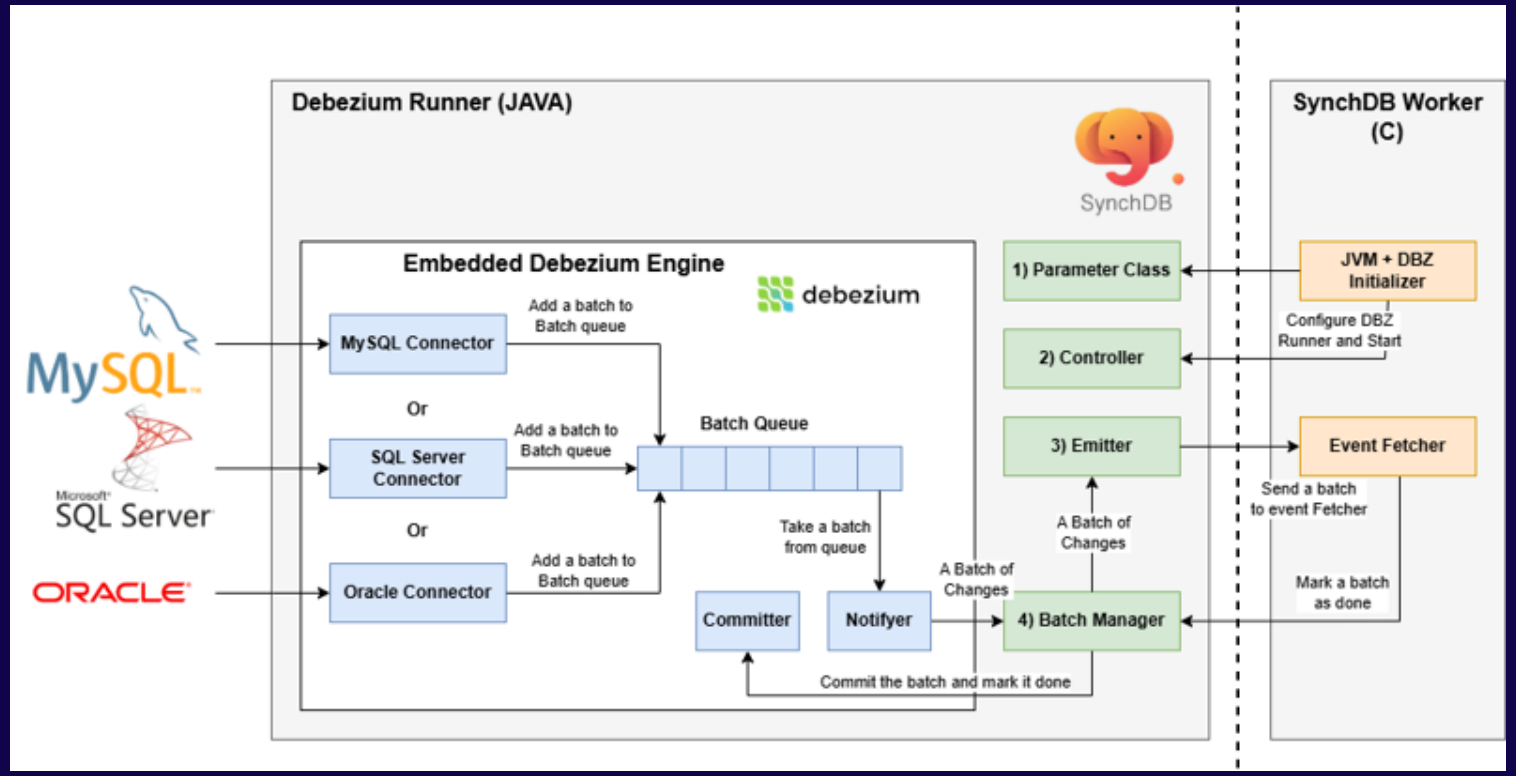
接收到的列数据在应用到 PostgreSQL 之前，会先经过用户定义的表达式进行处理：

- %d 替换为数据值
- %w 替换为 WKB 值（几何类型）
- %s 替换为 SRID 值（几何类型）

SynchDB 工作流 – 使用批量管理获取更改

HOW 2026
Hello Open-source World

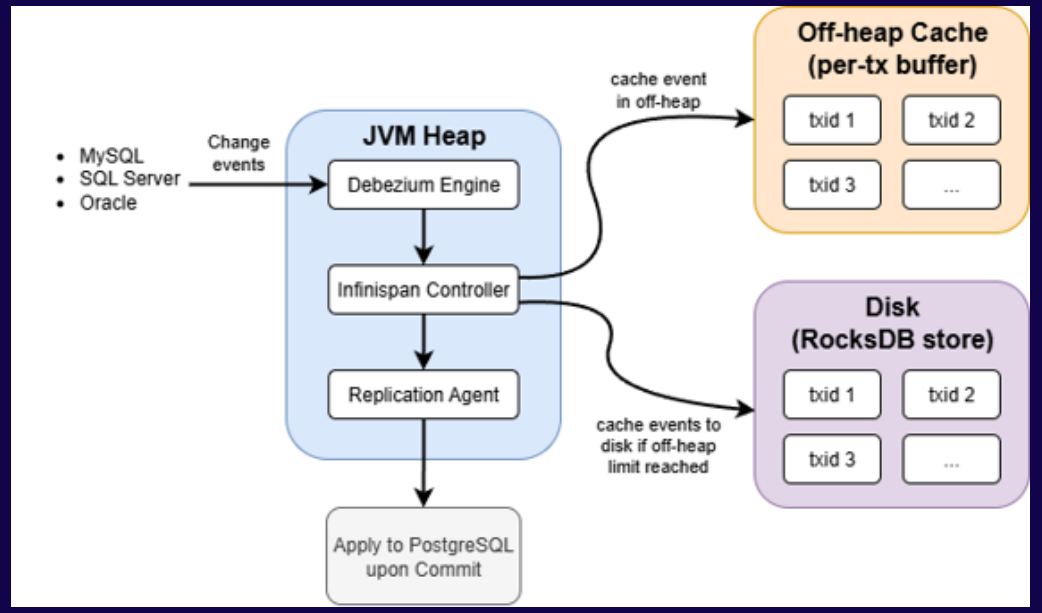
- 当批处理中的变更记录成功应用到 PostgreSQL 数据库后，该变更记录即被视为已完成。
- 批处理中已完成的变更记录允许 Debezium “推进偏移量” 。



更大的批处理大小 -> 更少的 JNI 调用 -> 更好的性能

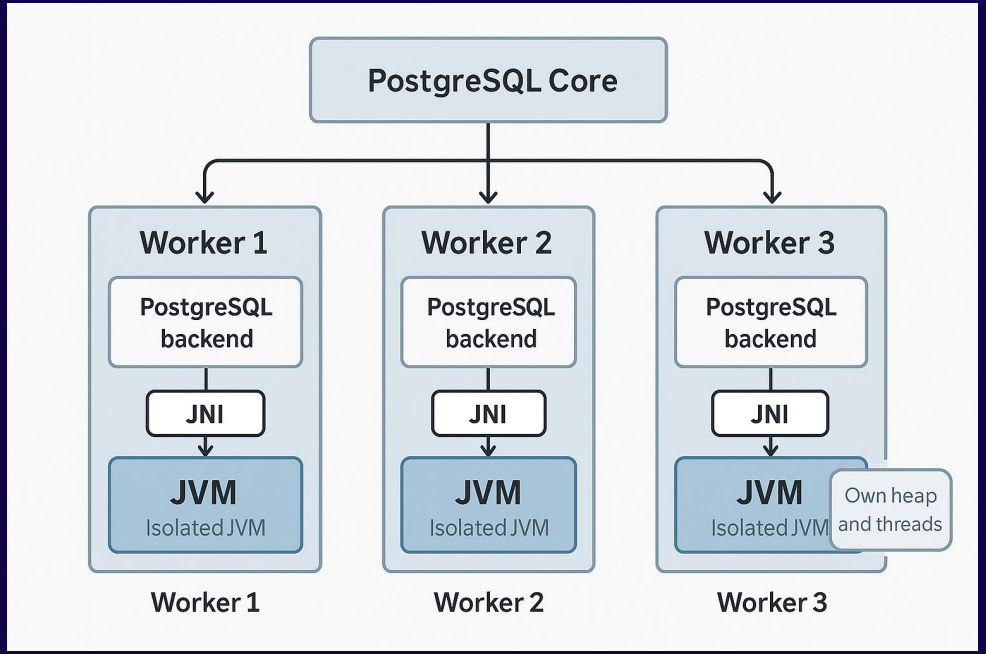
大型事务 - 使用 Infinispan 进行 off-heap 缓存

- 来自 MySQL、SQL Server 和 Oracle 的大型事务必须缓存到 COMMIT 阶段。
- 这可能会超出 JVM heap 内存限制并导致异常。
- SynchDB 可以配置 Infinispan，利用堆外内存（系统堆）或磁盘来缓存潜在的大型事务，然后再提交。



并发性问题

- JVM 是一个单进程环境，运行在单个操作系统进程（使用 `java -jar` 命令）或 PostgreSQL 后台工作进程（使用 JNI）中。
- 您不能在同一个进程内启动多个 JVM 。
- JVM 可以被同一个进程中的多个线程共享。
- PostgreSQL 是一个多进程数据库系统，每个进程都需要创建自己的 JVM 实例才能并行且独立地访问 Java。

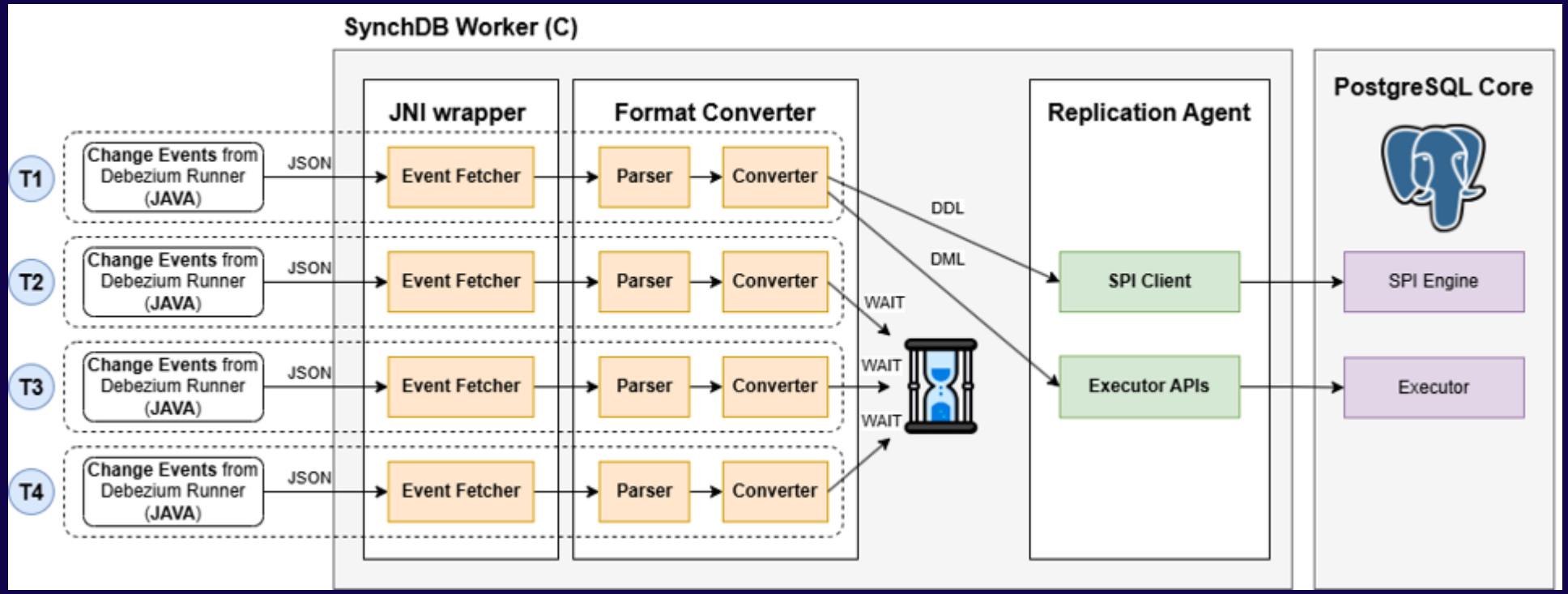


⚠ 多台 JVM 需要仔细规划内存，以防止资源耗尽。

并发性有帮助吗?

这取决于...。

- 这要看情况……线程应该只解析/处理 JVM 数据，而不能直接访问 PostgreSQL!
- 线程无法安全地运行事务、访问 PostgreSQL catalog 或对 PostgreSQL 做数据更改!

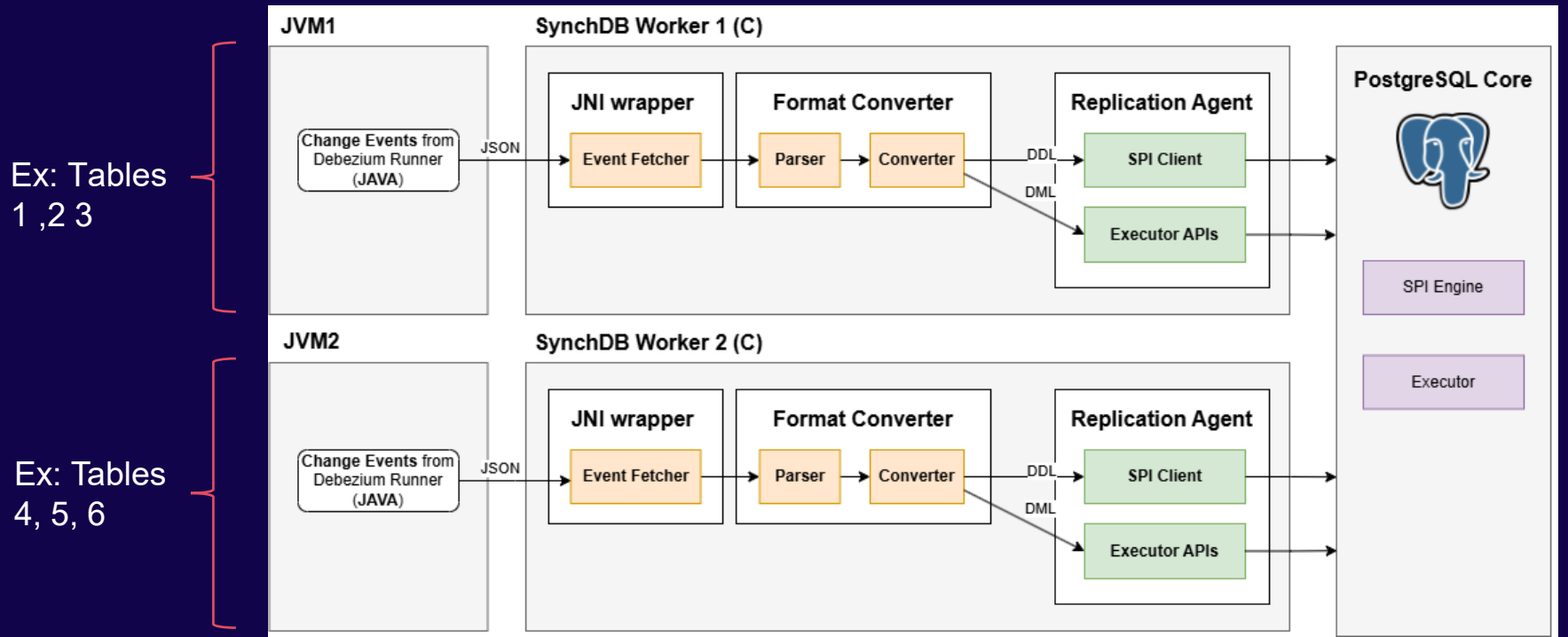


在这些限制条件下，使用线程能提升性能吗？不太可能。

并发问题 – 续

更安全的并发?

- 生成多个 SynchDB 工作进程，每个进程运行一个 JVM 来访问 Java 资源并并发应用更改。
- 每个工作进程负责不同的表列表，以防止冲突或死锁。
- 同一张表的并发操作？需要更严格的事务边界和顺序执行 = 工作量更大。



性能提升的代价是需要运行更多的JVM。

SynchDB v1.3

- SynchDB v1.3 将于 2025 年 11 月 25 日发布。
- GitHub 代码库地址：https://github.com/Hornetlabs/synchdb
- 文档网站地址：https://docs.synchdb.com/
- 欢迎新的贡献者！

异构数据库支持:

- MySQL
- SQLServer
- Oracle
- Openlog Replicator

转换功能:

- Data types
- Table names
- Column names
- Data values

DMLs:

- INSERT
- UPDATE
- DELETE
- TRUNCATE

PG 数据处理模式:

- SPI
- Executor APIs
- FDW initial snapshots

复制模式

- Initial Snapshot
- Change Data Capture (CDC)

数据快照模式:

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

监控:

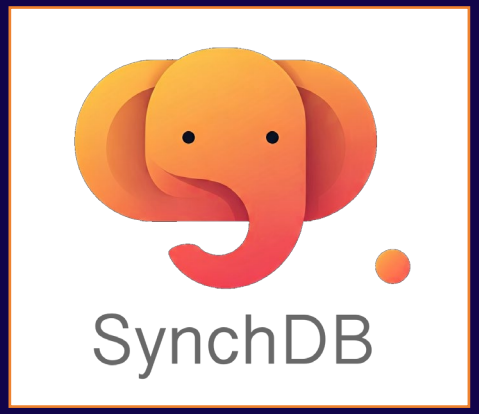
- JMX Mbeans
- Prometheus and Grafana

快速测试:

- Ezdeploy utility

DDLs:

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




HOW 2026
Hello Open-source World
Thanks