Go Concurrency powering a Gigabyte scale real-world data pipeline

Chinmay Naik

Chinmay Naik 

@chinmay185

🏗️ Founder One2N (Backend and Reliability engineering)

📖 Writes stories on Pragmatic Software Engineering

❤️ Engineering 👩‍💻, Psychology 🧐, Percussion 🥁 and

Age of Empires 2

MongoDB to RDBMS Data migration

MongoDB

ETL

One-time bulk transfer (snapshot)
+
Streaming data transfer

PostgreSQL

How do we map MongoDB documents to tables and rows in PostgreSQL?

student collection (MongoDB)

```
{
  "_id": "635b79e231d82a8ab1de863b",
  "name": "Selena Miller",
  "roll_no": 51,
  "is_graduated": false,
  "date_of_birth": "2000-01-30",
  "address": [
    {"line1": "481 Harborsburgh", "zip": "89799"},
    {"line1": "329 Flatside", "zip": "80872"}
  ],
  "phone": {"personal": "7678456640", "work": "8130097989"}
}
```

primary key

string field

numeric field

boolean field

nested array of objects

nested object

student table (PostgreSQL)

```sql
CREATE TABLE student
(
    _id             VARCHAR(255) PRIMARY KEY,
    name            VARCHAR(255),
    roll_no         FLOAT,
    is_graduated    BOOLEAN,
    date_of_birth   VARCHAR(255)
);
```

primary key

string field

numeric field

boolean field

Wait, what about nested fields?
(address and phone)

student - address and phone relationships

```sql
CREATE TABLE student_address
(
    _id          VARCHAR(255) PRIMARY KEY
    student__id  VARCHAR(255),
    line1        VARCHAR(255),
    zip          VARCHAR(255)
);

CREATE TABLE student_phone
(
    _id          VARCHAR(255) PRIMARY KEY
    student__id  VARCHAR(255),
    personal     VARCHAR(255),
    work         VARCHAR(255)
);
```

primary key

foreign key (logical)

other fields

primary key

foreign key (logical)

other fields

Data migration - MongoDB to PostgreSQL

```
{
  "_id": "635b79e231d82a8ab1de863b",
  "name": "Selena Miller",
  "roll_no": 51,
  "is_graduated": false,
  "date_of_birth": "2000-01-30",
  "address": [
    {"line1": "481 Harborsburgh", "zip": "89799"},
    {"line1": "329 Flatside", "zip": "80872"}
  ],
  "phone": {"personal": "7678456640", "work": "8130097989"}
}
```

student table: 1 record

student_address table: 2 records

student_phone table: 1 record

Data migration - MongoDB to PostgreSQL

JSON

```json
{
  "_id": "635b79e231d82a8ab1de863b",
  "name": "Selena Miller",
  "roll_no": 51,
  "is_graduated": false,
  "date_of_birth": "2000-01-30",
  "address": [
    {
      "line1": "481 Harborsburgh",
      "zip": "89799"
    },
    {
      "line1": "329 Flatside",
      "zip": "80872"
    }
  ],
  "phone": {
    "personal": "7678456640",
    "work": "8130097989"
  }
}
```

SQL

```sql
-- student table (1 row)
INSERT INTO student (_id, date_of_birth, is_graduated, name, roll_no)
VALUES ('635b79e231d82a8ab1de863b', '2000-01-30', false, 'Selena Miller', 51);

-- student_address table (2 rows)
INSERT INTO student_address (_id, line1, student__id, zip)
VALUES ('1', '481 Harborsburgh', '635b79e231d82a8ab1de863b', '89799');
INSERT INTO student_address (_id, line1, student__id, zip)
VALUES ('2', '329 Flatside', '635b79e231d82a8ab1de863b', '80872');

-- student_phone table (1 row)
INSERT INTO student_phone (_id, personal, student__id, work)
VALUES ('1', '7678456640', '635b79e231d82a8ab1de863b', '8130097989');
```

How MongoDB JSON data maps to SQL

One MongoDB Document -> N SQL statements

Nested JSON array and object -> related tables with foreign key logical constraint

1. Update SQL statement and
2. Alter table statement (for schema change)

Inserts are cool, what about updates and deletes in MongoDB?

Delete SQL statement

How do we migrate data?

One time bulk migration? 🤔

Streaming fashion? 🧐

How about **both** options? 🤨

Any thoughts?

We need a reliable way to track all updates (inserts/updates/deletes) to any of the MongoDB documents.

Mongo Oplog (operational log)

- Write ahead log for MongoDB
- `oplog.rs` capped collection in `local` database
- It tracks all edits to all databases (inserts, updates, deletes)

What does oplog record look like?

Insert

Similarly, delete

Update

```
{
  "op": "i",
  "ns": "test.student",
  "o": {
    "_id": "635b79e231d82a8ab1de863b",
    "name": "Selena Miller",
    "roll_no": 51,
    "is_graduated": false,
    "date_of_birth": "2000-01-30"
  }
}
```

db.collection

inserted object

```
{
  "op": "u",
  "ns": "test.student",
  "o": {
    "$v": 2,
    "diff": {
      "u": {
        "is_graduated": true
      }
    }
  },
  "o2": {
    "_id": "635b79e231d82a8ab1de863b"
  }
}
```

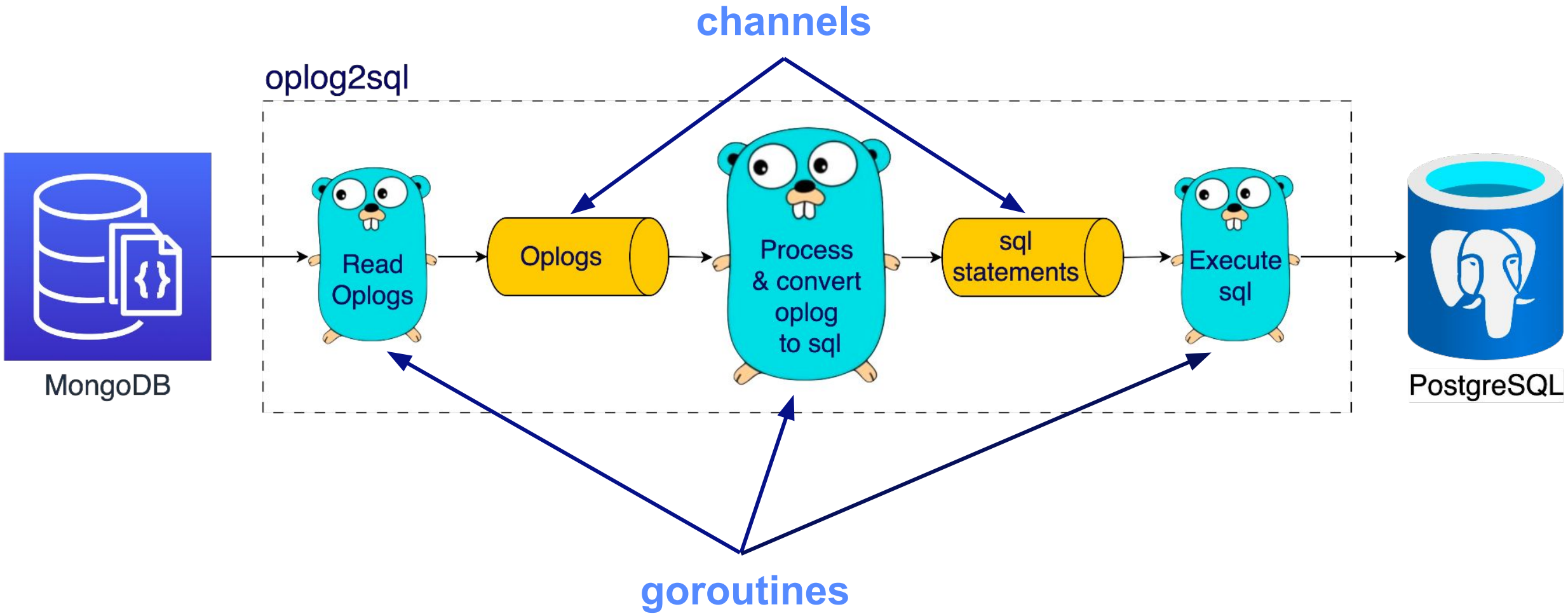
set field value

updated object

When are we getting to the Golang Concurrency part of the talk?

One2N

Sequential Data Pipeline



Mongo Oplog

```
{
  "op" : "i",
  "ns" : "test.student",
  "o" : {
    "_id" : "635b79e231d82a8ab1de863b",
    "name" : "Selena Miller",
    "roll_no" : 51,
    "is_graduated" : false,
    "date_of_birth" : "2000-01-30"
  }
}
```

PostgreSQL

Schema: **test**

Table: **student**

_id	name	roll_no	is_graduated	date_of_birth
635b79e231d82a8ab1de863b	Selena Miller	51	false	2000-01-30

Create Schema
Create Table
Insert data into table

Mongo Oplog

```
{
  "op" : "i",
  "ns" : "test.student",
  "o" : {
    "_id" : "635b79e231d82a8ab1de863b",
    "name" : "Selena Miller",
    "roll_no" : 51,
    "is_graduated" : false,
    "date_of_birth" : "2000-01-30"
  }
}
```

PostgreSQL

```
CREATE SCHEMA test;

CREATE TABLE test.student (_id VARCHAR(255) PRIMARY KEY, name VARCHAR(255),
 roll_no INTEGER, is_graduated BOOLEAN, date_of_birth VARCHAR(255));

INSERT INTO test.student (_id, name, roll_no, is_graduated, date_of_birth)
VALUES ('635b79e231d82a8ab1de863b', 'Selena Miller', 51, false, '2000-01-30');
```

One2N

Two Oplogs

```json
[
  {
    "op": "i",
    "ns": "test.student",
    "o": {
      "_id": "635b79e231d82a8ab1de863b",
      "name": "Selena Miller",
      "roll_no": 51,
      "is_graduated": false,
      "date_of_birth": "2000-01-30"
    }
  },
  {
    "op": "i",
    "ns": "test.student",
    "o": {
      "_id": "14798c213f273a7ca2cf5174",
      "name": "George Smith",
      "roll_no": 21,
      "is_graduated": true,
      "date_of_birth": "2001-03-23"
    }
  }
]
```

PostgreSQL

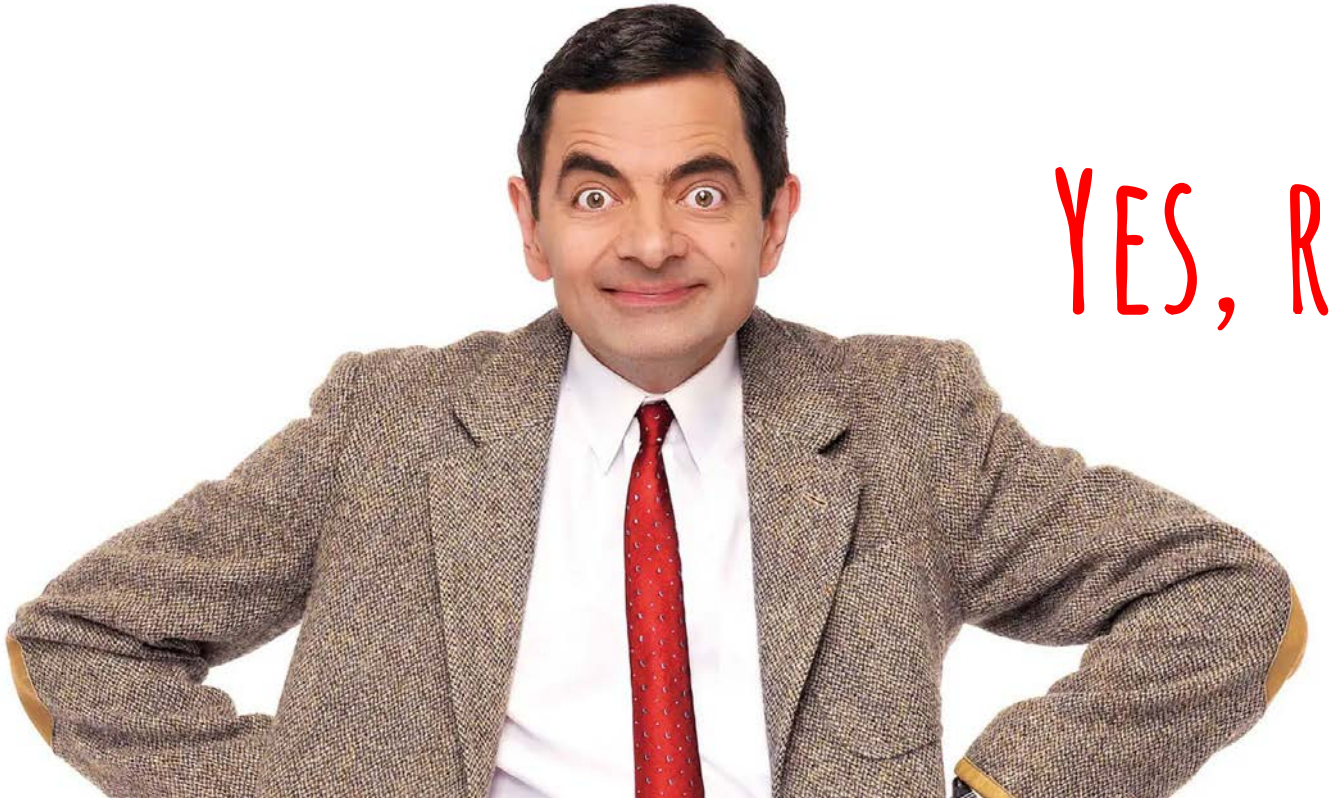
```sql
-- create schema and table (only once)
CREATE SCHEMA test;
CREATE TABLE test.student (_id VARCHAR(255) PRIMARY KEY, date_of_birth VARCHAR(255),
                           is_graduated BOOLEAN, name VARCHAR(255), roll_no FLOAT);

-- insert first record
INSERT INTO test.student (_id, date_of_birth, is_graduated, name, roll_no)
    VALUES ('635b79e231d82a8ab1de863b', '2000-01-30', false, 'Selena Miller', 51);

-- insert second record
INSERT INTO test.student (_id, date_of_birth, is_graduated, name, roll_no)
    VALUES ('14798c213f273a7ca2cf5174', '2001-03-23', true, 'George Smith', 21);
```

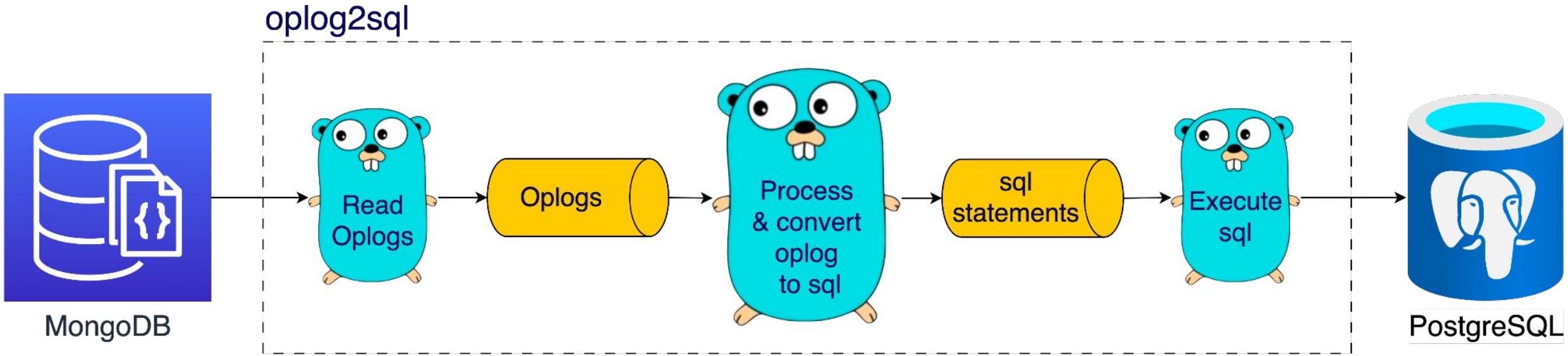
One2N

Trust me to handle all other edge cases related to updates and deletes

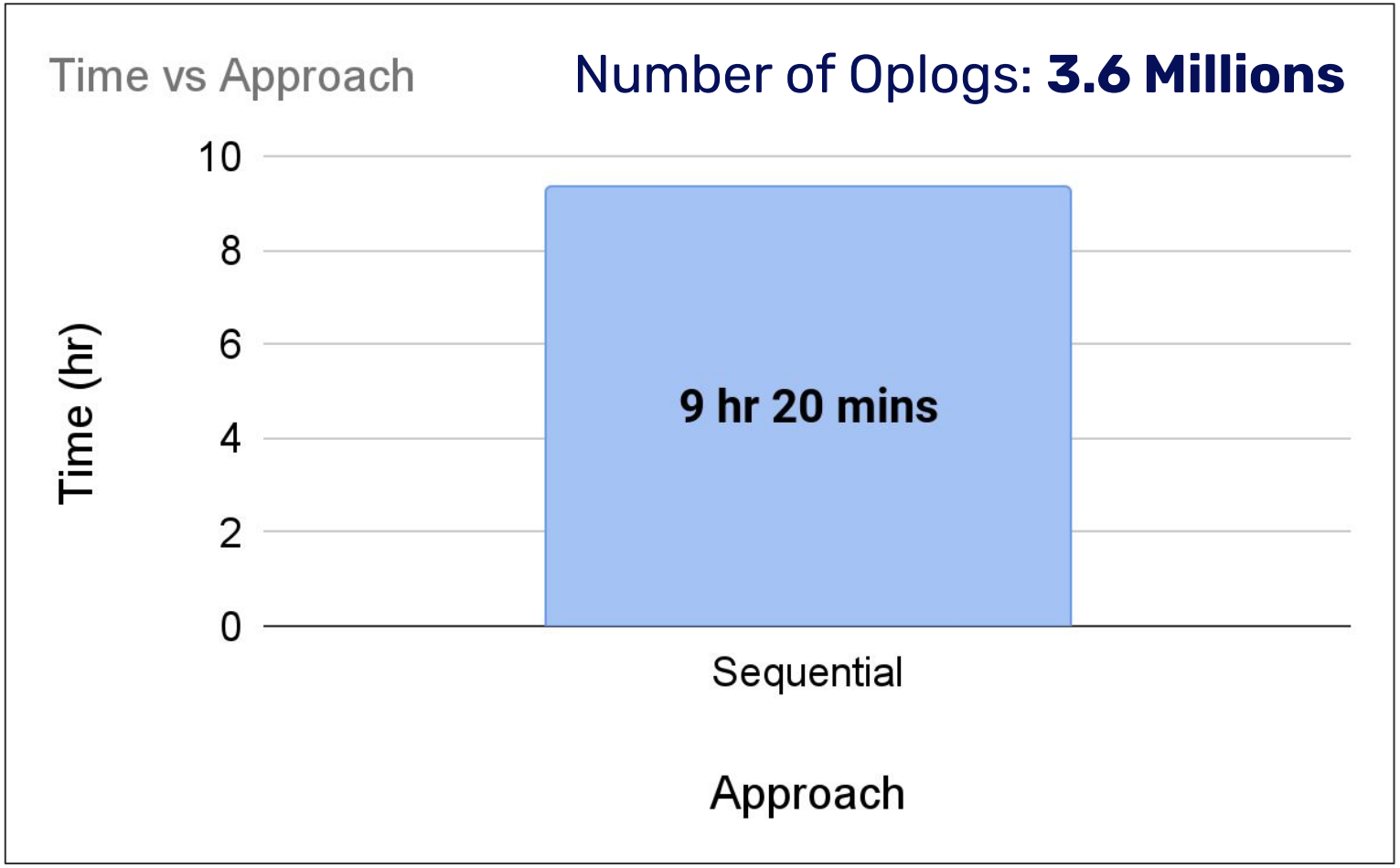


Yes, really!

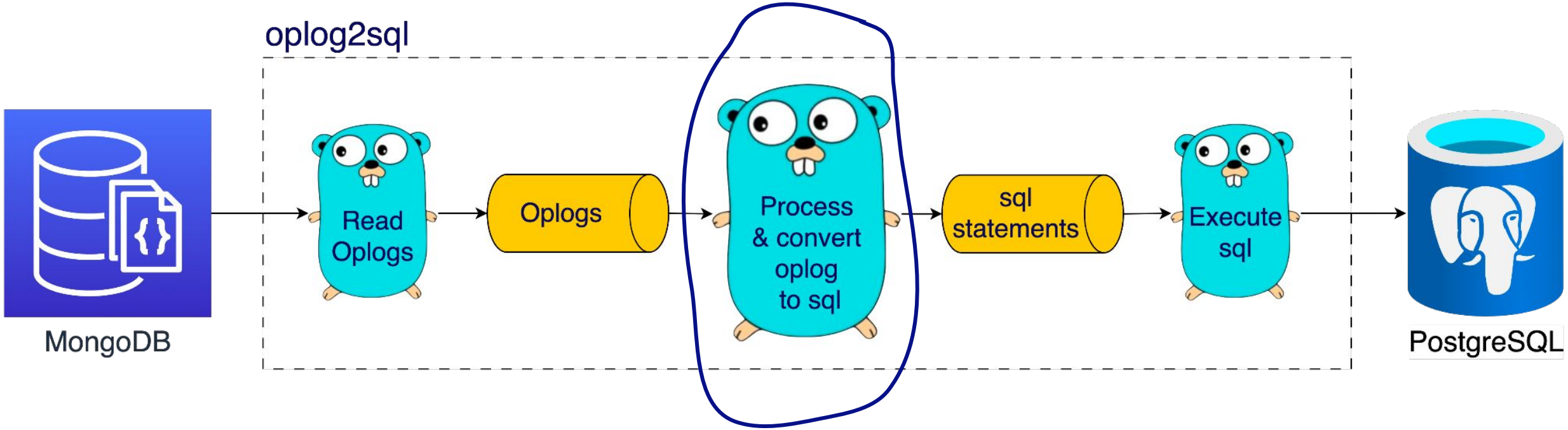
Sequential Data Pipeline (recap)



Sequential Pipeline Performance



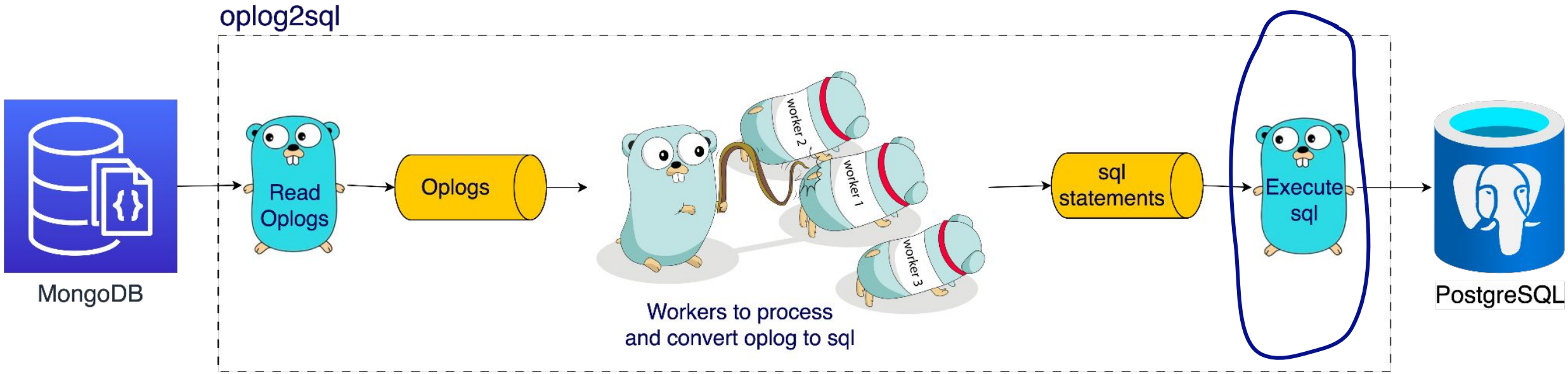
Perf improvement - Let's add Worker Pool



But where?

I am doing too much work
I need help

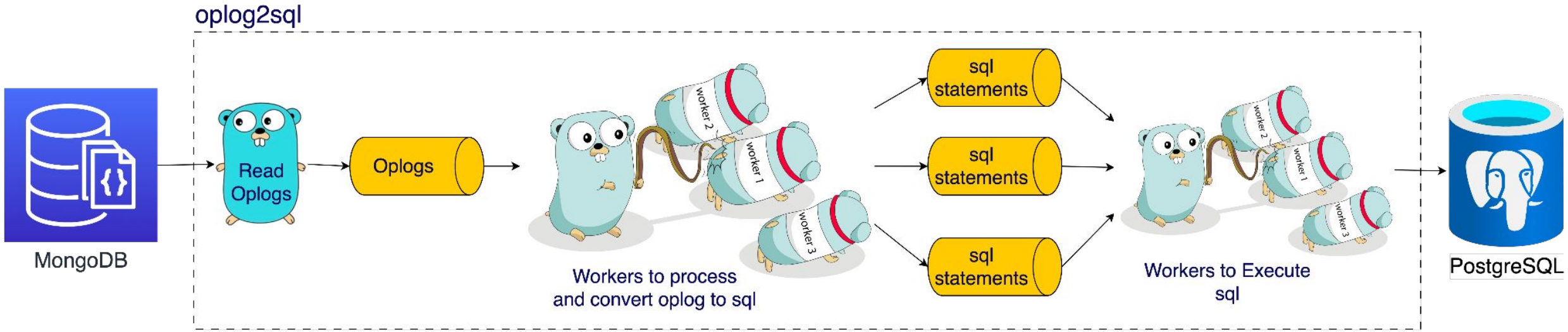
Worker Pool



I could use some help

Can we add more workers?

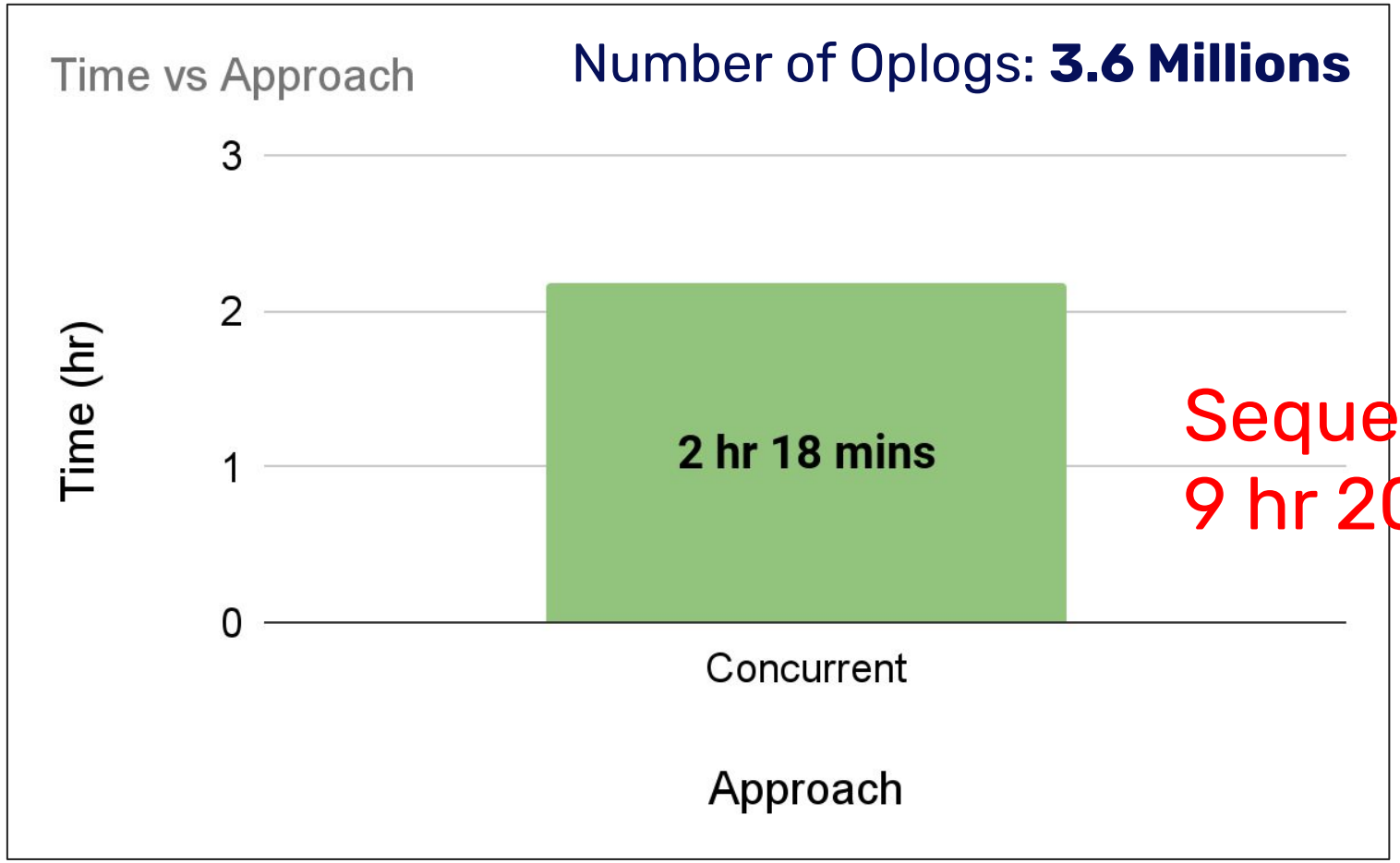
Worker Pools v2.0



No Gophers were harmed in this exercise 😝

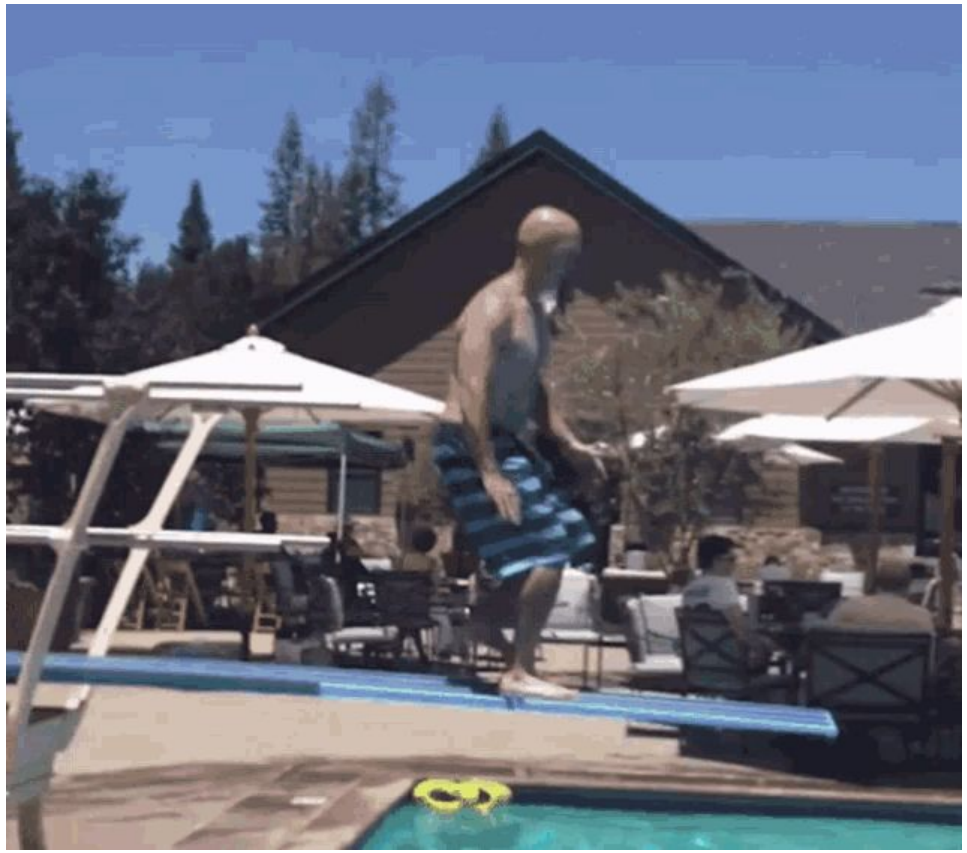
Worker Pool v2.0 Performance

4x improvement!

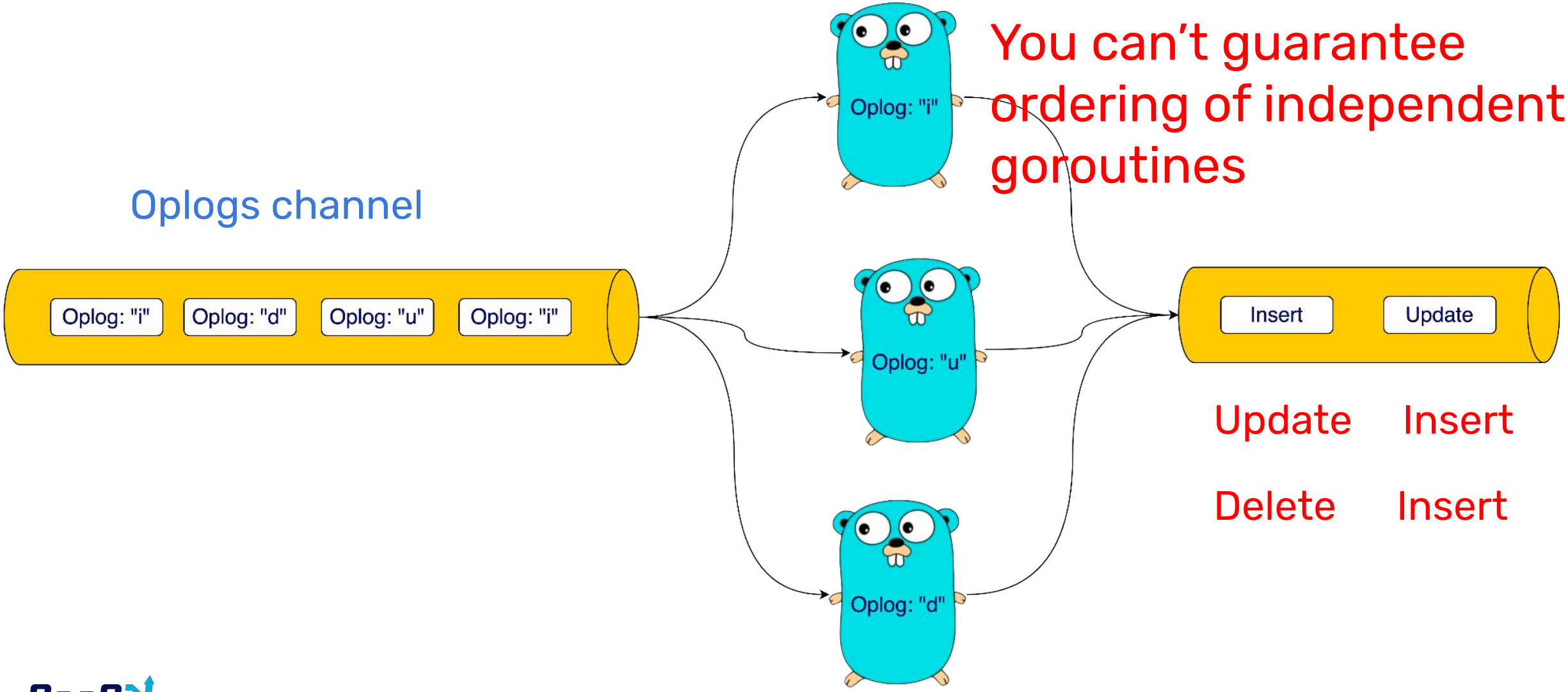


Sequential was
9 hr 20 mins

But wait, something’s wrong!



Can you guess the problem?

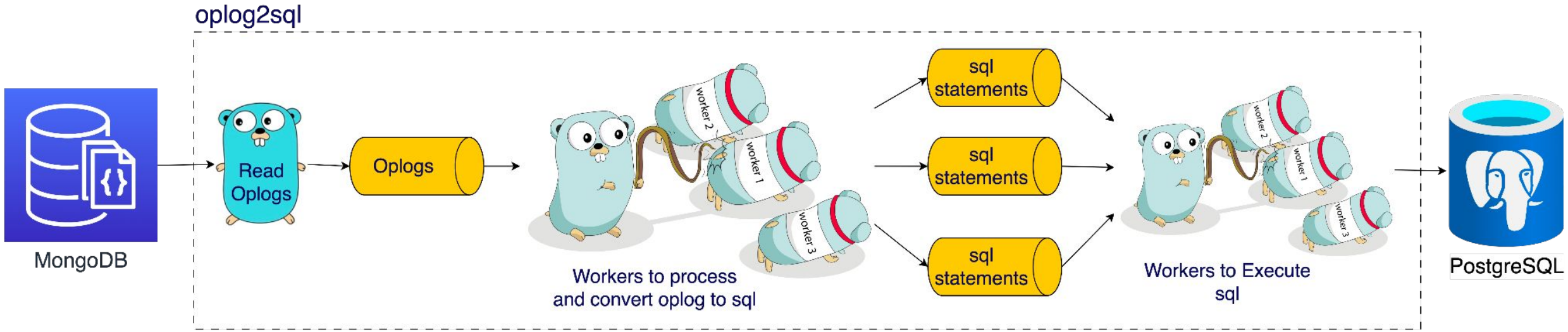


You can't guarantee ordering of independent goroutines

Update Insert

Delete Insert

Worker Pools v2.0 - The problem

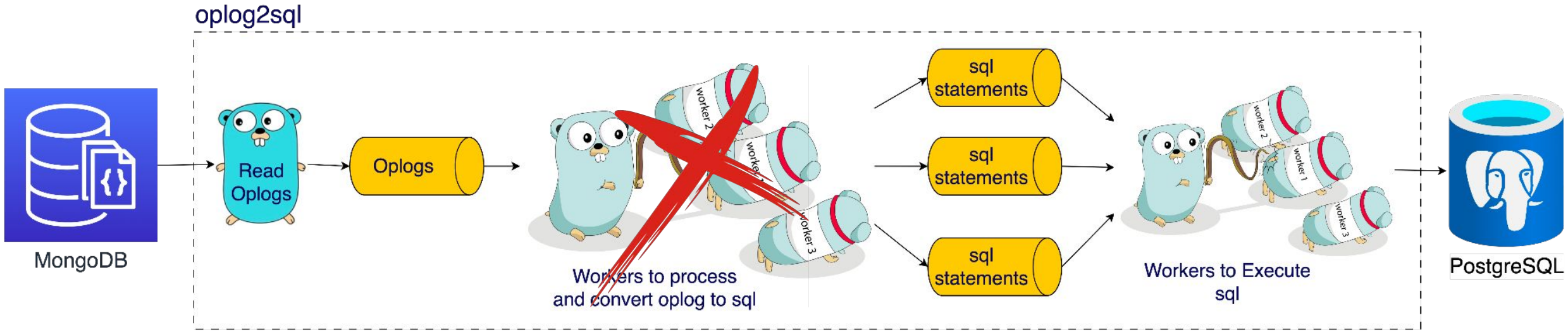


Same document
1. Insert
2. Update

SQL result
1. Update SQL
2. Insert SQL

SQL error!

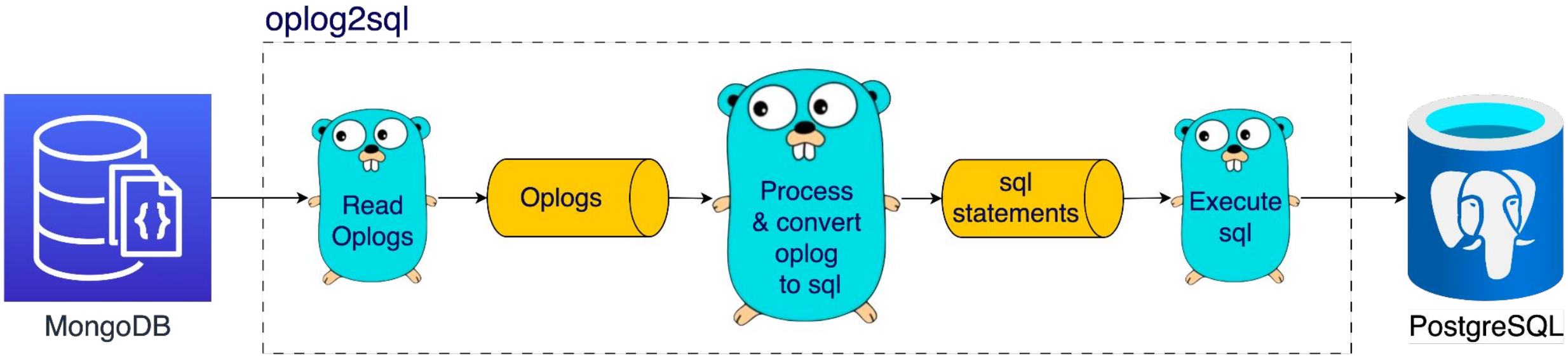
Why Worker Pools v2.0 won't work



Data integrity compromised

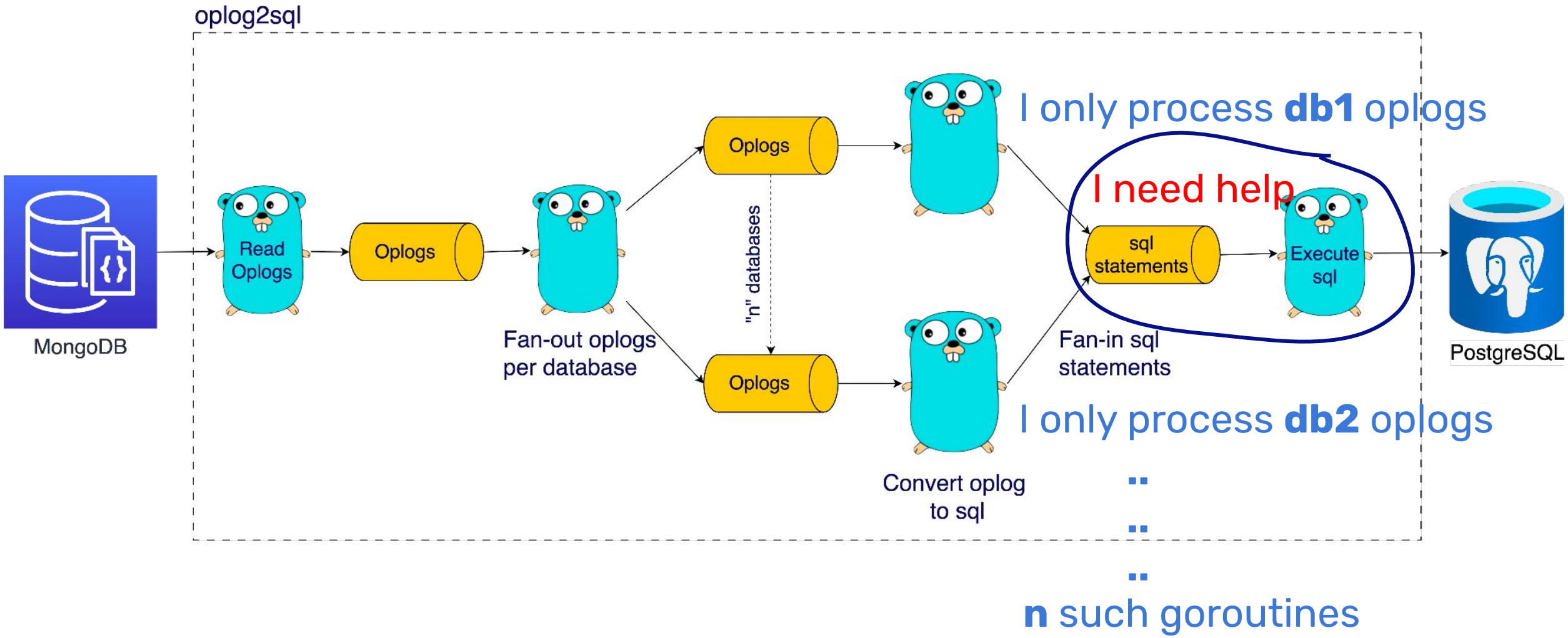
We can't throw worker pools at the problem without domain understanding

Back to drawing board?

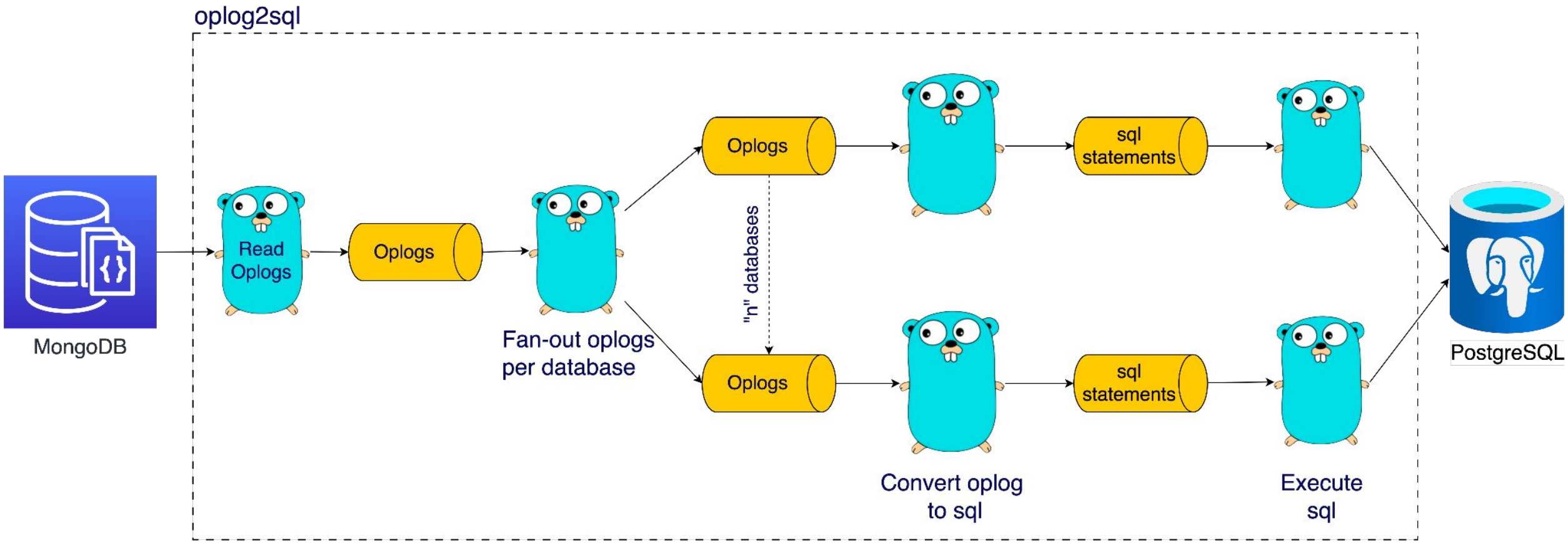


n databases
each database with
m collections

Fan-out for each database, fan-in for SQL



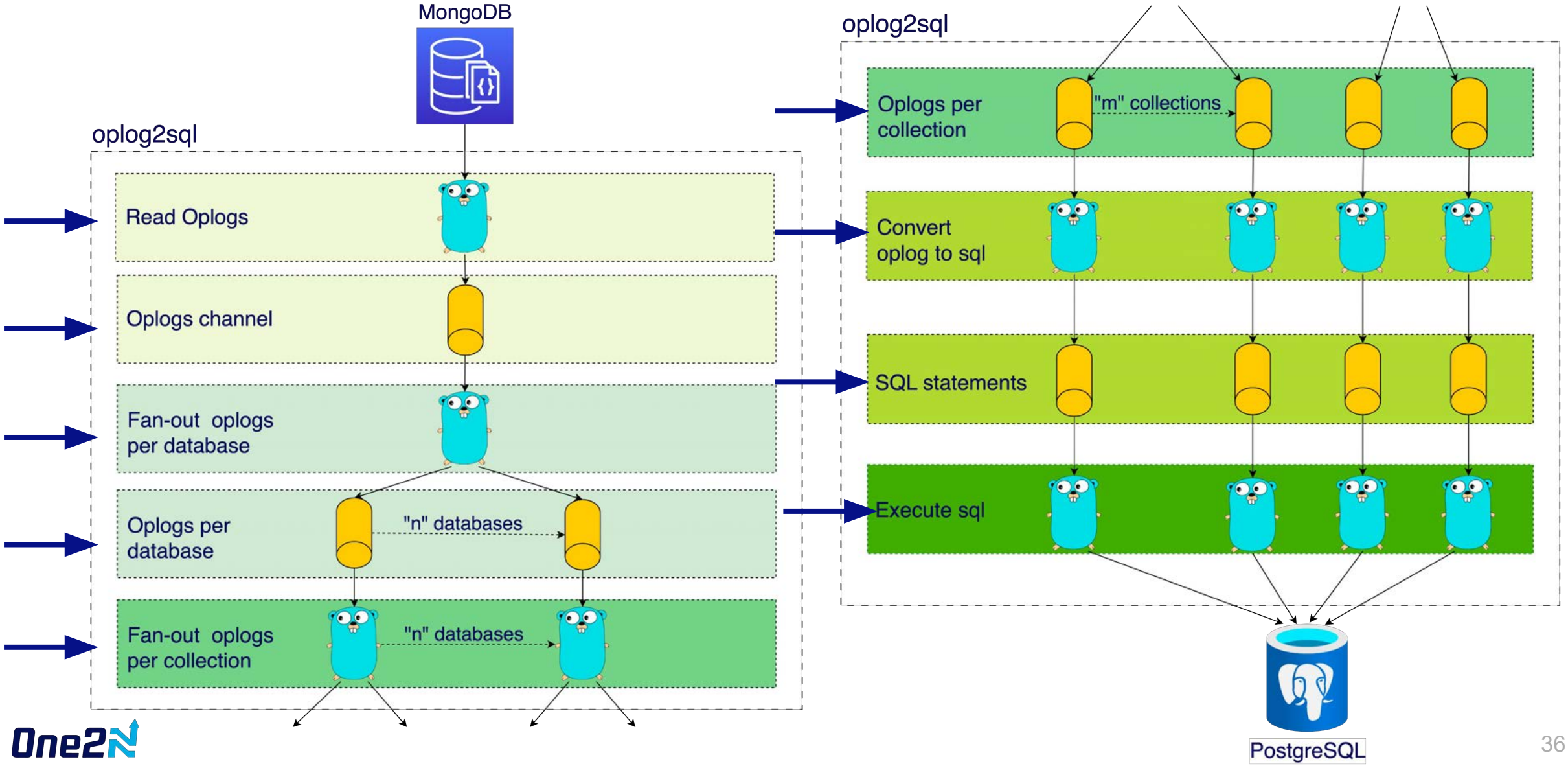
Fan-out for each database, without fan-in



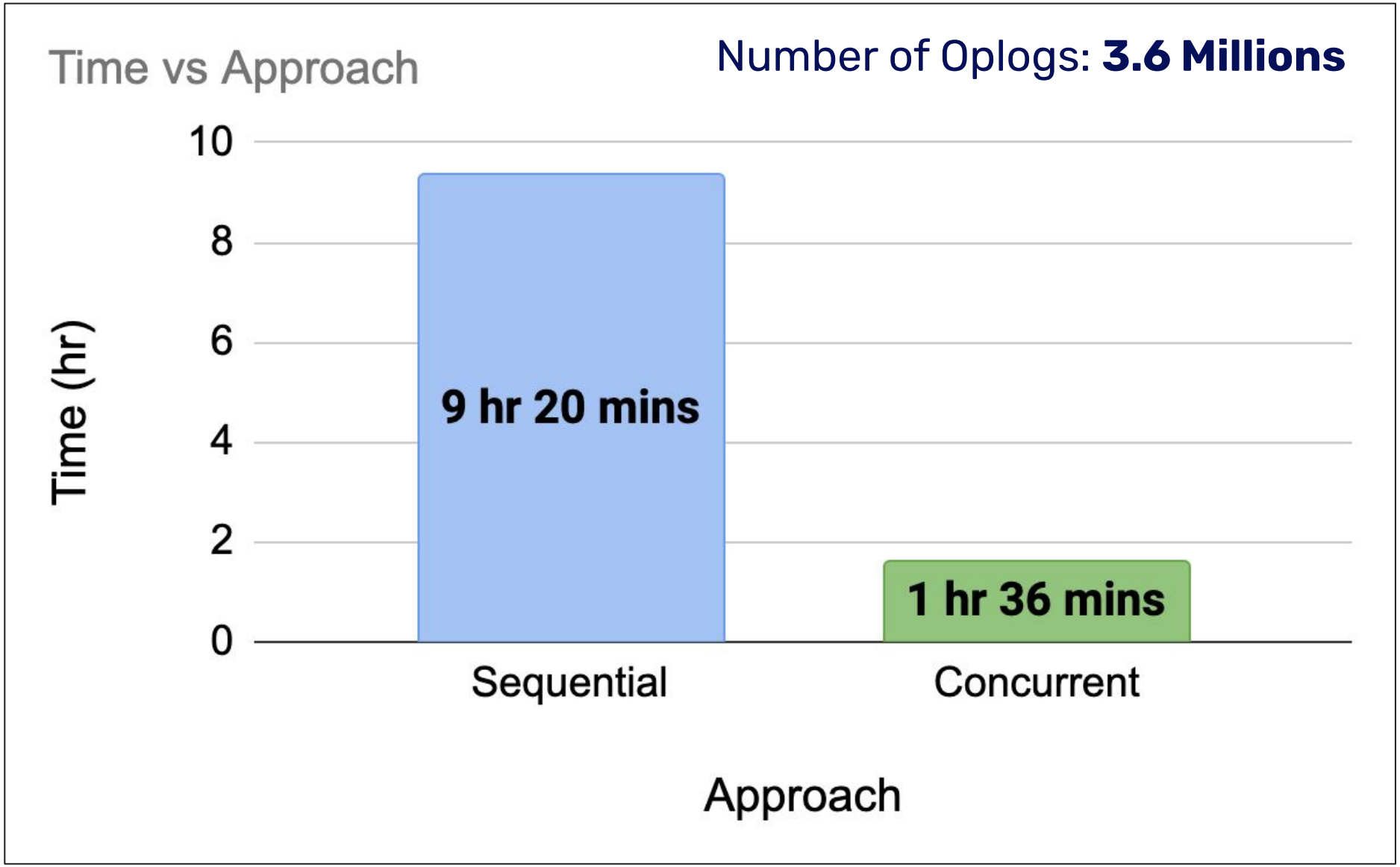
1 goroutine per database and 1 goroutine for SQL insert
Total number of goroutines => 2 * n (databases)

What if we spin up a goroutine for every database and collection combination?

Concurrent Data Pipeline



Performance Comparison



Resource utilization

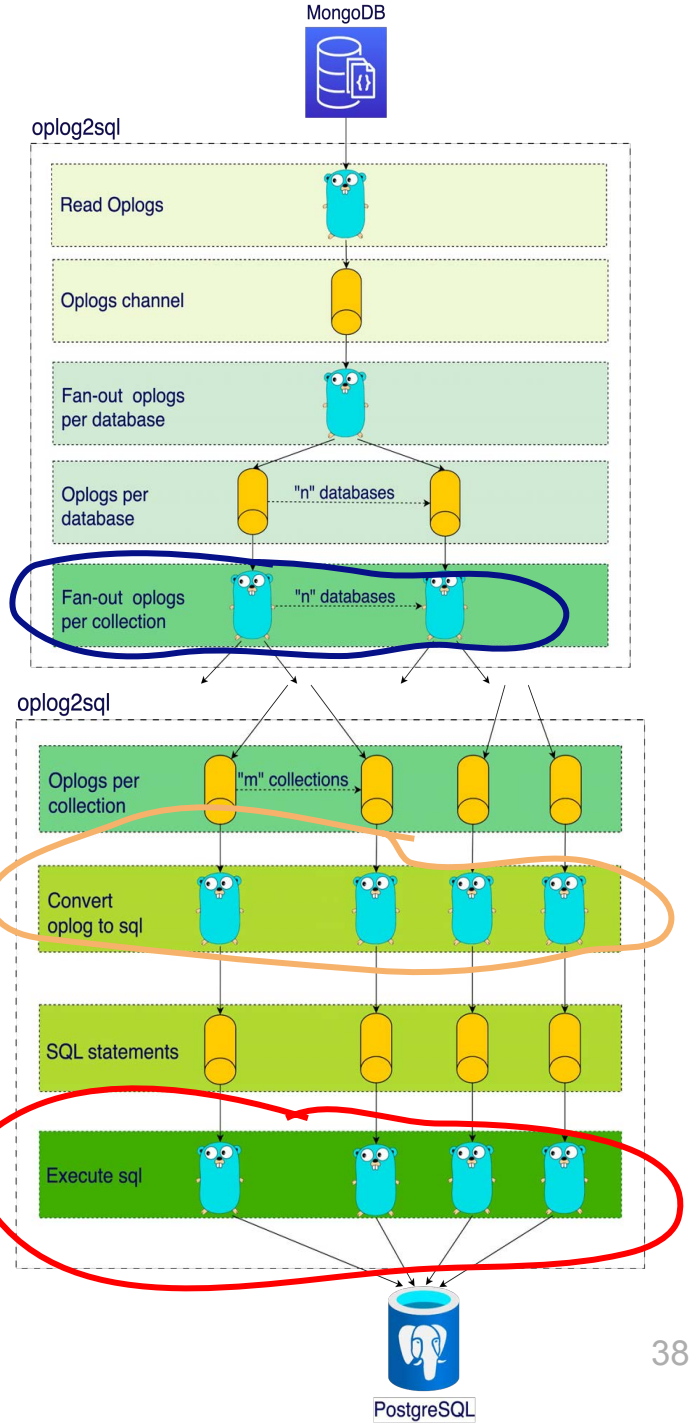
Number of databases: n (16)

Number of collections per db: m (128)

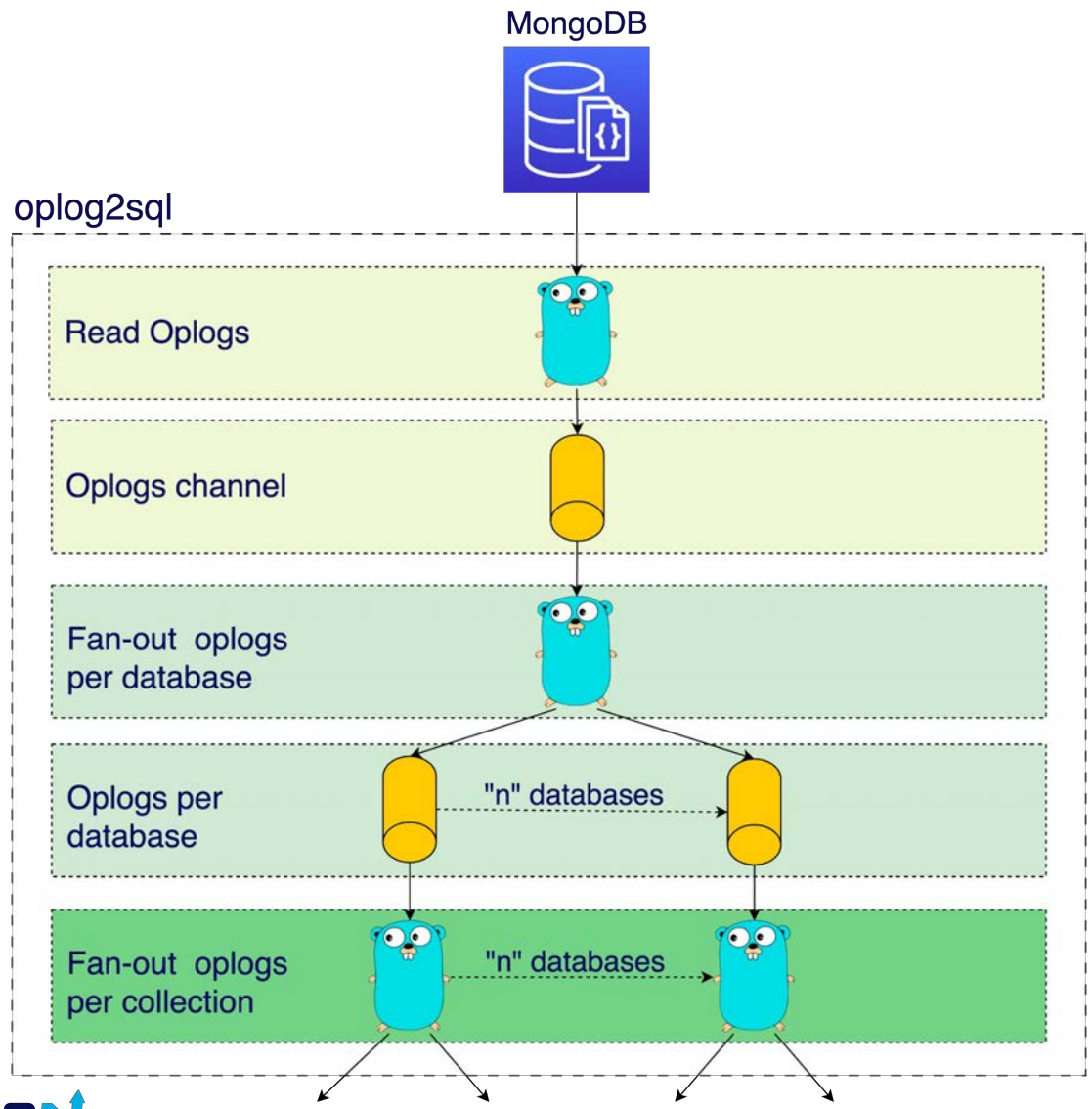
Number of Goroutines: n + n*m + n*m

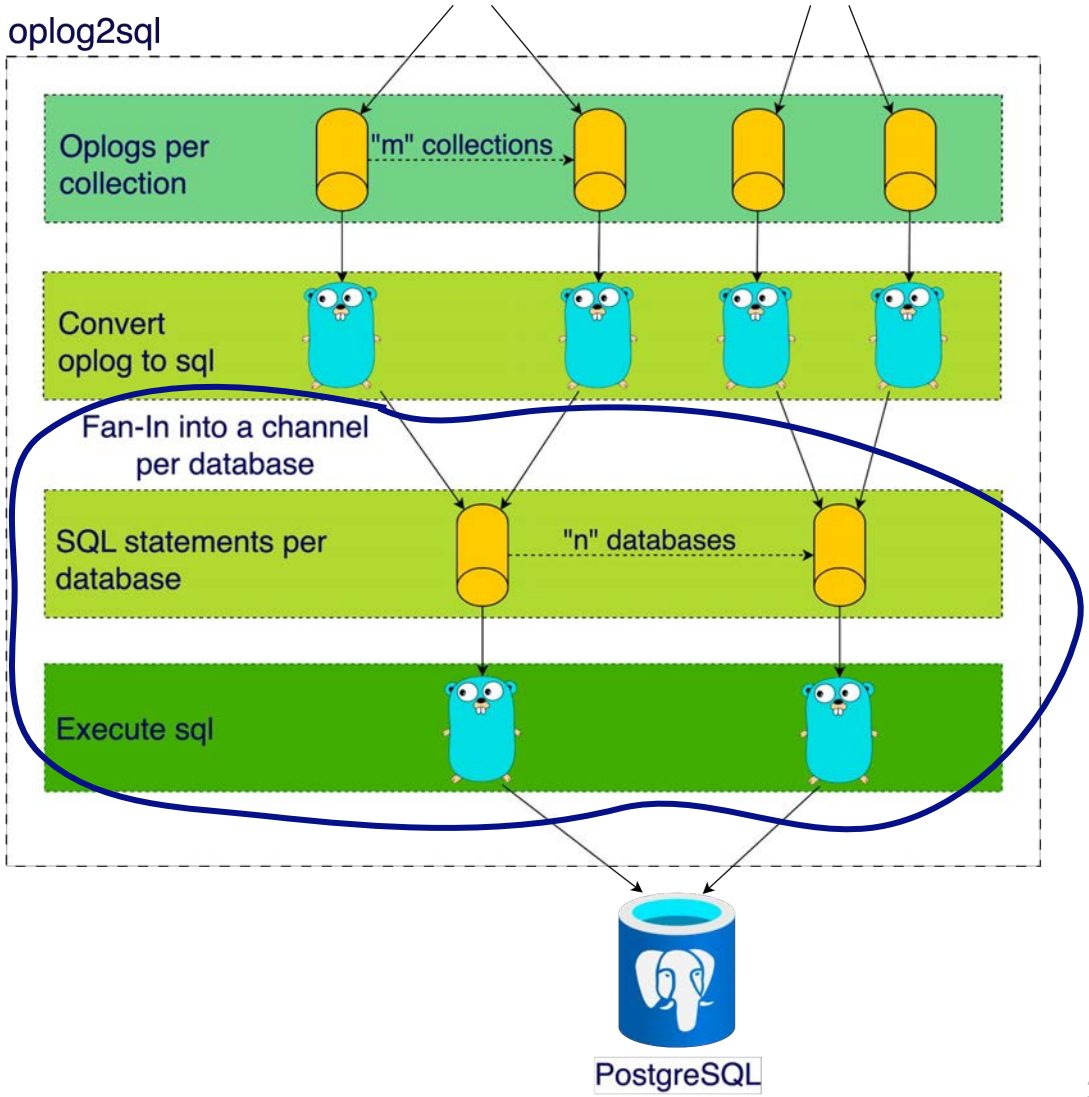
Number of Channels: n + n*m + n*m

Number of DB Connections: n*m = 2048 **Ouch!**



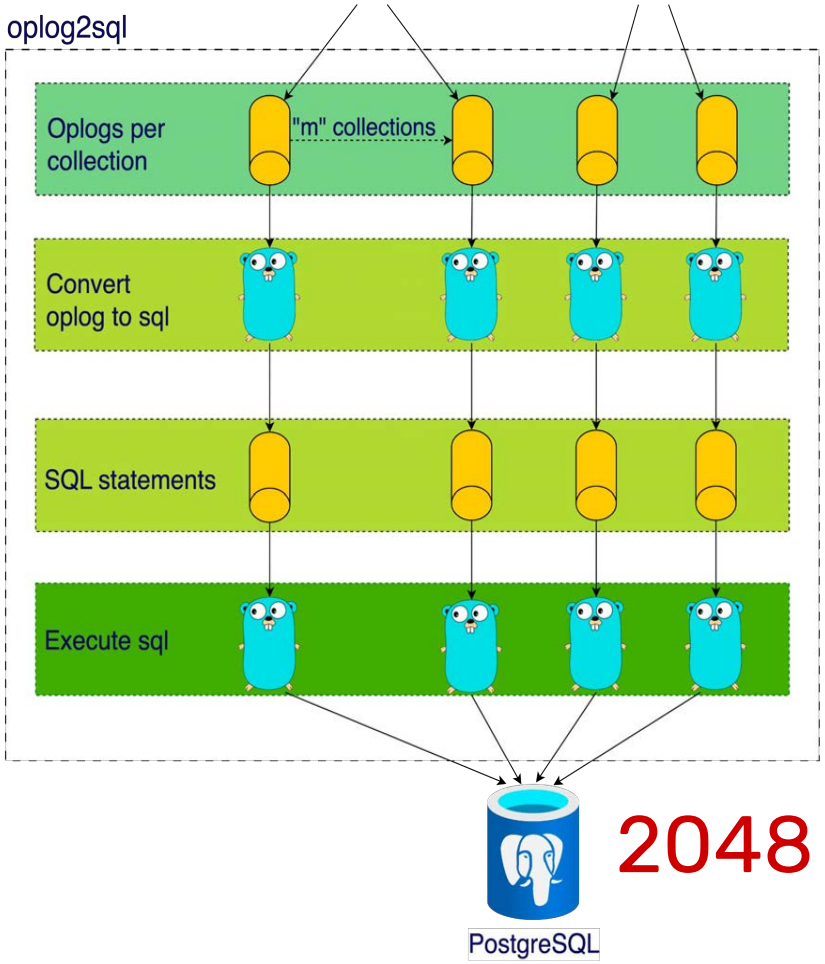
Concurrent Data Pipeline - Improvement





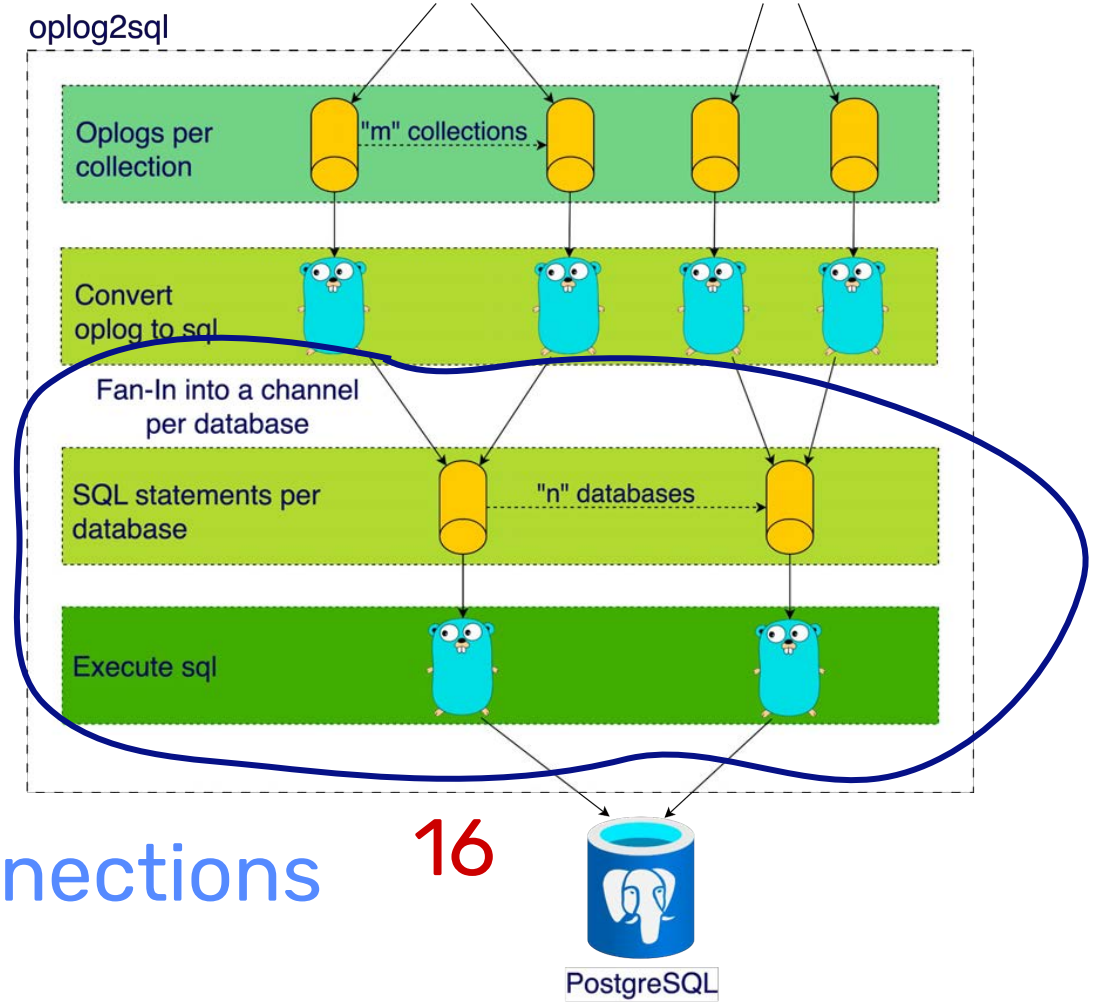
16 databases and 128 collections per db

Before



2048

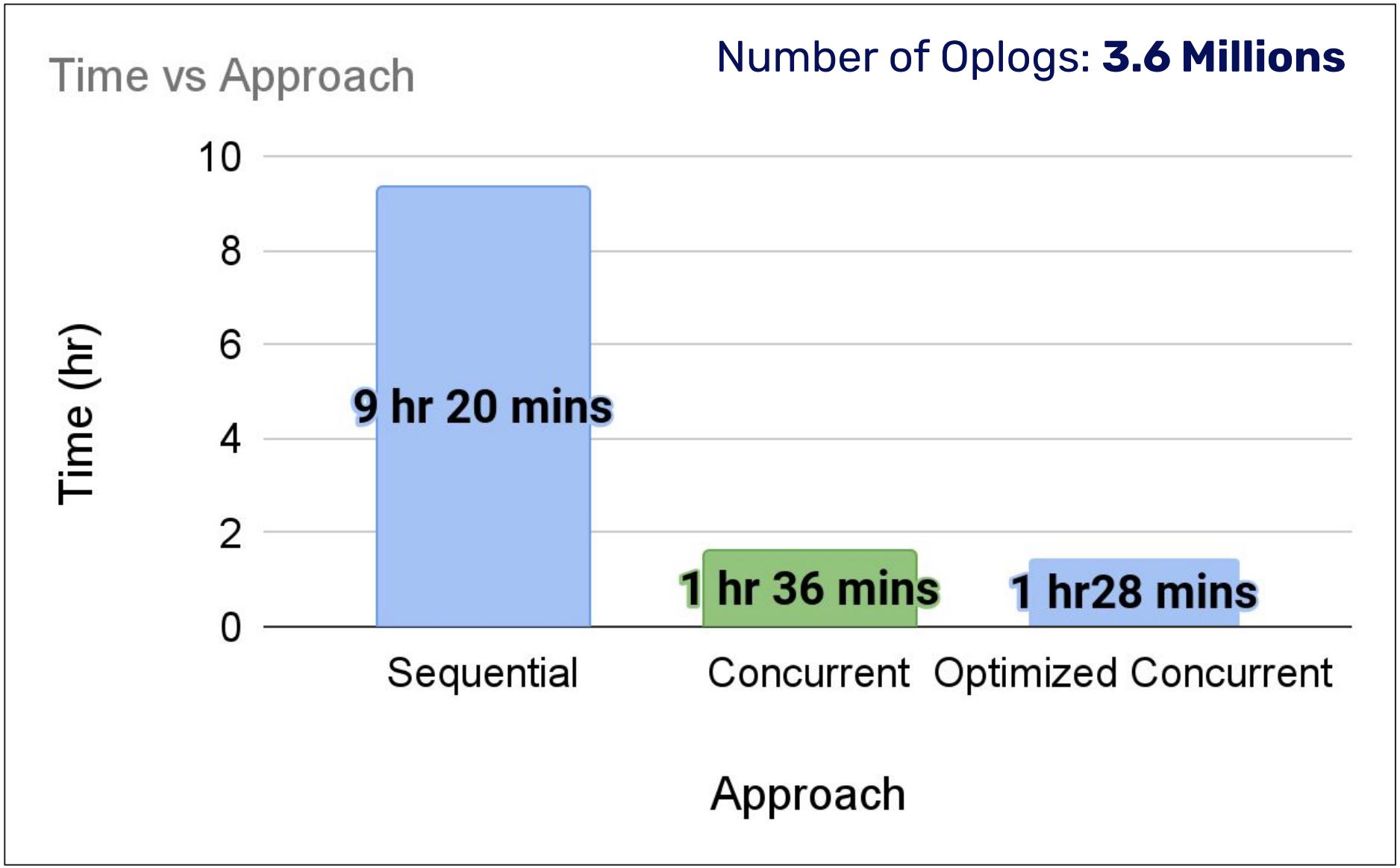
After



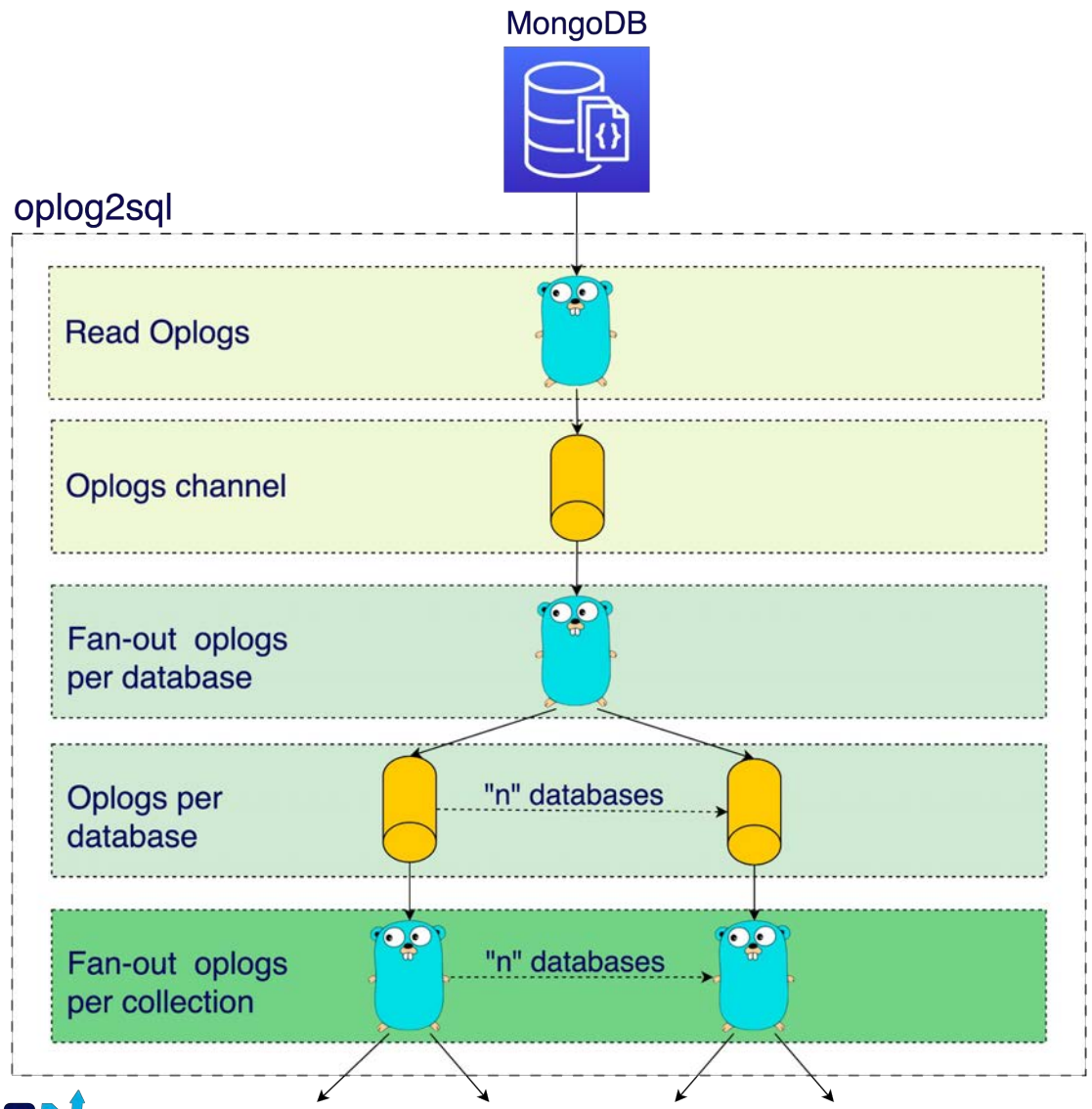
16

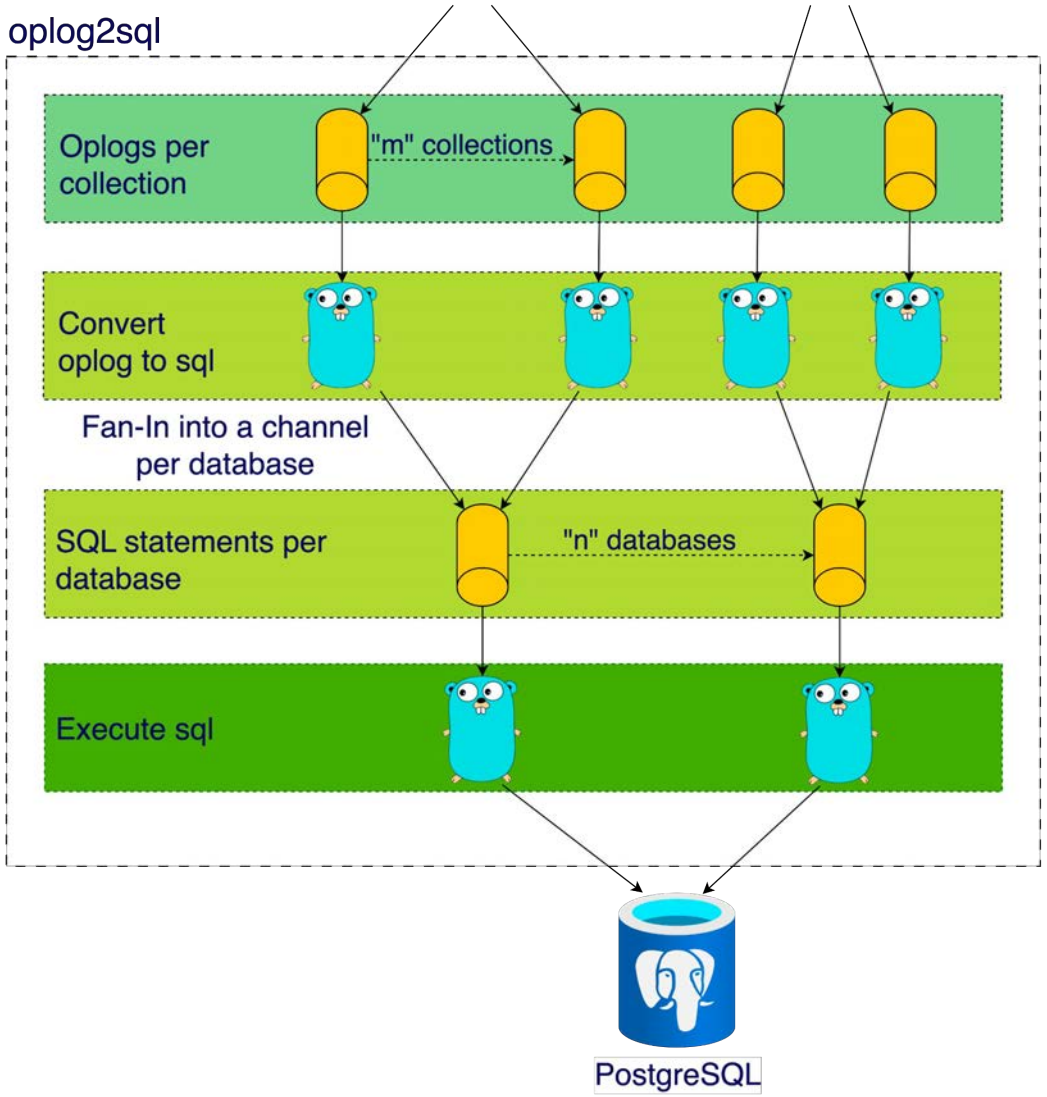
DB connections

Performance Comparison



Final Concurrent Data Pipeline





Key Takeaways

- Understand the problem domain (MongoDB to PostgreSQL data migration)
- Build the working solution first (sequential data pipeline)
- Identify the possible parallel portions of the program
- Avoid blindly applying concurrency patterns (worker pools and why it didn’t work)
- Consider Amdahl's Law
  - Simplicity may be more valuable than optimizing performance beyond a certain point
- Performant Concurrent Implementation with fan-in for database writes

Keep learning

- Connect with me **@chinmay185** on Twitter, LinkedIn and GitHub
- Check out One2N - https://one2n.in

https://playbook.one2n.in

MongoDB Oplog to SQL Parser exercise

Problem Statement

Story 1 (parsing insert oplog)

Story 2 (parsing update oplog)

Story 3 (parsing delete oplog)

Story 4 (create table with one oplog entry)

Story 5 (create table with multiple oplog entries)

Story 6 (alter table with multiple oplog entries)

Story 7 (handle nested Mongo documents)

Story 8 (reading oplogs from a file)

Story 9 (reading oplogs from MongoDB)

Story 10 (Bookmarking Support - nice to have)

Story 11 (Distributed Execution - nice to have)

Overall Instructions