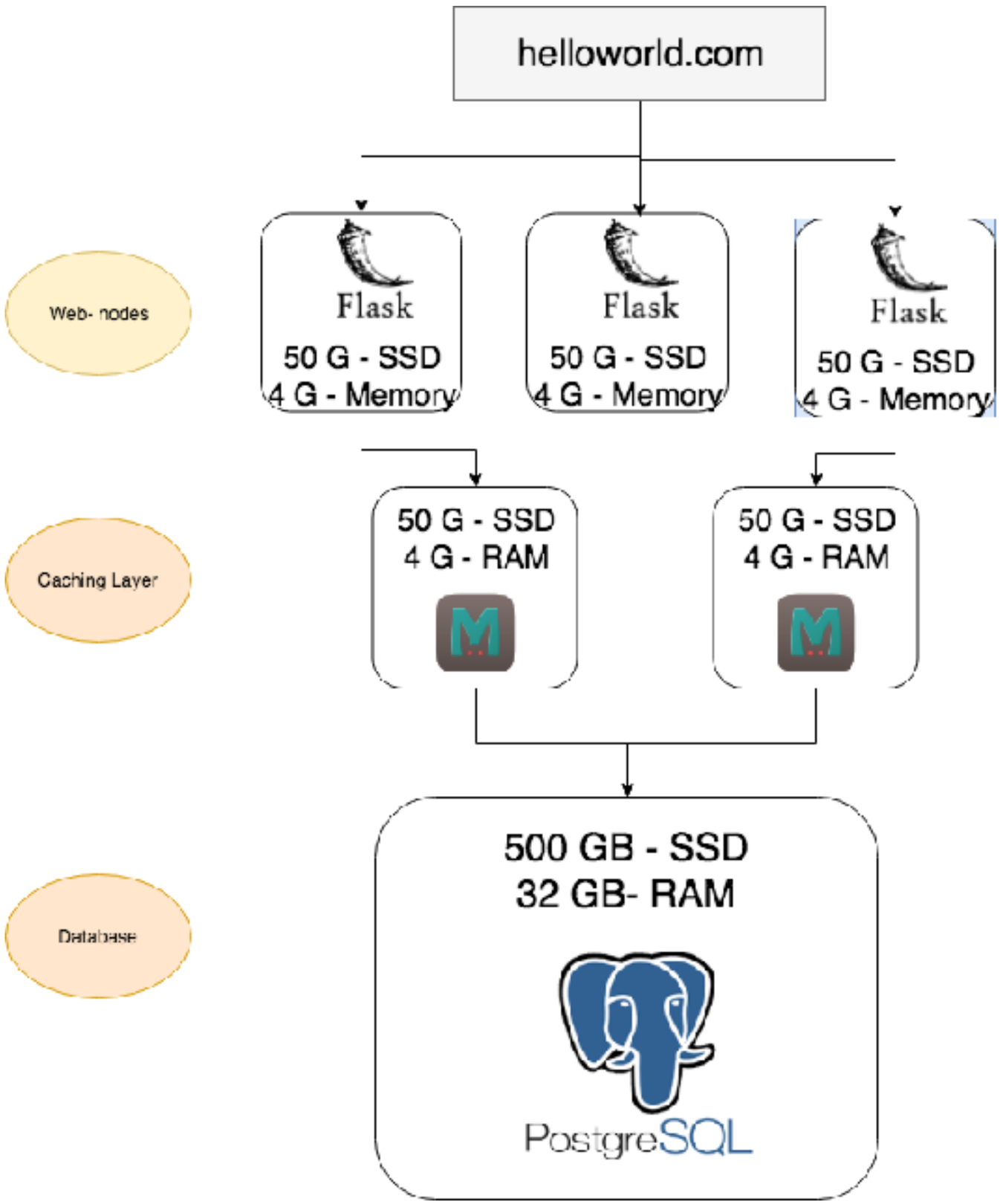
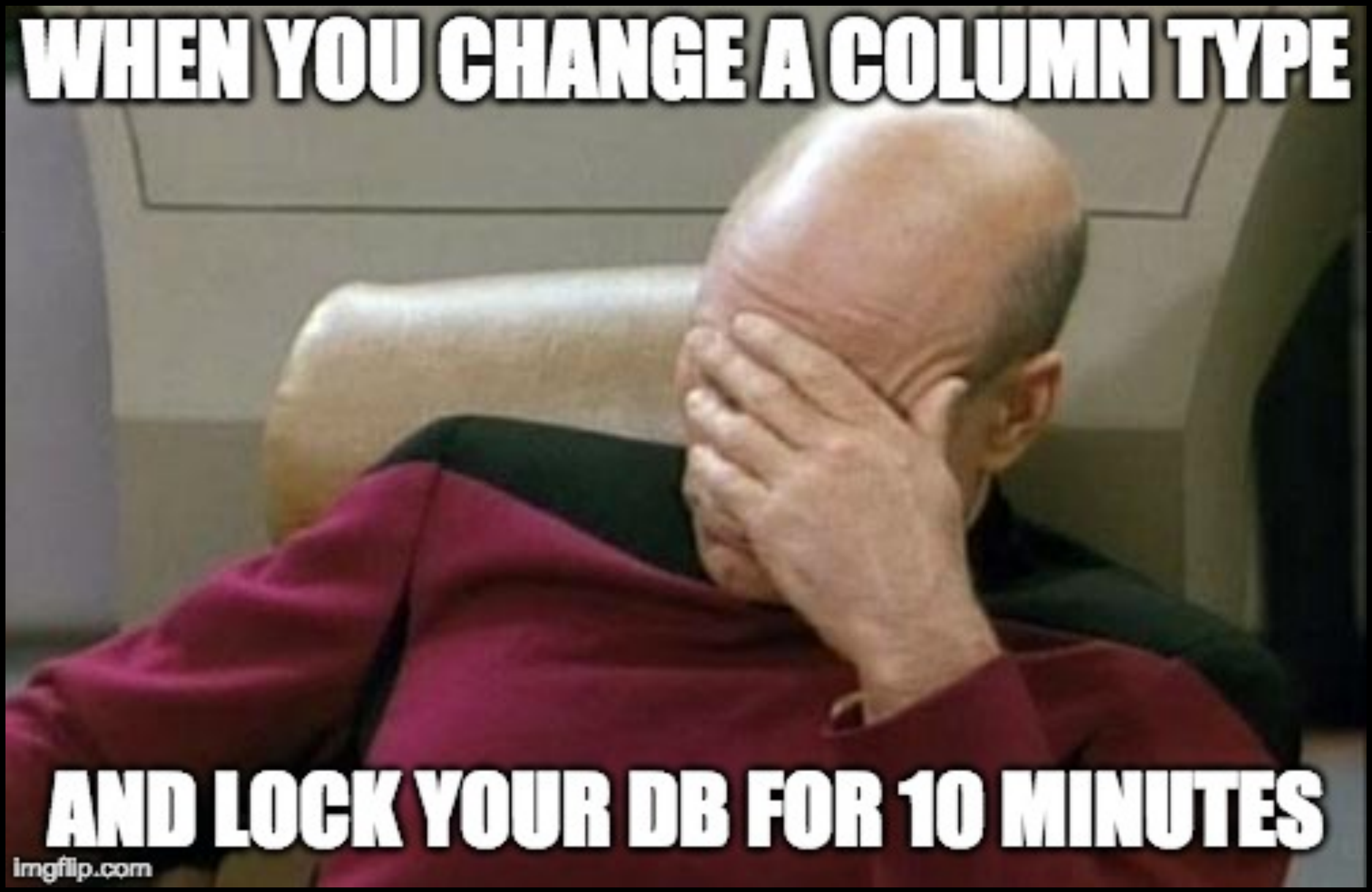
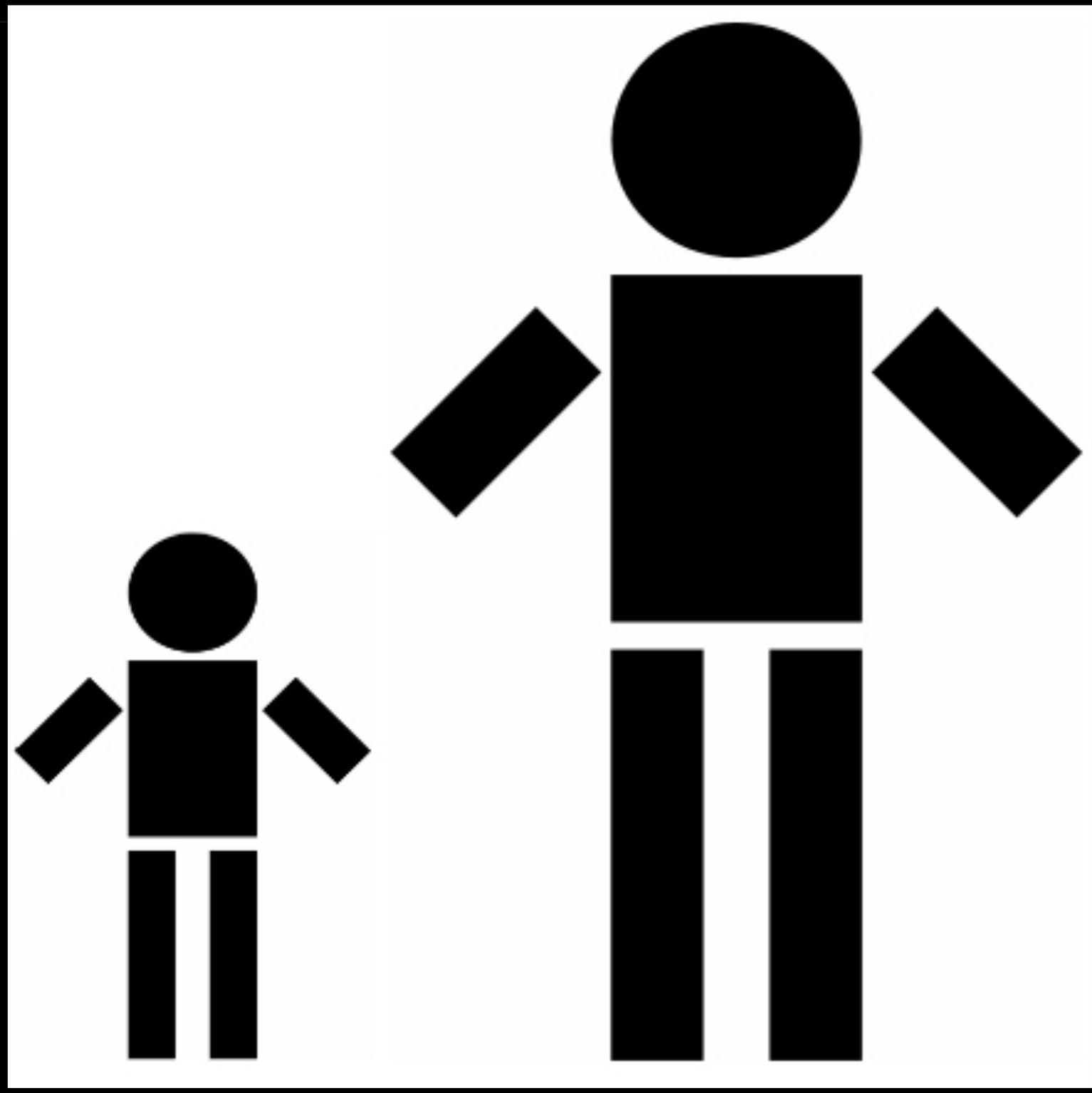
# HOW TO SCALE DATABASE MIGRATIONS

Jumana Bahrainwala, April 2018
@JumzB


helloworld.com
Web- nodes
Flask
50 G - SSD
4 G - Memory
Flask
50 G - SSD
4 G - Memory
Flask
50 G - SSD
4 G - Memory
Caching Layer
50 G - SSD
4 G - RAM
50 G - SSD
4 G - RAM
Database
500 GB - SSD
32 GB- RAM
PostgreSQL


WHEN YOU CHANGE A COLUMN TYPE
AND LOCK YOUR DB FOR 10 MINUTES
imgflip.com



scaling with tools

scaling with people

remaining issues getting to zero downtime

scaling with tools

Don’t lock yourself out

Break your migrations

Database + code compatibility

# don’t lock yourself out

Locks are the core of most Relational Databases.

Locks help with the concurrency and keeping data consistent in an RDBMS.

Locks are great… till they are NOT…especially when running migrations

“Look for locks that are conflicting.

Conflicting locks are what cause downtime”

# don’t lock yourself out

ACCESS SHARE - Select

# don’t lock yourself out

ACCESS SHARE - Select

ROW SHARE - Select for Update, Select for Share

# don’t lock yourself out

ACCESS SHARE - Select

ROW SHARE - Select for Update, Select for Share

ROW EXCLUSIVE - Update, Delete, Insert. Anything that modifies Data.

# don’t lock yourself out

ACCESS SHARE - Select

ROW SHARE - Select for Update, Select for Share

ROW EXCLUSIVE - Update, Delete, Insert. Anything that modifies Data.

ACCESS EXCLUSIVE - Alter Table, Drop Table, Truncate, Reindex, Cluster, Vacuum

# break your migrations

“Schema - organization of data as a blueprint of how the database is constructed”

# break your migrations

“Schema - organization of data as a blueprint of how the database is constructed”

## Schema Changes

- Create,  Alter commands change the underlying Data Definition of the DB.  They lock the table completely
- Want these migrations to be as quick as possible and in small transactions
- ACCESS EXCLUSIVE

# break your migrations

“An **index** is a copy of selected columns of **data** from a table that can be searched very efficiently that also includes a low-level disk block address or direct link to the complete row of **data** it was copied from.” - Wikipedia

# break your migrations

"An **index** is a copy of selected columns of **data** from a table that can be searched very efficiently that also includes a low-level disk block address or direct link to the complete row of **data** it was copied from." - Wikipedia

## Index Changes

- Adding an index can be done concurrently without locking a table (Postgres)
- If you are adding an index non-concurrently - recognize it will lock the table
- SHARE LOCK

# break your migrations

Reading from a table and writing to another can be long, especially during a migration.

# break your migrations

Reading from a table and writing to another can be long, especially during a migration.

## Data Migrations

- Chunk your data migrations
- And insert, update statements should all be in their own transactions as they are non-locking but long.
- ROW EXCLUSIVE

Back to Locking in theory

# don’t lock yourself out

```
Alter Table ORDERS
Alter column “final order total”
type Numeric(10, 4)
```

Acquires an ACCESS EXCLUSIVE LOCK

# don’t lock yourself out

```
Select * from Orders
```
ACCESS SHARE

```
Insert into Orders(amount)
Values(2)
```
ROW EXCLUSIVE

Both these locks conflict with ACCESS EXCLUSIVE

If the alter statement takes a long time as your orders table is large, your customers will not be able to READ or WRITE to the table at that point.

# don’t lock yourself out: ways to get around locking

Changing the type of a column

Add a new column with new type (Schema)

Write to both columns and then backfill new column. (Data)

New version of the code stops writing to the old column. (Process)

# don’t lock yourself out:
# ways to get around locking

Add a column that has a default:

Add column (SCHEMA)

Add default as a separate command (DATA)

Backfill the column with the default value (DATA)

# don’t lock yourself

| | |
|---|---|
| Add an Index | Add the index using the “CONCURRENTLY” keyword. |
| Add a column that is non-nullable | Create a new table with the addition of the non-nullable column, write to both tables, backfill, and then switch to the new table |
| Add a column with a unique constraint | Add Column, Add unique index concurrently and then add the constraint on the table. |

# database + code compatibility

- Your code should be backwards AND forward compatible
  - Read from BOTH old columns and new
- Don’t drop columns until multiple release versions ahead

scaling with your team

Migration User

Migration Tooling

Load Testing

Object Relational Mapping(ORM)

# migration - user account

## Make a user to run your migrations

If migrations can’t acquire a lock in a timely manner, other statements won’t be stuck behind it.

```
ALTER ROLE jzbahrai
SET lock_timeout='5s';
```

USERS TABLE

SELECT

SELECT

USERS TABLE

ALTER

SELECT

SELECT

USERS TABLE

# migration - user account

SELECT

SELECT

ALTER

SELECT

SELECT

USERS TABLE

SELECT

SELECT

SELECT

SELECT

SELECT

SELECT

SELECT

SELECT

SELECT

USERS TABLE

# migration tooling

1) Automate Scripts:
No one should be writing Create statements by going to your DB.

# migration tooling

```
 version_num
--------------
 39b980e419a9
 7ac4c29288a3
 16af97dd95c1
 24187609b002
 9d3ef281fb30
 8918b762b17b
 47f9fc04d6a7
 2badb70854a9
 1551e5d79984
```

1) Automate Scripts
No one should be writing Create statements by going to your DB

2) Have a way to know exactly what state your DB is in. Keep a hash table in your DB to indicate which migrations have run.

# migration tooling

```
 version_num
--------------
 39b980e419a9
 7ac4c29288a3
 16af97dd95c1
 24187609b002
 9d3ef281fb30
 8918b762b17b
 47f9fc04d6a7
 2badb70854a9
 1551e5d79984
```

1) Automate Scripts
No one should be writing Create statements by going to your DB

2) Have a way to know exactly what state your DB is in. Keep a hash table in your DB to indicate which migrations have run.

3) Easy set up on dev machines

# migration tooling

```
 version_num
--------------
 39b980e419a9
 7ac4c29288a3
 16af97dd95c1
 24187609b002
 9d3ef281fb30
 8918b762b17b
 47f9fc04d6a7
 2badb70854a9
 1551e5d79984
```

1) Automate Scripts
No one should be writing Create statements by going to your DB

2) Have a way to know exactly what state your DB is in. Keep a hash table in your DB to indicate which migrations have run.

3) Easy set up on dev machines

4) Use existing tooling

# load test your migrations (demo)

- Always run migrations during low traffic

# load test your migrations (demo)

- Always run migrations during low traffic
- What if low traffic still means you have 1000's of users on your site?

# load test your migrations (demo)

- Always run migrations during low traffic
- What if low traffic still means you have 1000's of users on your site?
- Load test your migration in stage to see where your application gets affected.
- Go back and either break up your migration further or ask for a maintenance window

# offload to your object relation model (ORM)

- Is your database the source of truth of your data or the ORM?

# offload to your object relation model (ORM)

- Is your database the source of truth of your data or the ORM?
- You can add sensible defaults to your ORM without having to let anyone go through SQL

# offload to your object relation model (ORM)

- Is your database the source of truth of your data or the ORM?
- You can add sensible defaults to your ORM without having to let anyone go through SQL
- If you do use your ORM as the source of truth:
  - Every model in every language needs to be constantly updated
  - You ORM needs to be backwardly compatible with your code

remaining issues getting to zero

downtime

Foreign Keys :(

Vacuuming

Slave and Master

# foreign keys :(

```
Alter Table Orders
Add constraint “user_fk”
foreign key(“user_id”) references
“users” (“id”)
```

Both ORDERS and USERS acquires an ACCESS EXCLUSIVE LOCK!

Both these tables can’t be read or written too while this is happening.

DO YOU REALLY NEED REFERENTIAL INTEGRITY?

# vaccuming

You can use https://github.com/reorg/pg_repack to vacuum tables without running a VACCUM FULL.

Vacuuming does lock the table to reads and writes and will create downtime otherwise.

# slave and master

As you scale out your slave and master instances, you want to move all your reads to the slave.

This means fast processing time on the master = less likely to have long locking schema changes

scaling with tools

Migration User

Migration Tooling

Break your migrations

Don’t lock yourself out

Foreign Keys :(

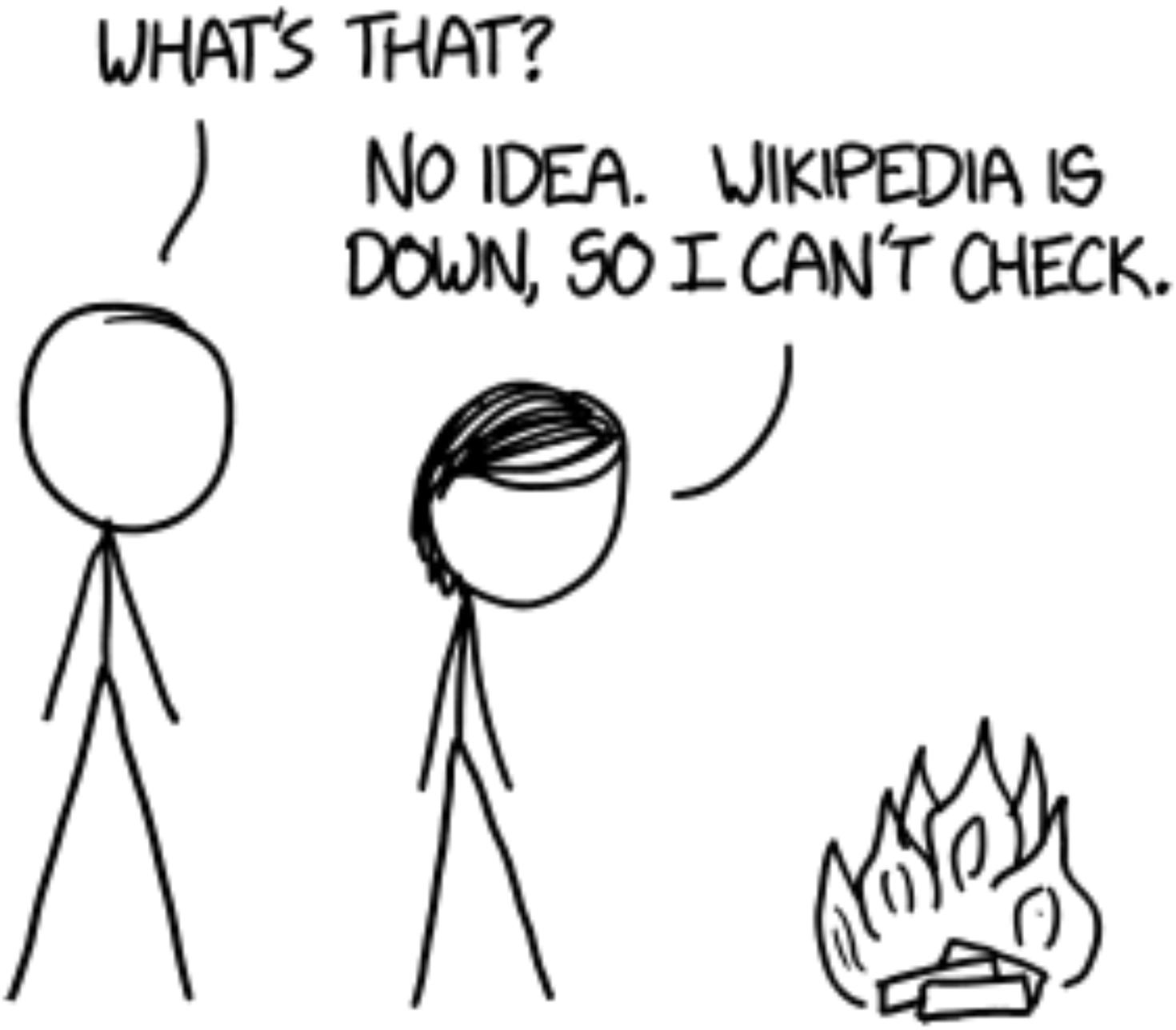
remaining issues getting to zero downtime

scaling with your team

# acknowledgments


- Nicolas Neu, Filipe Fernandes,  Alex Snurnikov, Zach Howard, Dileshni Jayasinghe and all the other wonderful folk @Unata


- Nordeus Engineering - PSQL Locking
- Travis of the North - PSQL adding FK with zero downtime
- Lob - Running Vaccum full with minimum downtime
- Sam Saffron - Managing DB schema Changes without downtime at Discord
- Citus - When PSQL blocks
- Postegresql - Explicit Locking


WHAT'S THAT?
NO IDEA. WIKIPEDIA IS DOWN, SO I CAN'T CHECK.

| Requested Lock Mode | Current Lock Mode | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE |
| ACCESS SHARE | | | | | | | | X |
| ROW SHARE | | | | | | | X | X |
| ROW EXCLUSIVE | | | | | | X | X | X |
| SHARE UPDATE EXCLUSIVE | | | | | X | X | X | X |
| SHARE | | | | X | X | X | X | X |
| SHARE ROW EXCLUSIVE | | | X | X | X | X | X | X |
| EXCLUSIVE | | X | X | X | X | X | X | X |
| ACCESS EXCLUSIVE | X | X | X | X | X | X | X | X |