HOW TO SCALE RELATIONAL DATABASE MIGRATIONS

Jumana Bahrainwala

@JumzB


```
   Table "public.users"

 Column |     Type      |
--------+---------------+
 id     |   integer     |
 active |   boolean     |
 first  |   string      |
 last   |   string      |
 email  |   string      |
```

TIME

```
BEGIN;

ALTER TABLE
USERS
RENAME
COLUMN
first TO
first_name;
```

```
   Table "public.users"

Column |      Type      |
-------+----------------+
id     |   integer      |
active |   boolean      |
first  |   string       |
last   |   string       |
email  |   string       |
```

TIME

```
BEGIN;

ALTER TABLE
USERS
RENAME
COLUMN
first TO
first_name;
```

```
BEGIN;

Select * from
Users where
id=2;

COMMIT;
```

```
   Table "public.users"

Column |      Type      |
-------+----------------+
id     |   integer      |
active |   boolean      |
first  |   string       |
last   |   string       |
email  |   string       |
```

TIME

```
BEGIN;

ALTER TABLE
USERS
RENAME
COLUMN
first TO
first_name;
```

```
BEGIN;

Select * from
Users where
id=2;

COMMIT;
```

```
    Table "public.users"

 Column |      Type      |
--------+----------------+
 id     |   integer      |
 active |   boolean      |
 first  |   string       |
 last   |   string       |
 email  |   string       |
```

```
  PID USER      PRI  NI  VIRT   RES S CPU% MEM%   TIME+  Command
 4511 root       32   0     0     0 ?  0.0  0.0   0:00.00 login
 4482 jzbahrai   24   0 4373M  3612 ?  0.0  0.0   0:00.00 postgres: jzbahrai jzbahrai ::1(56893) SELECT waiting
 4481 jzbahrai   25   0 4190M  5444 ?  0.0  0.0   0:00.01 psql -h localhost -U jzbahrai
```

TIME

```
BEGIN;

ALTER TABLE
USERS
RENAME
COLUMN
first TO
first_name;
```

```
BEGIN;

Select * from
Users where
id=2;

COMMIT;
```

```
COMMIT;
```

```
   Table "public.users"

 Column |      Type      |
--------+----------------+
 id     |   integer      |
 active |   boolean      |
 first  |   string       |
 last   |   string       |
 email  |   string       |
```

```
  PID USER      PRI  NI  VIRT   RES S CPU% MEM%   TIME+  Command
 4511 root       32   0     0     0 ?  0.0  0.0   0:00.00 login
 4482 jzbahrai   24   0 4373M  3612 ?  0.0  0.0   0:00.00 postgres: jzbahrai jzbahrai ::1(56893) SELECT waiting
 4481 jzbahrai   25   0 4190M  5444 ?  0.0  0.0   0:00.01 psql -h localhost -U jzbahrai
```

TIME

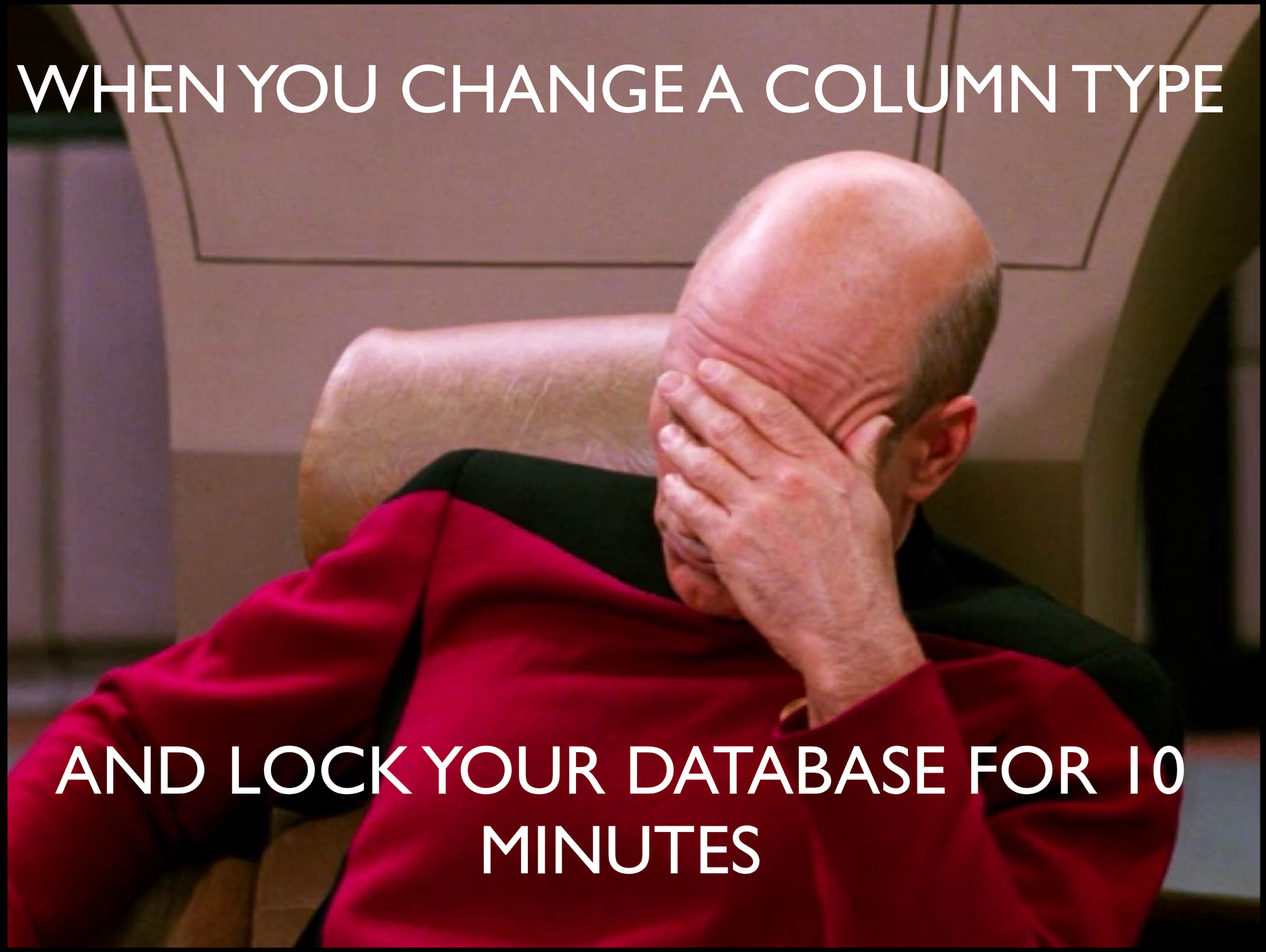
```
BEGIN;

ALTER TABLE
USERS
RENAME
COLUMN
first TO
first_name;
```

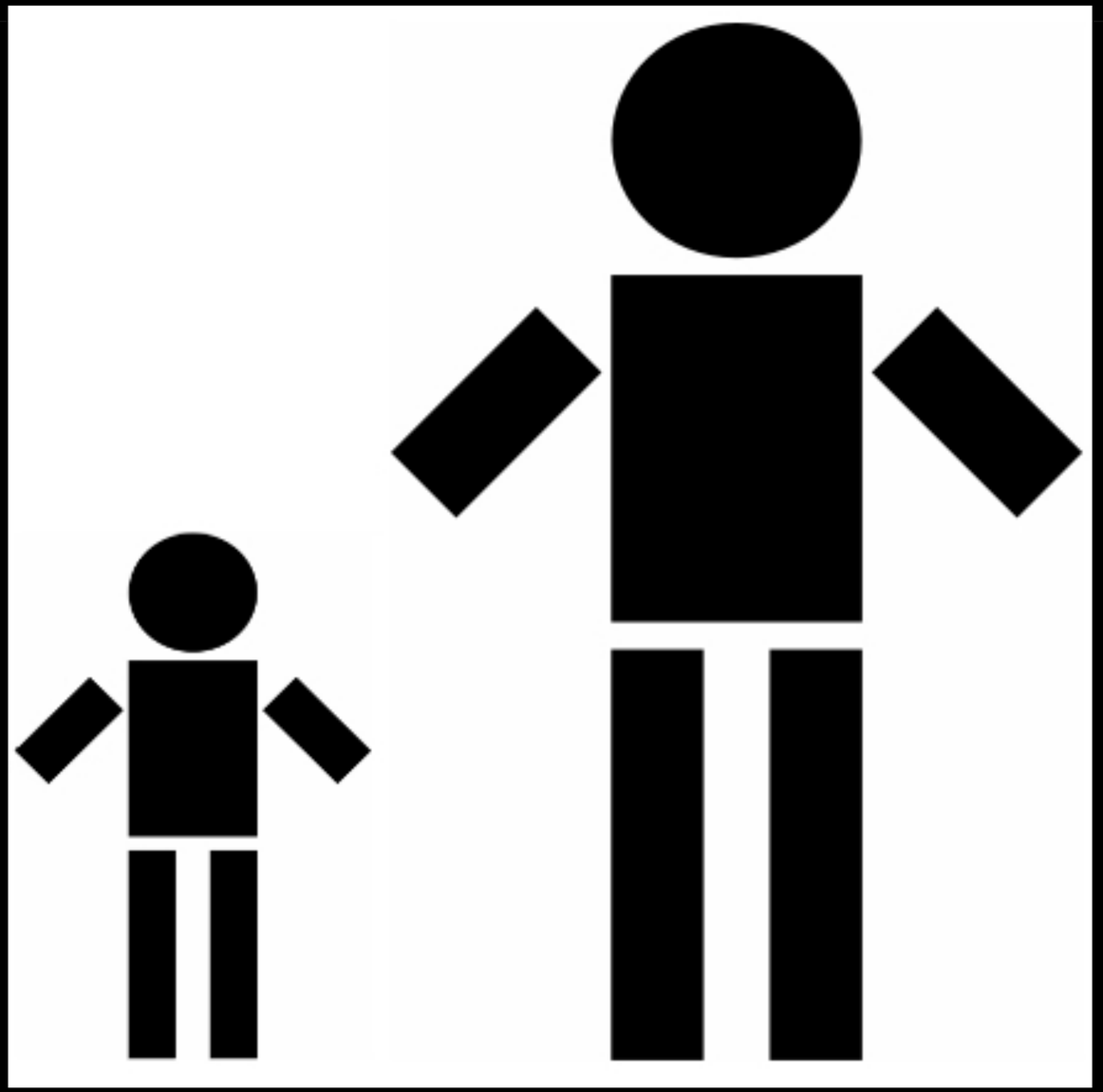
```
BEGIN;

Select * from
Users where
id=2;

COMMIT;
```

```
COMMIT;
```

```
Table "public.users"

Column |      Type      |
-------+----------------+
id     |   integer      |
active |   boolean      |
first  |   string       |
last   |   string       |
email  |   string       |
```


```
  PID USER      PRI  NI  VIRT   RES S CPU% MEM%   TIME+  Command
 4511 root       32   0     0     0 ?  0.0  0.0   0:00.00 login
 4482 jzbahrai   24   0 4373M  3612 ?  0.0  0.0   0:00.00 postgres: jzbahrai jzbahrai ::1(56893) SELECT waiting
 4481 jzbahrai   25   0 4190M  5444 ?  0.0  0.0   0:00.01 psql -h localhost -U jzbahrai
```


WHEN YOU CHANGE A COLUMN TYPE
AND LOCK YOUR DATABASE FOR 10 MINUTES



“Most of these examples are Postgres specific,
but the concepts apply to most SQL like DBs.”

scaling with tools

scaling with people

remaining issues getting to zero downtime

Don’t lock yourself out

scaling with tools

don’t lock yourself out

don’t lock yourself out

don’t lock yourself out

don’t lock yourself out

don’t lock yourself out

don’t lock yourself out

don’t lock yourself out

ACCESS EXCLUSIVE
Alter Table
Drop Table
Truncate
Reindex
Cluster
Vacuum

don’t lock yourself out

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

don’t lock yourself out

READS

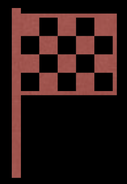
READS

don’t lock yourself out

READS

READS

READS



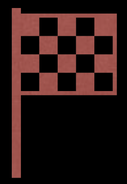
WRITES

don’t lock yourself out

READS

READS

READS



WRITES

READS,
WRITES

DDL CHANGES

“Look for locks that are conflicting.

Conflicting locks are what cause downtime”

Don’t lock yourself out

scaling with tools

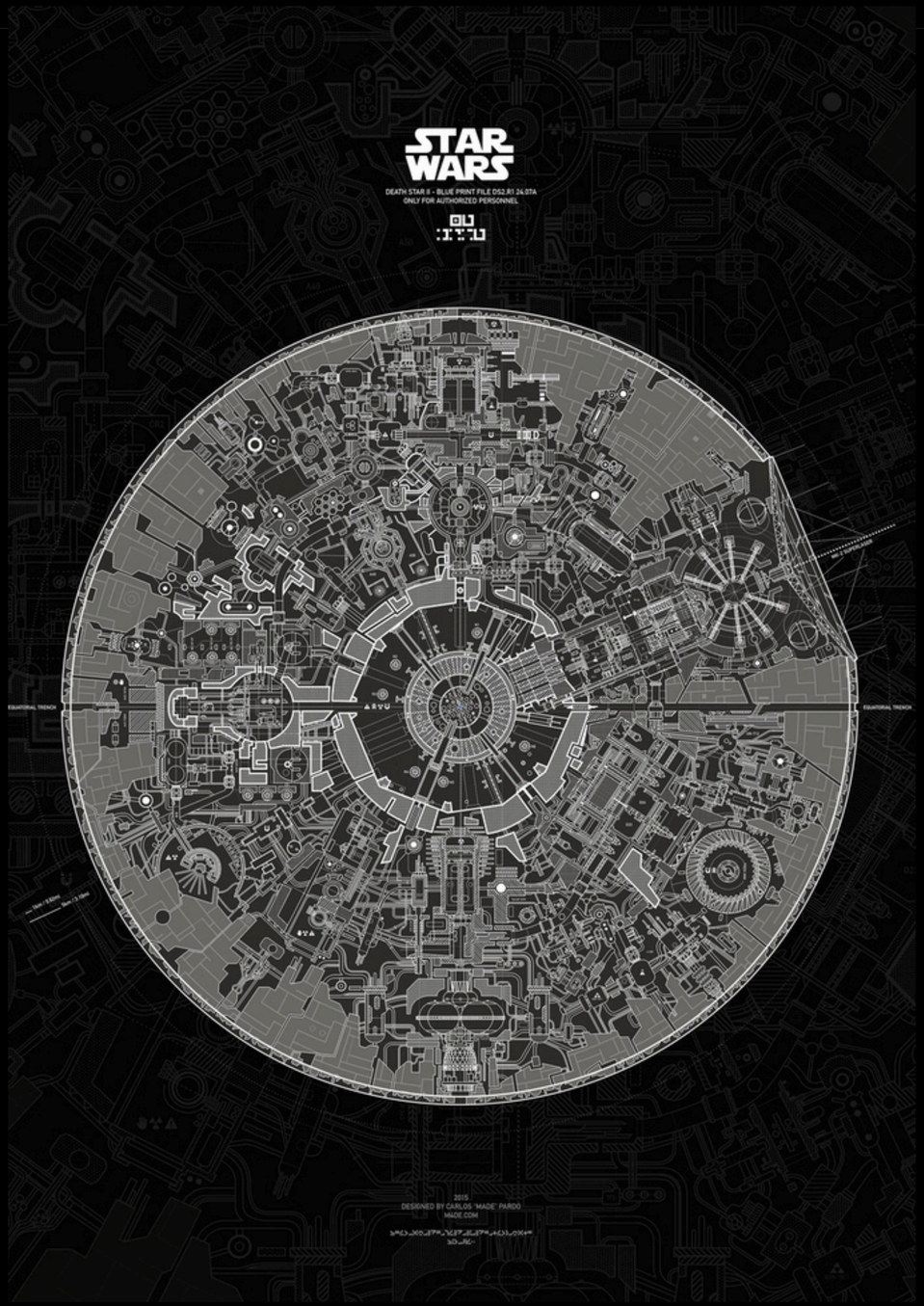
scaling with tools

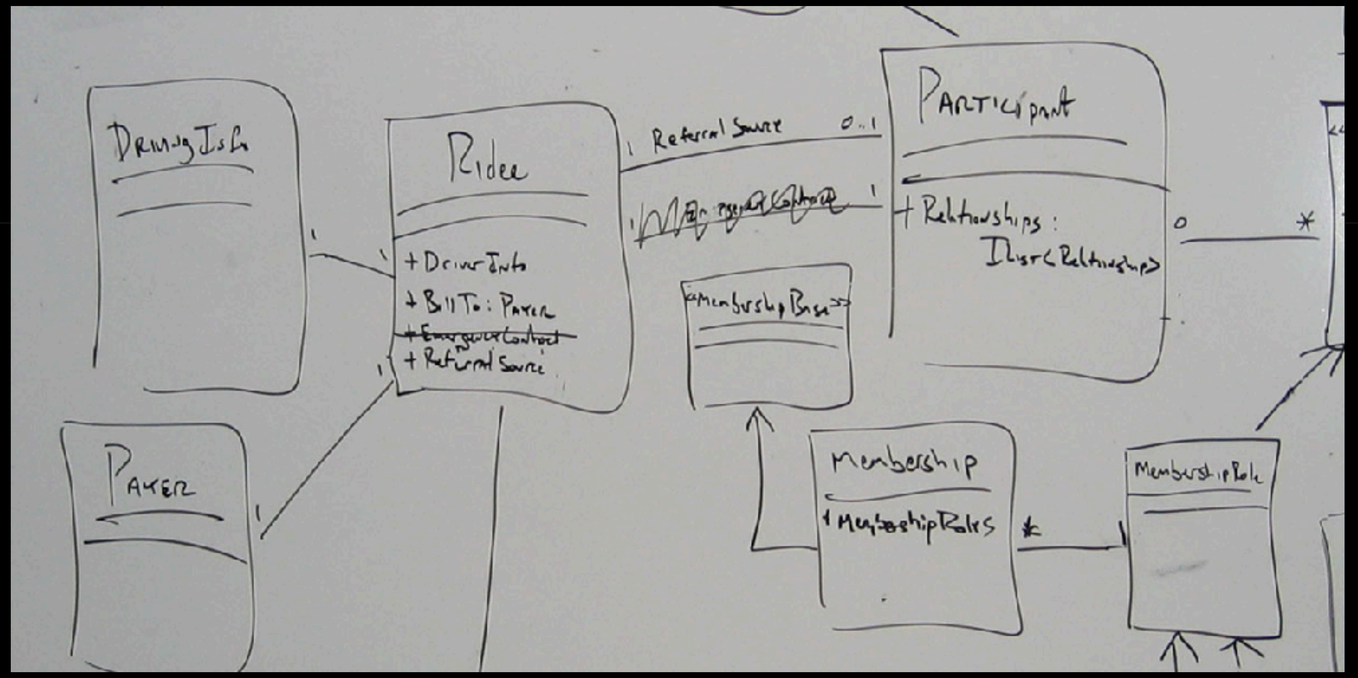
Don’t lock yourself out

Break your migrations

Schema Changes

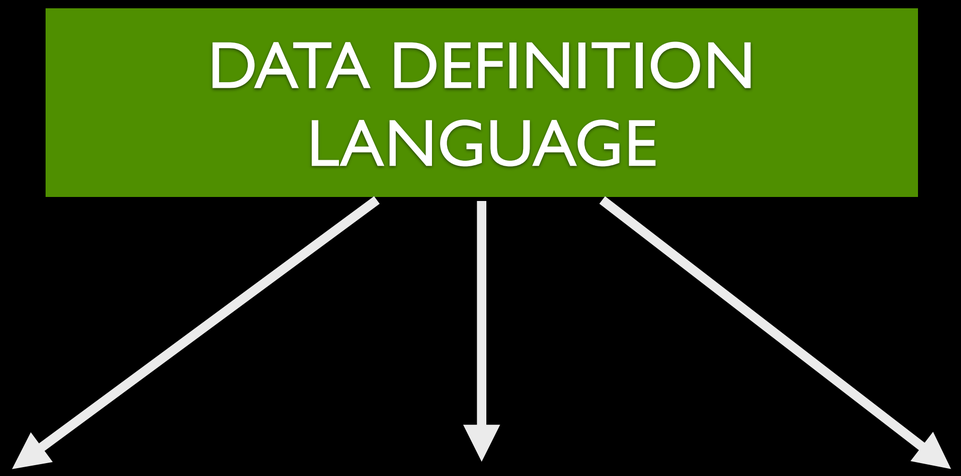
Schema Changes

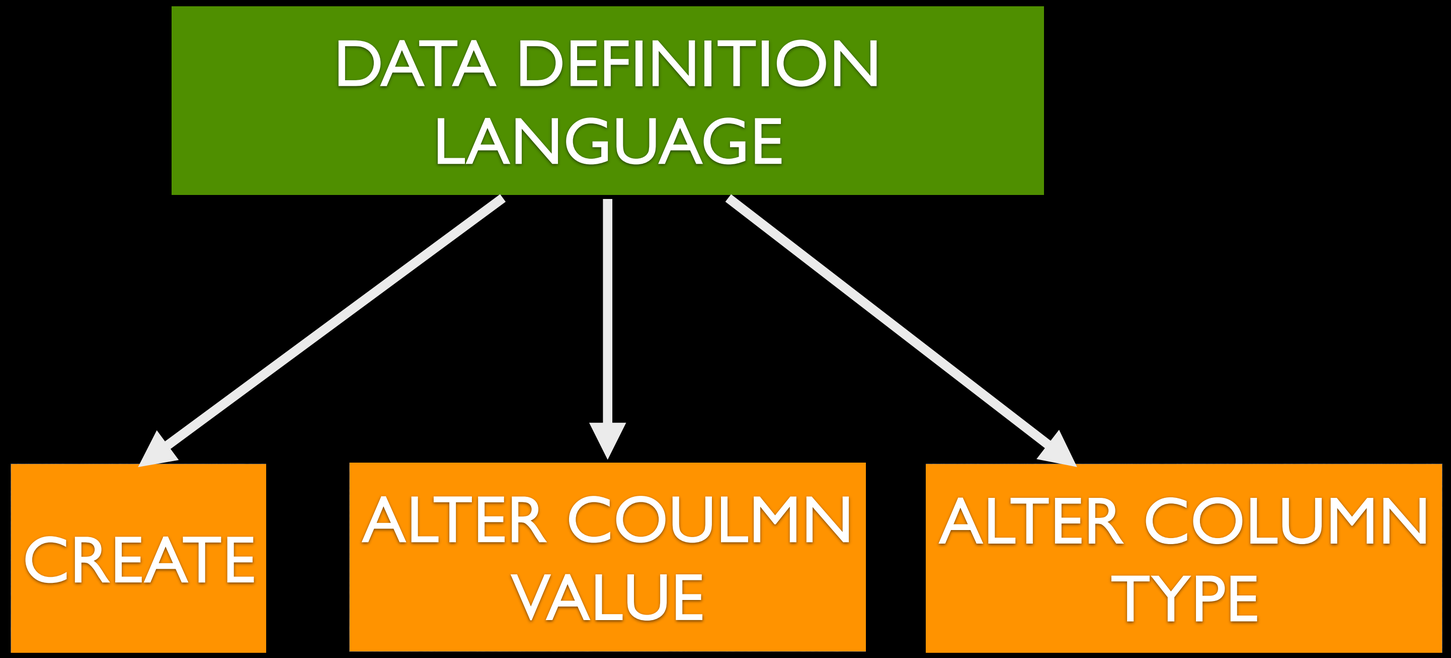


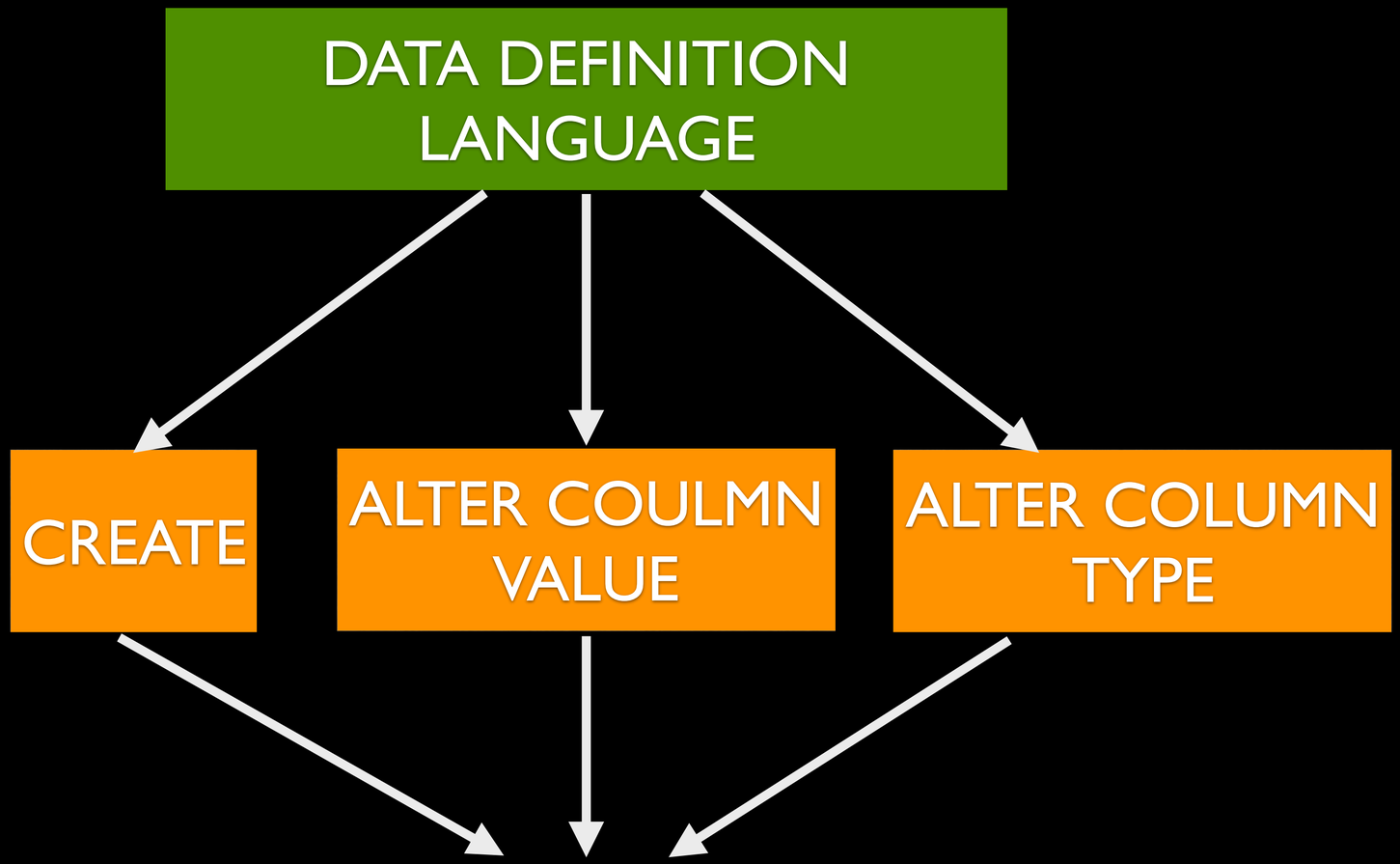


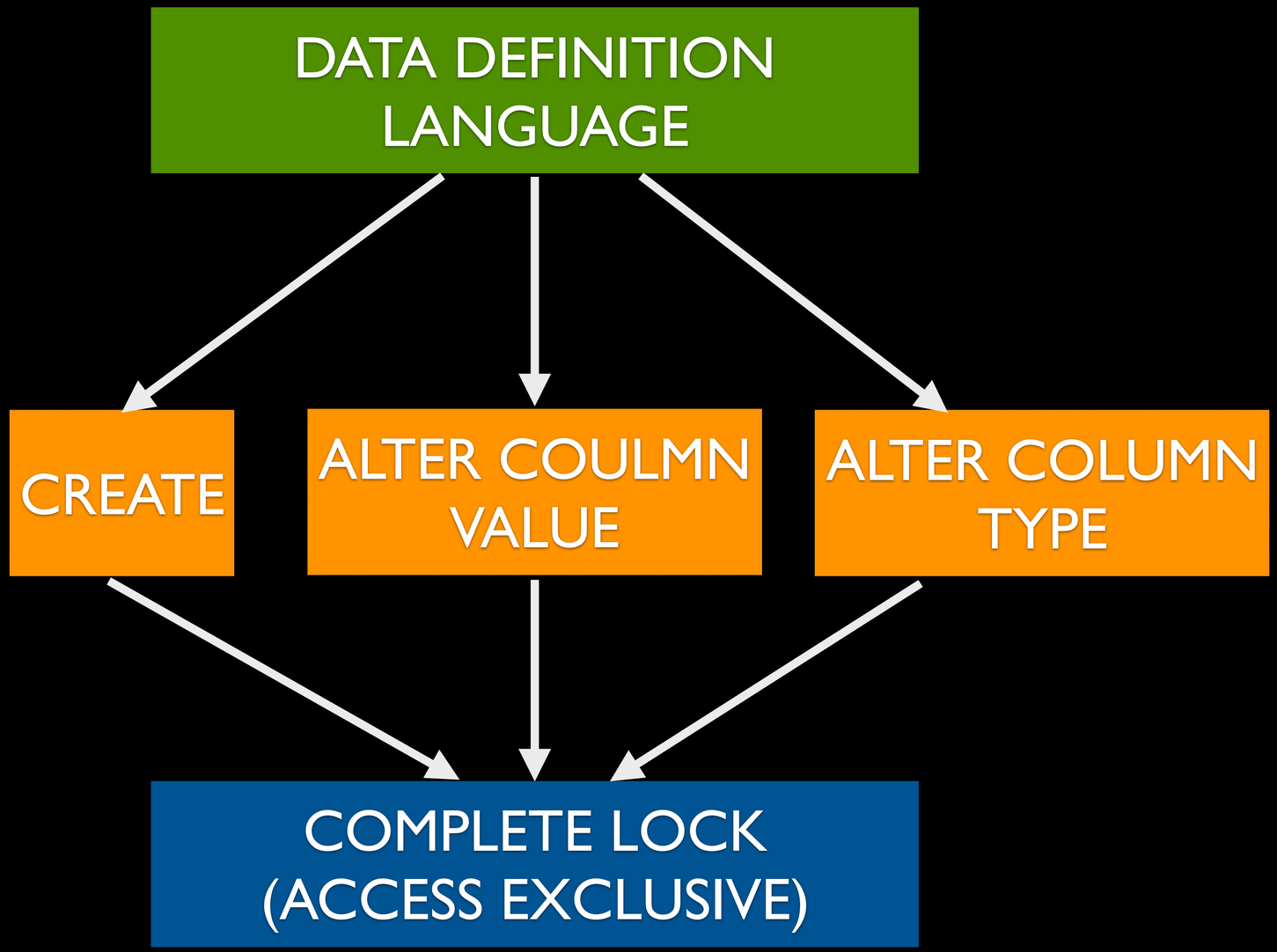
Data Definition Language

DATA DEFINITION LANGUAGE


DATA DEFINITION
LANGUAGE


DATA DEFINITION LANGUAGE
CREATE
ALTER COULMN VALUE
ALTER COLUMN TYPE


DATA DEFINITION LANGUAGE
CREATE
ALTER COULMN VALUE
ALTER COLUMN TYPE


DATA DEFINITION LANGUAGE
CREATE
ALTER COULMN VALUE
ALTER COLUMN TYPE
COMPLETE LOCK (ACCESS EXCLUSIVE)

break your migrations

Schema Changes

- QUICK
- SMALL TRANSACTIONS

break your migrations

Index Changes

break your migrations

An **index** is a way to access some data quickly.

break your migrations

Index Changes

- CONCURRENTLY
- NON-CONCURRENT will LOCK (expensive)
- SHARE LOCK

break your migrations

Data Changes

break your migrations

Data Changes

Copy Data

Insert, Update
statements should
all be in their own
transactions

ROW EXCLUSIVE

break your migrations

Data Changes

Copy Data

Insert, Update statements should all be in their own transactions

ROW EXCLUSIVE

Schema

```
CREATE TABLE
Users(
    id bigint
    first_name varchar(),
    last_name varchar(),
    email varchar(),
    phone varchar(),
    PRIMARY KEY (id)
);
```

Schema

Second Schema Migration

```
CREATE TABLE
Users(
    id bigint
    first_name varchar(),
    last_name varchar(),
    email varchar(),
    phone varchar(),
    PRIMARY KEY (id)
);
```

```
ALTER TABLE
Users
ADD COLUMN
age_int int;

ALTER TABLE
Users
ADD COLUMN
status boolean
```

Schema

```
CREATE TABLE
Users(
    id bigint
    first_name varchar(),
    last_name varchar(),
    email varchar(),
    phone varchar(),
    PRIMARY KEY (id)
);
```

Second Schema Migration

```
ALTER TABLE
Users
ADD COLUMN
age_int int;

ALTER TABLE
Users
ADD COLUMN
status boolean
```

Index Migration

```
CREATE
INDEX
CONCURRENTLY
status_email
ON Users
(email, status)
```

Schema	Second Schema Migration	Index Migration	Third Schema Migration
CREATE TABLE Users(id bigint first_name varchar(), last_name varchar(), email varchar(), phone varchar(), PRIMARY KEY (id));	ALTER TABLE Users ADD COLUMN age_int int; ALTER TABLE Users ADD COLUMN status boolean	CREATE INDEX CONCURRENTLY status_email ON Users (email, status)	**ALTER TABLE Users ADD COLUMN active_date datetime;**

Schema	Second Schema Migration	Index Migration	Third Schema Migration	Data Migration
CREATE TABLE Users(id bigint first_name varchar(), last_name varchar(), email varchar(), phone varchar(), PRIMARY KEY (id));	ALTER TABLE Users ADD COLUMN age_int int; ALTER TABLE Users ADD COLUMN status boolean	CREATE INDEX CONCURRENTLY status_email ON Users (email, status)	ALTER TABLE Users ADD COLUMN active_date datetime;	**UPDATE Users SET active_date=now() WHERE status**

Back to Locking in theory

Schema	Second Schema Migration	Index Migration	Third Schema Migration	Data Migration
CREATE TABLE Users(id bigint, first_name varchar(), last_name varchar(), email varchar(), phone varchar(), PRIMARY KEY (id));	ALTER TABLE Users ADD COLUMN age_int int; ALTER TABLE Users ADD COLUMN status boolean ACCESS EXCLUSIVE	CREATE INDEX CONCURRENTLY status_email ON Users (email, status)	ALTER TABLE Users ADD COLUMN active_date datetime; ACCESS EXCLUSIVE	UPDATE Users SET active_date=now() WHERE status

Second Schema Migration

ALTER TABLE
Users
ADD COLUMN
age_int int;

ALTER TABLE
Users
ADD COLUMN
status boolean

ACCESS EXCLUSIVE

SELECT *
FROM USERS

Third Schema Migration

ALTER TABLE
Users
ADD COLUMN
active_date
datetime;

ACCESS EXCLUSIVE

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)

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)

scaling with tools

Don’t lock yourself out

Break your migrations

scaling with tools

Don’t lock yourself out

Break your migrations

Database + Code Compatibility

database + code compatibility

- Backwards AND forward compatible
- Dropping columns (future)

scaling with tools

Don’t lock yourself out

Break your migrations

Database + Code Compatibility

Migration User

scaling with your
team

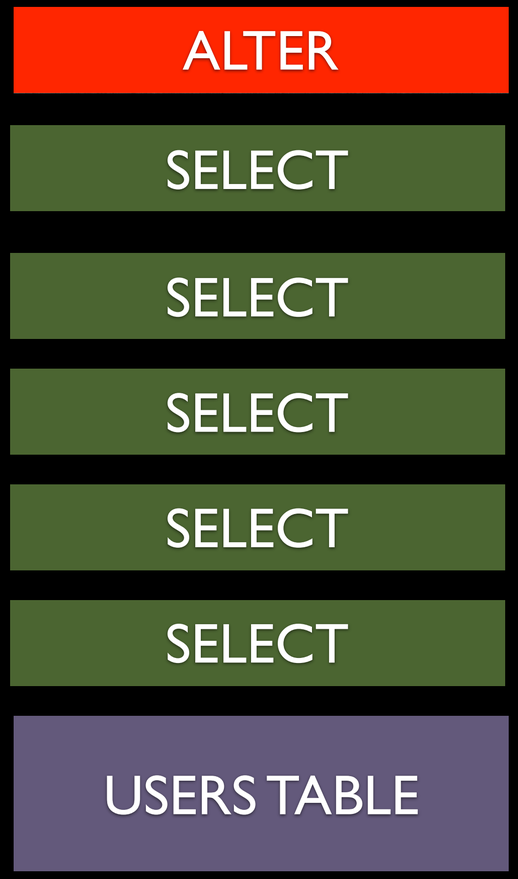
migration - user account

Make a user to run your migrations

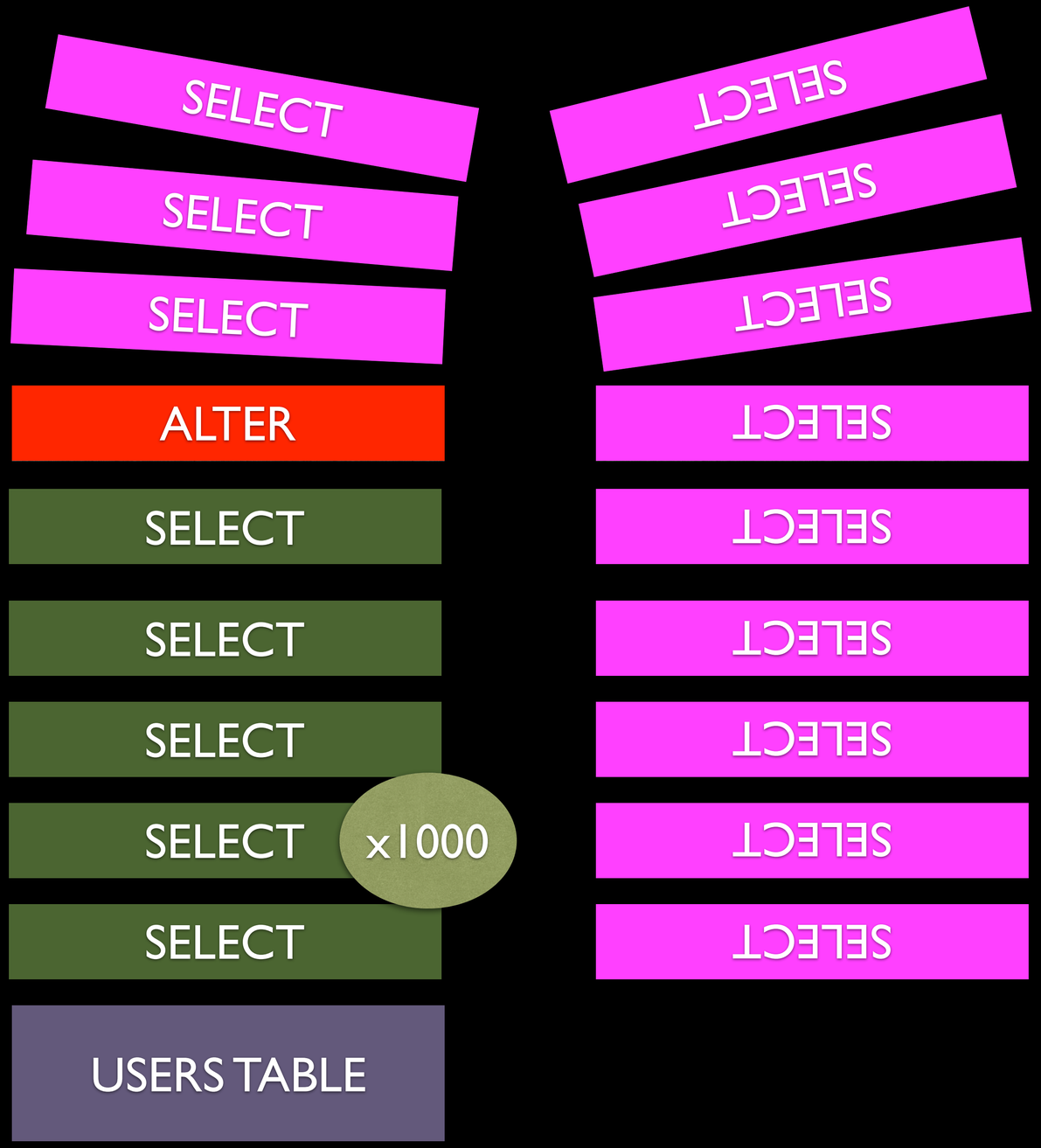
```
ALTER ROLE jumana
SET lock_timeout='5s';
```

USERS TABLE


SELECT
SELECT
SELECT
SELECT
SELECT
USERS TABLE


ALTER
SELECT
SELECT
SELECT
SELECT
SELECT
USERS TABLE


SELECT
SELECT
SELECT
ALTER
SELECT
SELECT
SELECT
SELECT
SELECT
USERS TABLE


SELECT
SELECT
SELECT
ALTER
SELECT
SELECT
SELECT
SELECT
x1000
SELECT
USERS TABLE
SELECT
SELECT
SELECT
SELECT
SELECT
SELECT
SELECT
SELECT

SELECT

SELECT

SELECT

ALTER

SELECT

SELECT

SELECT

SELECT

x1000

SELECT

USERS TABLE

SELECT

SELECT

SELECT

SELECT

SELECT

SELECT

SELECT

SELECT

scaling with your team

Migration User

Migration Tooling

migration tooling

iTerm2 Shell Edit View Session Profiles Toolbelt Window Help

~ psql -U jz -h localhost

jz=> create table users

migration tooling

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

Automate Scripts

migration tooling

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

Automate Scripts

State of your DB

migration tooling

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

Automate Scripts

State of your DB

Easy replication on local dev

scaling with your team

Migration User

Migration Tooling

Load Testing

load test your migrations

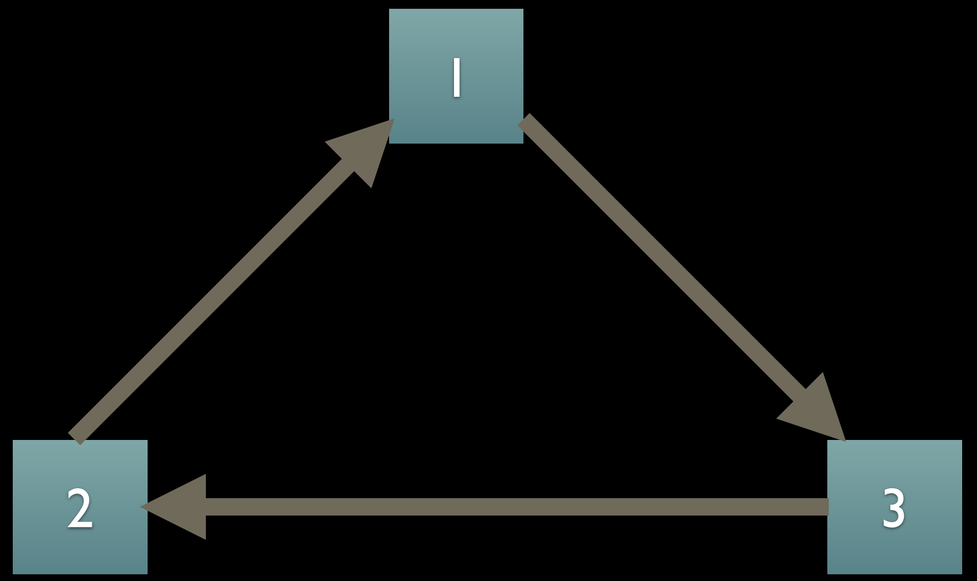
- Run migrations during low traffic

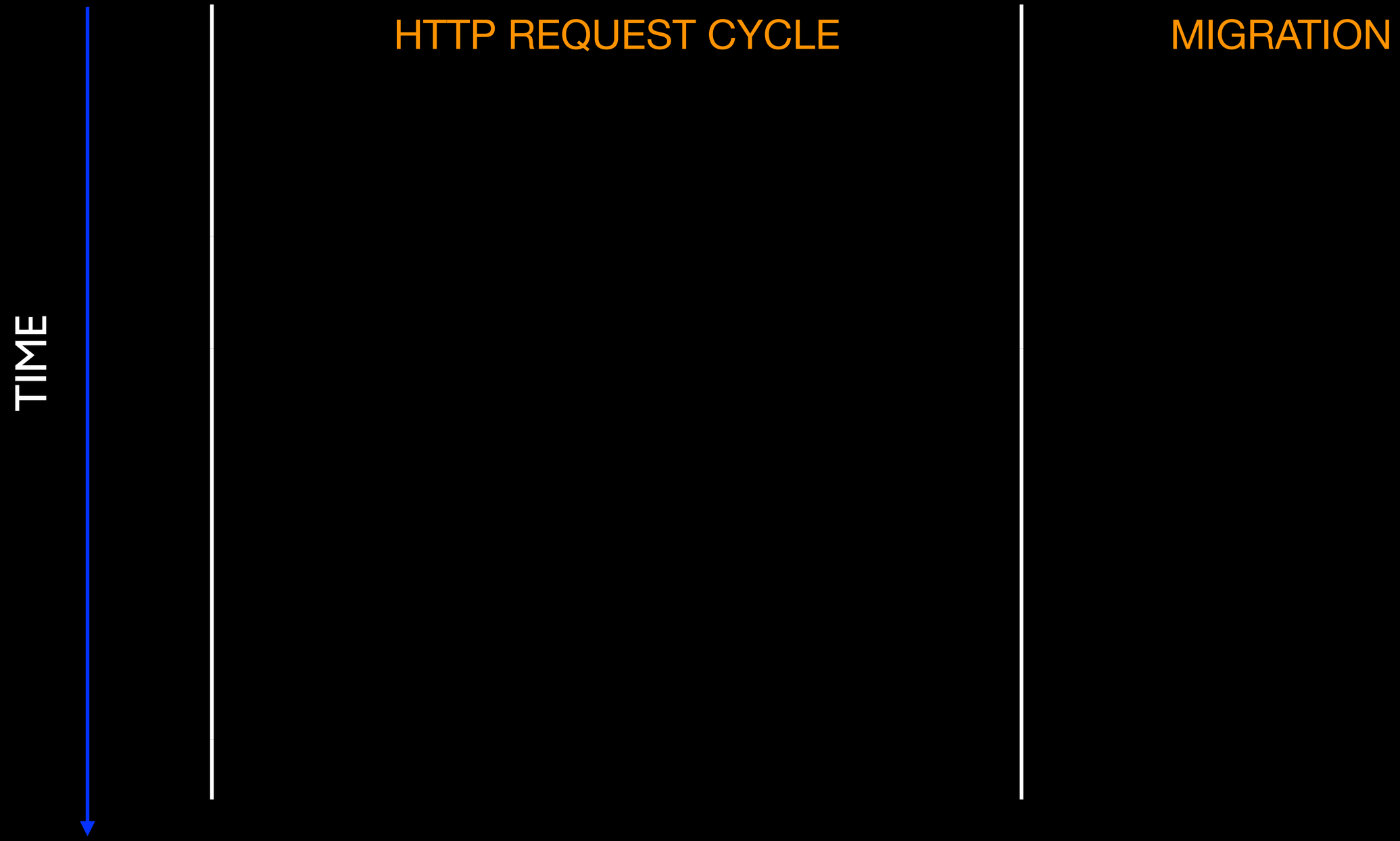
load test your migrations

- Run migrations during low traffic
- What if low traffic still means you have 1000’s of users on your site?

load test your migrations

- Run migrations during low traffic
- What if low traffic still means you have 1000’s of users on your site?
  - Load test your migration


1
2
3

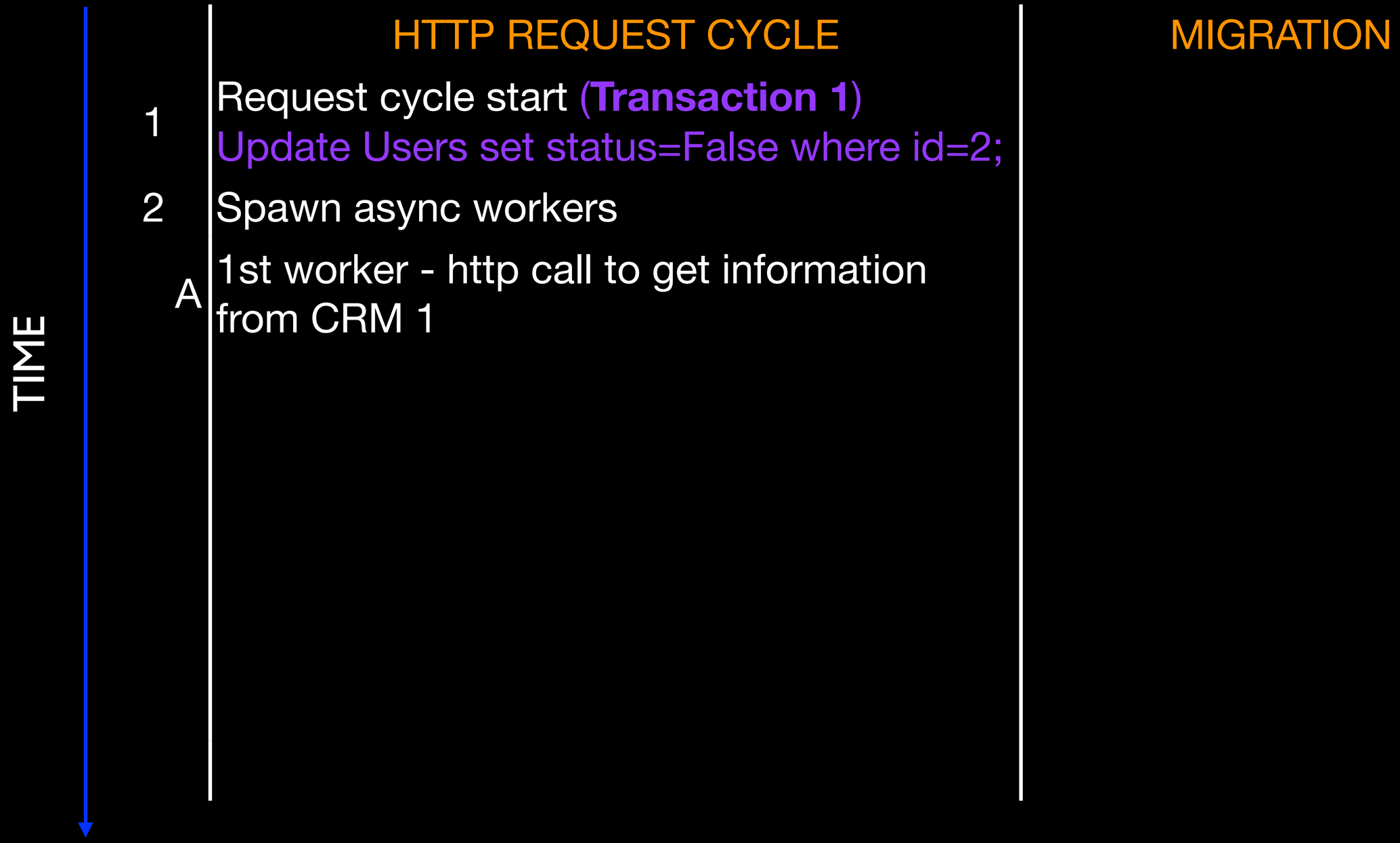

HTTP REQUEST CYCLE
MIGRATION
TIME

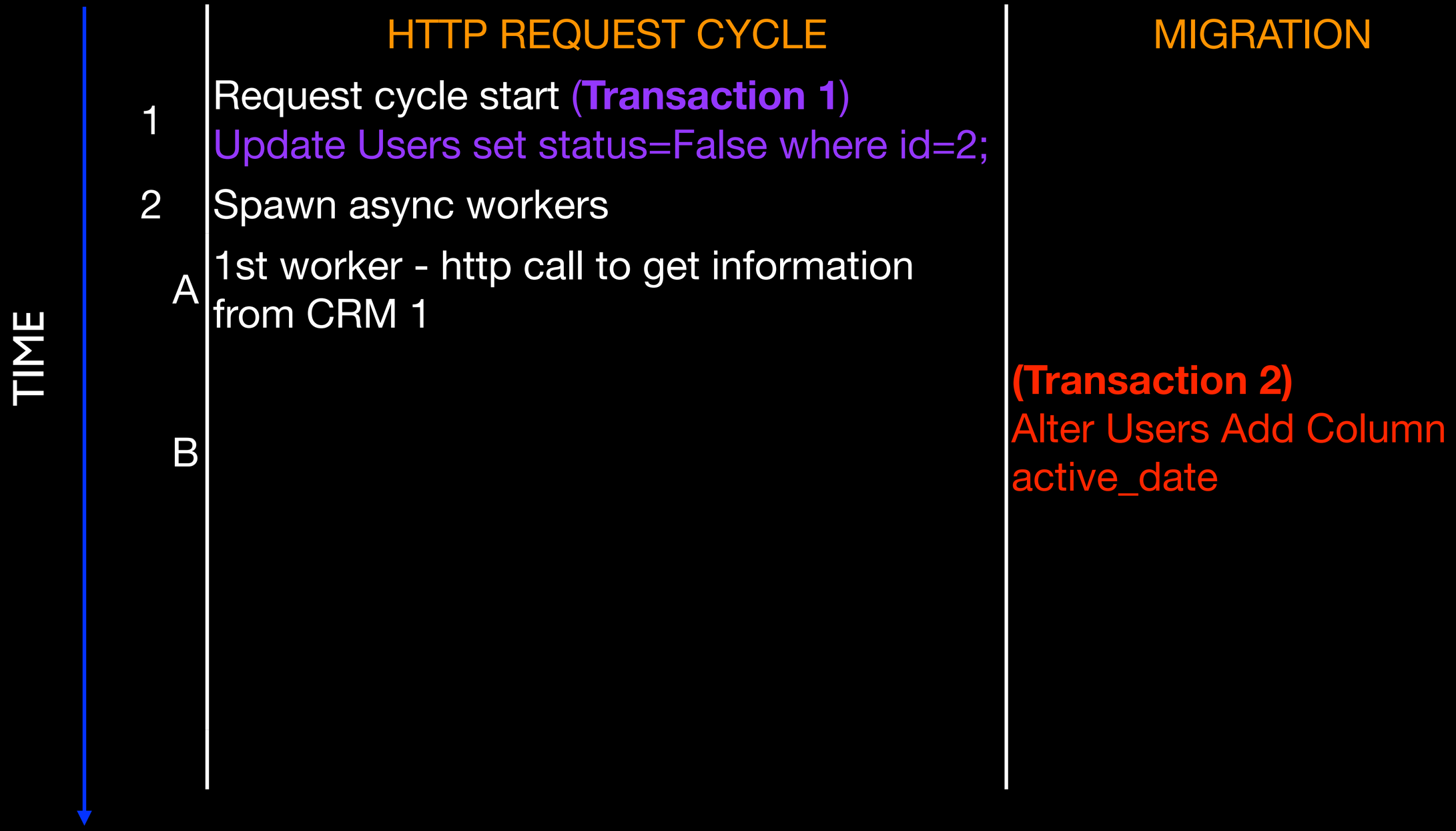
TIME

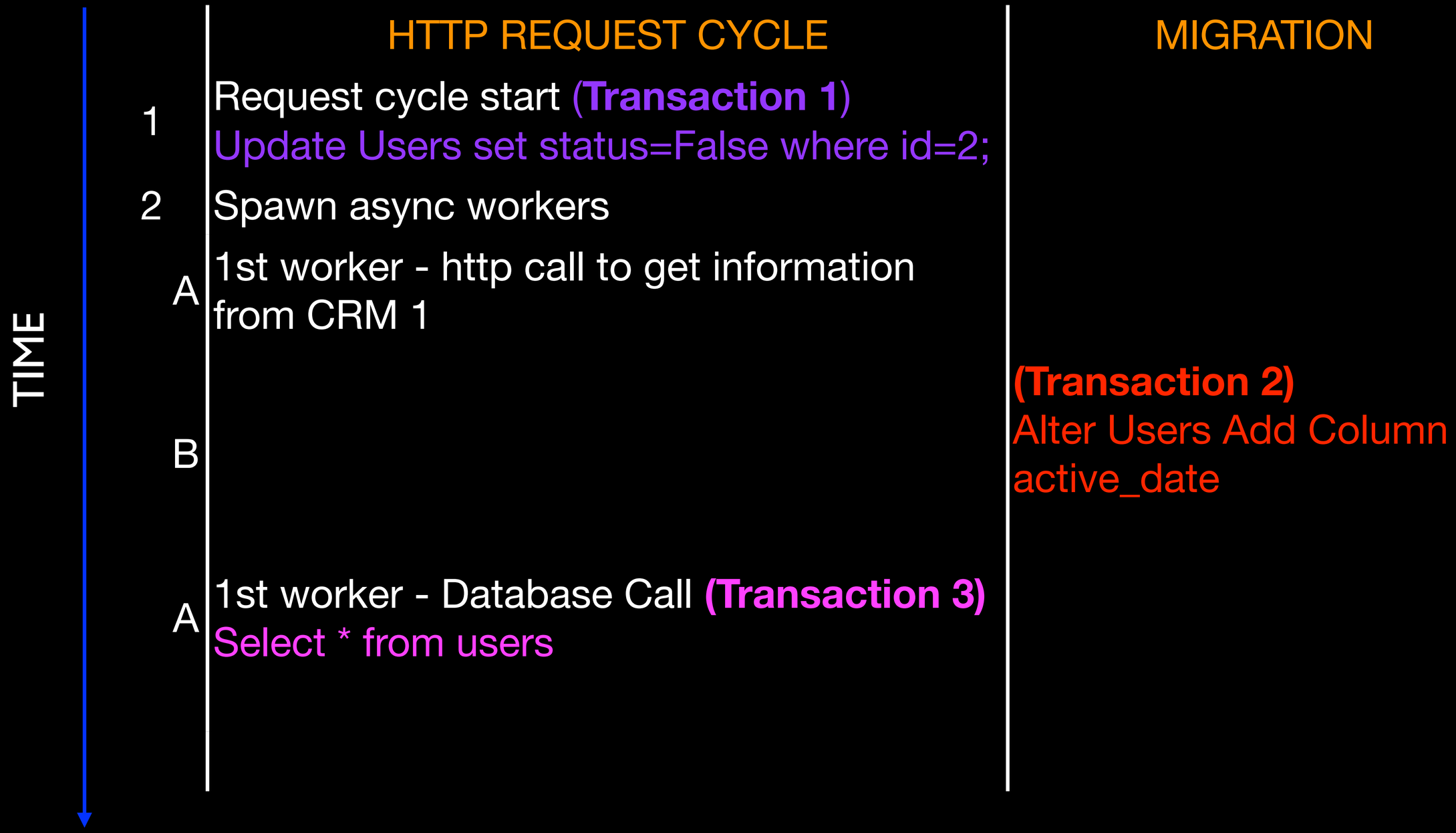
	HTTP REQUEST CYCLE	MIGRATION
1	Request cycle start (**Transaction 1**) Update Users set status=False where id=2;	

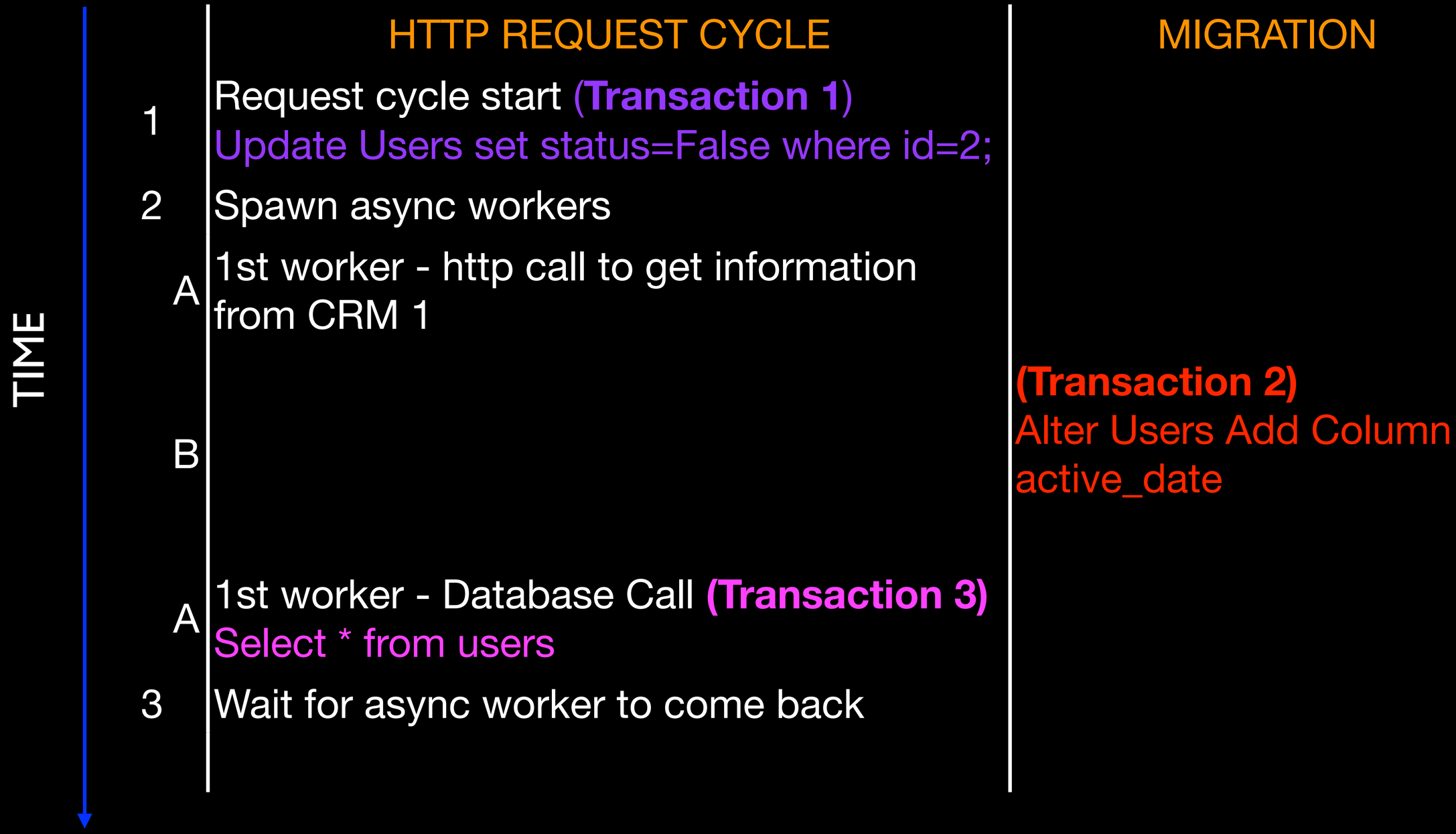
TIME

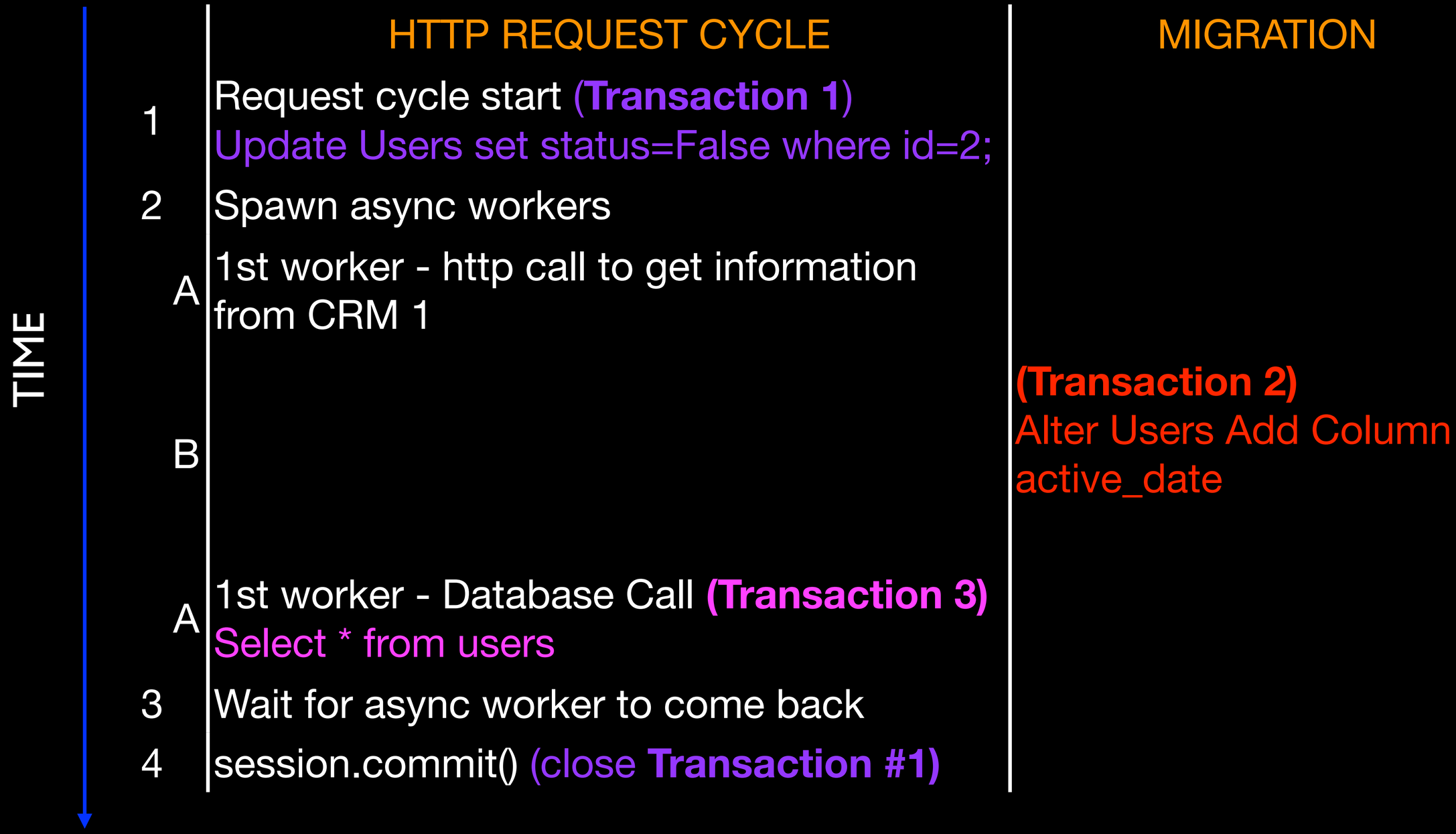
	HTTP REQUEST CYCLE	MIGRATION
1	Request cycle start (**Transaction 1**) Update Users set status=False where id=2;	
2	Spawn async workers	


TIME
HTTP REQUEST CYCLE
MIGRATION
1
Request cycle start (Transaction 1)
Update Users set status=False where id=2;
2
Spawn async workers
A
1st worker - http call to get information
from CRM 1


TIME
HTTP REQUEST CYCLE
MIGRATION
1
Request cycle start (Transaction 1)
Update Users set status=False where id=2;
2
Spawn async workers
A
1st worker - http call to get information from CRM 1
B
(Transaction 2)
Alter Users Add Column active_date


TIME
HTTP REQUEST CYCLE
MIGRATION
1
Request cycle start (Transaction 1)
Update Users set status=False where id=2;
2
Spawn async workers
A
1st worker - http call to get information from CRM 1
B
(Transaction 2)
Alter Users Add Column active_date
A
1st worker - Database Call (Transaction 3)
Select * from users


TIME
HTTP REQUEST CYCLE
MIGRATION
1
Request cycle start (Transaction 1)
Update Users set status=False where id=2;
2
Spawn async workers
A
1st worker - http call to get information from CRM 1
(Transaction 2)
Alter Users Add Column active_date
B
A
1st worker - Database Call (Transaction 3)
Select * from users
3
Wait for async worker to come back


TIME
HTTP REQUEST CYCLE
MIGRATION
1
Request cycle start (Transaction 1)
Update Users set status=False where id=2;
2
Spawn async workers
A
1st worker - http call to get information from CRM 1
B
(Transaction 2)
Alter Users Add Column active_date
A
1st worker - Database Call (Transaction 3)
Select * from users
3
Wait for async worker to come back
4
session.commit() (close Transaction #1)

TRANSACTION 1

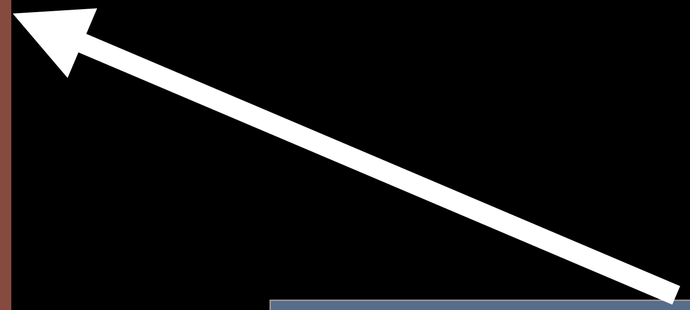
HTTP CALL,

REQUEST CYCLE,

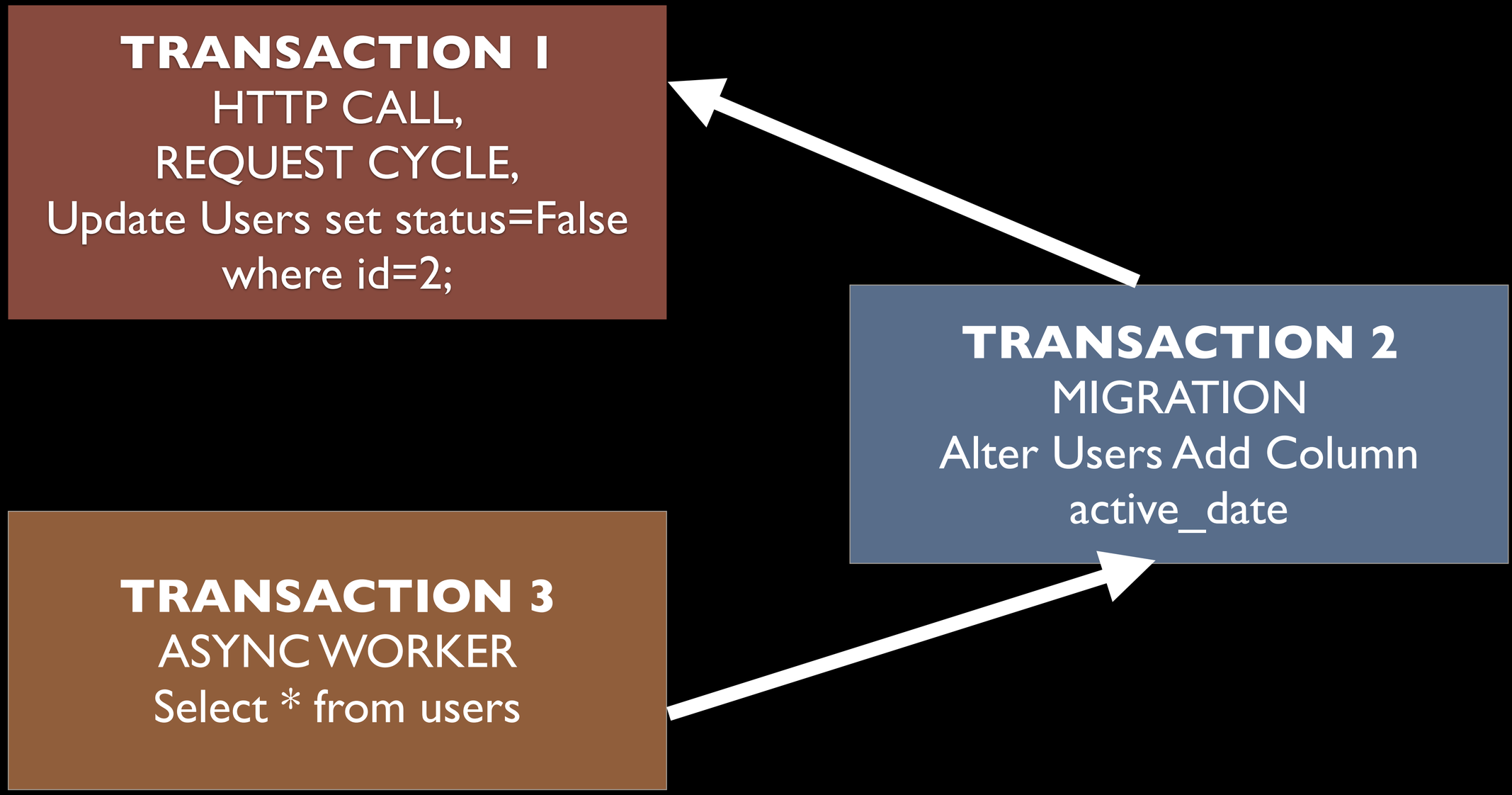
Update Users set status=False

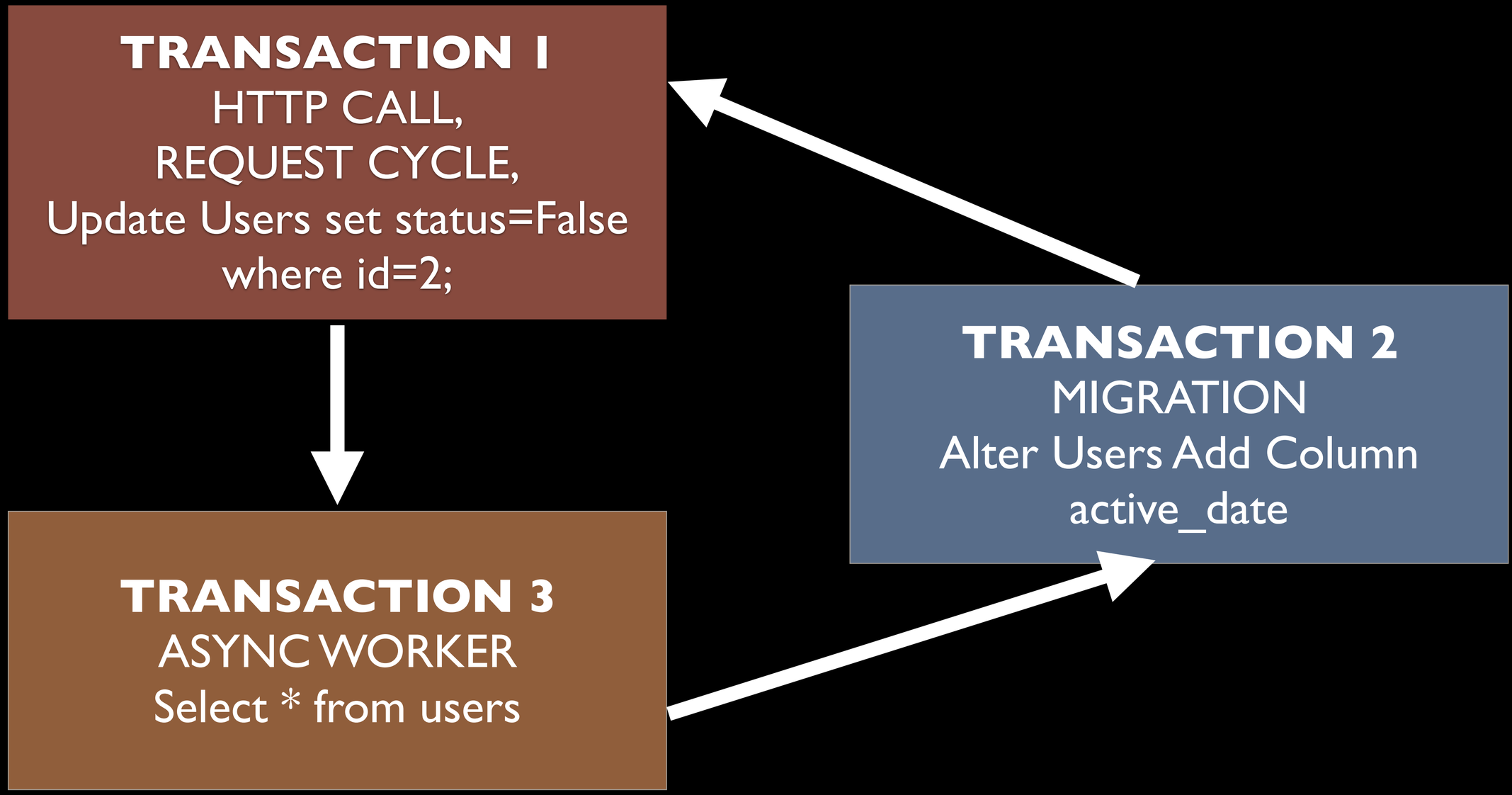
where id=2;

TRANSACTION 1
HTTP CALL,
REQUEST CYCLE,
Update Users set status=False
where id=2;



TRANSACTION 2
MIGRATION
Alter Users Add Column
active_date


TRANSACTION 1
HTTP CALL,
REQUEST CYCLE,
Update Users set status=False
where id=2;
TRANSACTION 2
MIGRATION
Alter Users Add Column
active_date
TRANSACTION 3
ASYNC WORKER
Select * from users


TRANSACTION 1
HTTP CALL,
REQUEST CYCLE,
Update Users set status=False
where id=2;
TRANSACTION 2
MIGRATION
Alter Users Add Column
active_date
TRANSACTION 3
ASYNC WORKER
Select * from users

TIME

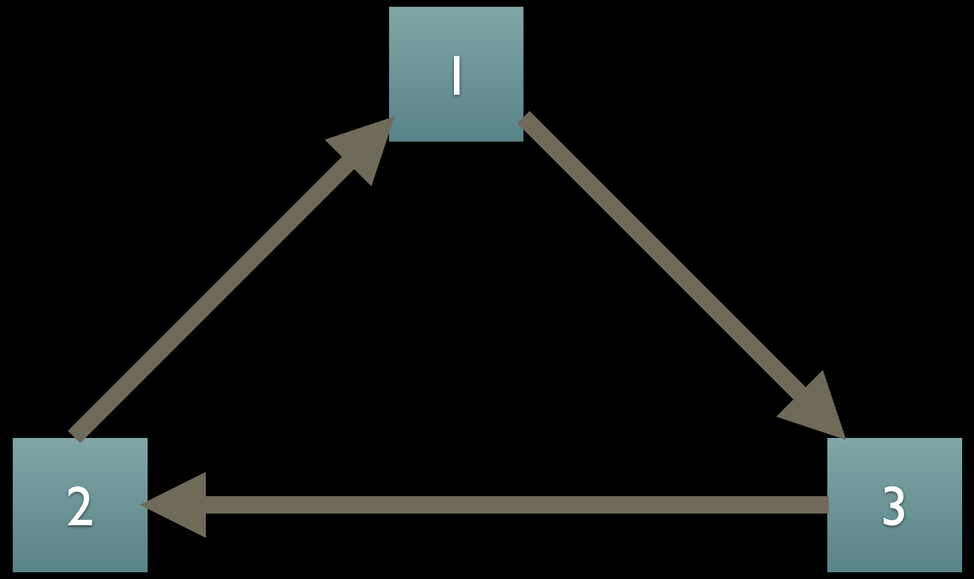
		HTTP REQUEST CYCLE	MIGRATION
1		Request cycle start (**Transaction 1**) Update Users set status=False where id=2;	
2		Spawn async workers	
	A	1st worker - http call to get information from CRM 1	
	B		**(Transaction 2)** Alter Users Add Column active_date
	A	1st worker - Database Call **(Transaction 3)** Select * from users	
3		Wait for async worker to come back	
4		session.commit() (close **Transaction #1)**	

2 blocked by # 1

3 blocked by # 2

1 blocked by # 3

LOAD TEST YOUR MIGRATIONS.



scaling with your team

Migration User

Migration Tooling

Load Testing

Object Relational Mapping(ORM)

offload to your object relation model (ORM)

- Source of Truth

offload to your object relation model (ORM)

- Source of Truth
- Add sensible defaults to your ORM

offload to your object relation model (ORM)

- Source of Truth
- Add sensible defaults to your ORM
- Every model in every language needs to be constantly updated

remaining issues getting to zero downtime

Foreign Keys :(

Vacuuming

Primary and Replica

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 to while this is happening.

DO YOU REALLY NEED REFERENTIAL INTEGRITY?

vacuuming

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

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

primary and replica

As you scale out your primary and replica instances, you want to move all your reads to the replica.

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

scaling with tools

Break your migrations

Don’t lock yourself out

Migration User

Migration Tooling

scaling with your team

remaining issues getting to zero downtime

Foreign Keys :(

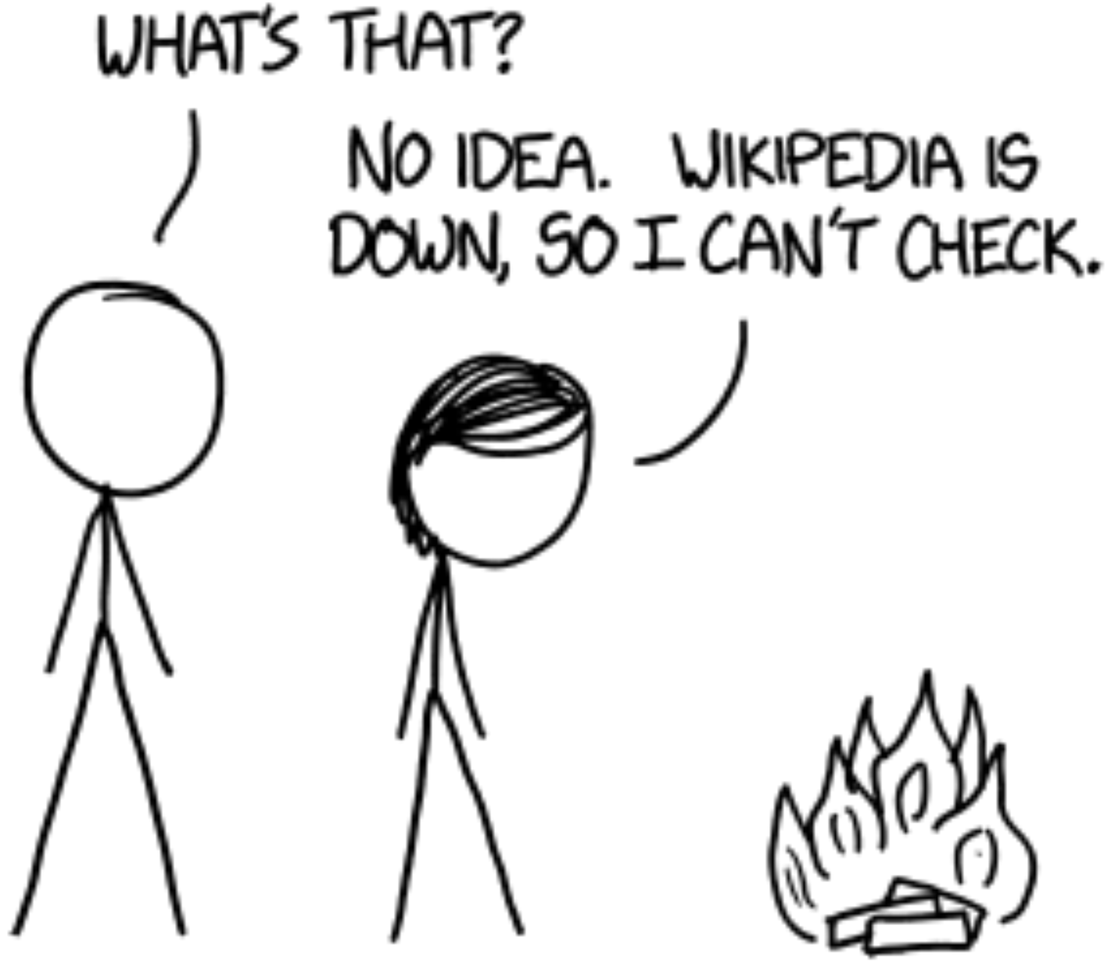
Jumana Bahrainwala
jzbahrai.github.io
@JumzB

Jumana Bahrainwala
jzbahrai.github.io
@JumzB

Can’t sing but will rap the Alexandra Hamilton song with you

acknowledgments

- Gary Brenhardt, Brent Sullivan, Nicolas Neu, Filipe Fernandes, Alex Snurnikov, Zach Howard, Dileshni Jayasinghe and all the other wonderful folk @Unata
- Braintree - Safe operations for High volume postgresql
- 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