pgloader, Your Migration Companion

PostgreSQL Conference Europe, Warsaw

Dimitri Fontaine

Mastering PostgreSQL

October 25, 2017

Dimitri Fontaine

PostgreSQL Major Contributor

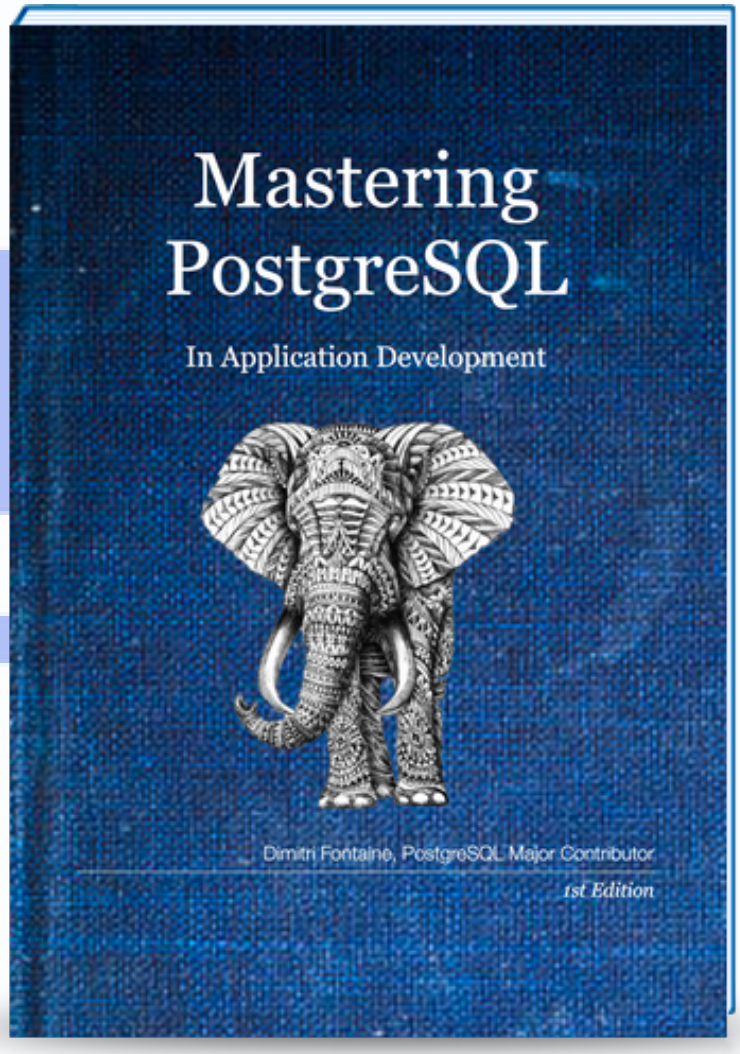
- **pgloader**
- `CREATE EXTENSION`
- `CREATE EVENT TRIGGER`
- *Bi-Directional Réplication*
- *apt.postgresql.org*

Mastering PostgreSQL in Application Development

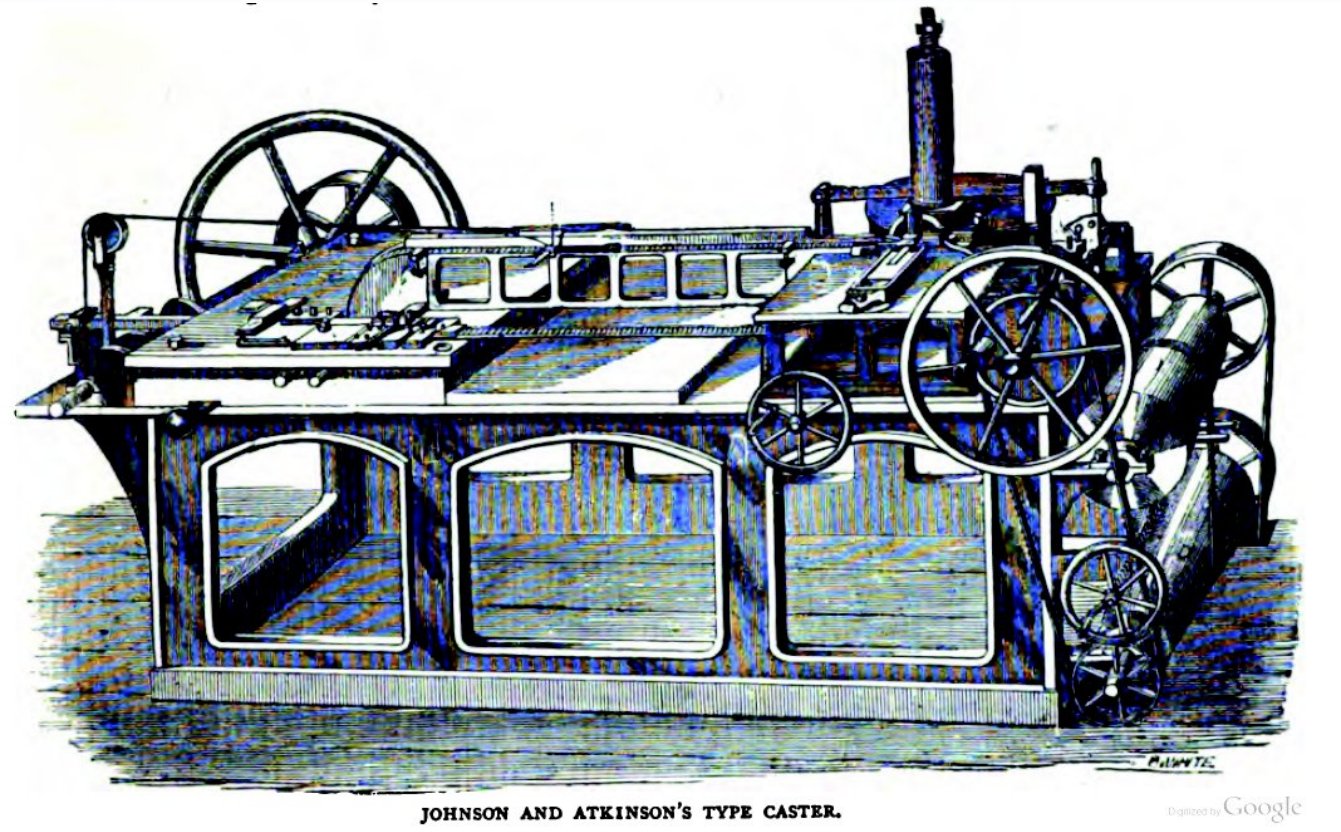
Hey, Im writing a book!

Register on the website to be the first to know when it launches. . . maybe next week!

`http://MasteringPostgreSQL.com`

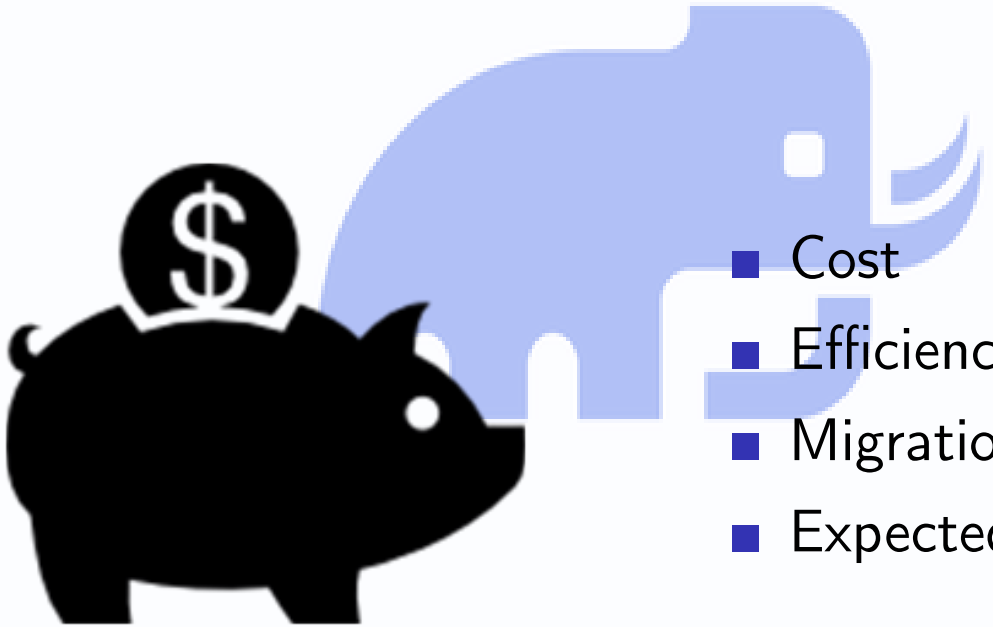


Migrating from another RDBMS to PostgreSQL



Why Migrate Over to PostgreSQL?

The reasons why migrating are usually a mix of technical choice and budget evaluation. Also *human factors*.



- Cost
- Efficiency
- Migration Budget, ROI
- Expected Behavior (ACID)

PostgreSQL is fully ACID

ACID includes resilience to Power Outages, and a safe and clean behavior when used concurrently.

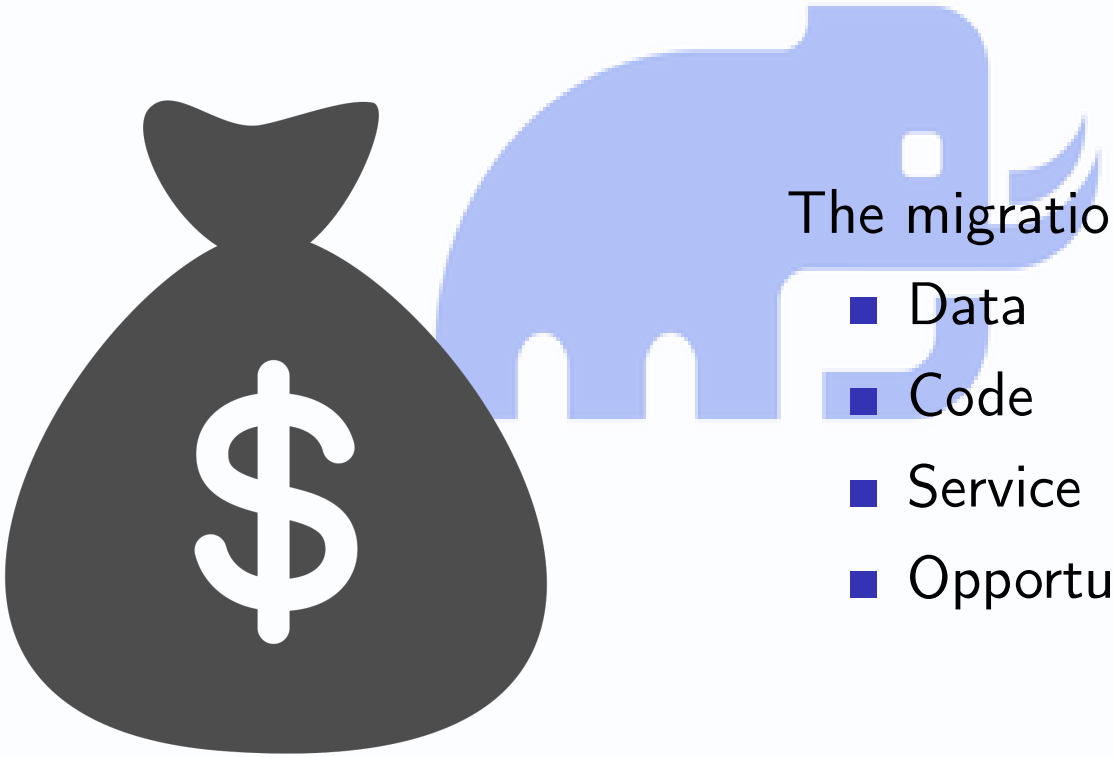
ACID stands for:

- Atomic
- Consistent
- Isolated
- **Durable**

Multithreaded programming

The Migration Project

The migration usually isn't done overnight. It requires proper resource allocation and planning. And a proper budget, which helps determining the return on investment, too.



The migration budget split:

- Data
- Code
- Service
- Opportunity Cost

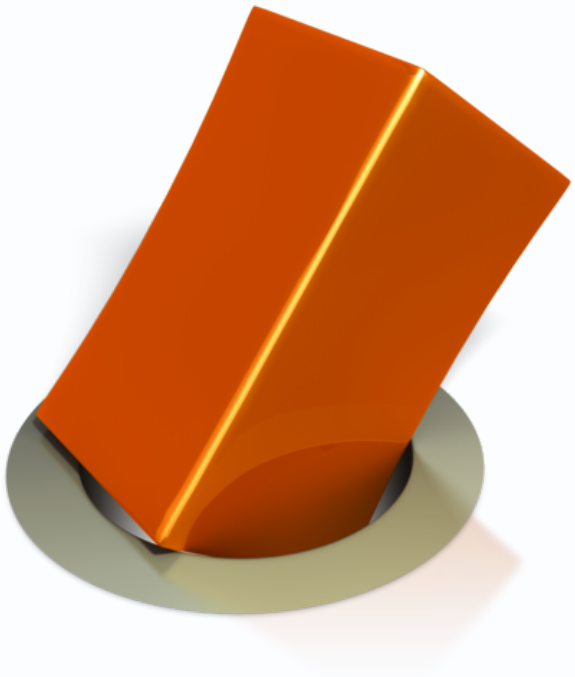
Migration Project Planning

"A goal without a plan is just a wish"

Antoine de Saint-Exupéry

Migration Planning

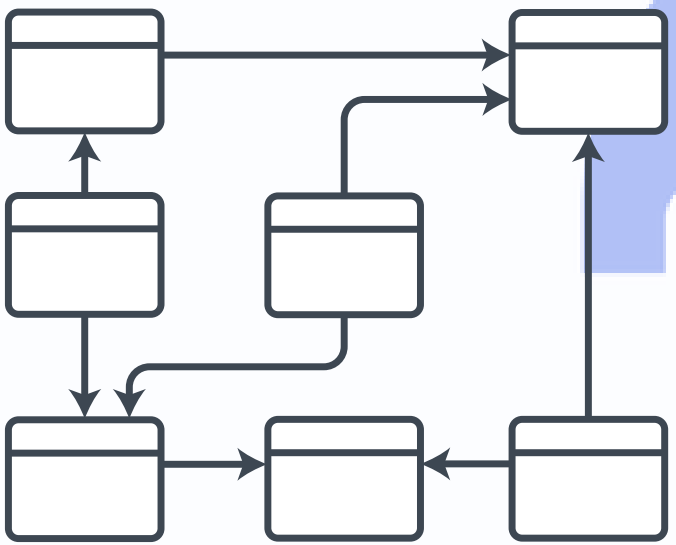
Migrating the data is often considered a one-off. Then it's not properly planned, and happens on the side. Fearing to spend too much time on this, proper engineering might not be applied.



1. Setup a PostgreSQL instance
2. Keep the default configuration
3. Migrate the data over to PostgreSQL
4. Manually
5. In many steps, *error*, *fix*, *rinse*, *repeat*
6. Including `$EDITOR`

Porting the Application

Now that we have a PostgreSQL database with the right dataset in there. . . wait, using what schema?



- Same schema, different RDBMS
- Sometimes manually converted
- Sometimes installed by the ORM
- Using only the basics

Porting the Application

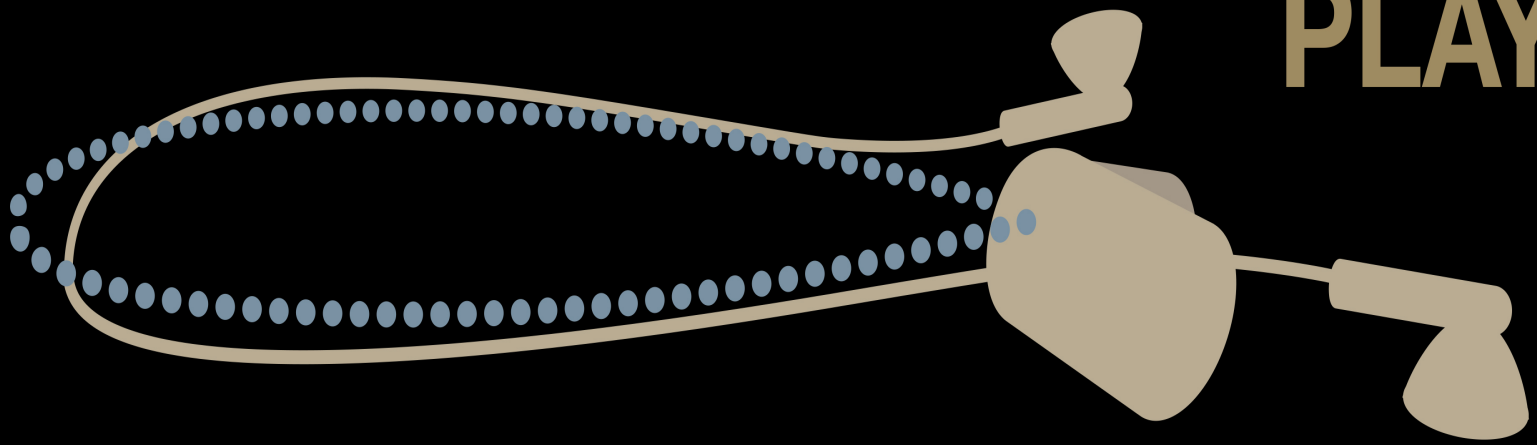
Now that we have a PostgreSQL database with the right dataset in there, the code is adjusted until it works as before, only this time using PostgreSQL.

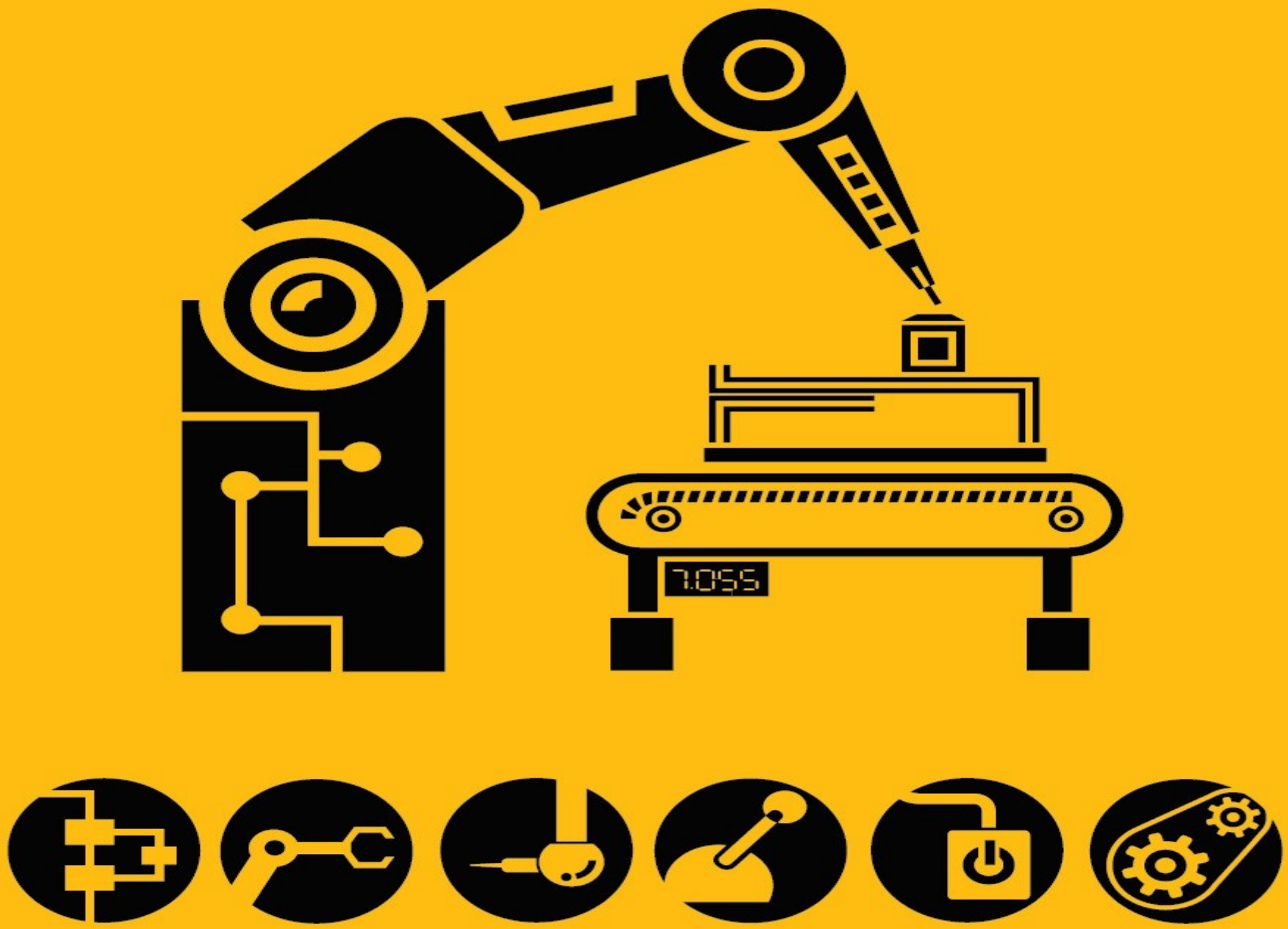
Some of the usual traps:

- SQL quoting rules
- Database Encoding
- Client Encoding
- SQL syntax

DEPLOYMENT

PLAYLIST




7.055

Continuous Integration, Continuous Delivery

Continuous Integration, Continuous Delivery

Instanciate a PostgreSQL version of your application in your *CI/CD* setup, from day one, even before doing anything else. Then automate all the steps from current production to PostgreSQL based production.

KEEP
CALM
AND
AUTOMATE ALL
THE THINGS

From Day One:

- Nightly database migration
- All automated, from production data
- Add a PostgreSQL coverage dashboard

How Do I Do That?

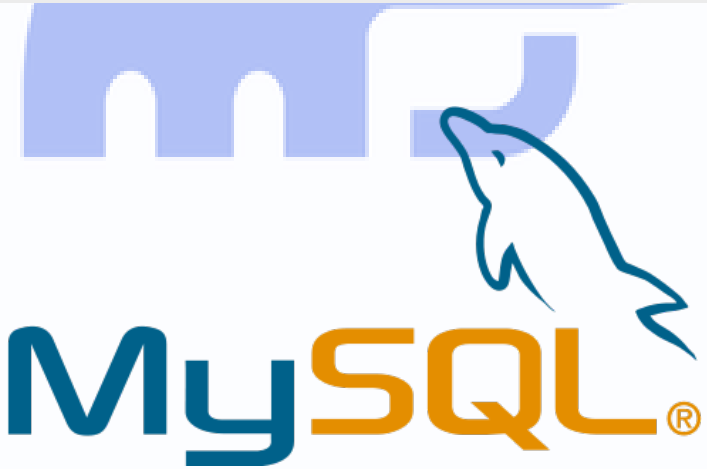
pgLoader loads data into PostgreSQL

http://pgloader.io/

pgLoader connects to MySQL

http://pgloader.io/howto/mysql.html

```
1  $ createdb -U user dbname
2  $ pgloader mysql://user@host/dbname \
3             pgsql://user@host/dbname
```



pgLoader automates all the things

When using pgLoader with a *load* command, it's possible to give more options:

```
$ pgloader f1db.load
```

```
load database
  from mysql://root@localhost/f1db
  into pgsql:///f1db
  with concurrency = 2,
       multiple readers per thread,
       rows per range = 50000
       prefetch rows = 1000000;
```

table name	errors	rows	bytes	total time
fetch meta data	0	33		0.325s
Create Schemas	0	0		0.001s
Create SQL Types	0	0		0.008s
Create tables	0	26		0.202s
Set Table OIDs	0	13		0.008s
f1db.circuits	0	73	8.5 kB	0.039s
f1db.constructorresults	0	11052	184.6 kB	0.252s
f1db.constructors	0	208	15.0 kB	0.054s
f1db.drivers	0	840	79.6 kB	0.094s
f1db.laptimes	0	417743	10.9 MB	6.320s
...				
f1db.results	0	23597	1.3 MB	0.987s
f1db.status	0	134	1.7 kB	0.068s
COPY Threads Completion	0	4		6.468s
Create Indexes	0	20		2.347s
Index Build Completion	0	20		1.458s
Reset Sequences	0	10		0.127s
Primary Keys	0	13		0.021s
Create Foreign Keys	0	0		0.000s
Create Triggers	0	0		0.001s
Install Comments	0	0		0.000s
Total import time	✓	511270	14.0 MB	10.422s

pgLoader example output, 1/3

The migration preparation steps: fetch source metadata, apply casting rules, transform *default values*, prepare target schema:

table name	rows	bytes	total time
fetch meta data	33		0.325s
Create Schemas	0		0.001s
Create SQL Types	0		0.008s
Create tables	26		0.202s
Set Table OIDs	13		0.008s

pgLoader example output, 2/3

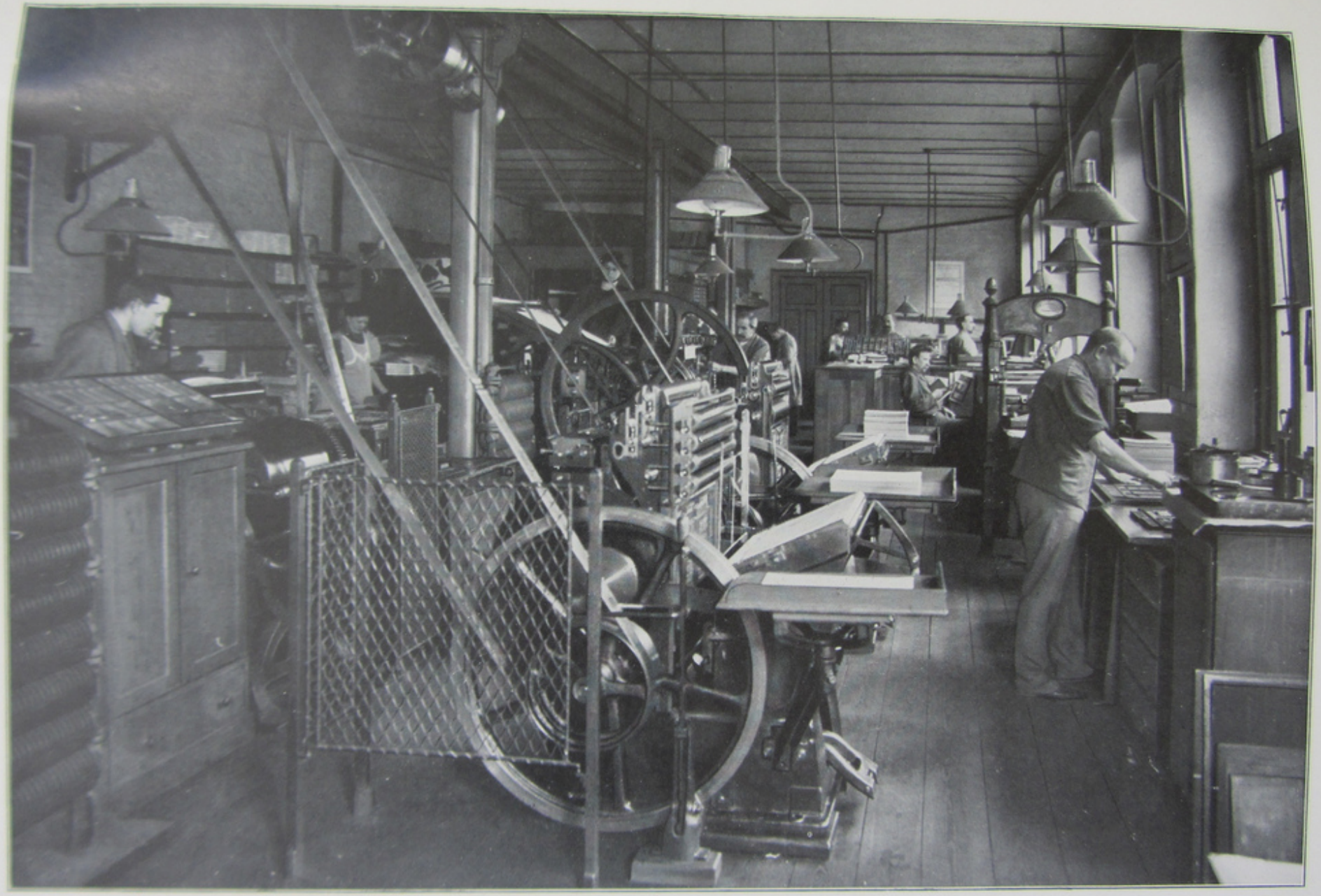
Moving the data over, transforming the data on the fly, and keeping batches of rows around in case of copy error(s):

table name	rows	bytes	total time
f1db.circuits	73	8.5 kB	0.039s
f1db.constructorresults	11052	184.6 kB	0.252s
f1db.constructors	208	15.0 kB	0.054s
f1db.drivers	840	79.6 kB	0.094s
f1db.laptimes	417743	10.9 MB	6.320s
f1db.constructorstandings	11806	247.3 kB	0.312s
f1db.driverstandings	31509	714.0 kB	0.971s
f1db.pitstops	5927	198.7 kB	0.437s
f1db.races	976	98.4 kB	0.310s
f1db.seasons	68	3.9 kB	0.395s
f1db.qualifying	7337	279.0 kB	0.139s
f1db.results	23597	1.3 MB	0.987s
f1db.status	134	1.7 kB	0.068s

pgLoader example output, 3/3

Now that the data has been migrated over, complete the PostgreSQL schema with *Primary Keys*, *Foreign Keys*, *Sequences*, etc:

table name	rows	bytes	total time
COPY Threads Completion	4		6.468s
Create Indexes	20		2.347s
Index Build Completion	20		1.458s
Reset Sequences	10		0.127s
Primary Keys	13		0.021s
Create Foreign Keys	0		0.000s
Create Triggers	0		0.001s
Install Comments	0		0.000s
Total import time	511270	14.0 MB	10.422s



PHOT. FRANZ KOMPEL - HAMBURG

Haus-Buchdruckerei

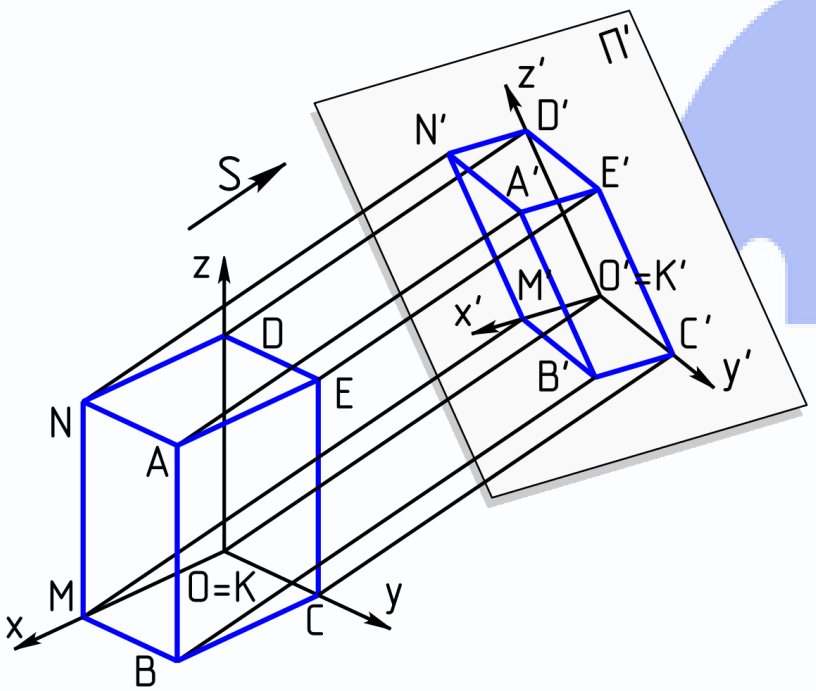
pgLoader Casting Rules

In order to be fully automated, pgloader allows its users to redefine any default casting rule.

```
load database
     from mysql://root@unix:/tmp/mysql.sock:3306/pgloader
     into postgresql://dim@localhost/pgloader

 alter schema 'pgloader' rename to 'mysql'

 CAST column base64.id to uuid drop typemod drop not null,
      column base64.data to jsonb using base64-decode,

      type decimal when (and (= 18 precision) (= 6 scale))
        to "double precision" drop typemod

 before load do $$ create schema if not exists mysql; $$;
```

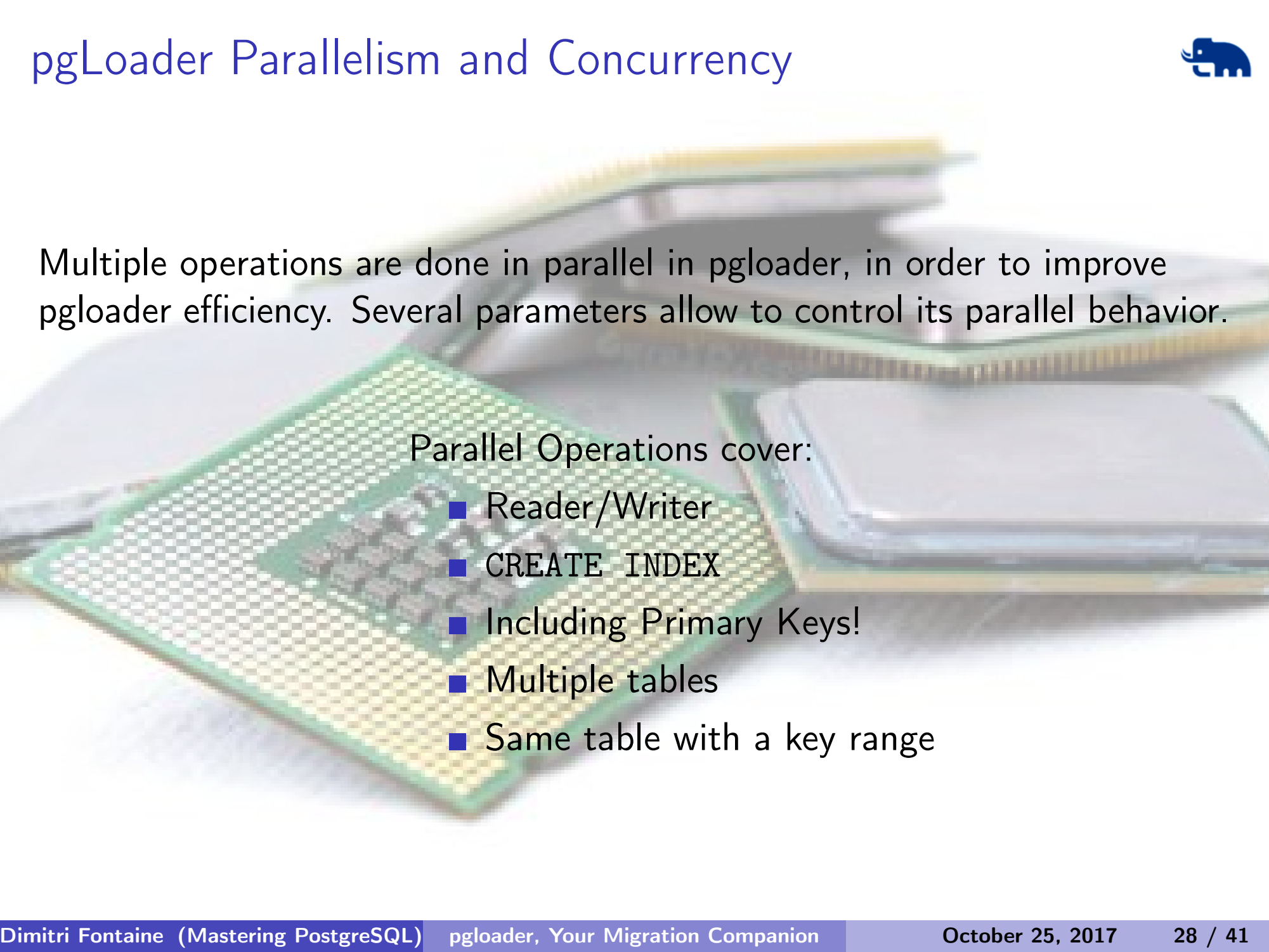
pgLoader Catalog Mapping

pgloader maintains an internal representation of both the *source* and *target* catalogs, allowing to apply some internal commands in order to implement the mapping:




```
1  ALTER SCHEMA '...'
2        RENAME TO '...'
3
4  ALTER TABLE NAMES MATCHING ...
5        IN SCHEMA '...'
6
7  ALTER TABLE NAMES MATCHING ...
8        RENAME TO '...'
```

pgLoader Parallelism and Concurrency



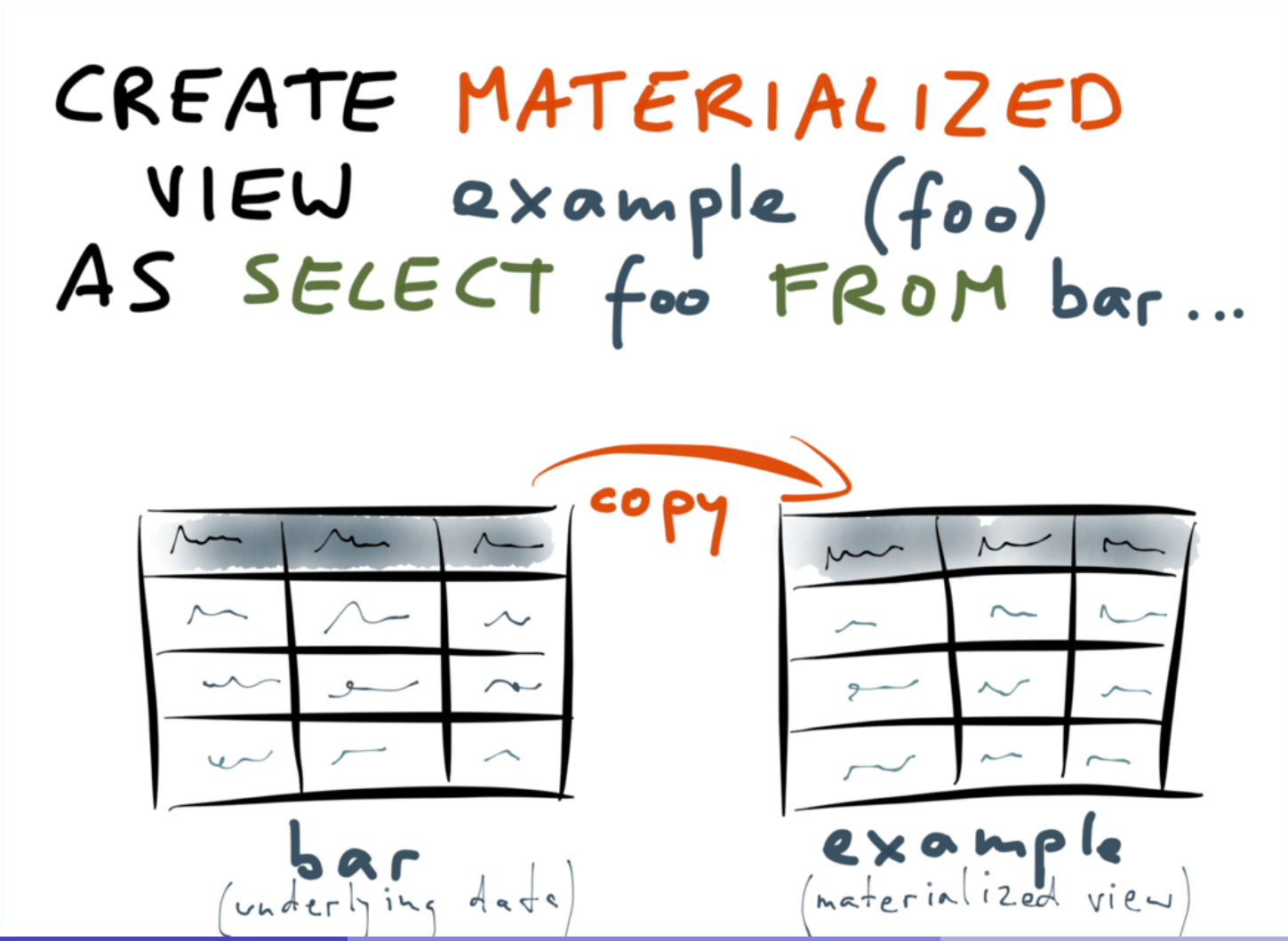
Multiple operations are done in parallel in pgloader, in order to improve pgloader efficiency. Several parameters allow to control its parallel behavior.

Parallel Operations cover:

- Reader/Writer
- `CREATE INDEX`
- Including Primary Keys!
- Multiple tables
- Same table with a key range

pgLoader Materialize Views

Rather than copying plain table from MySQL to PostgreSQL, it is possible to copy the result of `SELECT * FROM view;` instead.



Other pgLoader Options

Taken from several test files, a *hodgepodge*:

```
load database
     from '{{DBPATH}}'
     into postgresql:///pgloader

 WITH on error stop, concurrency = 2, workers = 6,
      prefetch rows = 25000, rows per range = 50000,
      multiple readers per thread,
      max parallel create index = 4,
      quote identifiers

  SET PostgreSQL PARAMETERS
      maintenance_work_mem to '128MB', work_mem to '12MB',
      search_path to 'sakila, public, "$user"'

  SET MySQL PARAMETERS
      net_read_timeout  = '120', net_write_timeout = '120'
```

Other pgLoader Options

Taken from the `test/sakila.load` file:

```
BEFORE LOAD
  DO
    $$ create extension if not exists ip4r; $$,
    $$ create schema if not exists geolite; $$

  EXECUTE 'geolite.sql'

-- WITH create no tables, include drop, truncate,

--      include drop, create tables, no truncate,
--      create indexes, reset sequences, foreign keys

INCLUDING ONLY TABLE NAMES MATCHING ~/film/, 'actor'
EXCLUDING TABLE NAMES MATCHING ~<ory>
```

pgLoader Data Sources

Some User Testimonials

Feedback from Twitter

Tommaso Patrizi

@tommasomatic

@tapoueh #pgloader is so good I'm almost crying! Say goodbye to all mysql implementations! Thanks!

https://twitter.com/tommasomatic/status/884181490724155392

Feedback from Twitter

Majid Hajian

@mhadaily

Successfully migrated from #MySQL 5 to #PostgreSQL 9.5, roughly 16GB data, thanks to tapoueh for a fantastic #pgloader tool

https://twitter.com/mhadaily/status/806763214092414976

Feedback from Twitter

whitequark

@whitequark

---

so I need to migrate ∼17 million rows from MySQL to Postgres. am I in for pain or for a lot of pain?

---

pgloader appears to be able to do it in ∼1 hour, with 512MB of RAM (probably faster with more RAM)

---

https://twitter.com/whitequark/status/768208585503354881

PARTICIPATION

pgloader: Open Source, github

https://github.com/dimitri/pgloader

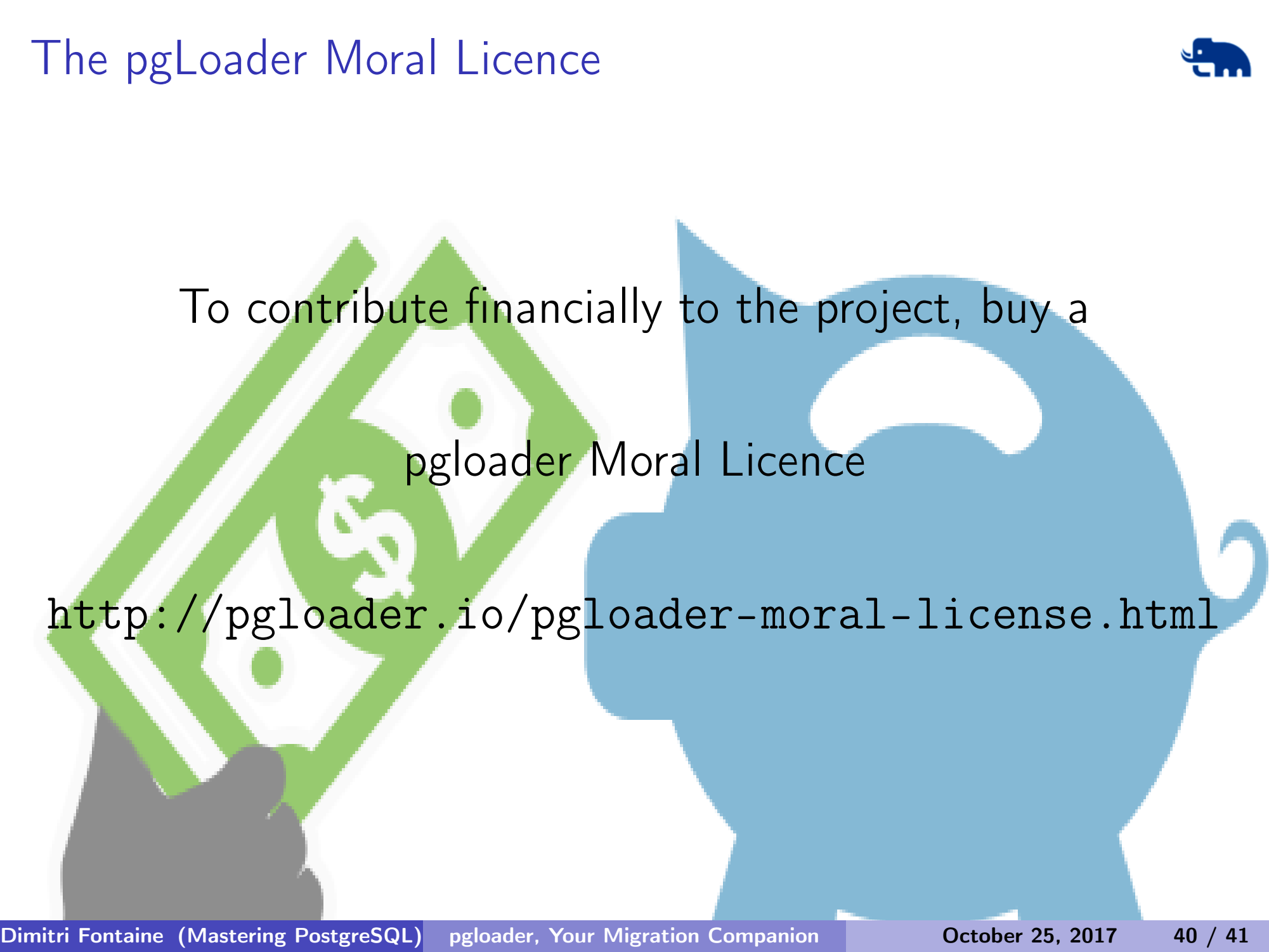
Contributing to pgLoader

Maintaining pgloader is fun: you get to help automate advanced PostgreSQL migrations, from diverse environments. Common Lisp is easy enough to learn, of course new APIs and ecosystems are not always easy to grasp.

Ideas of areas where to contribute:

- Windows™ Automatic Builds
- Keep Build Artefacts from Travis
- More tests coverage
- Documentation: tutorials
- From issues to the wiki
- Other Feature Requests

The pgLoader Moral Licence



To contribute financially to the project, buy a

pgloader Moral Licence

`http://pgloader.io/pgloader-moral-license.html`

Questions?