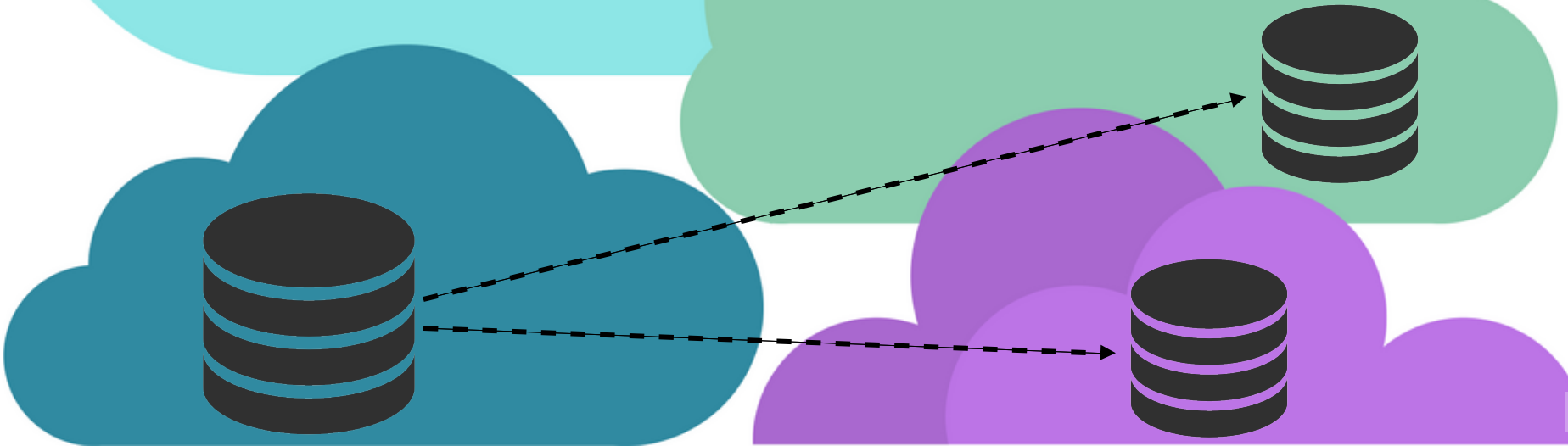
Gülçin Yıldırım

2ndQuadrant®
PostgreSQL

PGDAY'17 RUSSIA

КОНФЕРЕНЦИЯ ПО БАЗАМ ДАННЫХ

Automated Upgrades of PostgreSQL Clusters in Cloud



select * from me;

Site Reliability Engineer @ 2ndQuadrant

Board Member @ PostgreSQL Europe

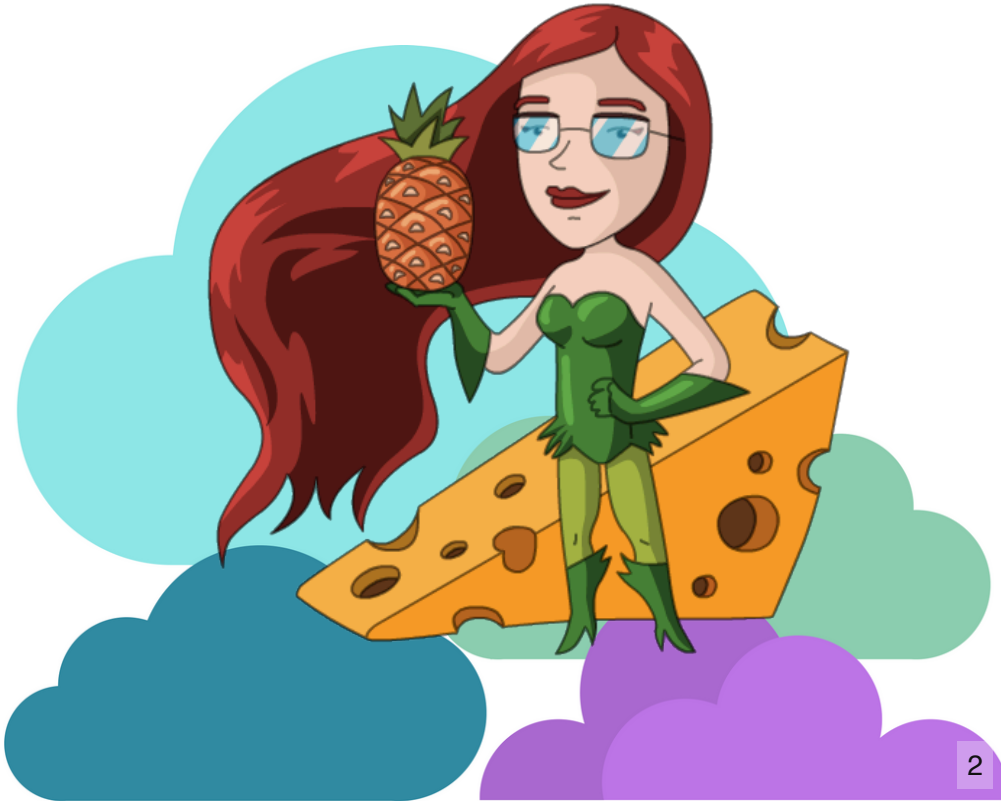
MSc Comp. & Systems Eng. @ Tallinn University of Technology

Writes on 2ndQuadrant blog

Does some childish paintings

Loves independent films

From Turkey



@ apatheticmagpie

Skype: gulcin2ndq

Github: gulcin

Agenda

- Database Upgrades
- Why Logical Replication?
- Platform Implementation
- Case Studies & Results
- Applicability & Limitations
- Conclusion

Problems?

DOWNTIME

SLAs?

Low Capacity?

Revenue Loss

Reputation Loss

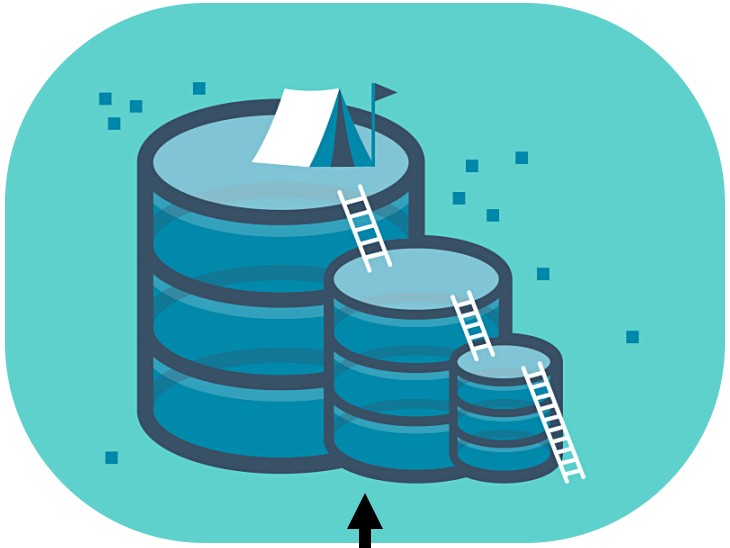
High Availability?

Database Anyone?

Banks

Social Networks

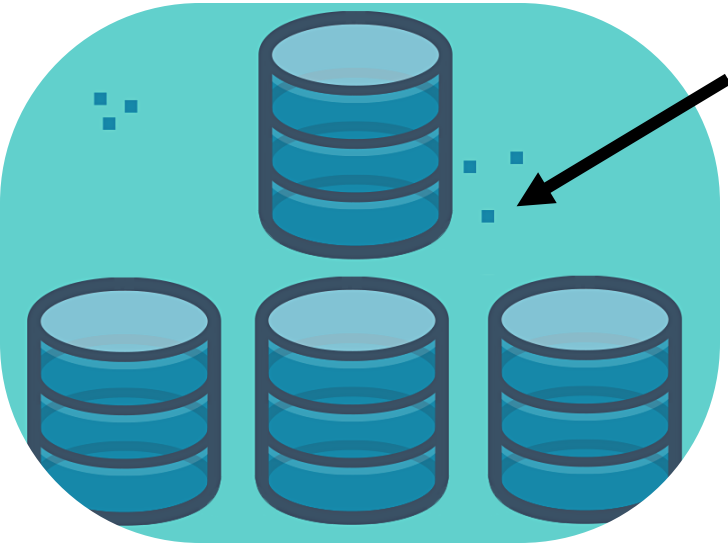
Desktop Apps

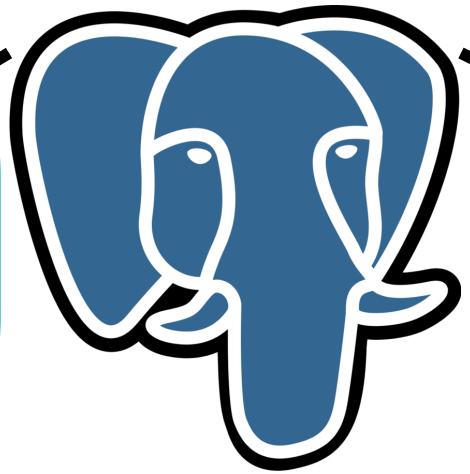


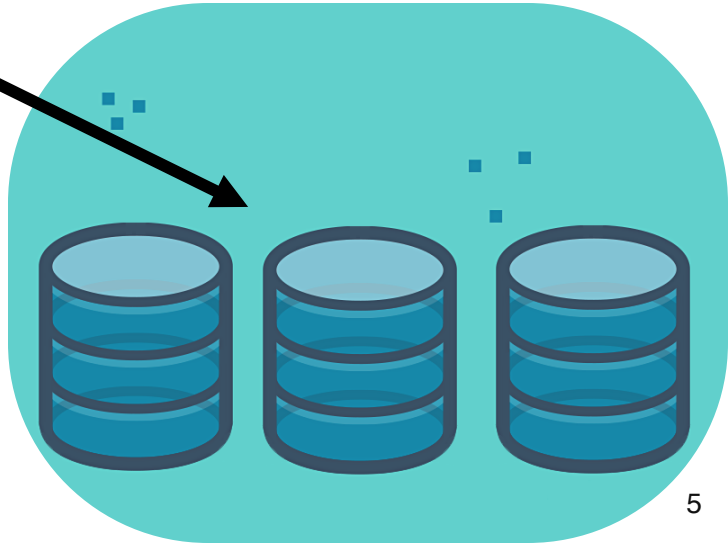
Startups

Medium-size

Enterprises





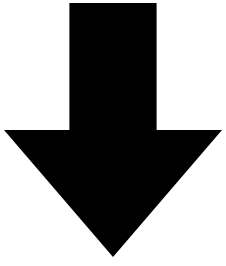


Upgrade,or not to Upgrade

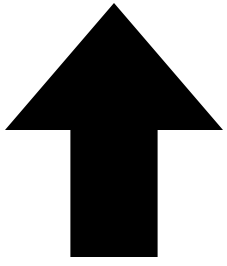
- New features
- Security patches
- Perfomance Updates
- Bug fixes

- Outdated, no support
- Vulnerable to attacks
- Poor Perfomance
- Buggy, hard to maintain

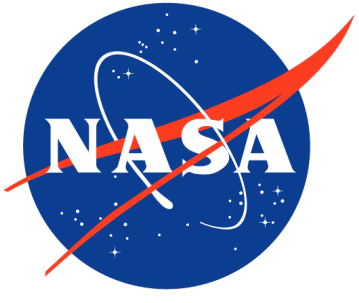
Why Automate?



- Risk & Errors
- Cost
- Time-to-market



- Reproducibility
- Repeatability
- Efficiency



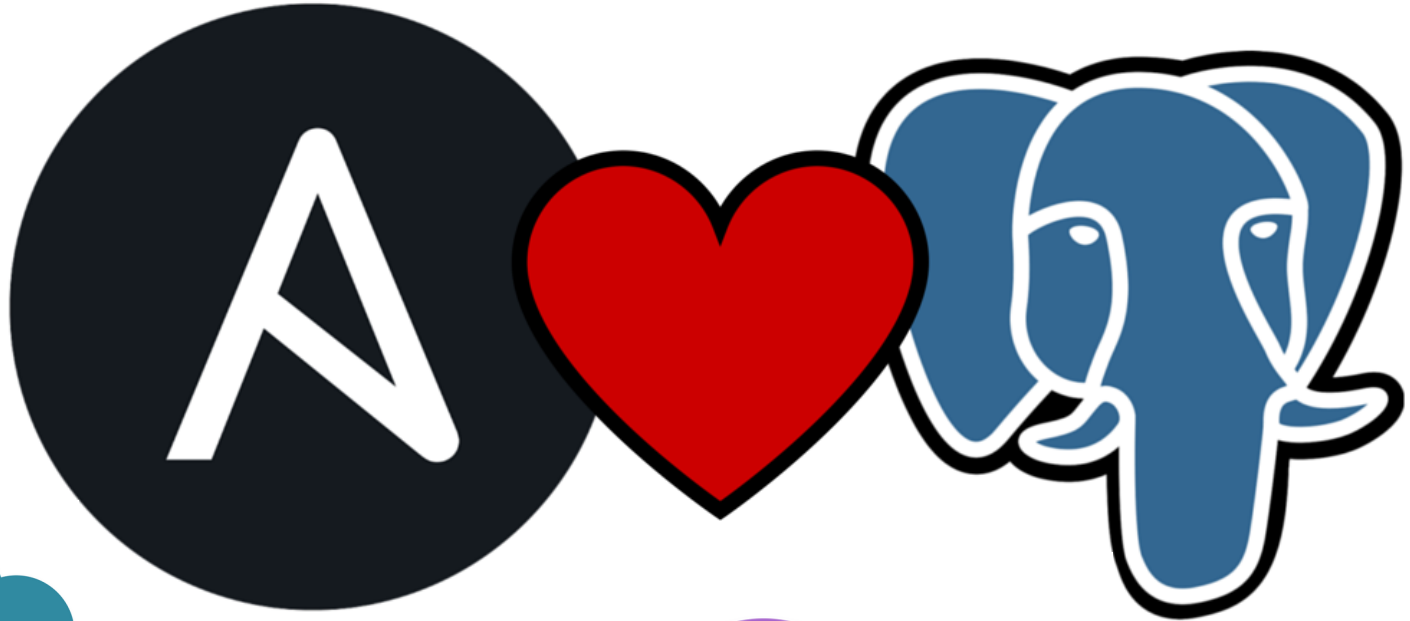
- *Updating nasa.gov:* 1 hr to 5 mins
- *Patching updates:* Multi-day to 45 mins
- *Application stack setup:* 1-2 hrs to 10 mins

Ansible Loves PostgreSQL

(in the s)

AWS Modules: 100

Postgres Modules: 6



50
Upgrade

116
Cloud

252
PostgreSQL

208
Database

Database Upgrades

PGDAY'17 RUSSIA

1

- same or compatible storage format
- hard to guarantee
- performance optimization - data structures

2

- logical copy (dump)
- load into new server
- traditional approach
- offline (downtime)

3

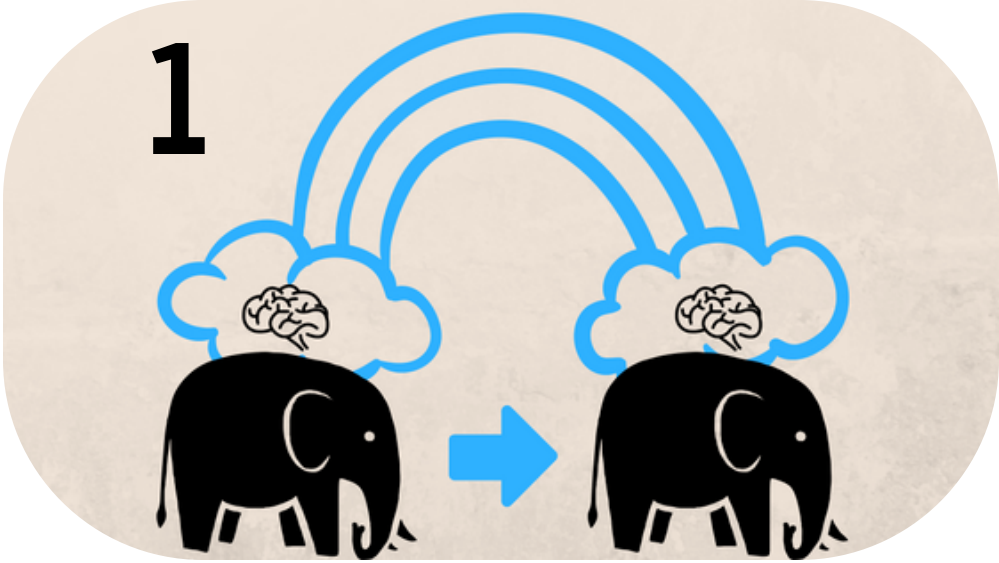
- convert data from old format to new
- can be online (perf?)
- offline (downtime)
- often shorter (2nd)

4

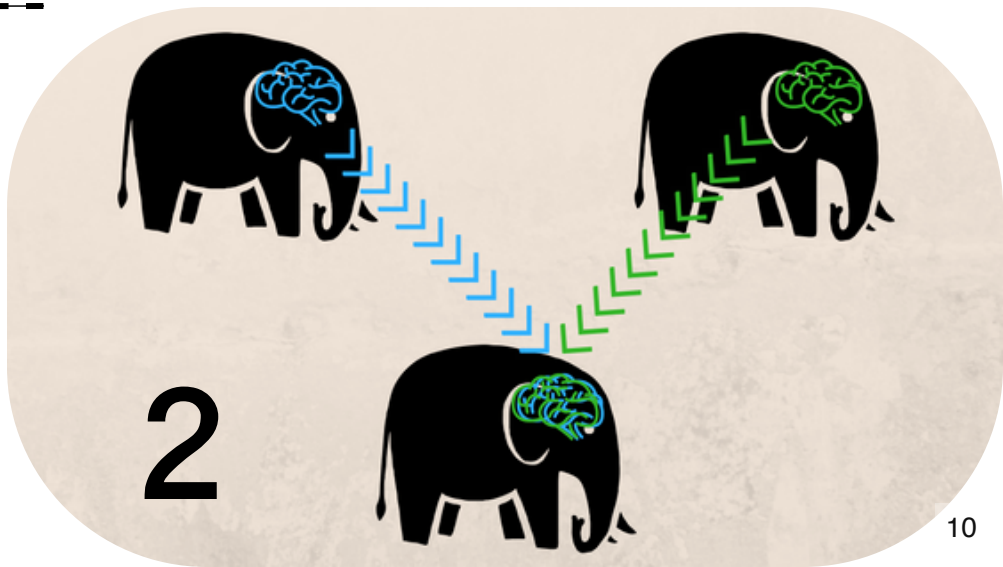
- logical dump (restore)
- capture changes while upgrade
- replicate after restore
- **min downtime**

Logical Replication Rocks!

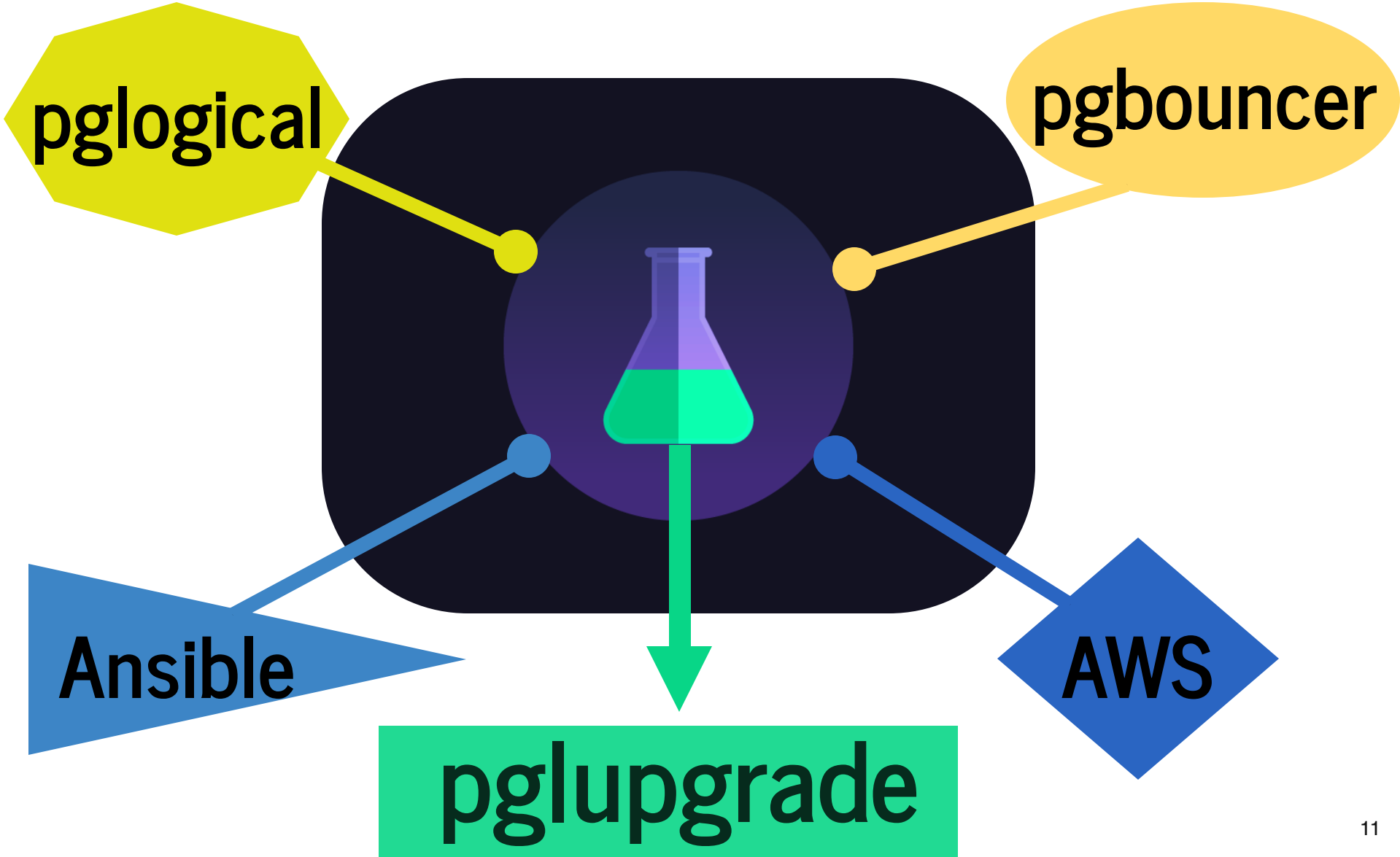
- Offline Conversion
  - pg_dump/pg_restore
  - pg_upgrade



- Online Conversion
  - pglogical
  - pglupgrade



Elements of the solution

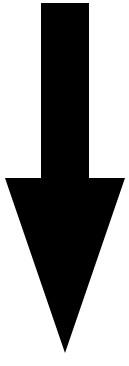


Pglupgrade playbook

8 plays

config.yml

host.ini



orchestrates the upgrade operation

```
[old-primary]
54.171.211.188

[new-primary]
54.246.183.100

[old-standbys]
54.77.249.81
54.154.49.180

[new-standbys:children]
old-standbys

[pgbouncer]
54.154.49.180
```

Inventory file host.ini

```
$ ansible-playbook -i hosts.ini pglupgrade.yml
```

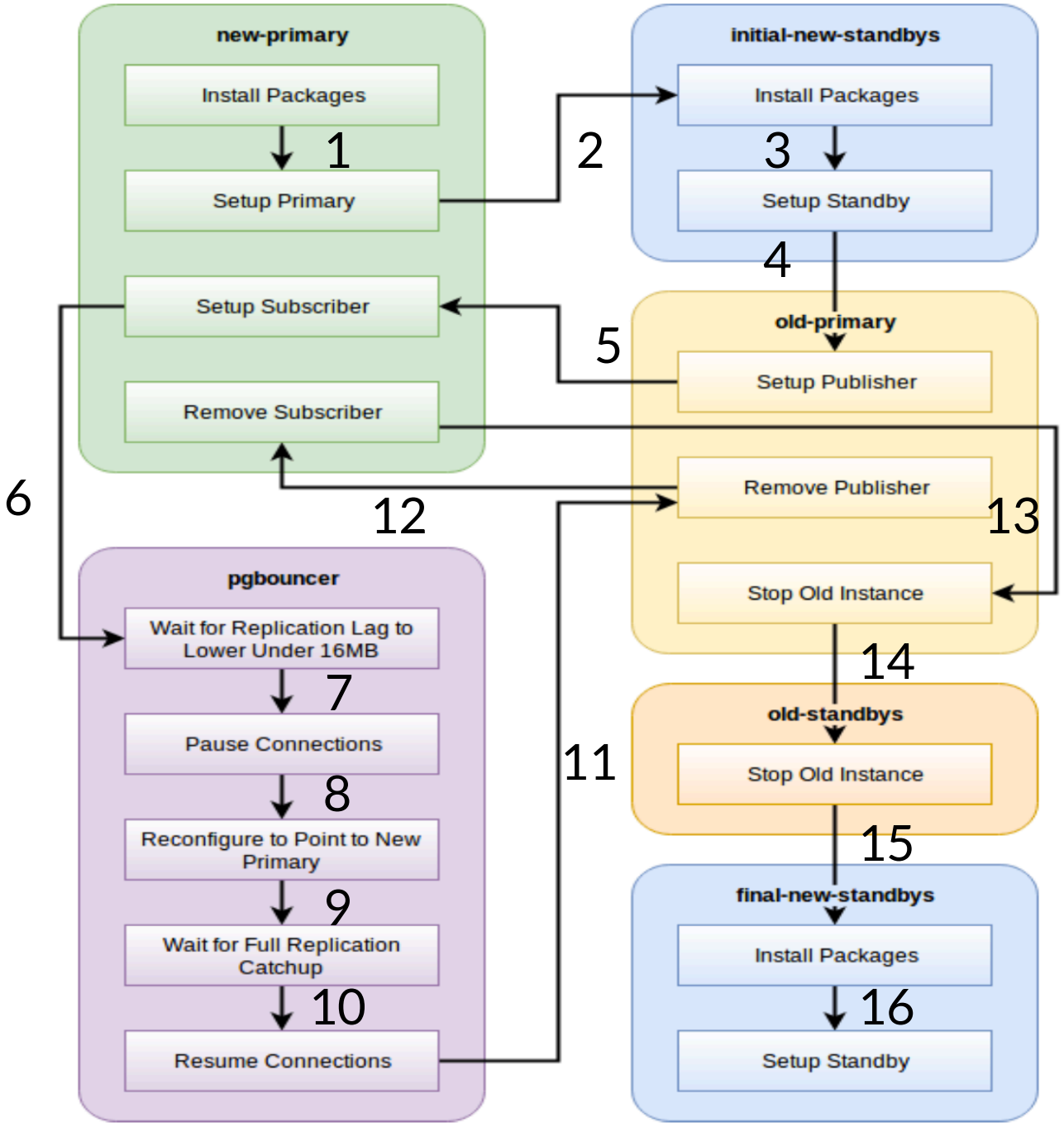
Running pglupgrade playbook

Pglupgrade playbook

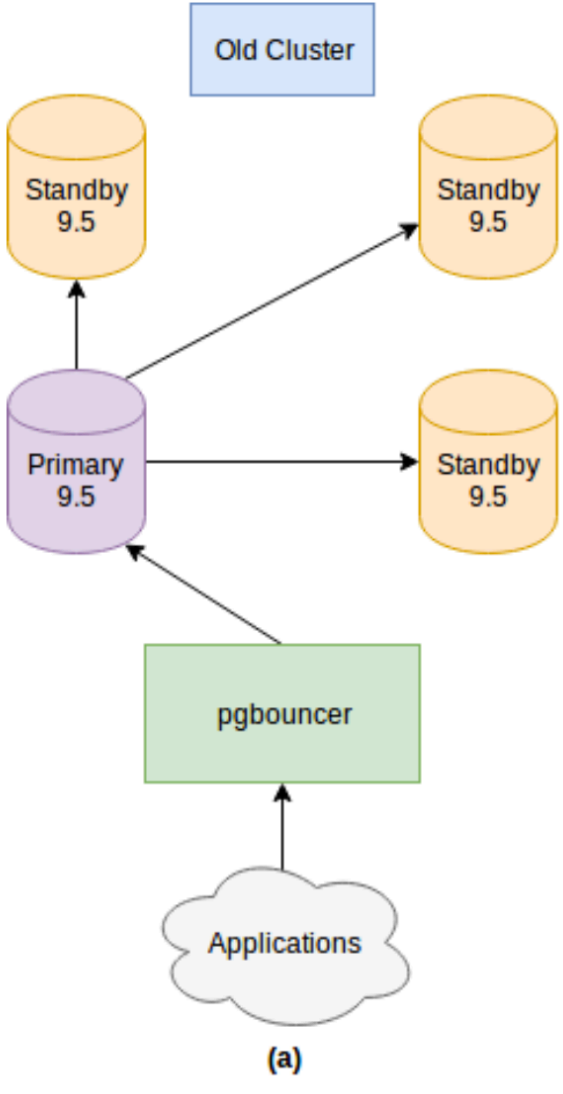
```
ansible_user: admin

pglupgrade_user: pglupgrade
pglupgrade_pass: pglupgrade123
pglupgrade_database: postgres

replica_user: postgres
replica_pass: ""

pgbouncer_user: pgbouncer

postgres_old_version: 9.5
postgres_new_version: 9.6

subscription_name: upgrade
replication_set: upgrade

initial_standbys: 1

postgres_old_dsn: "dbname={{pglupgrade_database}} host={{groups['old-primary'][0]}} user={{pglupgrade_user}}"
postgres_new_dsn: "dbname={{pglupgrade_database}} host={{groups['new-primary'][0]}} user={{pglupgrade_user}}"

postgres_old_datadir: "/var/lib/postgresql/{{postgres_old_version}}/main"
postgres_new_datadir: "/var/lib/postgresql/{{postgres_new_version}}/main"

postgres_new_confdir: "/etc/postgresql/{{postgres_new_version}}/main"
```

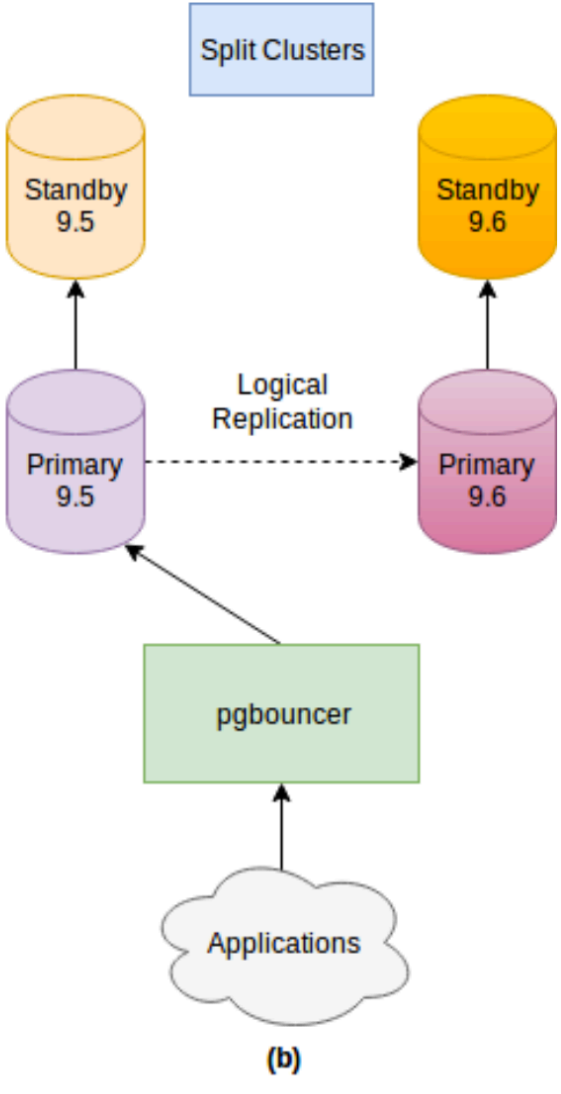
config.yml

How Does It Work?

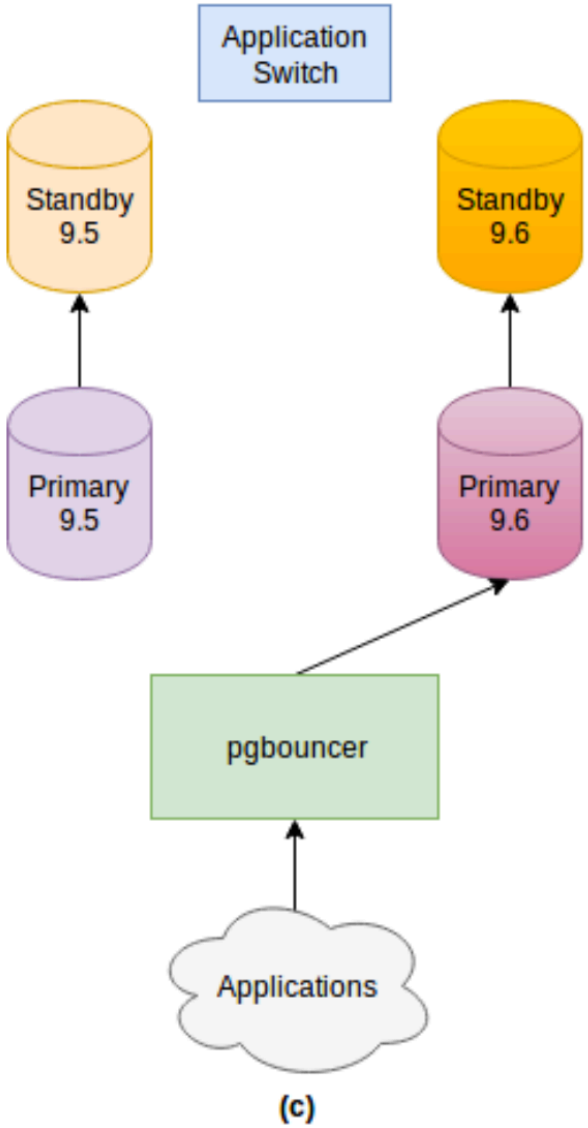


1st Case: High Availability

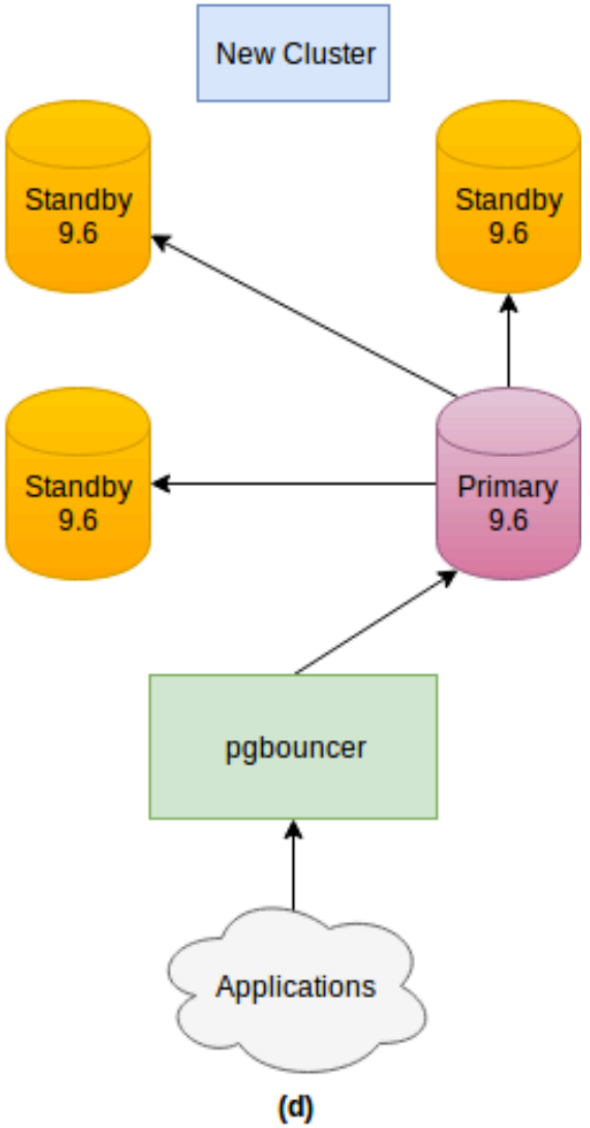




1st Case: High Availability

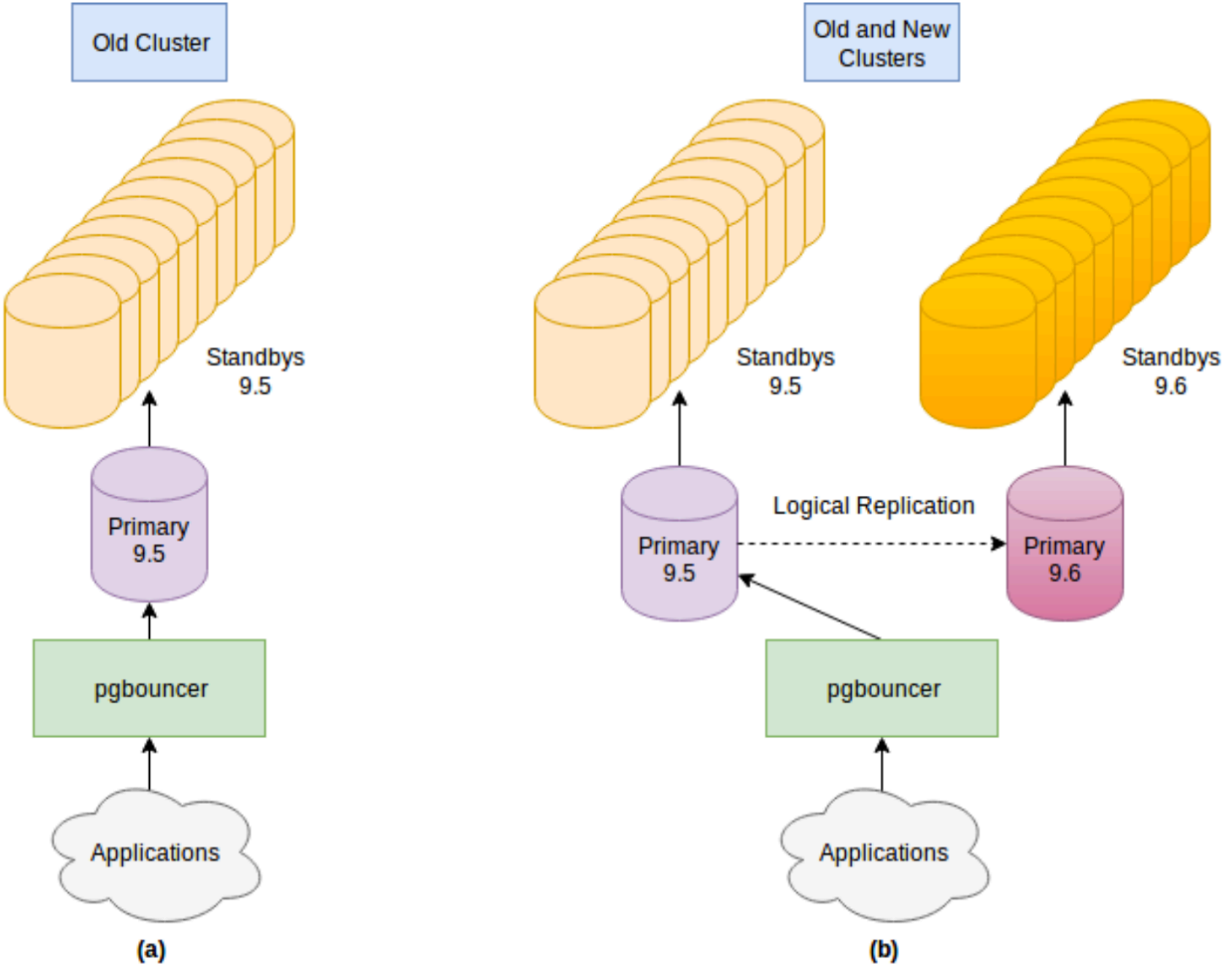


(c)

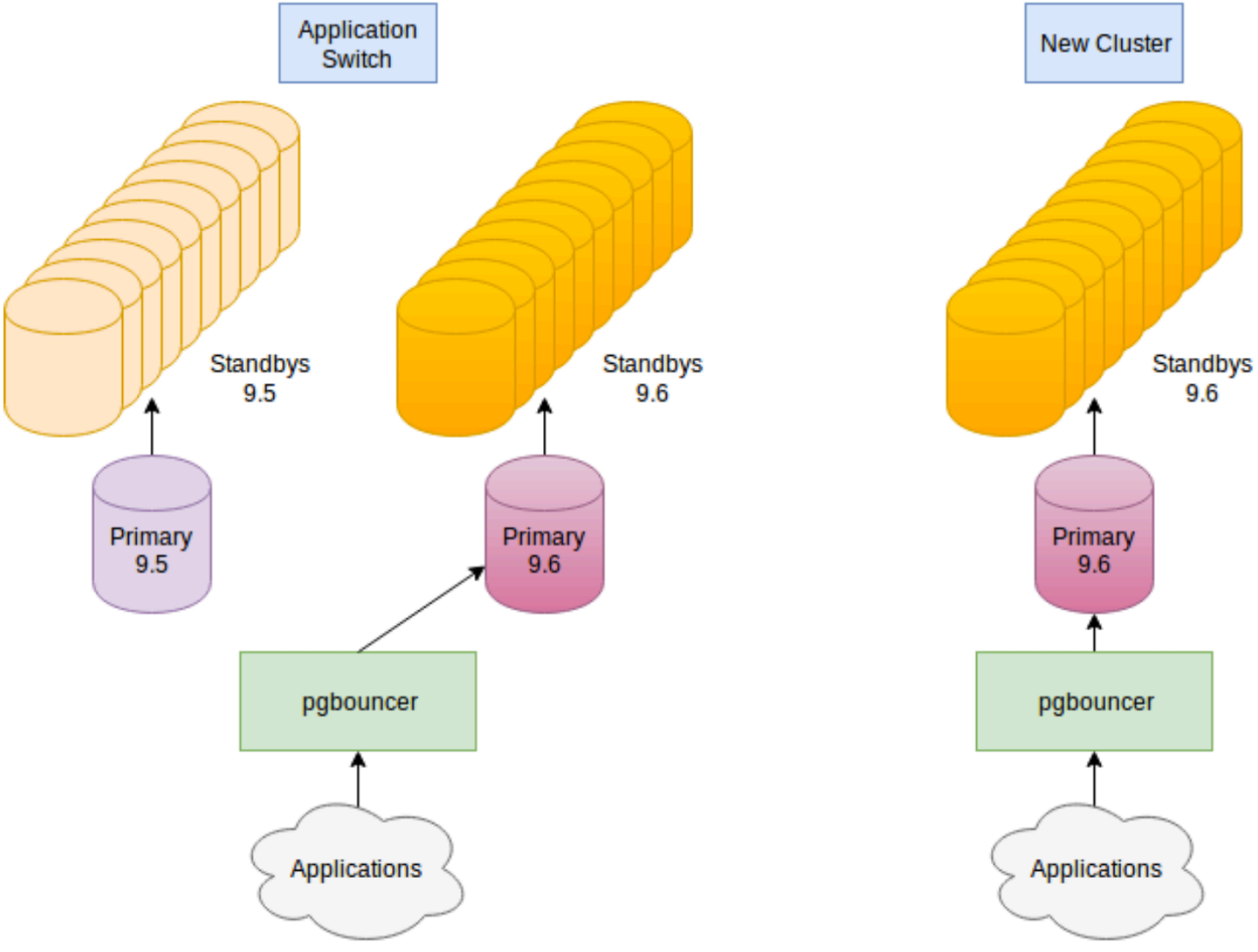


(d)

2nd Case: Read Scalability



2nd Case: Read Scalability



Test Environment

- Amazon EC2 t2.medium instances
- 2 Virtual CPUs
- 4 GB of RAM for memory
- 110 GB EBS for storage
- pgbench scale factor 2000

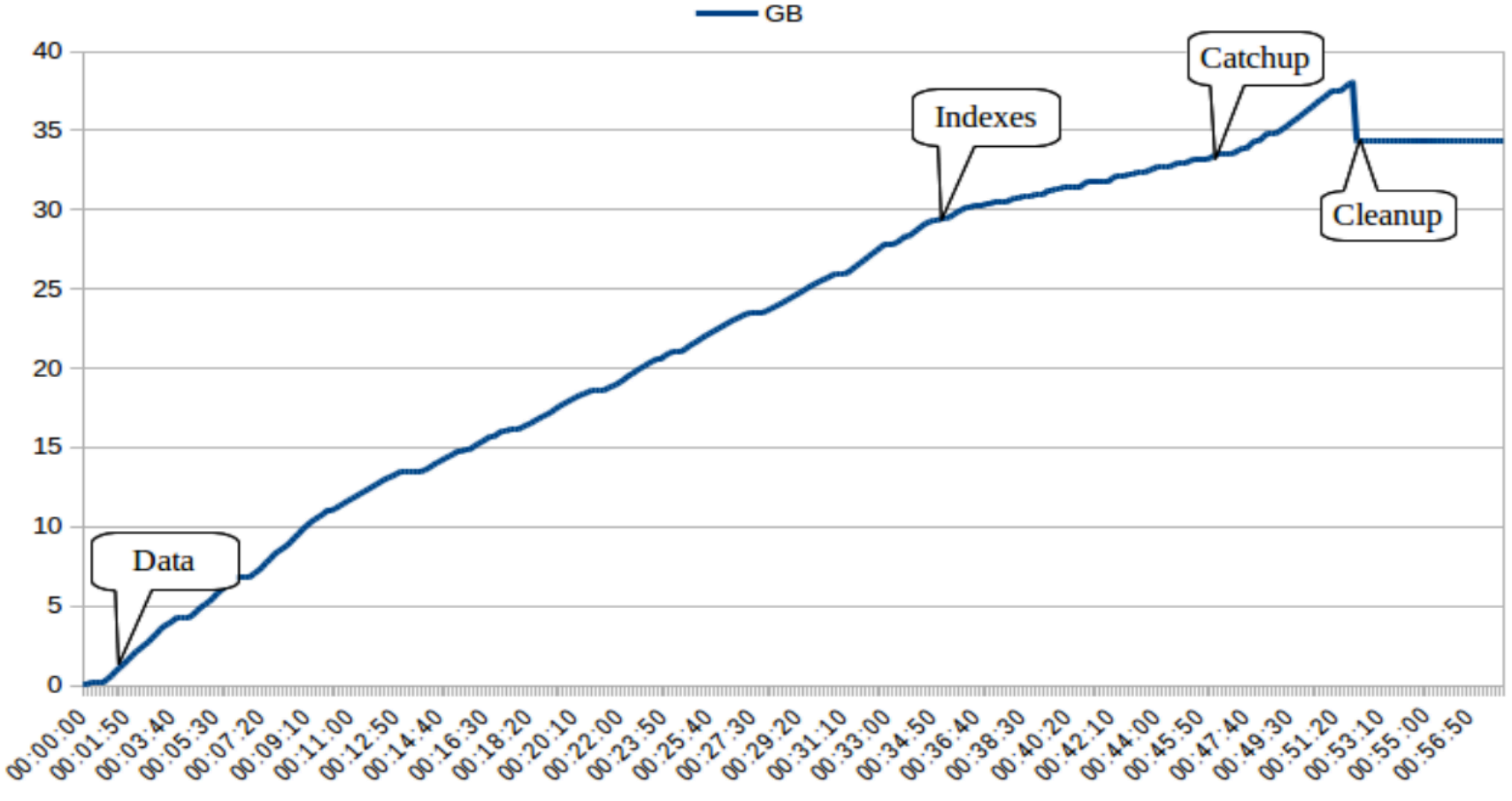
- PostgreSQL 9.5.6
- PostgreSQL 9.6.1
- Ubuntu 16.04
- PgBouncer 1.7.2
- Pglogical 2.0.0

Results

Metric (1st case)	pg_dump/pg_restore	pg_upgrade	pglupgrade
Primary Downtime	00:24:27	00:16:25	**00:00:03**
Partial cluster HA	00:24:27	00:28:56	**00:00:03**
Full cluster capacity	01:02:27	00:28:56	00:38:00
Length of upgrade	01:02:27	00:28:56	01:38:10
Extra disk space	800 MB	27 GB	10 GB

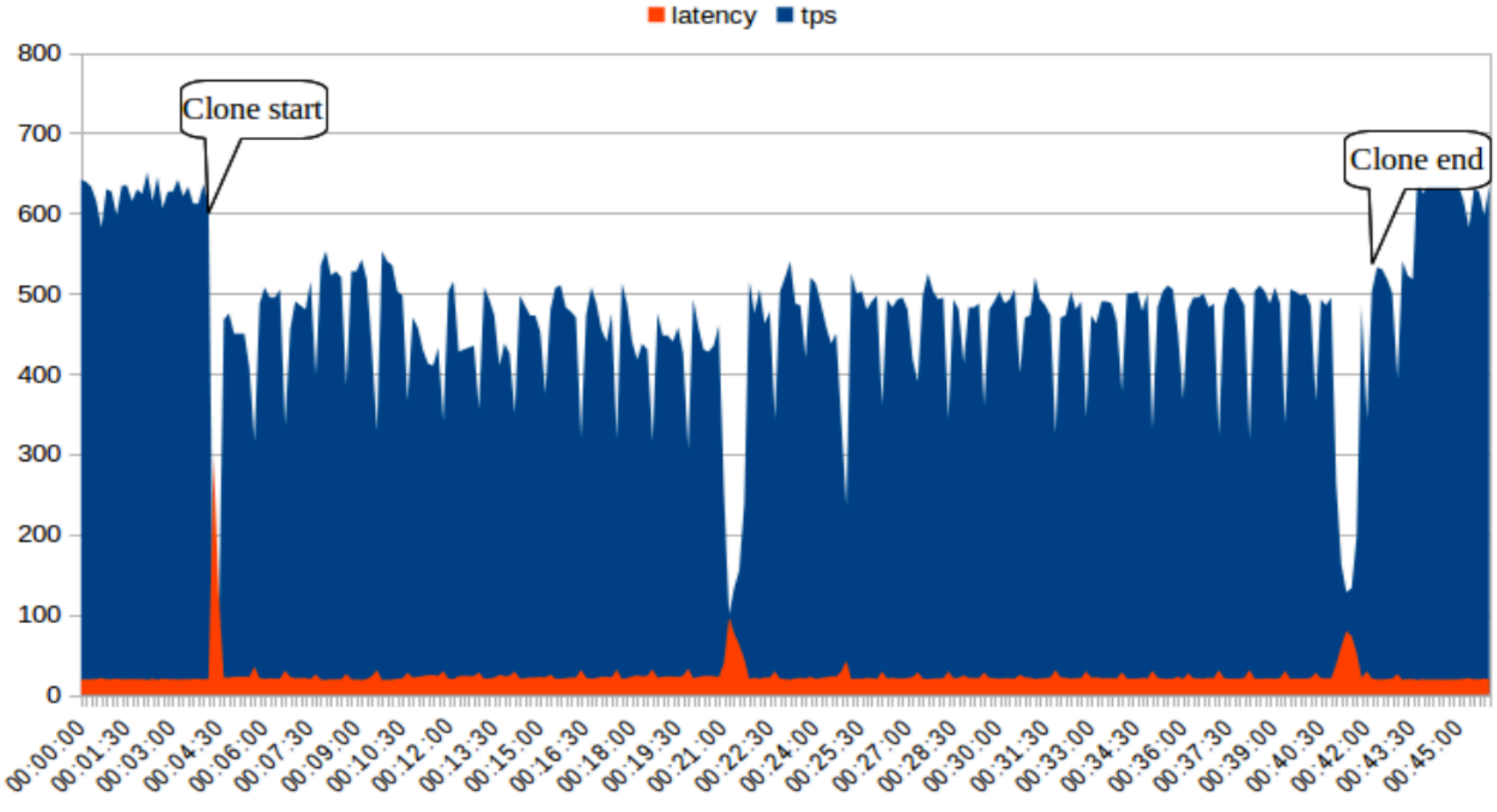
Metric (2nd case)	pg_dump/pg_restore	pg_upgrade	pglupgrade
Primary Downtime	00:23:52	00:17:03	**00:00:05**
Partial cluster HA	00:23:52	00:54:29	**00:00:05**
Full cluster capacity	00:23:52	03:19:16	**00:00:05**
Length of upgrade	00:23:52	03:19:16	01:02:10
Extra disk space	800 MB	27 GB	10 GB

Interpreting the Results



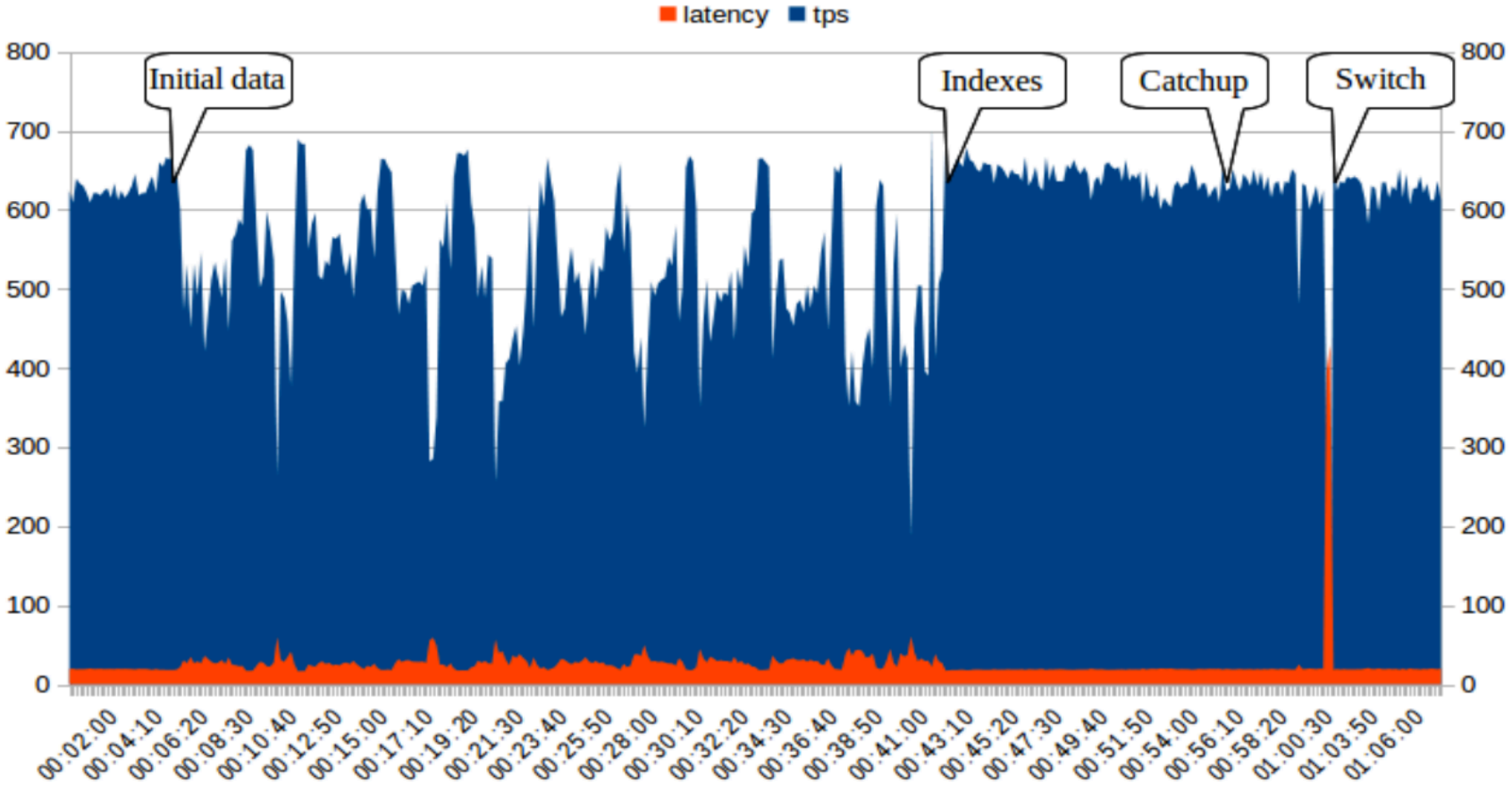
Database size growth during logical replication initialization

Interpreting the Results



Transaction rate and latency during **standby cloning** process

Interpreting the Results



Transaction rate and latency during the **upgrade** process

Back to the Future

- Need for a near-zero downtime automated upgrade solution for PostgreSQL [PgCon 2017 Developer Meeting]
- The tool will have a use in 2ndQuadrant and our customers
- PostgreSQL 10 will have built-in logical replication

Applicability

- Traditional data centers (bare-metal or virtual)
- Other Operating Systems (i.e Windows)
- Other DBMSs through logical replication (i.e MySQL, Oracle)
- Can work without PgBouncer

Limitations

- Spare resources on primary server (initial data copy)
- Cluster with too many writes (logical rep. catchup)
- Tables with PKs (or insert-only tables) - Postgres 10
- No transparent DDL replication

Conclusion

- Database clusters can be upgraded with **minimal downtime** without users being affected while the upgrade is happening.
- An application can still **respond** to the request only with a small drop in performance.
- Case studies prove that **pglupgrade** minimizes the downtime to the level of **3-5 seconds**.

Thanks!

Questions