PostgreSQL Sharding and HA: Theory and Practice

Aleksander Alekseev

A few words about me

- I live in Moscow, Russia;
- Develop software since 2007;
- Contribute to PostgreSQL since 2015;
- Work in Postgres Professional company;
- Interests: OSS, functional programming, electronics, SDR, distributed systems, blogging, podcasting;

In this talk

- A brief introduction to PostgreSQL replication (physical & logical);
- Solutions for HA / failover;
- Solutions for sharding;
- Q&A section :)

Target audience

- You believe that the replication is something very complicated;
- You think that the only way to scale is to scale horizontally;
- You’ve never configured physical and/or logical replication in PostgreSQL;
- You don’t know how to configure an auto-failover / HA;
- You would like to know what’s new in recent releases of PostgreSQL;
- You are looking for an idea for a project =).

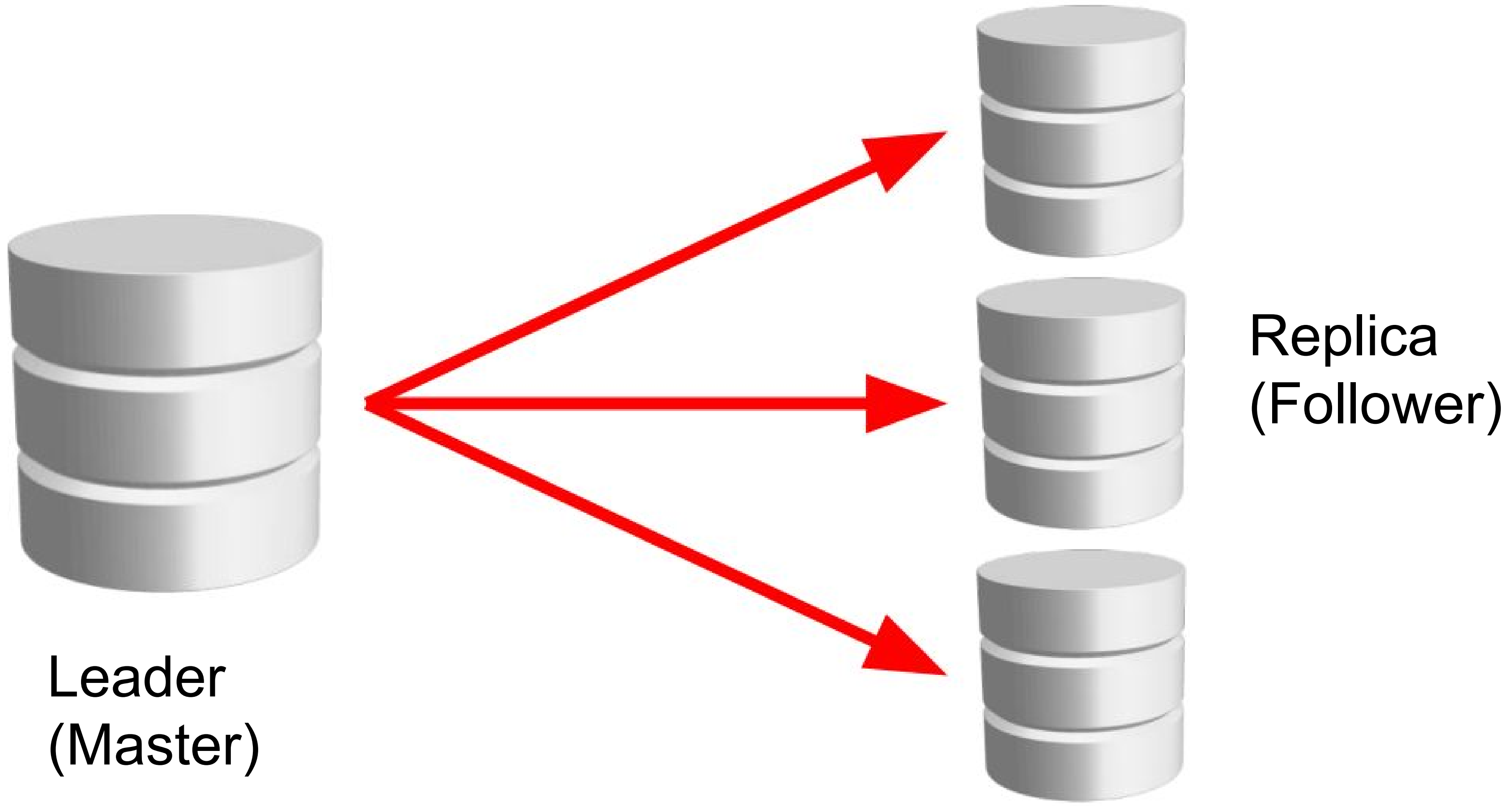
What is ***not*** in this talk

- A boring retelling of the documentation;
- For the interested listeners there will be links to additional materials;

Disclaimer

In this talk I will mention a lot of databases, extensions, etc. It doesn’t mean that I’m an expert in all of them.

Replication



What for?

- Load balancing
  - OLTP: writing to the leader, reading from replicas;
  - OLAP: analytical queries on a separate replica;
  - Taking backups from a separate replica;
- Failover / High Availability
  - Failover can be manual or automatic
- Delayed replication
- **Replication doesn’t replace backups!**

Streaming (or physical) replication

- In essence, it represents a transfer of the WAL over the network;
- Asynchronous
  - Fast, but the recent data can be lost;
- Synchronous
  - Slower (not as much in the same datacenter) but more reliable. It’s better to have at least two replicas;
- Also - cascading replication (I had to mention it on *some* slide).

Fun facts!

Streaming replication:

- Doesn’t work between servers with different architecture;
- Doesn’t work between different versions of PostgreSQL [1];
- May not work between different operating systems / compilers [2];
- Also transactions may become visible on the leader and the replica in different order;

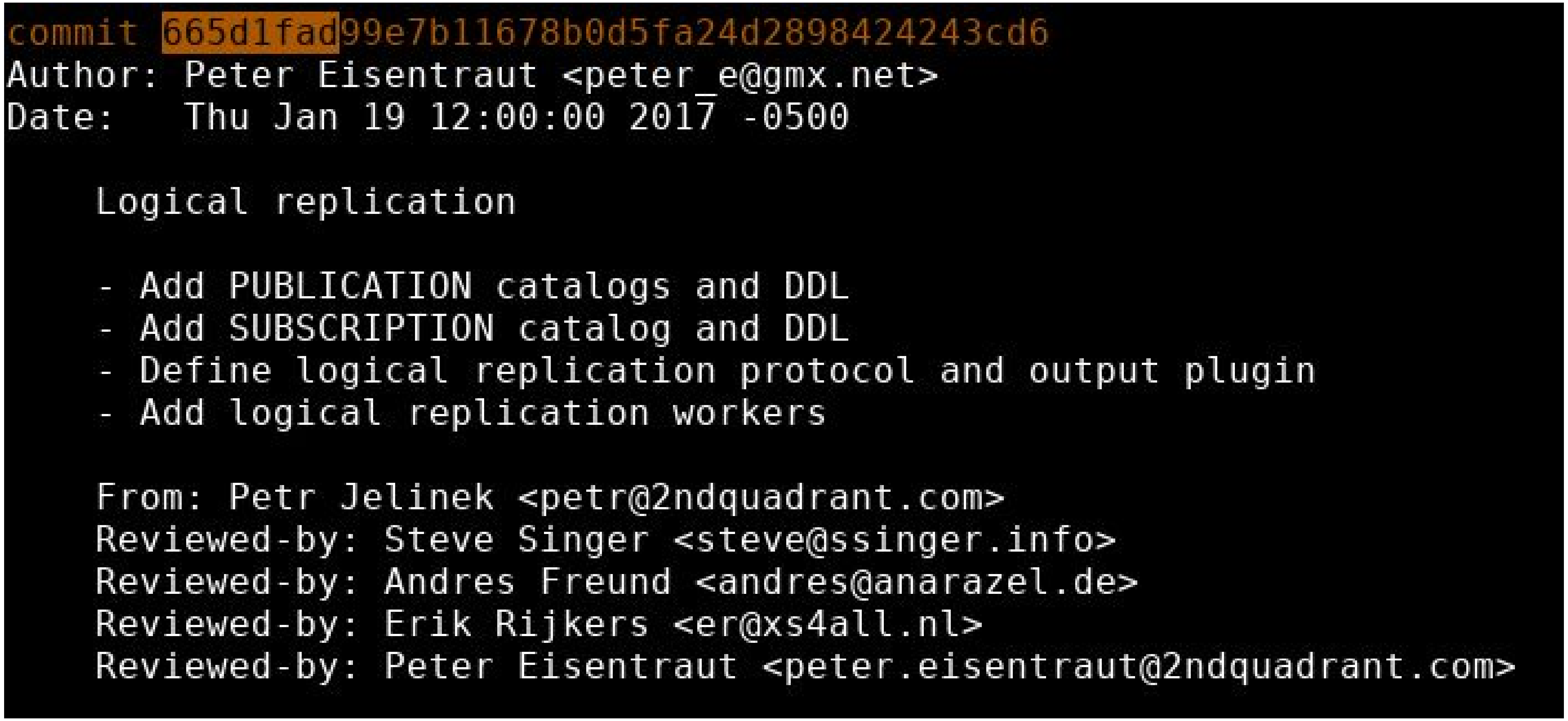
[1]: According to https://simply.name/ru/upgrading-postgres-to-9.4.html a typical downtime during the upgrade is a few minutes.

[2]: google://sizeof long compilers

Logical replication

- Out-of-the-box starting from PostgreSQL 10;
- Previous approaches: Slony, Londiste, pglogical;
    - I personally would not recommend them, unless you are *already* using one of these.

Credits: logical replication



```
commit 665d1fad99e7b11678b0d5fa24d2898424243cd6
Author: Peter Eisentraut <peter_e@gmx.net>
Date:   Thu Jan 19 12:00:00 2017 -0500

    Logical replication

    - Add PUBLICATION catalogs and DDL
    - Add SUBSCRIPTION catalog and DDL
    - Define logical replication protocol and output plugin
    - Add logical replication workers

    From: Petr Jelinek <petr@2ndquadrant.com>
    Reviewed-by: Steve Singer <steve@ssinger.info>
    Reviewed-by: Andres Freund <andres@anarazel.de>
    Reviewed-by: Erik Rijkers <er@xs4all.nl>
    Reviewed-by: Peter Eisentraut <peter.eisentraut@2ndquadrant.com>
```

Yet another type of replication? Why?

- To replicate only part of the data;
- To upgrade without a downtime;
- To create temporary tables on replicas;
- To write any other data on replicas;
- To write to the *replicated* tables;
- One replica can replicate data from multiple leaders;
- *In theory* — you can build a multimaster*;
- Other scenarios, when physical replication for some reason doesn’t work well.

* but it will be complicated and ugly.

Fun facts!

- Replicated tables may differ on the leader and the replica;
- The order of the columns may differ;
- The replica may have additional NULLable columns;
- The leader **can’t** have more columns then the replica, even if values in these columns are always NULL.

Limitations of the logical replication

- All replicated tables should have a primary key;
- DDL, TRUNCATE & sequences are not replicated;
- Triggers are not executed in some cases [1].

[1]: https://postgr.es/m/20171009141341.GA16999@e733.localdomain

synchronous_commit

- synchronous_commit = off
  - Asynchronous writing to the WAL, part of recent changes can be lost;
  - Unlike fsync = off can’t cause a database inconsistency;
- synchronous_commit = on
  - Synchronous writing to the WAL — leader’s and replica’s
- synchronous_commit = remote_write
  - Ditto, but without fsync() on replicas;
- synchronous_commit = local
  - Synchronous writing to the WAL on the leader only;
- synchronous_commit = remote_apply (>= 9.6)
  - Same as ‘on’ but also wait until changes will be applied to the data on replicas;

Fun fact!

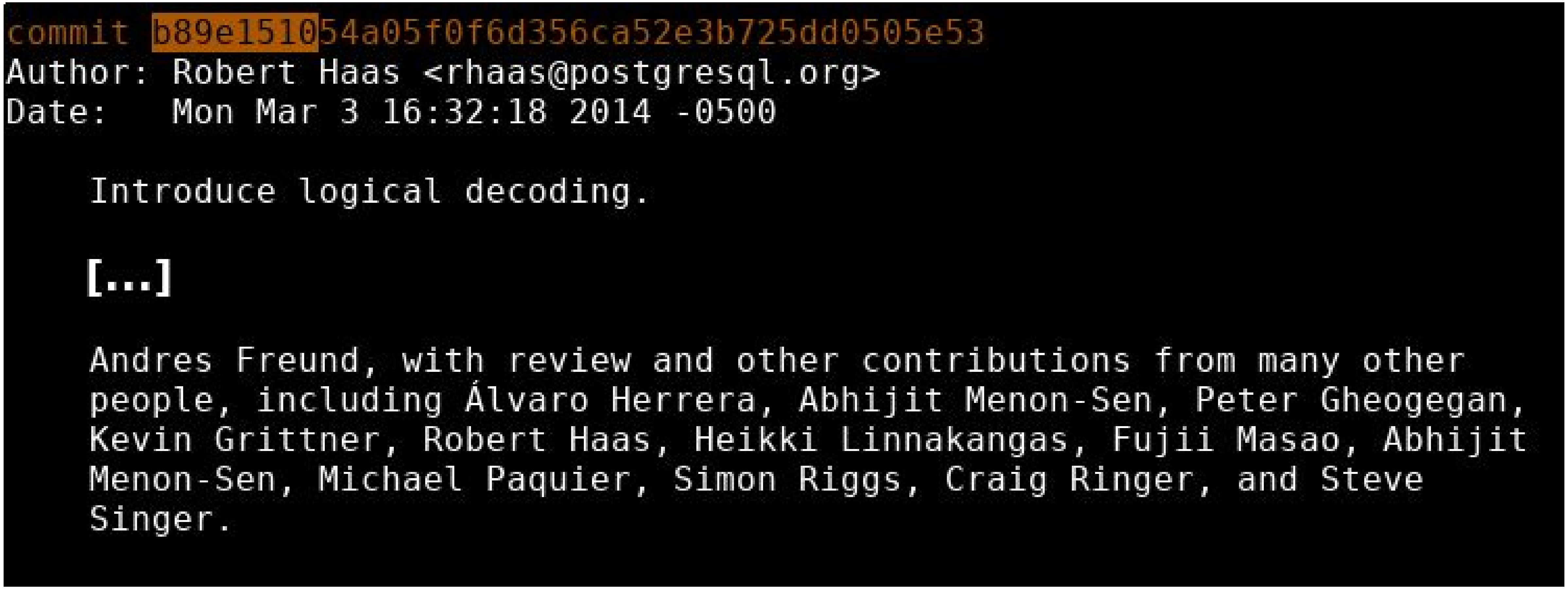
- synchronous_commit can be changed not only in postgresql.conf, but also in the session using SET command.

synchronous_standby_names

- synchronous_standby_names = ‘*’
  - Wait for ‘ack’ from any one replica;
- synchronous_standby_names = ANY 2(node1,node2,node3);
  - Quorum commit;
  - PostgreSQL >= 10;
- Other possible values [1] IMHO are not as interesting;

[1]: https://www.postgresql.org/docs/current/static/runtime-config-replication.html

Credits: logical decoding (PostgreSQL 9.4)



```
commit b89e151054a05f0f6d356ca52e3b725dd0505e53
Author: Robert Haas <rhaas@postgresql.org>
Date:   Mon Mar 3 16:32:18 2014 -0500

    Introduce logical decoding.

    [...]

    Andres Freund, with review and other contributions from many other
    people, including Álvaro Herrera, Abhijit Menon-Sen, Peter Gheogegan,
    Kevin Grittner, Robert Haas, Heikki Linnakangas, Fujii Masao, Abhijit
    Menon-Sen, Michael Paquier, Simon Riggs, Craig Ringer, and Steve
    Singer.
```

Logical decoding

```
$ pg_recvlogical --slot=myslot --dbname=eax --user=eax \
    --create-slot --plugin=test_decoding
$ pg_recvlogical --slot=myslot --dbname=eax --user=eax --start -f -
```

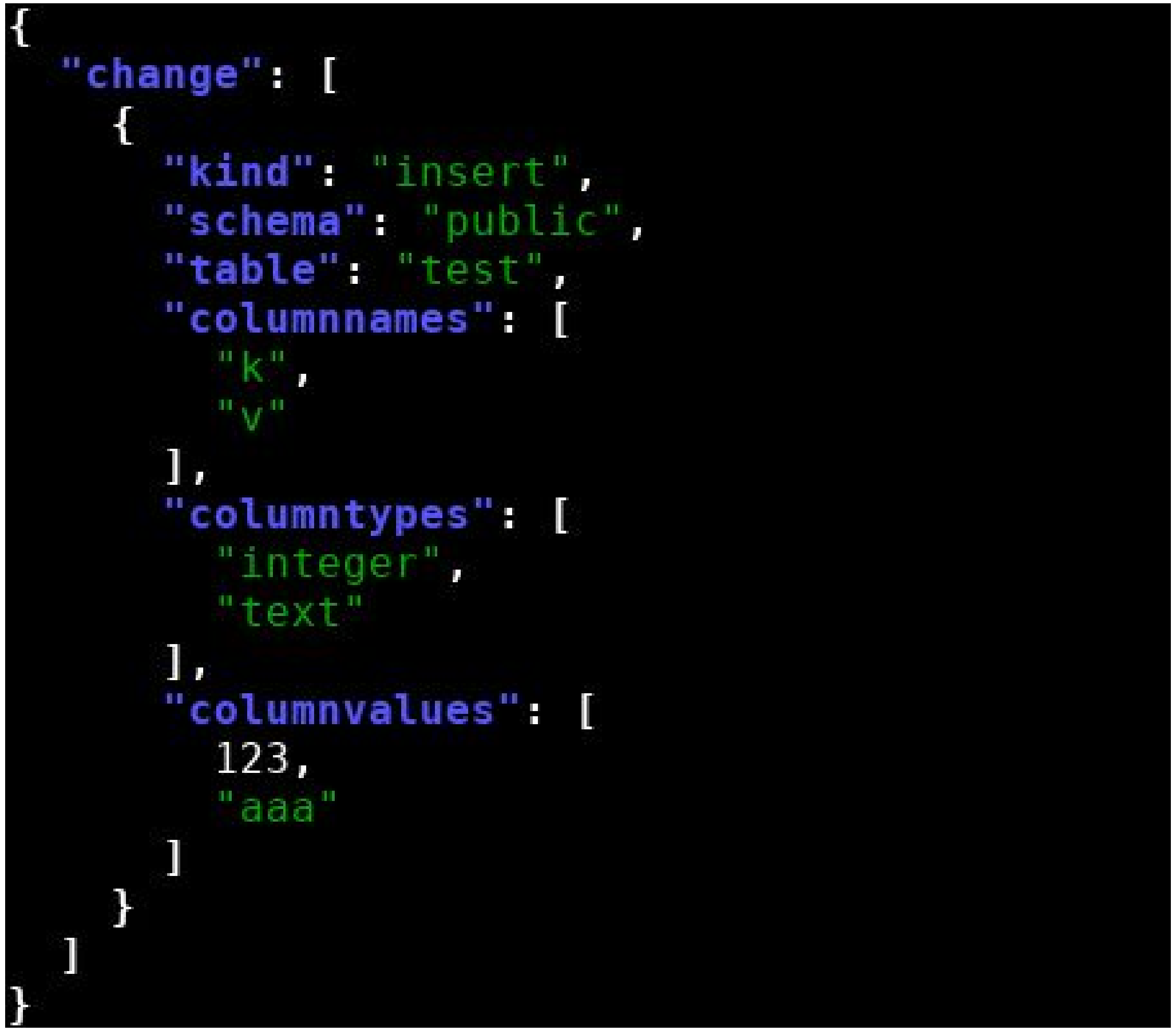
Logical decoding: output

```
BEGIN 560

COMMIT 560

BEGIN 561

table public.test: INSERT: k[text]:'aaa' v[text]:'bbb'

COMMIT 561
```

Logical decoding & JSON

```
$ pg_recvlogical --slot=myslot --dbname=eax --user=eax \
    --create-slot --plugin=wal2json

$ pg_recvlogical --slot=myslot --dbname=eax --user=eax --start -f - | jq
```

Logical decoding & JSON: output



```
{
  "change": [
    {
      "kind": "insert",
      "schema": "public",
      "table": "test",
      "columnnames": [
        "k",
        "v"
      ],
      "columntypes": [
        "integer",
        "text"
      ],
      "columnvalues": [
        123,
        "aaa"
      ]
    }
  ]
}
```

HA / Failover

- Manual
  - Used by many companies in practice;
  - OK if you have a moderate number of database servers (e.g. ~10);
  - See next two slides about modern hardware;
- Automatic
  - If your company is as big as Google :)

A few words about hardware: RAM

- You can put up to 3 TB of RAM in a single physical server these days;
- AWS instance x1.32xlarge (128 vCPU, 1952 GB RAM, 2 x 1920 GB SSD) costs 9603$ a month [1];
- Also AWS announced new instances with 4-16 TB of RAM [2][3].

[1]: https://aws.amazon.com/ec2/pricing/on-demand/

[2]: https://aws.amazon.com/ec2/instance-types/x1e/

[3]: https://www.theregister.co.uk/2017/05/16/aws_ram_cram/

A few words about hardware: hard drives

- You can buy a 1 TB SSD for ~300$ [1];
- You can put up to 900 TB of data in a single physical server these days;
- Next year: up to 1.5 PB.

[1]: Samsung MZ-75E1T0BW, https://amazon.com/dp/B00OBRFFAS

Manual failover howto

- Configure metrics and alerts using Nagios / Zabbix / Datadog / … ;
- Check them, make sure everything works;
- When something breaks:
  - Wake up in the night;
  - Figure out what’s going on;
  - Fix it (e.g. promote a replica);
- Since there are not many servers it will happen like once a year, so it’s OK;
- In many regards it’s more reliable than automatic failover;

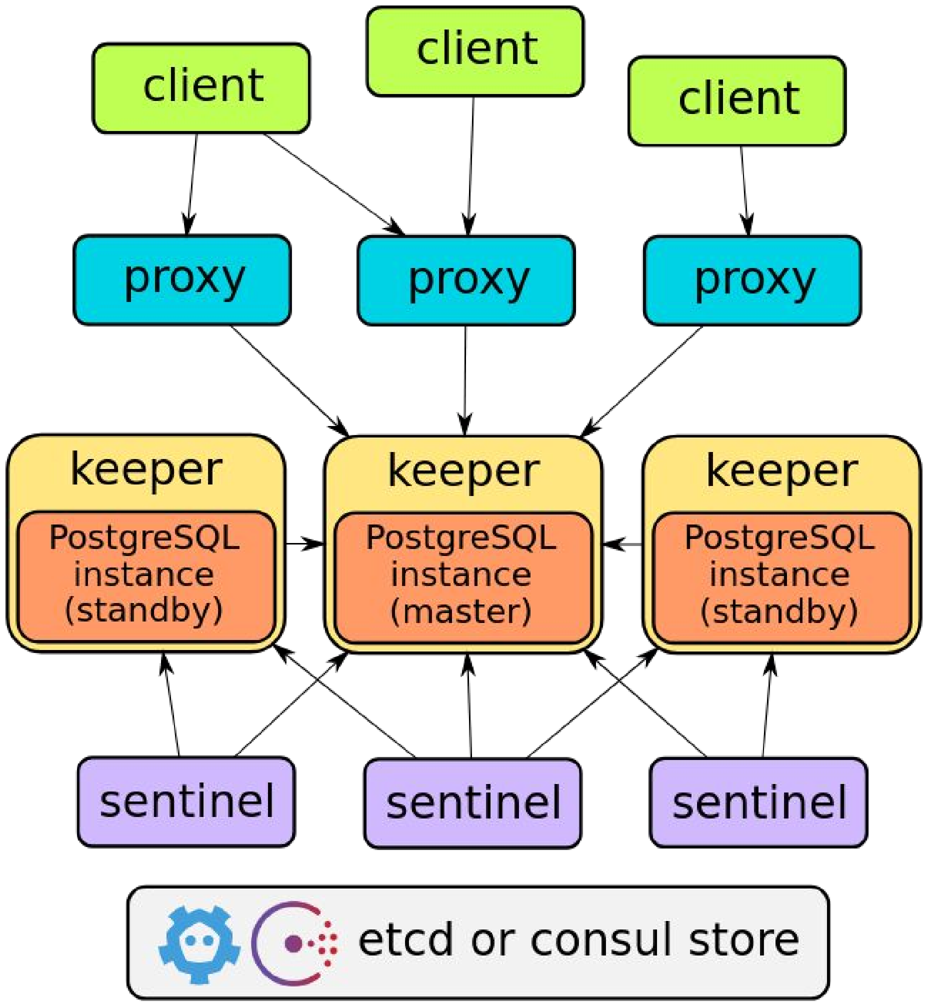
Automatic HA / failover

- Repmgr
- Patroni
- Stolon
- Postgres-XL
- Postgres-X2, ex. Postgres-XC (abandoned?)
- Multimaster (part of Postgres Pro Enterprise)
- ???

Stolon

- Developed since 2015 by Sorint.lab;
- Written in Go;
- Relies on Consul or etcd for service discovery;
- Supports integration with Kubernetes;
- Very easy to configure and maintain;
- Handles crashes and netsplits correctly;

Stolon: how does it work?



Fun facts!

- Stolon routes both reads and writes to the leader. There is a workaround [1];
- It uses Consul or etcd only as a key-value storage. In particular, it doesn’t rely on DNS support in Consul and other features.

[1]: https://github.com/sorintlab/stolon/issues/132

Postgres Pro Multimaster

- Looks like a regular RDBMS for the user;
- Is a part of Postgres Pro Enterprise;
- Based on paper “Clock-SI: Snapshot Isolation for Partitioned Data Stores Using Loosely Synchronized Clocks” [*];
- Developers: Konstantin Knizhnik, Stas Kelvich, Arseny Sher;

- https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/samehe-clocksi.srds2013.pdf

The Multimaster team

Existing solutions for sharding

- Manual sharding
- Citus
- Greenplum
- pg_shardman (part of Postgres Pro Enterprise)
- ???

Existing solutions for sharding

- Manual sharding
- Citus
- Greenplum
- pg_shardman (part of Postgres Pro Enterprise)
- ???

AFAIK designed mostly for analytics

Manual sharding

- Used in practice by many companies;
- It’s OK if you don’t have (many) distributed transactions;
- Rebalancing is done quite simple with logical replication;
- For distributed transactions you can use Percolator-like approach [1];
  - Provides snapshot isolation only, so the write skew anomaly is possible. On the other hand in some RDBMS this is the best you can get [2].
- Or even something simpler (e.g. based on log of idempotent operations);

[1]: http://rystsov.info/2012/09/01/cas.html

[2]: https://github.com/ept/hermitage/

pg_shardman

- Developed by the Postgres Pro Multimaster team;
- Is a part of Postgres Pro Enterprise;
- Supports replication factor > 1 (which is not true for some alternatives);
- It is currently in a state of beta-release;
- Please contact info@postgrespro.com and ask for a trial;
- Note that PostgresPro Enterprise is free for educational and non-commercial use;

Not quite PostgreSQL

- Amazon Aurora
- CockroachDB

Amazon Aurora

- ACID with transparent failover, sharding and distributed transactions;
- Announced in 2014;
- Exists only in a cloud;
- Is compatible with MySQL and PostgreSQL [1] on the protocol level;
- There is a paper [2];

[1]: since Nov 2016 https://news.ycombinator.com/item?id=13072861

[2]: http://www.allthingsdistributed.com/files/p1041-verbitski.pdf

CochroachDB

- ACID with transparent failover, sharding and distributed transactions;
- Announced in 2014, is written in Go, is developed by ex-Google employees;
- Free and open source software;
- Is compatible with PostgreSQL on the protocol level;
- Passes Jepsen [*];
- Based on Spanner paper [*];

- https://www.cockroachlabs.com/blog/cockroachdb-beta-passes-jepsen-testing/
- https://static.googleusercontent.com/media/research.google.com/en//archive/spanner-osdi2012.pdf

Links

- https://www.postgresql.org/docs/10/static/index.html
- https://github.com/sorintlab/stolon/
- https://github.com/eulerto/wal2json
- https://github.com/posix4e/jsoncdc
- https://github.com/citusdata/citus
- http://greenplum.org/
- https://postgrespro.com/products/postgrespro/enterprise
- https://aws.amazon.com/rds/aurora/
- https://www.cockroachlabs.com/

See you in Russia!

Thank you for your attention!

- a.alekseev@postgrespro.ru
- https://afiskon.github.io/
- https://postgrespro.com/
- https://github.com/postgrespro/