Standby in Production

Konstantin Evteev

Ottawa 2019

Avito has a nationwide audience

Every month more than 35 million people use Avito

Pageviews per month:

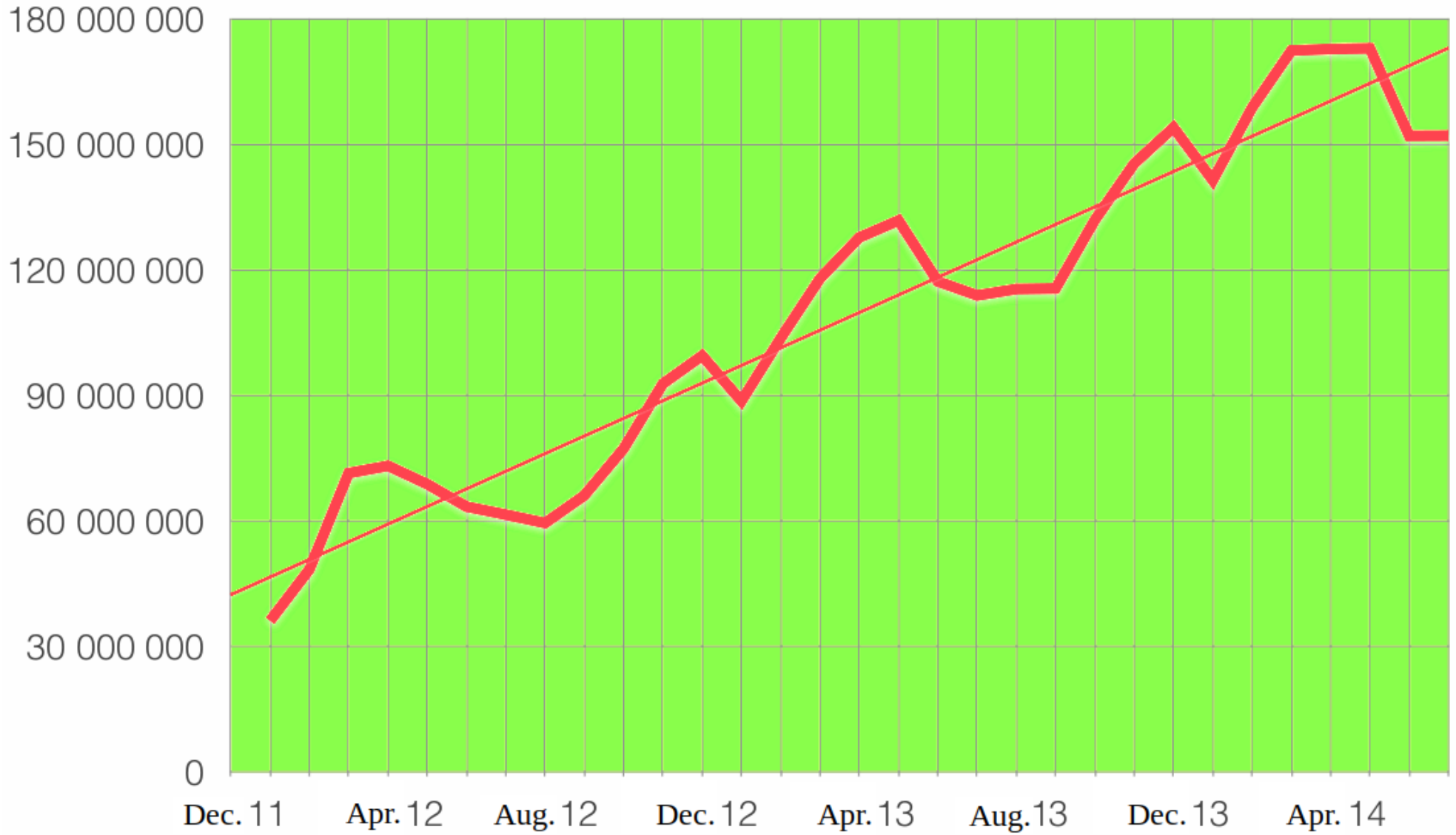
Statistics: February 2018

Some figures

- **600+ servers;**
- **4,5 Gbit/sec TX, 2 Gbit/sec RX without static;**
- **about a million queries per minute to the backend;**
- **270TB of images;**
- **>20 TB in Postgres on 100 nodes:**
  - **7 - 8K TPS on most nodes;**
  - **The largest - 20k TPS, 5 TB;**

Avito

pageviews



Avito

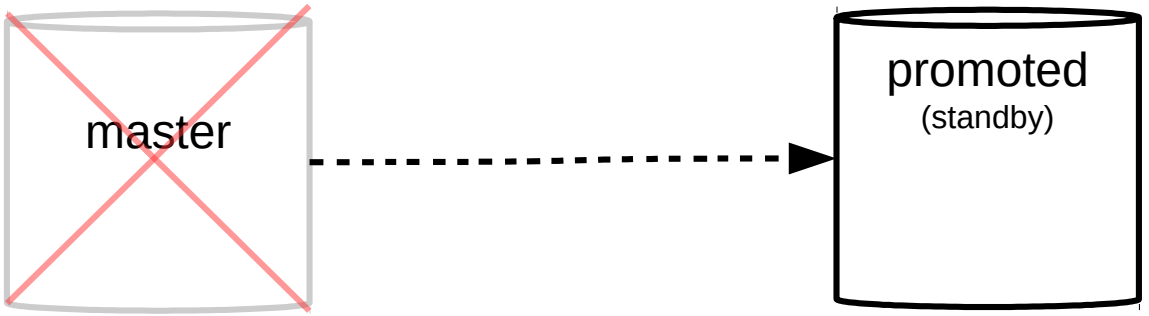
Contents

- a few words about standby and its history in general
- problems and solutions in replication based horizontal scale-out;
- Avito’s solution to avoid stale reads from replica implementation;
- possible pitfalls while using standby with high request rate, applying DDL, receiving WAL files from archive;
- handling issues with technique of using several standbys in production and routing queries between them;
- logical replication based scaling example to compare
- conclusions and standby major upgrade features

Avito

Standby

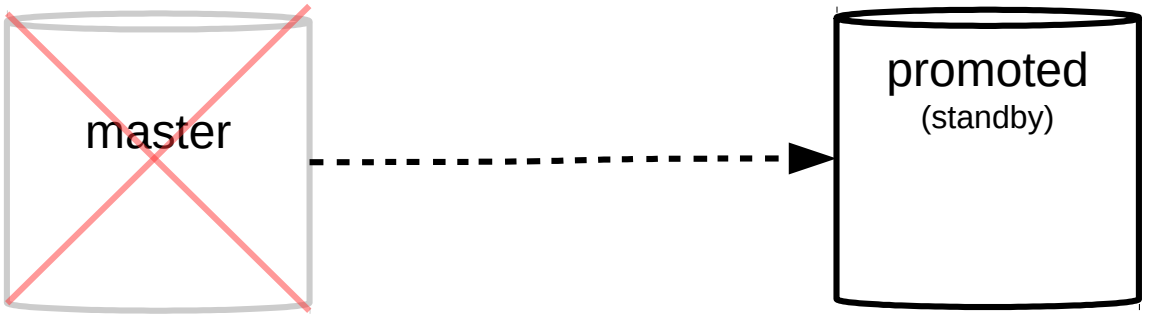
1 High Availability



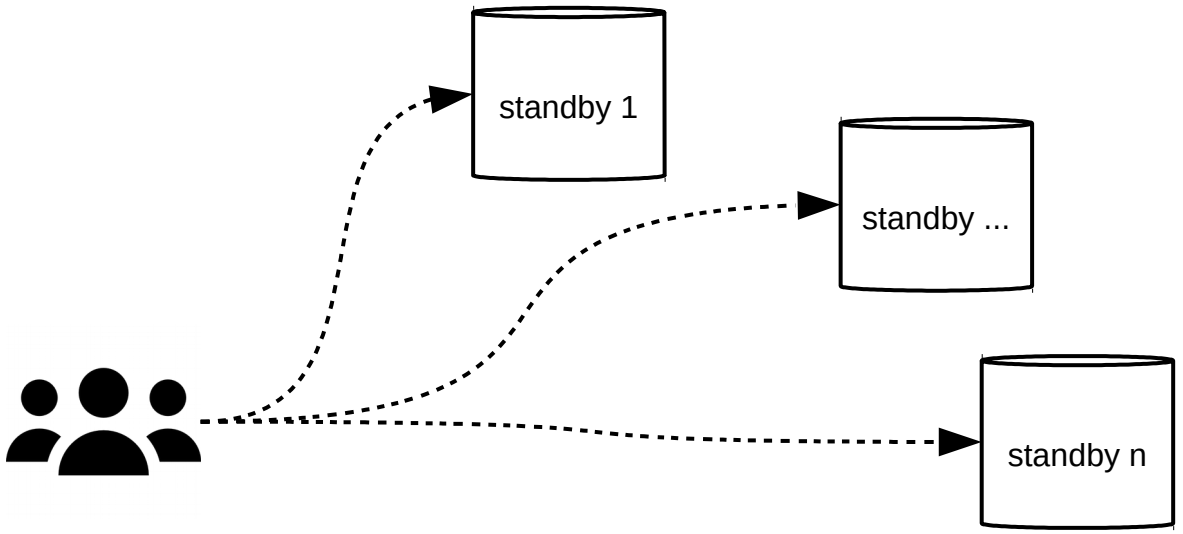
Avito

Standby

1 High Availability



2 Scaling



Avito

History

2000: RServ

Avito

History

2000: RServ

2001: PostgreSQL 7.1: write-ahead log

Avito

History

2000: RServ

2001: PostgreSQL 7.1: write-ahead log

2004: Slony

Avito

History

2000: RServ

2001: PostgreSQL 7.1: write-ahead log

2004: Slony

2005: PostgreSQL 8.0: point-in-time recovery

Avito

History

2000: RServ

2001: PostgreSQL 7.1: write-ahead log

2004: Slony

2005: PostgreSQL 8.0: point-in-time recovery

2007: SkyTools – PgQ & Londiste

Avito

History

2000: RServ

2001: PostgreSQL 7.1: write-ahead log

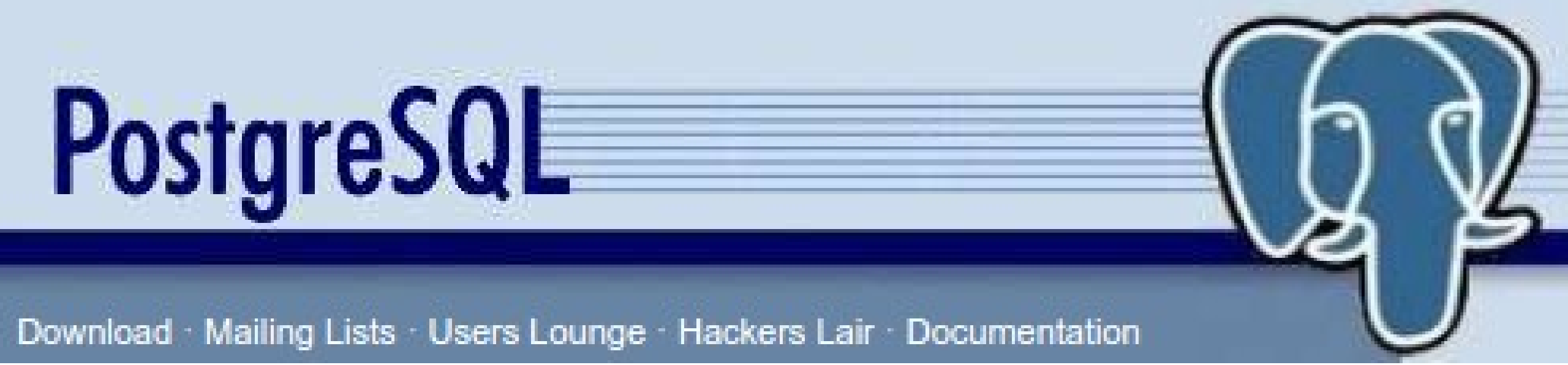
2004: Slony

2005: PostgreSQL 8.0: point-in-time recovery

2007: SkyTools – PgQ & Londiste

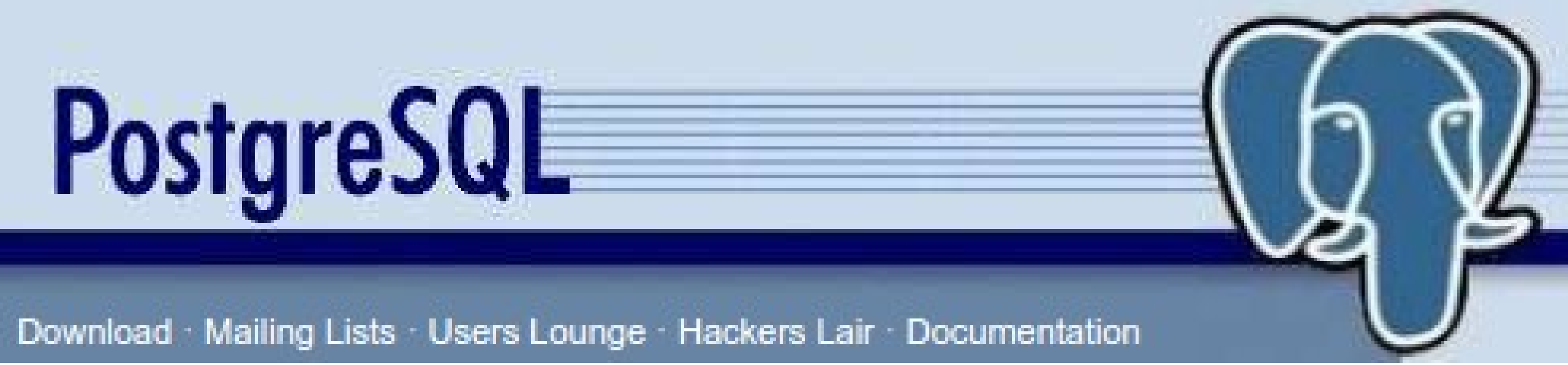
2008: 8.3 Easier administration of warm standby servers

Avito



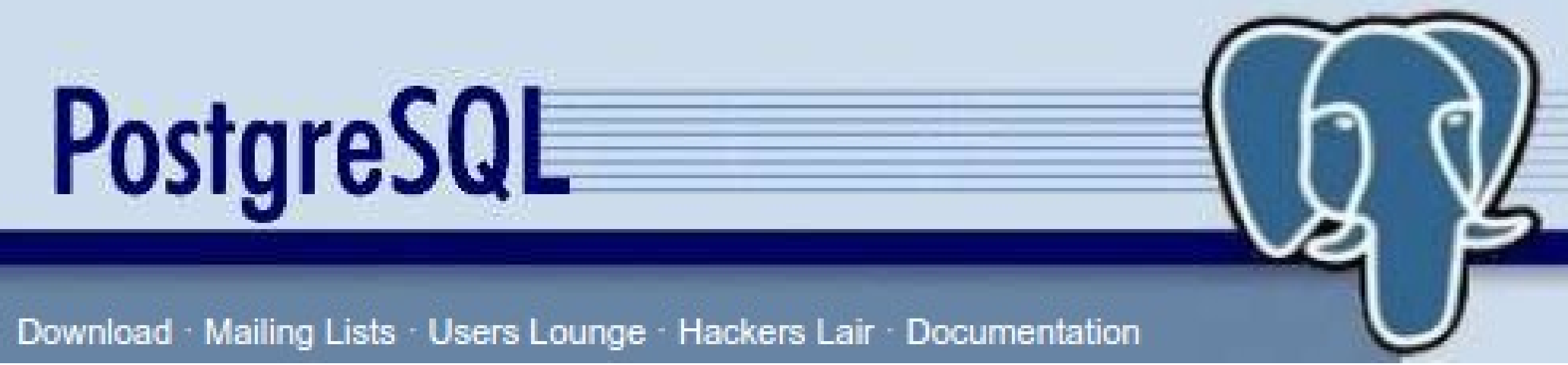
1. 2010: 9.0: hot standby, streaming replication

Avito



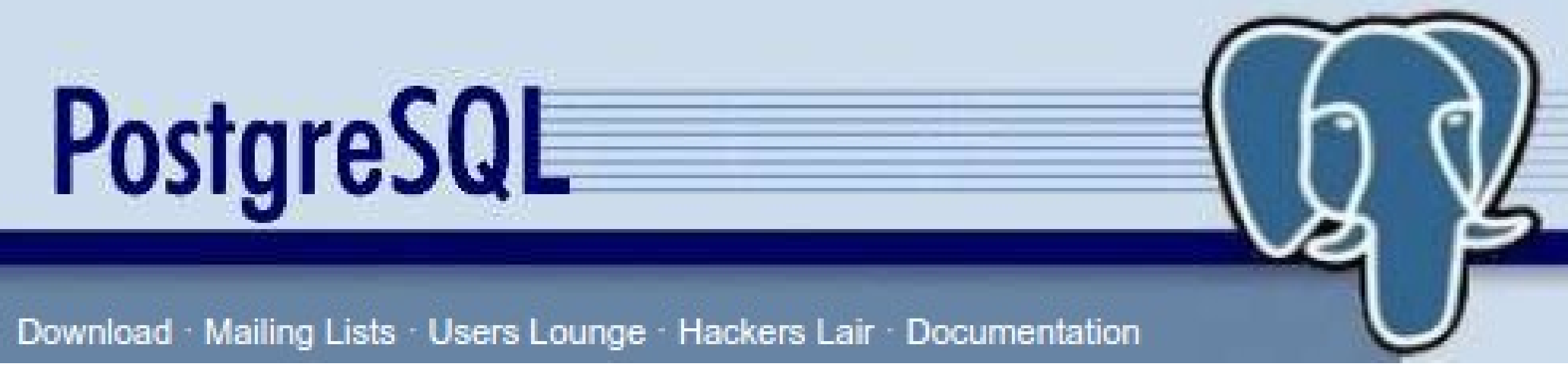
1. 2010: 9.0: hot standby, streaming replication
2. 2011: 9.1: synchronous replication

Avito



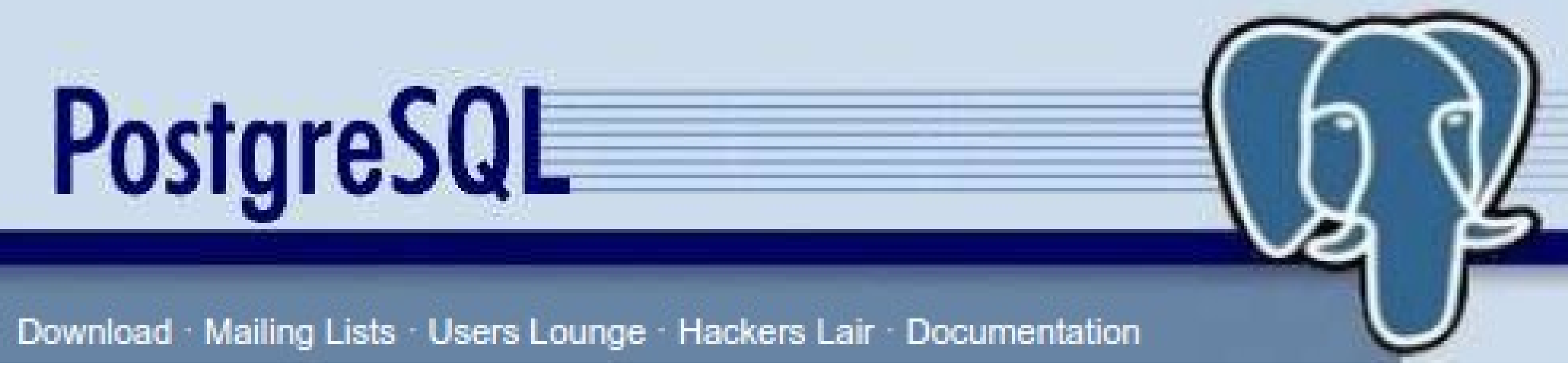
1. 2010: 9.0: hot standby, streaming replication
2. 2011: 9.1: synchronous replication
3. 2013: 9.3: sb can follow timeline switch

Avito



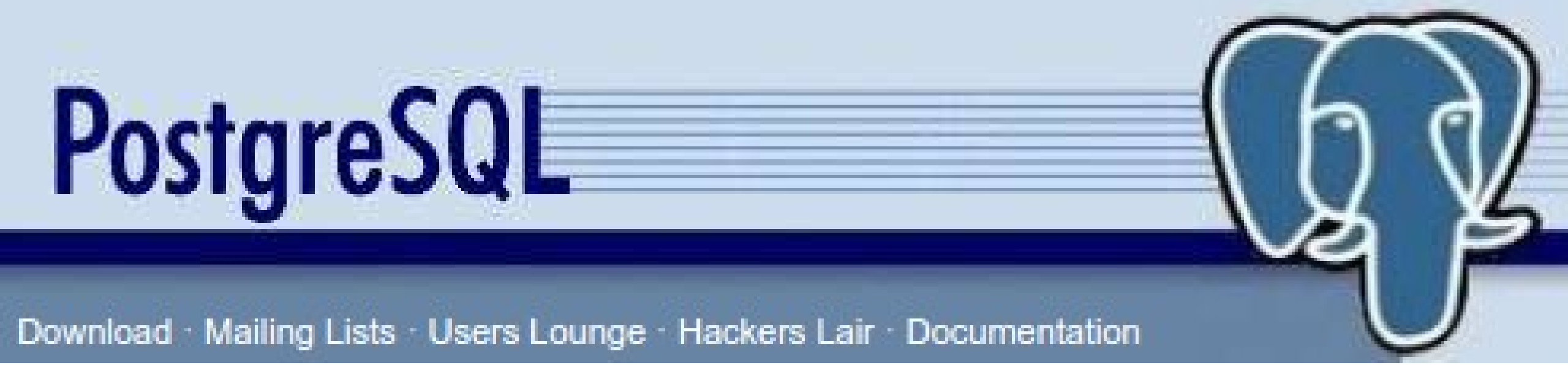
1. 2010: 9.0: hot standby, streaming replication
2. 2011: 9.1: synchronous replication
3. 2013: 9.3: sb can follow timeline switch
4. 2014: 9.4: replication slots, logical decoding

Avito



1. 2010: 9.0: hot standby, streaming replication
2. 2011: 9.1: synchronous replication
3. 2013: 9.3: sb can follow timeline switch
4. 2014: 9.4: replication slots, logical decoding
5. 2016: 9.6 multiple synchronous standbys, remote_apply

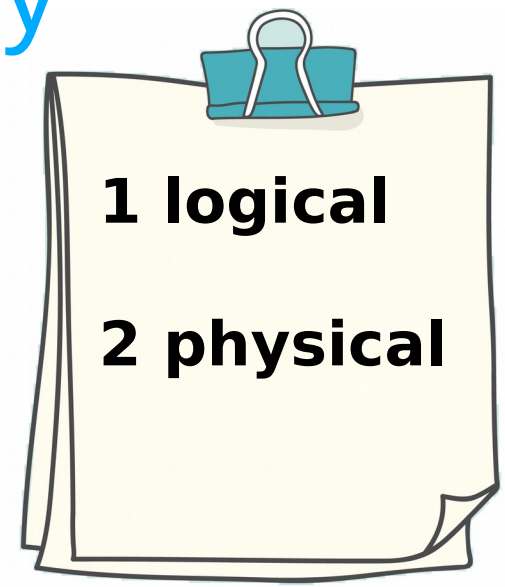
Avito



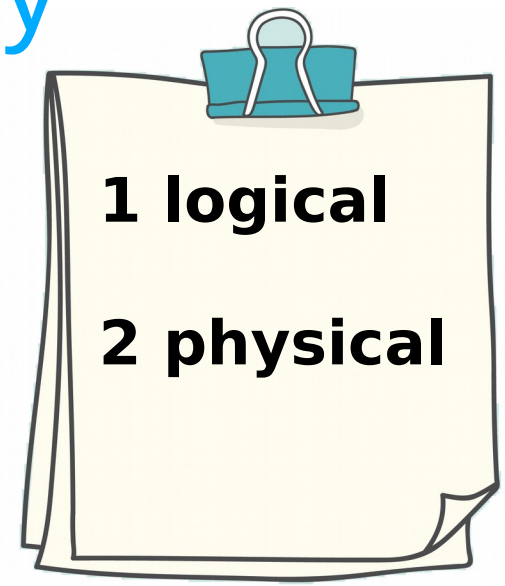
1. 2010: 9.0: hot standby, streaming replication
2. 2011: 9.1: synchronous replication
3. 2013: 9.3: sb can follow timeline switch
4. 2014: 9.4: replication slots, logical decoding
5. 2016: 9.6 multiple synchronous standbys, remote_apply
6. 2017: 10: logical replication

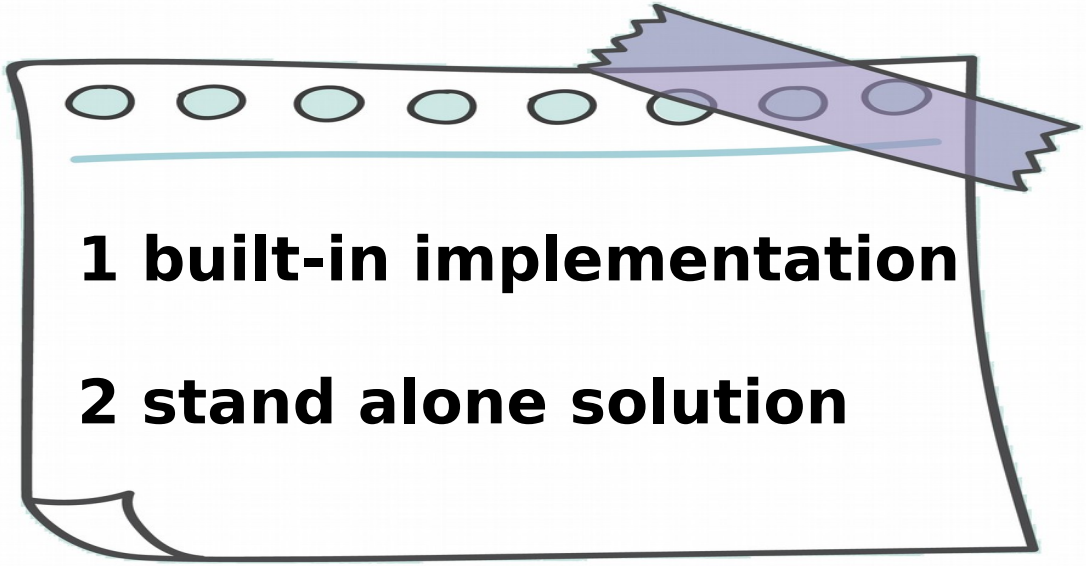
Avito

Standby



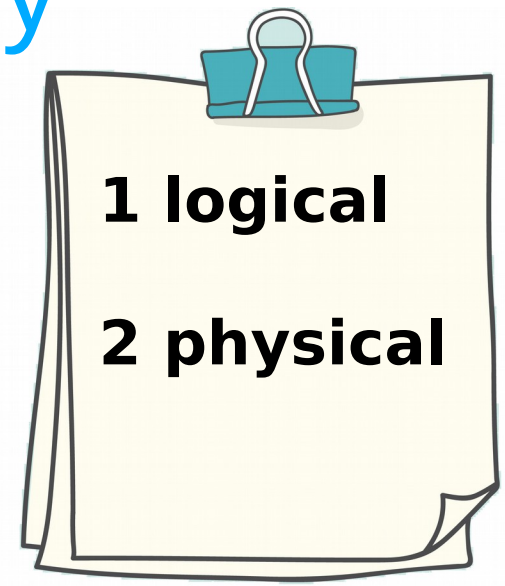
Standby

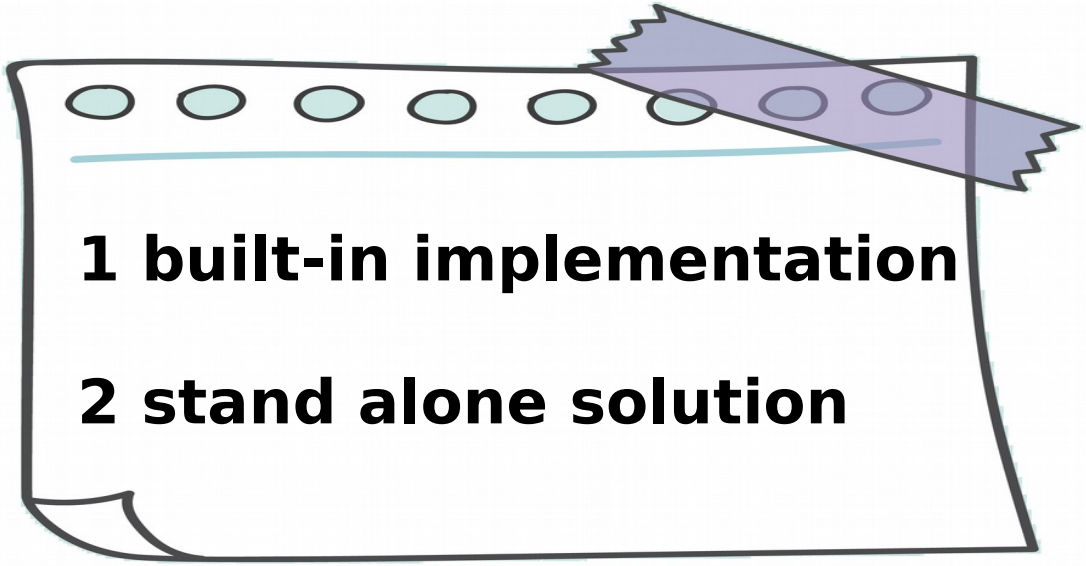


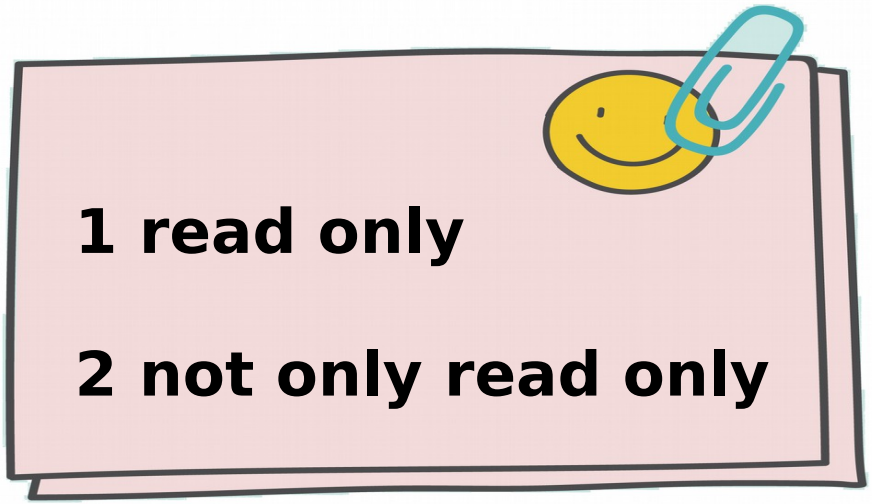


Avito

Standby

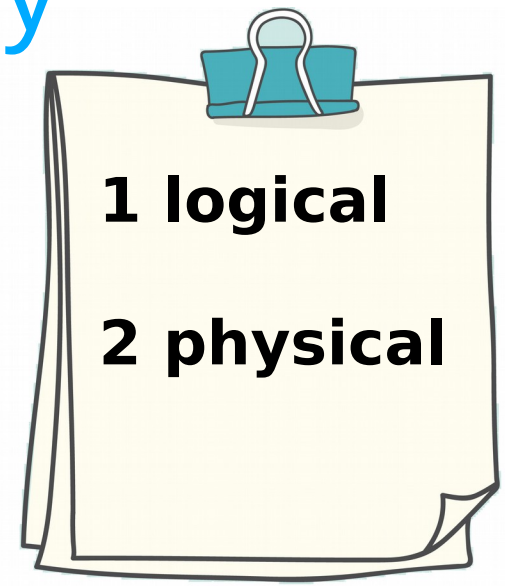


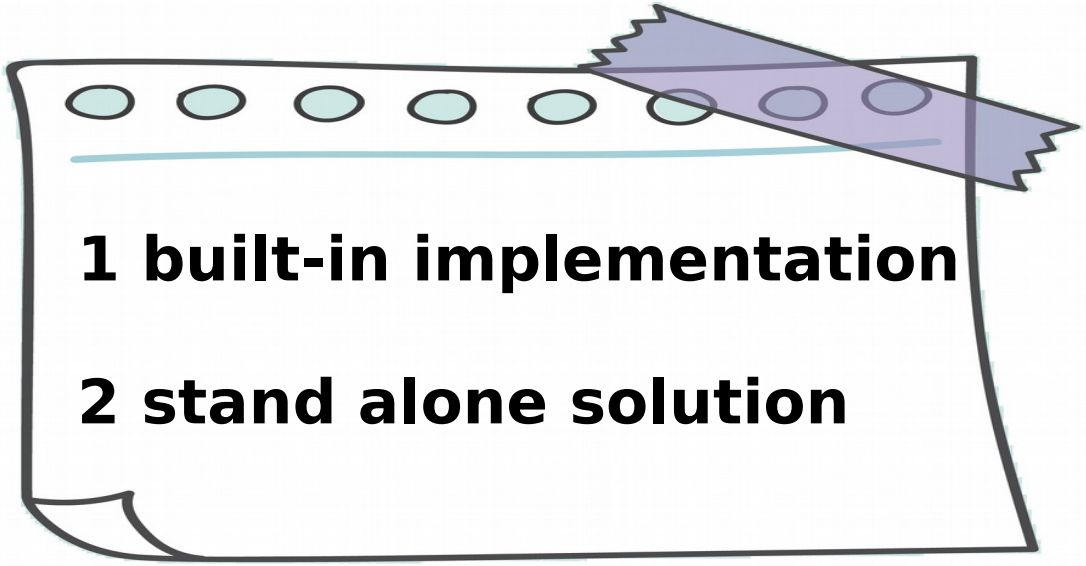


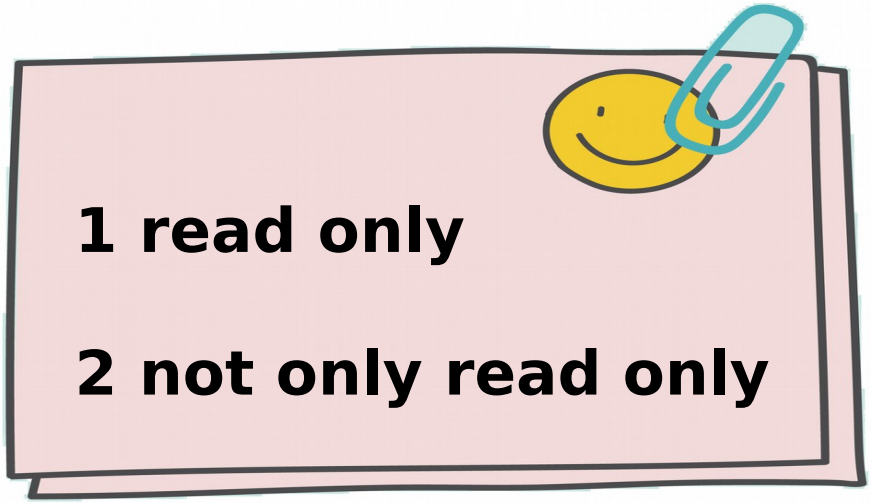


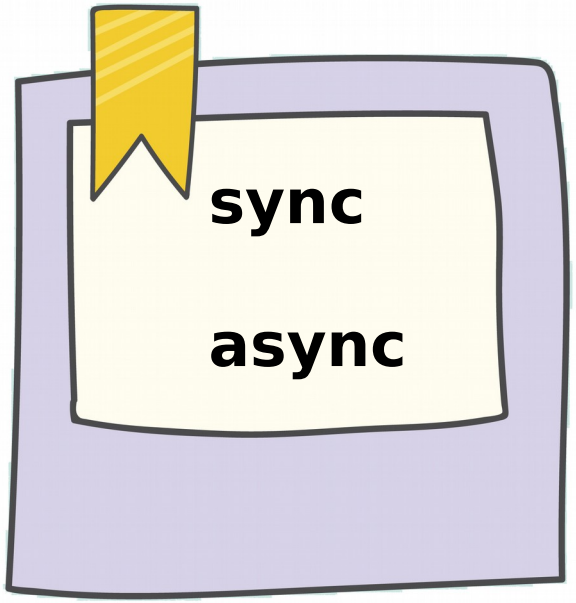
Avito

Standby





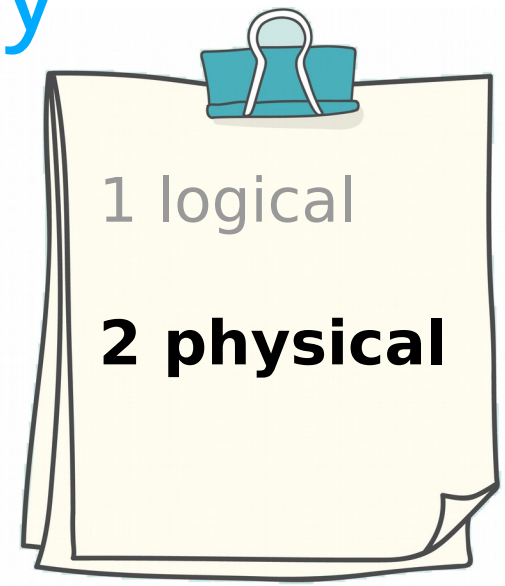


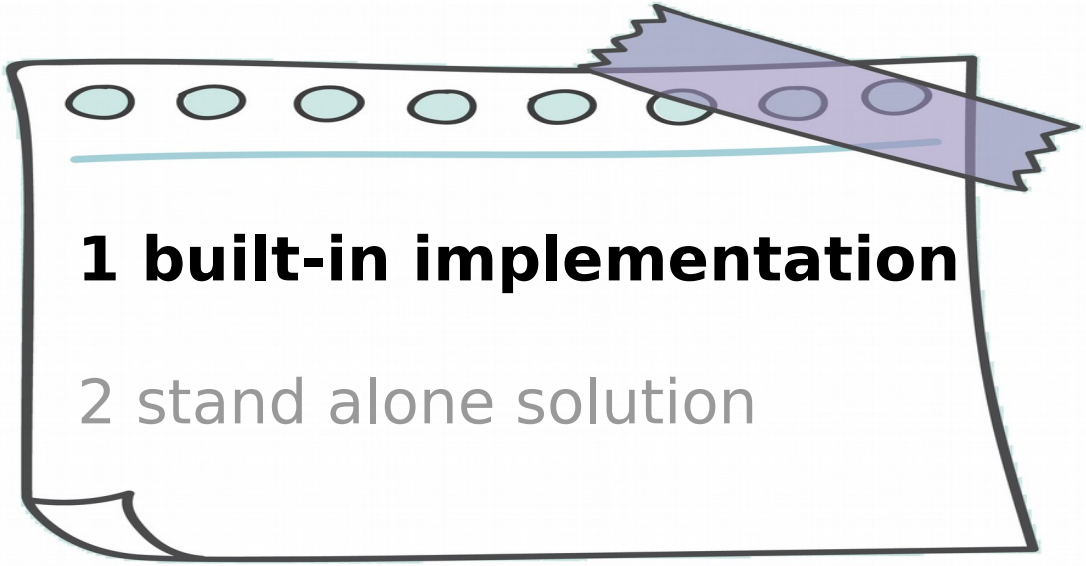


...

Avito

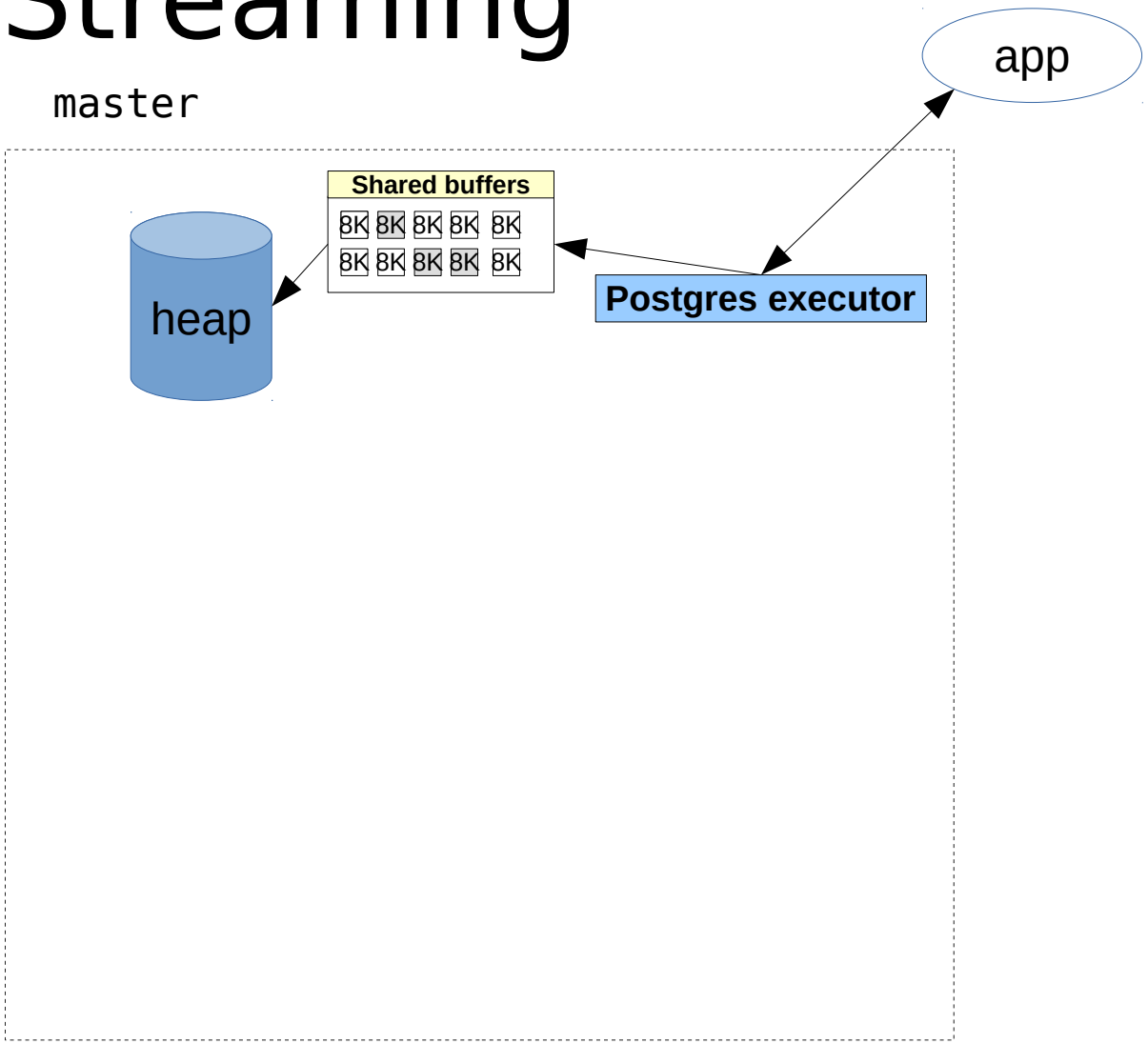
Standby





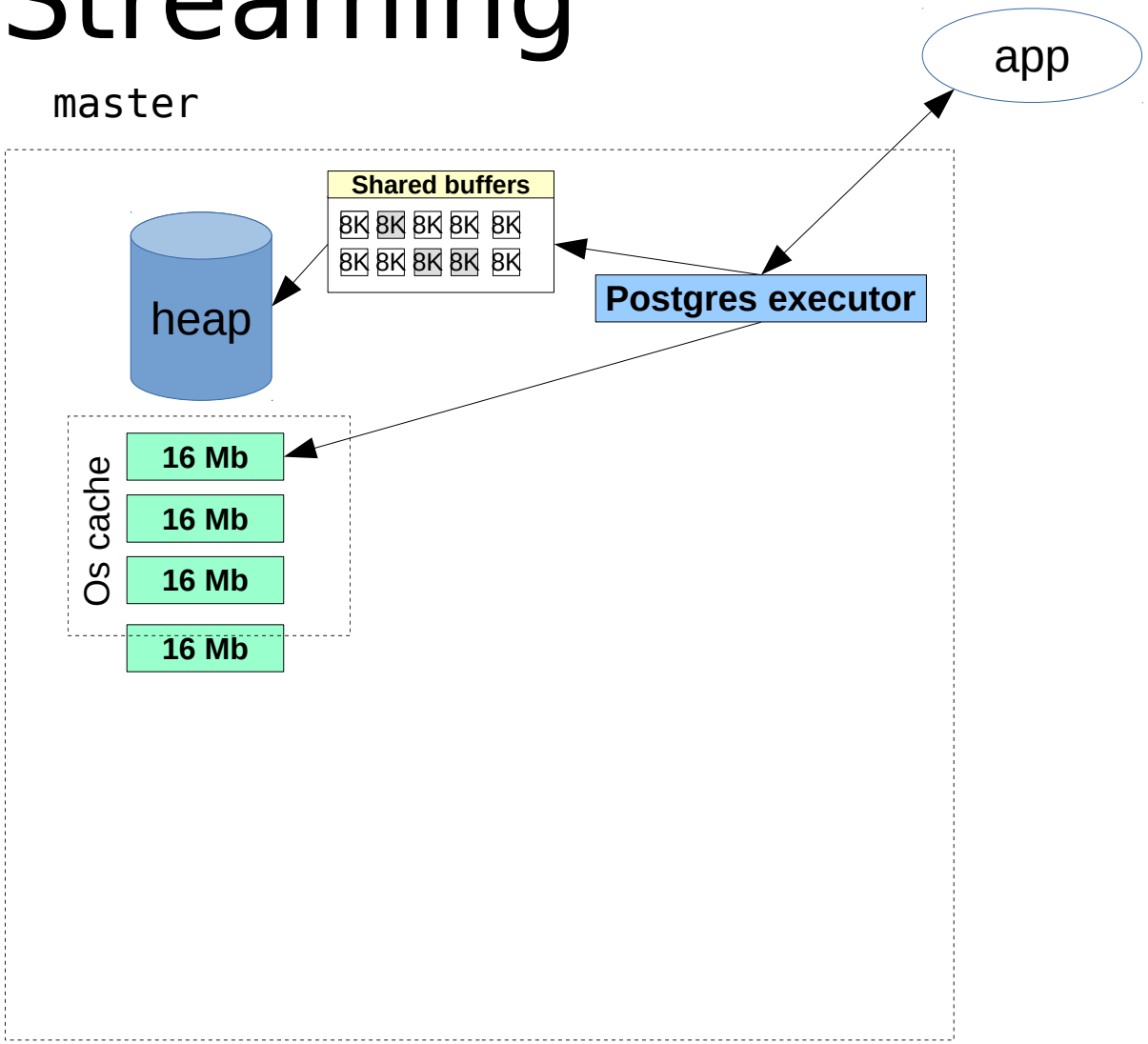
Avito

Streaming



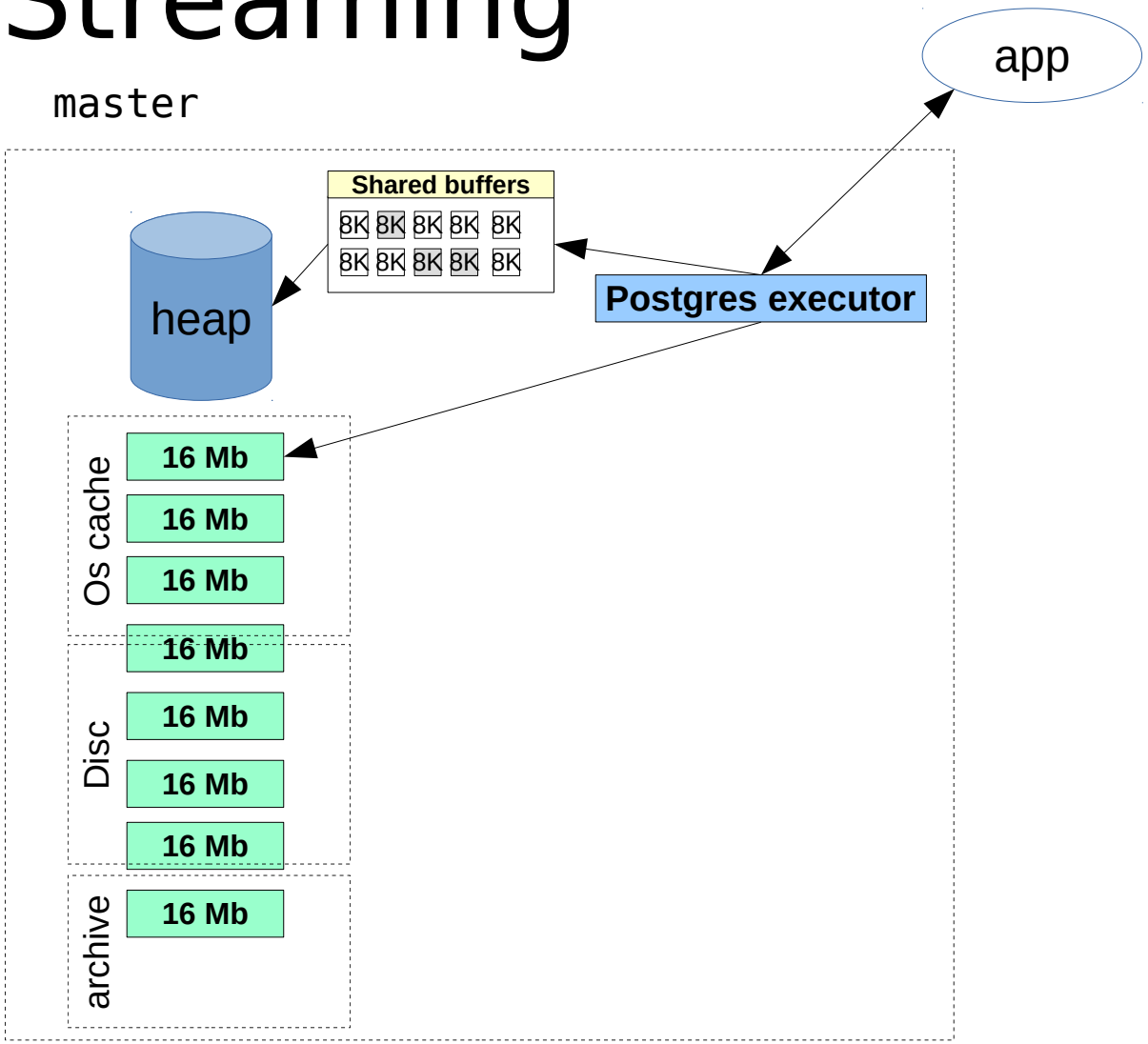
Avito

Streaming



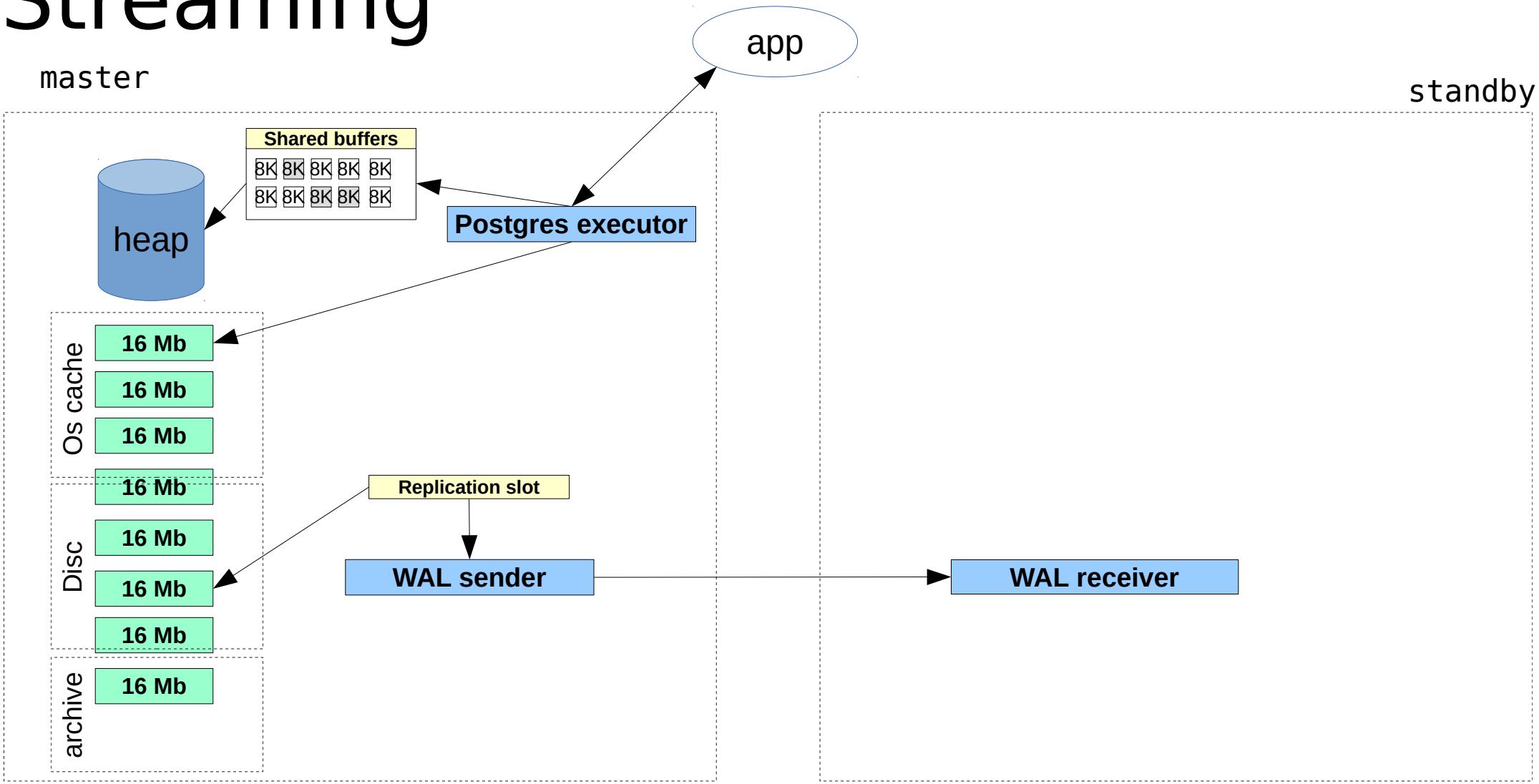
Avito

Streaming

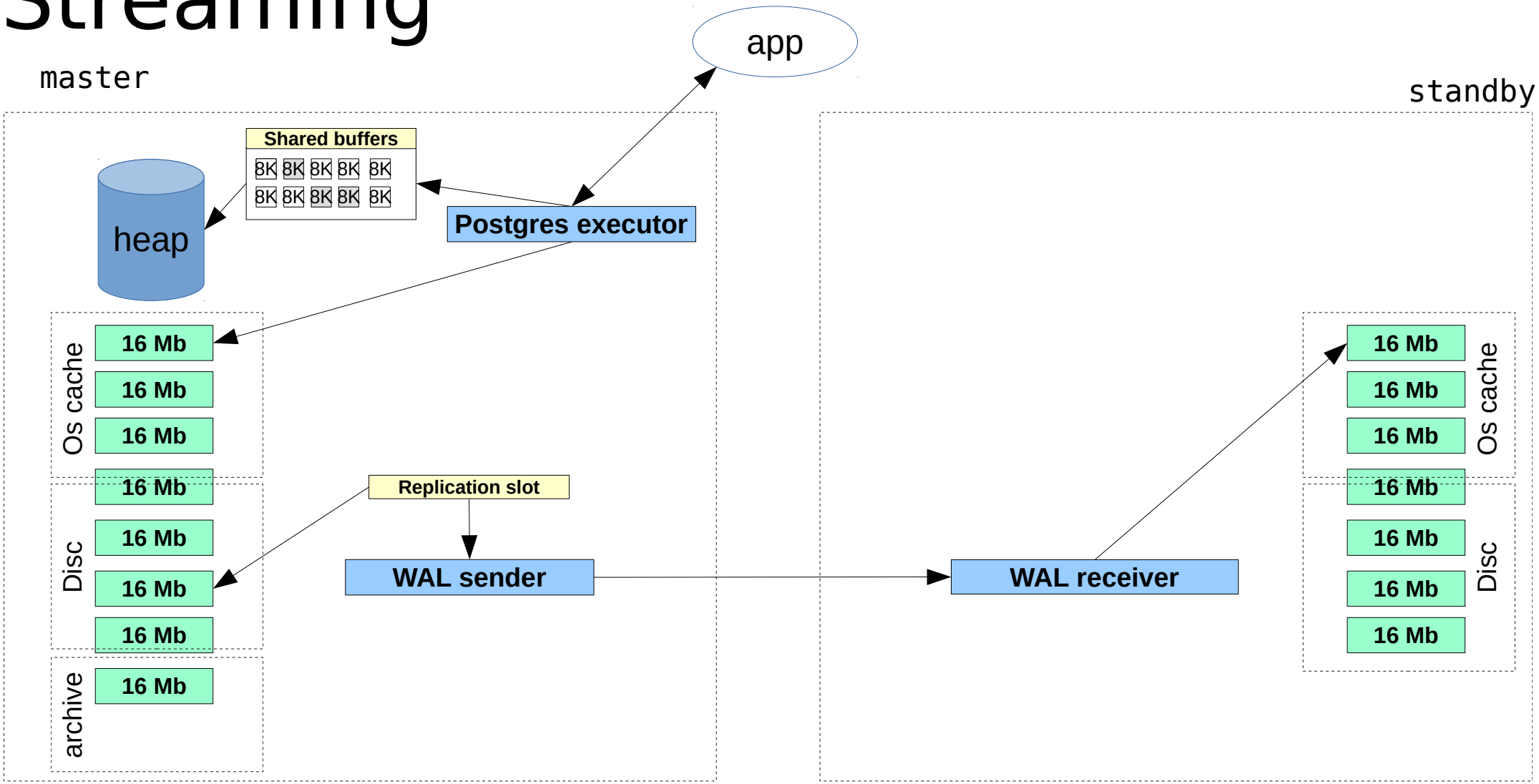


Avito

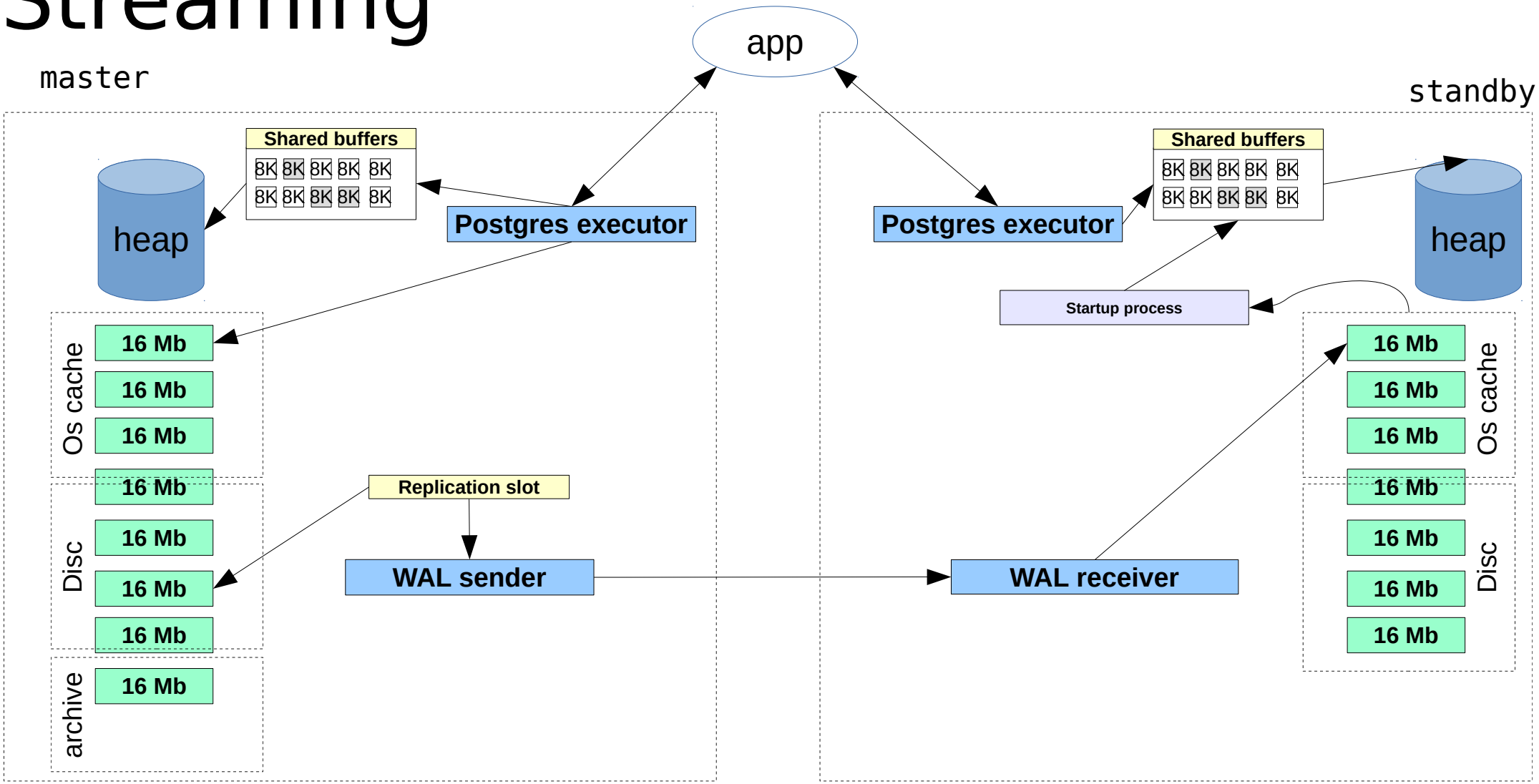
Streaming


app
master
standby
Shared buffers
8K 8K 8K 8K 8K
8K 8K 8K 8K 8K
heap
Postgres executor
Os cache
16 Mb
16 Mb
16 Mb
16 Mb
Disc
16 Mb
16 Mb
16 Mb
archive
16 Mb
Replication slot
WAL sender
WAL receiver

Streaming

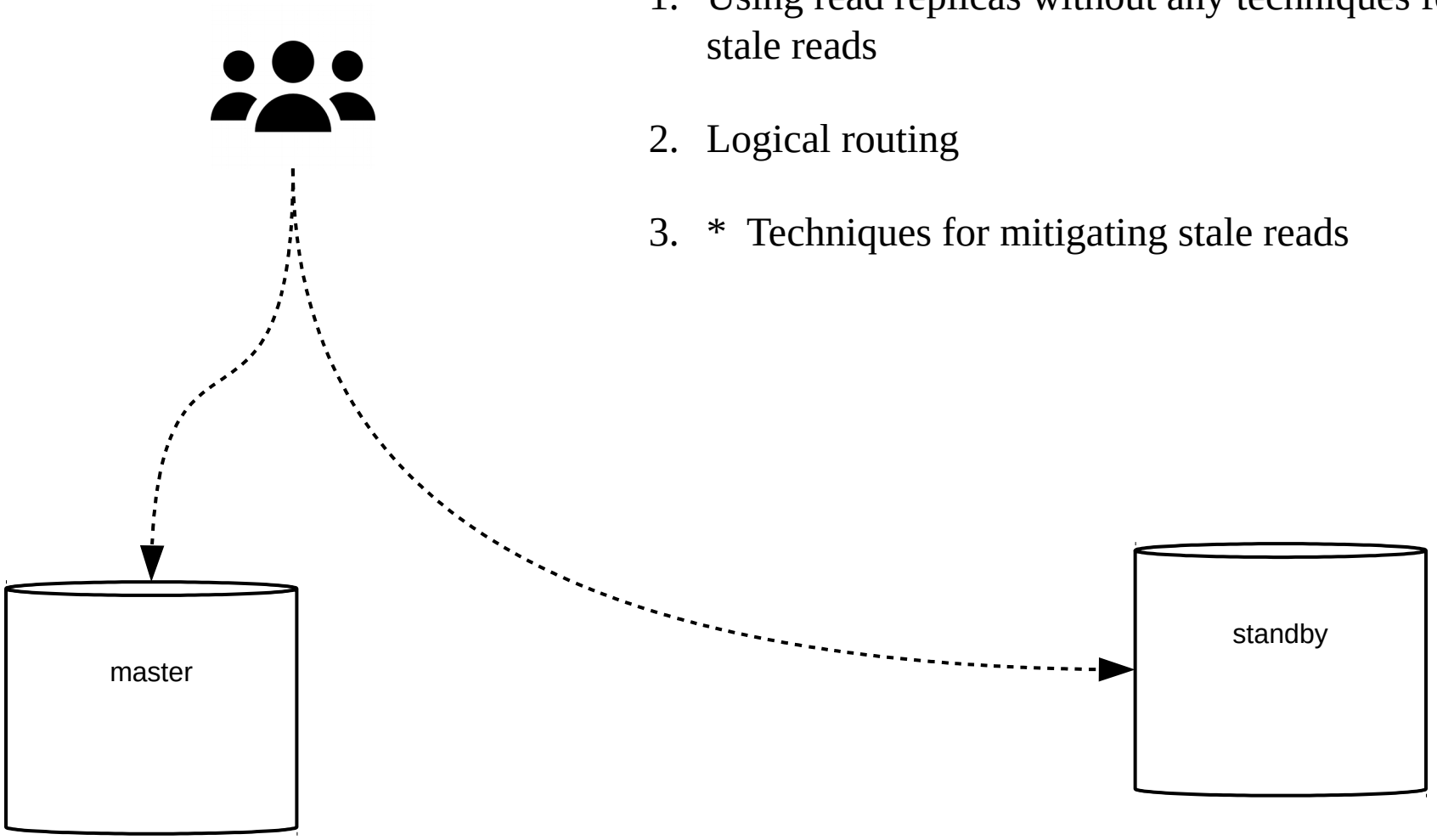

app
master
standby
Shared buffers
8K 8K 8K 8K 8K
8K 8K 8K 8K 8K
heap
Postgres executor
Os cache
16 Mb
16 Mb
16 Mb
16 Mb
Disc
16 Mb
16 Mb
16 Mb
archive
16 Mb
Replication slot
WAL sender
WAL receiver
16 Mb
16 Mb
16 Mb
Os cache
16 Mb
16 Mb
16 Mb
Disc

Streaming



Avito

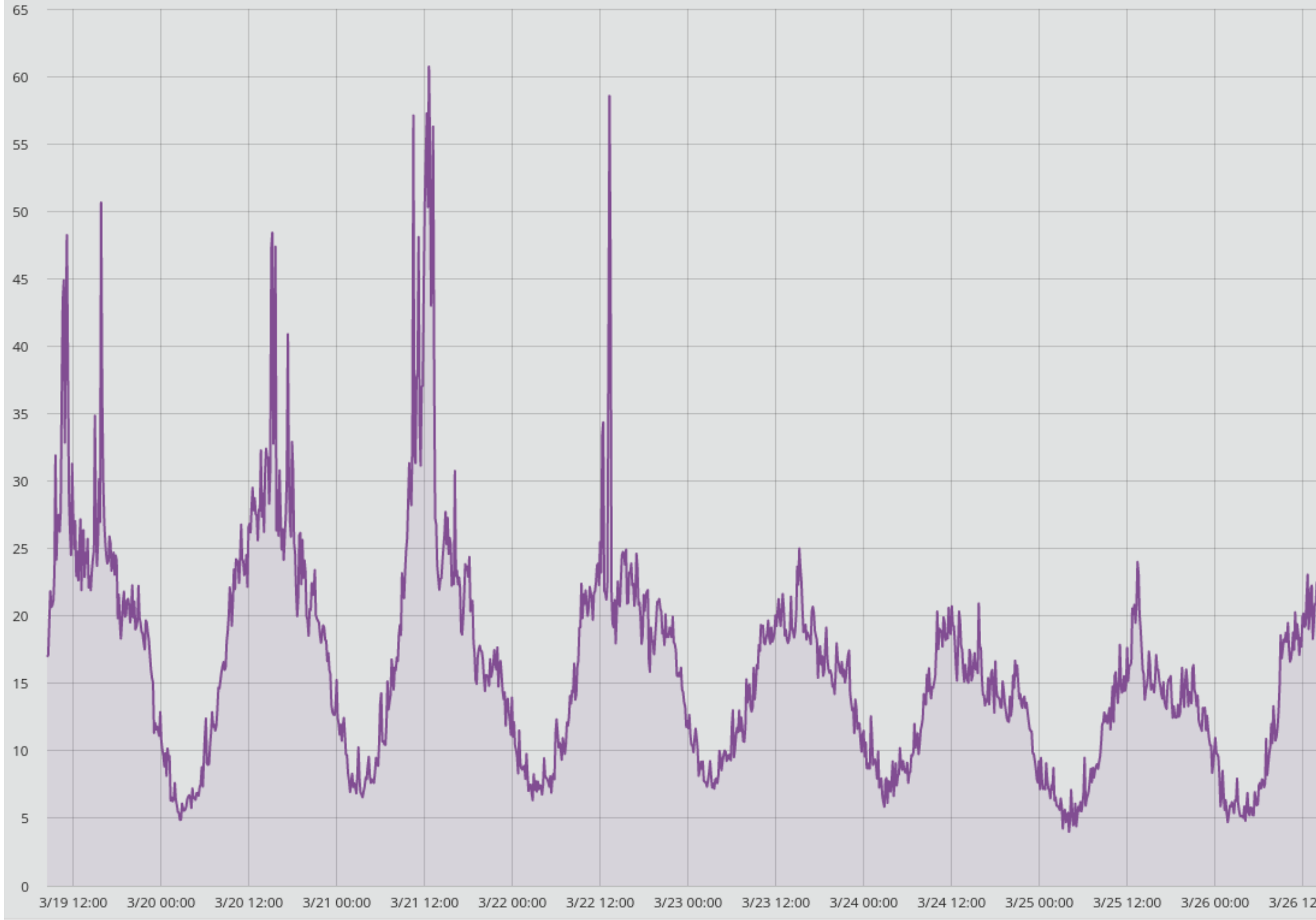
Scaling reads



1. Using read replicas without any techniques for mitigating stale reads
2. Logical routing
3. * Techniques for mitigating stale reads

Avito

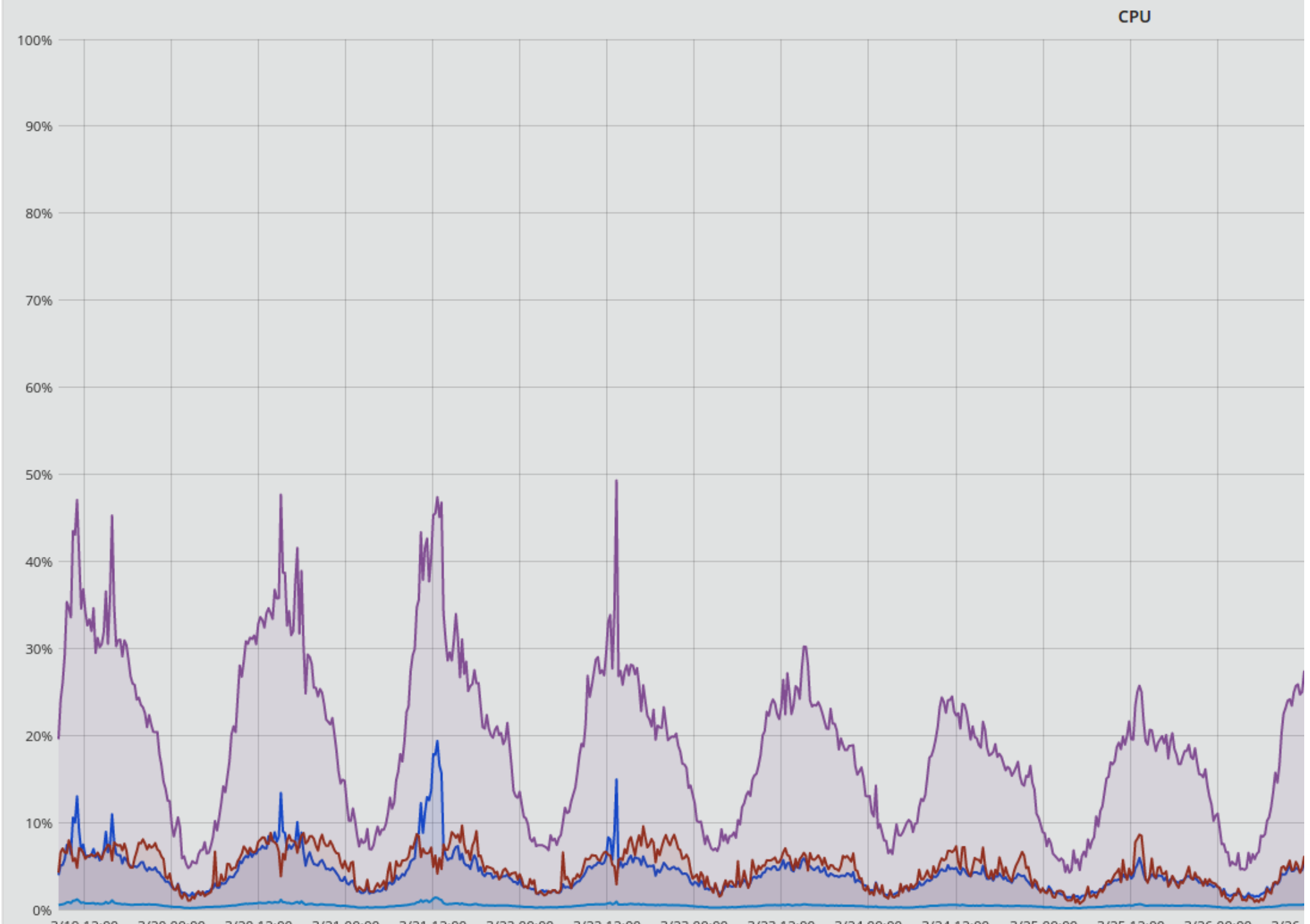
Load Average



** 28 physical cores*

Avito

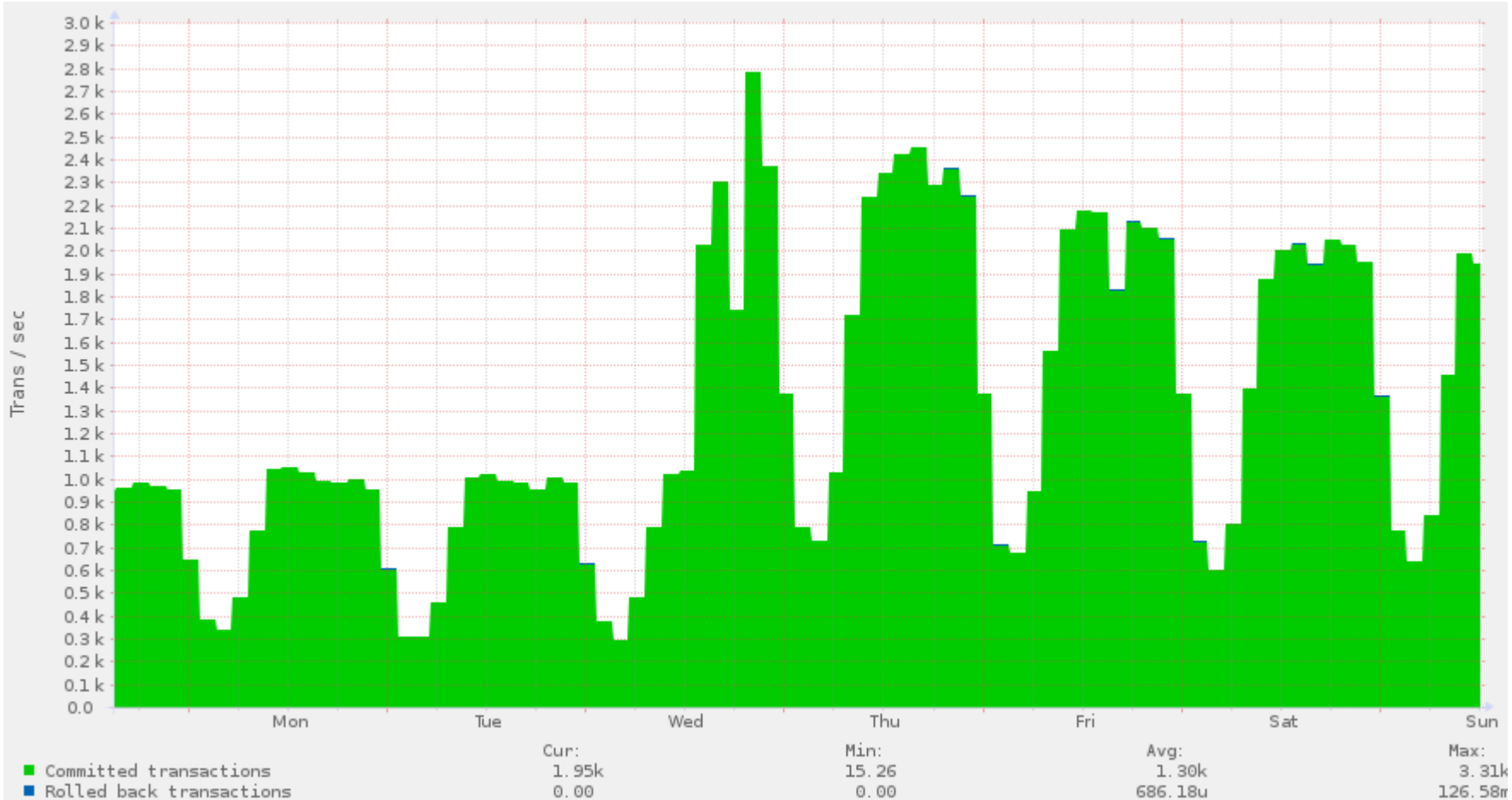
CPU



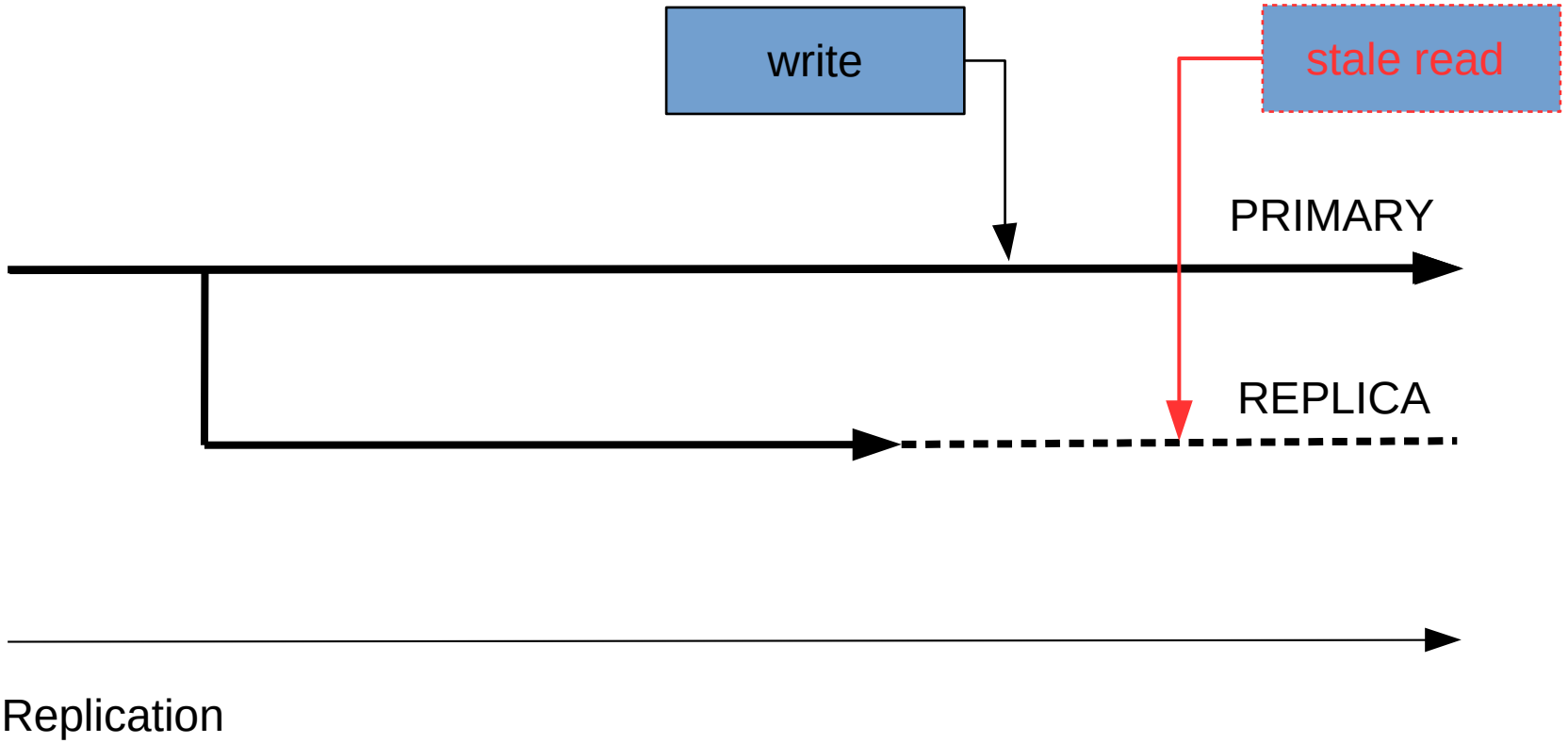
** 28 physical cores*

Avito

TPS


Trans / sec
3.0 k
2.9 k
2.8 k
2.7 k
2.6 k
2.5 k
2.4 k
2.3 k
2.2 k
2.1 k
2.0 k
1.9 k
1.8 k
1.7 k
1.6 k
1.5 k
1.4 k
1.3 k
1.2 k
1.1 k
1.0 k
0.9 k
0.8 k
0.7 k
0.6 k
0.5 k
0.4 k
0.3 k
0.2 k
0.1 k
0.0
Mon
Tue
Wed
Thu
Fri
Sat
Sun
Cur:
Min:
Avg:
Max:
Committed transactions
1.95k
15.26
1.30k
3.31k
Rolled back transactions
0.00
0.00
686.18u
126.58m

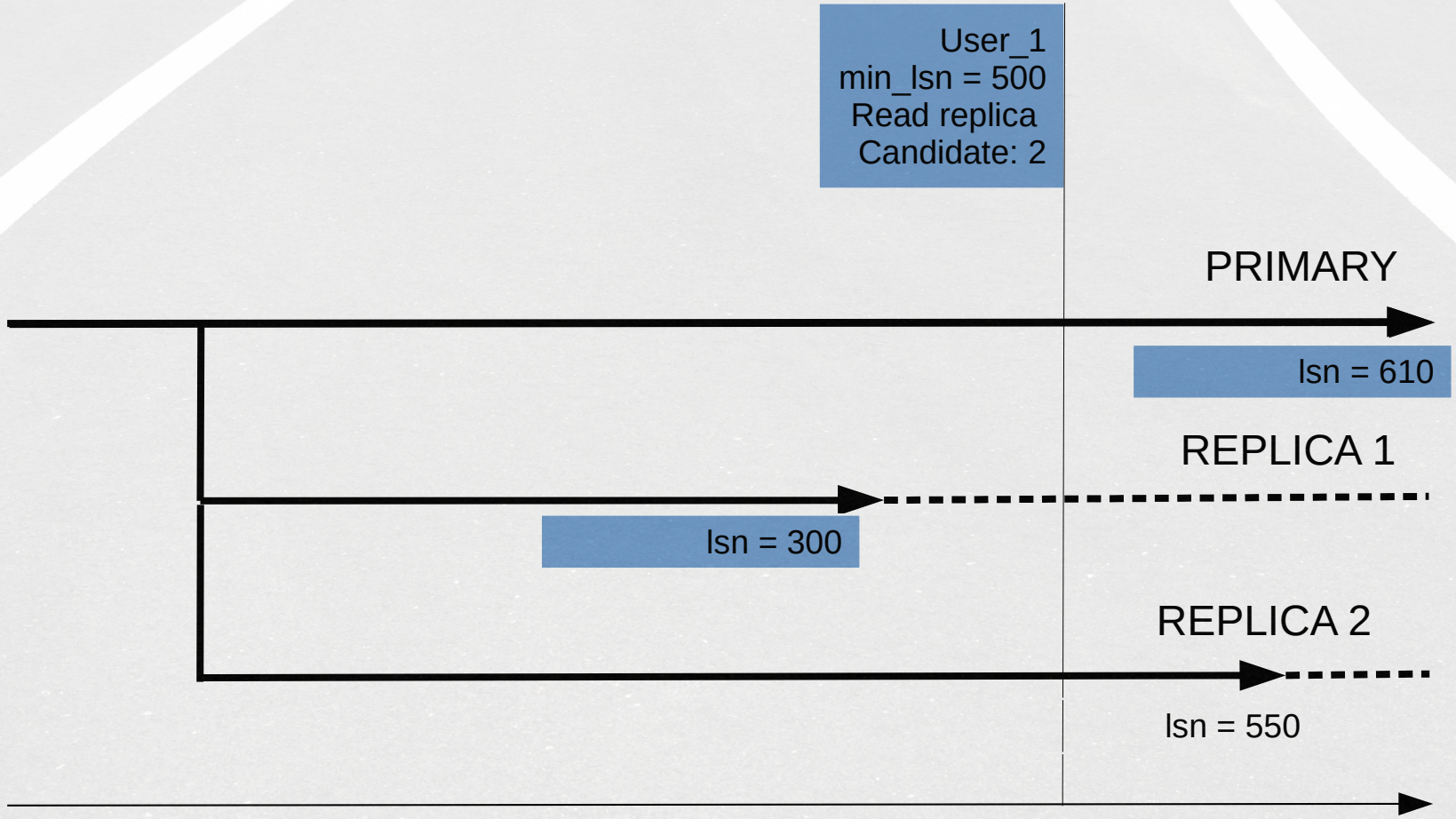
Stale Reads



Avito

https://brandur.org/postgres-reads
https://about.gitlab.com/2017/10/02/scaling-the-gitlab-database/#sticky-connections

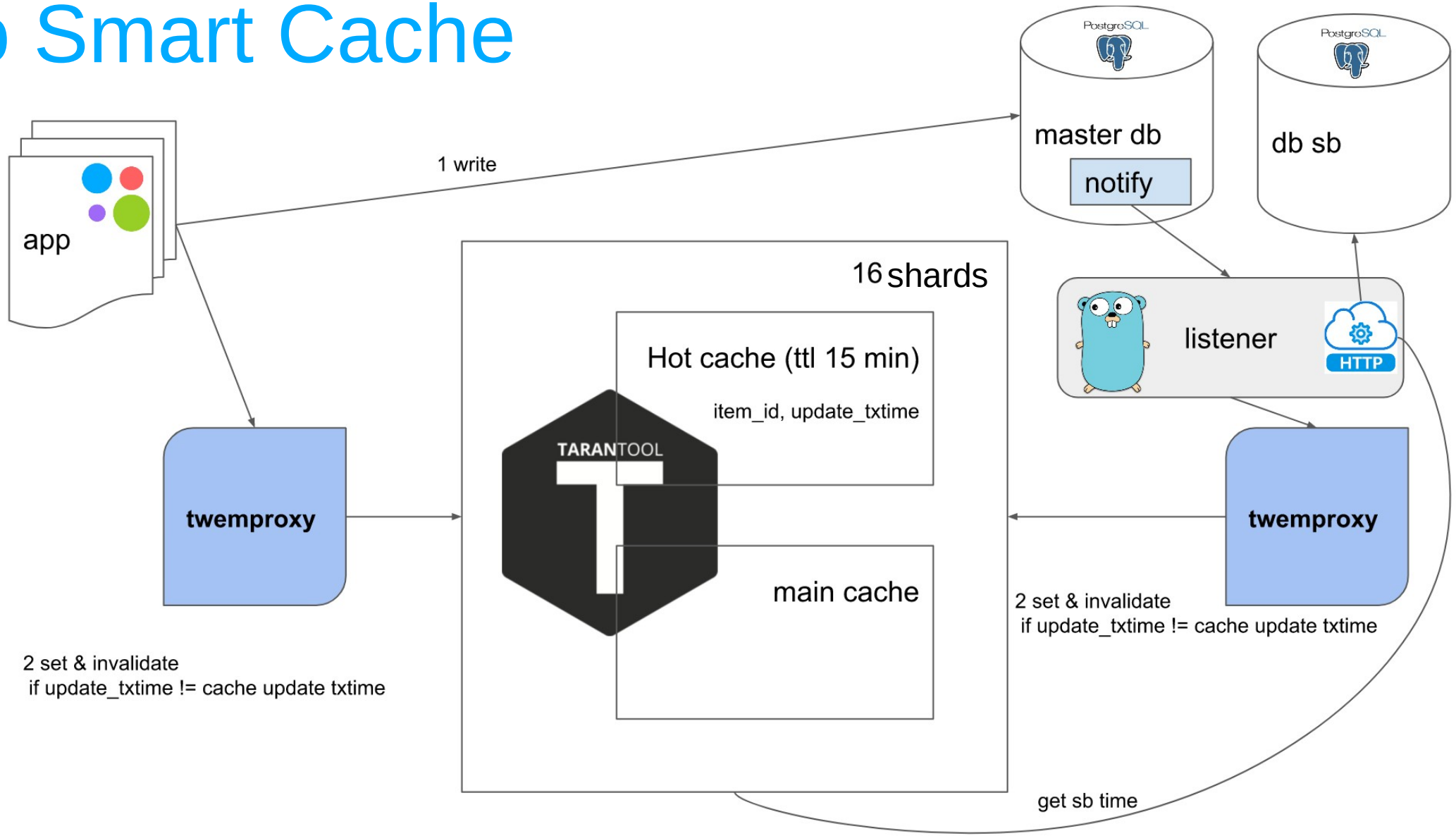
Routing



Avito

https://brandur.org/postgres-reads
https://about.gitlab.com/2017/10/02/scaling-the-gitlab-database/#sticky-connections

Avito Smart Cache



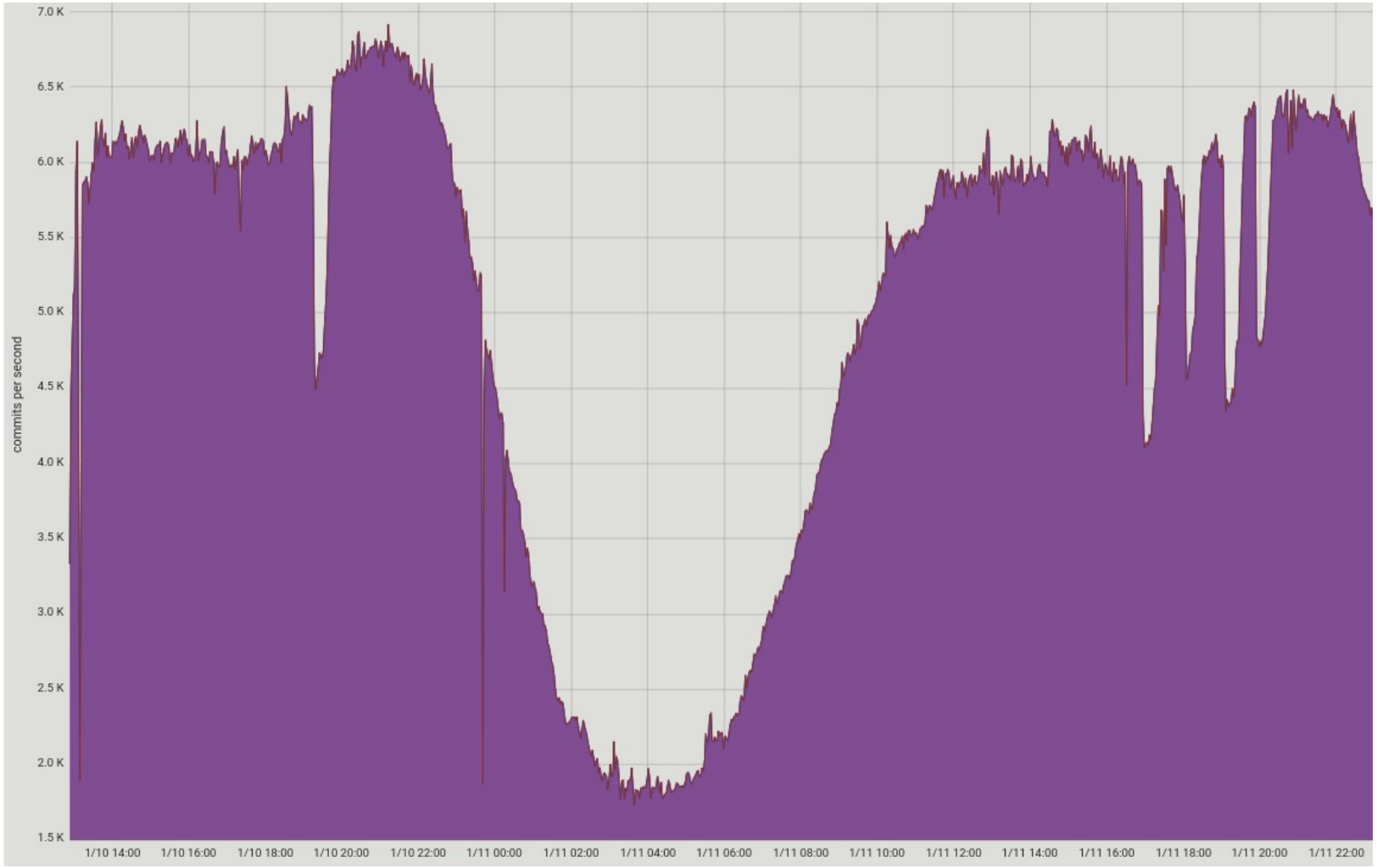
Avito

Two levels of cache

```
app.get_item(key):
        data = main_cache.get(key) // try to get data from cache

        if found then return

        hot_cache.get(key)          // get data from hot level of cache

        if found then               // if data was recently changed then route to master
            data =db_master.get_item(key)
            main_cache.set(key, data) // ttl 1 hour
        else if sb too old then     // if standby is falling behind – route to master
                data =db_master.get_item(key)
            main_cache.set(key, data) // ttl 24 hours
        else                        // in other cases we can route to standby
            data =db_slave.get_item(key)
            main_cache.set(key, data) // ttl 24 hours
        end if
```

Avito

Everything seems fine but ...



Avito

Pitfalls

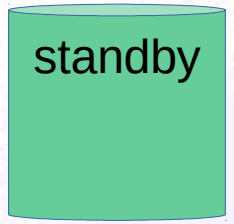
(1) Deadlock on standby

(2) DDL (statement_timeout and deadlock_timeout)

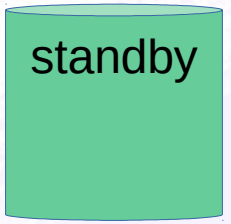
(3) Vacuum replaying on standby and truncating data file

(4) Restoring WAL from archive

Avito



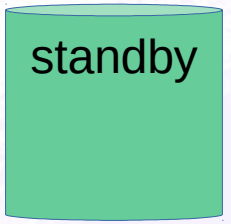

```
create table items(item_id int);
create table options(item_id int, v1 text);
```




```
create table items(item_id int);
create table options(item_id int, v1 text);
```

Begin;

alter table options add v2 int;



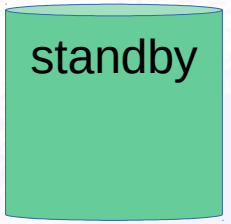

```
create table items(item_id int);
create table options(item_id int, v1 text);
```

```
Begin;
```

```
alter table options add v2 int;
```

```
Begin;
```

```
select * from items;
```



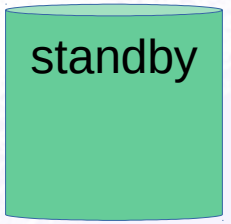

```
create table items(item_id int);
create table options(item_id int, v1 text);
```

```
Begin;

alter table options add v2 int;
```

```
Begin;

select * from items;
```

```
alter table items add a text;
```

Avito




```
create table items(item_id int);
create table options(item_id int, v1 text);
```

```
Begin;

alter table options add v2 int;

alter table items add a text;
```

```
Begin;

select * from items;

select * from options;
--DEADLOCK is not detected
```

Avito

PostgreSQL 10

```
ERROR:  deadlock detected
LINE 1: select * from options;
                      ^
DETAIL:  Process 25364 waits for
AccessShareLock on relation 10000
of database 9000; blocked by
process 25322.
Process 25322 waits for
AccessExclusiveLock on relation
10000 of database 9000; blocked
by process 25364.
HINT:  See server log for query
details.
```

25322 is the PID of the apply process

Avito

Pitfalls

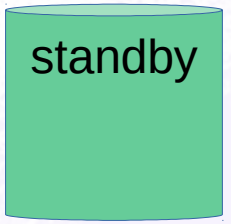
(1) Deadlock on standby

(2) DDL (statement_timeout and deadlock_timeout)

(3) Vacuum replaying on standby and truncating data file

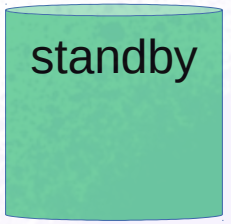
(4) Restoring WAL from archive

Avito




```
alter table options add v2 int;
statement_timeout + deadlock_timeout
```

Avito




```
alter table options add v2 int;
statement_timeout + deadlock_timeout
```

```
select * from options;
….
select * from options;
```

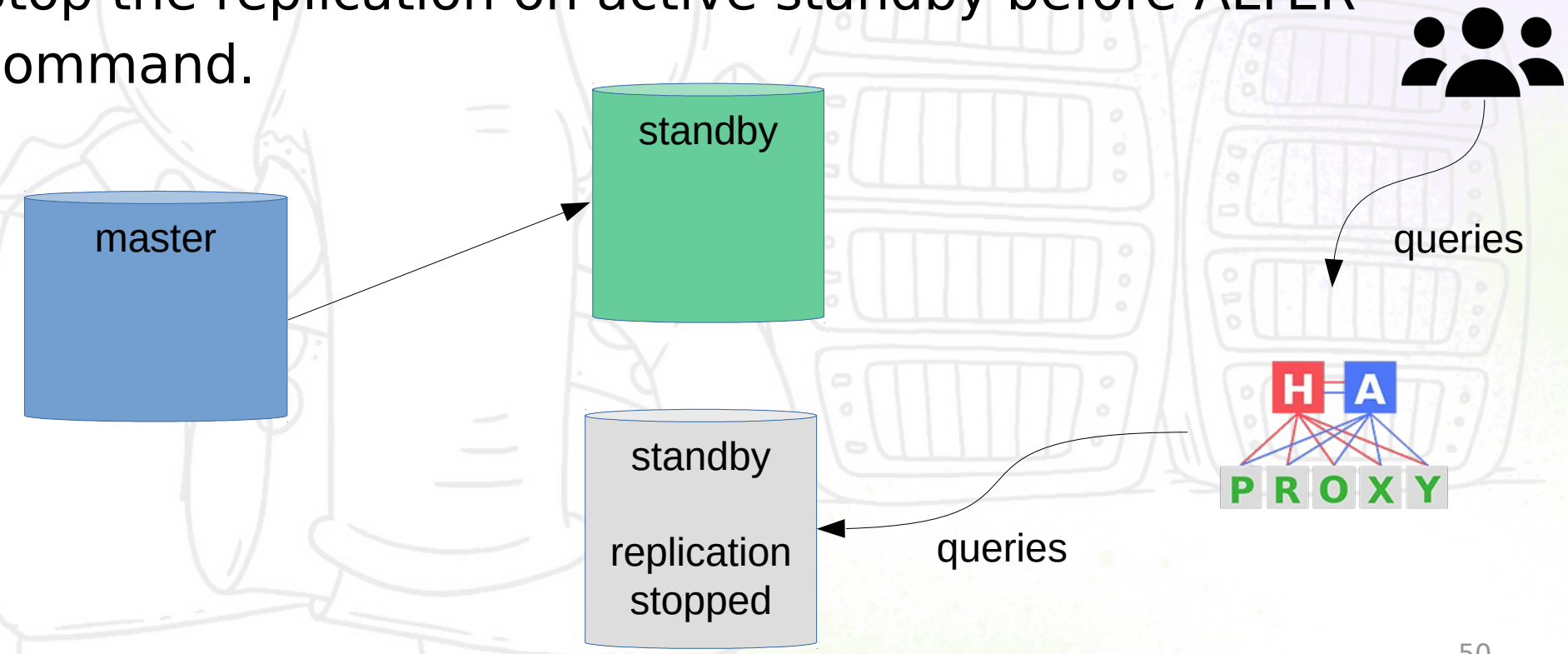
```
2018-01-12 16:54:40.208 MSK
pid=20949,user=user_3,db=test,hos
t=127.0.0.1:55763 LOG: process
20949 still waiting for
AccessShareLock on relation 10000
of database 9000 after 5000.055
ms
2018-01-12 16:54:40.208 MSK
pid=20949,user=user_3,db=test,hos
t=127.0.0.1:55763 DETAIL: Process
holding the lock: 46091. Wait
queue: 18639, 20949, 53445,
20770, 10799, 47217, 37659, 6727,
37662, 25742, 20771,
```

Avito

(2) DDL (statement_timeout and deadlock_timeout)

Script for HAProxy to implement external control (switching your traffic from all nodes)

Stop the replication on active standby before ALTER command.

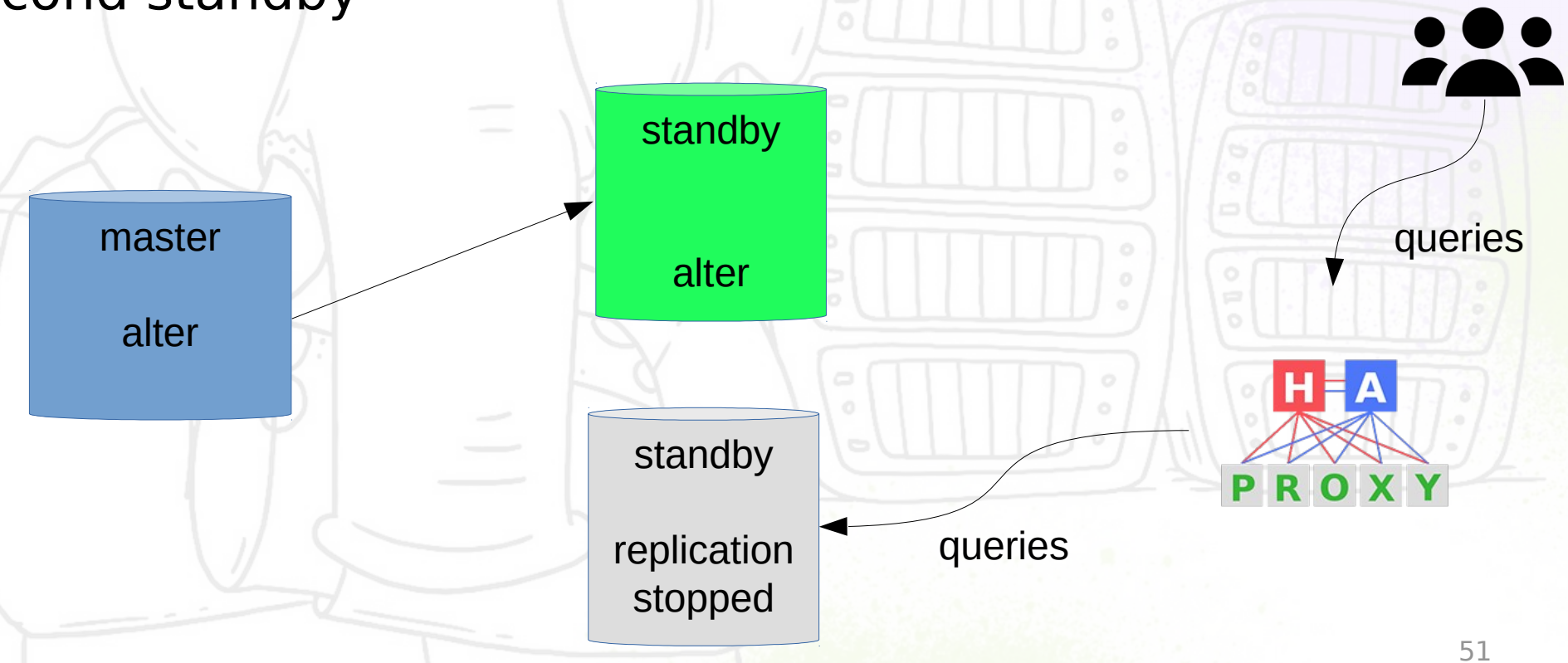


Avito

(2) DDL (statement_timeout and deadlock_timeout)

Run ALTER command

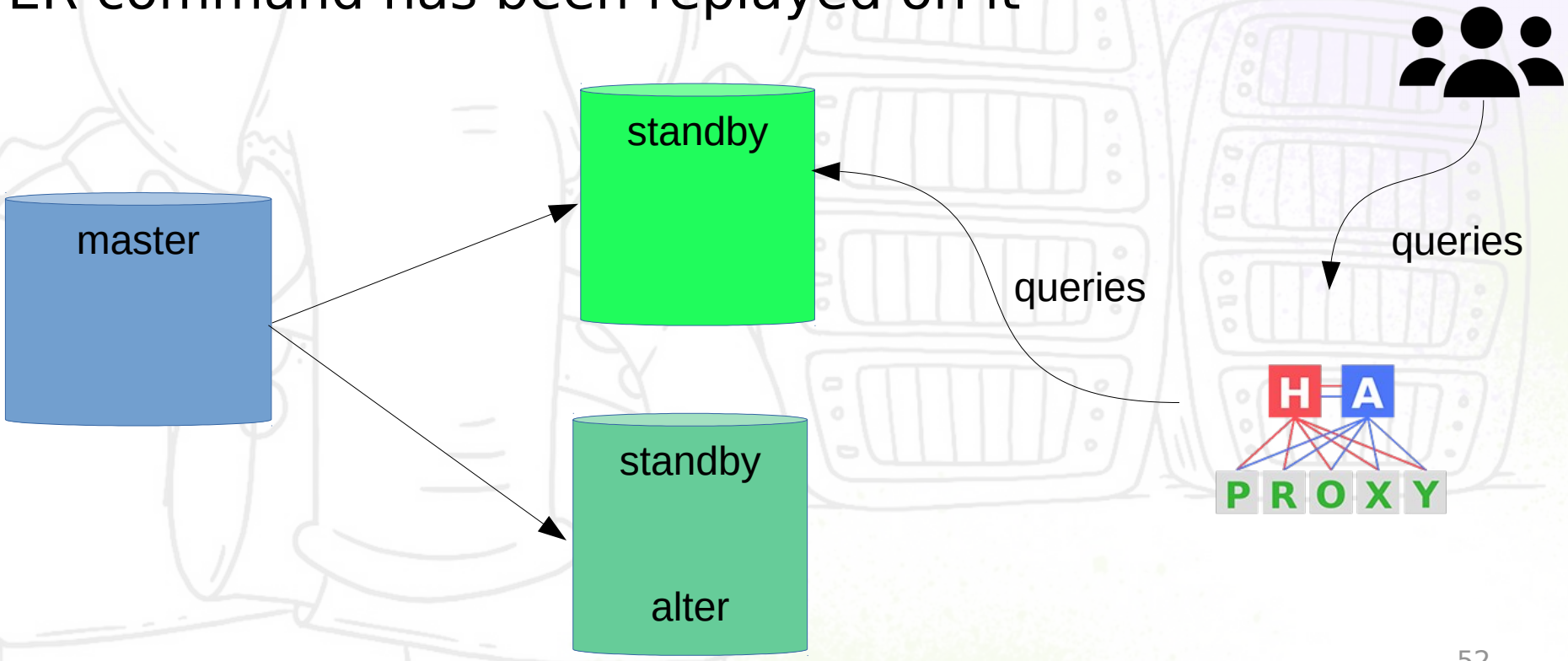
Wait till the ALTER command has been replayed on the second standby



(2) DDL (statement_timeout and deadlock_timeout)

Switch traffic on the second standby

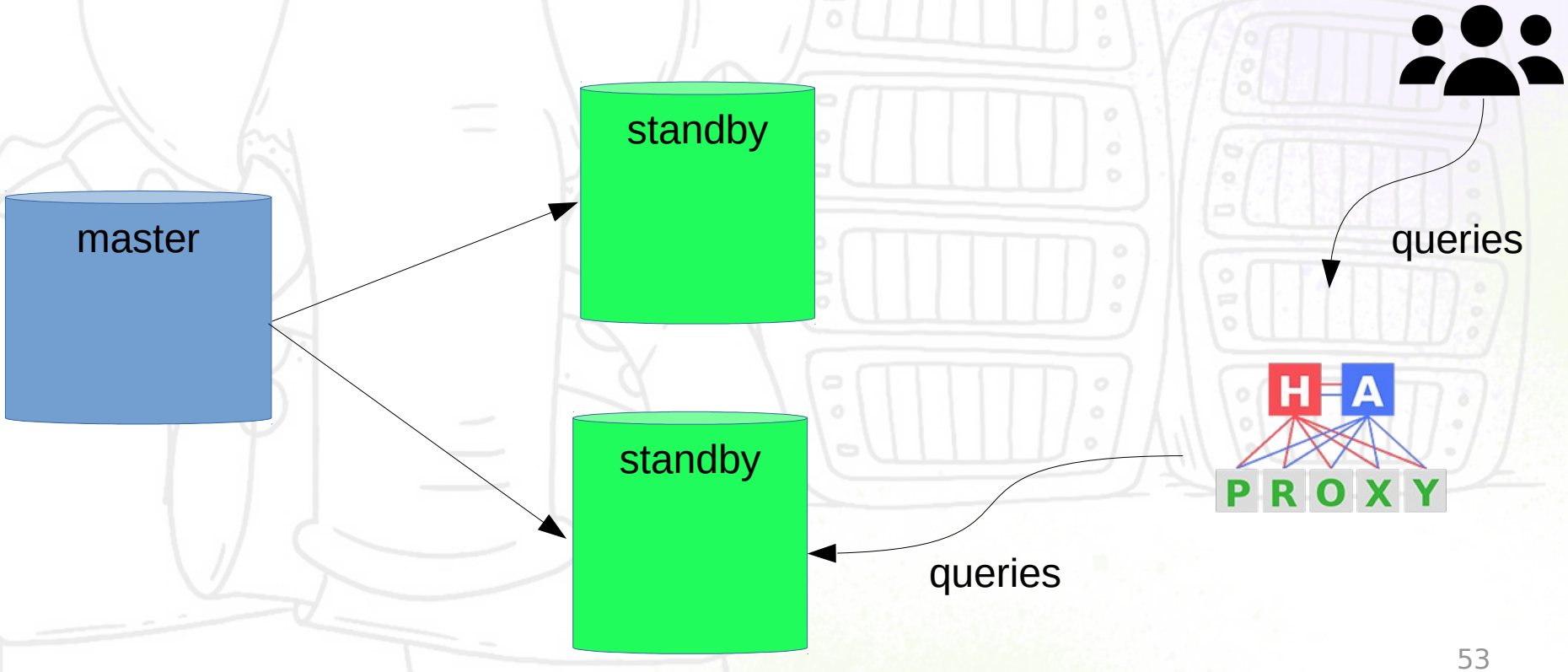
Start replication on the first standby and wait till the ALTER command has been replayed on it



Avito

(2) DDL (statement_timeout and deadlock_timeout)

Return the first standby to the pool of active standbys



Avito

Pitfalls

(1) Deadlock on standby

(2) DDL (statement_timeout and deadlock_timeout)

(3) Vacuum replaying on standby and truncating data file

(4) Restoring WAL from archive

Avito

(3) Vacuum replaying on standby and truncating data file

- Vacuum can truncate the end of data file — the exclusive lock is needed for this action. At this moment on the standby long locks between read only queries and recovery process occur
- It happens because some unlock actions are not written to WAL .
- On next slide you can see few AccessExclusive locks in one xid 920764691, and not a single unlock...
- Unlock happens much later. When standby replays commit

Avito

(3) Vacuum replaying on standby and truncating data file

```
tx: 920764691, lsn: 73CE0/10605980, desc: AccessExclusive locks: xid 920764691 db 16445 rel 3326466
tx: 920764691, lsn: 73CE0/10694568, desc: file truncate: base/16445/3326466 to 1965248 blocks
tx: 920764691, lsn: 73CE0/1105AB98, desc: AccessExclusive locks: xid 920764691 db 16445 rel 3326466
tx: 920764691, lsn: 73CE0/11116A88, desc: file truncate: base/16445/3326466 to 1965152 blocks
tx: 920764691, lsn: 73CE0/116C89C0, desc: AccessExclusive locks: xid 920764691 db 16445 rel 3326466
tx: 920764691, lsn: 73CE0/117211E0, desc: file truncate: base/16445/3326466 to 1965088 blocks
tx: 920764691, lsn: 73CE0/128DFF00, desc: AccessExclusive locks: xid 920764691 db 16445 rel 3326466
tx: 920764691, lsn: 73CE0/129A5DD0, desc: file truncate: base/16445/3326466 to 1964960 blocks
tx: 920764691, lsn: 73CE0/1315C4E8, desc: AccessExclusive locks: xid 920764691 db 16445 rel 3326466
tx: 920764691, lsn: 73CE0/134CF9E0, desc: file truncate: base/16445/3326466 to 1964832 blocks
```

In our example there is 75-WAL-file interval between first lock and success truncate (unlock relation)

Avito

(3) Vacuum replaying on standby and truncating data file

The solution can be like:

* https://commitfest.postgresql.org/22/1981/

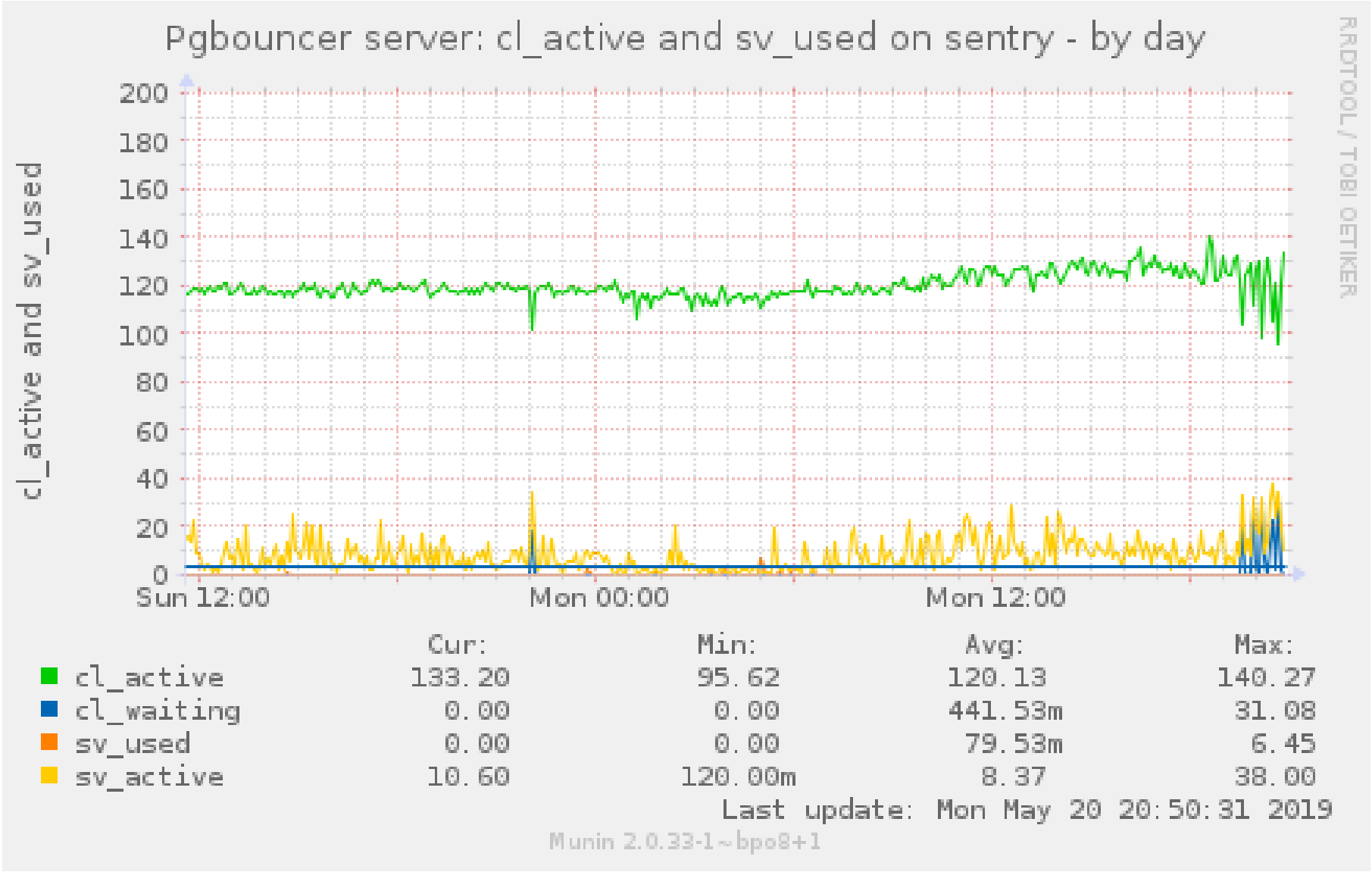
  Skip table truncation at VACUUM is coming in PostgreSQL 12

* Decrease the number of locks on standby?

  (Postgres Professional)

Avito

(3) Vacuum and truncating data file on primary



https://sentry.io

Avito

Pitfalls

(1) Deadlock on standby

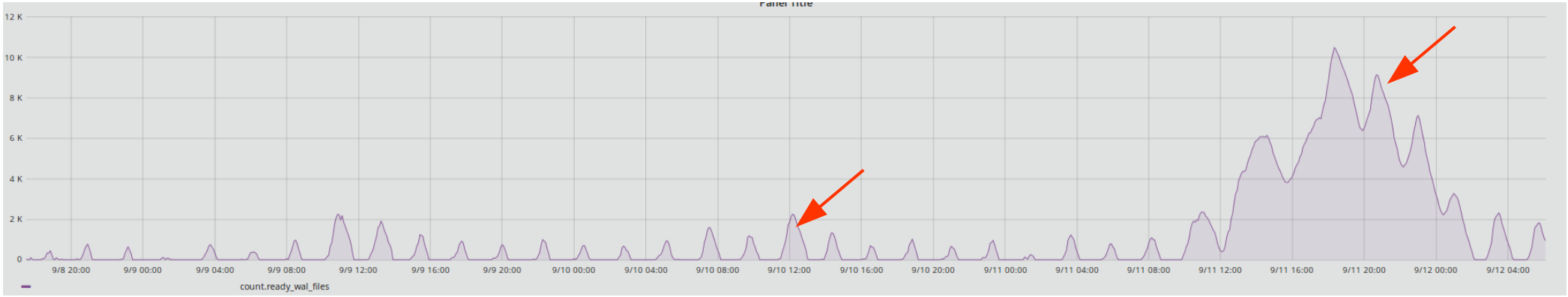
(2) DDL (statement_timeout and deadlock_timeout)

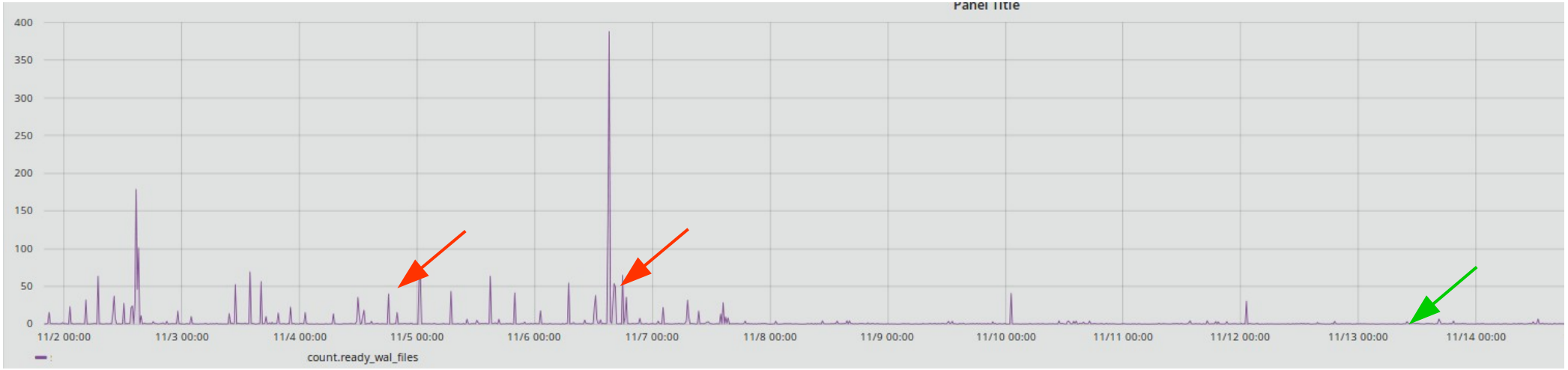
(3) Vacuum replaying on standby and truncating data file

(4) Restoring WAL from archive

Avito

More and more WAL





Avito

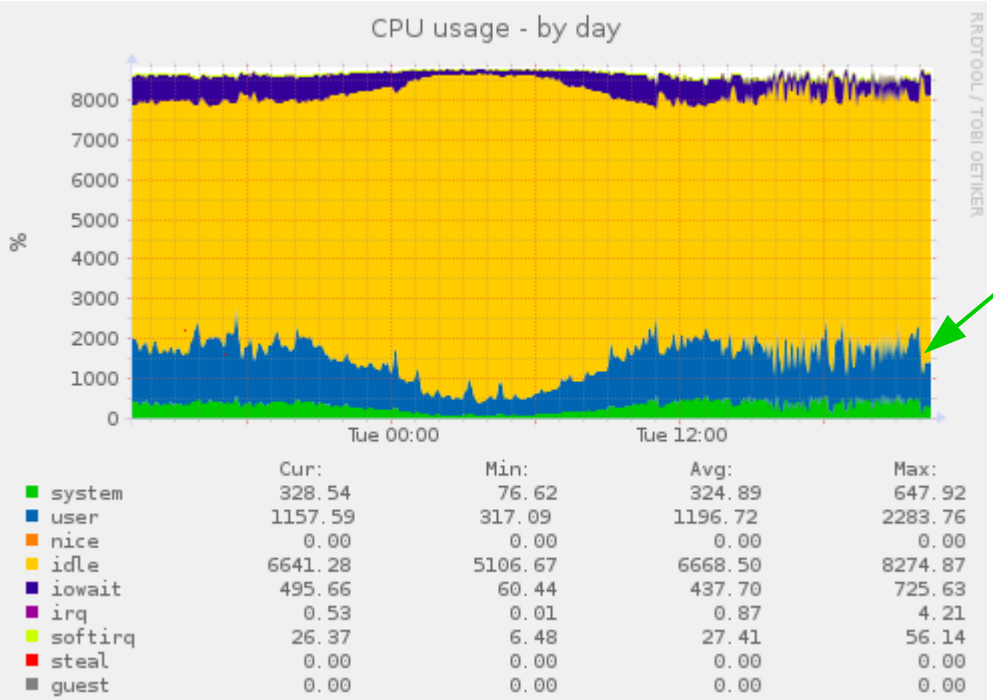
Avito archive 2016

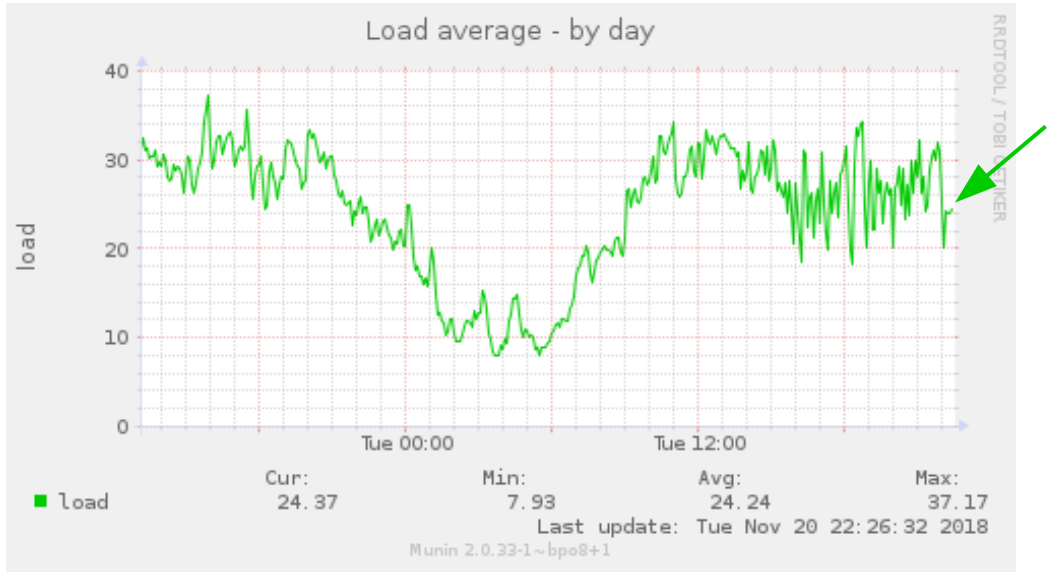
https://github.com/avito-tech/dba-utils/tree/master/pg_archive

```
archive_command = '/usr/local/bin/archive_cmd HOSTNAME /postgresql/walldir/logs.complete %p %f'

usage: archive_cmd DST-HOSTNAME DST-DIR SRC-WAL-FILENAME-WITH-PATH SRC-WAL-FILENAME

DST-HOSTNAME                - for scp

DST-DIR                     - archive directory for WALs

SRC-WAL-FILENAME-WITH-PATH - %p (file name with path)

SRC-WAL-FILENAME            - %f (file name)


# archive in one thread if

# - ready WAL files lower than threshold ready_wals_for_parallel
```

Avito

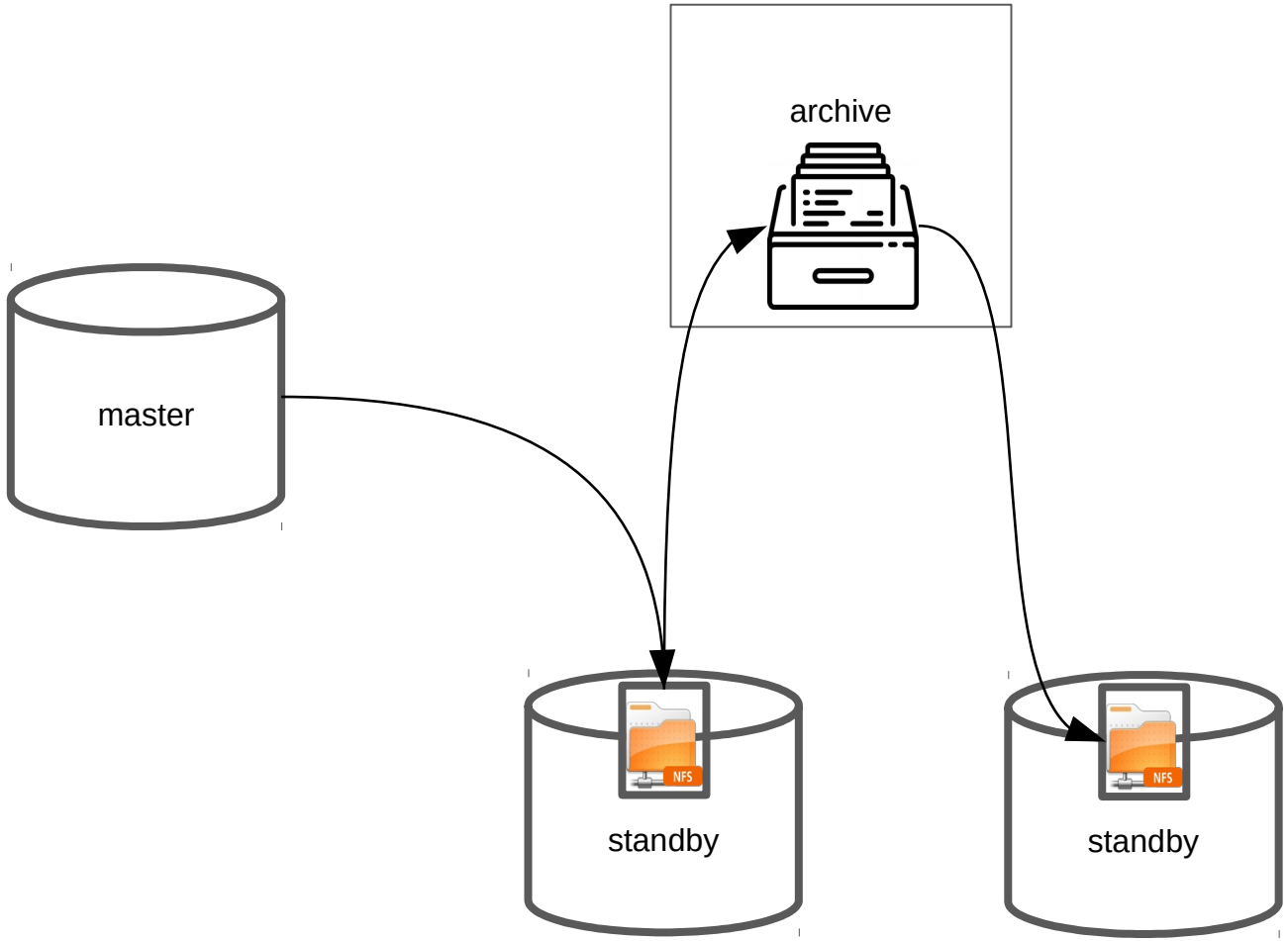
Archiving 1 WAL ~ 60ms





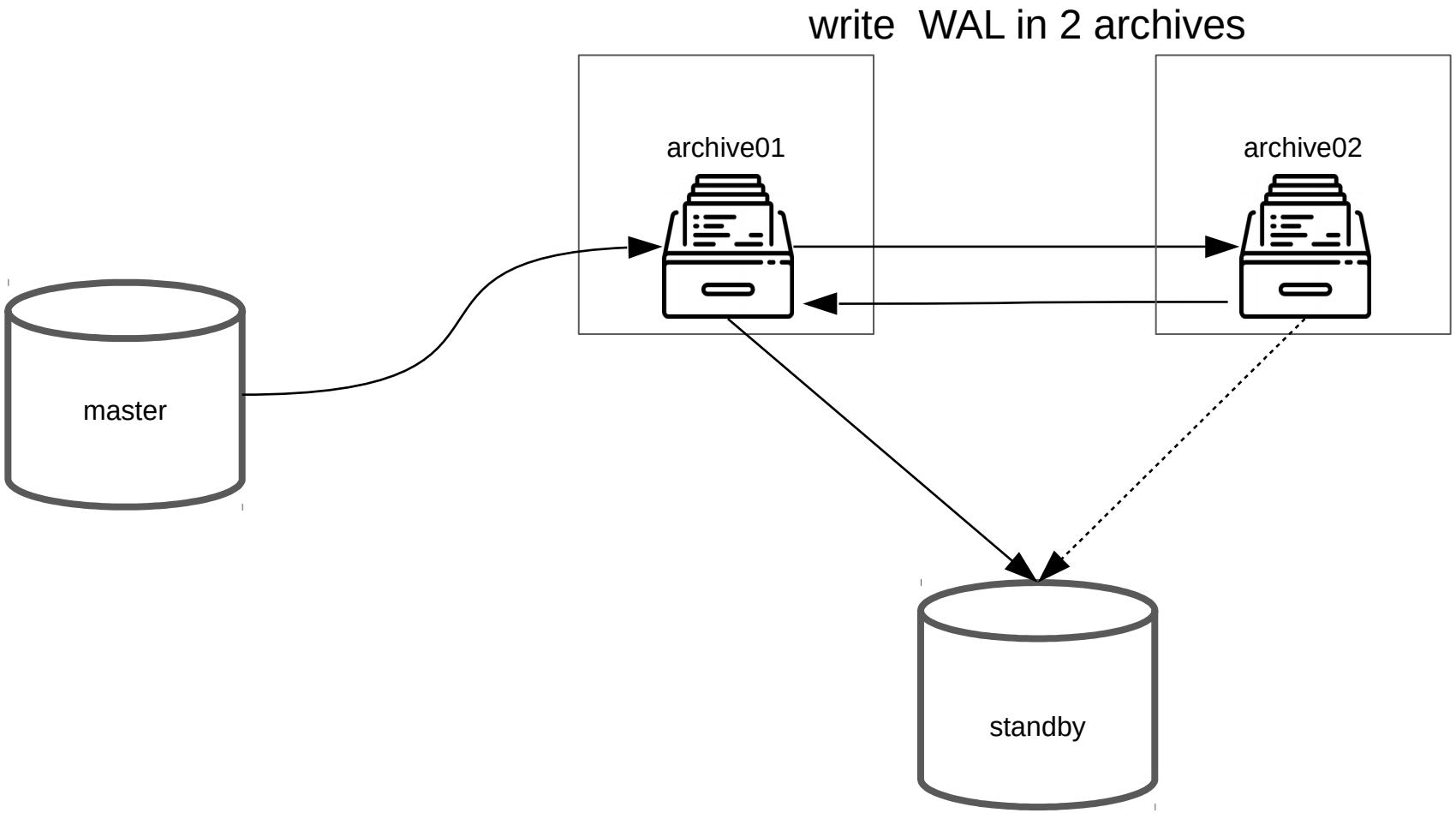
Avito

Old archive schema



New archive schema

https://github.com/avito-tech/dba-utils/tree/master/pg_archive2

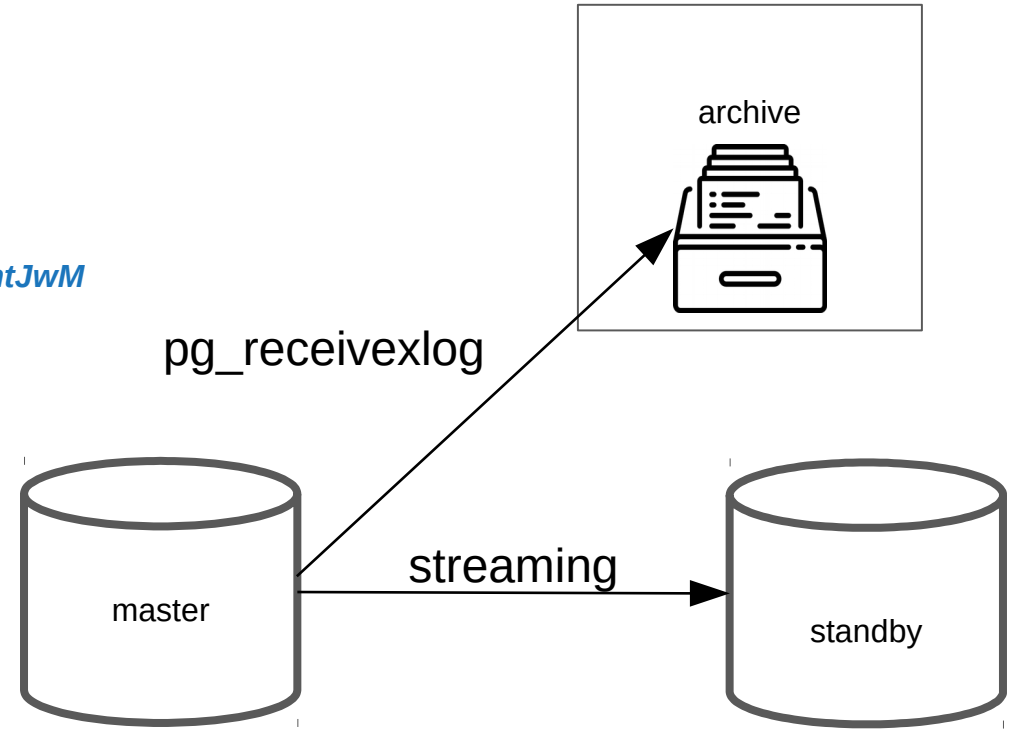


Avito

Streaming

WARM standby done right
Heikki Linnakangas

https://pgday.ru/ru/2015/papers/8
https://www.youtube.com/watch?v=mlQ90MntJwM

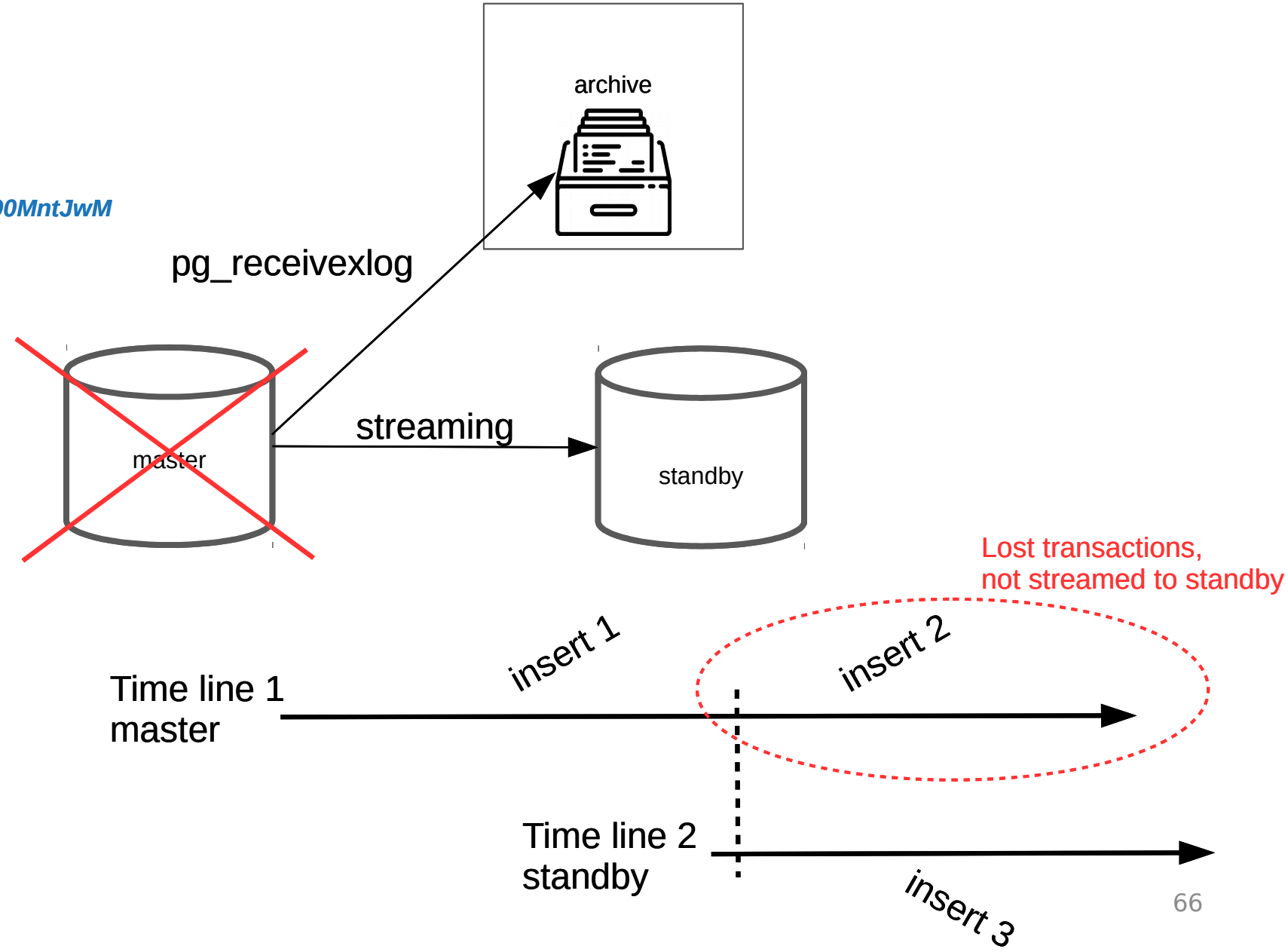


Avito

Streaming

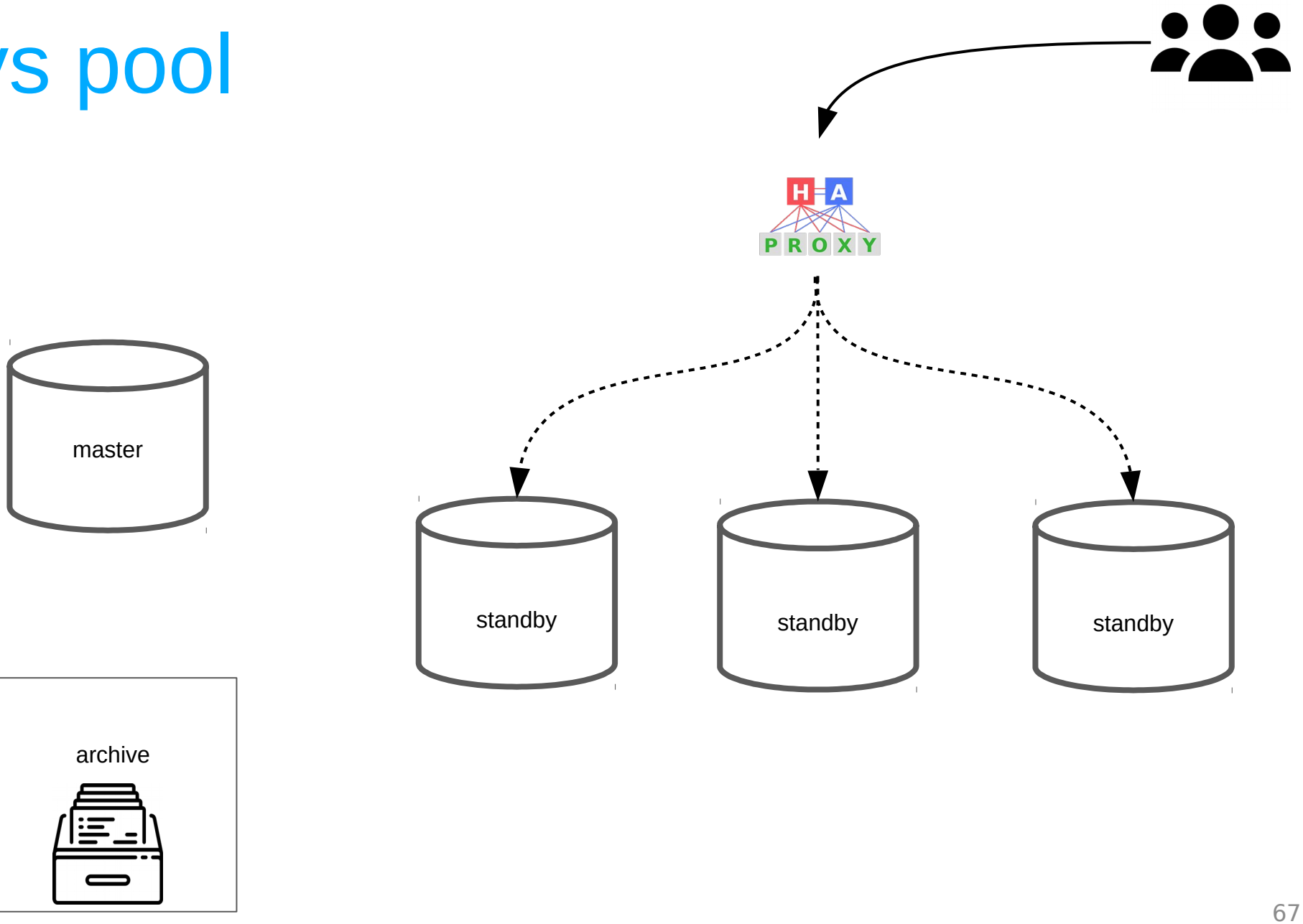
WARM standby done right
Heikki Linnakangas

https://pgday.ru/ru/2015/papers/8
https://www.youtube.com/watch?v=mlQ90MntJwM



Avito

Standbys pool



HAProxy check function

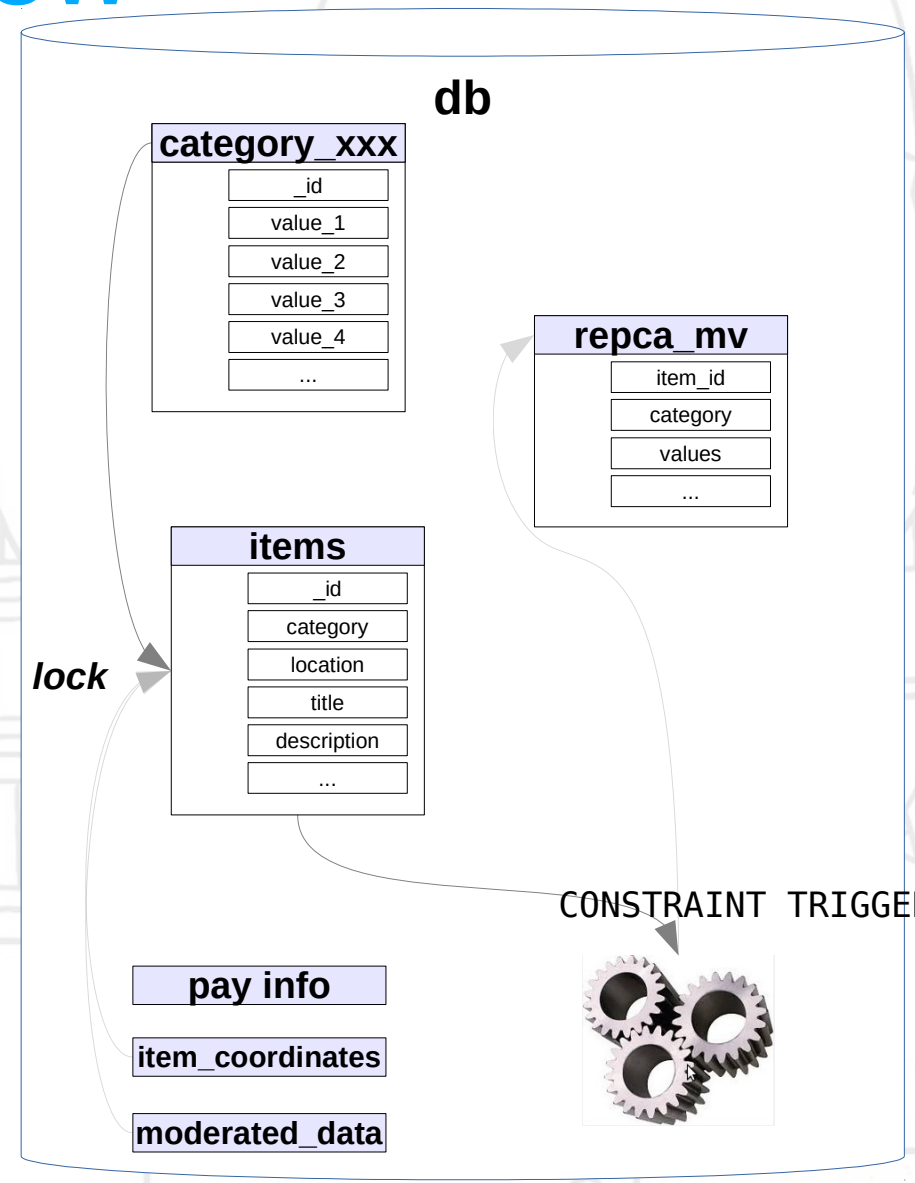
```
if master
     then false
if lag > max
     then create file and return false
if lag > min and file exists
     then return false
if lag < min and file exists
     then remove file and return true
else
     true
```


Avito

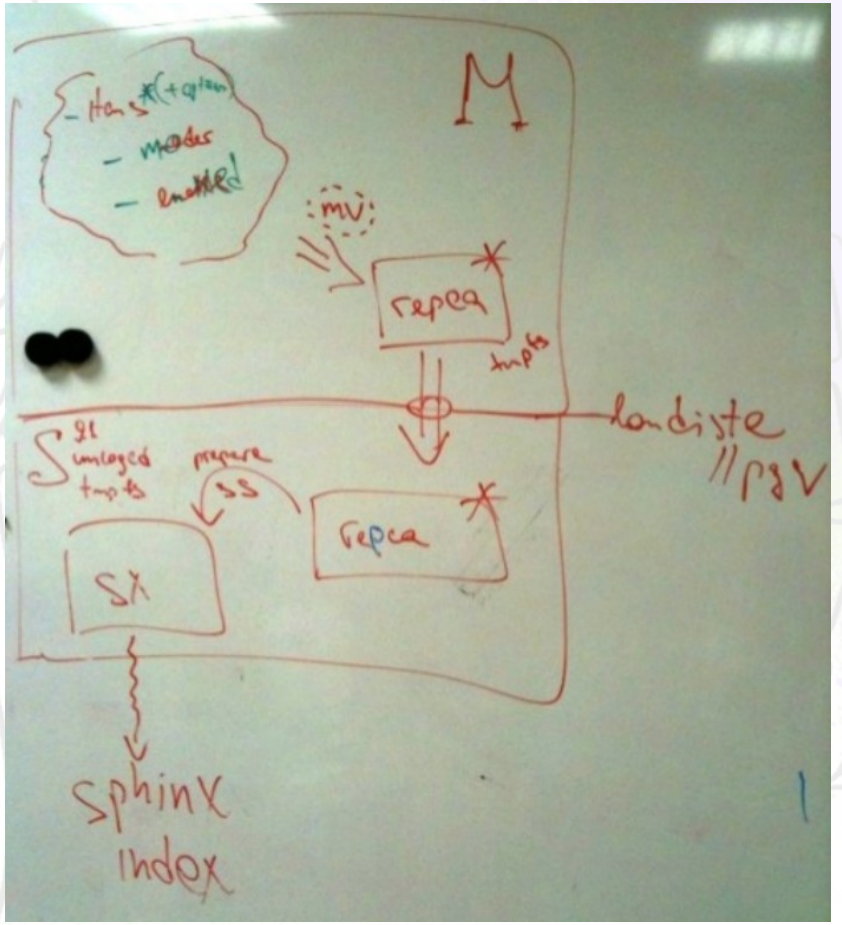
www.pgcon.org/2008/schedule/attachments/63_BSDCan2008-MaterializedViews.pdf

www.pgcon.org/2008/schedule/attachments/64_BSDCan2008-MaterializedViews-paper.pdf

MatView

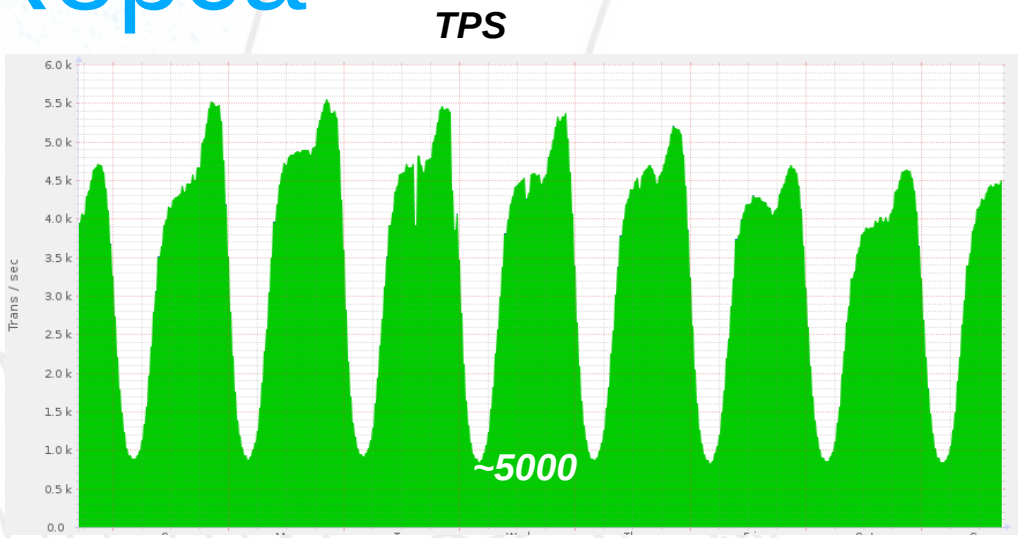


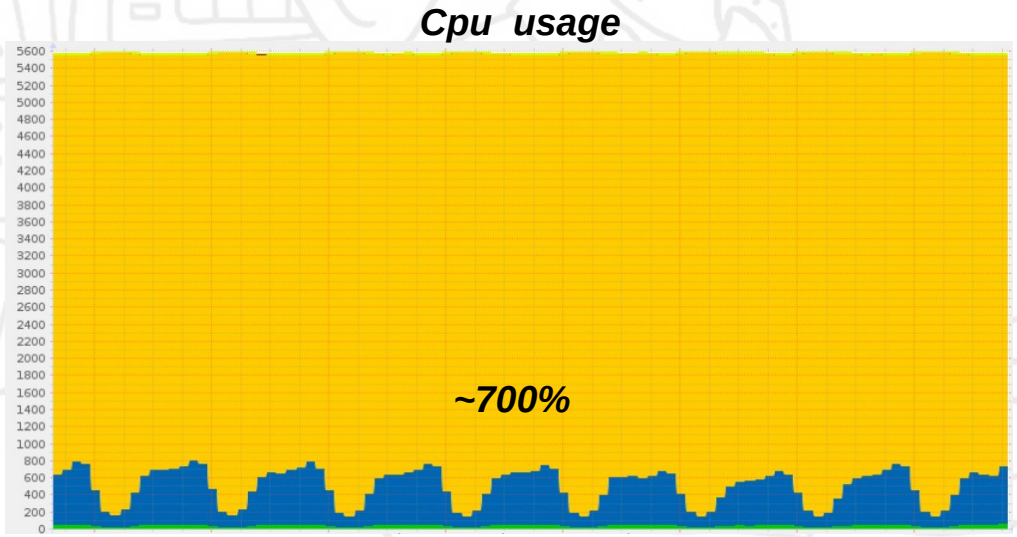
!2009 Michael Tyurin

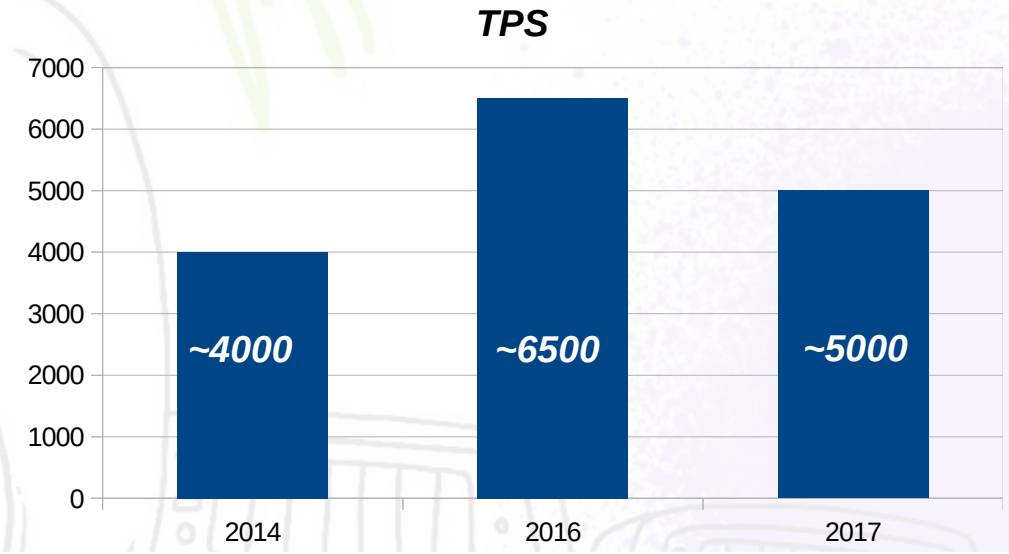


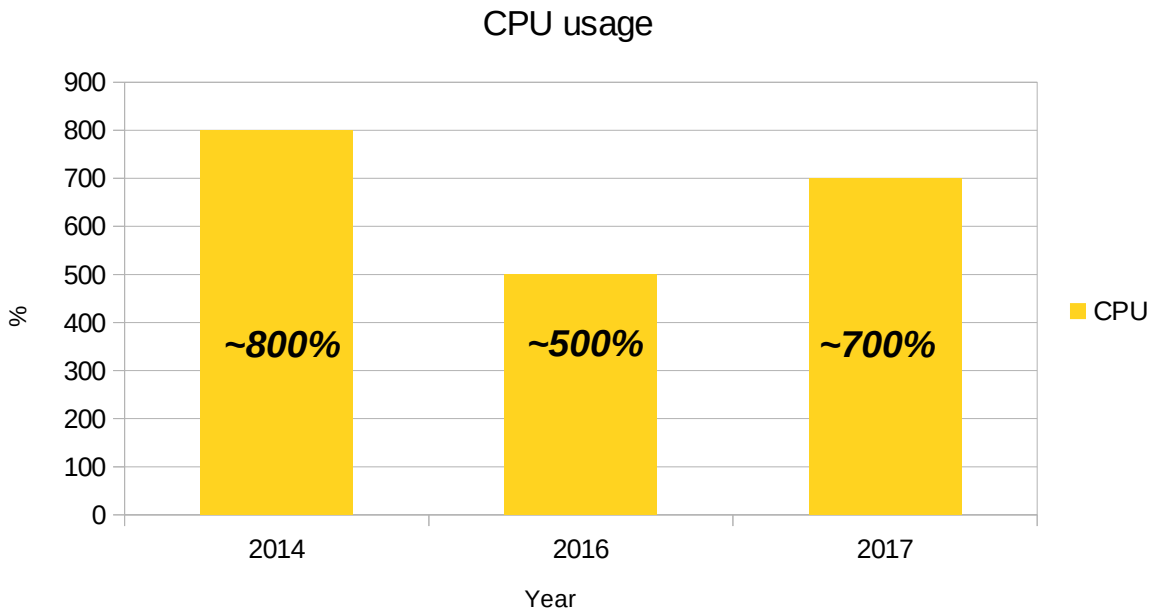
https://goo.gl/sXABqK

Repca


TPS
Trans / sec
6.0 k
5.5 k
5.0 k
4.5 k
4.0 k
3.5 k
3.0 k
2.5 k
2.0 k
1.5 k
1.0 k
0.5 k
0.0
~5000


Cpu usage
5600
5400
5200
5000
4800
4600
4400
4200
4000
3800
3600
3400
3200
3000
2800
2600
2400
2200
2000
1800
1600
1400
1200
1000
800
600
400
200
0
~700%


TPS
7000
6000
5000
4000
3000
2000
1000
0
~4000
~6500
~5000
2014
2016
2017

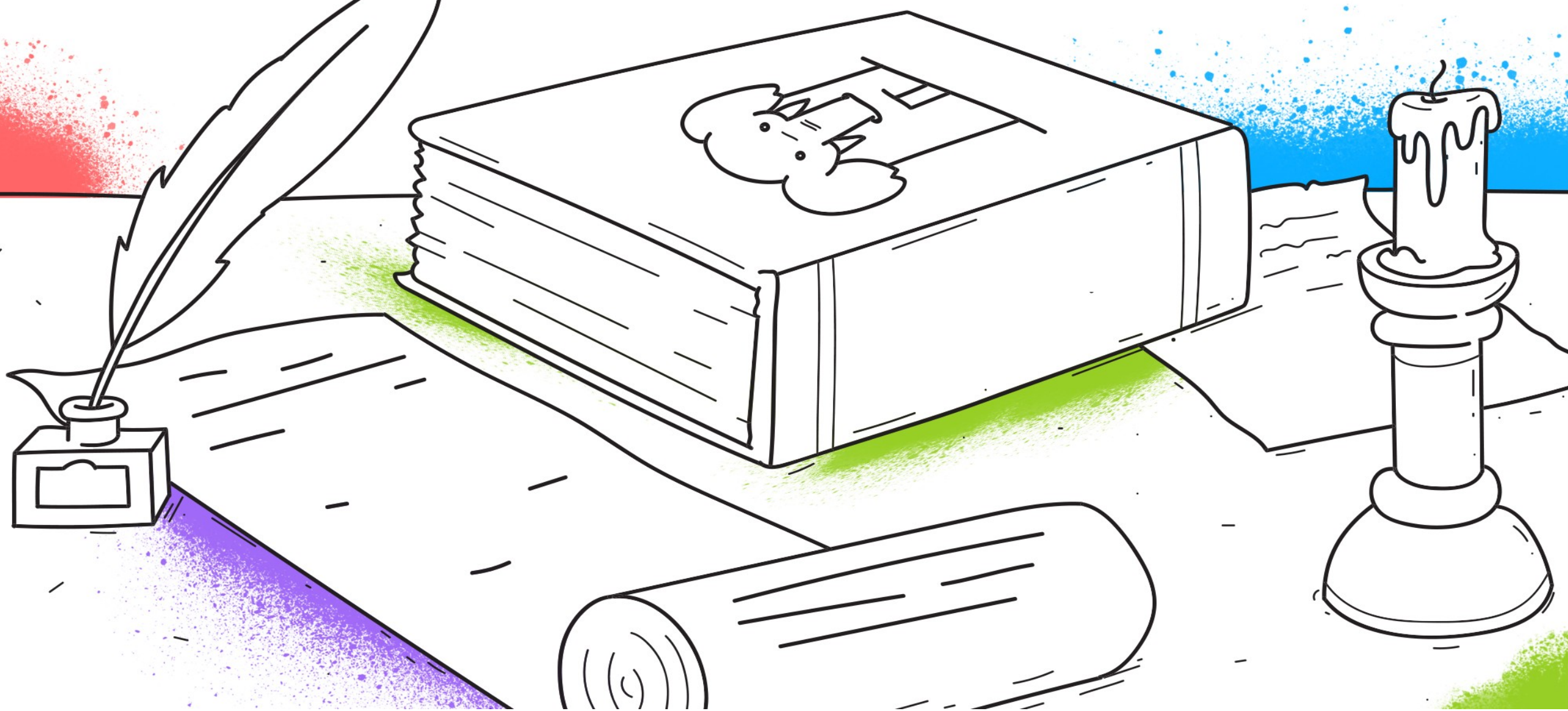

CPU usage
900
800
700
600
500
400
300
200
100
0
%
~800%
~500%
~700%
2014
2016
2017
Year
CPU

Avito

Conclusion

- There are a few kinds of standbys
- We can scale reads with the help of standby
  - ignore stale reads
  - logical routing
  - hot cache
- There are some pitfalls with standby in production
- Archive and backup depends on your DRP
- Major upgrade with standby also needs advanced manipulations

*** https://www.postgresql.org/message-id/15615-a64615b9b466c18f%40postgresql.org



https://www.avito.ru/company/job/dp-eng

Avito

kevteev@avito.ru