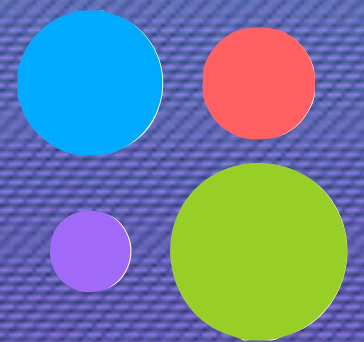
PgBouncer and 20,000 TPS on one node

advanced tuning, hacks and problem solving

Victor Yagofarov, DBA
vyagofarov@avito.ru



Avito

About Avito

- avito.ru is the biggest classified site in Russia
- Third largest classified site in the world (after Craigslist in the US and 58.com in China)
- Audience of 35+ million active users monthly
- 15-25 thousand transactions per second at the most heavy-loaded PostgreSQL nodes
- Over 300 PgBouncer instances

About me

- Victor Yagofarov, DBA
- I am a PostgreSQL specialist with deep systems administration background in HA/HL environments.

- For last three years my main occupation has been connected with improvement of postgres HA-clusters in two of the biggest Russian IT-companies.

About this talk

- How we use PgBouncer in Avito
- Capacity planning
- Load-balancing and high availability
- Tuning the most important config variables
- Hidden abilities
- Limitations
- Monitoring
- Patches
- What doesn’t work in PgBouncer

Few words about PgBouncer

Role of PgBouncer in Avito

- Reduces PostgreSQL-backends forking
- Connections economy
- Capacity planning (limiting resources)
- Prepared statements cache
- Convenient authentication

With PgBouncer

```
[vyagofarov@        :~] $ cat test.sql
select 1;
[vyagofarov@        :~] $ pgbench -n -N -r -f test.sql -C -T 60 -p 6104 -Uvyagofarov db_test_1
transaction type: Custom query
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
duration: 60 s
number of transactions actually processed: 411234
tps = 6853.889034 (including connections establishing)
tps = 12864.428641 (excluding connections establishing)
statement latencies in milliseconds:
        0.071523        select 1;
[vyagofarov@        :~] $
```

('-C' - makes a new connection for each query)

Without PgBouncer

```
[vyagofarov@          :~] $ pgbench -n -N -r -f test.sql -C -T 60 -p 5432 -Uvyagofarov db_test_1
transaction type: Custom query
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
duration: 60 s
number of transactions actually processed: 41028
tps = 683.792364 (including connections establishing)
tps = 6391.368008 (excluding connections establishing)
statement latencies in milliseconds:
        0.148527        select 1;
[vyagofarov@          :~] $
```

10x slower for a typical website workload

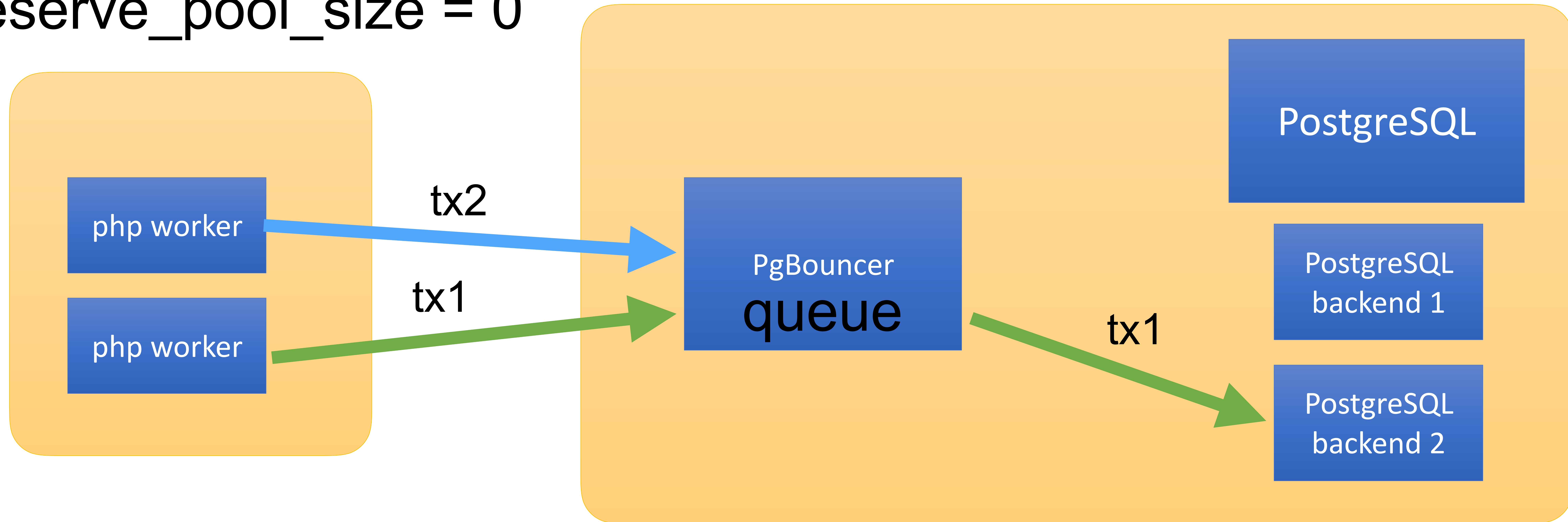
('-C' - makes a new connection for each query)

Multiplexer

pool_size=1

pool_mode = transaction

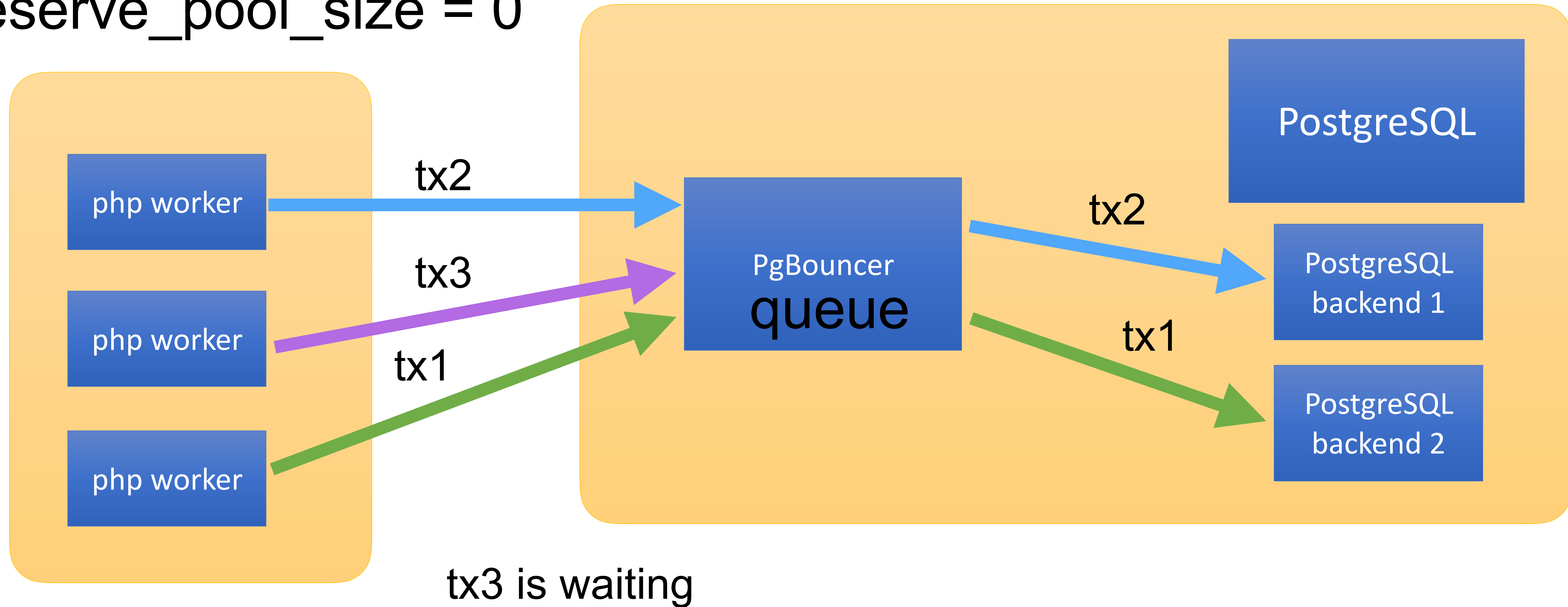
; reserve_pool_size = 0



Multiplexer (part 2)

pool_size=2 pool_mode = transaction

; reserve_pool_size = 0

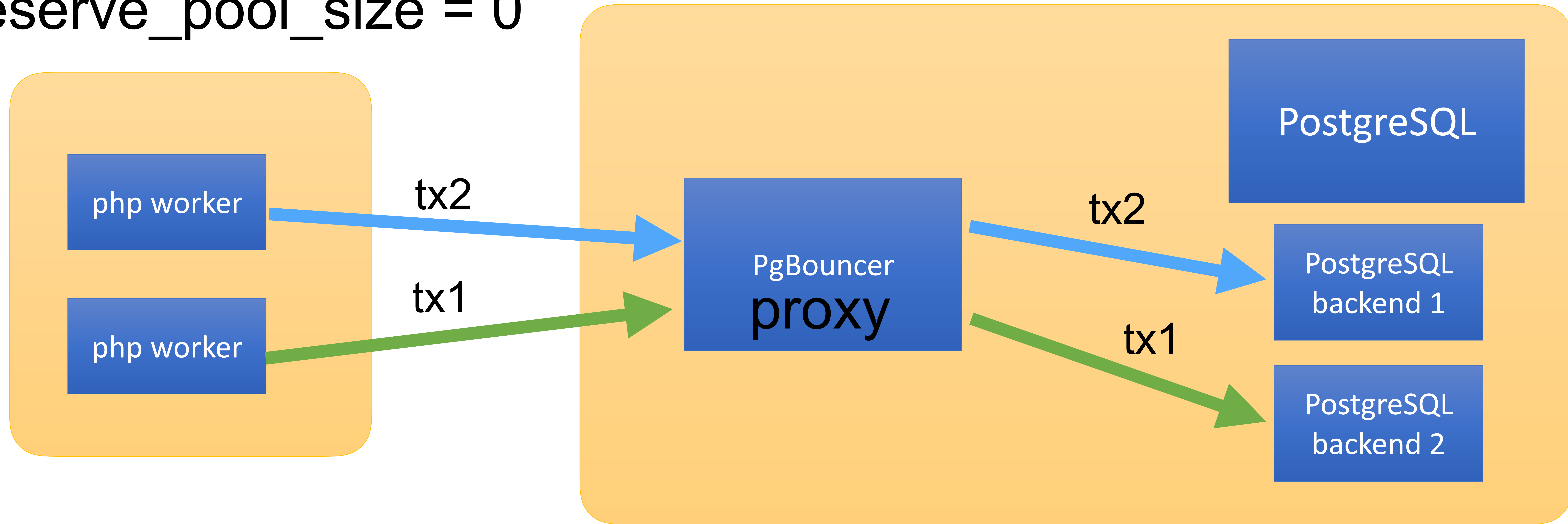


Multiplexer (part 3)

pool_size=2

; reserve_pool_size = 0

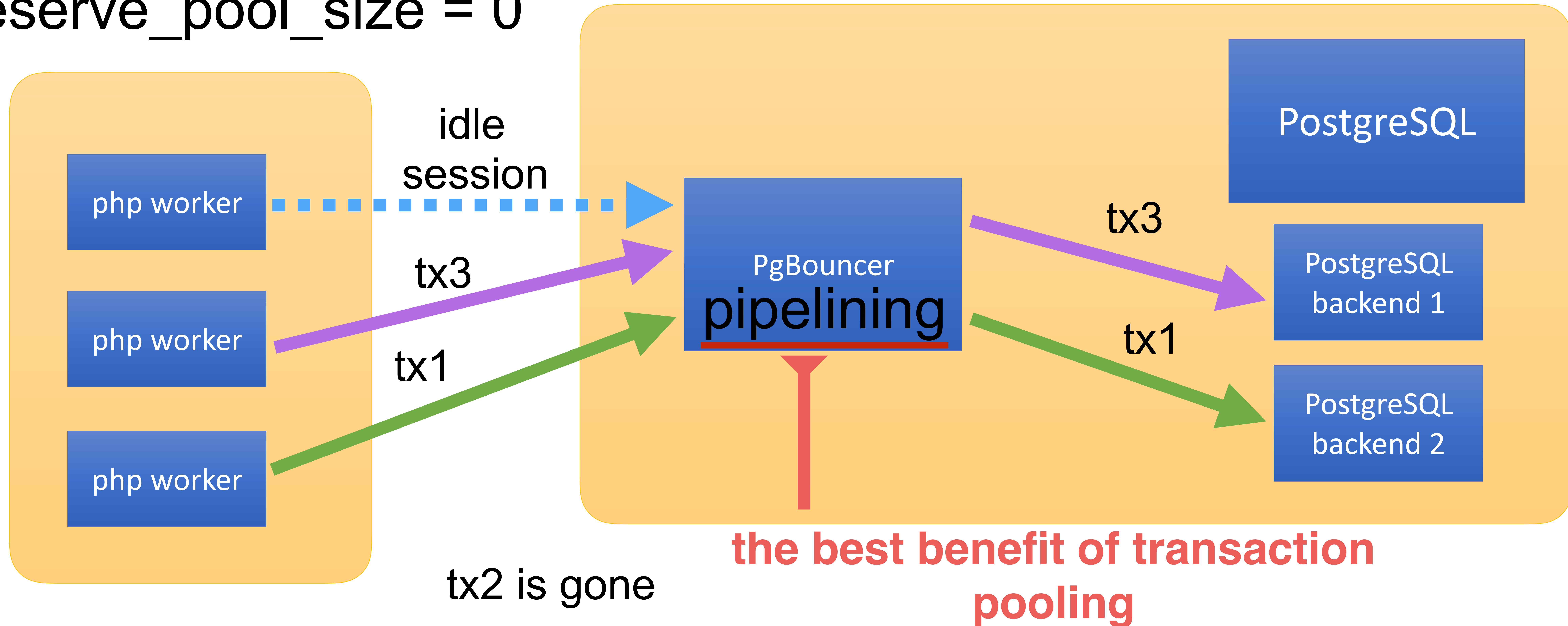
pool_mode = transaction



Multiplexer (part 4)

pool_size=2 pool_mode = transaction

; reserve_pool_size = 0



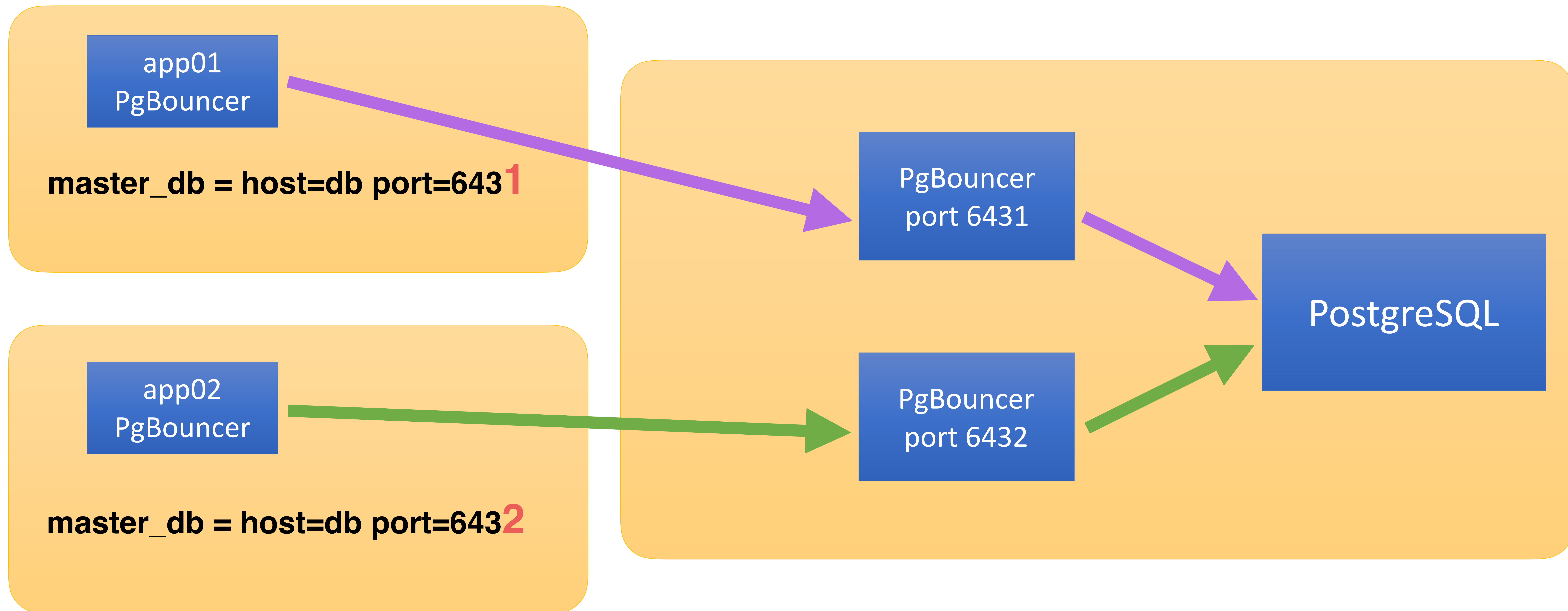
Transaction pooling

pool_size=160

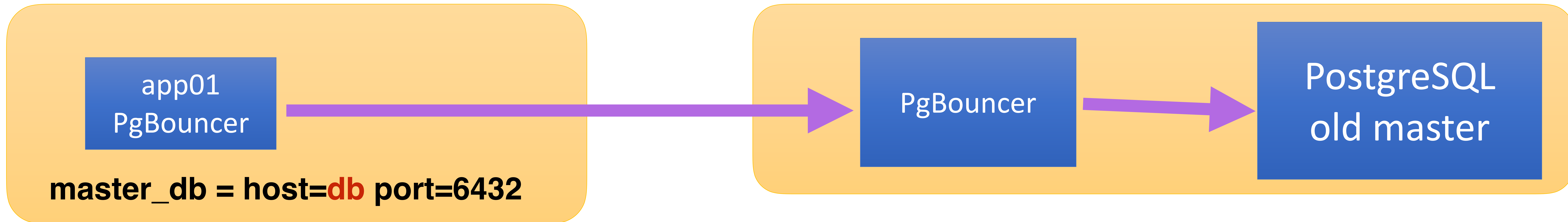
only 160 postgresql backends

serve 25 000 TPS on one node

When 1 CPU core is not enough

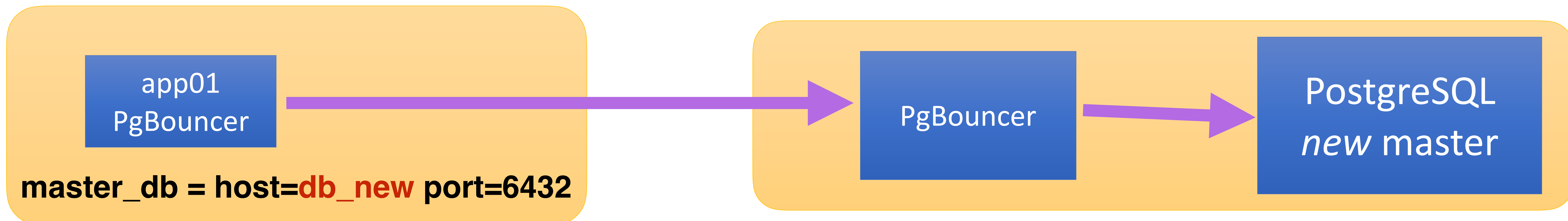


Moving to another PG server



Just change *host* and reload (HUP) app-side PgBouncer.

Be afraid of split-brain.

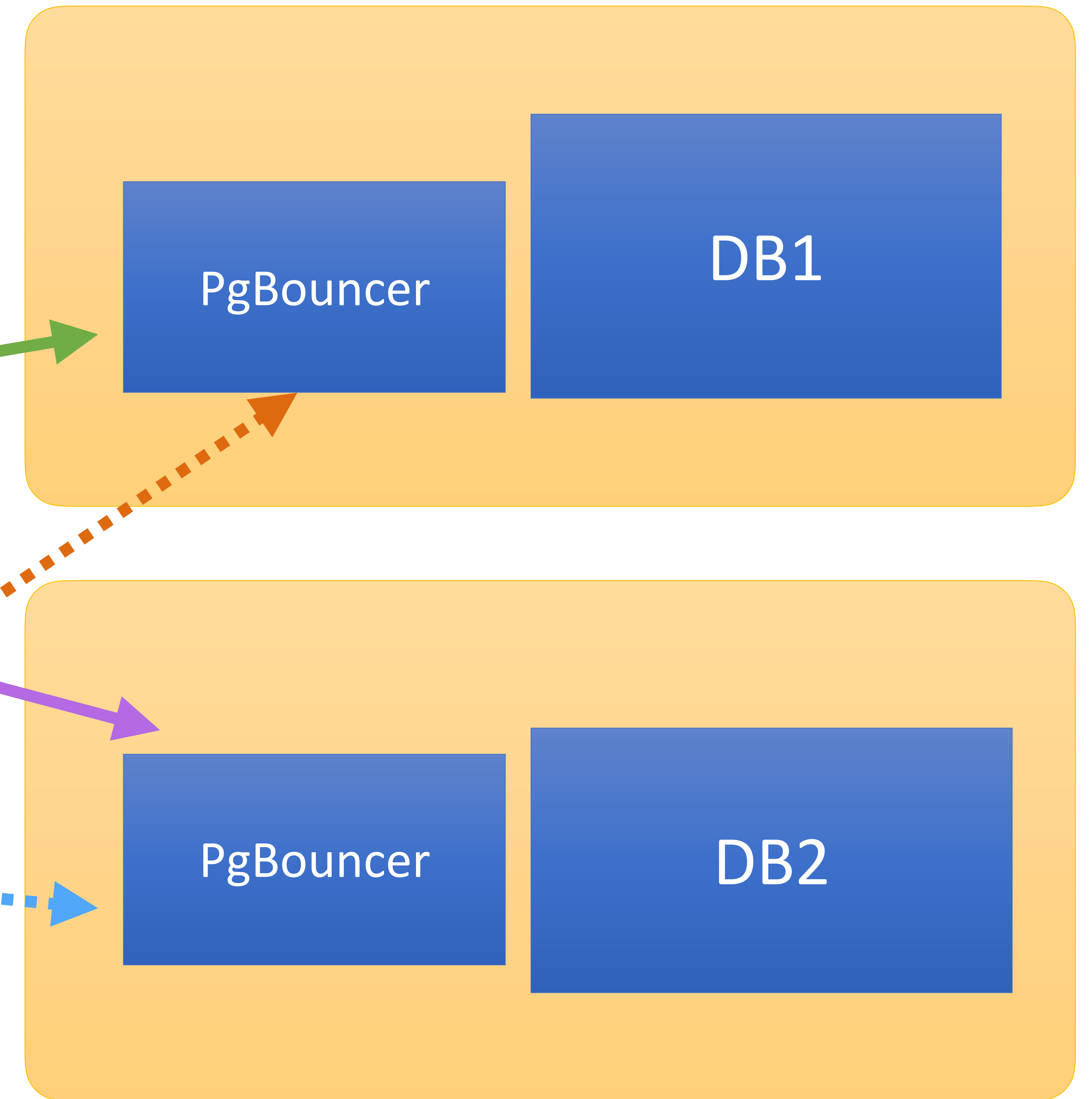


Shoot yourself in the foot?

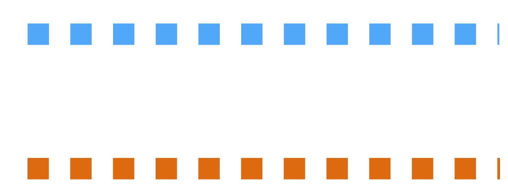
pool_size=1

; reserve_pool_size = 0

	php function A	php function B
1	tx1 db1 **does work, then 'idle in transaction'**	
2		tx1 db2 **does work, then 'idle in transaction'**
3	tx2 db2	
4		tx2 db1



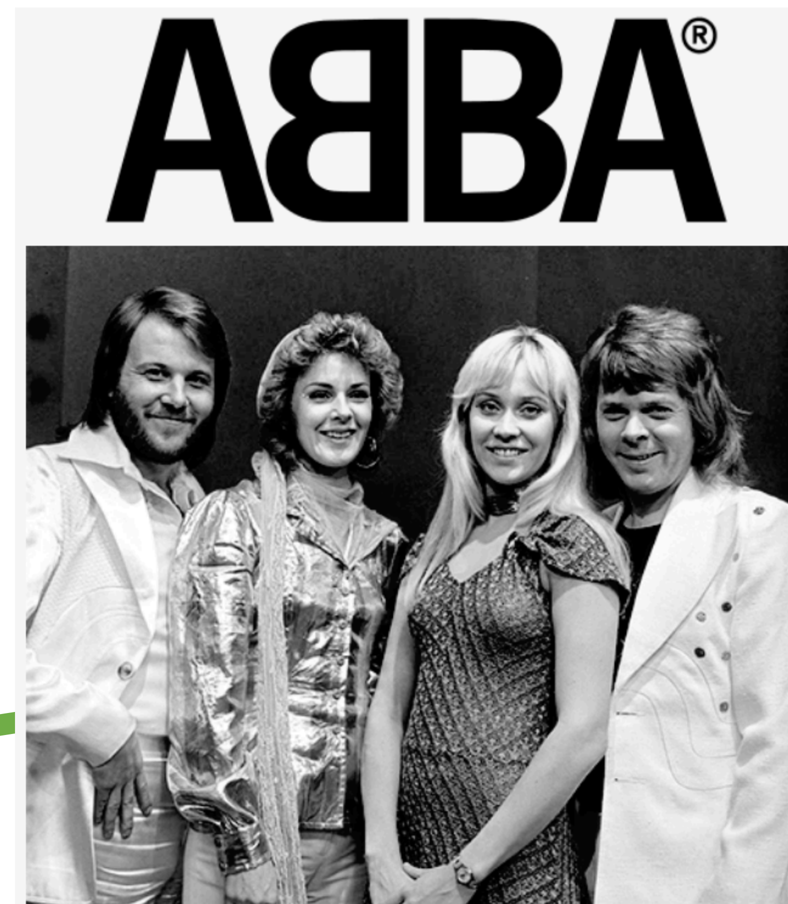
- waiting for a pool

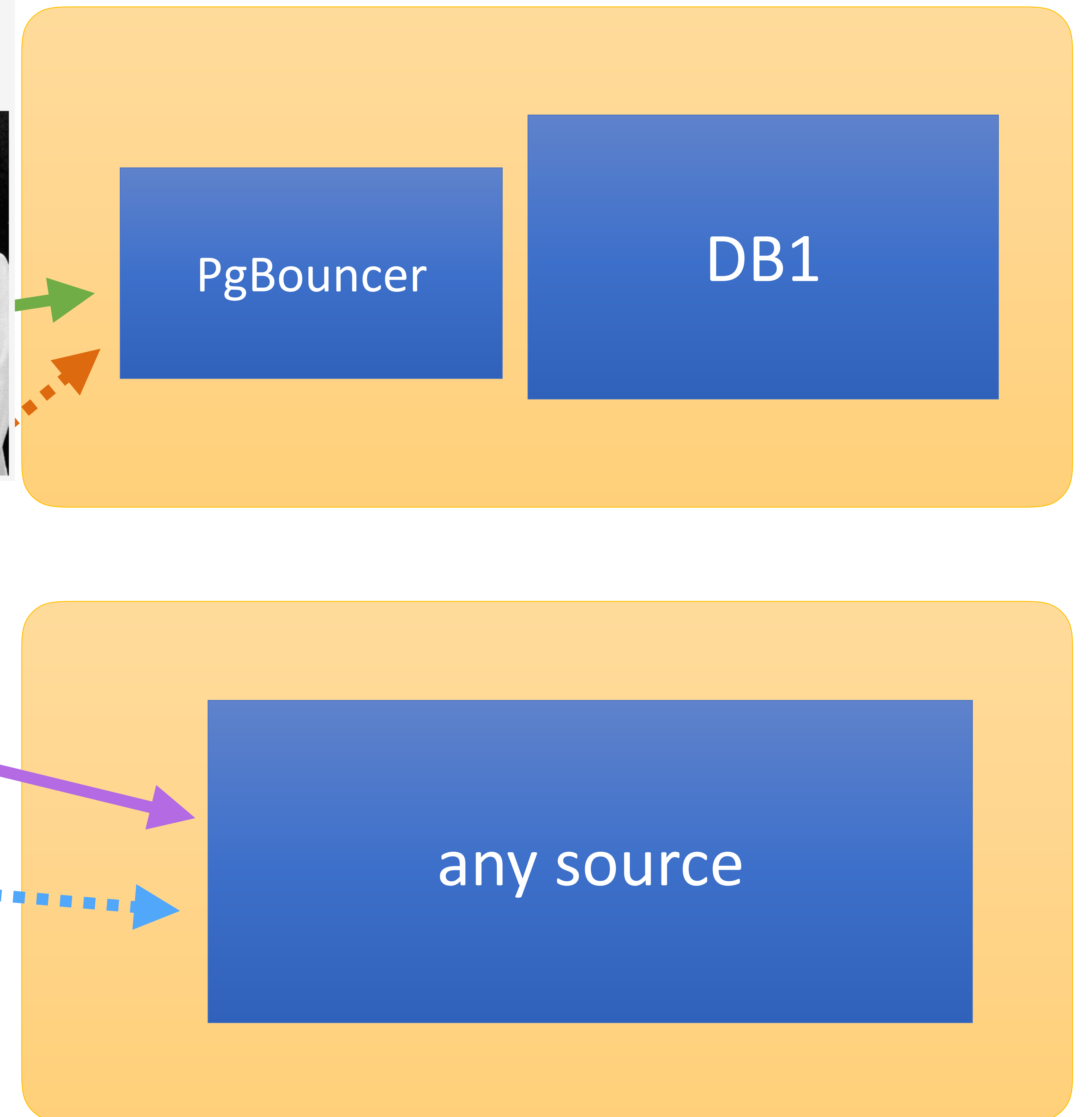


Classic

pool_size=1
; reserve_pool_size = 0

	php function A	php function B
1	tx1 db1 idle in transaction	
2		tx1 db2 idle in transaction
3	tx2 db2	
4		tx2 db1





'deadlock detection' is not possible here

'idle in transaction' statements

'Idle in transaction' is bad… M’kay?

How we use PgBouncer in Avito

PgBouncer in Avito

- We use 'server-side' PgBouncer near PostgreSQL instances
- We use 'app' (local, client-side) PgBouncer at each application node
- We use separate pools for each application at the 'server-side' bouncer (some services use the same DB)
- We use a special pgbouncer instance for developers at each database server with 'session pooling mode'.

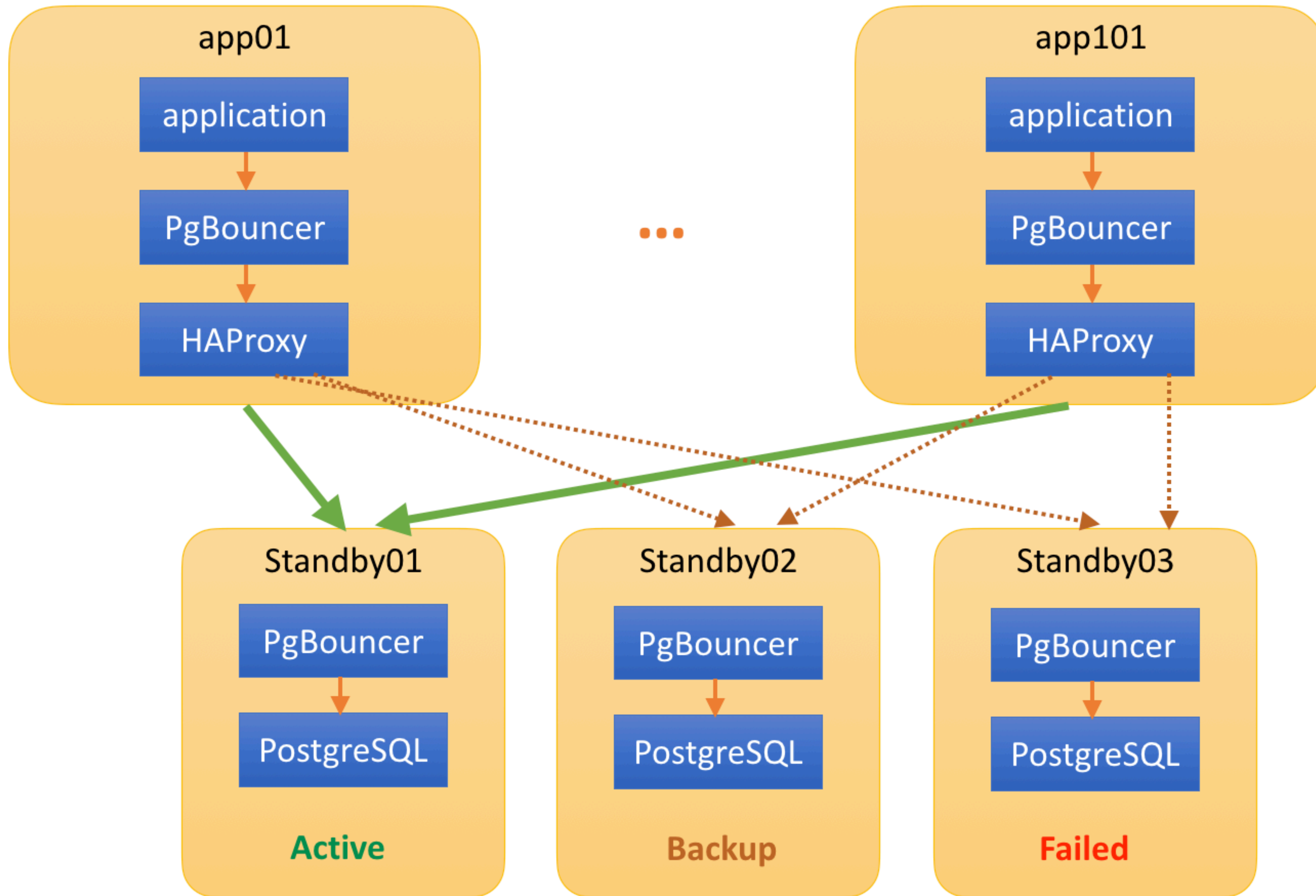
Our scheme

Pros:

- This scheme allows to keep constant number of connections to heavy-loaded PgBouncer (*max_client_conn* exceeding)
- One single app cannot 'explode' and occupy the whole server pool(s)

Cons:

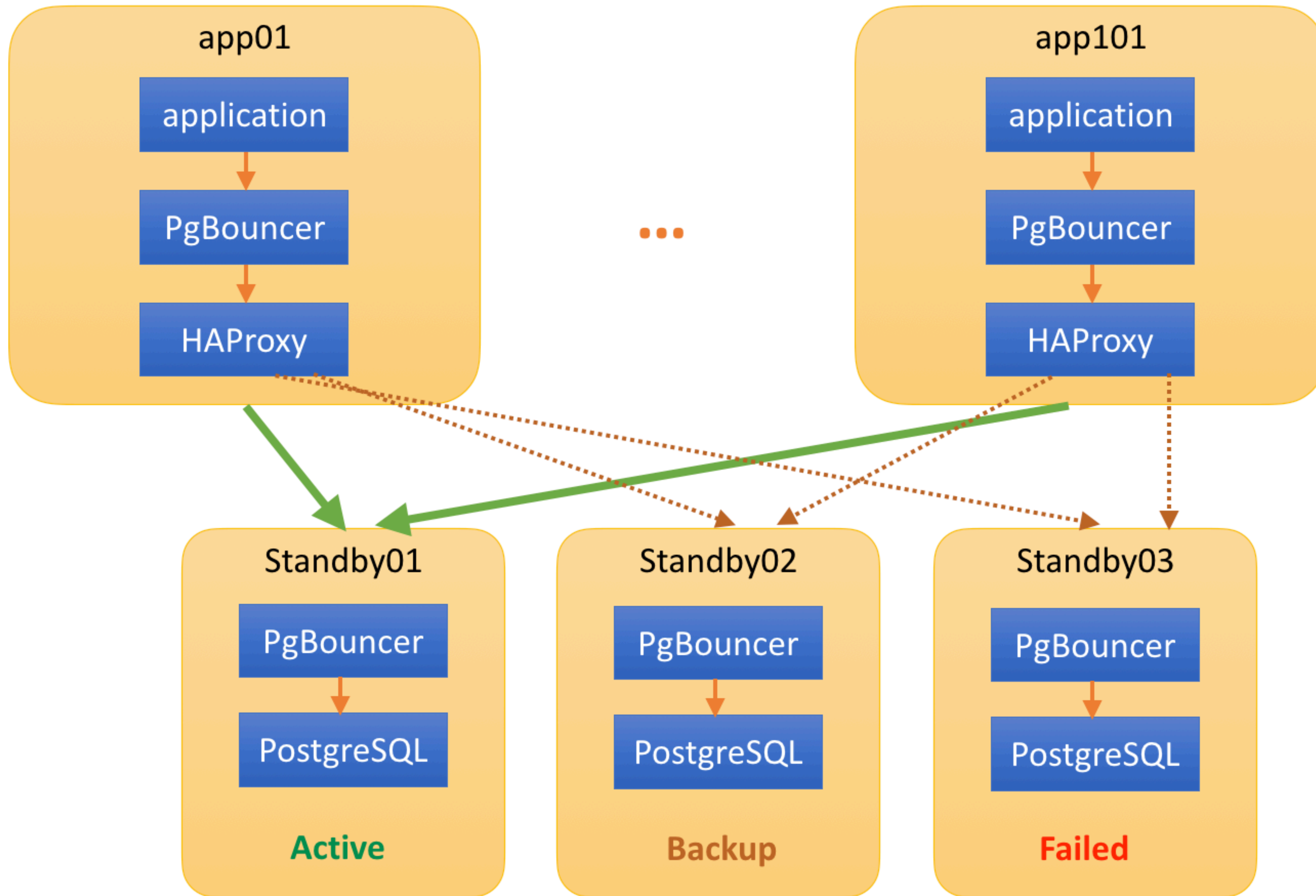
- Requires flexible and smart config management system
- Not easy to change 'upstream' host for all apps atomically


app01
application
PgBouncer
HAProxy
...
app101
application
PgBouncer
HAProxy
Standby01
PgBouncer
PostgreSQL
Active
Standby02
PgBouncer
PostgreSQL
Backup
Standby03
PgBouncer
PostgreSQL
Failed

Capacity planning

Capacity planning

- Count the number of 'app backends' of ~~micro~~service
- Count the number of simultaneous transactions for each backend
- Place 'app-side' PgBouncer near each 'app backend'
- Set *pool_size* for each 'app-side' **PgBouncer = *max. sim. transactions per backend* + *1 (reserve_pool_size)***
- Add pool into 'server-side' PgBouncer for this service
- Set pool_size of 'server-side' **PgBouncer = *'app-side' pool_size * number of 'app backends'* + *1***



app01
application
PgBouncer
HAProxy
...
app101
application
PgBouncer
HAProxy
Standby01
PgBouncer
PostgreSQL
Active
Standby02
PgBouncer
PostgreSQL
Backup
Standby03
PgBouncer
PostgreSQL
Failed

Load-balancing and high availability

HA, load-balancing

PgBouncer (app-side) pool example:

```
[databases]
db_main_s    = host=127.0.0.1            port=16002 pool_size=10
```

HA, load-balancing

HAproxy config example:

```
listen  pgsql-db_main_s
        bind 127.0.0.1:16002
        timeout client 20m
        timeout connect 1s
        timeout server 20m
        balance roundrobin
        option log-health-checks
        option tcpka
        option tcplog
        option httpchk GET /db_main_s?username=app_ro&port=6432 # checker's settings
        http-check send-state

        server host-sb01 host-sb01:6432 check addr 127.0.0.1 port 5777 inter 6s fall 5 rise 3
        server host-sb02 host-sb02:6432 check addr 127.0.0.1 port 5777 inter 6s fall 5 rise 3 backup
        server host-sb03 host-sb03:6432 check addr 127.0.0.1 port 5777 inter 6s fall 5 rise 3 backup
```

HA, load-balancing

xinetd:

```
cat /etc/xinetd.d/pgcheck

service pgcheck
{
        disable         = no
        type            = UNLISTED
        flags           = REUSE
        socket_type     = stream
        port            = 5777
        wait            = no
        user            = nobody
        server          = /usr/local/bin/pgcheck
        log_on_failure  += USERID
        only_from       = 127.0.0.1/32
        per_source      = UNLIMITED
}
```

HA, load-balancing

pgcheck (simplified example, simulates http-server, collects logs):

```
#!/usr/bin/env perl
…
$| = 1; # disable buffering

# Set whole script timeout to 5 seconds via alarm
$SIG{ ALRM } = sub {
	http 504 => "Timeout checking database health";
};
alarm 5; ### whole script timeout
…

my $dbh = DBI->connect("dbi:Pg:dbname=$db;host=$host;port=$port", "$username", '',
		{ PrintError => 0, RaiseError => 0, pg_server_prepare => 0 } ) or # disable prepared statements
	http 502 => "Error occured connecting database ($DBI::errstr)";
…
```

Avito

HA, load-balancing

pgcheck, simplified example

```
…

# do not use database if check_ha() returns 'false'

my $sth = $dbh->prepare("select public.check_ha()");
my $rv = $sth->execute or
	http 503 => "Error occured while 'select check_ha()' on '$db' at '$host' ($DBI::errstr)";
my @row = $sth->fetchrow_array;
if ( $row[0] == 0 ) {
	http 503 => "Error occured while 'select check_ha()' on '$db' at '$host': service disabled
manually";
}
…
# If everything is ok, return 200
http 200 => "Database '$db' at '$host' is alive";
```

HA, load-balancing

check_ha() (simplified example of stored procedure):

```
db_main=# \df+ check_ha

use Sys::Hostname;
my $h = Sys::Hostname::hostname;

if ($h eq 'unknown-host') {
    return 0;
} elsif ($h eq 'db-sql02') { # standby
    return 1;
} elsif ($h eq 'db-sql03') { # master
    return 0;
} elsif ($h eq 'db-sql05') { # standby
    return 1;
} else {
    return 0;
}
```

HA, load-balancing

pgcheck puts each result of check into time-series database (in non-blocking way):

```
 2 avito_stats=# select distinct on (pool, client_node) * from
 3 haproxy.stats where txtime > now() - '2 seconds'::interval
 4 order by pool, client_node limit 2 ;
 5 -[ RECORD 1 ]-----+-----------------------------------------
 6 client_node       | store-app86
 7 pool              | pgsql-avito_market
 8 backend           | db-fe01
 9 state             | up
10 error_description | Database 'avito_market' at 'db-fe01' is alive
11 txtime            | 2017-04-09 14:51:26.661409+03
12 -[ RECORD 2 ]-----+-----------------------------------------
13 client_node       | store-app19
14 pool              | pgsql-avito_market
15 backend           | db-fe02
16 state             | up
17 error_description | Database 'avito_market' at 'db-fe02' is alive
18 txtime            | 2017-04-09 14:51:25.011711+03
19
20 avito_stats=#
```

HA, load-balancing

Monitoring uses collected data from time-series database:

```sql
select total.pool, total.cnt as total, coalesce(down.cnt,0) as down, coalesce(alive.cnt,0) as alive, coalesce(real_down.cnt,0) as real_down from
  ( select pool, count(pool) as cnt from
    (
      select pool, client_node, count(pool)
      from haproxy.stats where txtime > now() - '5 minutes'::interval and state = 'down'
      group by pool, client_node
      having count(pool) > ${failed_pings}
    ) c
    group by pool
  ) real_down
  right join
  ( select pool, count(client_node) as cnt from
    (
      select distinct on (client_node, pool) client_node, pool, state
      from haproxy.stats where txtime > now() - '5 minutes'::interval and state = 'down'
      order by client_node, pool, txtime desc
    ) b
    group by pool
  ) down
  on ( real_down.pool = down.pool )
  right join
  ( select pool, count(pool) as cnt from
    (
      select distinct on (pool, client_node) pool, client_node from
      haproxy.stats where txtime > now() - '5 minutes'::interval
      order by pool, client_node
    ) a
   group by pool
  ) total
  on ( down.pool = total.pool )
  right join
  ( select pool, count(client_node) as cnt from
    (
      select distinct on (client_node, pool) client_node, pool, state
      from haproxy.stats where txtime > now() - '50 seconds'::interval and state = 'up'
      order by client_node, pool, txtime desc
    ) by
    group by pool
  ) alive
  on ( total.pool = alive.pool )
group by total.pool, total.cnt, down.cnt, alive.cnt, real_down.cnt;
```

Anomalies detection for free!

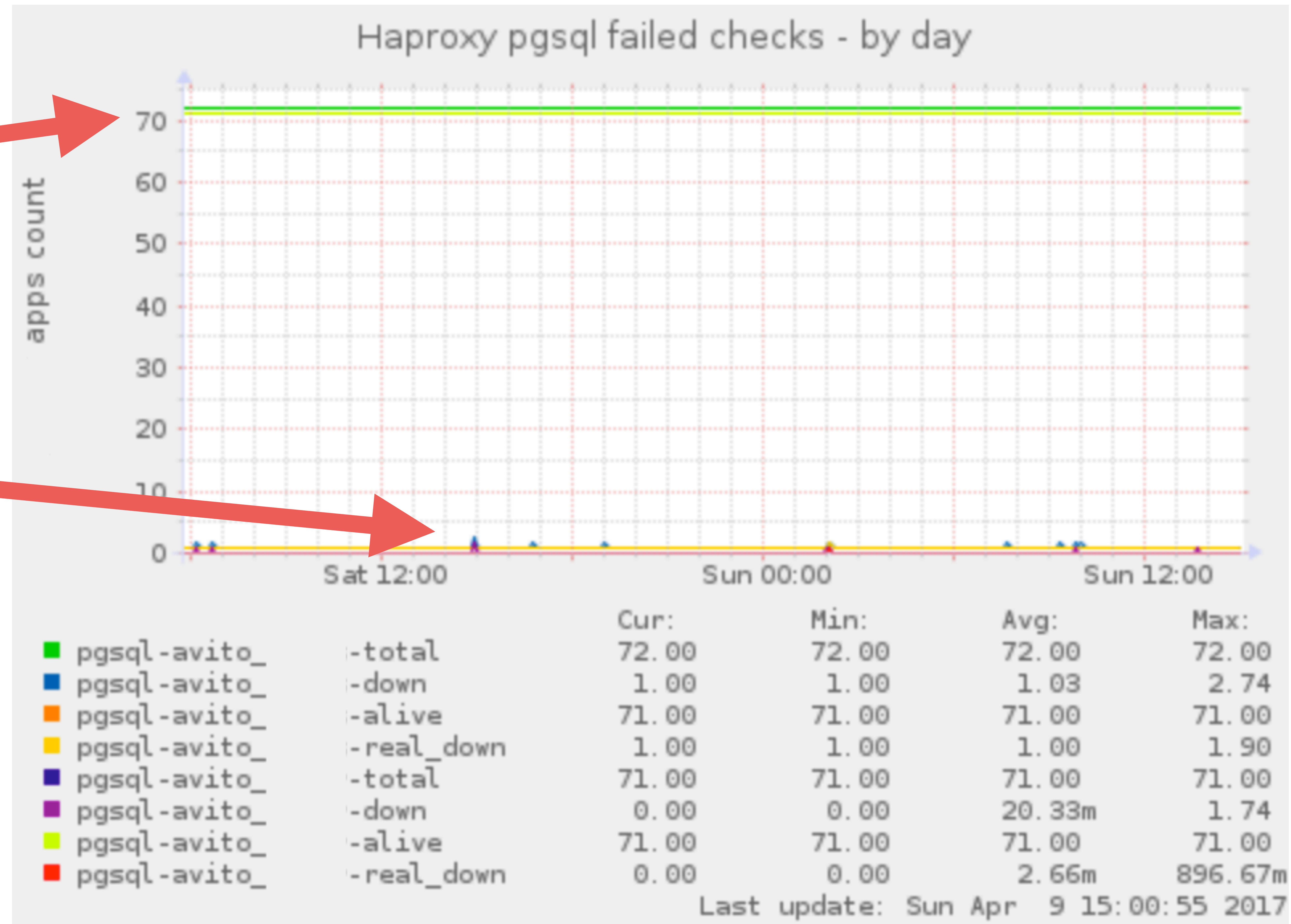
… helps to find anomalies

number of app containers

WTF?!

zero == good

1 == one check failed



Anomalies detection for free!

This helps us to find micro-freezes in our RAID controllers.

Tuning the most important config variables

Example of pool and settings in our PgBouncer

```
db_new      =  user=user15 pool_size=10 datestyle='ISO,DMY' \
connect_query='select x_init();' pool_mode=transaction
```

```
unix_socket_dir = /var/run/postgresql

auth_type = hba
auth_hba_file = /etc/pgbouncer/pg_hba-server01.conf
auth_file = /etc/pgbouncer/userlist-server01.txt
```

'server-side pgbouncer'

max_client_conn = 2600

default_pool_size = 10

'app-side pgbouncer'

max_client_conn = 200

default_pool_size = 5

Optimal settings (for us)

'app and server pgbouncers'

reserve_pool_size = 1
reserve_pool_timeout = 1

query_wait_timeout = 10
client_idle_timeout = 7200
pkt_buf = 8192
; sbuf_loopcnt
tcp_keepalive = 1
tcp_keepidle = 600

'server-side pgbouncer'

server_lifetime = 1200
server_idle_timeout = 300

'app-side pgbouncer'

server_lifetime = 60
server_idle_timeout = 30

Hidden abilities

- *connect_query='select x_init();'*
(pool connection string)
may be used for:

- preparing of plans
- setting variables, f.e. '*set statement_timeout = 600000;*'

- *reserve_pool_size, reserve_pool_timeout,*
query_wait_timeout config variables are almost useless,
but help to find issues with pool saturation

Limitations

PgBouncer 'cache poisoning'

```
cat include/varcache.h

enum VarCacheIdx {
	VDateStyle = 0,
	VClientEncoding,
	VTimeZone,
	VStdStr,
	VAppName,
	NumVars
};
```

Setting any of these variables via *SET* may totally ruin all PgBouncers for the whole service.

e.g. " *SET datestyle TO postgres, ymd* "
via psql or IDE connected to 'production' pool
(automatically)

Use **dev** PgBouncer for development purposes

Some IDEs can’t work with PgBouncer via transaction pooling:

- prepared statements issues
- something else (it depends on IDE)

IDE must **not** use the same PgBouncer instance as a production/test code:

- **pgbouncers cache poisoning**
- too many connections per IDE
- search_path changing

pool name must be equal to physical postgres database name

pool_mode = statement ; the best choice for **dev** pgbouncer

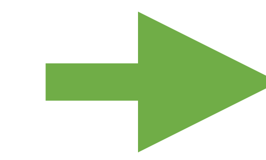
The pools are not what they seem…

pgbouncer-dev.ini:
avi_market = datestyle='ISO,DMY' **pool_size=5**

userlist-dev.txt:
"oleg" "md5s7df1986ec33k591add33c104c9ceb53"
"vlad" "md5s7df1986ec33k591add33c104c9ceb53"

pg_hba_pgbouncer-dev.conf:

host	avi_market	oleg	10.3.109.4/24	md5
host	avi_market	vlad	10.2.118.7/24	md5



Will create **2** pools with size == 5.
12 database connections!
(including reserve connection)

max_db_connections = 5 ;
will **really** limit db connections
to 5 for a **pool**.

max_db_connections is global
or per-pool setting

What else does not work as intended?

; idle_transaction_timeout

Timer is broken

https://github.com/pgbouncer/pgbouncer/issues/125

; max_db_connections (per pool mode)

Contradicts the description.
Only limits number of active sessions from any users for the **pool**

; query_timeout =

Timer is broken
https://github.com/pgbouncer/pgbouncer/issues/22

What does not work as intended?

1. Add/remove pool
2. Try to upgrade pgbouncer via "online restart" cool feature

   sudo -u postgres /bin/sh -c "/usr/sbin/pgbouncer **-R** -d /etc/pgbouncer/$NAME.ini"
3. Get crashed PgBouncer! *

** probably happens only in heavy load*

Other limitations

pool_mode=transaction is the best and the only choice for high-load*

But your code should be written carefully:

- do not allow 'idle in transactions' for a long time
- beware of changing/setting *session* variables

^^ A lot of ORMs ignore these rules

** Is PostgreSQL high load possible without PgBouncer or another pooler?*

pgjdbc and PgBouncer

setReadOnly does not work with pgbouncer and transaction pooling mode.

Currently setReadOnly change mode for all session, not for transaction. It breaks work through pgbouncer in transaction pooling mode.

https://github.com/pgjdbc/pgjdbc/issues/848 (Avito)

Monitoring

Monitoring

```
[vyagofarov@█████████~] $ sudo -u postgres psql -Upgbouncer pgbouncer -p 6432
psql (9.5.6, server 1.7.2/bouncer)
Type "help" for help.

pgbouncer=# show pools ;
     database      |    user    | cl_active | cl_waiting | sv_active | sv_idle | 
 maxwait |   pool_mode
------------------+-----------+-----------+------------+-----------+---------+-
---------+-------------
 mu_postgres      | xuser     |         0 |          0 |         0 |       1 |
       0 | transaction
 mu_protools_test | xuser     |         0 |          0 |         0 |       1 |
       0 | transaction
 pgbouncer        | pgbouncer |         1 |          0 |         0 |       0 |
       0 | statement
 protools         | xuser     |         0 |          0 |         0 |       0 |
       0 | transaction
 protools_test    | xuser     |         0 |          0 |         0 |       0 |
       0 | transaction
(5 rows)
```

Avito

Patches

cl_waiting is not a counter

We can’t correctly detect or measure database (pool) saturation while *cl_waiting* shows 'current value'.

Trying to fix it here:
https://github.com/pgbouncer/pgbouncer/pull/168

Orphan query issue

1. Client connects to PostgreSQL
2. Client starts a long-running query
3. Client dies for any reason (bug, OOM, whatever)
4. Query continues to run and consume resources (PostgreSQL behaviour)*

Almost 2 year-old pull-request:
https://github.com/pgbouncer/pgbouncer/pull/79

** I have seen never-ending queries!*

Expectations

Our Wishlist for PgBouncer

- cl_waiting as a **counter**
  https://github.com/pgbouncer/pgbouncer/pull/168 (Avito)

- Possibility to 'kill' queries from 'dead' clients
  https://github.com/pgbouncer/pgbouncer/pull/79 (Avito)

- Normal ERROR logging

- Fixed timers
  https://github.com/pgbouncer/pgbouncer/pull/127 (Zalando)
  https://github.com/pgbouncer/pgbouncer/issues/22 (mail.ru)

- Normal project development
  We have a lot of ideas how to make PgBouncer a very powerful tool

Summary

- We should write code with an understanding of the uncommon features of PgBouncer
- There are a lot of opportunities to shoot yourself in the foot (ABBA, SET, IDEs, etc.)
- While using chain '*app -> app-pgbouncer -> server-side pgbouncer -> postgres'* we can keep constant number of connections to heavy-loaded PgBouncer (server-side) and PostgreSQL
- It is convenient to move *load-balancing* and *database switching* into HAproxy
- Separate **dev** PgBouncer should be used for development purposes
- It is good to monitor a lot of metrics but without *cl_waiting* this monitoring is almost useless
- Some config parameters <u>do not</u> work as described in the manual

Thank you!

Questions?

Victor Yagofarov
DBA
Avito

skype: nas__tradamus
telegram: @nas_tradamus
email: vyagofarov@avito.ru

So sad without PgBouncer…