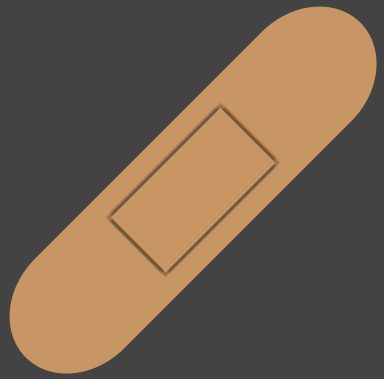
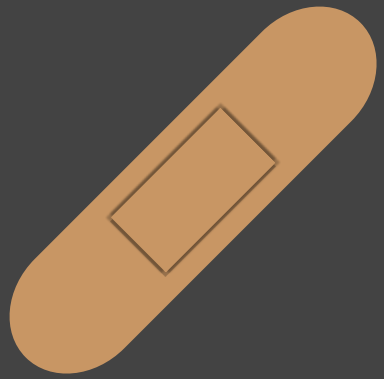


Practical PgBouncer Pain Prevention

Nick Meyer @ Academia.edu

SCaLE 23x 2026

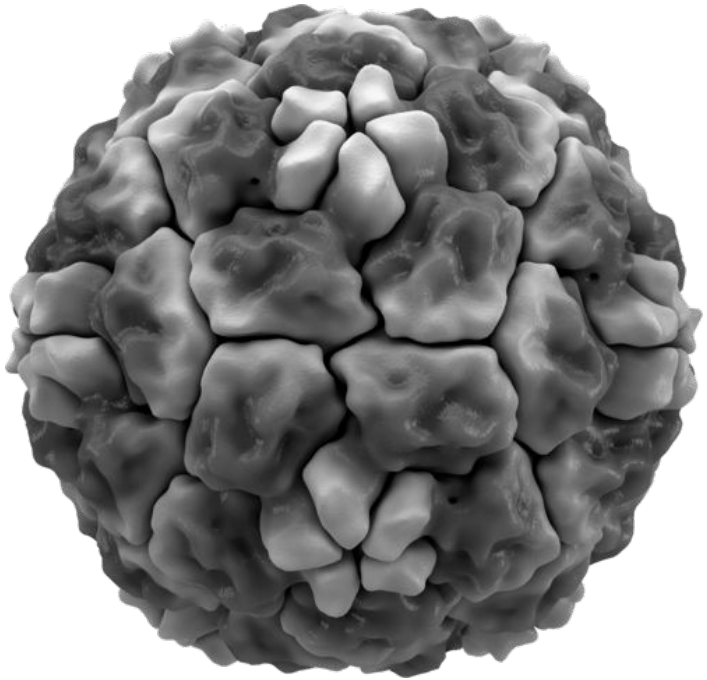


Practical PgBouncer Pain Prevention

Nick Meyer @ Academia.edu

SCaLE 23x 2026

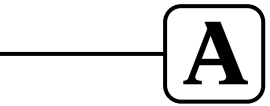
postgres problems:



“common cold”

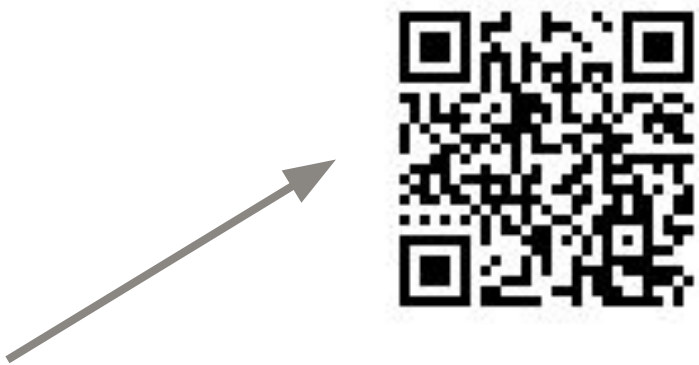
pgbouncer problems:

?



A bit about me (Nick Meyer)

- **Team lead of Platform Engineering**
- **@** Academia.edu
- https://github.com/aristocrates



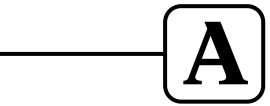
Slides: https://github.com/aristocrates/SCaLE23x_2026

A Objectives

PgBouncer:

1. **Configuration**
2. **Monitoring**
3. **Scaling (“multi-bouncer”)**

Slides: https://github.com/aristocrates/SCaLE23x_2026

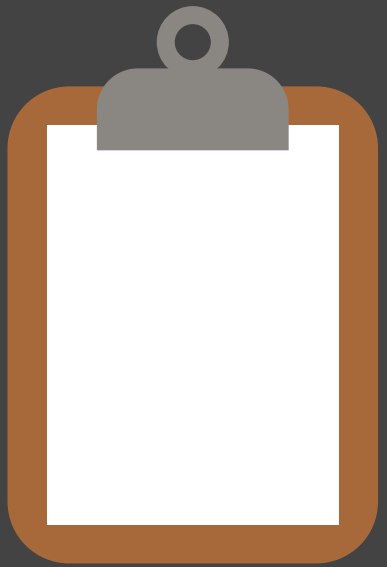


Out of scope

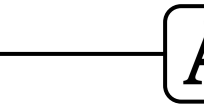
- PgBouncer alternatives
- TLS/authentication
- Medical advice

Slides: https://github.com/aristocrates/SCaLE23x_2026

Part 1
Configuration

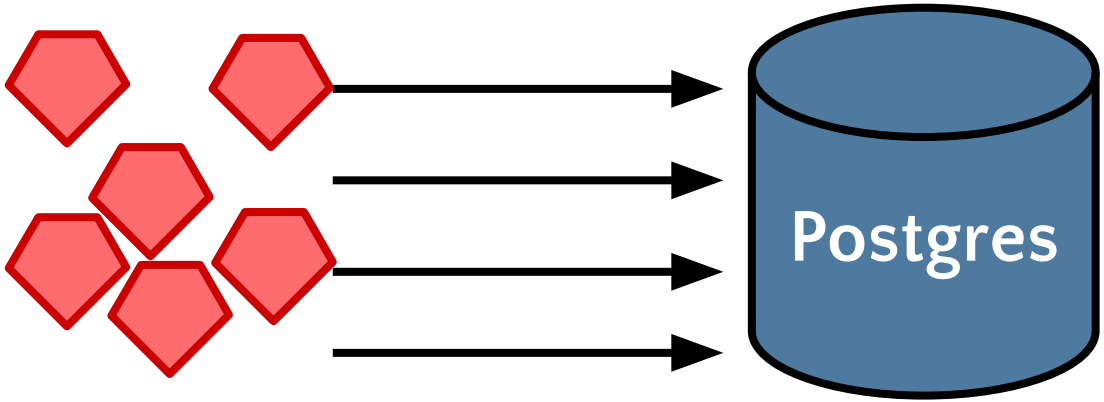


A

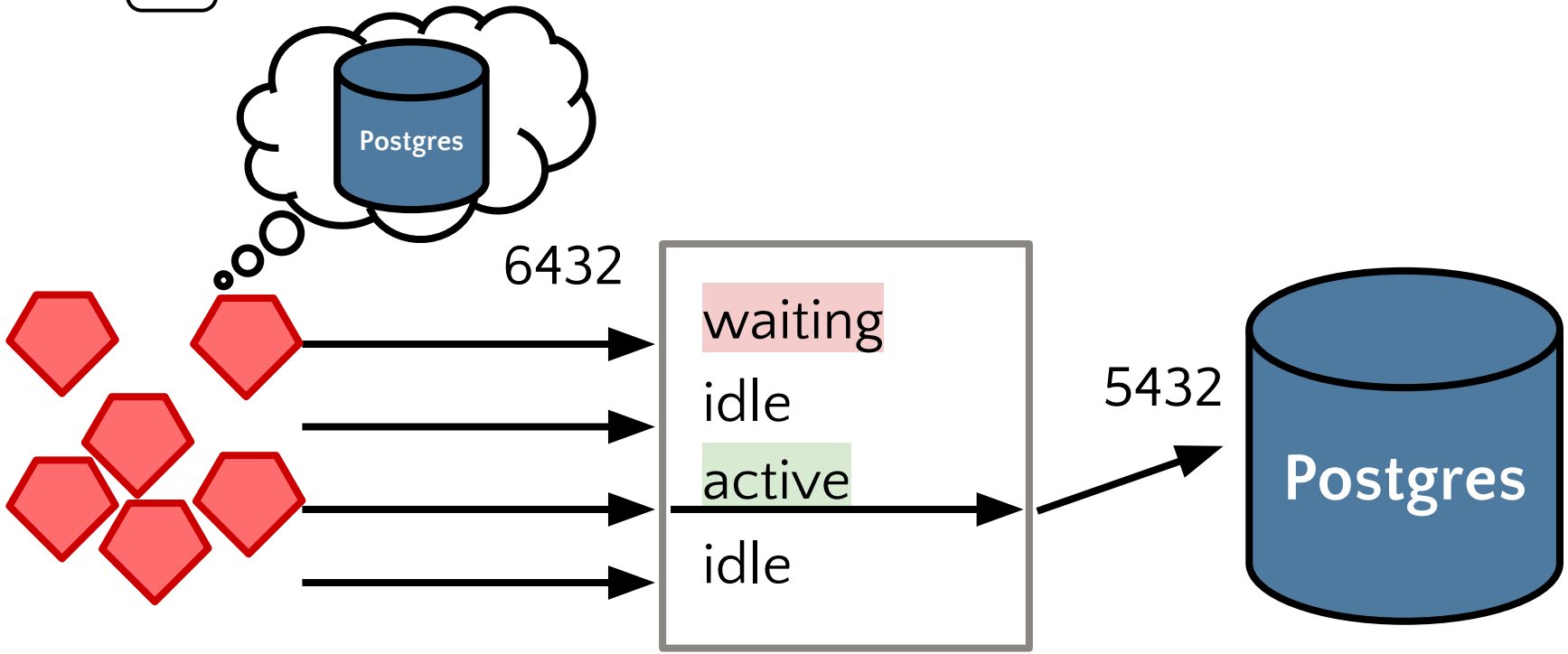


Problem: too many connections

- Tons of clients
- Tons of connections
- Connections mostly idle
- Postgres connection = process = $$$$



A Solution: PgBouncer, “connection pooler”



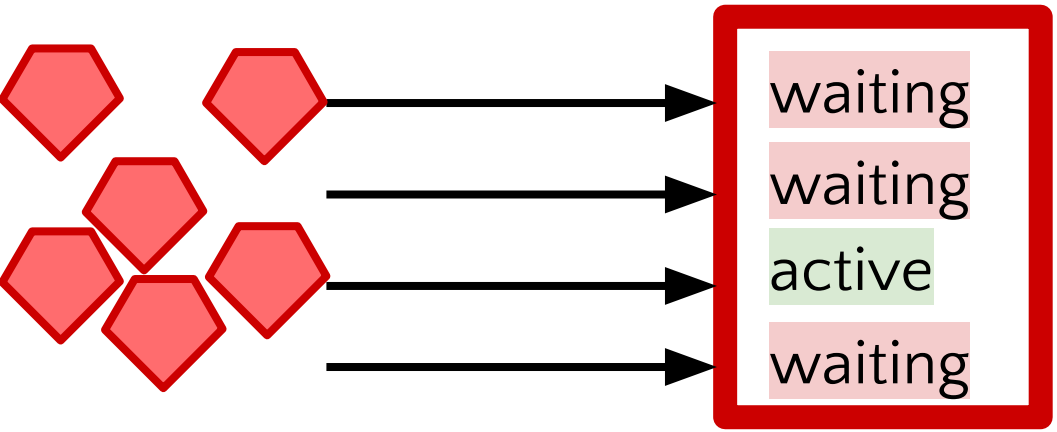
pool_mode

- Session
- Transaction
- Statement

```
BEGIN;

UPDATE users
SET name = 'Nick'
WHERE id = 1;

UPDATE user_countries
SET country = 'US'
WHERE user_id = 1;

COMMIT;


CREATE INDEX CONCURRENTLY
[...]

SELECT name
FROM users
WHERE id = 1;
```

A pool_mode

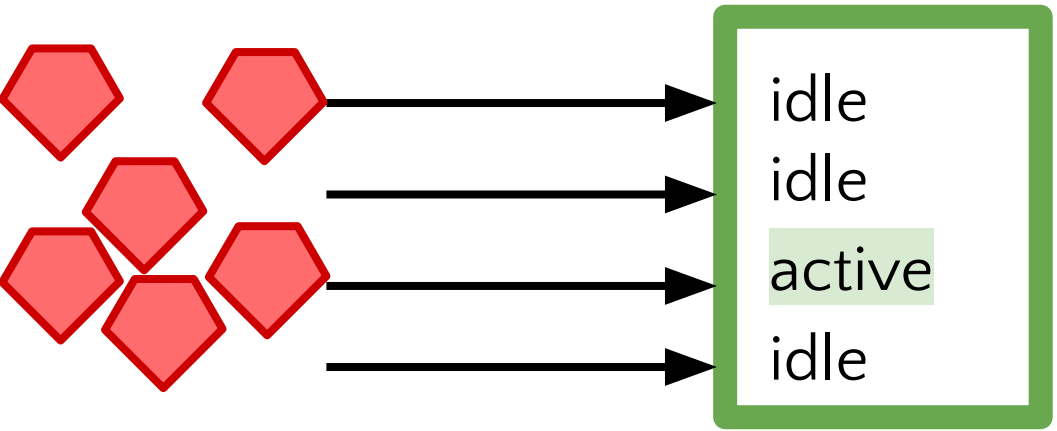
- Session
- Transaction
- Statement




```
BEGIN;

UPDATE users
SET name = 'Nick'
WHERE id = 1;

UPDATE user_countries
SET country = 'US'
WHERE user_id = 1;

COMMIT;


CREATE INDEX CONCURRENTLY
[...]

SELECT name
FROM users
WHERE id = 1;
```

A pool_mode

- Session
- Transaction
- Statement



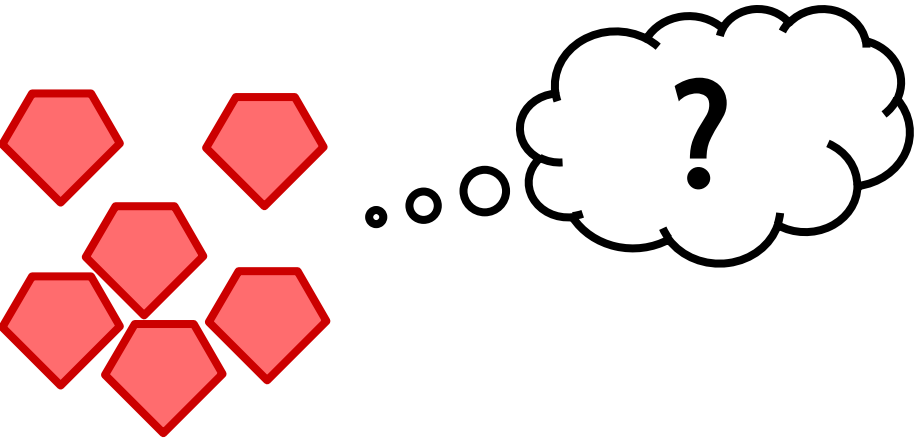

```
BEGIN;

UPDATE users
SET name = 'Nick'
WHERE id = 1;

UPDATE user_countries
SET country = 'US'
WHERE user_id = 1;

COMMIT;
```

```
CREATE INDEX CONCURRENTLY
[...]
```

```
SELECT name
FROM users
WHERE id = 1;
```

A pool_mode

- Session
- Transaction
- Statement



X ~~BEGIN;~~

```
UPDATE users
SET name = 'Nick'
WHERE id = 1;
```

```
UPDATE user_countries
SET country = 'US'
WHERE user_id = 1;
```

X ~~COMMIT;~~

```
CREATE INDEX CONCURRENTLY
[...]
```

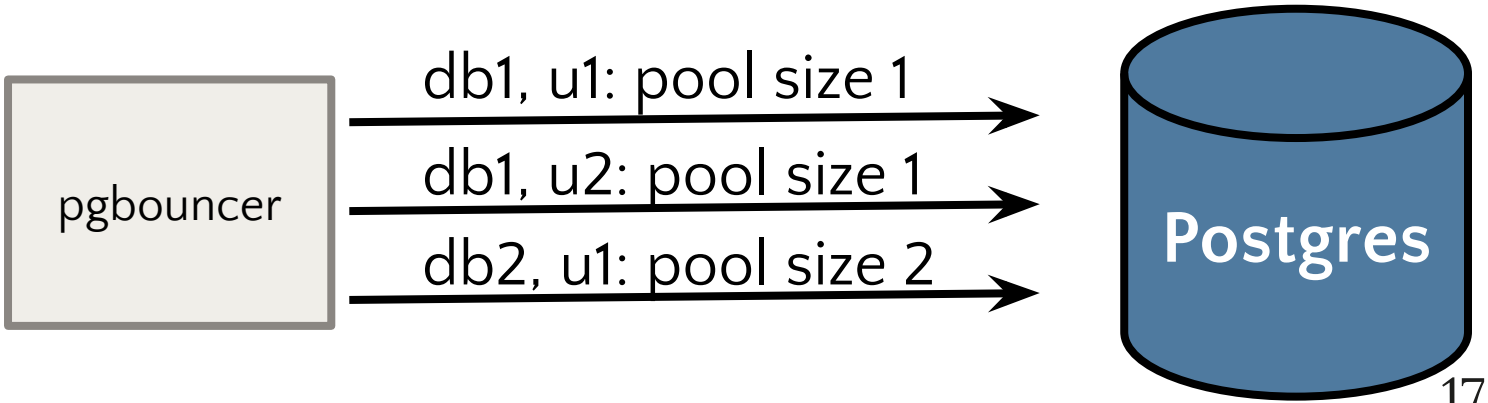
```
SELECT name
FROM users
WHERE id = 1;
```

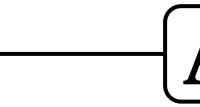
/etc/pgbouncer/pgbouncer.ini

```
[databases]
db = host=A port=B dbname=C pool_size=N

[pgbouncer]
pool_mode = transaction
(...)
```

Server connections

- **pool_size**
  - per (database, user)
- **sum(pool_sizes) < max_connections**
  - (Leave some for admin, superuser)

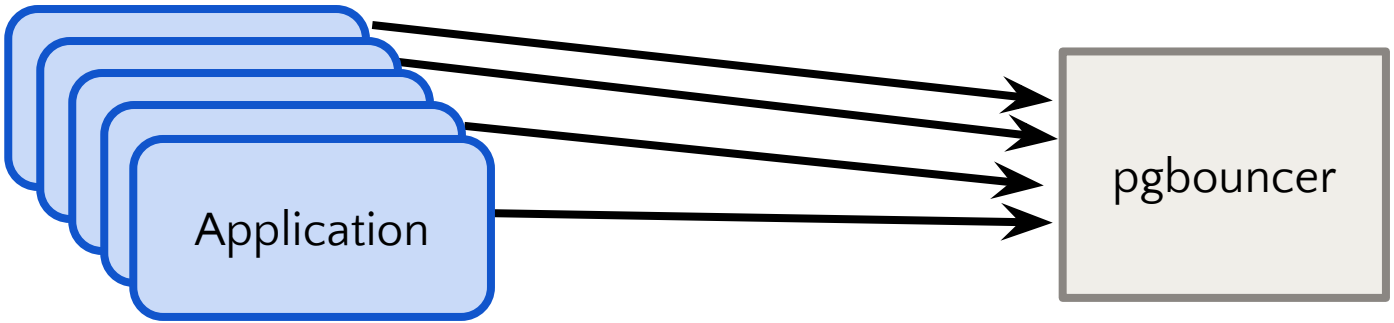




Client connections: `max_client_conn`

Hanging on connect to pgbouncer? Check file descriptors

- **In use:** `ls /proc/[pid]/fd/ | wc`
- **Allowed:** `/proc/[pid]/limits` **“Max open files”**



Transaction mode: unsupported features

- Some statements are not supported
- Others can cause problems if they leak
- https://www.pgbouncer.org/features.html
- pgbouncer will not detect this

Transaction mode: unsupported features

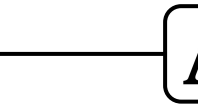
SET/RESET	Never
LISTEN	Never
WITH HOLD CURSOR	Never
PREPARE / DEALLOCATE	Never
PRESERVE/DELETE ROWS temp tables	Never
LOAD statement	Never
Session-level advisory locks	Never

Prepared statements

- **(Before 1.21.0) need to avoid**
- **(1.21.0+) Protocol-level prepared statements**
- **SQL (`PREPARE`) not supported**
- **Deallocation**
  - PQclosePrepared
  - Check language driver for support

Schema changes

- **lock_timeout**
- **Make ORM set this**
  - Ruby on Rails: Strong Migrations gem
- **SET lock_timeout = ‘10s’**
  - SET is not supported in transaction mode

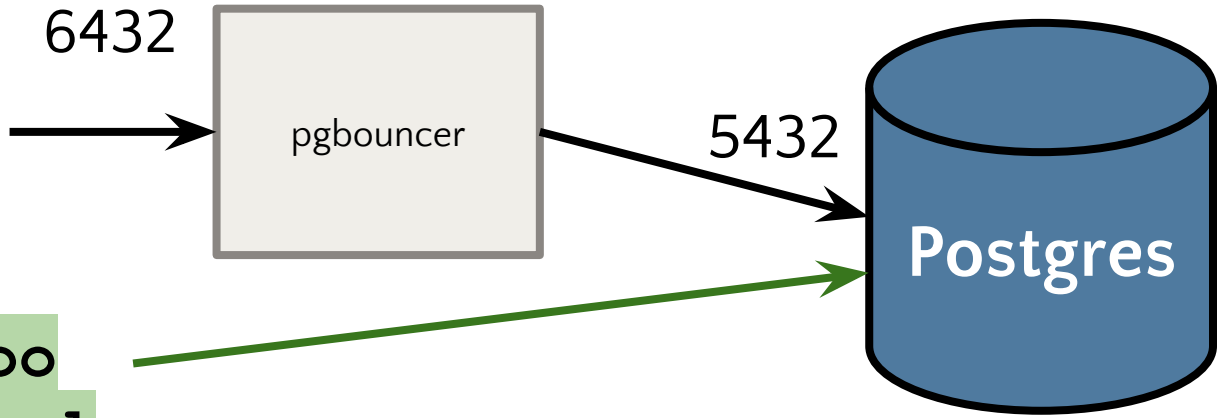


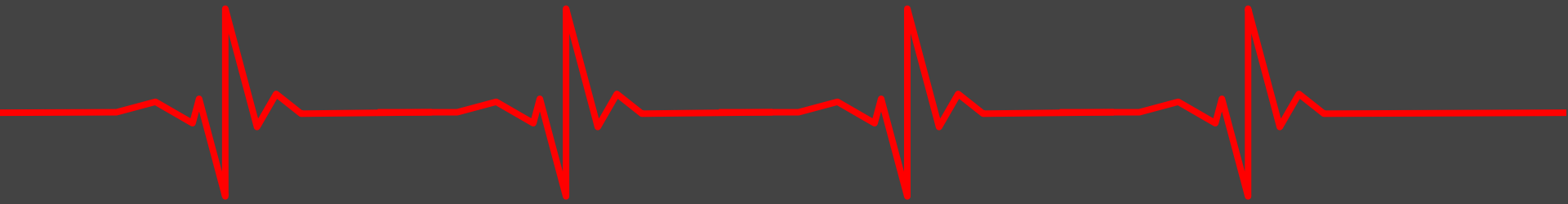
Schema changes

- Most reliable: Connect directly to postgres for DDL

```
SELECT name
FROM users
WHERE id = 1;
```

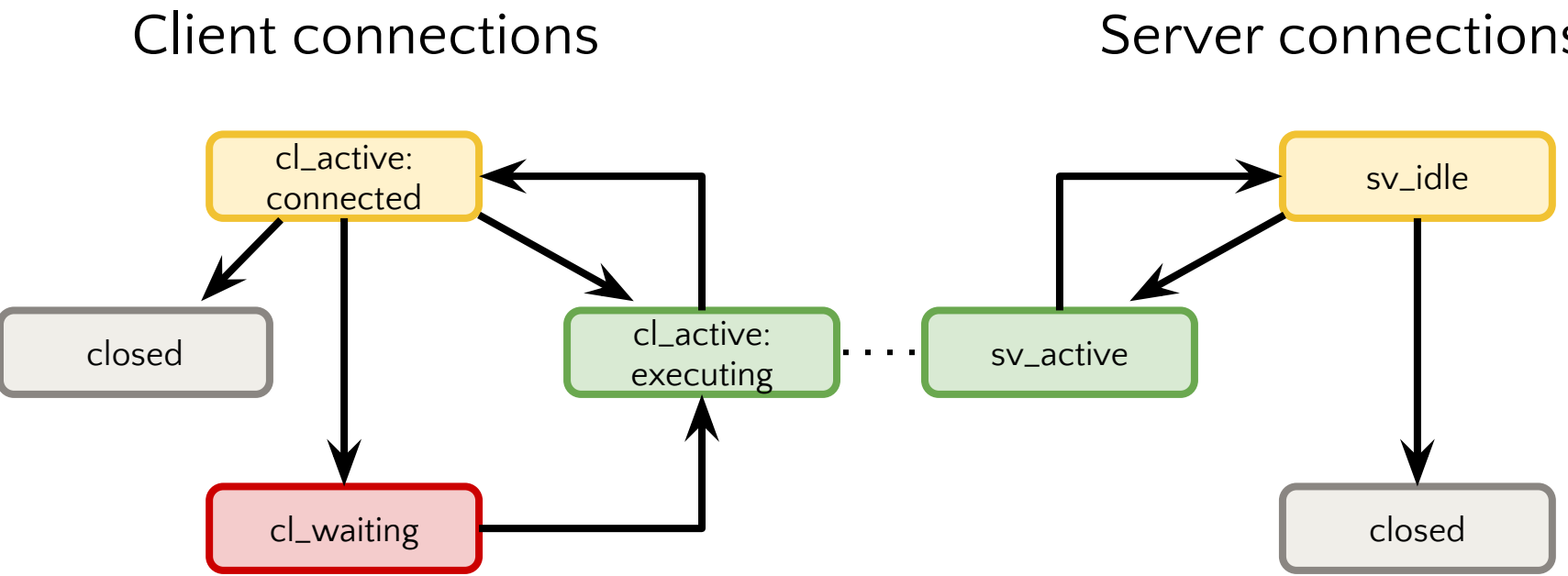
```
ALTER TABLE foo
ADD COLUMN [...]
```





Part 2
Monitoring

A PgBouncer state machine (conceptual, not literal)



Connecting to pgbouncer

- **Admin console**
- **psql -p 6432 -U pgbouncer -d pgbouncer**
  - Commands: management, or viewing statistics
  - Does not actually allow general SQL
- **You can avoid password if:**
  - User pgbouncer + DB pgbouncer
  - Connection via socket (not -h localhost or IP)
  - Client user matches user running pgbouncer

SHOW pools;

- Pool = (database, user)
- cl_active
- cl_waiting
- sv_active

```
pgbouncer=# show pools;
-[ RECORD 1 ]---------+----------
database              | pgbouncer
user                  | pgbouncer
cl_active             | 1
cl_waiting            | 0
cl_active_cancel_req  | 0
cl_waiting_cancel_req | 0
sv_active             | 0
sv_active_cancel      | 0
sv_being_canceled     | 0
sv_idle               | 0
sv_used               | 0
sv_tested             | 0
sv_login              | 0
maxwait               | 0
maxwait_us            | 0
pool_mode             | statement
load_balance_hosts    |
```

cl_active and cl_waiting

- **cl_active includes both:**
  - Actively running a query
  - Connected-but-idle, no queries waiting
    - Improved client idle state in 1.25.0, see 046505d
- **cl_waiting**
  - Sent a query
  - And: waiting for a server connection
- **Want cl_waiting to be 0**

sv_active

- Server connections linked to a client
- If it “flatlines” at pool_size, possible issue

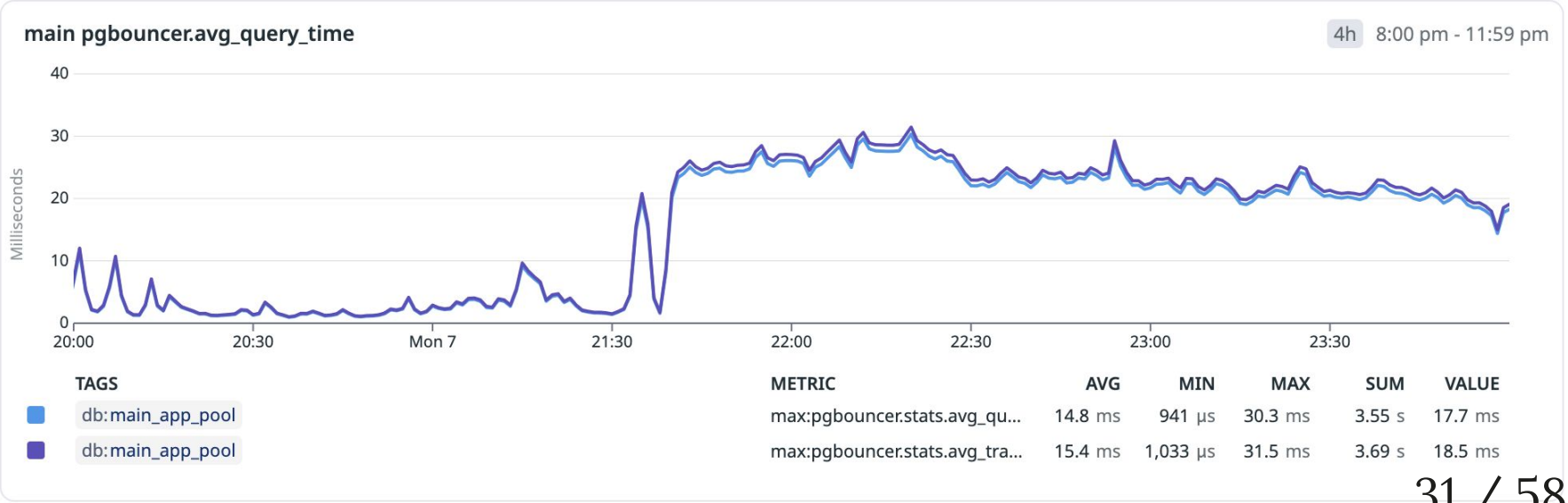
`SHOW stats;`

- SHOW stats_averages;
- avg_query_time + avg_xact_time
- avg_wait_time
- Units: microseconds

```
database|xact_count|query_count|bytes_received|bytes_sent|xact_time|query_time|wait_time
 app    |      4237 |       4364 |       2353687 |  8524081 |      793 |       694 |        0
```

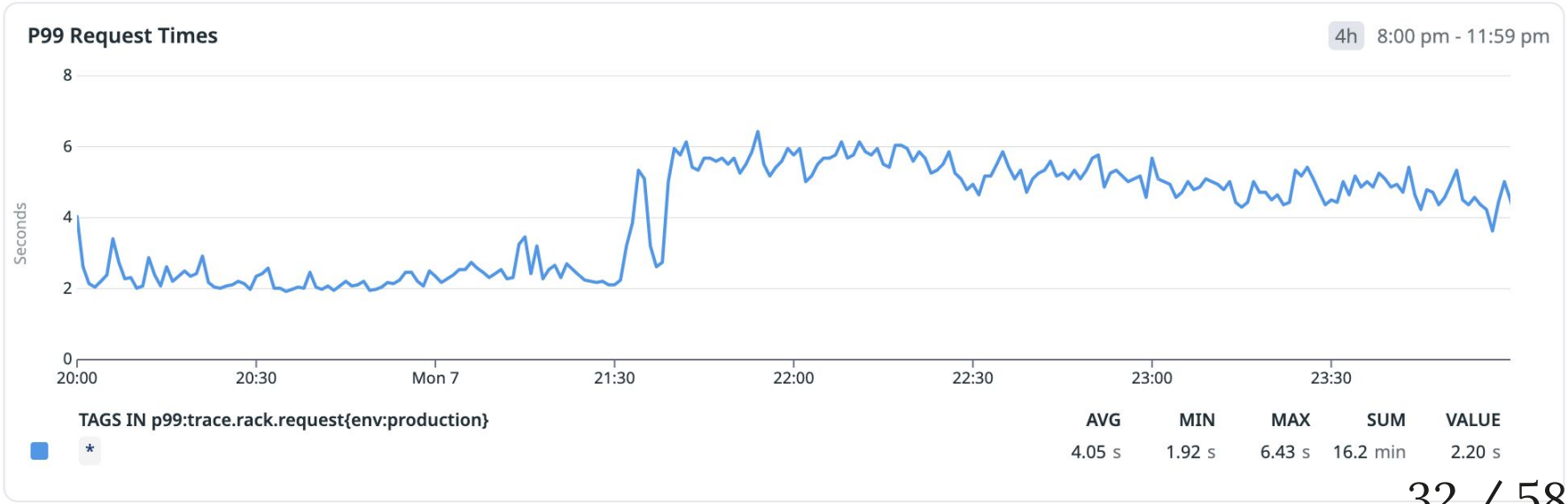
A avg_query_time + avg_xact_time

- Useful for postgres query performance



avg_query_time + avg_xact_time

- Useful for postgres query performance



So-called “avg_wait_time”

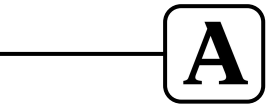
- **< 1.23.0: confusing units**
  - Number of clients is **not** in the denominator
- **>= 1.23.0: it’s actually the average wait time**
- **Either way: watch for spikes**

`SHOW clients;`

- **Interactive debugging**
- **psql [...] --csv**

link not null => has server connection

```
pgbouncer=# show clients;
-[ RECORD 1 ]-------+-----------------------
type                | C
user                | app
database            | app
replication         | none
state               | active
addr                | unix
port                | 6432
local_addr          | unix
local_port          | 6432
connect_time        | 2025-09-30 13:30:00 EDT
request_time        | 2025-09-30 13:30:05 EDT
wait                | 31
wait_us             | 912710
close_needed        | 0
ptr                 | 0x5db77e850be0
link                | 0x1c38c40
remote_pid          | 403917
tls                 |
application_name    | psql
prepared_statements | 0
id                  | 7
```



PgBouncer process CPU

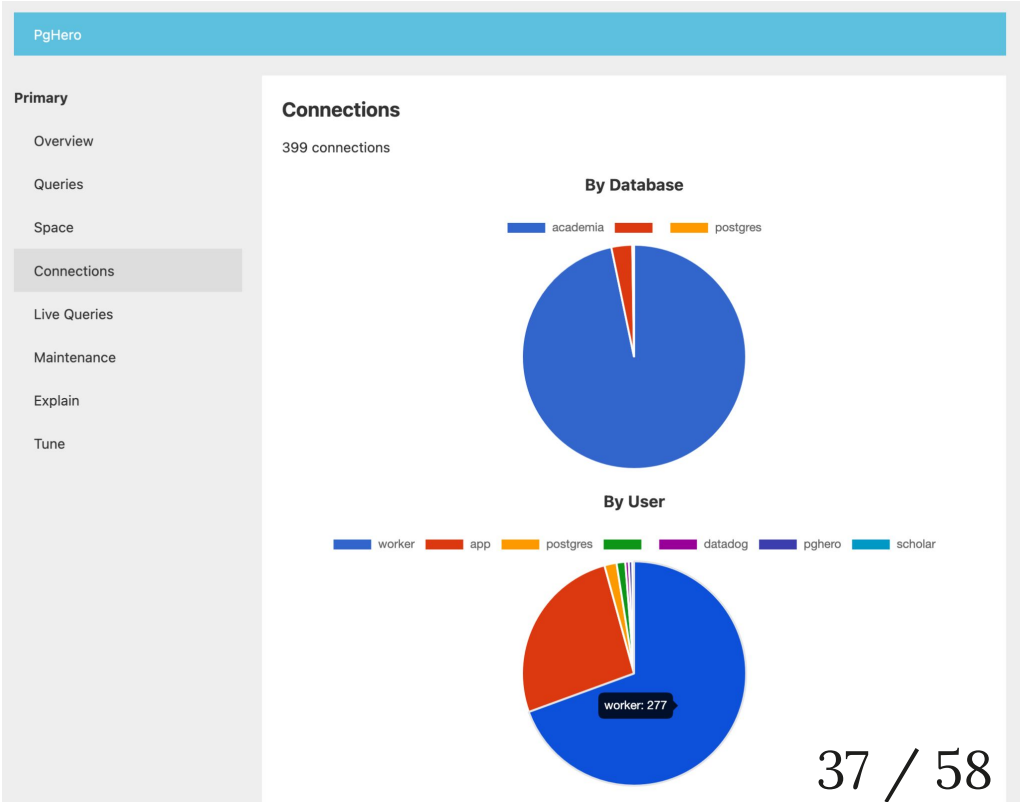
- PgBouncer is single-threaded
- `SHOW stats;` does not show CPU util
- top
- `pidstat -p $PID 1 1`
- If it hits 100%, other metrics start to look confusing

Postgres monitoring - Queries

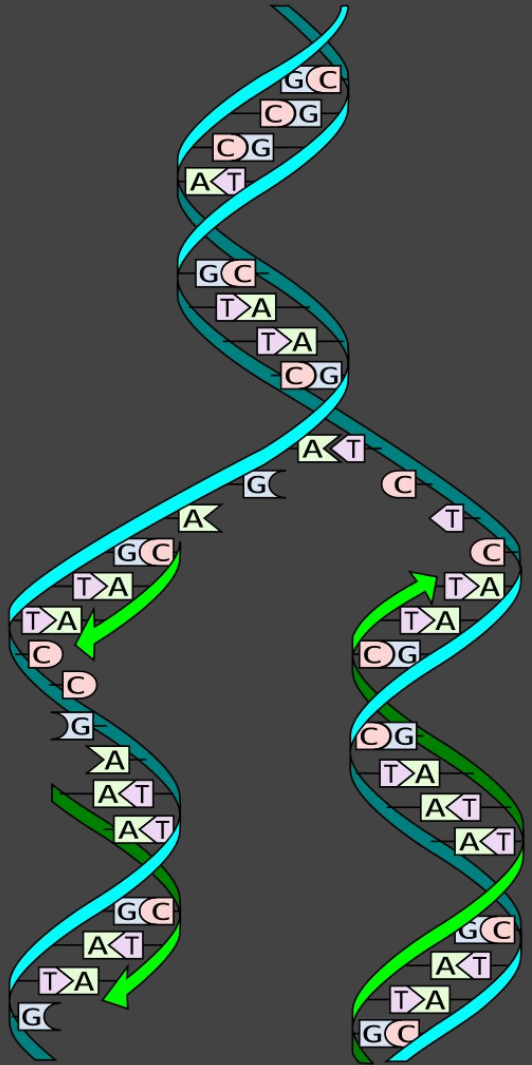
- **Live queries**
  - pg_stat_activity
  - idle in transaction count
- **Queries over time**
  - pg_stat_statements
- **Wait events**
  - pg_wait_sampling
  - Datadog database monitoring
  - AWS RDS performance insights

Postgres monitoring - PgHero

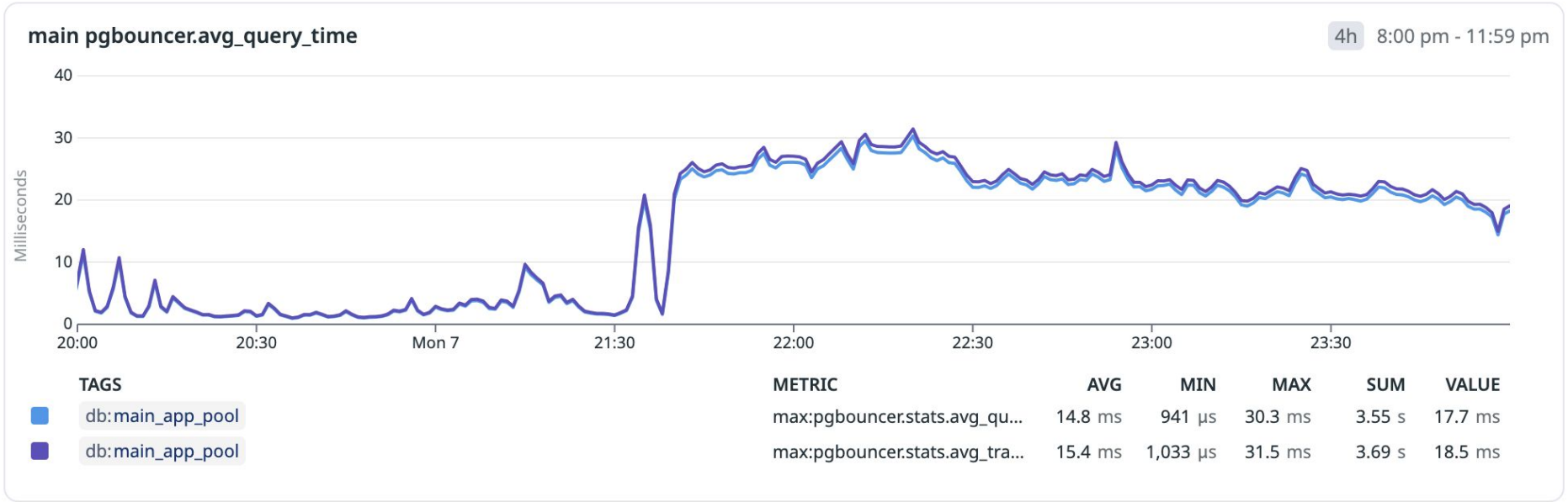
- Connections
- Live Queries



Part 3
Scaling

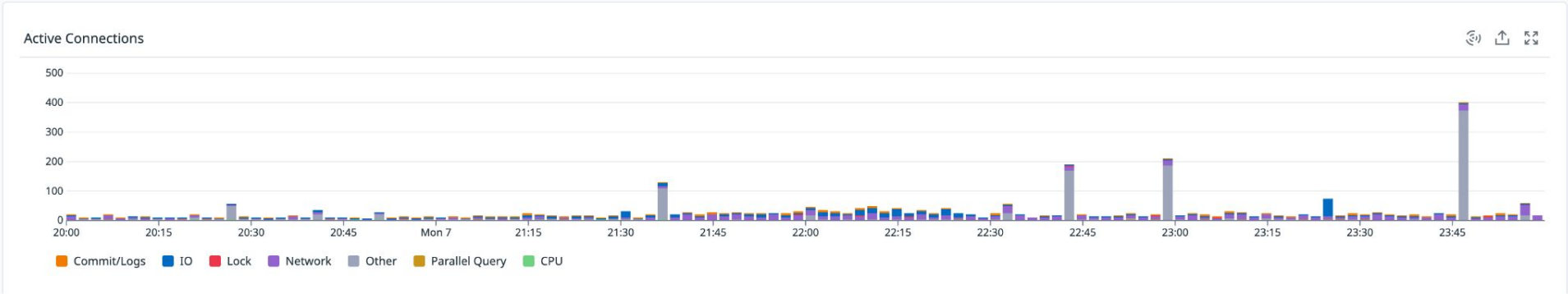


A PgBouncer struggling, but Postgres is fine?



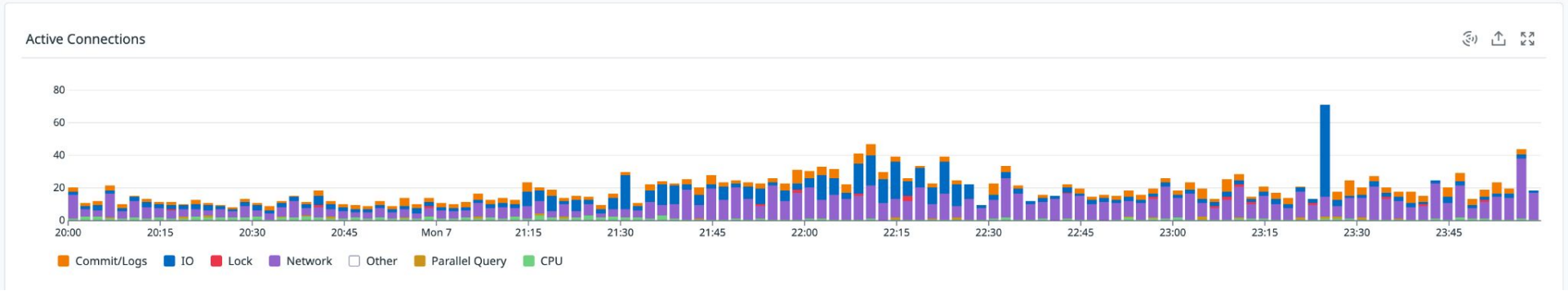
PgBouncer struggling, but Postgres is fine?

- Wait events – not the same graph



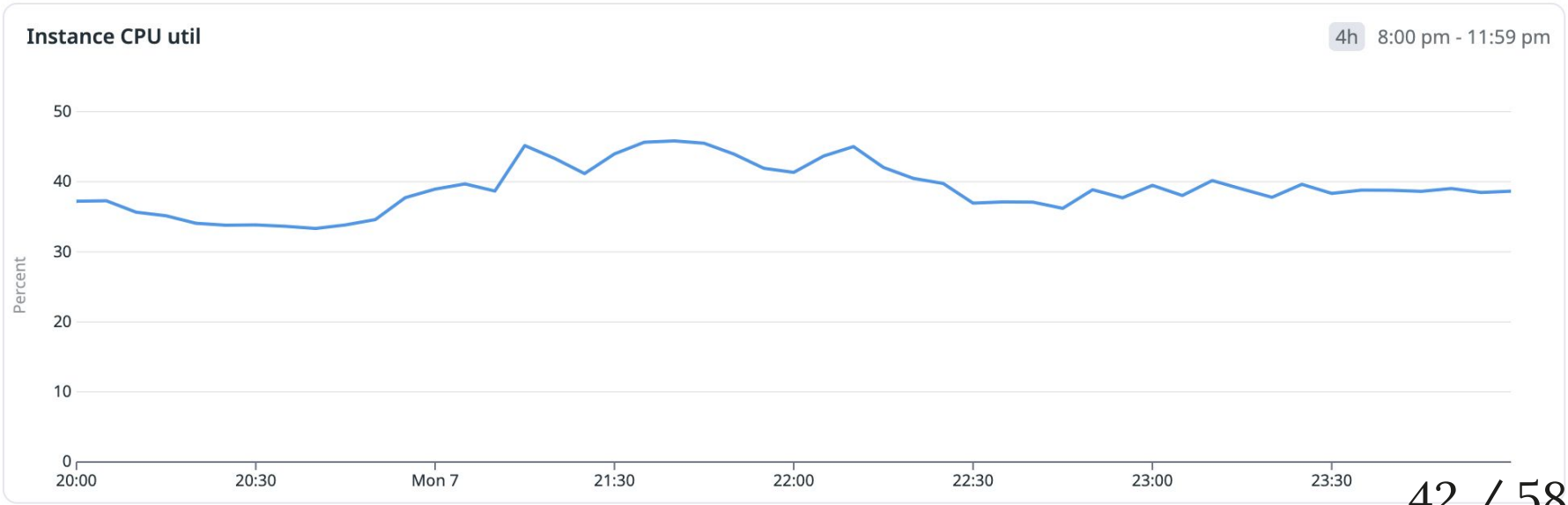
PgBouncer struggling, but Postgres is fine?

- Wait events – not the same graph



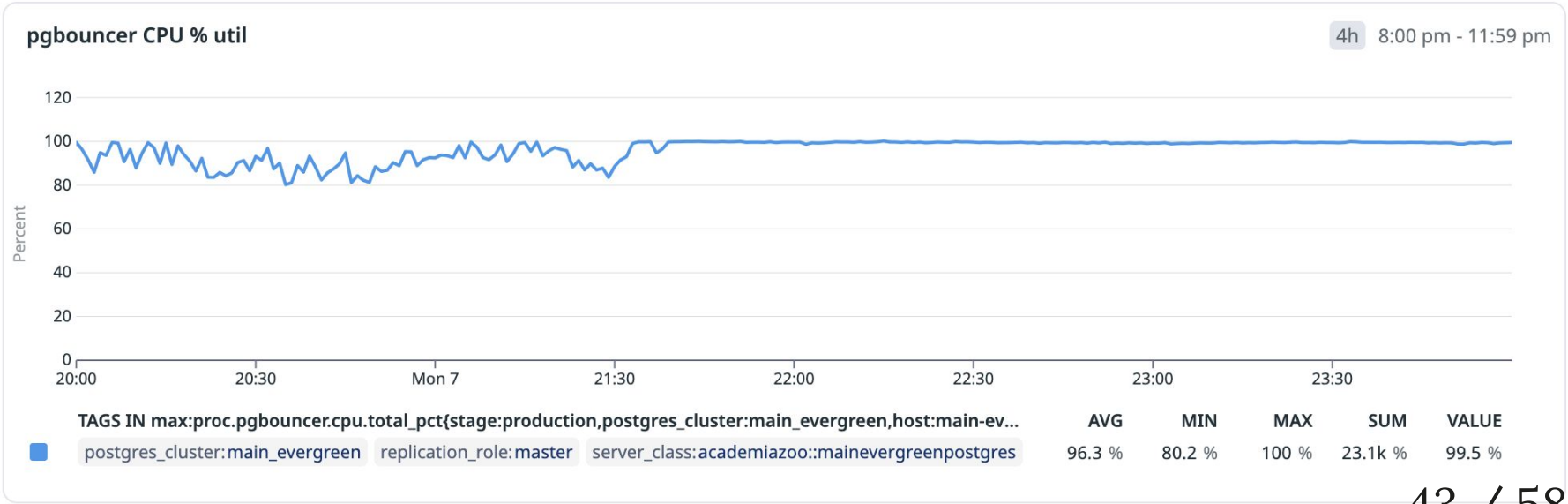
PgBouncer struggling, but Postgres is fine?

- Instance CPU util not elevated



PgBouncer struggling, but Postgres is fine?

- pgbouncer CPU at 100%



Signs that you need multiple PgBouncer processes

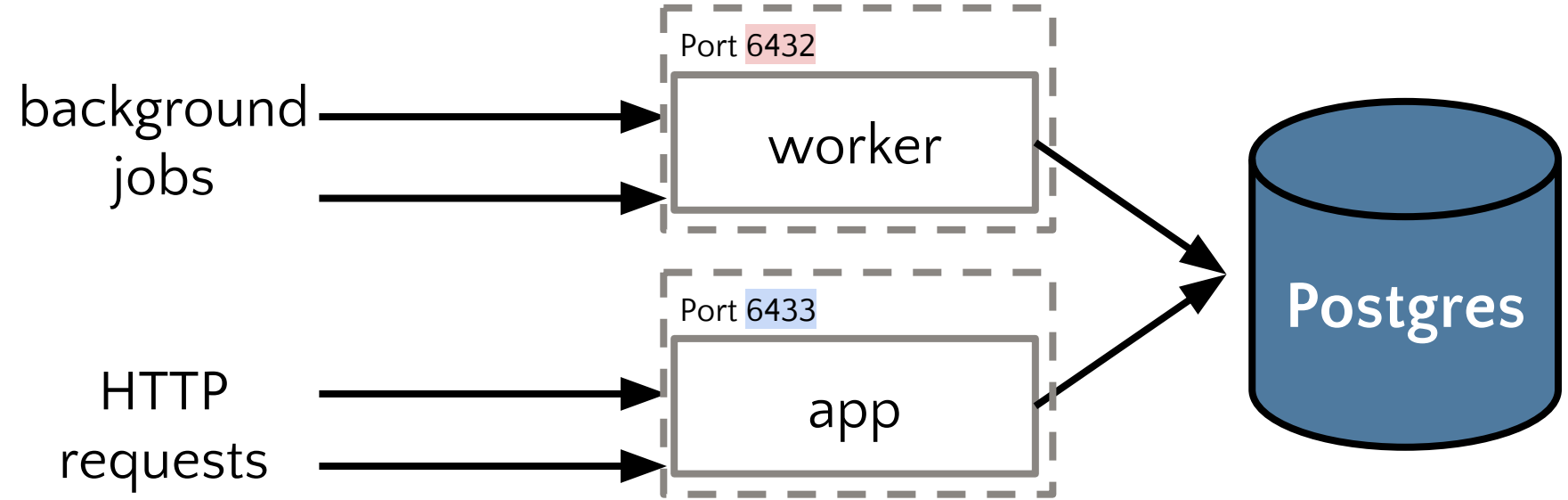
You *will* see:

- PgBouncer 100% CPU
- Application sees high end-to-end query times

You *might* see:

- High avg_query_time and/or avg_wait_time
- High cl_waiting
- Low/stable active sessions on postgres

Multiple PgBouncers: by user



Multiple PgBouncers: sharing the same port

(normally) If you try to listen on 6432 on another process:

```
$ pgbouncer /etc/pgbouncer/pgbouncer.ini
FATAL unix socket is in use, cannot continue
```

Multiple PgBouncers: sharing the same port

- **SO_REUSEPORT**
- **Load balancing from the networking stack**
  - Hash of (remote ip, remote port, local ip, local port)
- **Two ways of doing this:**
  - In pgbouncer.ini
  - With systemd + socket activation

1. pgbouncer.ini (and pgbouncer2.ini, etc...)

- In the pgbouncer config:

```
so_reuseport = 1
```

- Then copy the config file, run another process
- Separate config file (and init config) for every process

1. pgbouncer.ini (and pgbouncer2.ini, etc...)

```
[databases]
db = host=[...] pool_size={{total/N}}
[pgbouncer]
logfile = /path/to/pgbouncer{{i}}.log
pidfile = /path/to/pgbouncer{{i}}.pid
unix_socket_dir = /path/to/pgbouncer{{i}}
so_reuseport = 1
```

A 2. systemd template + socket activation

- **No so_reuseport = 1 in the pgbouncer config**
- **Instead systemd handles:**
  - Logfile
  - “pidfile”
  - socket / socket directory
  - so_reuseport
- **https://www.enterprisedb.com/blog/running-multiple-pgbouncer-instances-systemd**

2. systemd template + socket activation

```
[Unit]
Description=sockets for PgBouncer
[Socket]
ListenStream=6432
ListenStream=/var/run/postgresql/.s.PGSQL.%i
ReusePort=true
[Install]
WantedBy=sockets.target
```

2. systemd template + socket activation

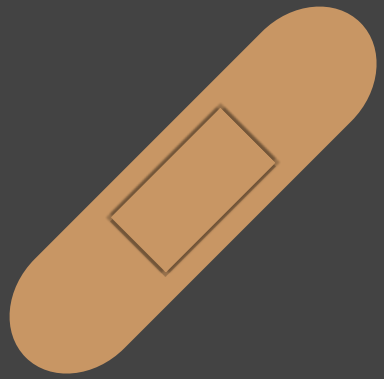
```
[Unit]
[...]
Requires=remote-fs.target pgbouncer@%i.socket
[Service]
[...]
LimitNOFILE=65535
[Install]
WantedBy=multi-user.target
```

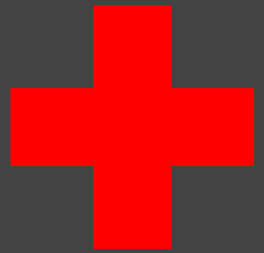
2. systemd template + socket activation

```
systemctl start pgbouncer@50001
systemctl start pgbouncer@50002
[...]
```

Multiple PgBouncer tips

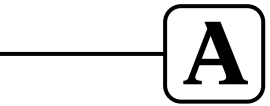
- **Check all things that depend on pgbouncer**
  - Monitoring
  - High availability solution
  - Upgrade process
  - Sysadmin scripts and playbooks
- **Check connection counts to each**
- **Confirm client query times**





Summary

A



Monitor PgBouncer process CPU util

- PgBouncer is single threaded
- CPU at 100% => bottleneck
- Might not be “out of the box” with monitoring solution

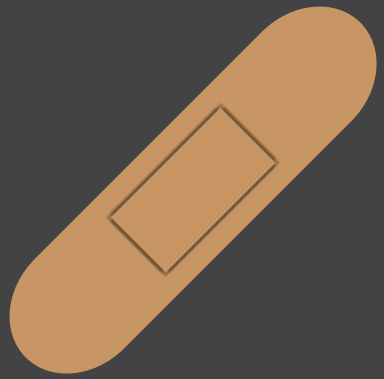
Do we need PgBouncer alternatives?

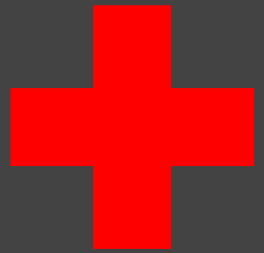
- **Managing multiple processes can be a pain**
- **Transaction mode has limitations**
- **A lot of pain points have been fixed in recent versions**
  - avg_wait_time
  - (protocol) prepared statements

Summary

- **Monitor pgbouncer process CPU util**
  - If pgbouncer hits 100%, run more
- **A lot of fixes, enhancements in latest pgbouncer**
- **Be aware of transaction mode limitations**

Questions?





Appendix

A

References

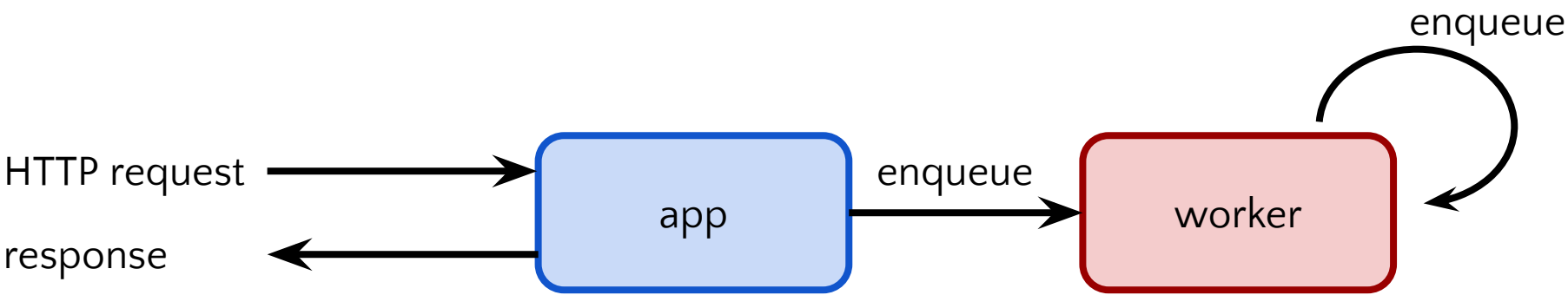
- https://www.heap.io/blog/decrypting-pgbouncers-diagnostic-information
- https://jpcamara.com/2023/04/12/pgbouncer-is-useful.html
- https://www.crunchydata.com/blog/postgres-at-scale-running-multiple-pgbouncers
- https://www.enterprisedb.com/blog/running-multiple-pgbouncer-instances-systemd
- https://www.crunchydata.com/blog/prepared-statements-in-transaction-mode-for-pgbouncer

A Image credits

- **https://en.wikipedia.org/wiki/File:Rhinovirus_isosurface.png CC BY-SA**
  - Author: Thomas Splettstoesser
- **https://en.wikipedia.org/wiki/File:DNA_replication_split.svg CC0**
  - Author: Madprime

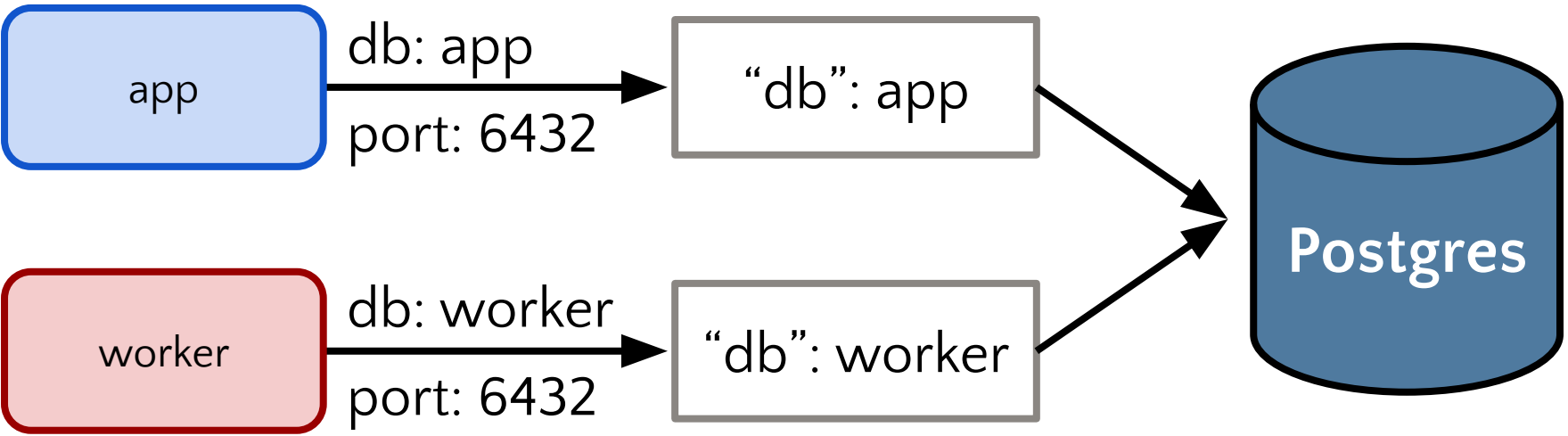
Academia.edu uses multiple “databases”

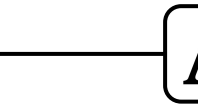
```
[databases]
app    = host= port= dbname=db pool_size=A
worker = host= port= dbname=db pool_size=B
```



Academia.edu uses multiple “databases”

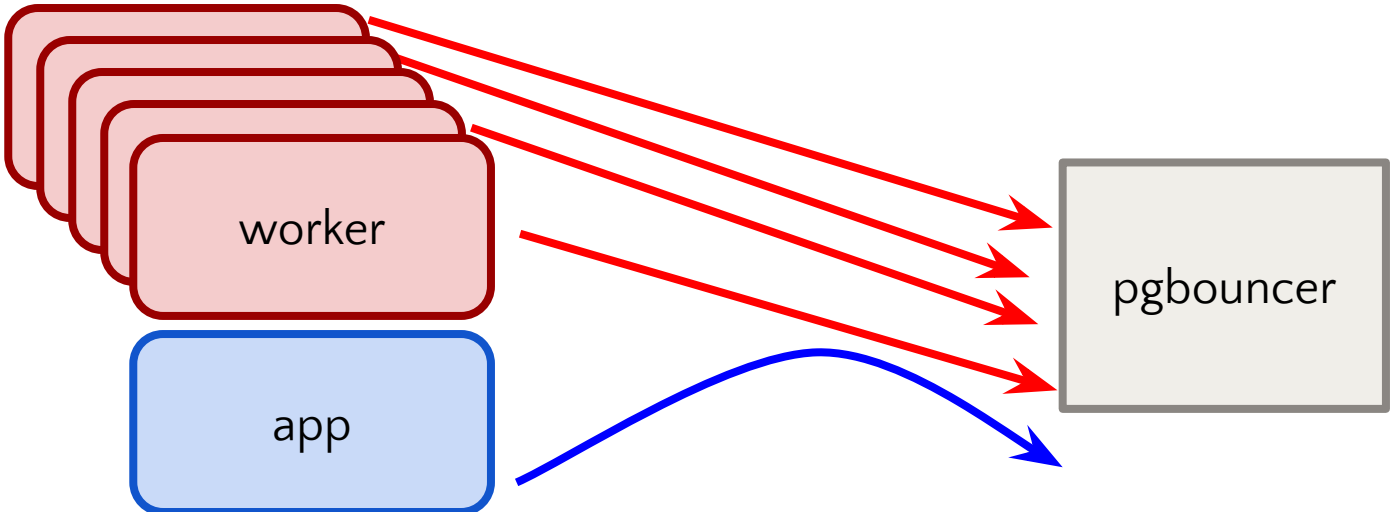
- Separate “app” and “worker” with different pool sizes





Client connections (multiple workloads)

- max_db_client_connections (1.24.0+)
- max_user_client_connections (1.24.0+)

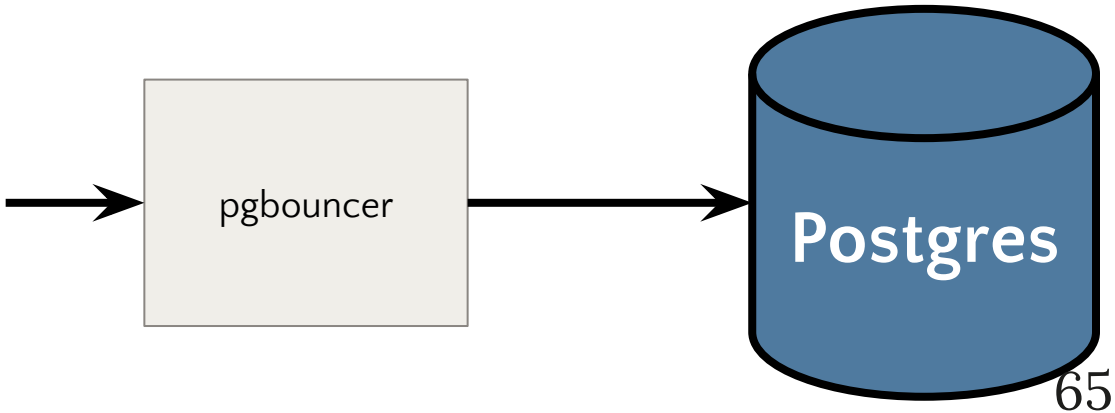


Schema changes - other approaches

In theory: Transactions + `SET LOCAL`

- **Still doesn't help with** CREATE INDEX CONCURRENTLY
- **Your ORM probably does not support this**

```
BEGIN;
SET LOCAL lock_timeout = '10s';
ALTER TABLE foo ADD COLUMN [...]
COMMIT;
```



Schema changes - other approaches

Add a database with session pool_mode, use that

```
[databases]
db         = host= port= dbname=
migrations = host= port= dbname= pool_mode=session
[pgbouncer]
pool_mode = transaction
(...)
```

Info about sockets

- **socket = abstraction over network communication, file**
- **Connection socket**
  - remote ip, remote port, local ip, local port, protocol (TCP|UDP)
- **Listening socket**
  - Bind: (protocol, ip, port) -> process