Практики, особенности и нюансы при работе с Postgres в Go

Артемий Рябинков

Tools

database/sql

provides a generic interface around SQL (or SQL-like)

database/sql

sql.**Open**(driver, url string) (*DB, error)

db.**ExecContext**(...) (Result, error)

db.**QueryContext**(...) (*Rows, error)

database/sql

Working with Transactions

db.**BeginTx**(...) (*Tx, error)

tx.**Commit**() error

tx.**Rollback**() error

go-database-sql.org

```
var (
    id int
    name string
    desc string
    updated_at time.Time
    created_at time.Time
)

rows.Scan(&id, &name, &desc, &updated_at, &created_at)
```

github.com/jmoiron/sqlx

general purpose **extensions** to golang's database/sql

github.com/jmoiron/sqlx — Marshal Rows into Structs

```
type Place struct {
    Country       string
    City          sql.NullString
    TelephoneCode int `db:"telcode"`
}

var p Place
err = rows.StructScan(&p)
```

github.com/jmoiron/sqlx — **Bindvars**

Bindvars are database specific

MySQL uses the **?** variant shown above

PostgreSQL uses an enumerated **$1**, **$2**

SQLite accepts both **?** and **$1** syntax

Oracle uses a **:name** syntax

github.com/jmoiron/sqlx — **Bindvars**

```
`.. WHERE country=(? or $1 or :name)`
```

github.com/jmoiron/sqlx — **Bindvars**

```
sqlx.DB.Rebind(`.. WHERE country=?`)
```

github.com/jmoiron/sqlx — **Named Params**

```
p := Place{Country: "South Africa"}

sql := `.. WHERE country=:country`
rows, err := db.NamedQuery(sql, p)
```

github.com/jmoiron/sqlx — **Get** and **Select**

```
var p Place
var pp []Place

// pull the first place directly into p
err = db.Get(&p, ".. LIMIT 1")

// pull places with telcode > 50 into the slice pp
err = db.Select(&pp, ".. WHERE telcode > ?", 50)
```

github.com/lib/pq — pure Go Postgres driver for database/sql

github.com/jackc/pgx - PostgreSQL driver and toolkit for Go

github.com/jackc/pgx — Features

Perfomance (63% faster than lib/pq)

Support for ~60 different PostgreSQL **types**

Extendable **logging** support

Binary format support for custom types

No panics in code

Arbitrary connection setup

Logical replication connections

Driver Insides

Default read buffer - **4096 bytes** per connection

```
rows, err := s.db.QueryContext(ctx, sql)

for rows.Next() {
    err = rows.Scan(...)
}
```

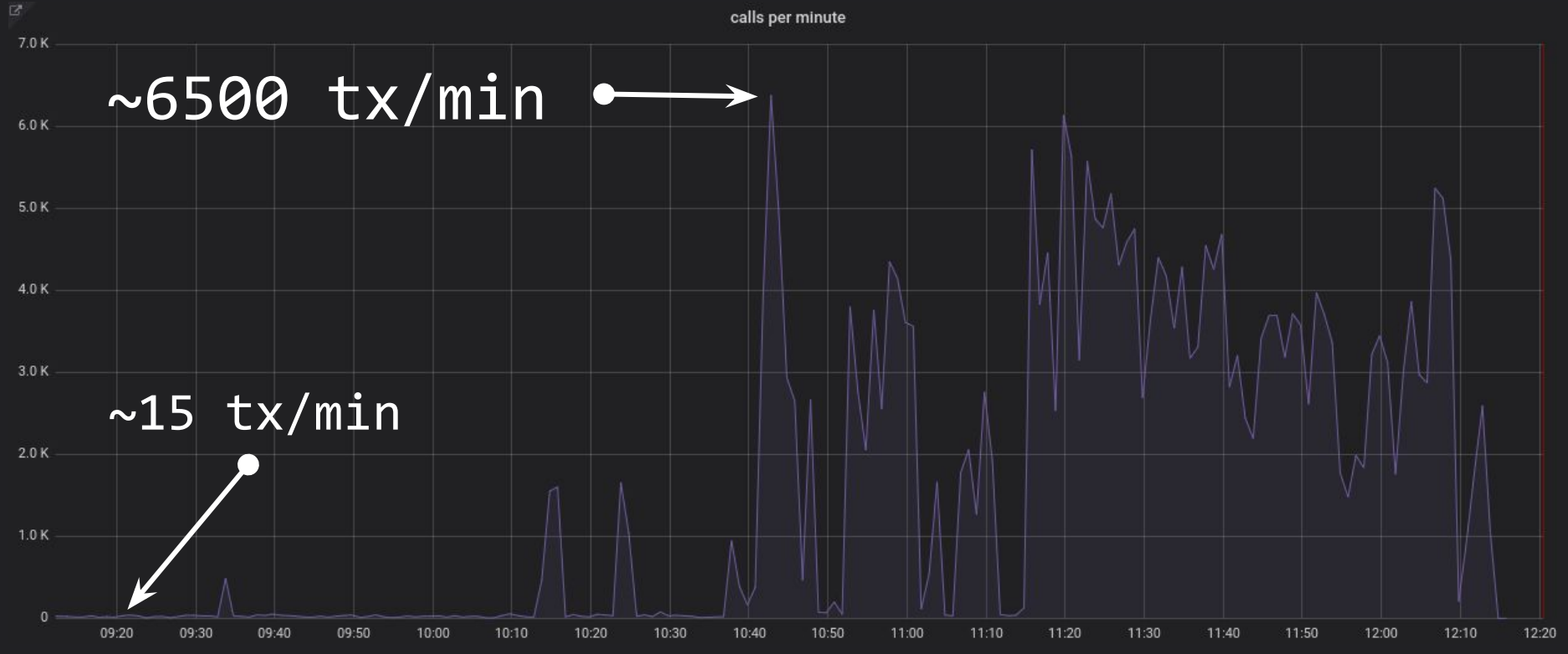
github.com/jackc/pgx receive list of OID on connection establishing

```
map[string]Value{
    "_aclitem": 2,
    "_bool": 3,
    "_int4": 4,
    "_int8": 55,
    ...
}
```

github.com/jackc/pgx receive list of OID on

each

connection establishing


calls per minute
~6500 tx/min
~15 tx/min
7.0 K
6.0 K
5.0 K
4.0 K
3.0 K
2.0 K
1.0 K
0
09:20
09:30
09:40
09:50
10:00
10:10
10:20
10:30
10:40
10:50
11:00
11:10
11:20
11:30
11:40
11:50
12:00
12:10
12:20

Pool settings configure

```
db.SetMaxIdleConns(n int)

db.SetMaxOpenConns(n int)
```

Importance of Monitoring and Logging

database/sql#DB.Stats() DBStats

```go
type DBStats struct {
        MaxOpenConnections int

        // Pool Status
        OpenConnections int
        InUse           int
        Idle            int

        // Counters
        WaitCount         int64
        WaitDuration      time.Duration
        MaxIdleClosed     int64
        MaxLifetimeClosed int64
}
```

Importance of Monitoring and Logging

github.com/jackc/pgx#ConnPool.Stat() ConnPoolStat

```
type ConnPoolStat struct {
    MaxConnections       int
    CurrentConnections   int
    AvailableConnections int
}
```

Importance of Monitoring and Logging

https://godoc.org/github.com/jackc/pgx#Logger

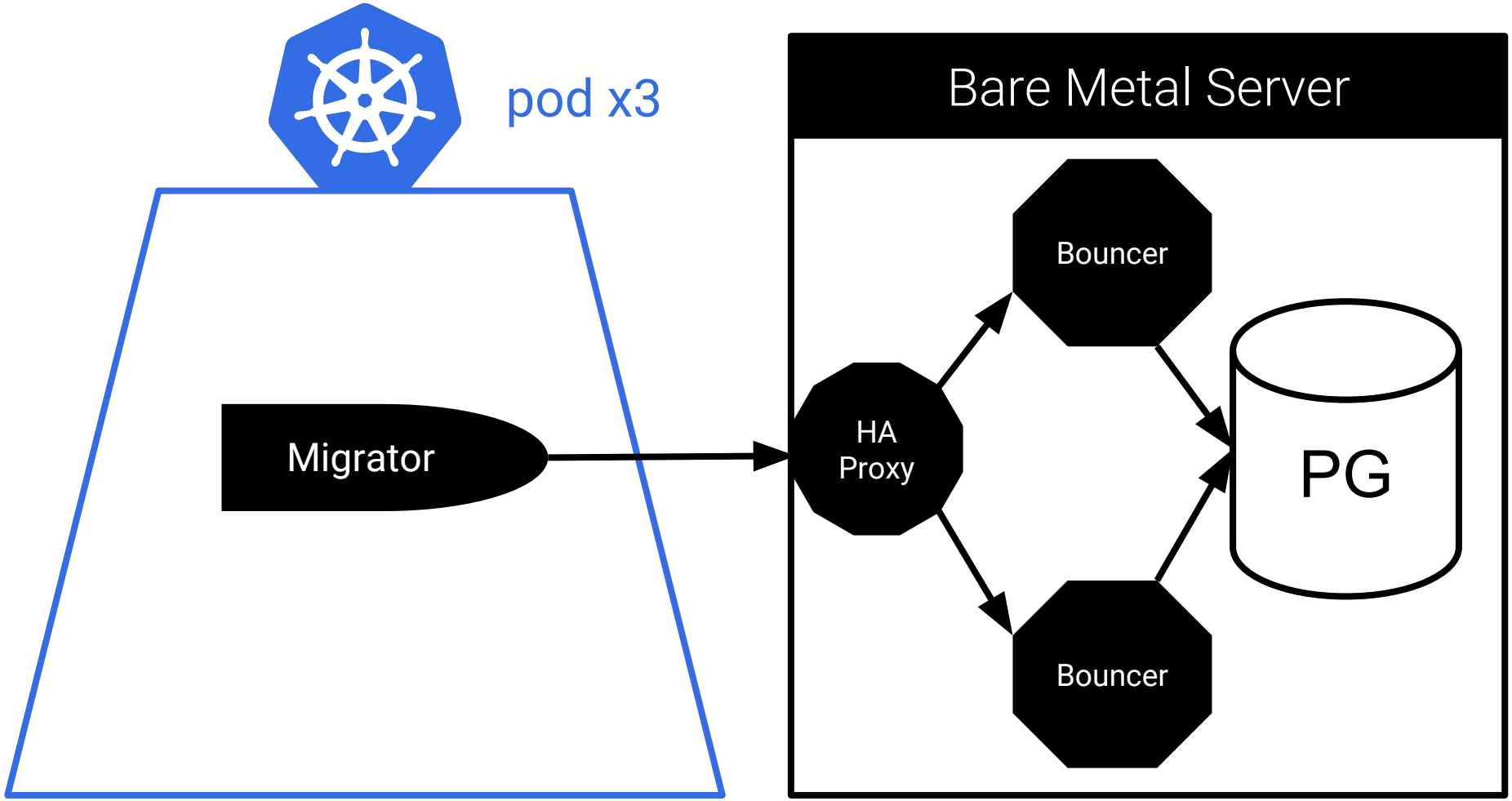
```
type Logger interface {
    // Log a message at the given level with data key/value pairs.
    // data may be nil.
    Log(level LogLevel, msg string, data map[string]interface{})
}
```

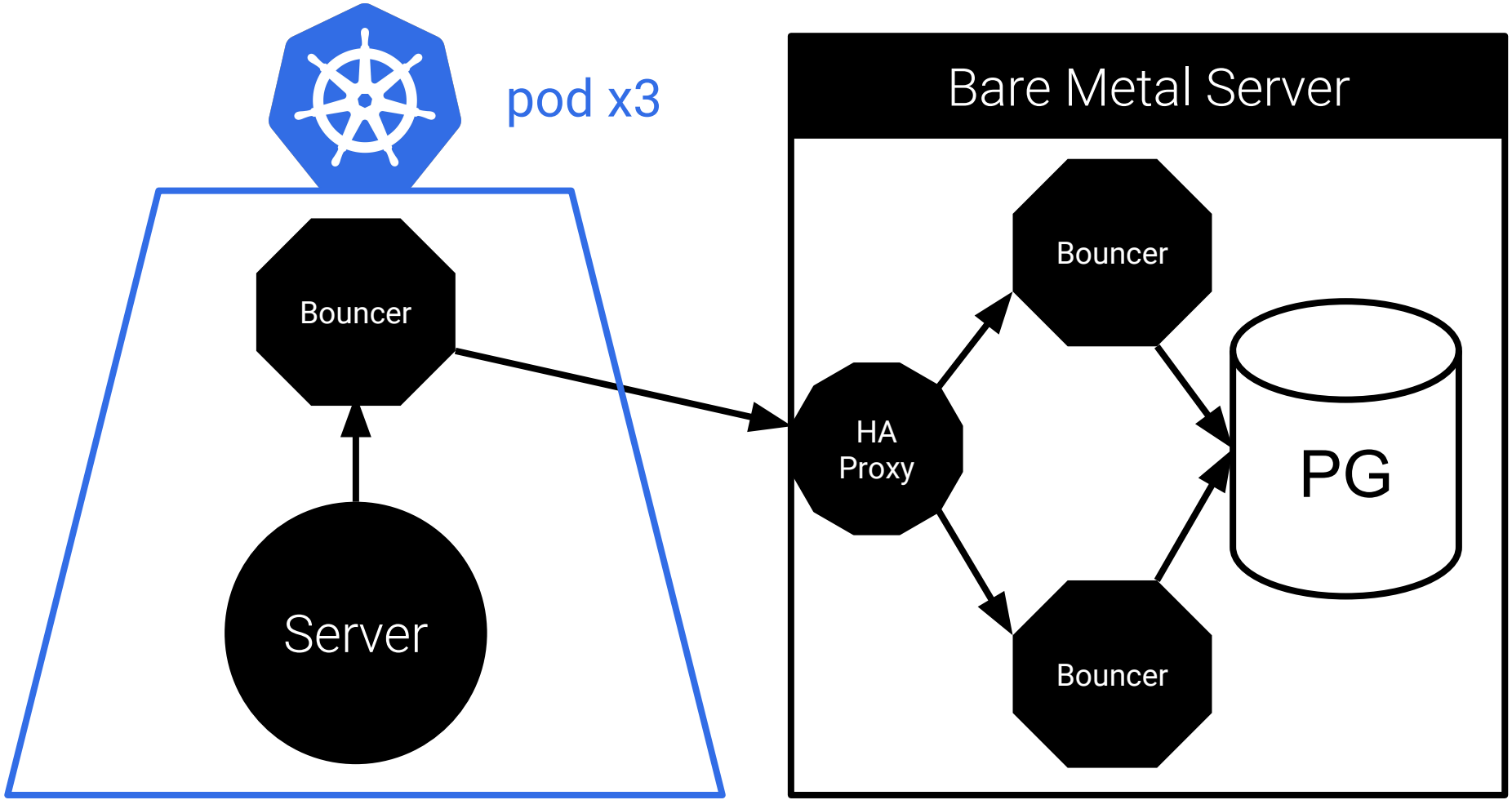
Cache of OID’s as a Kludge

```
github.com/jackc/pgx/stdlib.OpenDB(pgx.ConnConfig{
    CustomConnInfo: func(c *pgx.Conn) (*pgtype.ConnInfo, error) {
        cachedOids := Fetch OIDs from cache here

        info := pgtype.NewConnInfo()
        info.InitializeDataTypes(cachedOids)

        return info, nil
    }
})
```

PgBouncer

lightweight connection pooler for PostgreSQL


pod x3
Migrator
Bare Metal Server
HA
Proxy
Bouncer
Bouncer
PG


pod x3
Bouncer
Server
Bare Metal Server
Bouncer
HA
Proxy
PG
Bouncer

PgBouncer pooling **modes**

Session pooling

Transaction pooling

Statement pooling

https://wiki.postgresql.org/wiki/PgBouncer#Feature_matrix_for_pooling_modes

PgBouncer pooling modes

Session pooling

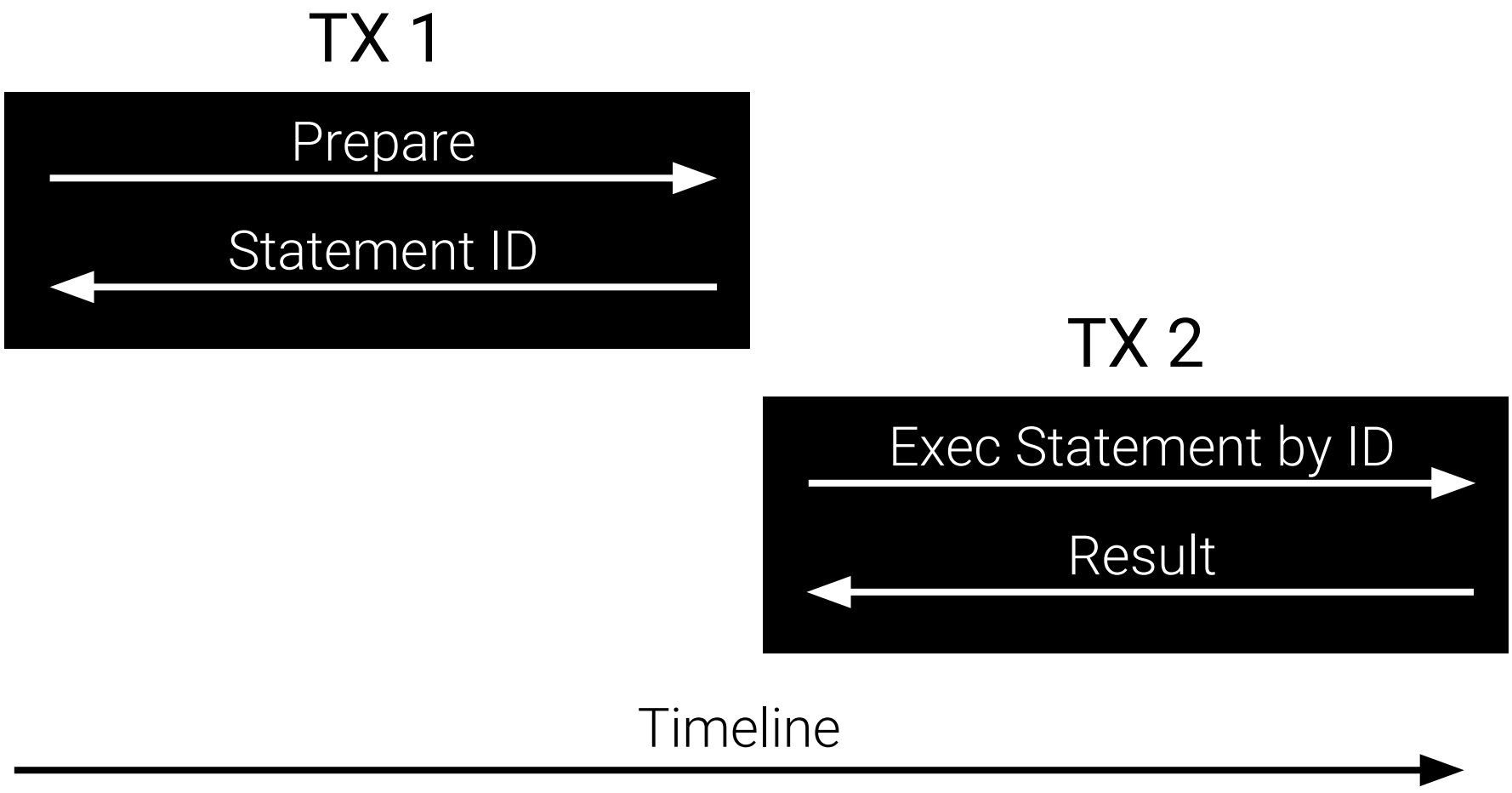
Transaction pooling

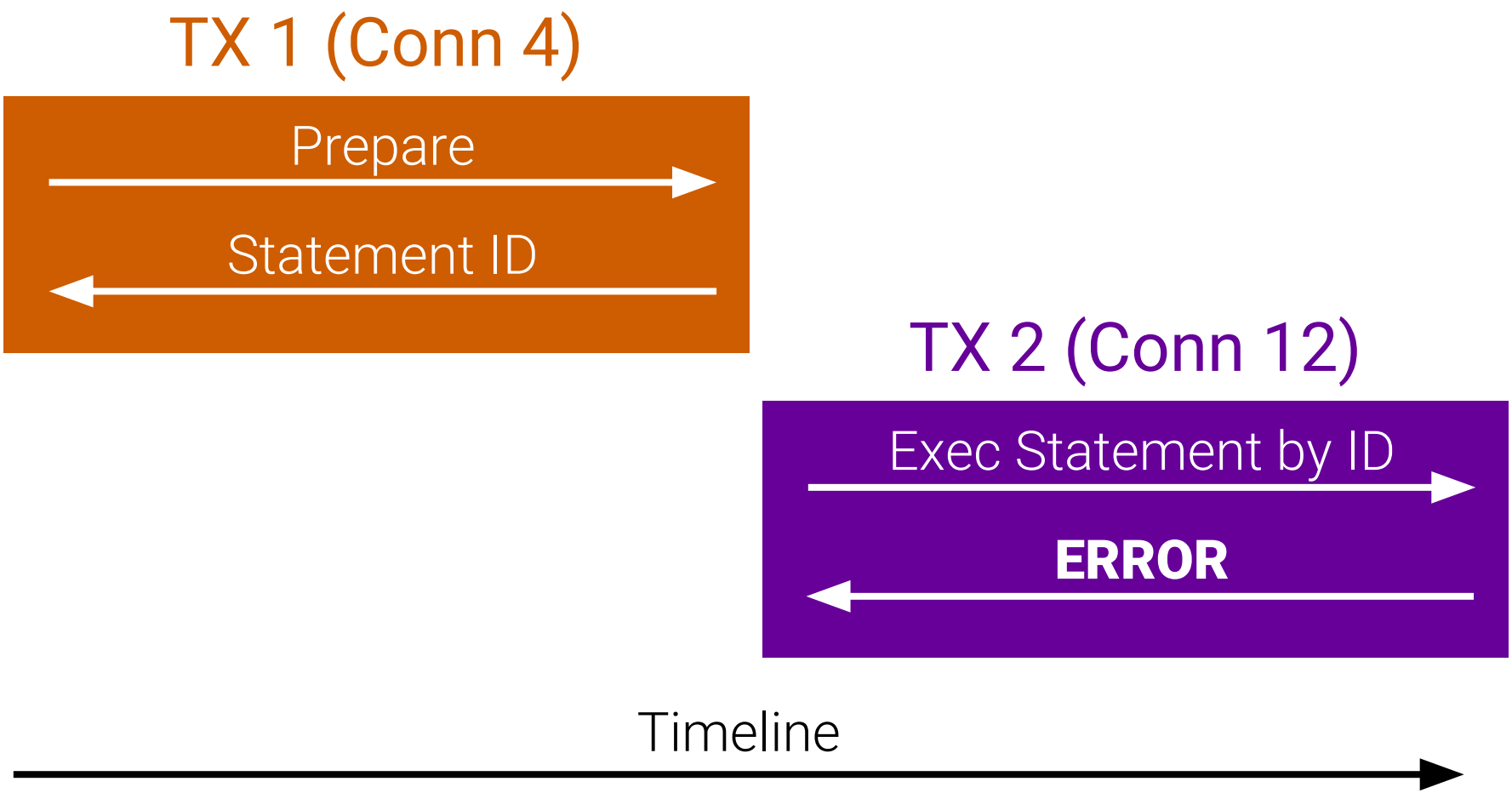
Statement pooling

Transaction Pooling

+

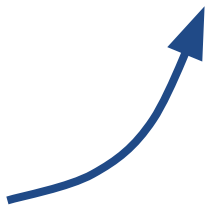
Prepared Statements


TX 1
Prepare
Statement ID
TX 2
Exec Statement by ID
Result
Timeline


TX 1 (Conn 4)
Prepare
Statement ID
TX 2 (Conn 12)
Exec Statement by ID
ERROR
Timeline

What a problem?

```
sql := `select * from places where city = ?`

rows, err := s.db.QueryContext(ctx, sql)
```



Underlying Prepared Statement here!

How it works in PHP **for years**?

```
PDO::ATTR_EMULATE_PREPARES => true
```

+

```
pg_escape_literal
```

How to make it work in Go?

Session pooling

Inefficient — low connection utilization

Client-side preparation

Potential **SQL-injection**

More code

Wrap in Transaction

Inefficient — a lot of network requests

More code

Parameters Binary Representation

https://www.postgresql.org/docs/current/protocol-overview.html#PROTOCOL-FORMAT-CODES

Enable Binary Params for github.com/jackc/pgx

```
cfg := pgx.ConnConfig{
   PreferSimpleProtocol: true,
}

db := github.com/jackc/pgx/stdlib.OpenDB(cfg)
```

Binary Params

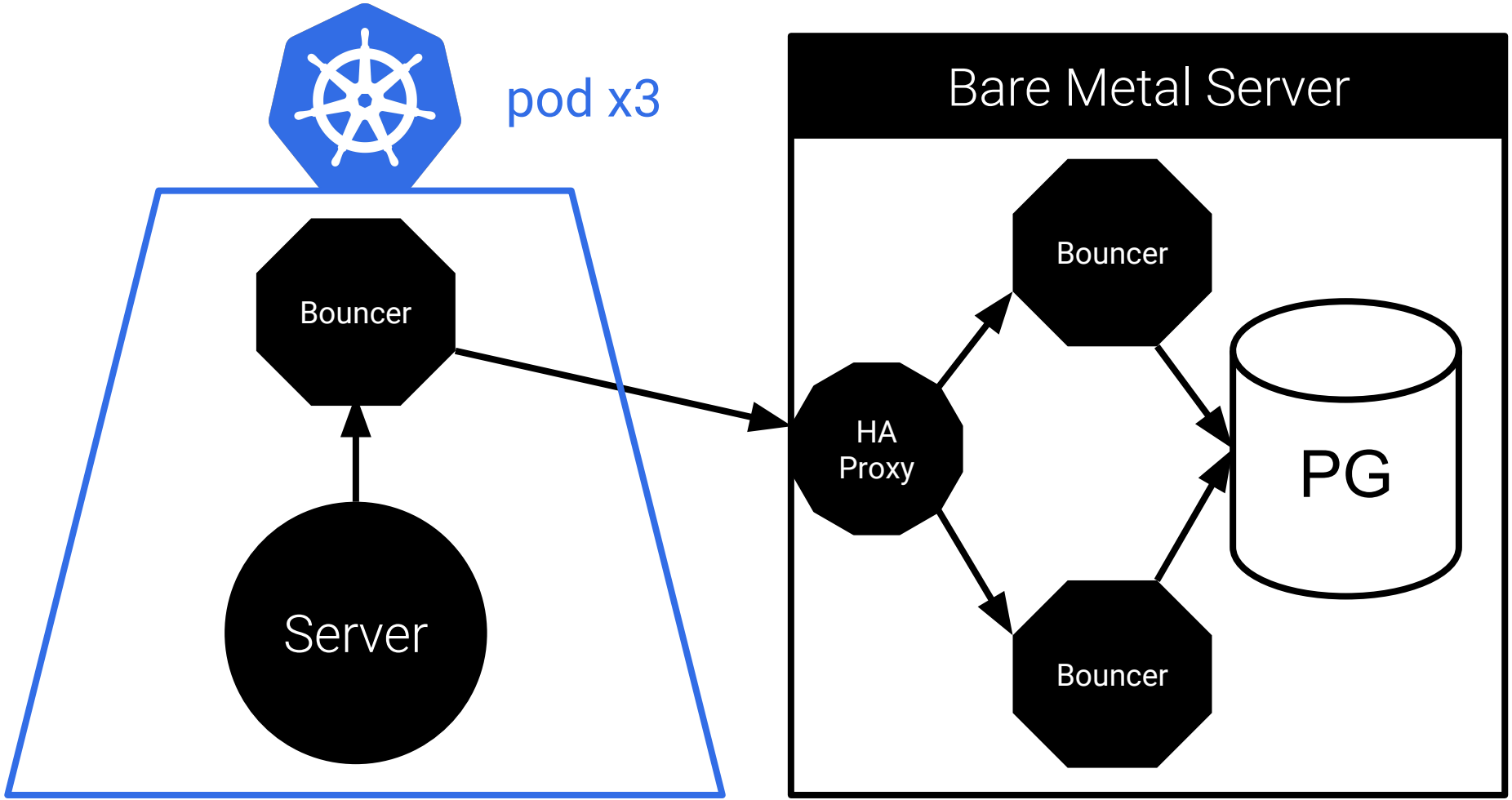
Simplicity

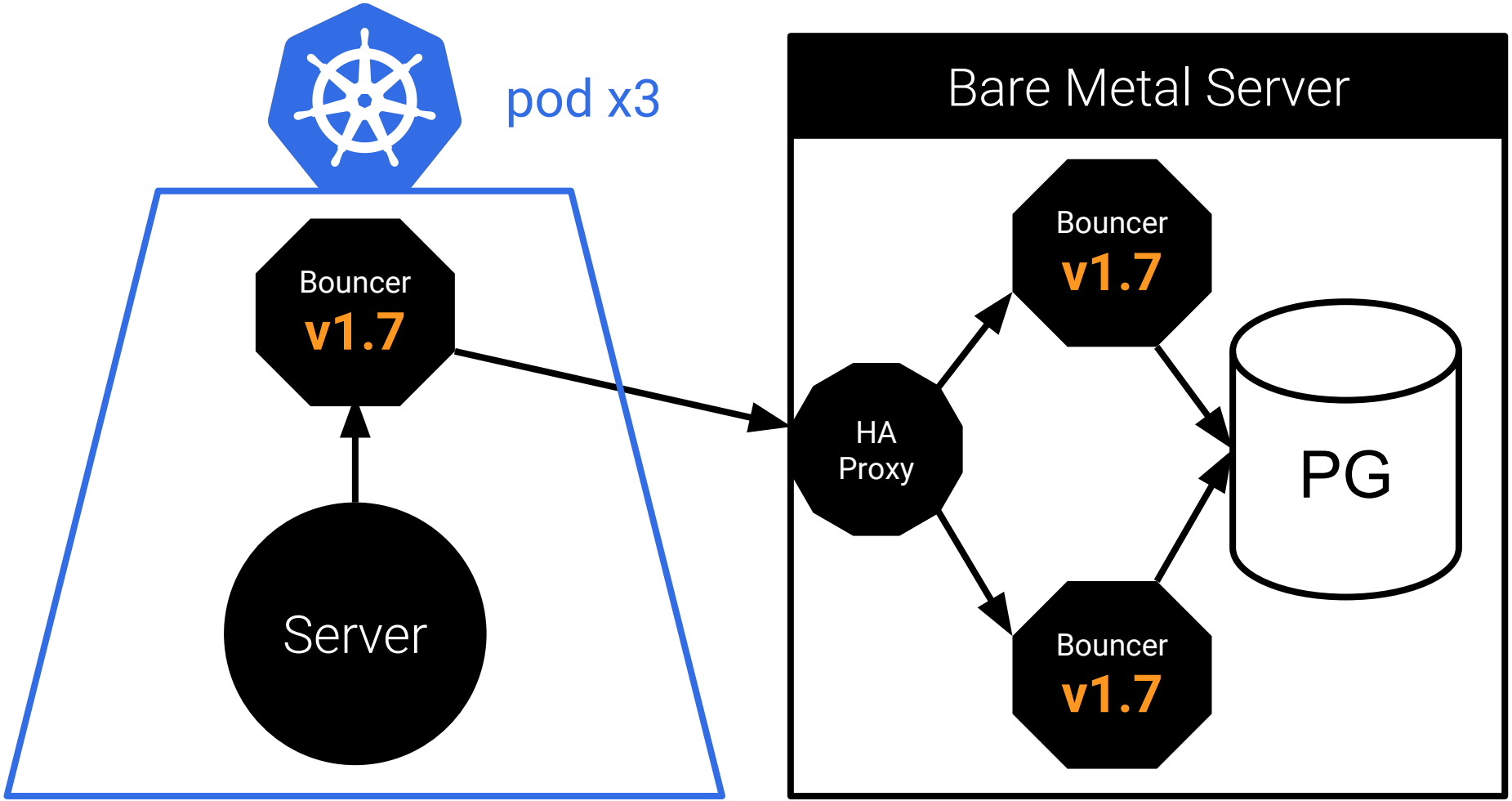
Performance

Security

Now we are ready to fight!

Are you sure?


pod x3
Bouncer
Server
Bare Metal Server
Bouncer
HA
Proxy
PG
Bouncer


pod x3
Bouncer
v1.7
Server
Bare Metal Server
Bouncer
v1.7
HA
Proxy
PG
Bouncer
v1.7

Cancellation causes
unexpected ReadyForQuery
to be sent
in transaction mode

https://github.com/pgbouncer/pgbouncer/issues/223

```
sql := `select * from places where city = ?`

rows, err := s.db.QueryContext(ctx, sql)
```

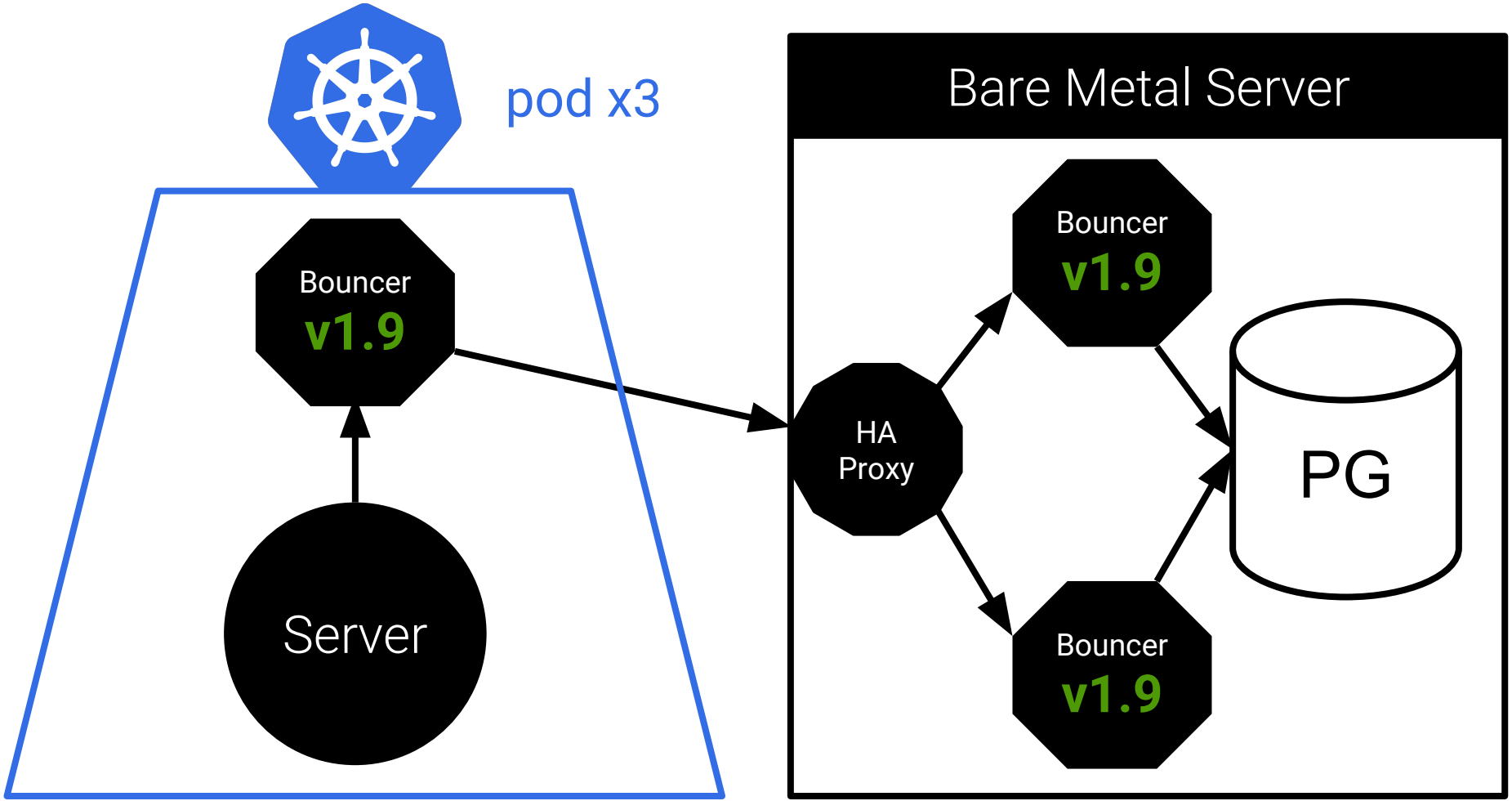
Context **Cancellation** here!

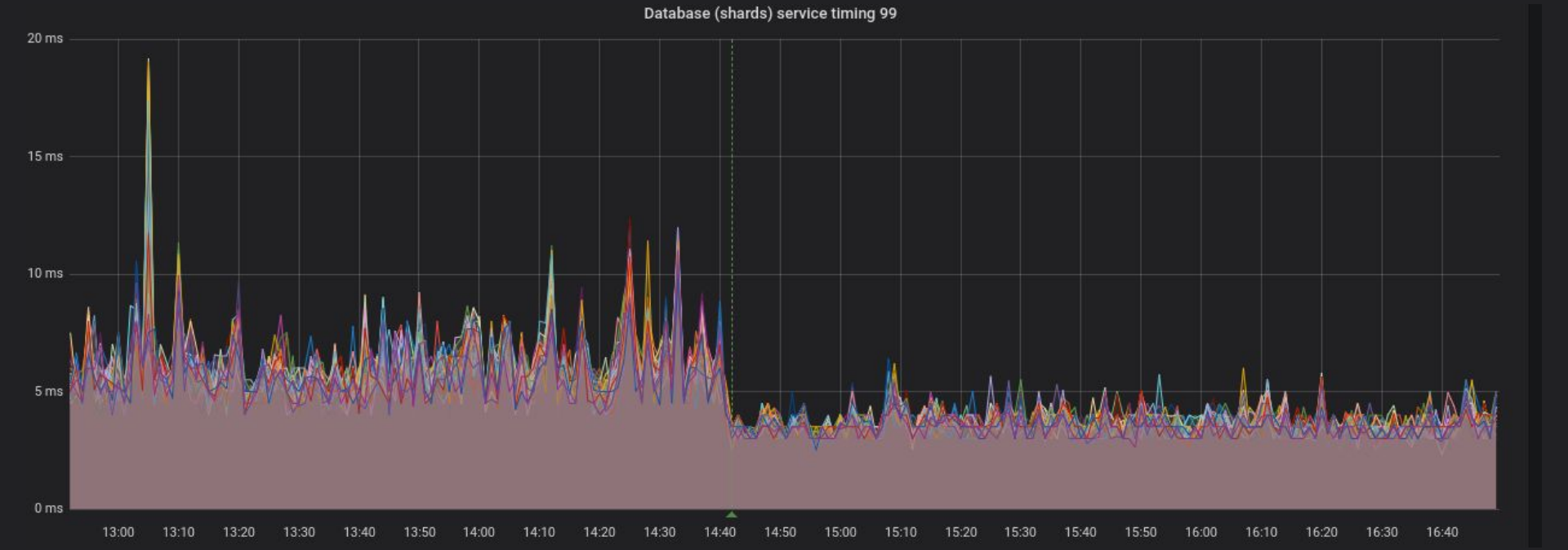
```
sql := `select * from places where city = ?`

ctx := context.Background()

rows, err := s.db.QueryContext(ctx, sql)
```

Cancellation causes
unexpected ReadyForQuery
to be sent
in transaction mode

https://github.com/pgbouncer/pgbouncer/issues/223

Fixed since PgBouncer 1.8


pod x3
Bouncer
v1.9
Server
Bare Metal Server
Bouncer
v1.9
HA
Proxy
PG
Bouncer
v1.9


Database (shards) service timing 99
20 ms
15 ms
10 ms
5 ms
0 ms
13:00
13:10
13:20
13:30
13:40
13:50
14:00
14:10
14:20
14:30
14:40
14:50
15:00
15:10
15:20
15:30
15:40
15:50
16:00
16:10
16:20
16:30
16:40

Go ❤ Postgres

Артемий Рябинков

github.com/furdarius

facebook.com/furdarius

getlag@ya.ru

https://blog.twitch.tv/how-twitch-uses-postgresql-c34aa9e56f58 - Как в Twitch готовят Postgres

https://www.youtube.com/watch?v=Wq7wQ9oyvSw - Odyssey от Yandex

https://wingedpig.com/2017/09/ - Streaming Postgres Changes

https://momjian.us/main/writings/pgsql/aw_pgsql_book/node70.html - OIDs by momjian