Microsoft

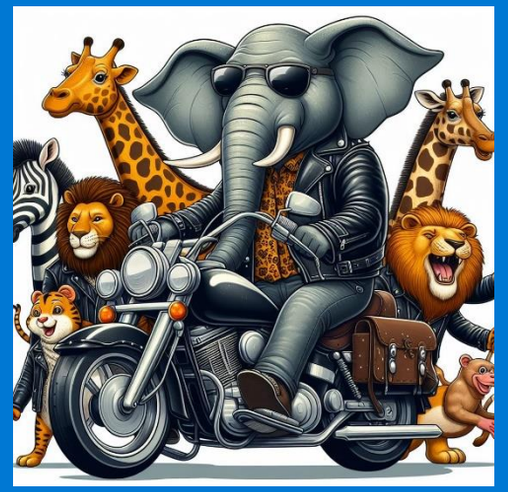
The PostgreSQL Protocol: The Good, the Bad and the Future

Jelte Fennema-Nio

Principal Software Engineer @ Microsoft
Maintainer of Citus & PgBouncer
Contributing to Postgres semi-regularly

@JelteF

2024-05-29

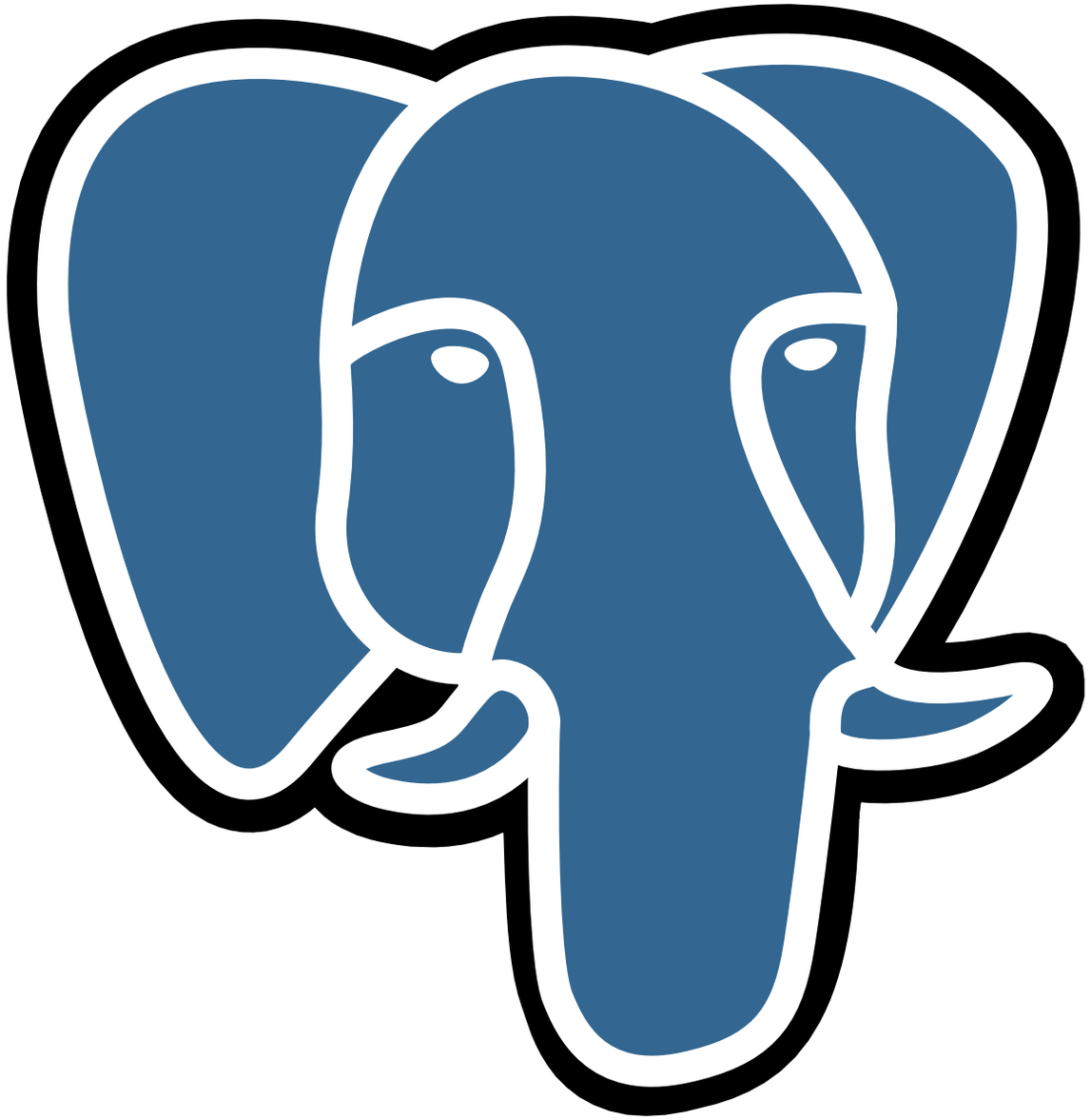


What do we consider the protocol?

How your query gets to Postgres

Not the letter but the envelope and delivery

Only used by Postgres?




UMBRA


CockroachDB


yugabyteDB


HyPer

Nope!


amazon
REDSHIFT


QuestDB

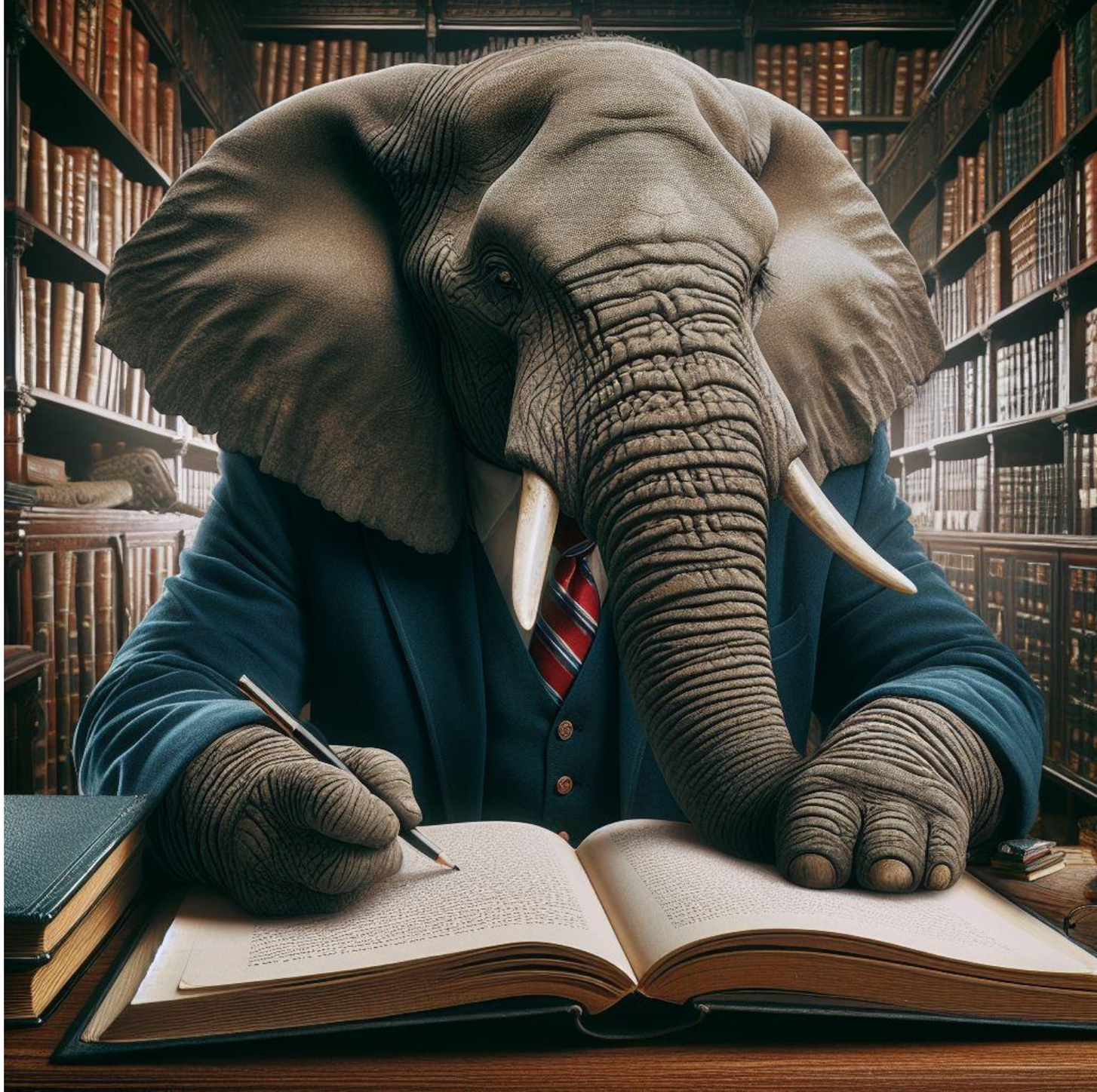

CrateDB

And there are clients and connection poolers

The Good

Many clients

Well documented



What does it consist of?

- Protocol version 3.0
- TCP based protocol
- Messages
    - 1 byte for type
    - 4 bytes for length
    - bytes for the rest of the message based on type field

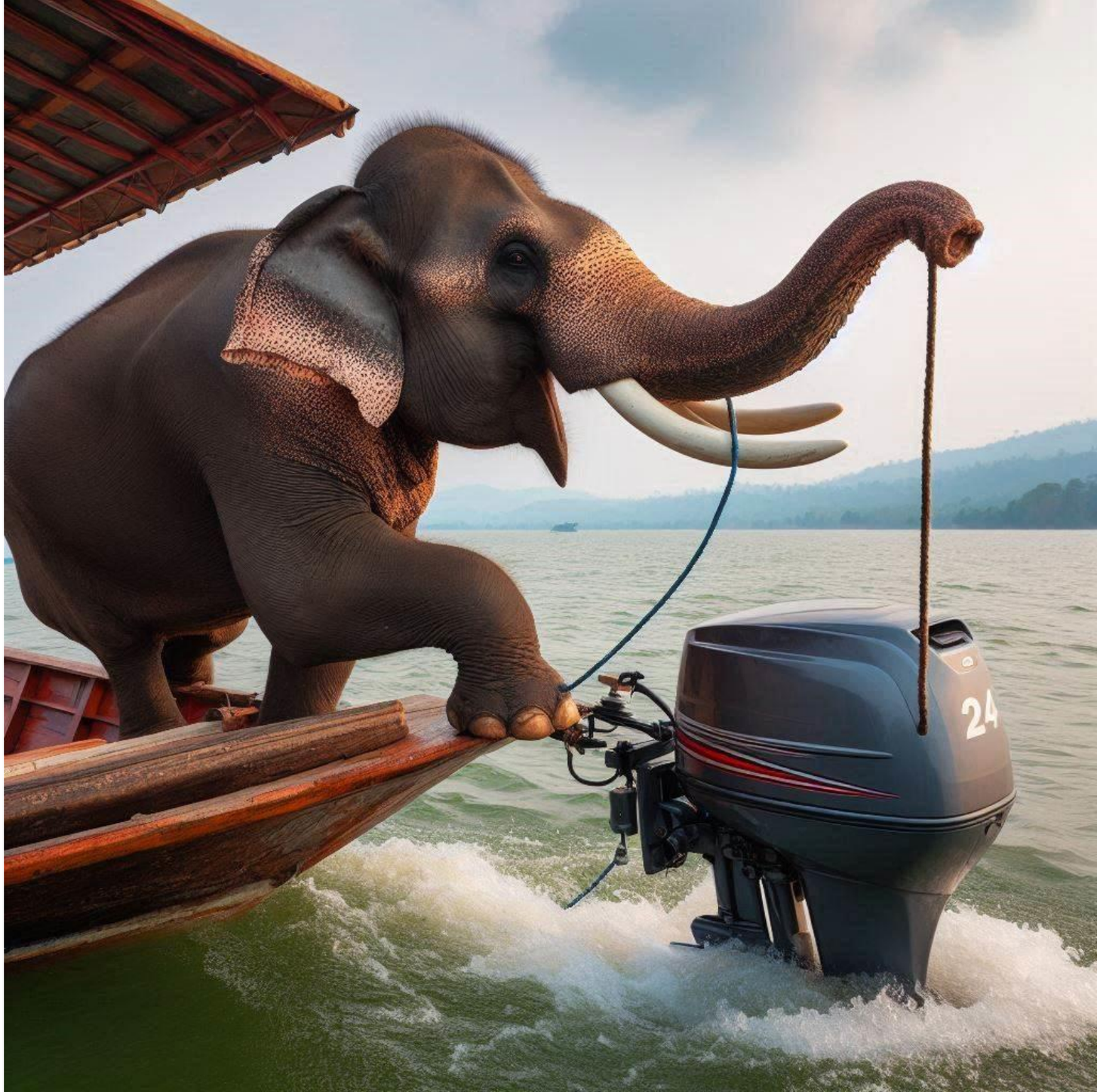
Different phases/sub protocols

- Start-up
- Simple Query Protocol
- Extended Query Protocol
- COPY Protocol
- Logical Replication
- Physical Replication
- Query Cancelling

Different phases/sub protocols

- Start-up
- Simple Query Protocol
- Extended Query Protocol
- COPY Protocol
- Logical Replication
- Physical Replication
- Query Cancelling

Start-up



Start-up

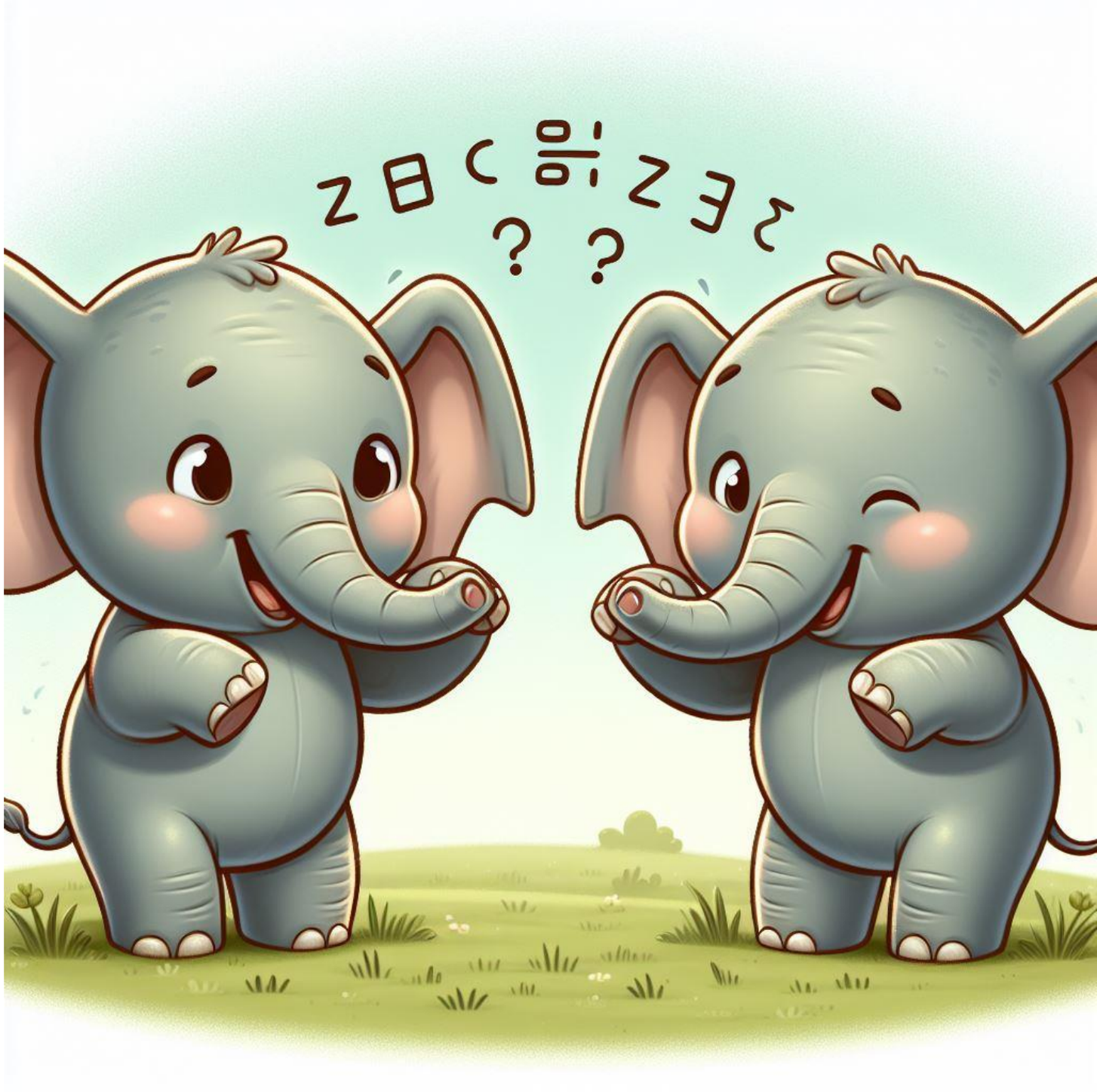
- StartupMessage: One of the only four messages without a type
- Protocol version encoded in 4 bytes
- Key value pairs
- “user” only required key
- “database” and “replication” exist too
- Can pass arbitrary GUCs
- Also “options” key can be used to set GUCs: “-c work_mem=128MB”

  Supports any postgres process startup flag even -e (means Europe)
  Or -d 5 (which enables lots of debug like debug_print_parse)

Encrypted Start-up

- SSLRequest or GSSENCRequest instead of Startup
- Magic version numbers 1234.5679 and 1234.5670
- Postgres answers yes/no
- Set up encryption
- Continue with Startup

Simple Query



Simple Query Protocol

1. -> Query(“SELECT id, name FROM users”)
2. <- RowDescription(
   num_fields=2,
   “id”, INT4OID, text/~~binary~~
   “name”, TEXTOID, text/~~binary~~
)
3. <- DataRow(“123”,”Han Solo”)
4. <- DataRow(“456”,”Luke Skywalker”)
5. <- ReadyForQuery(‘I’) # means “idle”, can also be
   # “T” (“in transaction”) or “E” (“error occurred”)

Extended Query Protocol

1. -> Parse(“SELECT * FROM users WHERE id = $1)
2. -> Bind(params=[123], column_formats=[binary, text])
3. -> Describe()
4. -> Execute()
5. <- RowDescription(
   num_fields=1,
   “id”, INT4OID, ~~text~~/binary
   “name”, TEXTOID, text/~~binary~~
)
6. <- DataRow(123,“Han Solo”)
7. <- ReadyForQuery(‘I’)

Extended Query Protocol

- What almost every client driver uses
- Free SQL escaping for clients
- Used for protocol level prepared statements:

```
Parse(“hard-to-plan-query”, “SELECT … JOIN … JOIN … id = $1”)
Bind(“hard-to-plan-query”, 123)
Execute()
Bind(“hard-to-plan-query”, 456)
Execute()
```

Query pipelining

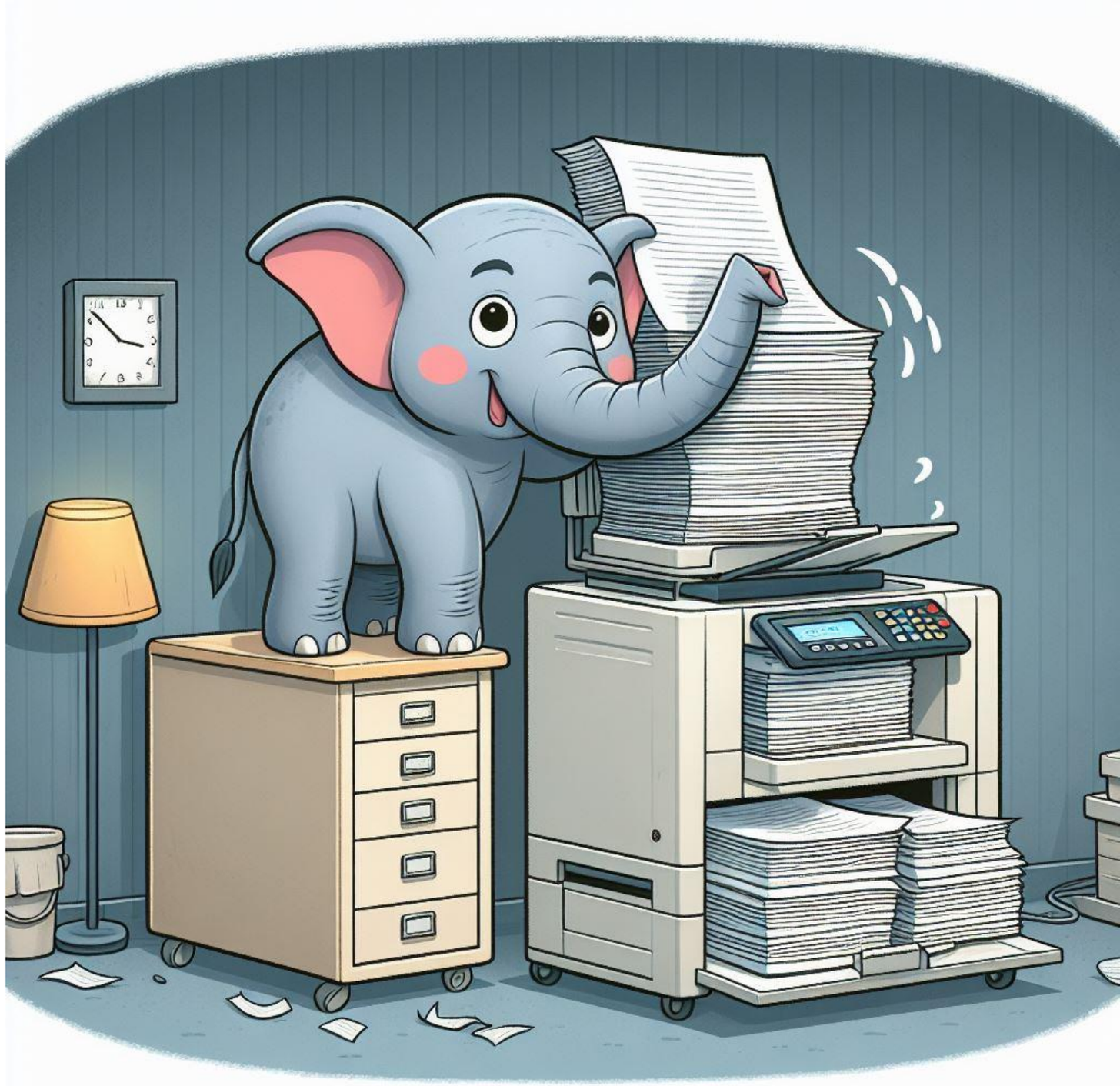
1. Parse(“INSERT VALUES users(name) VALUES ($1))
2. Bind(“Padme”)
3. Execute()
4. Bind(“Yoda”)
5. Execute()
6. Bind(“Boba Fett”)
7. Execute()
8. Sync

Query pipelining failures

1. Parse(“INSERT VALUES users(name) VALUES ($1))
2. Bind(“Yoda”)
3. Execute()
4. Bind(“Yoda”) # Unique conflict
5. Execute()
6. Bind(“Boba Fett”)
7. Execute()
8. Sync

Rolls everything back, and ignores commands until Sync

COPY protocol



COPY protocol (TO STDOUT)

1. -> Prepare("**COPY TO STDOUT** (select generate_series(10000))")
2. -> Bind()
3. -> Execute()
4. **<- CopyOutResponse()**
5. **<- CopyData("<bytes>")**
6. **<- CopyData("<bytes>")**
7. **<- CopyDone()**
8. -> Sync

COPY protocol (FROM STDIN)

1. -> Prepare("**COPY FROM STDIN**")
2. -> Bind()
3. -> Execute()
4. **<- CopyInResponse()**
5. **-> CopyData("<bytes>")**
6. **-> CopyData("<bytes>")**
7. **-> CopyDone()**
8. -> Sync

COPY protocol (FROM STDIN)

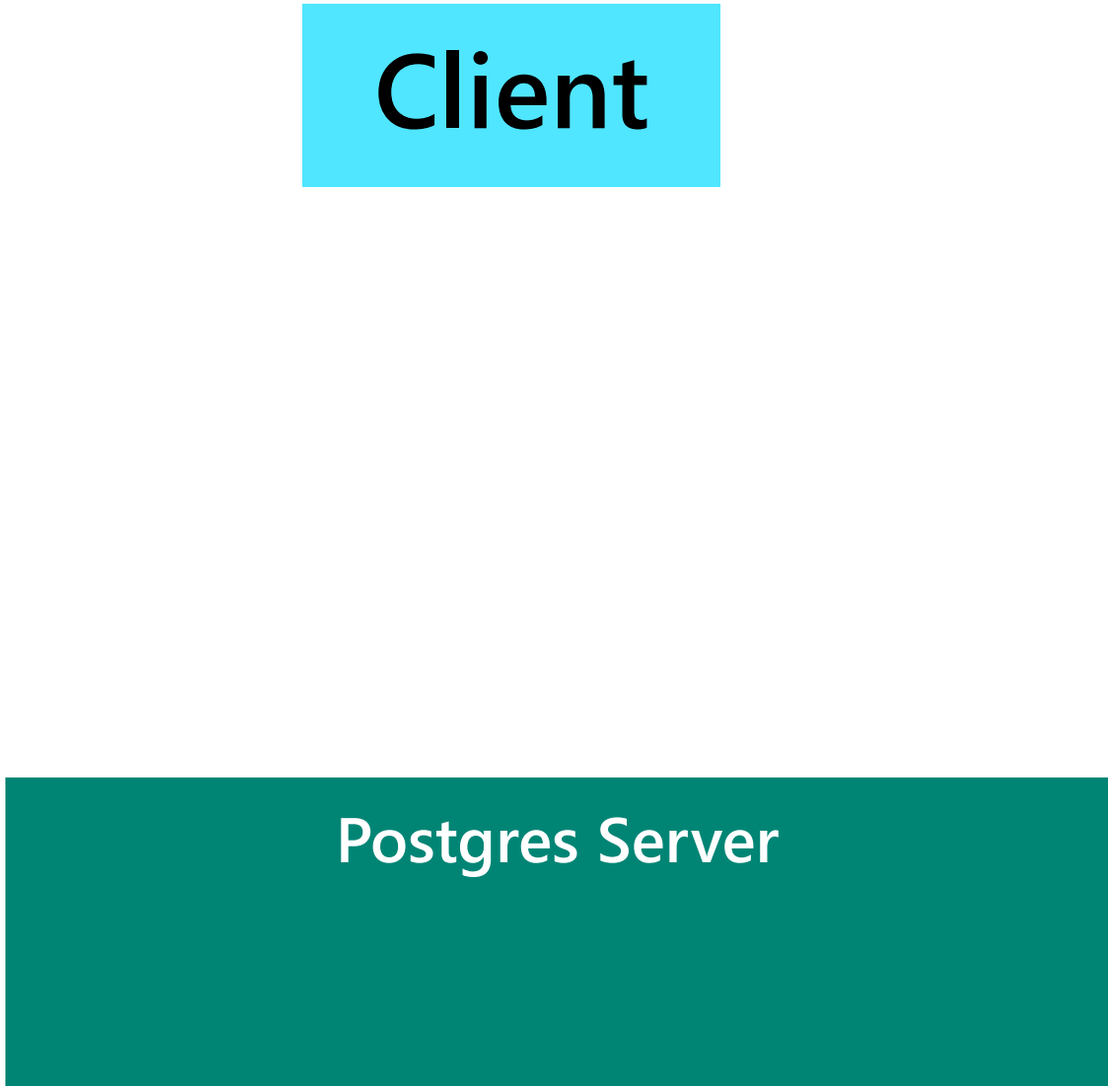
- Weird thing: Postgres ignores Sync messages until CopyDone

Cancel protocol

Cancel requests

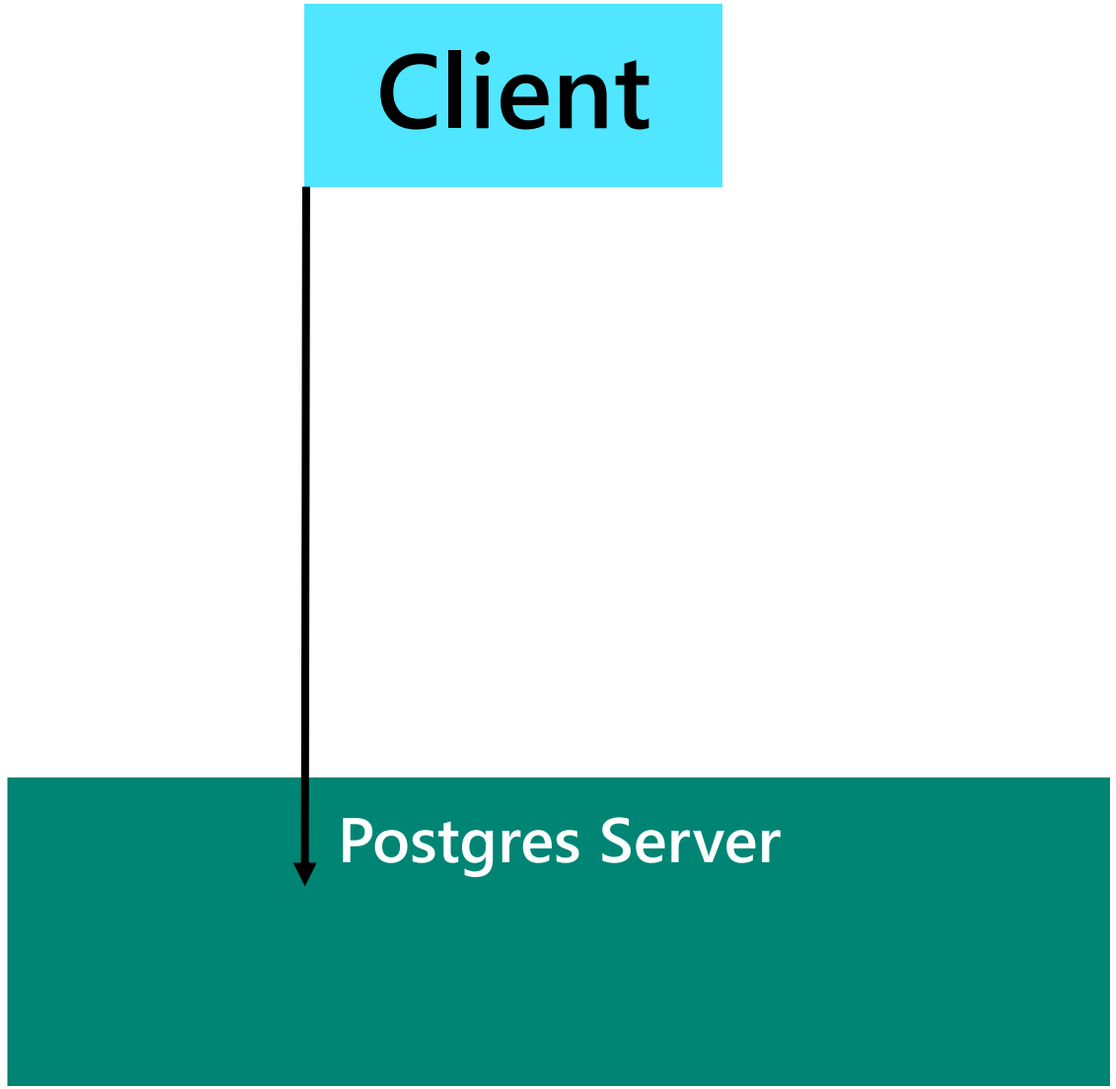
- BackendKeyData(pid, secret) is sent as response to StartupMessage
- CancelRequest(pid, secret) cancels any query
- Magic version number 1234.5678
- New connection needed
- Similar to StartupMessage, but connection is closed immediatly

How Postgres cancellation requests work



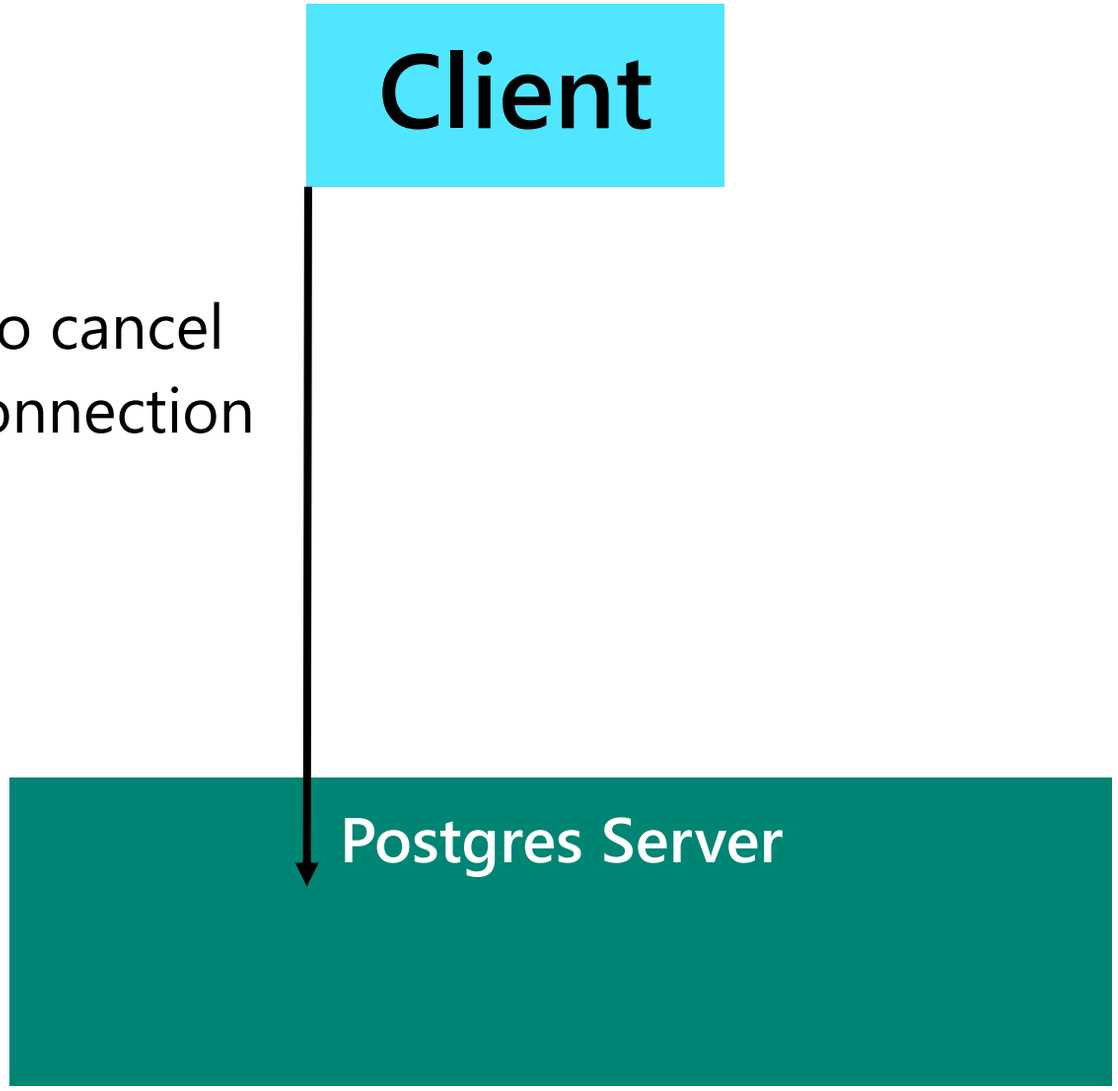
How Postgres cancellation requests work

-> CONNECT



How Postgres cancellation requests work

-> CONNECT
<- You can send me
SECRET-TOKEN-123 to cancel
any queries on this connection

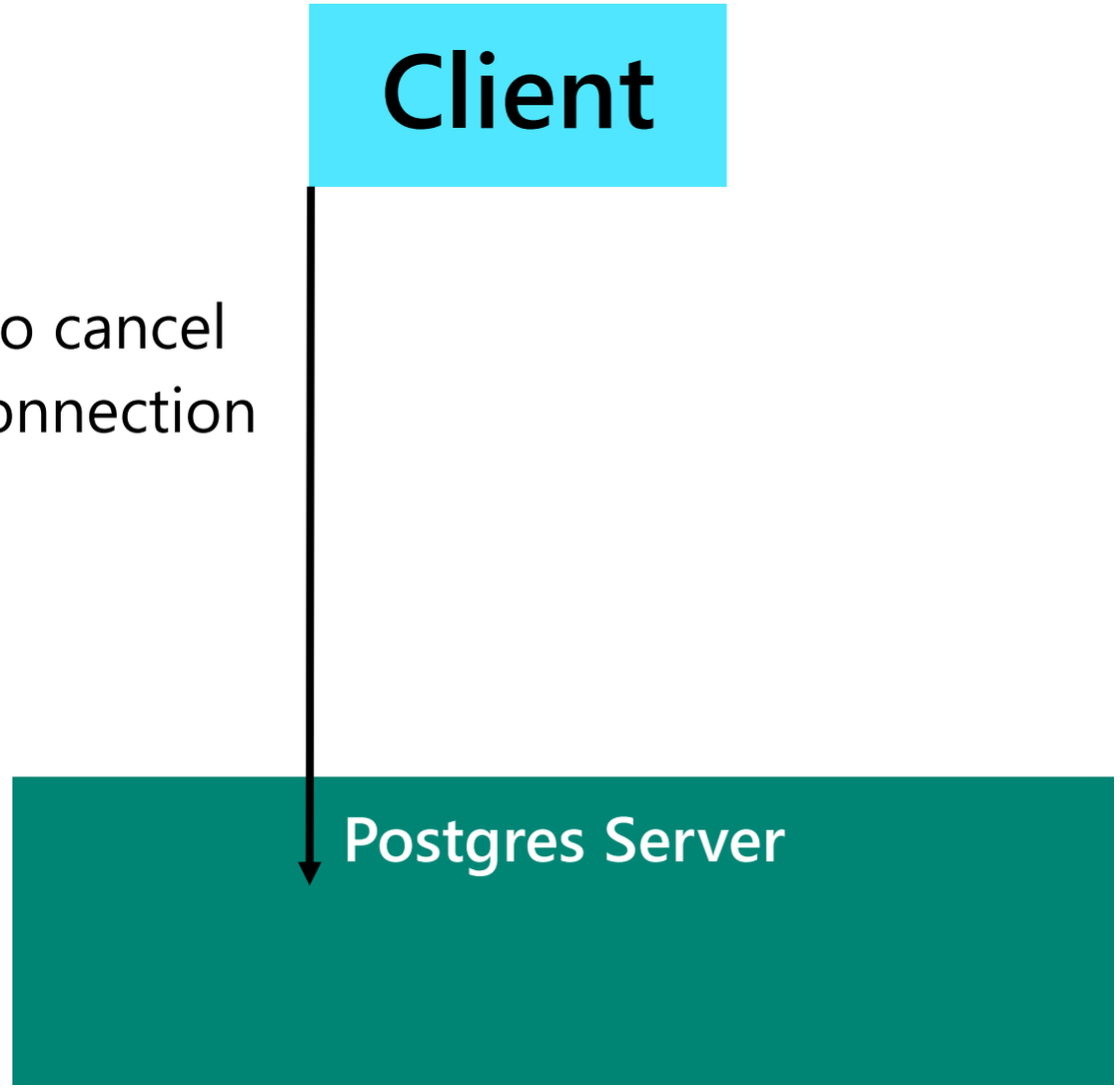


How Postgres cancellation requests work

-> CONNECT

<- You can send me
SECRET-TOKEN-123 to cancel
any queries on this connection

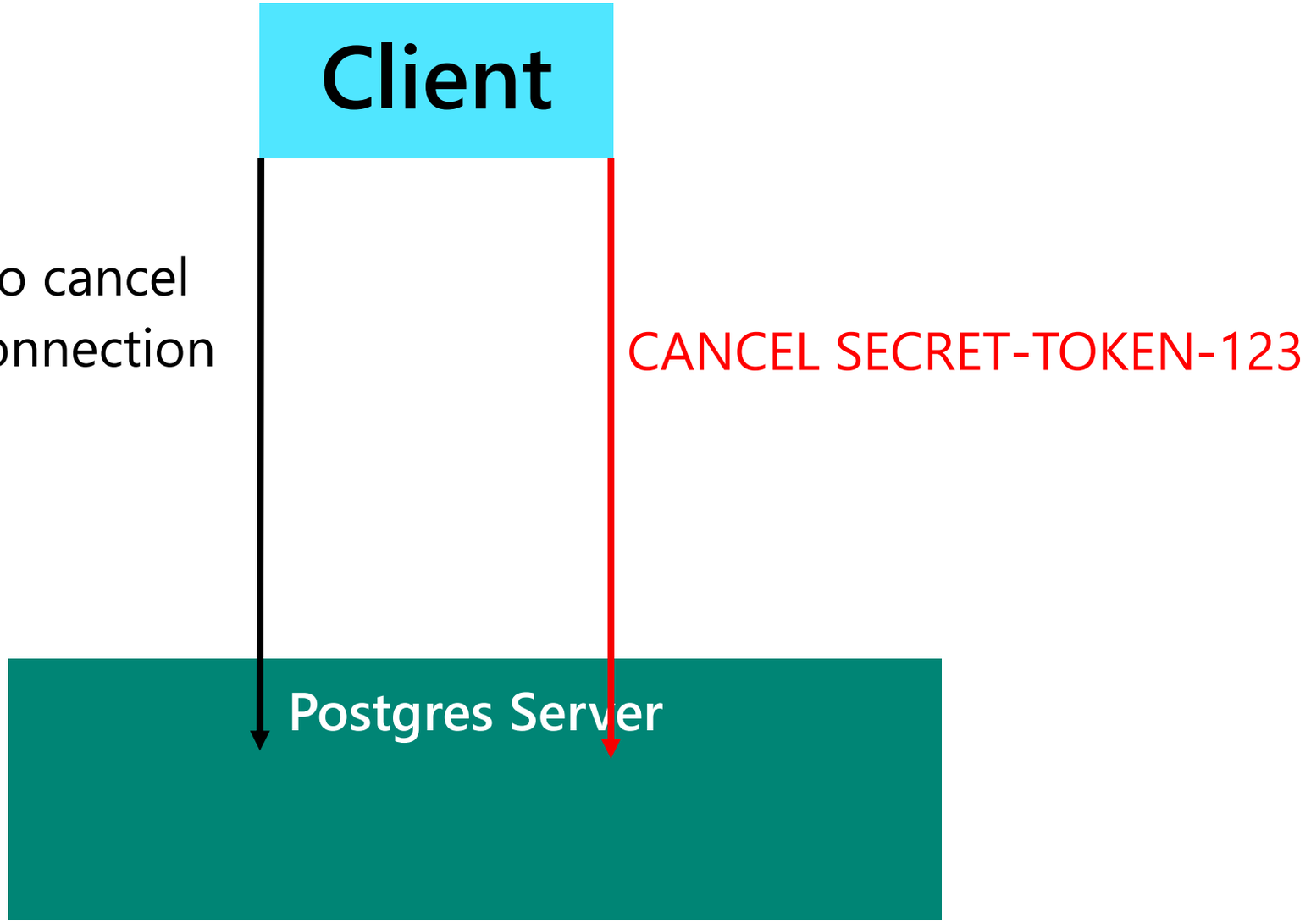
-> RUN:
DELETE FROM users;



How Postgres cancellation requests work

-> CONNECT
<- You can send me
SECRET-TOKEN-123 to cancel
any queries on this connection

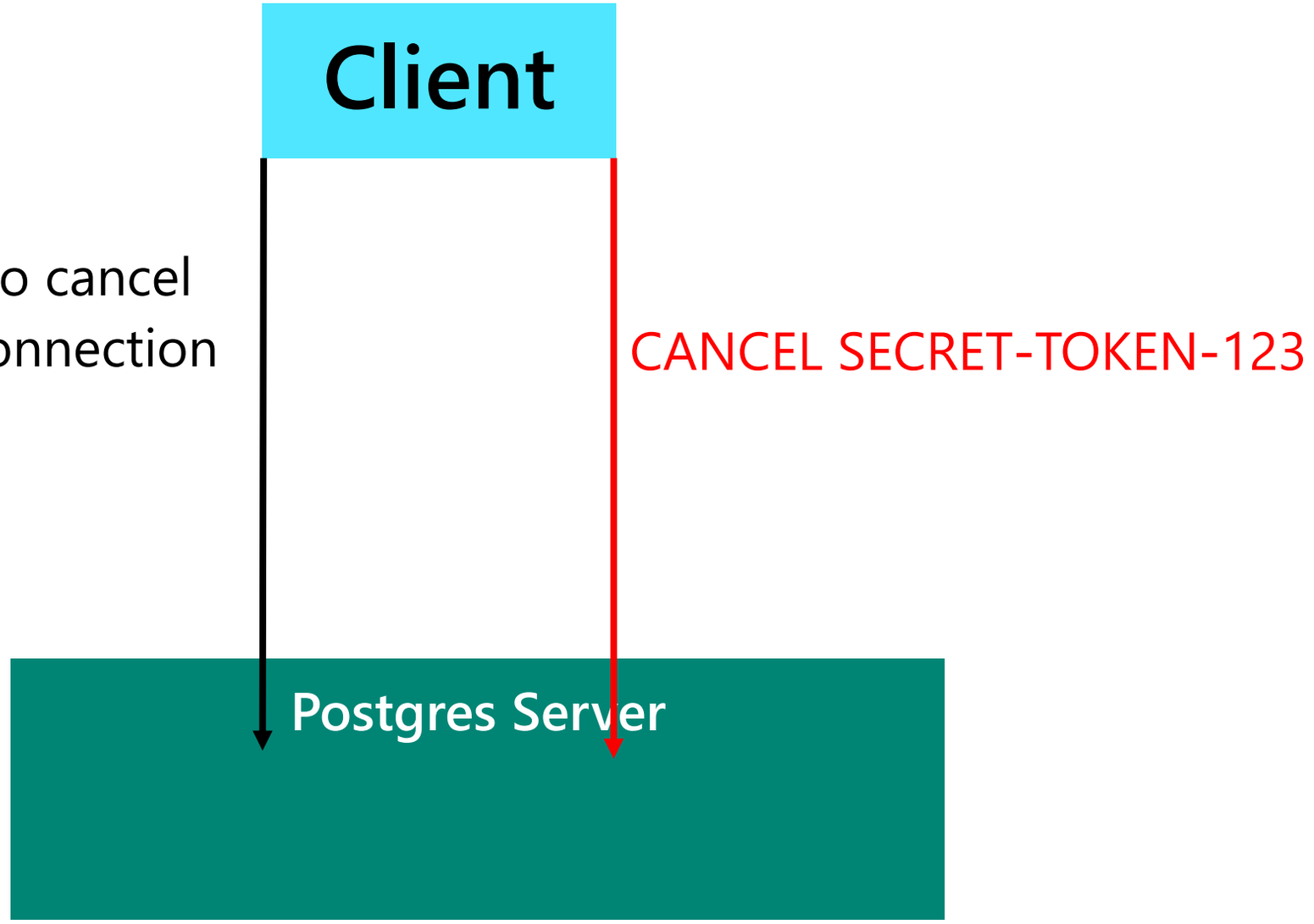
-> RUN:
DELETE FROM users;



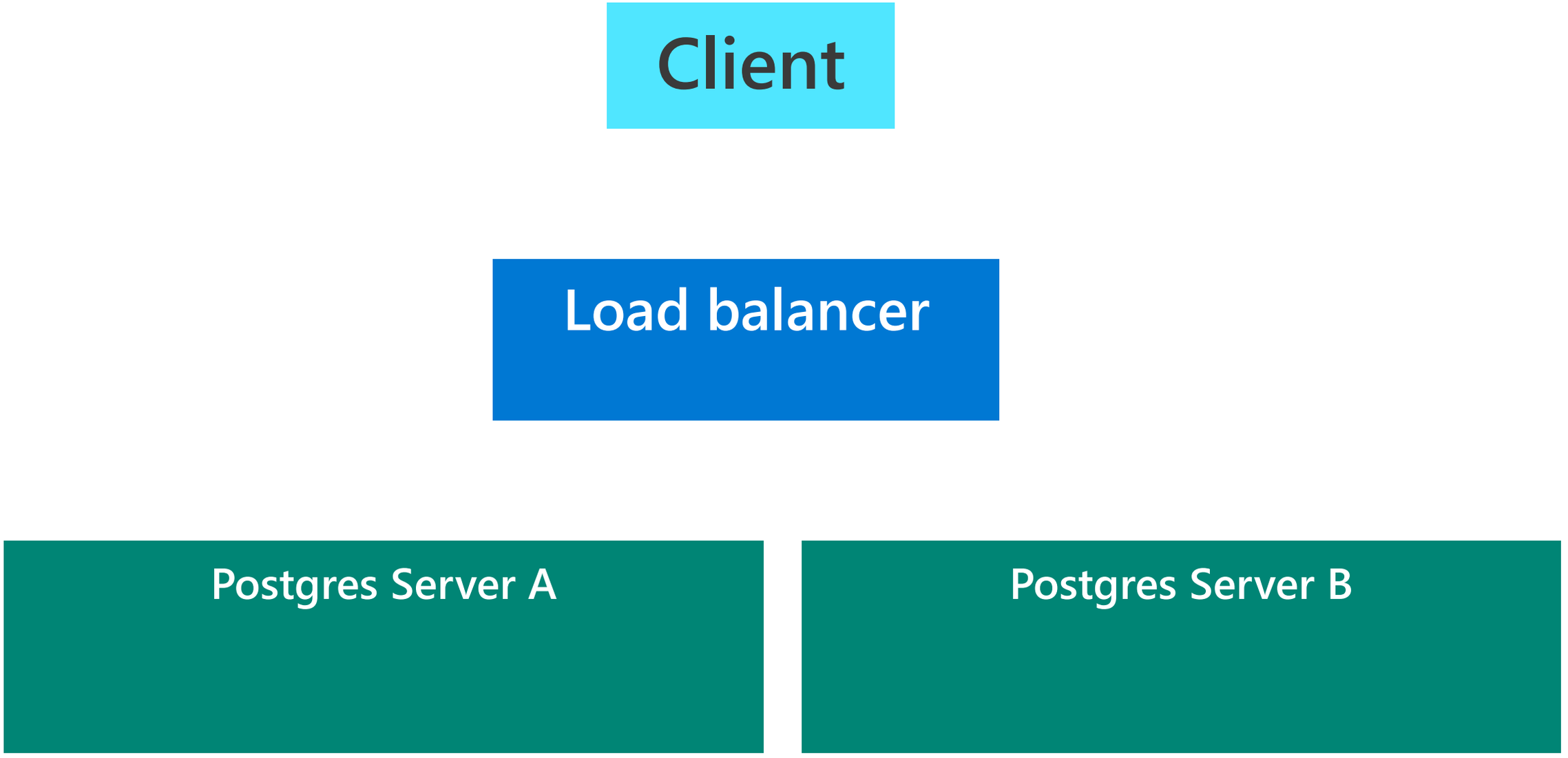
How Postgres cancellation requests work

-> CONNECT
<- You can send me
SECRET-TOKEN-123 to cancel
any queries on this connection

-> RUN:
DELETE FROM users;
<- CANCELLED QUERY

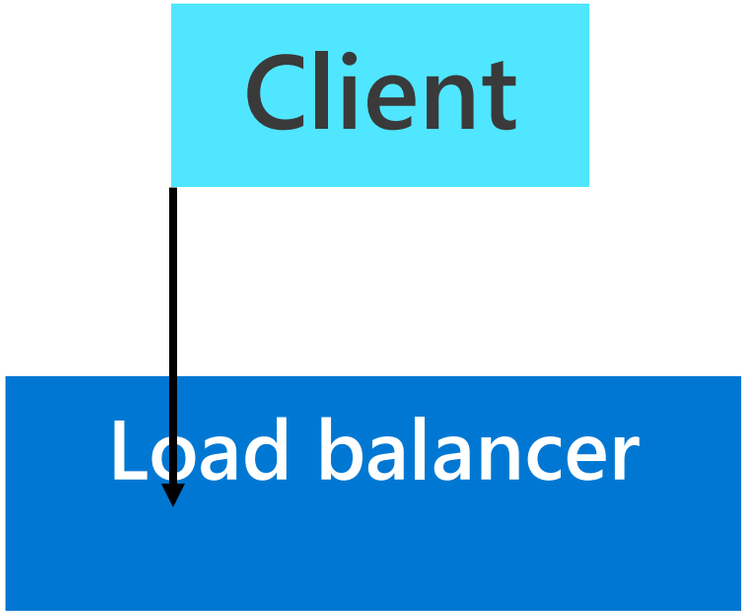


So what happens with cancellations and a load balancer



So what happens with cancellations and a load balancer

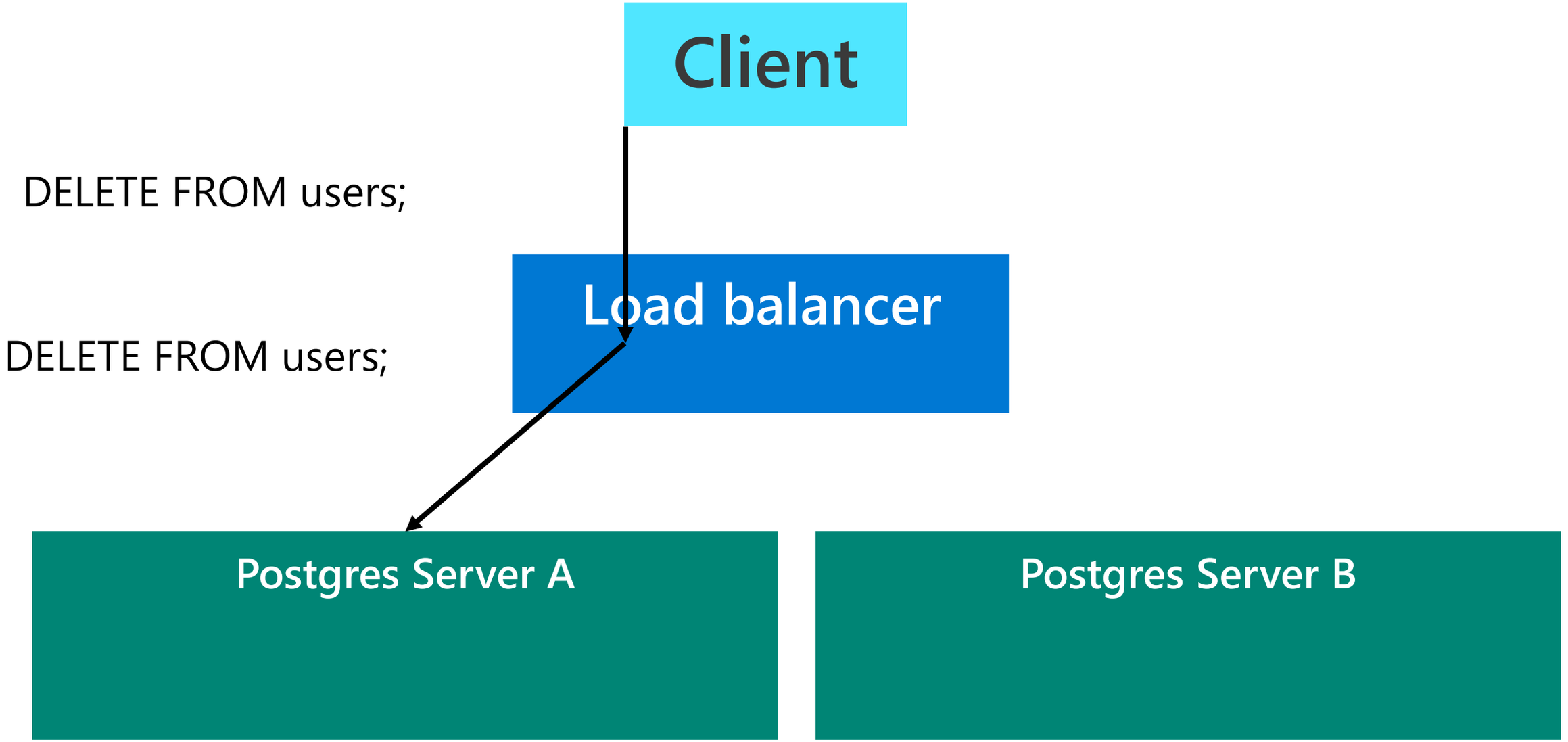
DELETE FROM users;



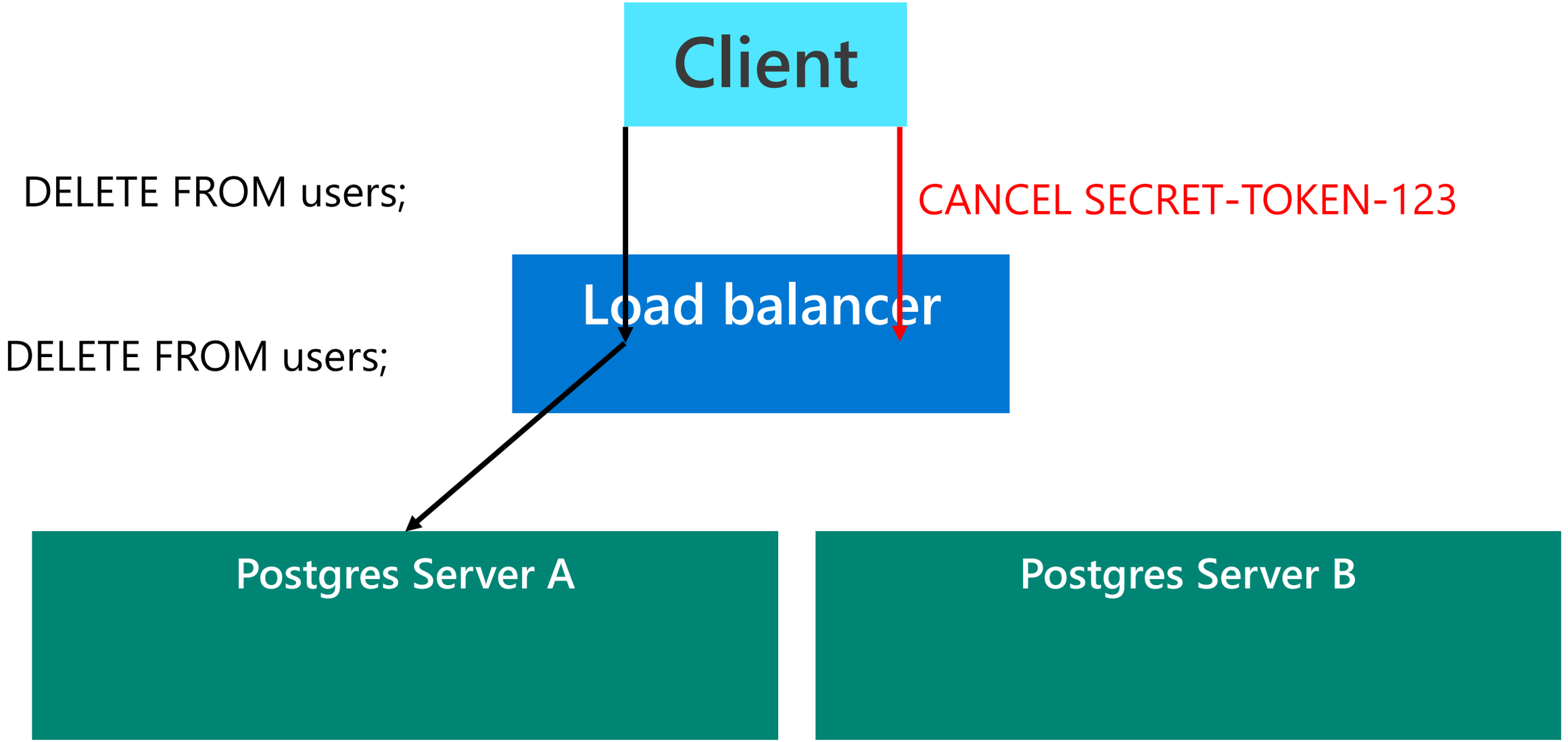
Postgres Server A

Postgres Server B

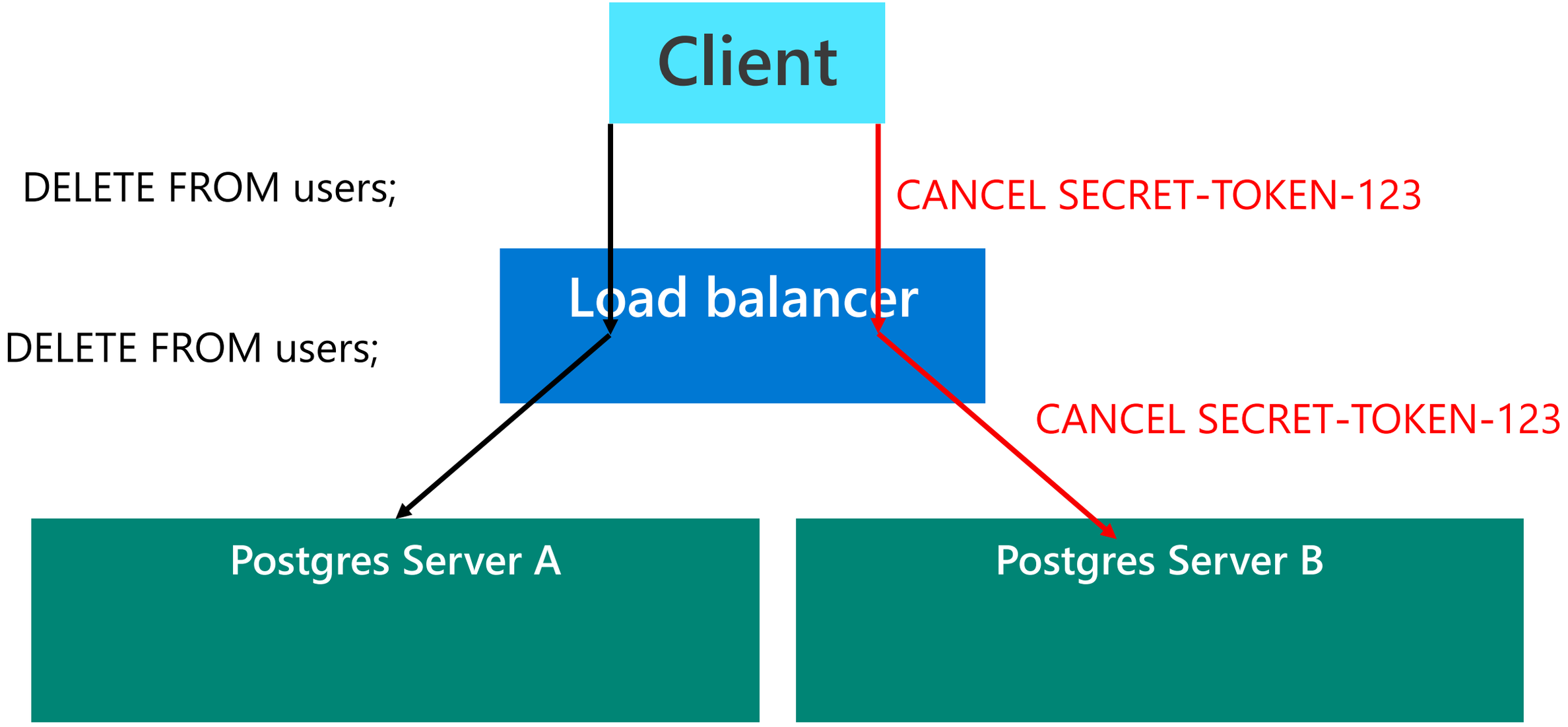
So what happens with cancellations and a load balancer



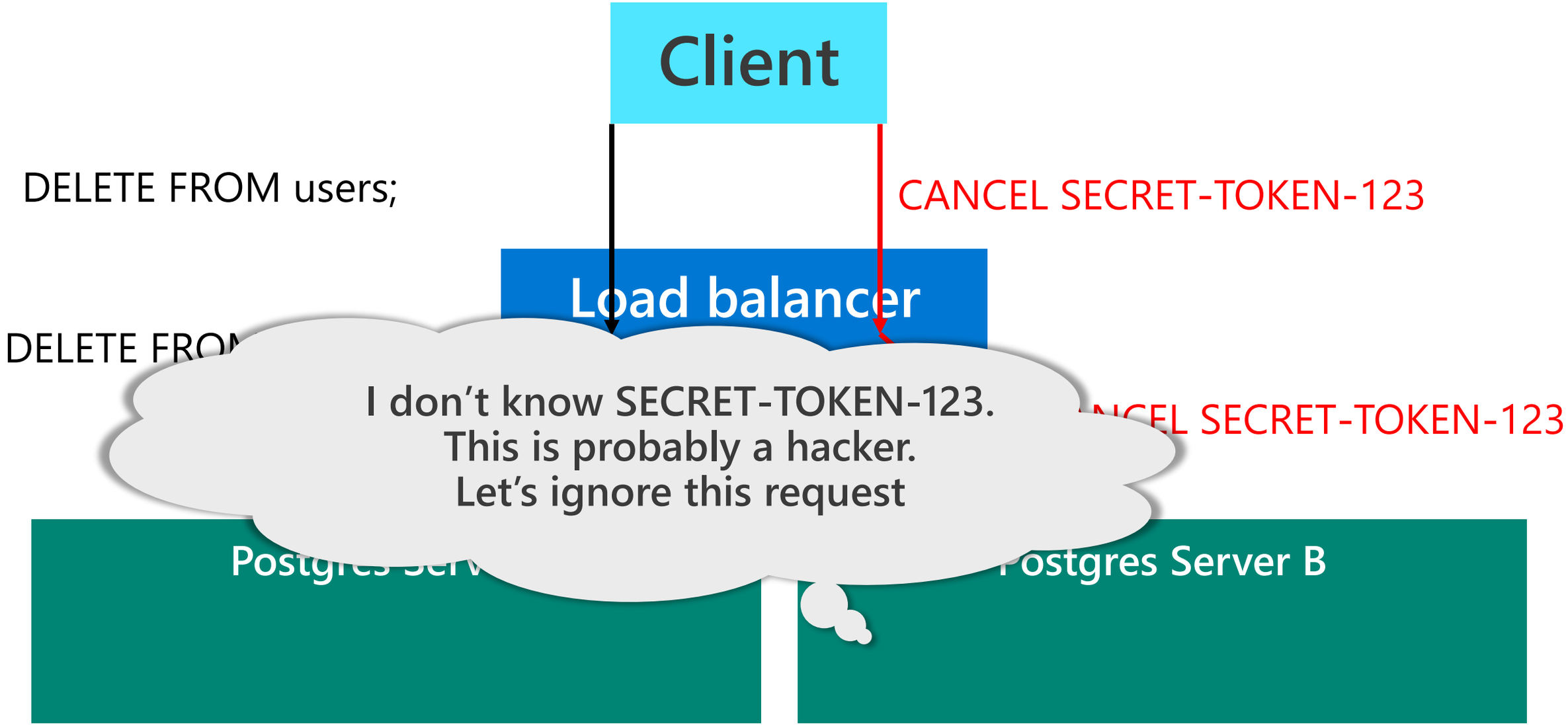
So what happens with cancellations and a load balancer



So what happens with cancellations and a load balancer

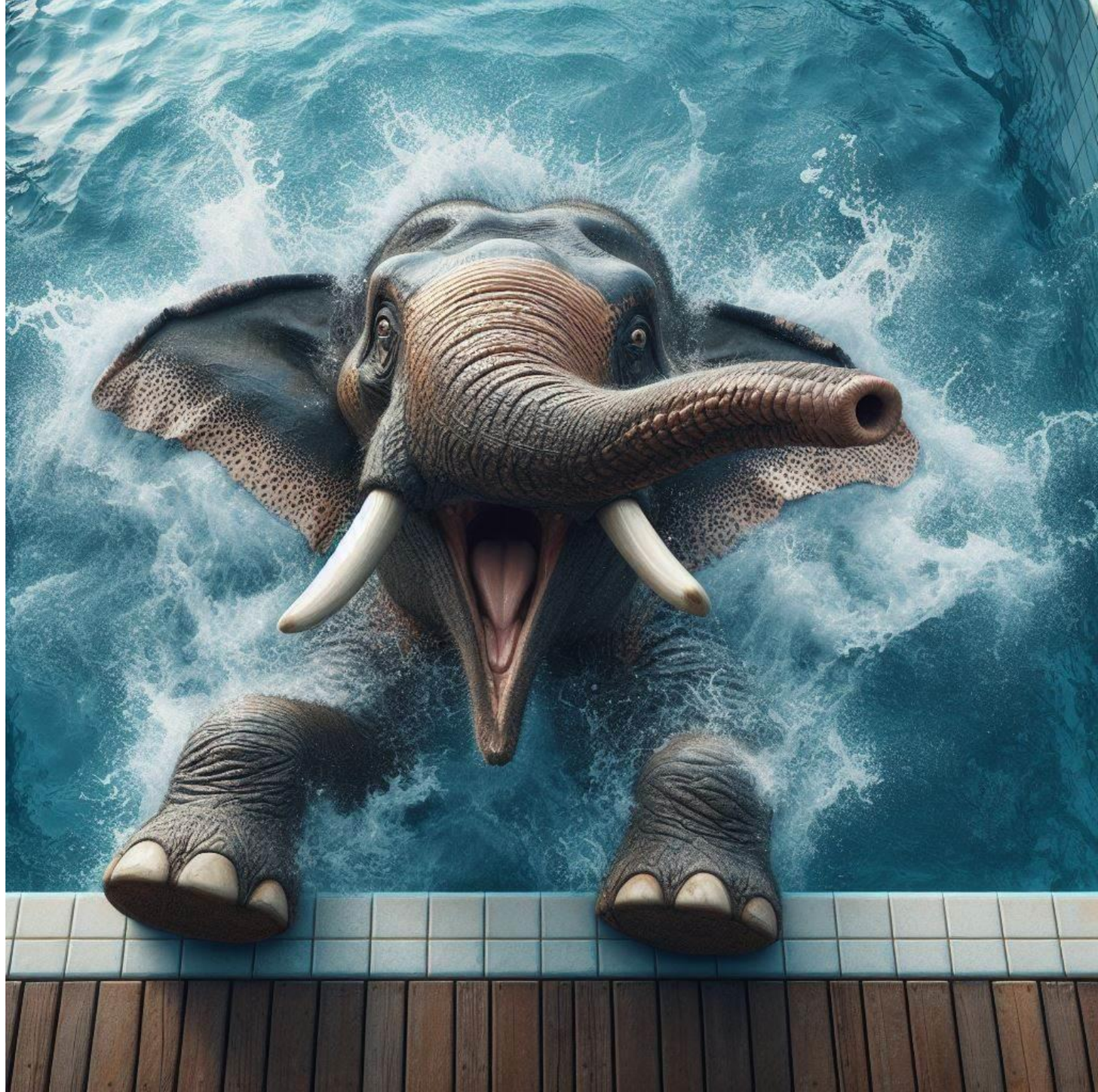


So what happens with cancellations and a load balancer

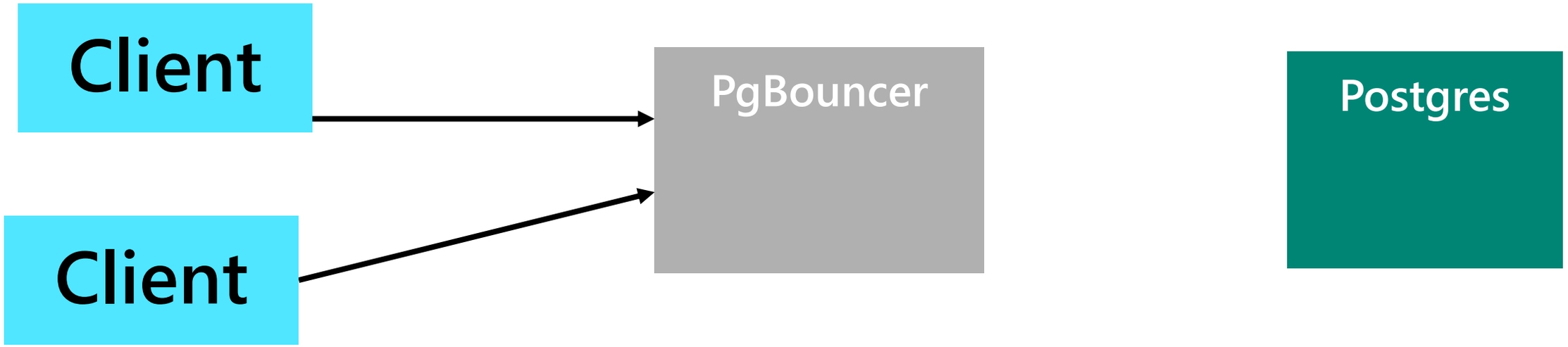


The Bad

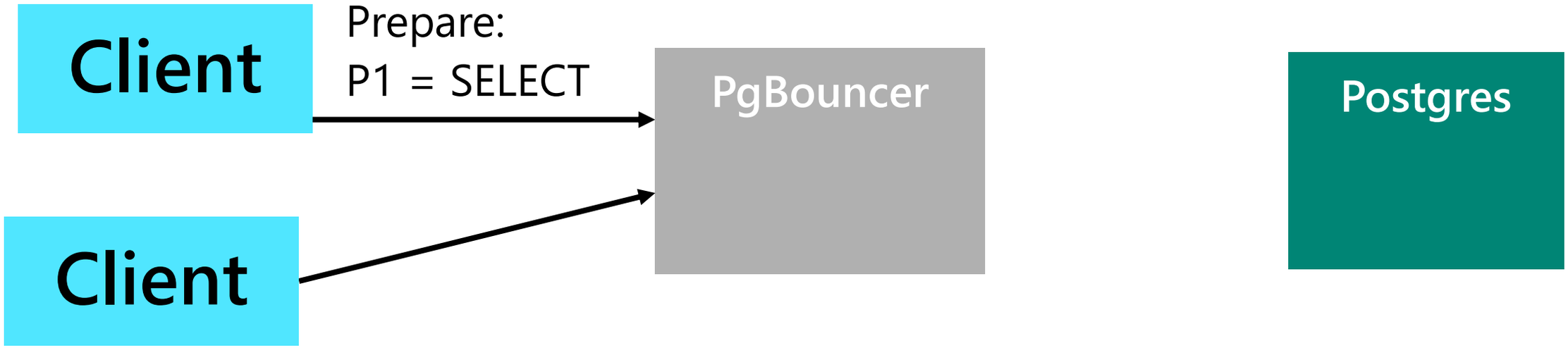
Hard for transaction pooling



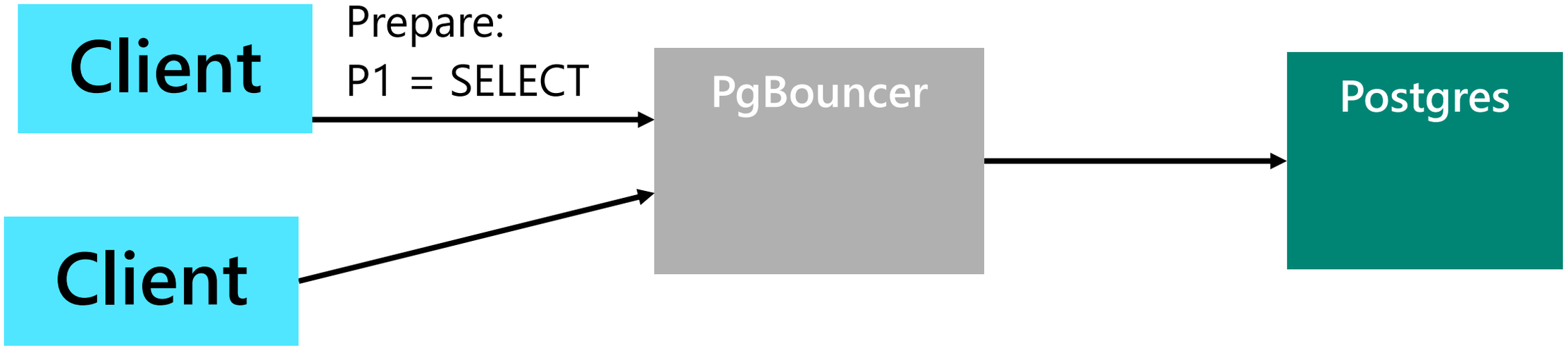
An example prepared statements



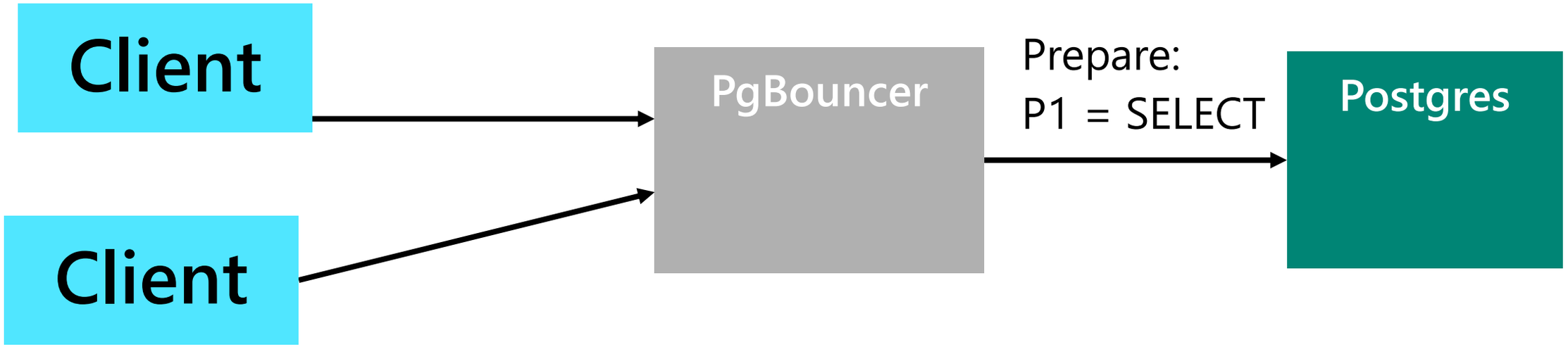
An example prepared statements



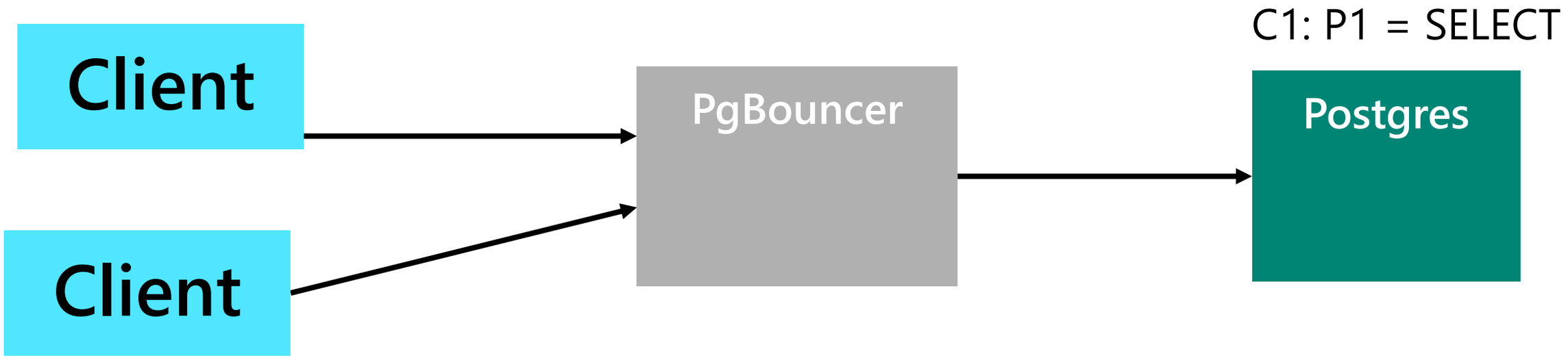
An example prepared statements



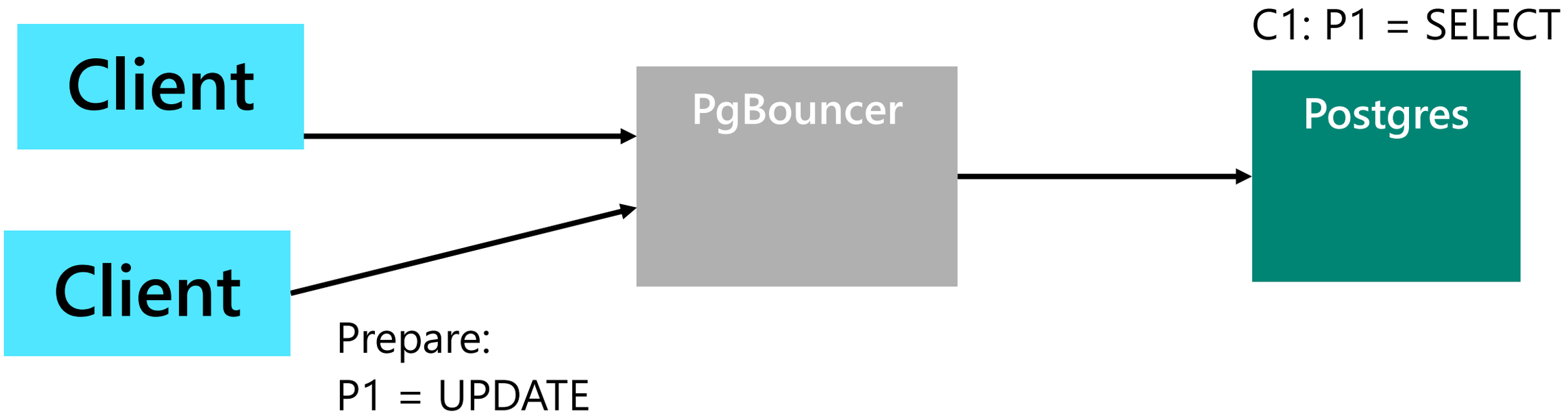
An example prepared statements



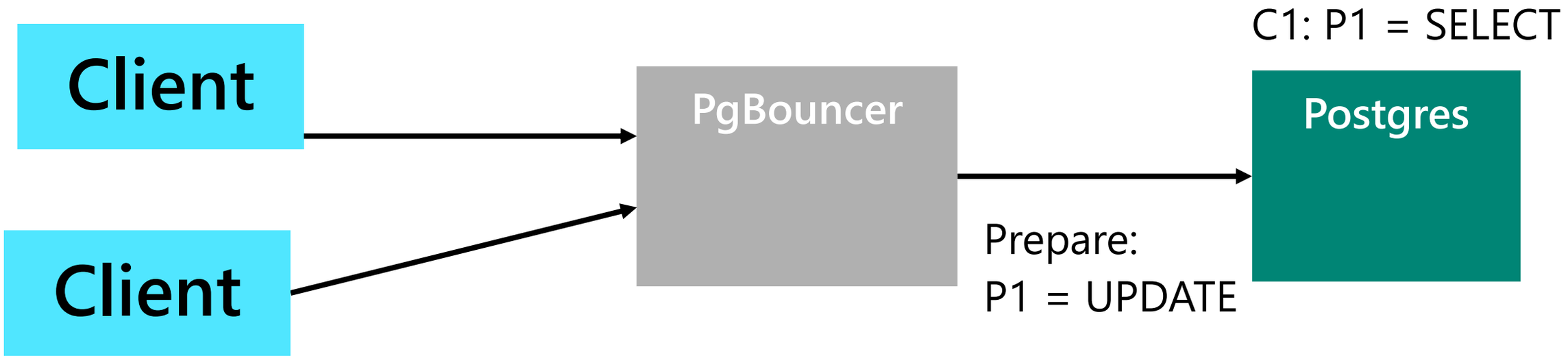
An example prepared statements



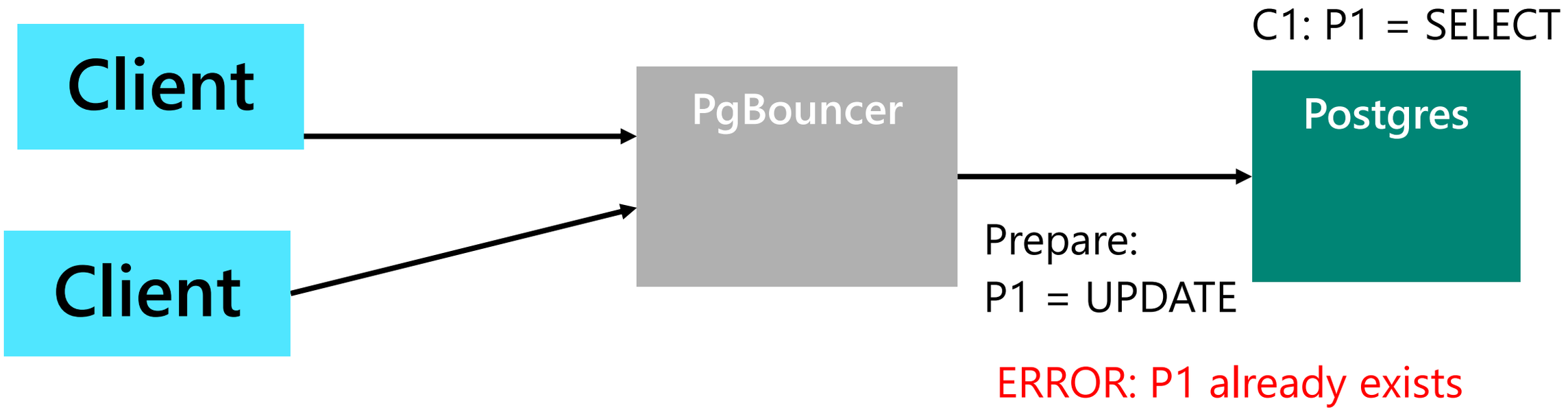
An example prepared statements



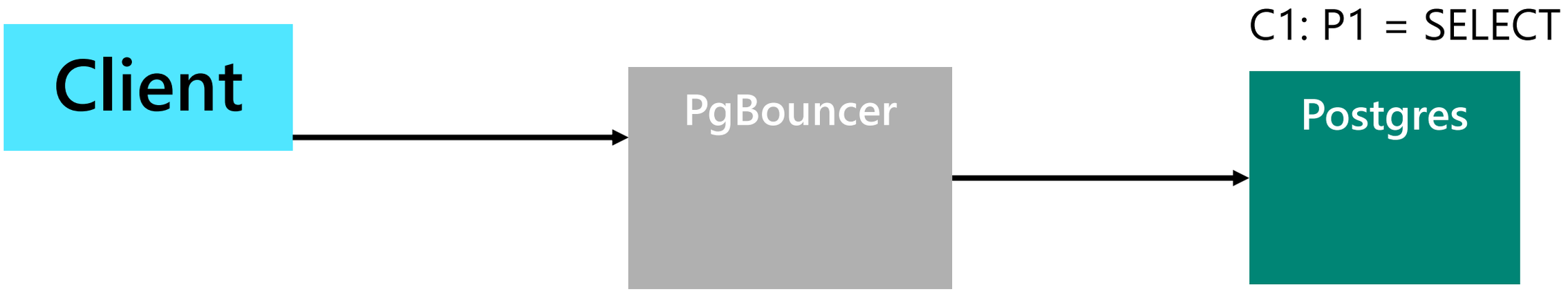
An example prepared statements



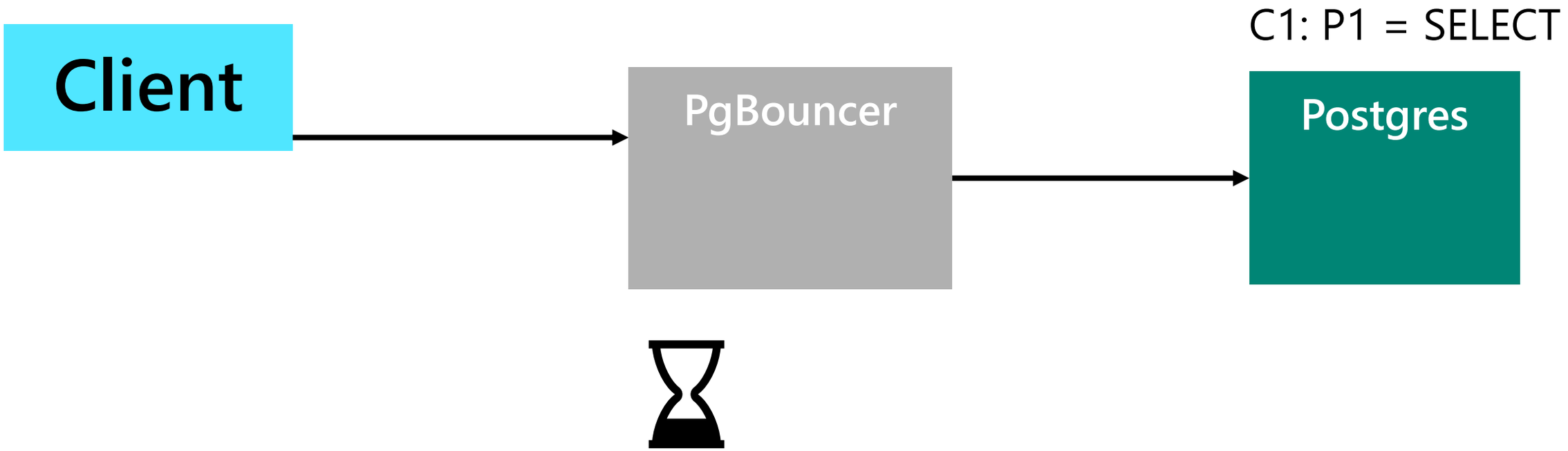
An example prepared statements



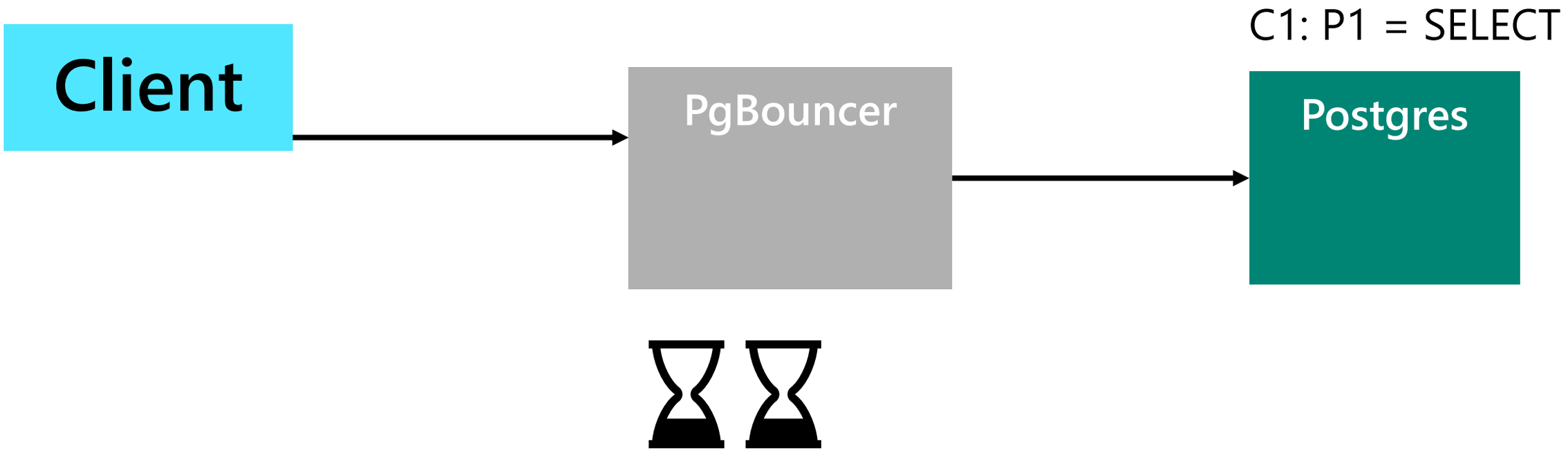
An example prepared statements



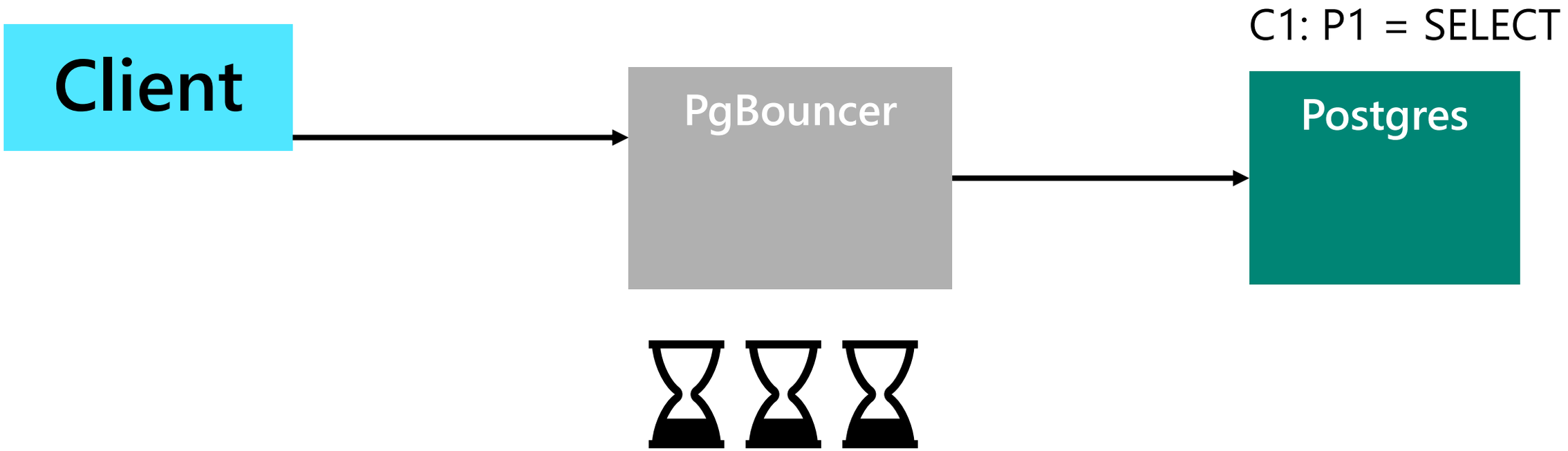
An example prepared statements



An example prepared statements

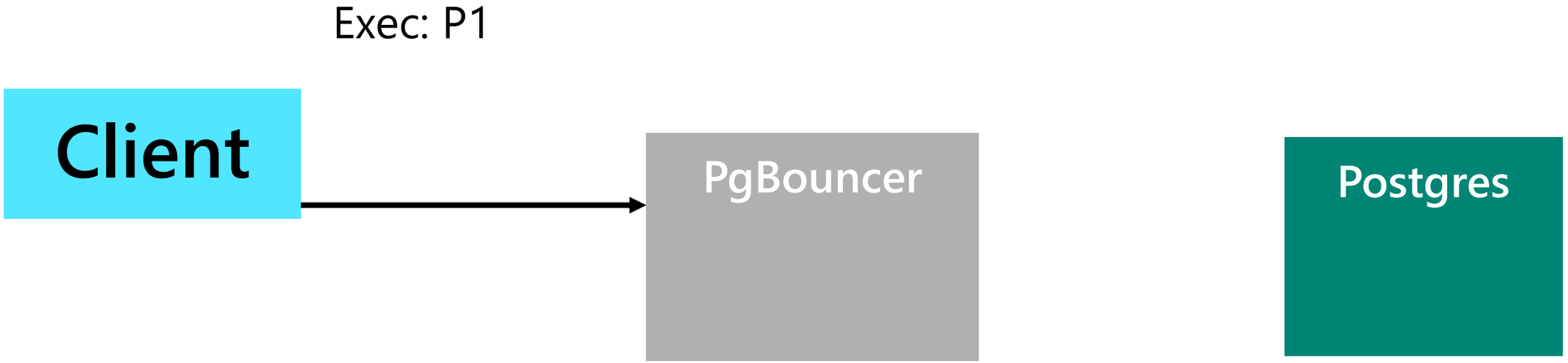


An example prepared statements

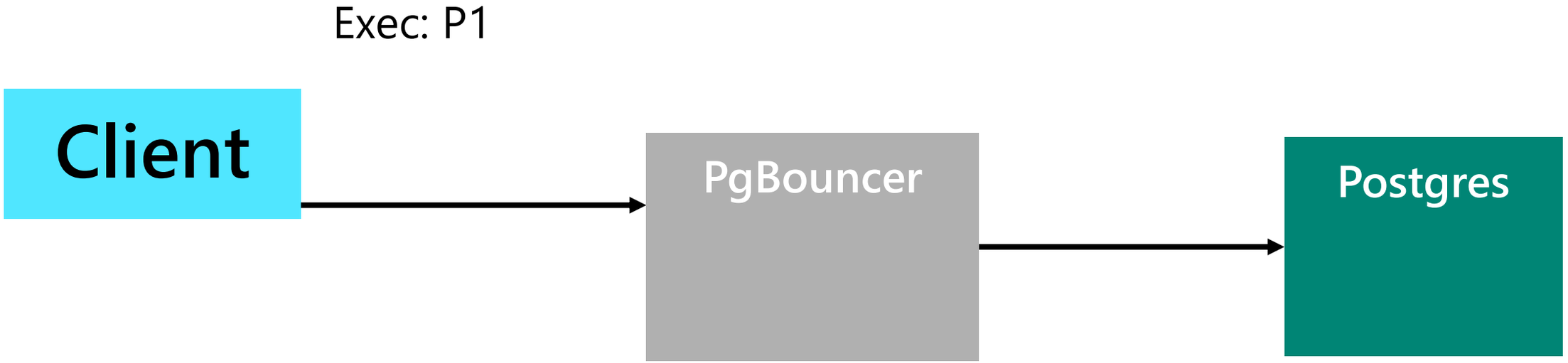


An example prepared statements

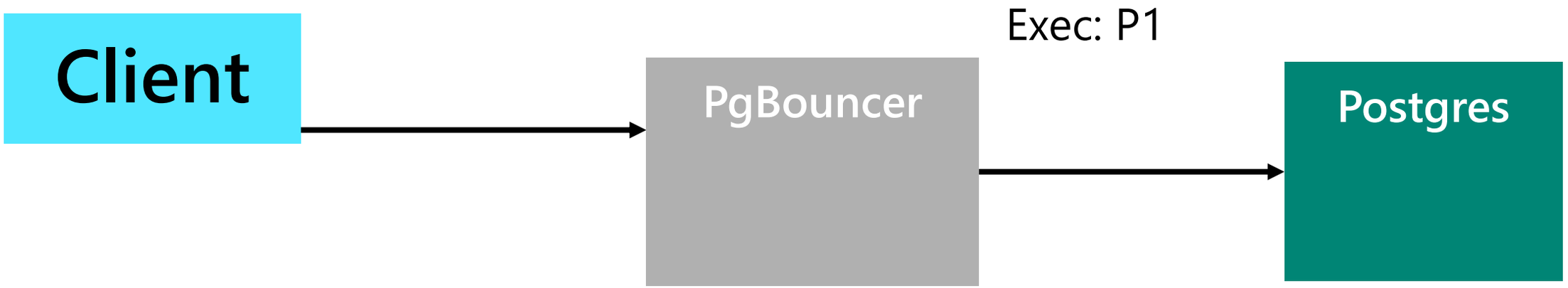
An example prepared statements



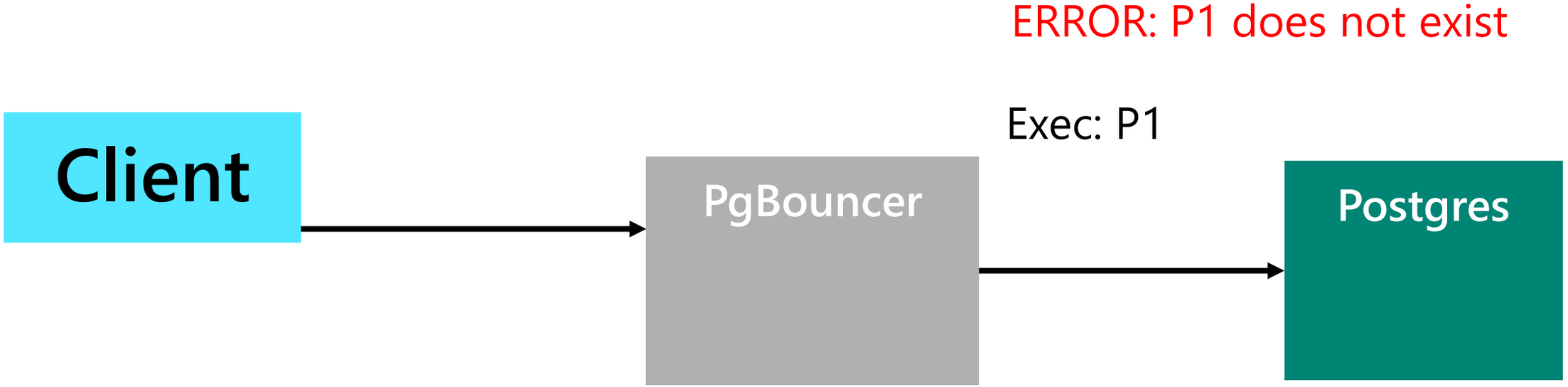
An example prepared statements



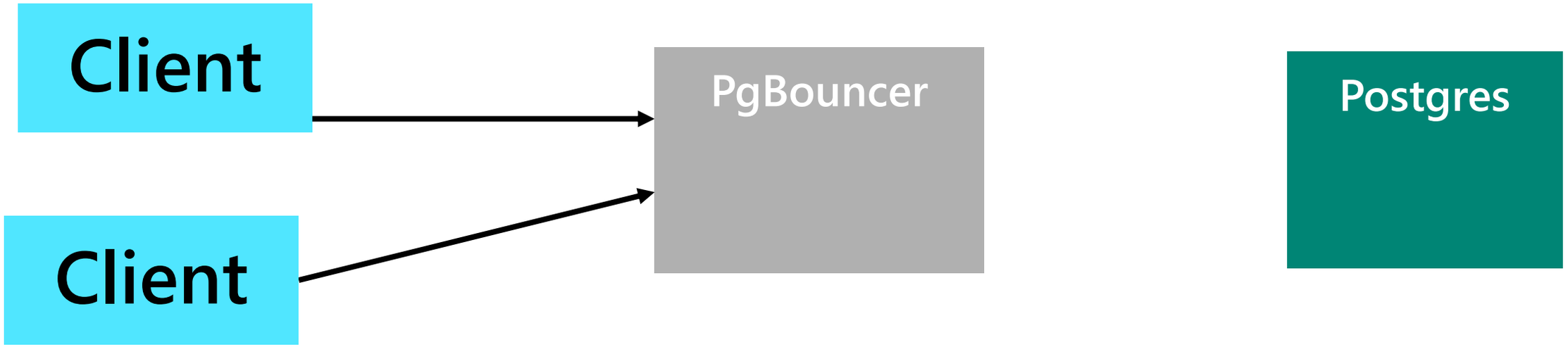
An example prepared statements



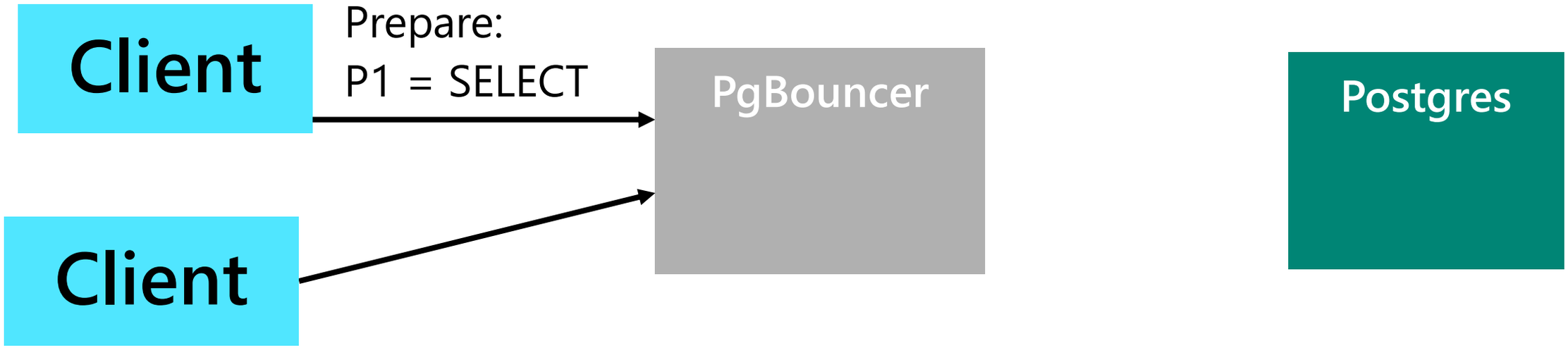
An example prepared statements



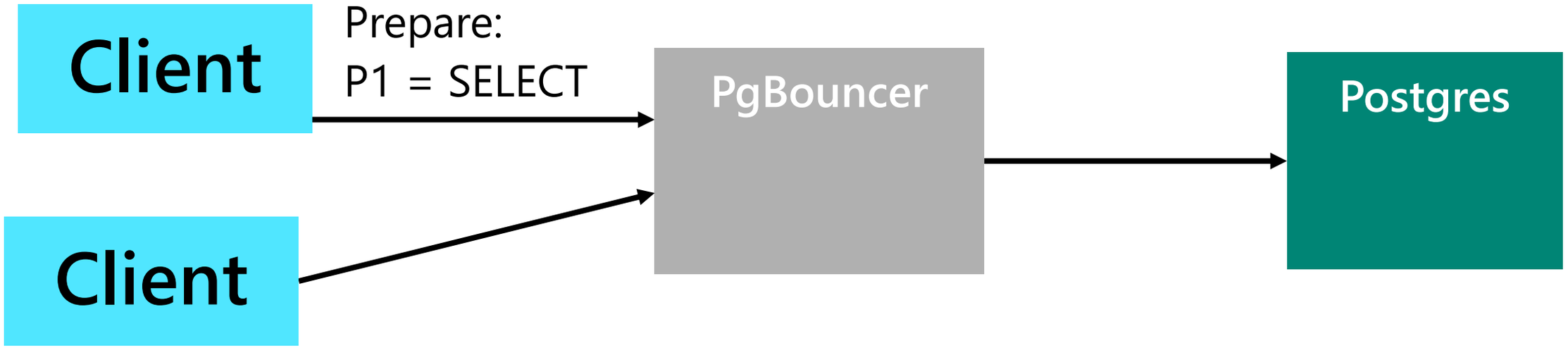
And then you need tricks like this



And then you need tricks like this

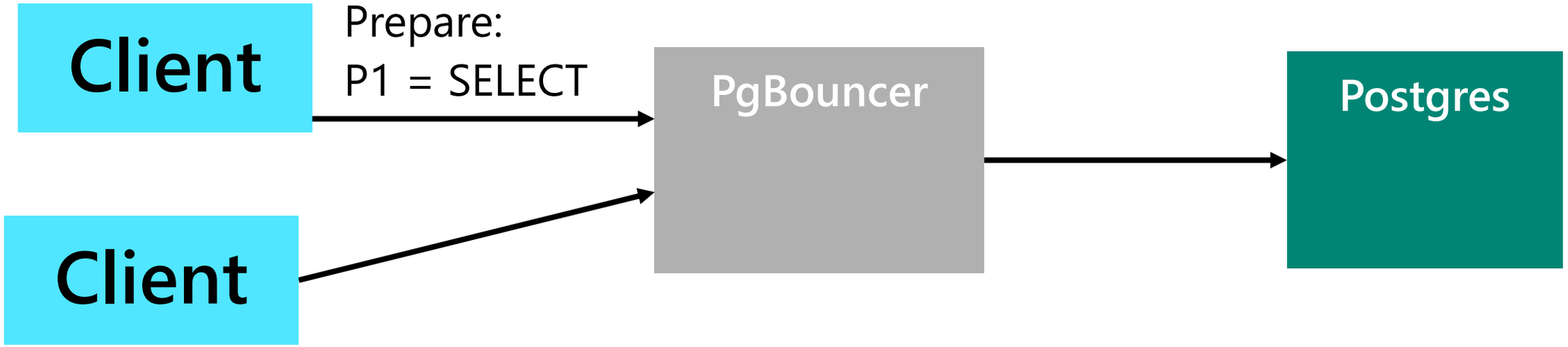


And then you need tricks like this



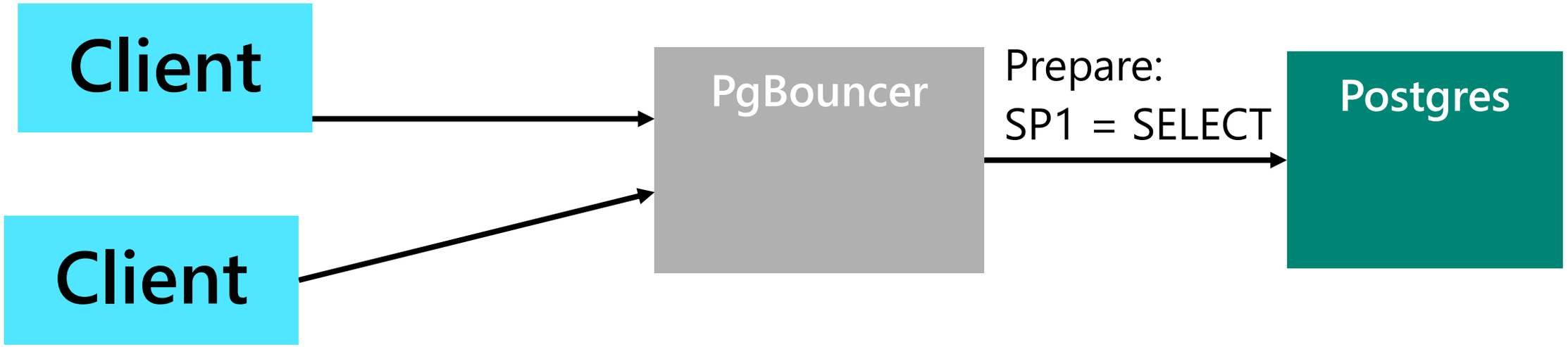
And then you need tricks like this

C1: P1 = SELECT = SP1



And then you need tricks like this

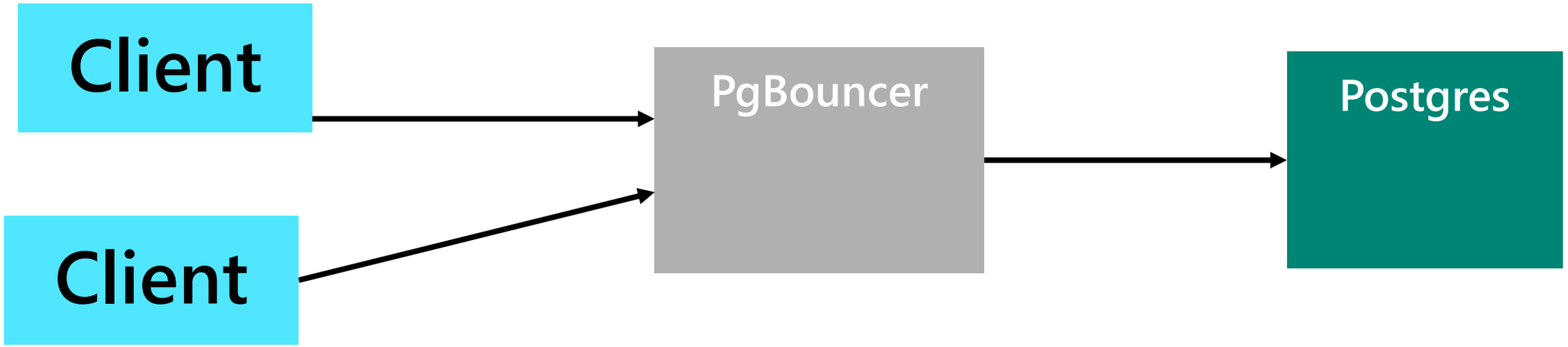
C1: P1 = SELECT = SP1



And then you need tricks like this

C1: P1 = SELECT = SP1

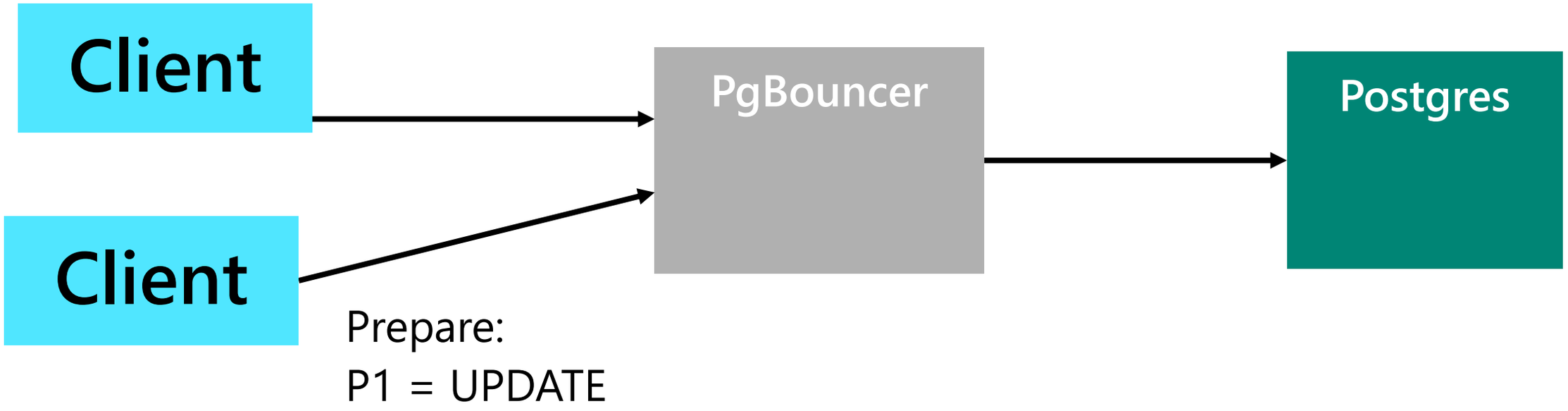
C1: SP1 = SELECT



And then you need tricks like this

C1: P1 = SELECT = SP1

C1: SP1 = SELECT

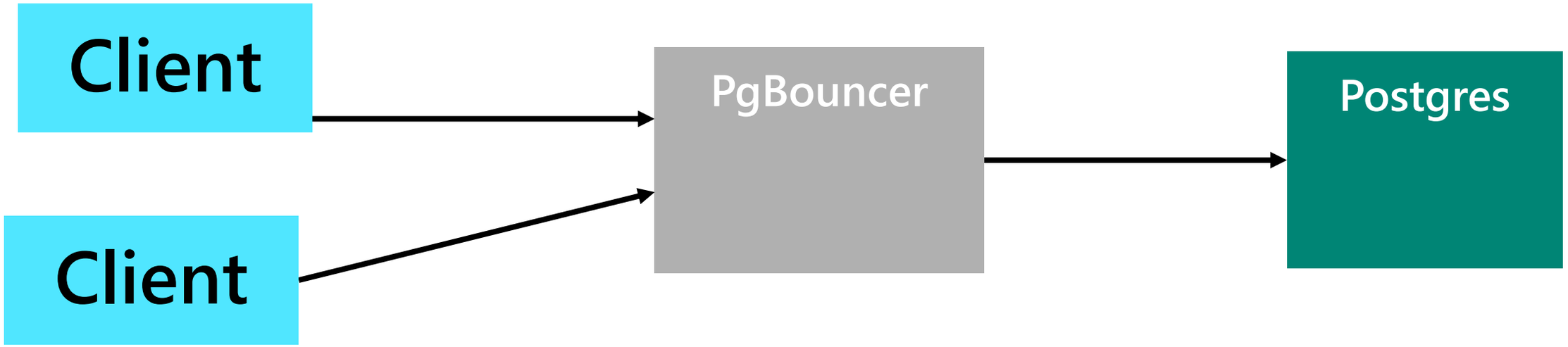


And then you need tricks like this

C1: P1 = SELECT = SP1

C2: P1 = UPDATE = SP2

C1: SP1 = SELECT

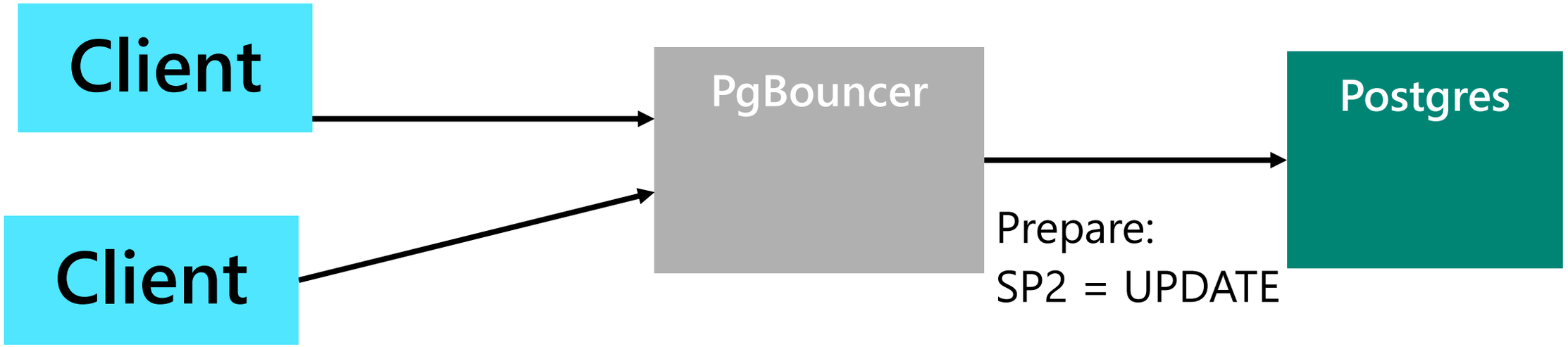


And then you need tricks like this

C1: P1 = SELECT = SP1

C2: P1 = UPDATE = SP2

C1: SP1 = SELECT



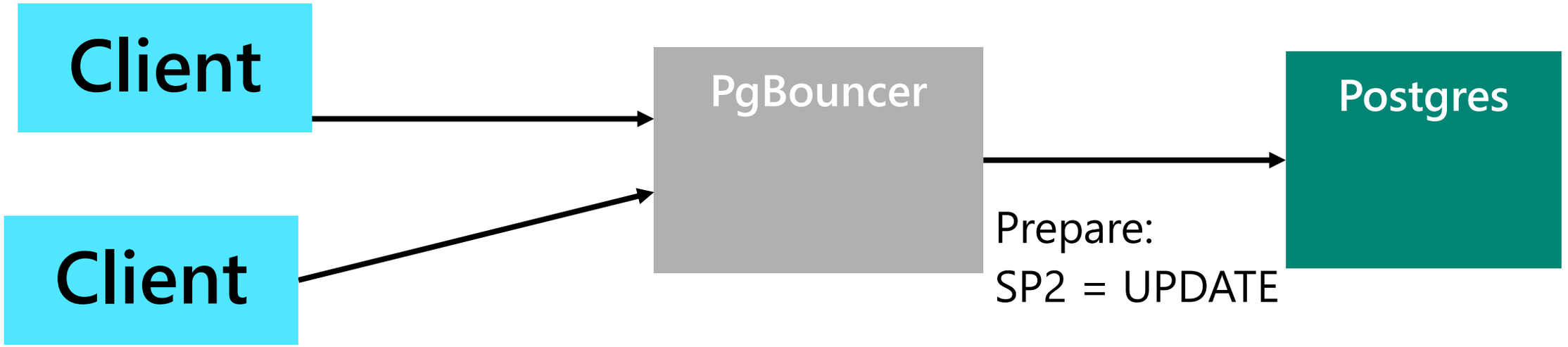
And then you need tricks like this

C1: P1 = SELECT = SP1

C2: P1 = UPDATE = SP2

C1: SP1 = SELECT

C1: SP2 = UPDATE



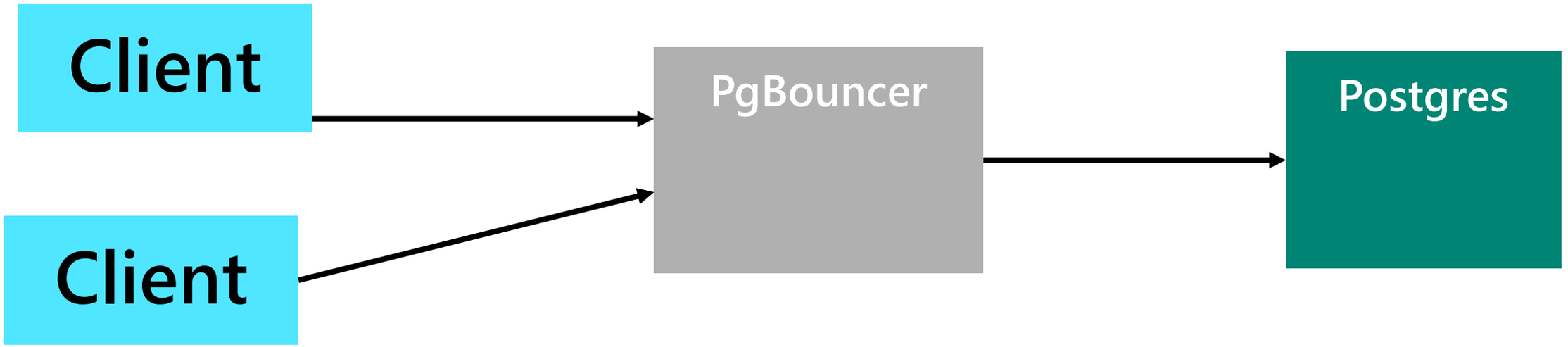
And then you need tricks like this

C1: P1 = SELECT = SP1

C2: P1 = UPDATE = SP2

C1: SP1 = SELECT

C1: SP2 = UPDATE



SUCCESS

And then you need tricks like this

C1: P1 = SELECT = SP1

C1: SP1 = SELECT

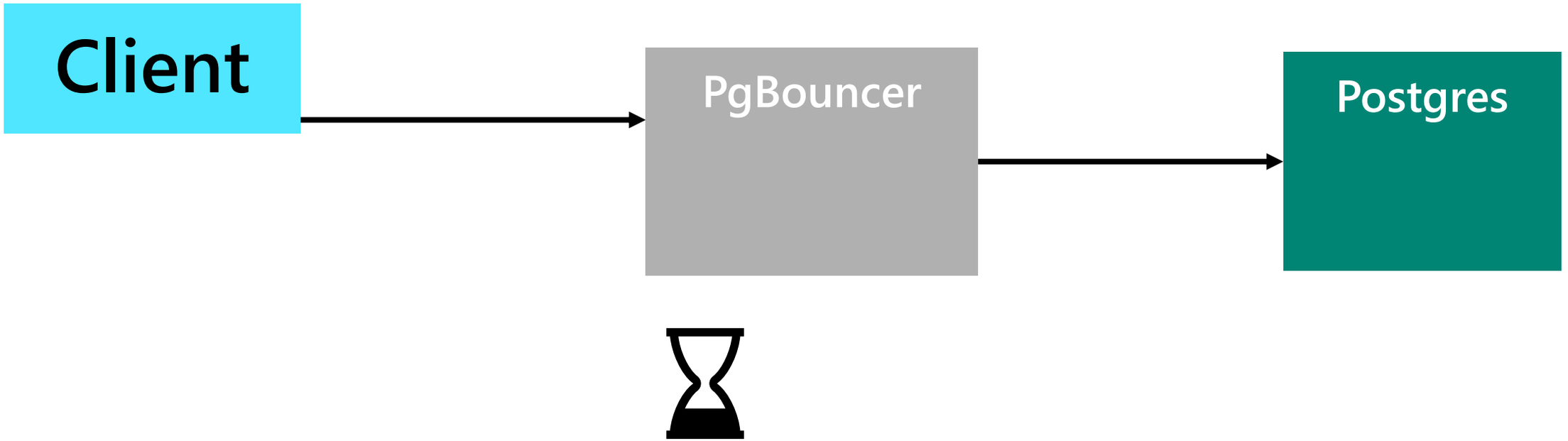
C1: SP2 = UPDATE

And then you need tricks like this

C1: P1 = SELECT = SP1

C1: SP1 = SELECT

C1: SP2 = UPDATE

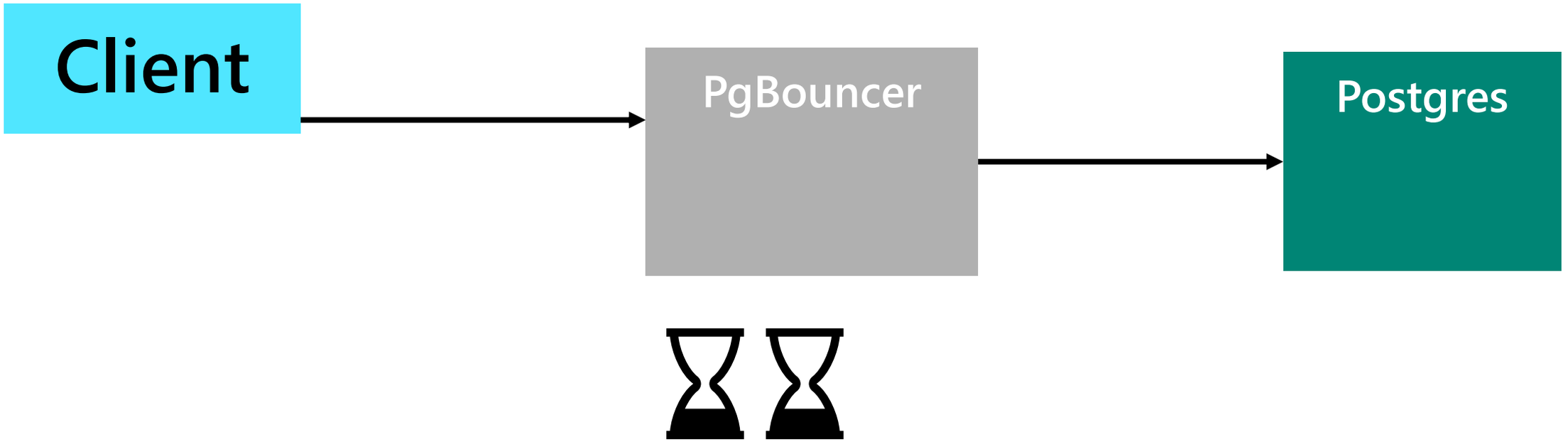


And then you need tricks like this

C1: P1 = SELECT = SP1

C1: SP1 = SELECT

C1: SP2 = UPDATE

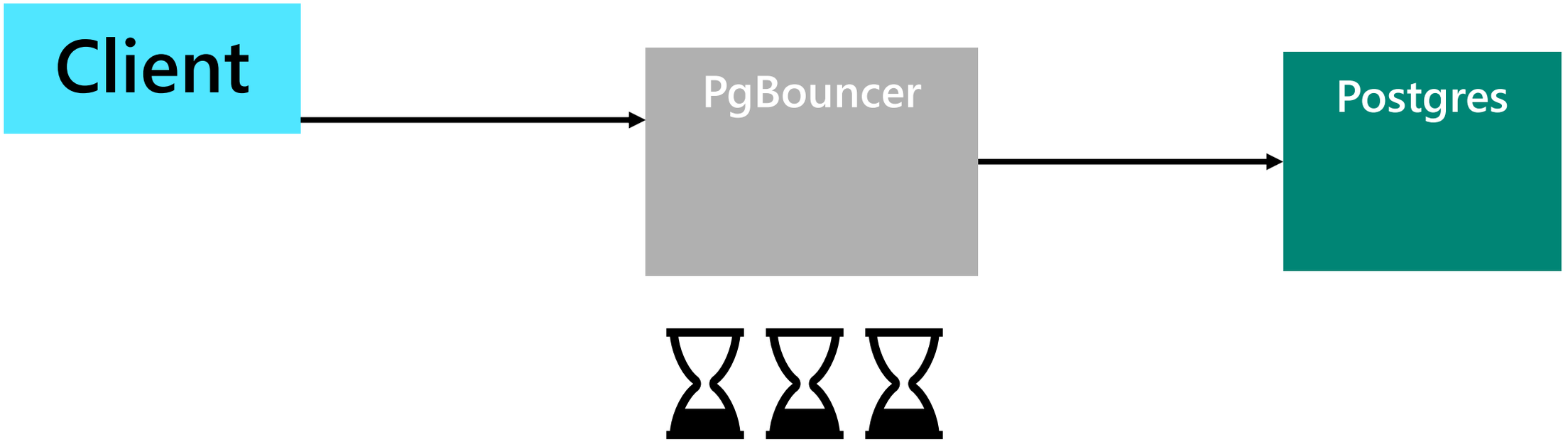


And then you need tricks like this

C1: P1 = SELECT = SP1

C1: SP1 = SELECT

C1: SP2 = UPDATE



And then you need tricks like this

C1: P1 = SELECT = SP1

And then you need tricks like this

C1: P1 = SELECT = SP1

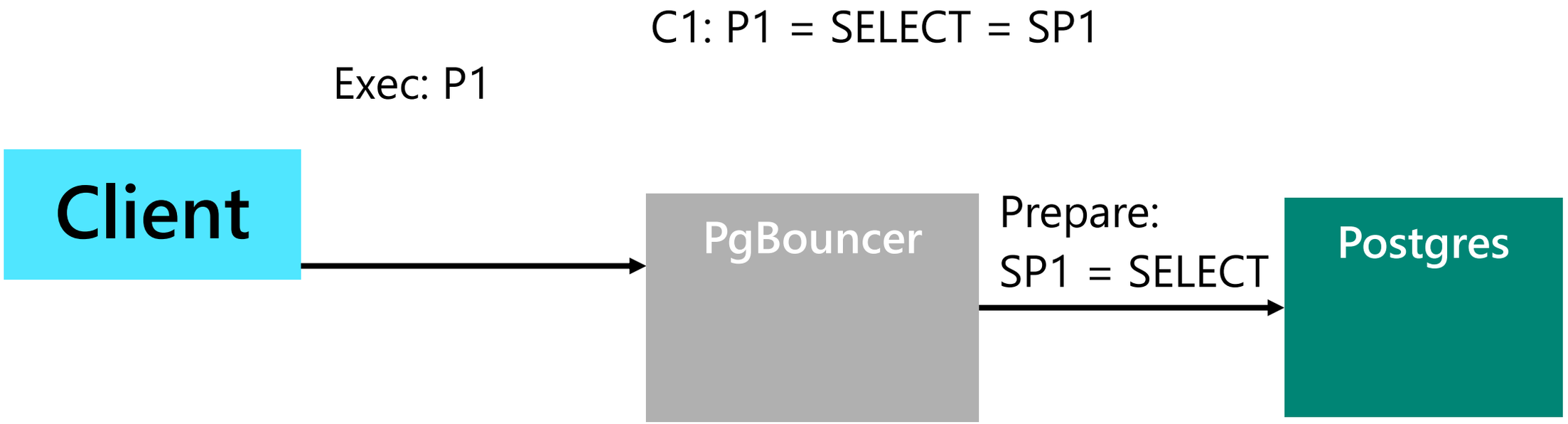
Exec: P1

And then you need tricks like this

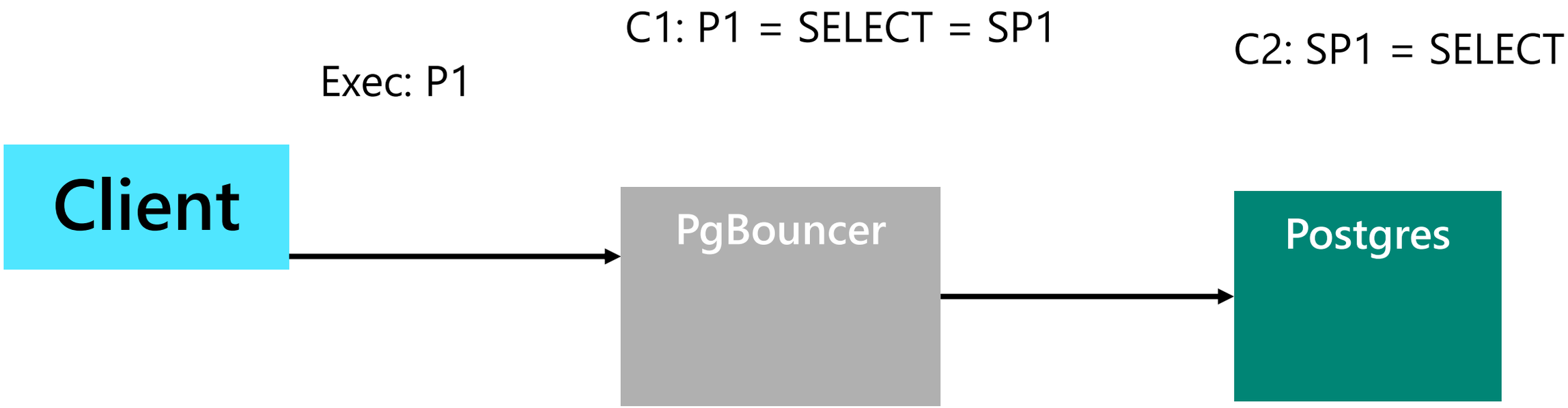
C1: P1 = SELECT = SP1

Exec: P1

And then you need tricks like this



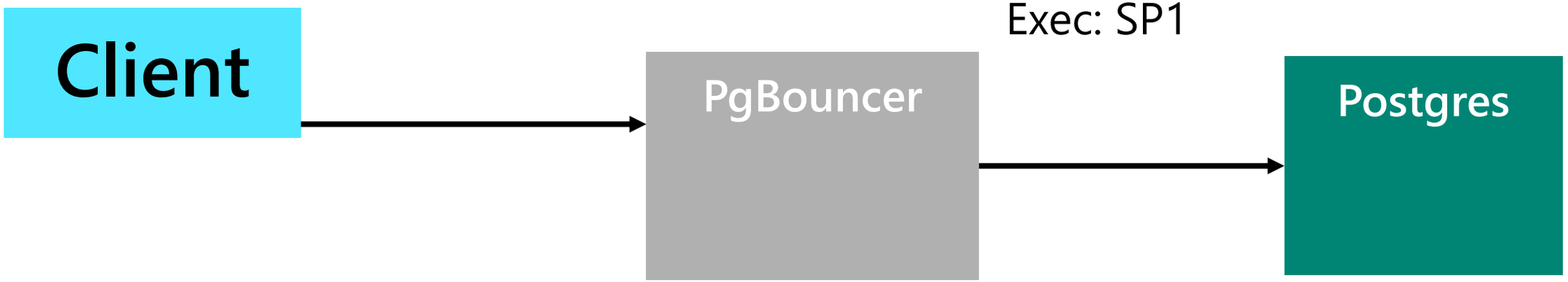
And then you need tricks like this



And then you need tricks like this

C1: P1 = SELECT = SP1

C2: SP1 = SELECT

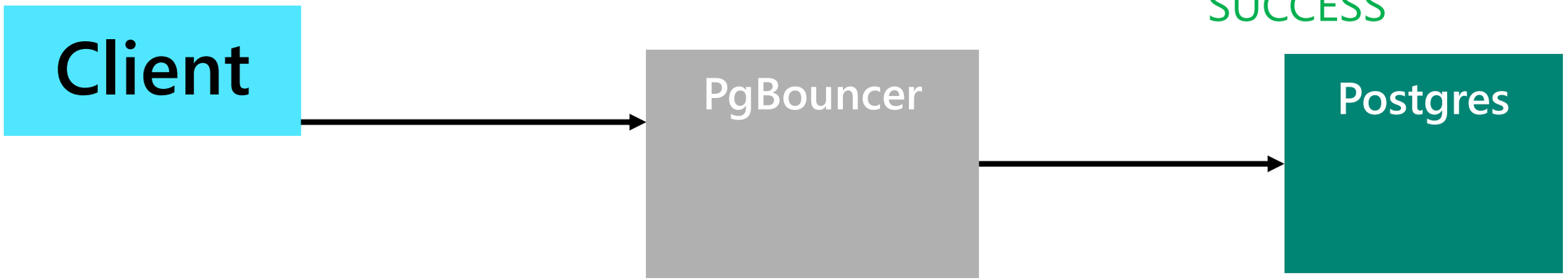


And then you need tricks like this

C1: P1 = SELECT = SP1

C2: SP1 = SELECT

SUCCESS



Old!

No changes in 20 years

One change: NegotiateProtocolVersion

- Introduced 5 years ago
- Allows server to trigger protocol version downgrade
- Or advertise non-support for requested protocol extensions
- So far unused
- Active discussion ongoing how we should use it

The Future

Direct TLS

Direct TLS

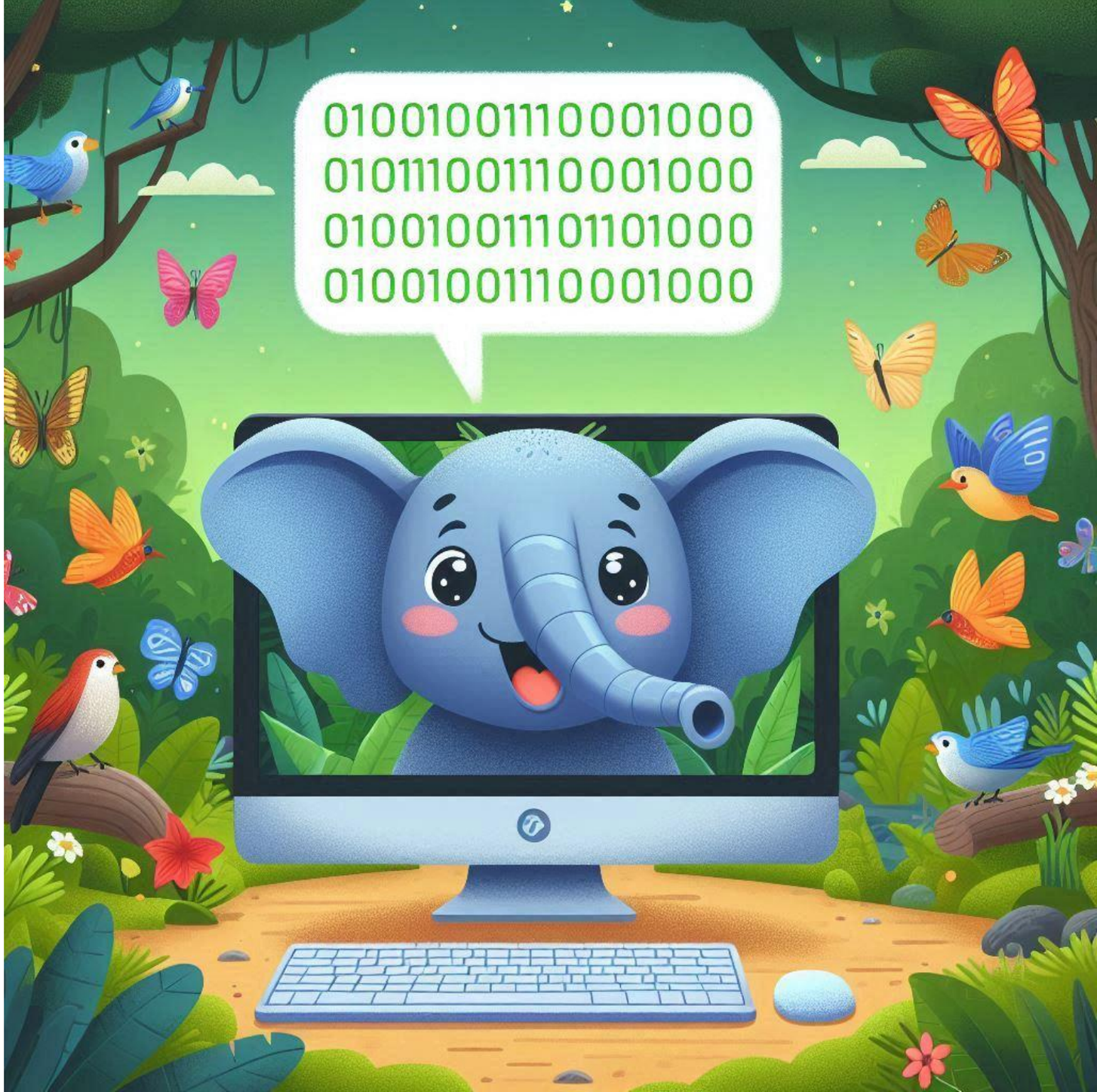
- One less round trip
- Can use off-the-shelf TLS proxies
- sslnegotiation=direct
- Committed in PG17

Compression

Compression

- compression = lz4
- CompressedData()
- Compression might weaken encryption

Automatic binary encoding



Automatic binary encoding

- Binary encoding is usually faster
- Some text some binary
- Needs extra round trip to find out which
- binary_formats=INTOID,TIMESTAMPOID

Smaller rows

Reducing the size of DataRow

- Every cell for every row contains length
- Wasteful for fixed-length cells
- Number of columns is also reported for every row
- RowDescription can be used for these
- No official proposal yet

Or even more radical

- Mix between columnar and row based
- Helps with compression
- Relevant paper with Postgres POC: https://www.vldb.org/pvldb/vol10/p1022-muehleisen.pdf
- Shows 4x-8x improvement

Automatic RowDescription

Automatic RowDescription

- RowDescription is usually the same for the same query
- Wasteful to request over and over for prepared statement
- Changes after DDL
- Causes errors when using connection poolers
- “SELECT * FROM table” is the worst
- Proposal: Notify the client know when this happens
- https://commitfest.postgresql.org/48/4518/

Configurable GUC_REPORT

Configurable GUC_REPORT

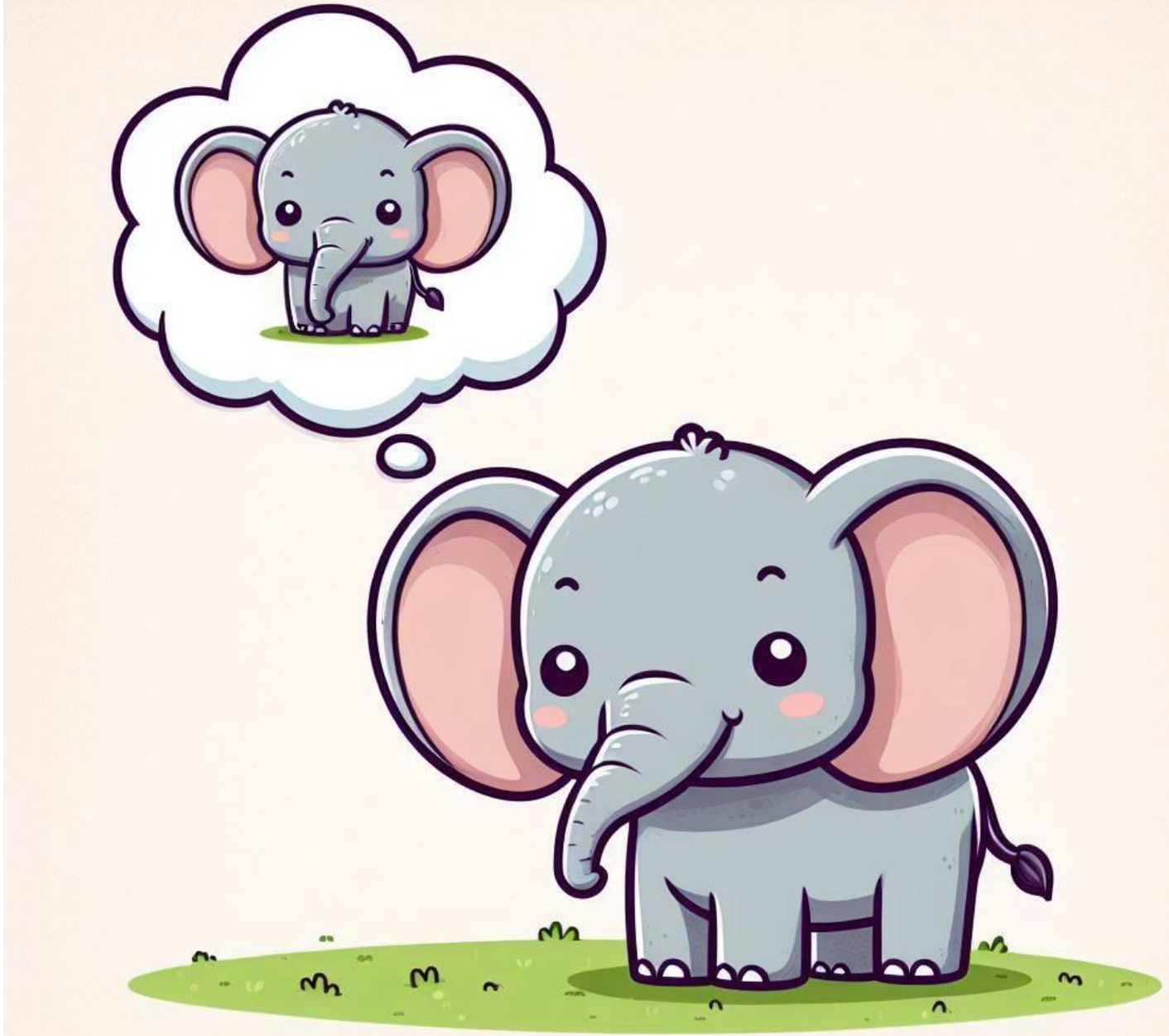
- ParameterStatus reports changes to GUCs with GUC_REPORT
- Very useful for connection poolers
- Currently hardcoded list
- search_path by far most requested

Larger secret in BackendData

Larger secret in BackendData

- Secret in BackendData is 32 bits
- Not a lot security wise
- Proxies/poolers encode metadata in secret

Meta: ParameterSet



ParameterSet

- Changing protocol parameters after startup
- Critical for poolers
- Needs protocol message for security

What now?

What now?

1. Consensus on how to use NegotiateProtocolVersion
2. Consensus on if protocol parameter are GUCs or not
3. Get ParameterSet in
4. Get more protocol changes in
5. Add support for the protocol changes to popular poolers

Microsoft Azure

Any questions?