Microsoft Azure

Postgres load balancing is secretly broken: The cancellation problem

Jelte Fennema (@JelteF)

Developing Citus and Postgres at Microsoft & PgBouncer maintainer

2022-09-14

What am I going to talk about?

- Load balancing across Postgres servers
- Read replicas
- Citus
- Low level details about cancelling queries
- PgBouncer

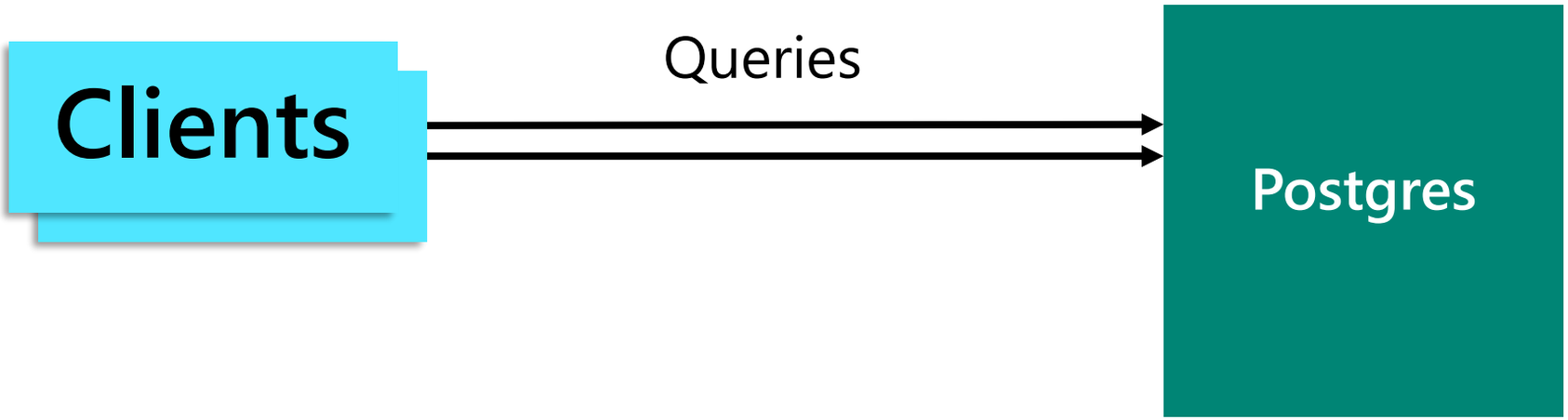
What is load balancing?

- Sharing workload across servers
- Different servers handle independent requests

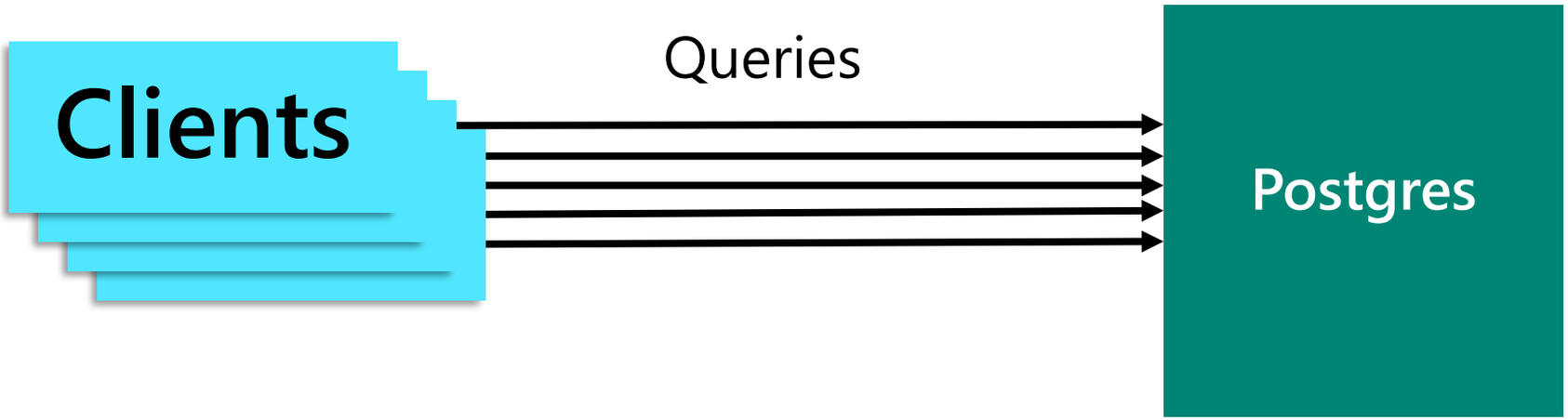
Why load balancing for Postgres?

- Performance
- Scaling reads with Postgres read replicas
- Scaling writes with the Citus extension for Postgres

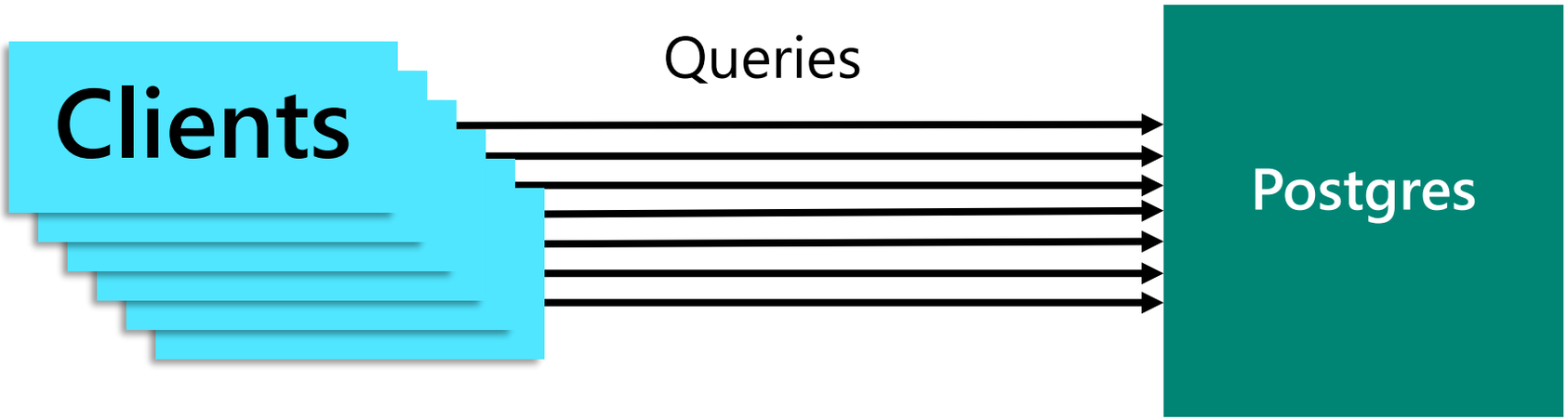
Scaling reads or writes? Why?



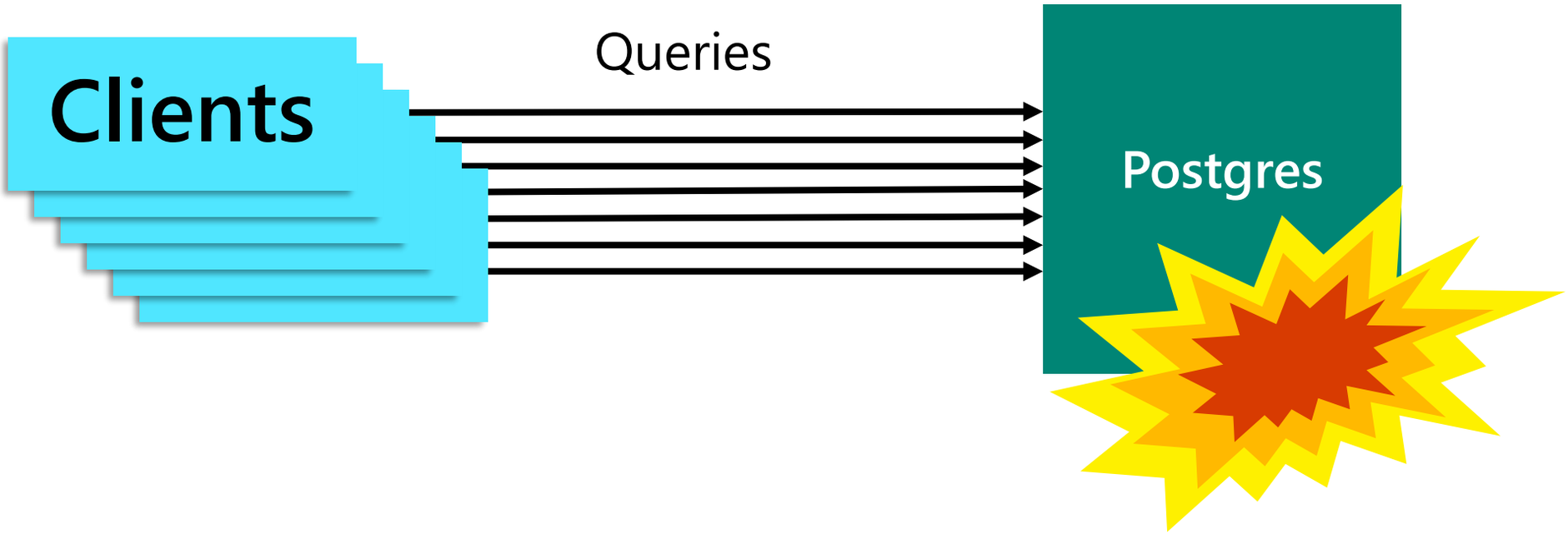
Scaling reads or writes? Why?



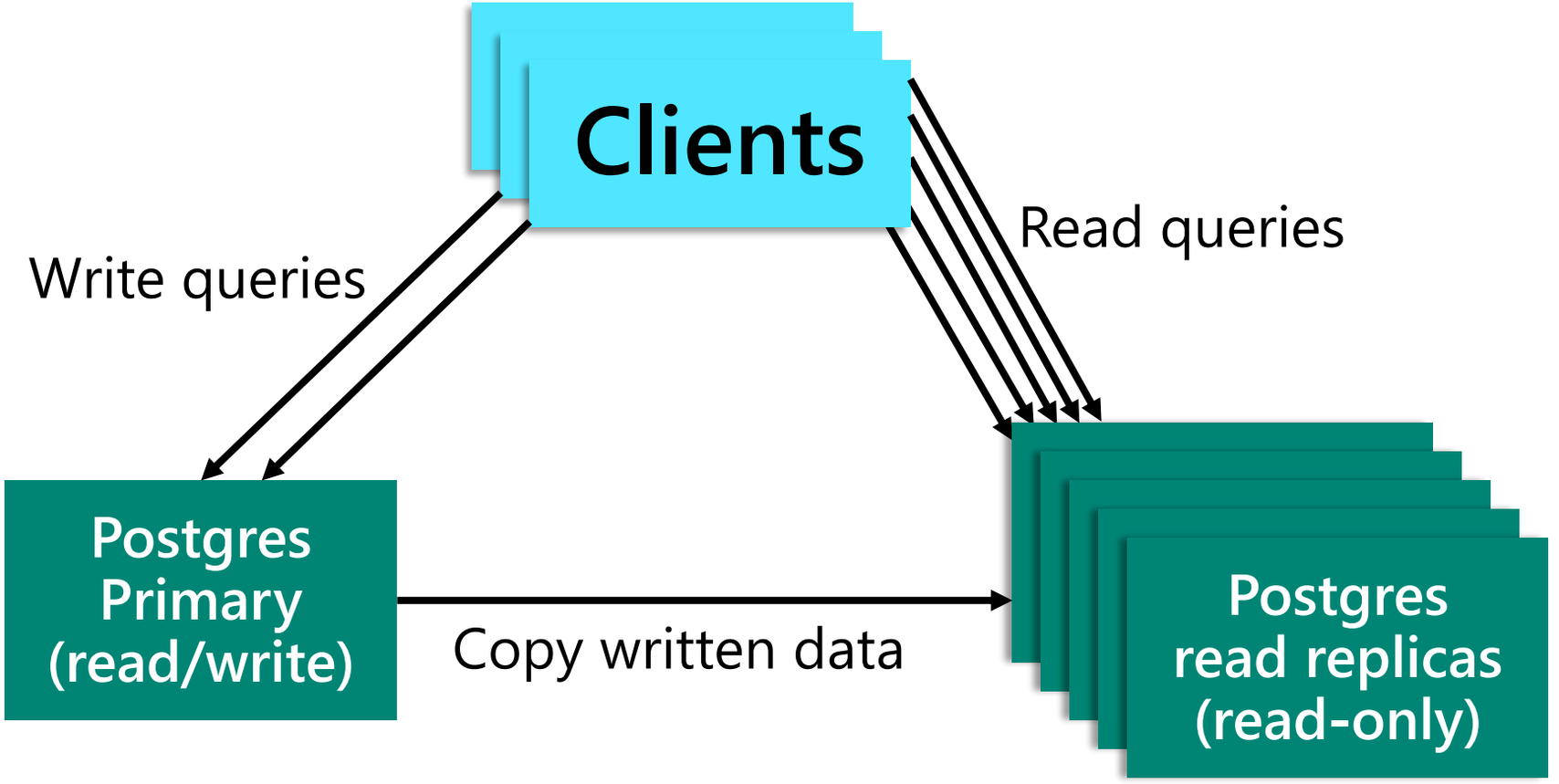
Scaling reads or writes? Why?



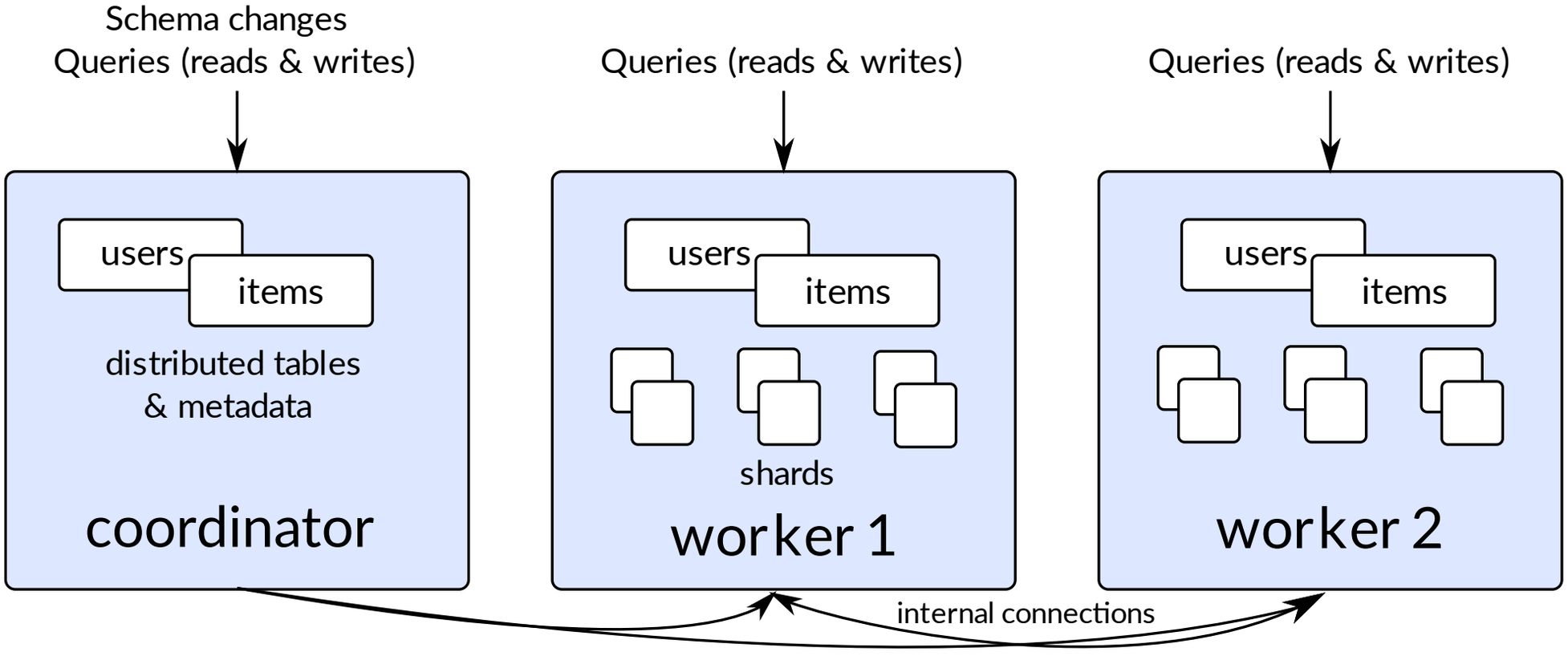
Scaling reads or writes? Why?



What are read replicas?



What is Citus?



How to do load balancing?

- Option 1: Client-side load balancing
  Client knows about all servers and connects to all of them
- Option 2: Server-side load balancing
  Client only knows a single logical server, that server “secretly” forwards requests to multiple servers

Client-side load balancing explained

Common approaches:

- Hardcoded list of IPs or domains used by client
- Multiple DNS records for the same domain, one for each server
  - With smart client
  - Or with round-robin resolver

Downsides:

- Every client needs to know about all the servers
- Every client needs logic to choose a server
- Caching: what happens when you add a server

Client-side load balancing with Postgres

- JDBC (Java) and Npgsql (C#) support
- PgBouncer released client-side load balancing last March in v1.17.0
- libpq **does not** support client-side load balancing natively
  I submitted a patch to add support for this: https://commitfest.postgresql.org/39/3679/
- Many DNS resolvers always return a single fixed result
  So, a single client does not load balance

Server-side load balancing explained

Common approaches:
- Proxy server
- Software defined networking (SDN)

Problems:
- Proxy server introduces latency
- Costs extra money

Big advantage:
- Clients don’t need special support and/or configuration

Server-side load balancing with Postgres

Options:

1. A dedicated PgBouncer server configured in client-side load balancing mode

- Can use transaction or session load balancing
- Extra network hop introduces latency
- Single threaded

2. Off the shelf TCP load balancer

- Each TCP stream is assigned to a different server
- Probably has “fancy” things like health-checks
- No extra network hop in case of SDN based load balancer

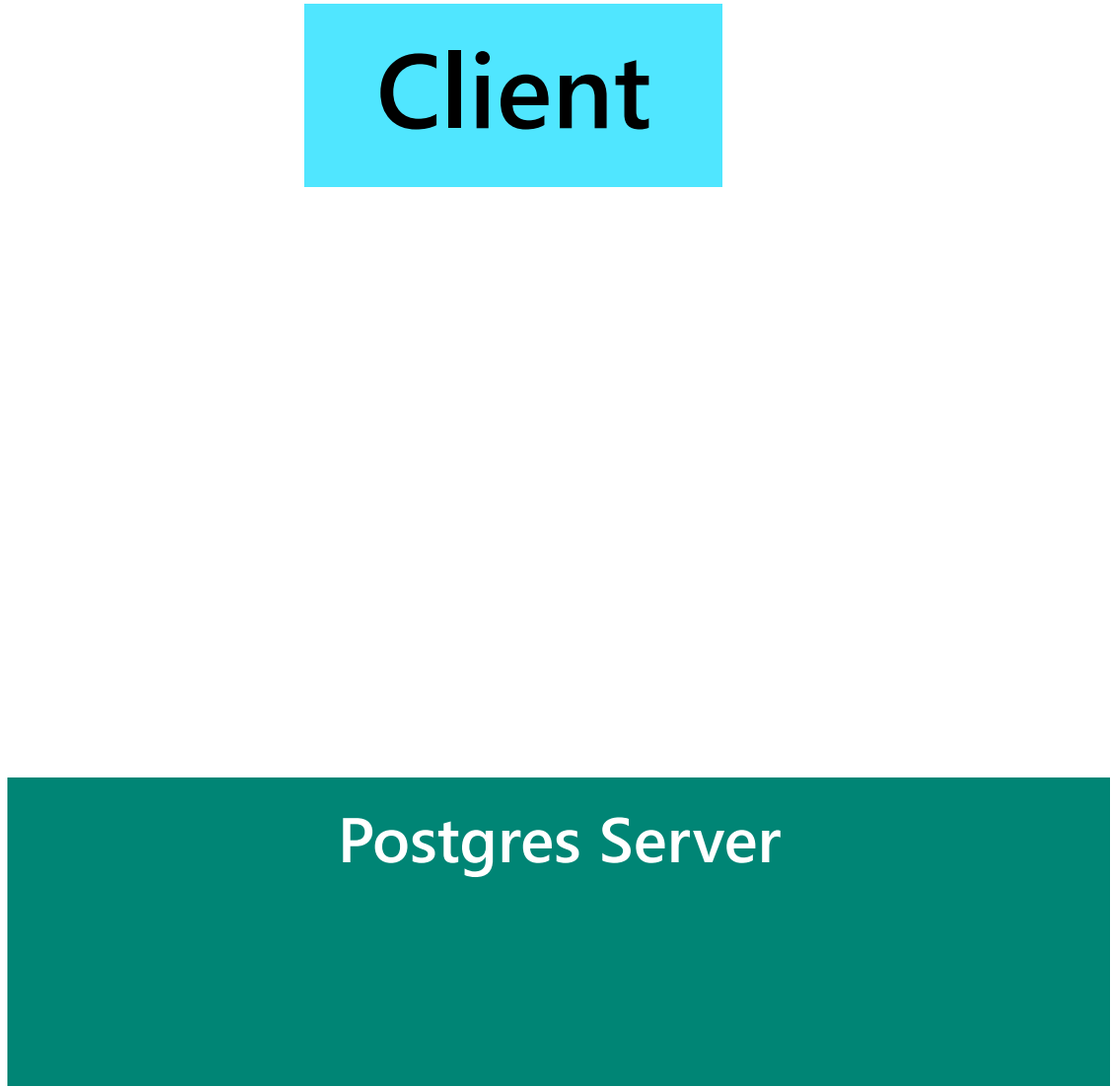
Decision seems simple

Off the shelf TCP based load balancer it is

Not so fast

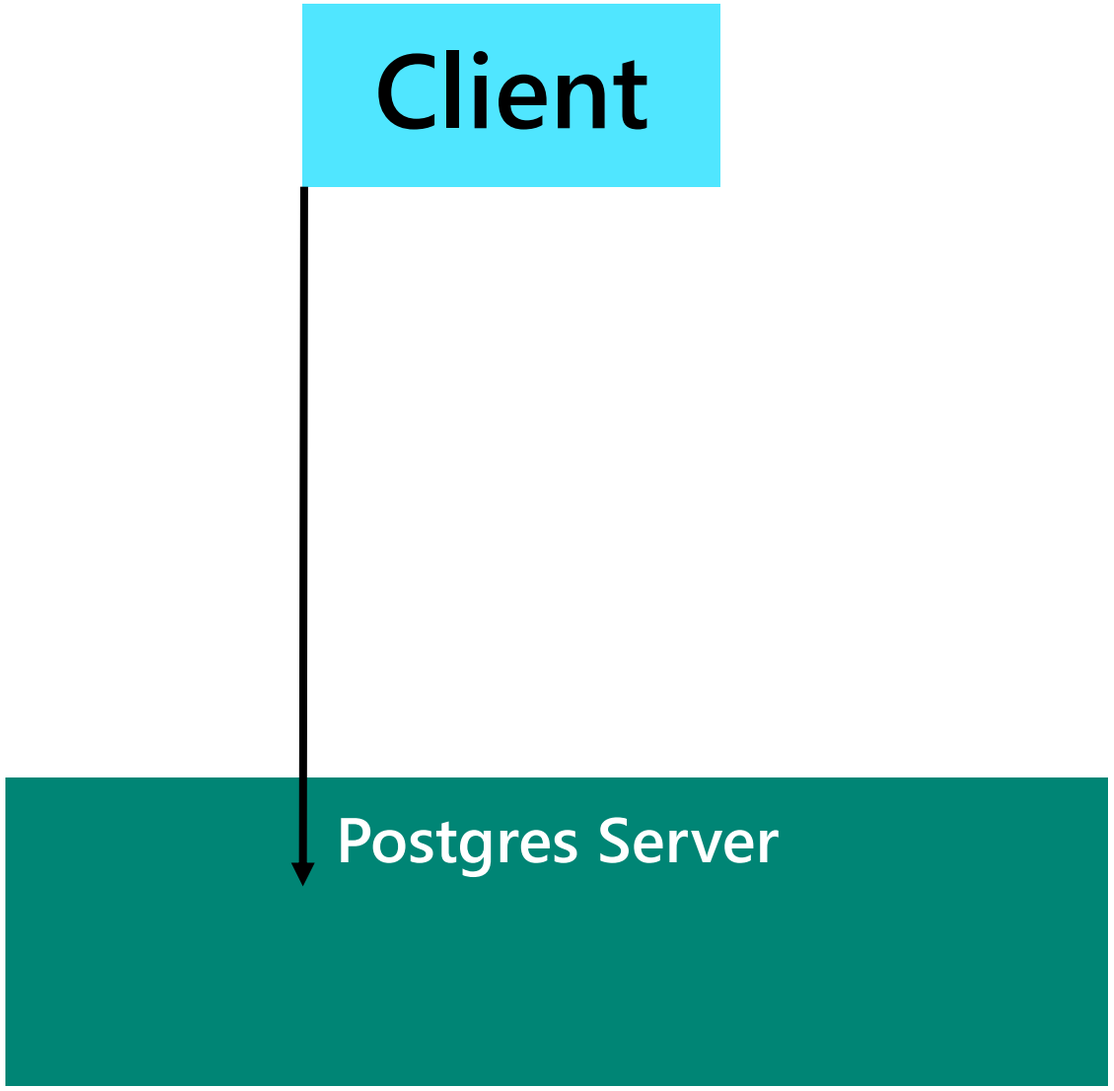
- Important assumption: Different TCP streams are independent

How Postgres cancellation requests work



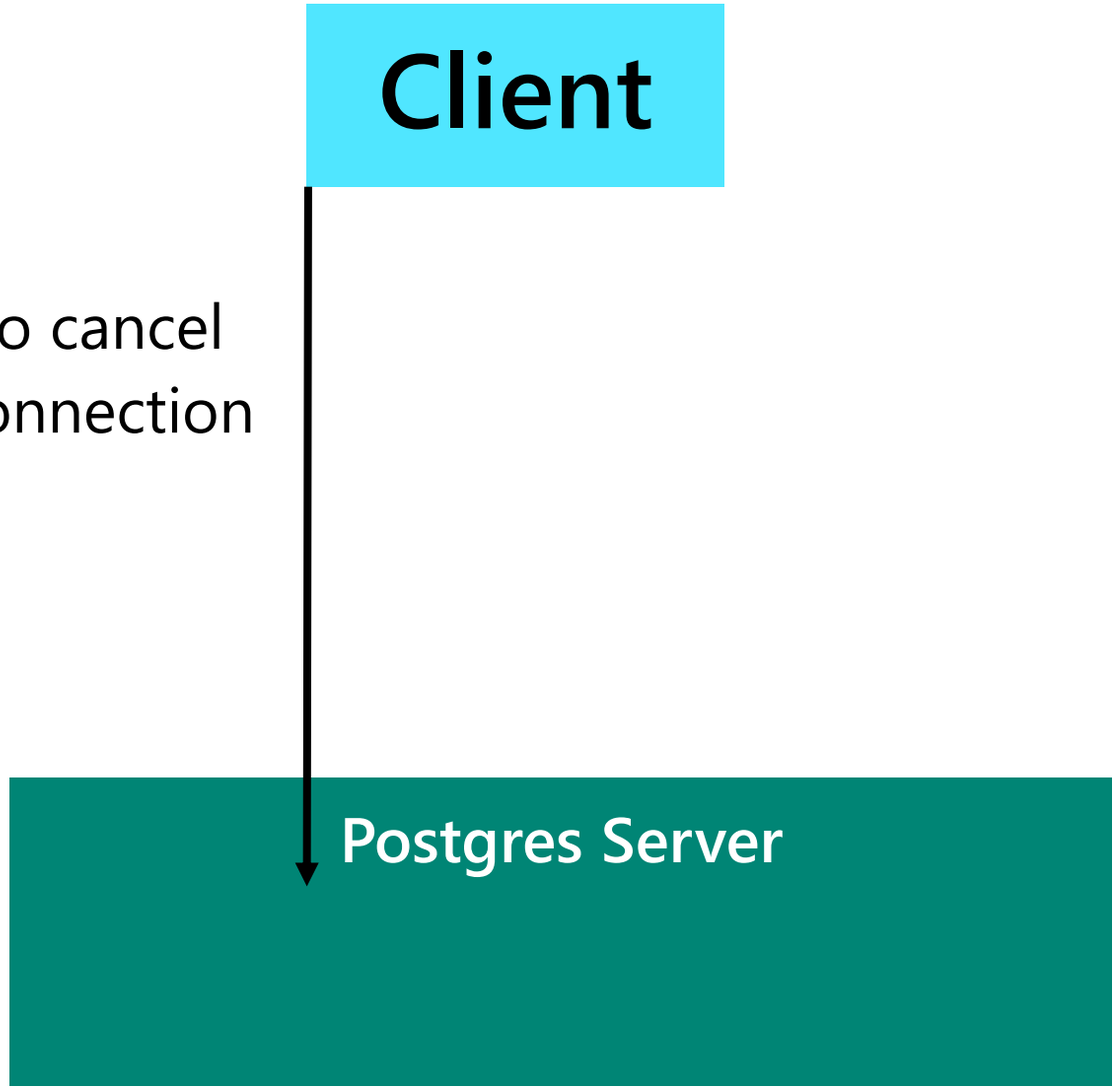
How Postgres cancellation requests work

-> CONNECT



How Postgres cancellation requests work

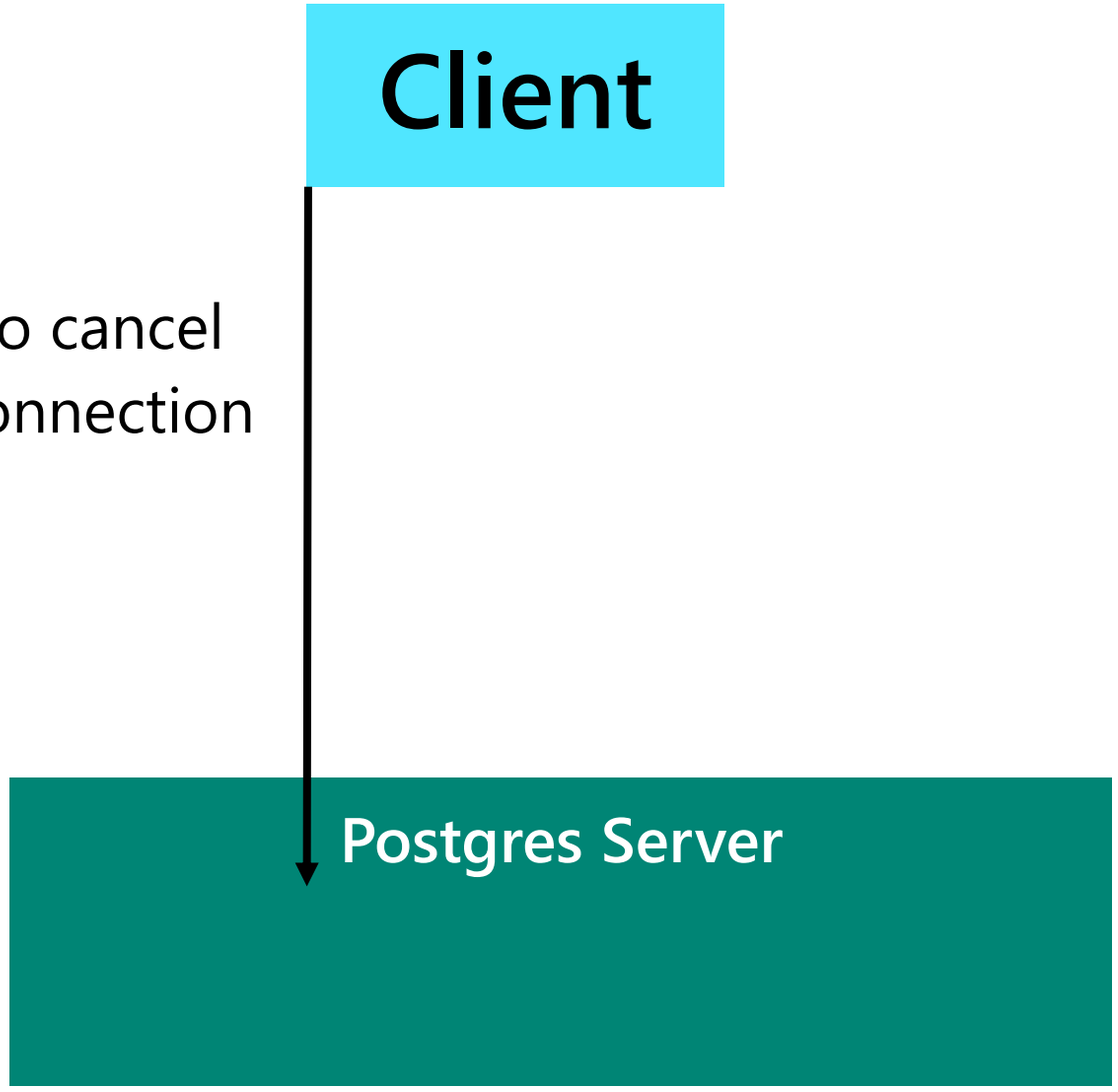
-> CONNECT
<- You can send me
SECRET-TOKEN-123 to cancel
any queries on this connection



How Postgres cancellation requests work

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

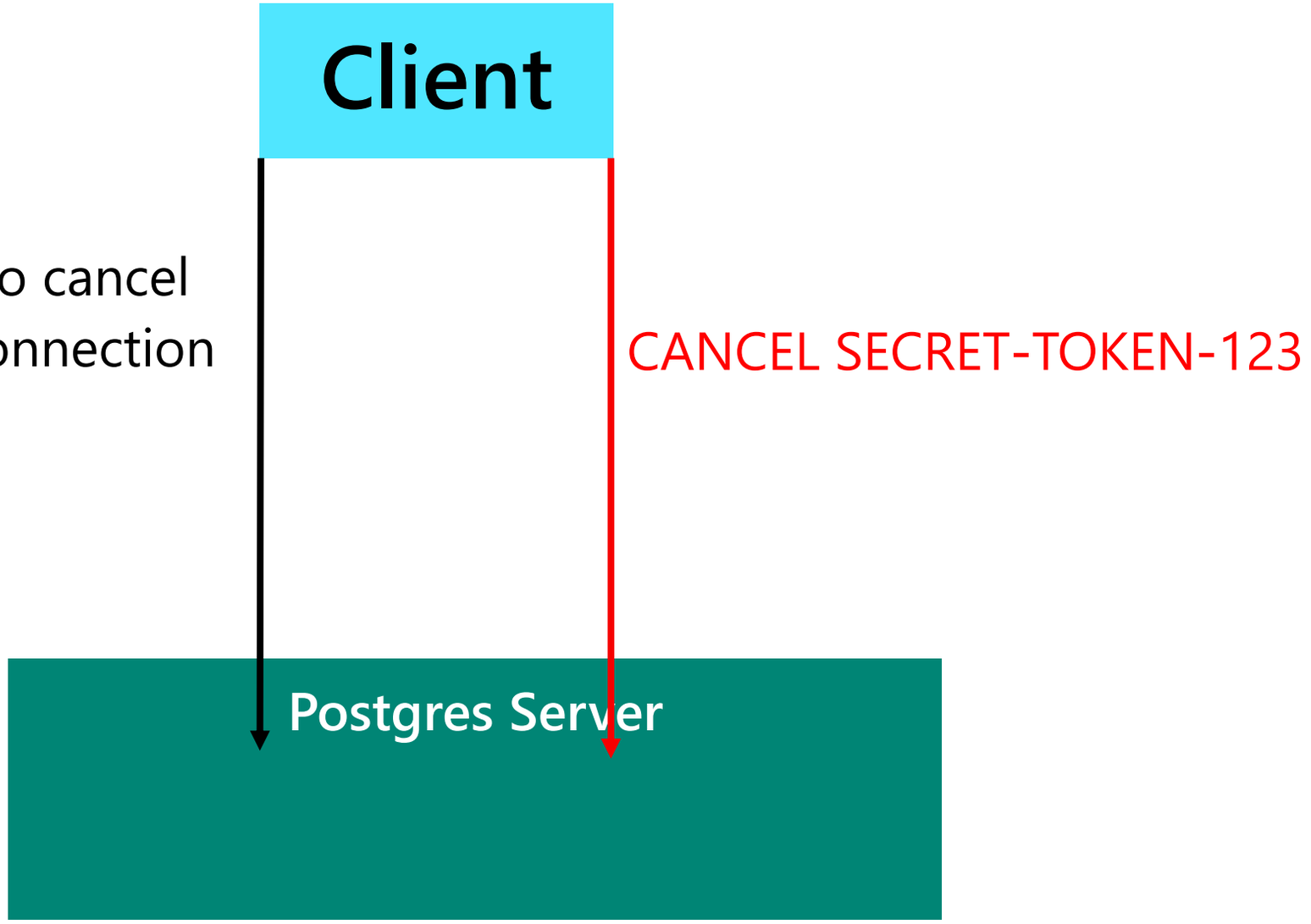
-> RUN:
DELETE FROM users;



How Postgres cancellation requests work

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

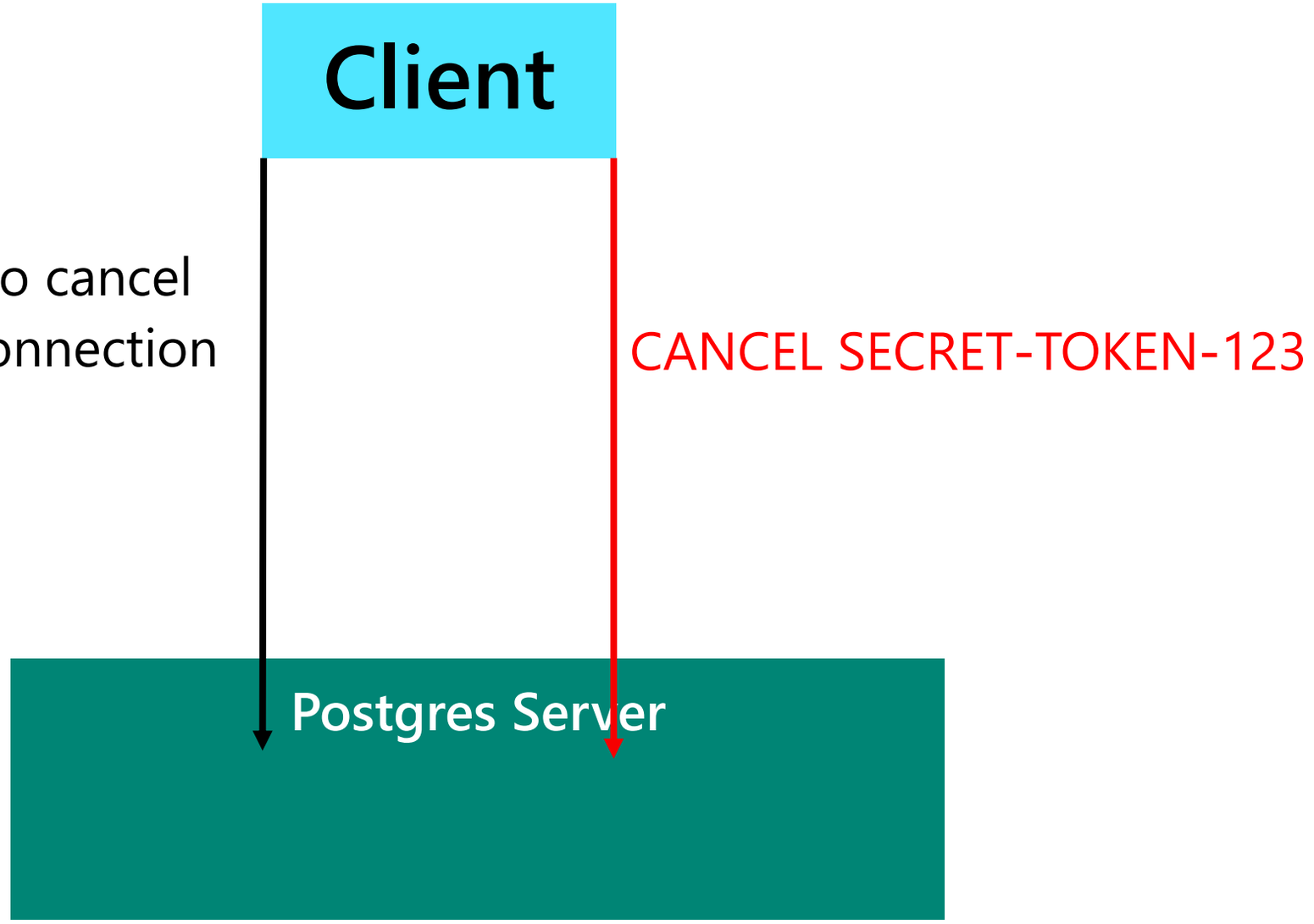
-> RUN:
DELETE FROM users;



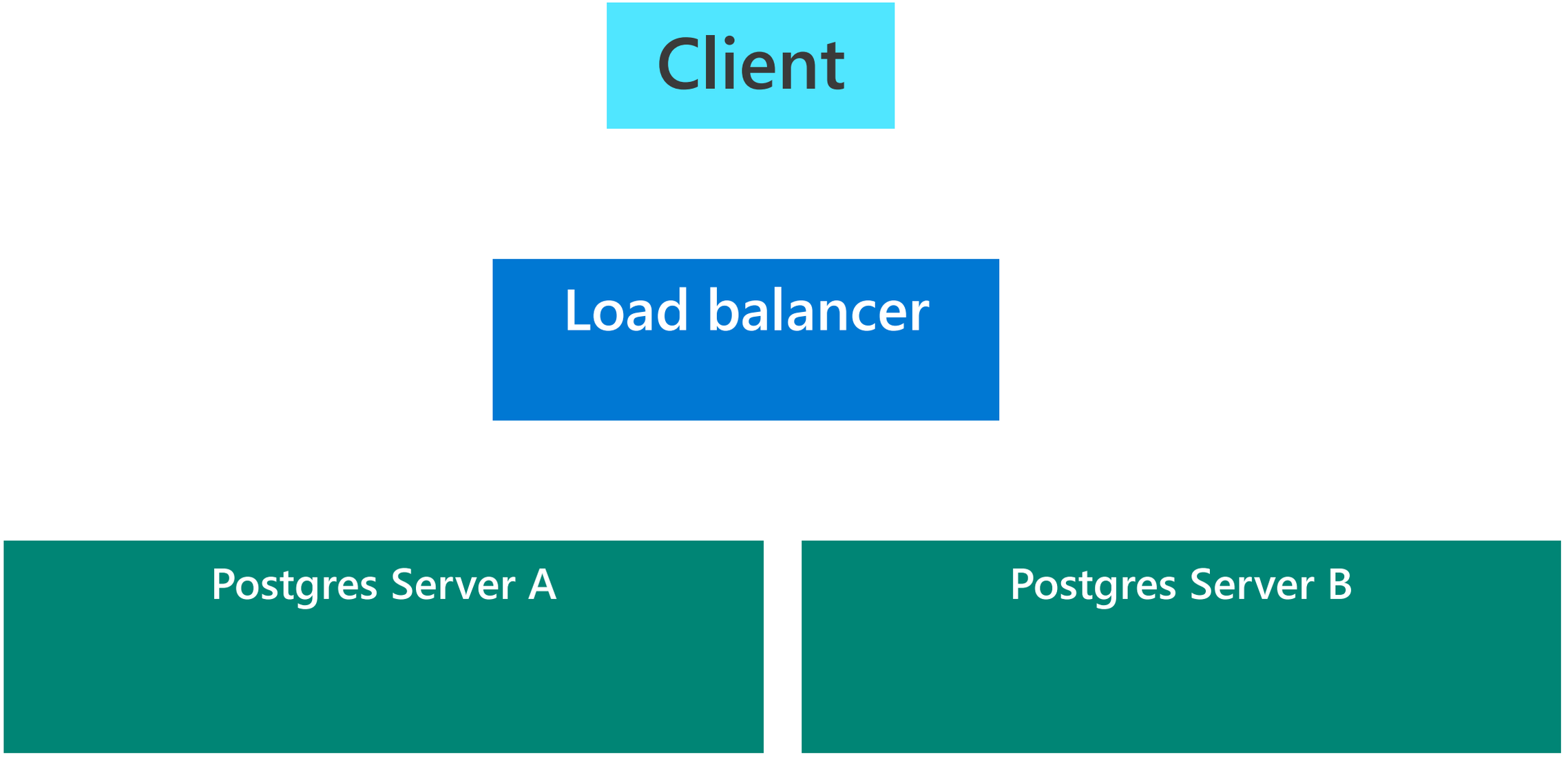
How Postgres cancellation requests work

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

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

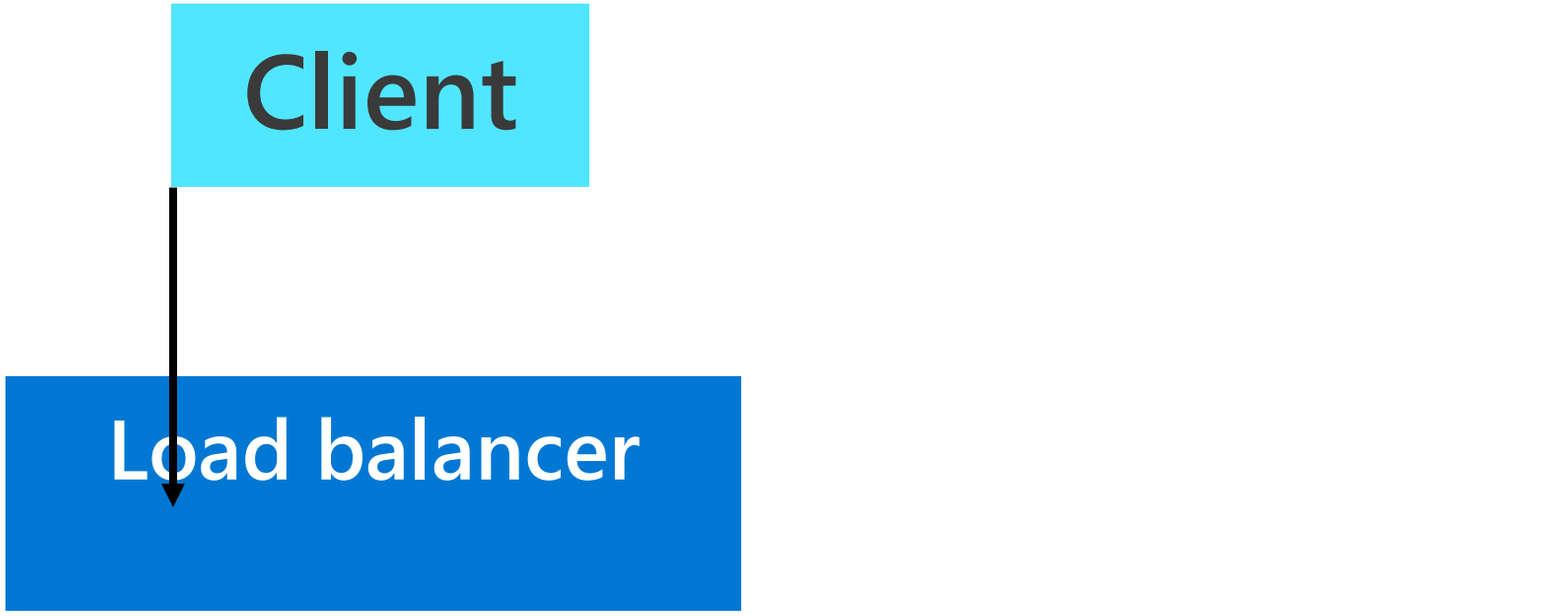


So what happens with cancellations and a load balancer



So what happens with cancellations and a load balancer

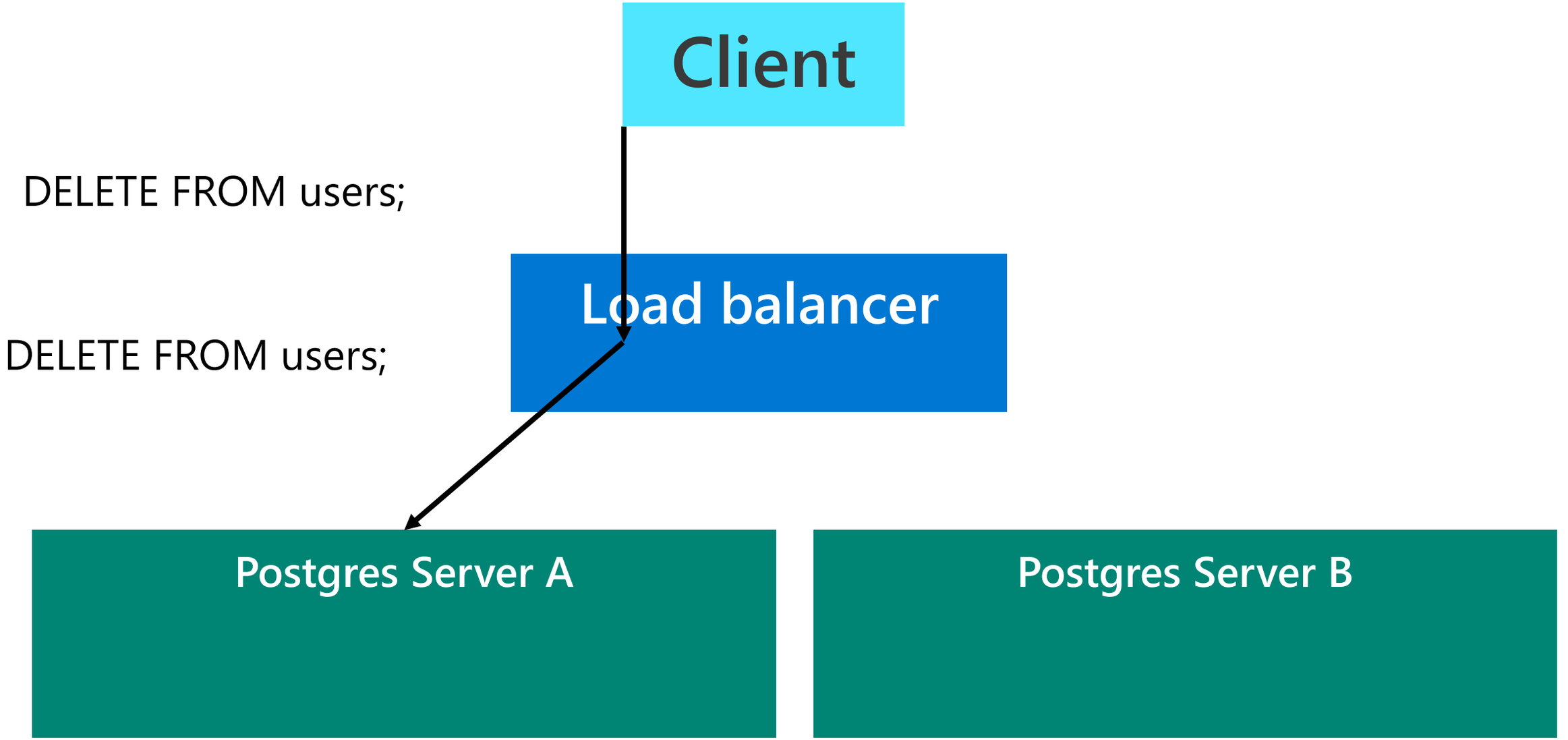
DELETE FROM users;



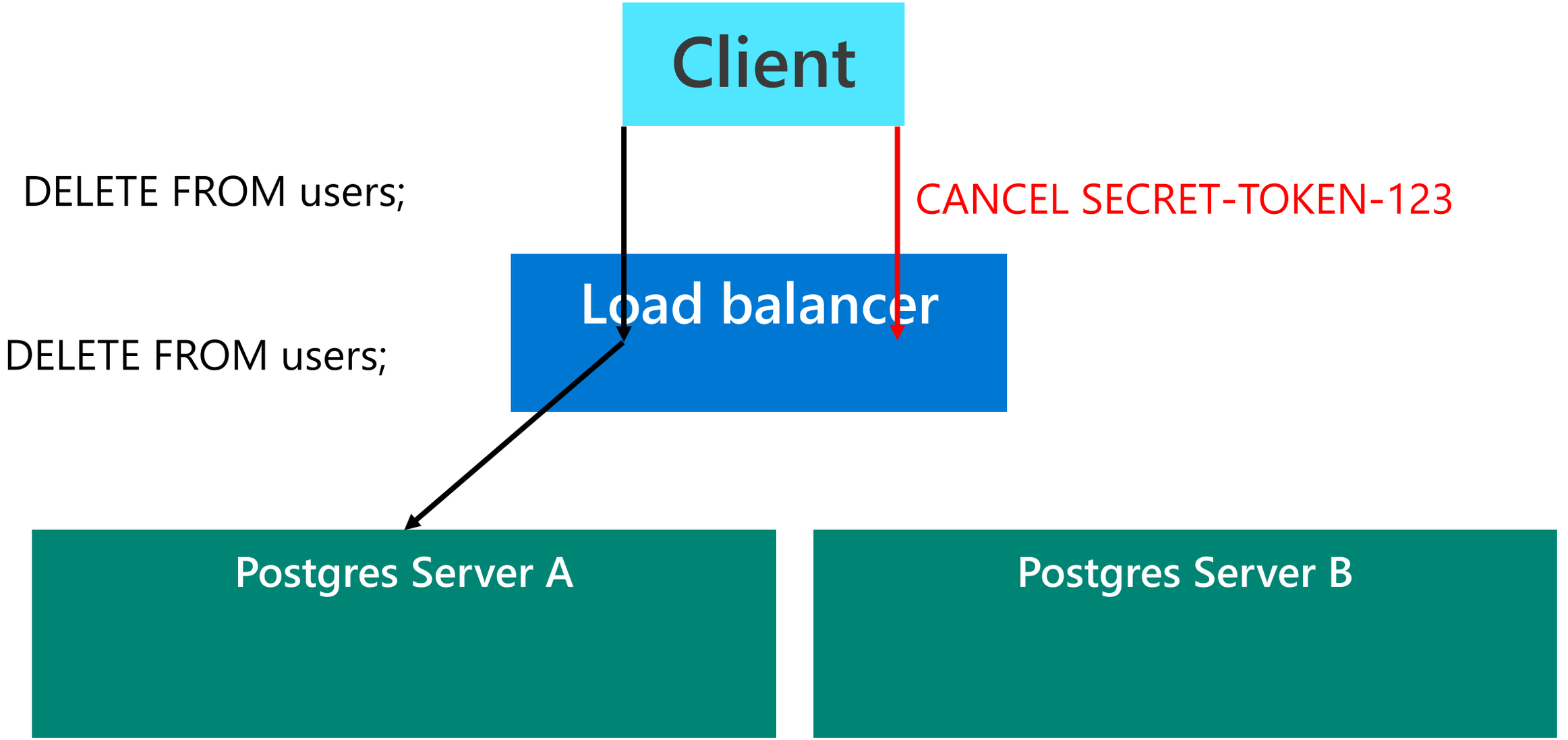
Postgres Server A

Postgres Server B

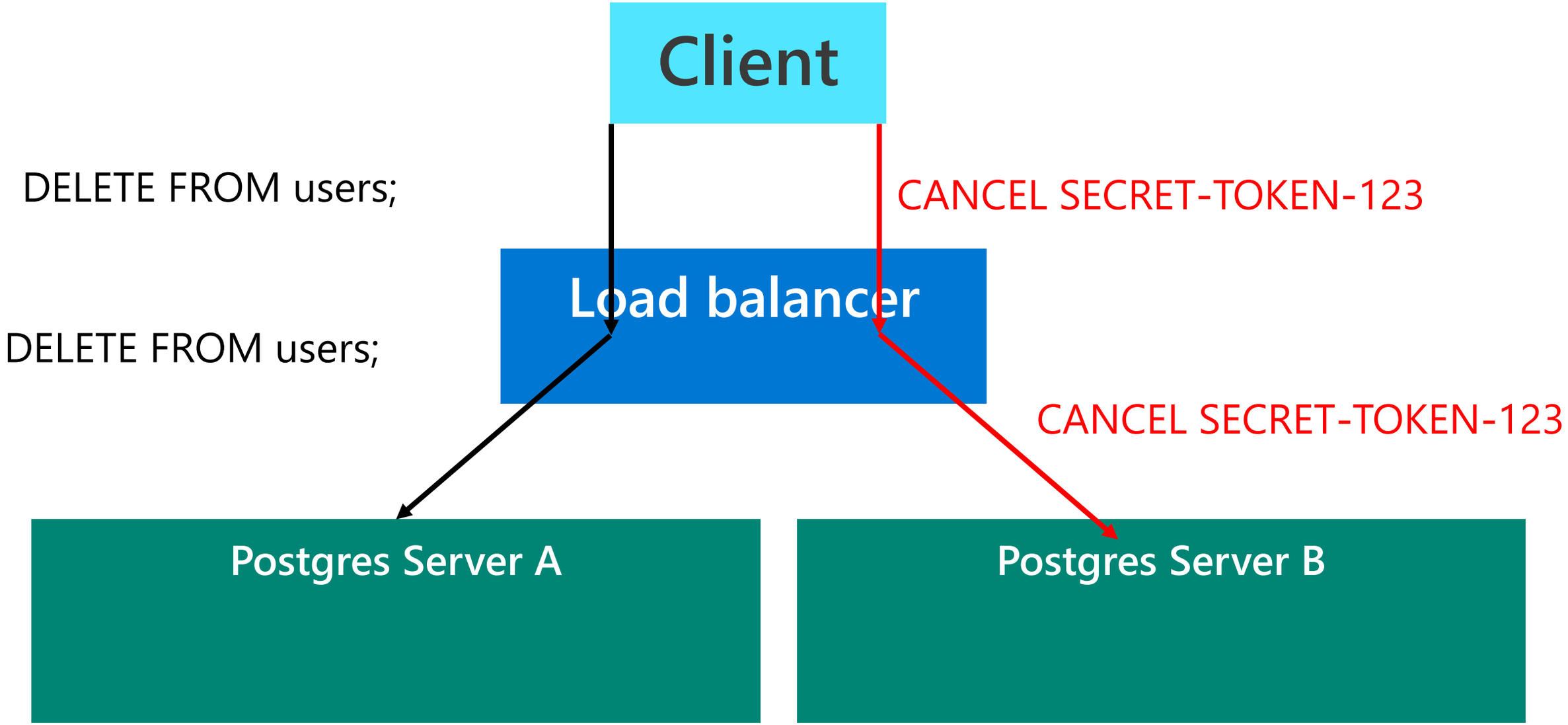
So what happens with cancellations and a load balancer



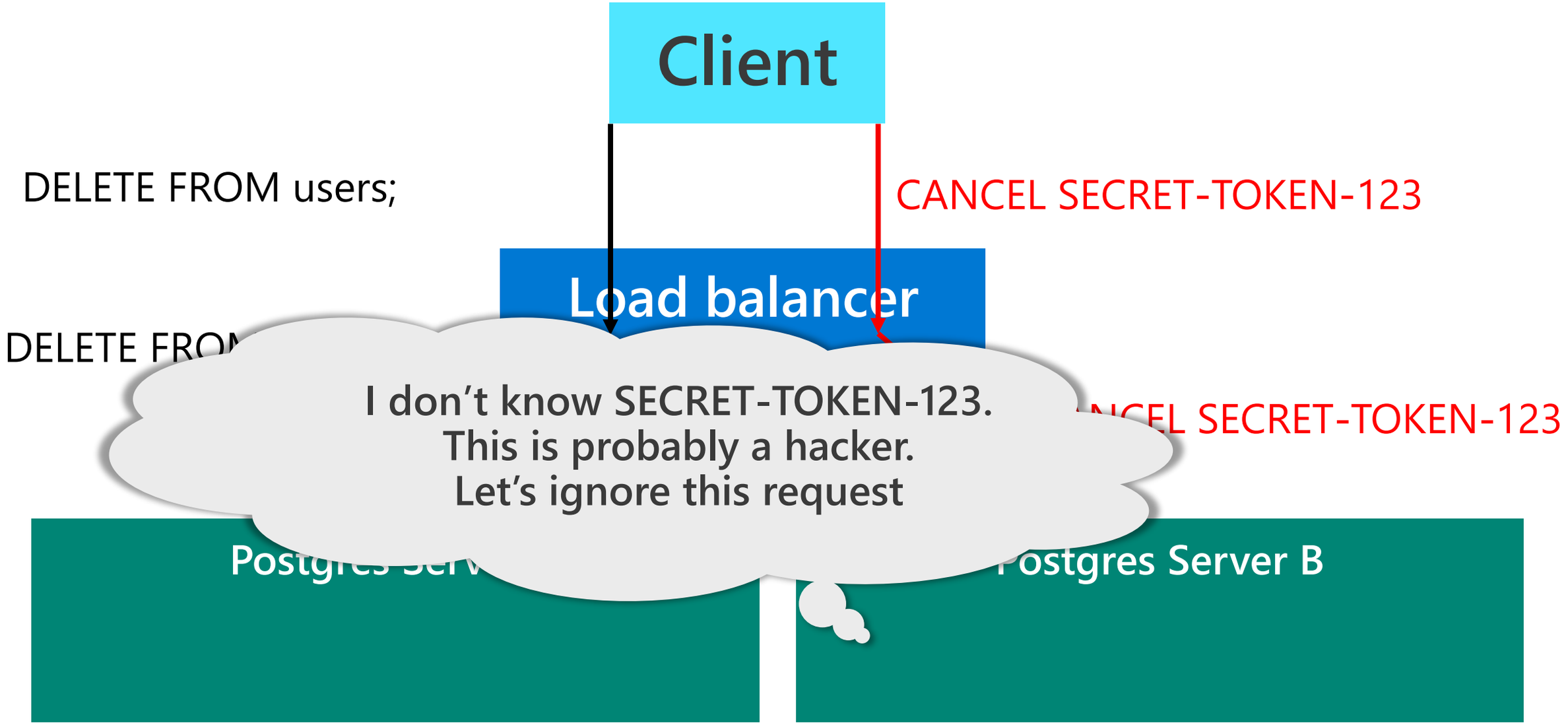
So what happens with cancellations and a load balancer



So what happens with cancellations and a load balancer



So what happens with cancellations and a load balancer



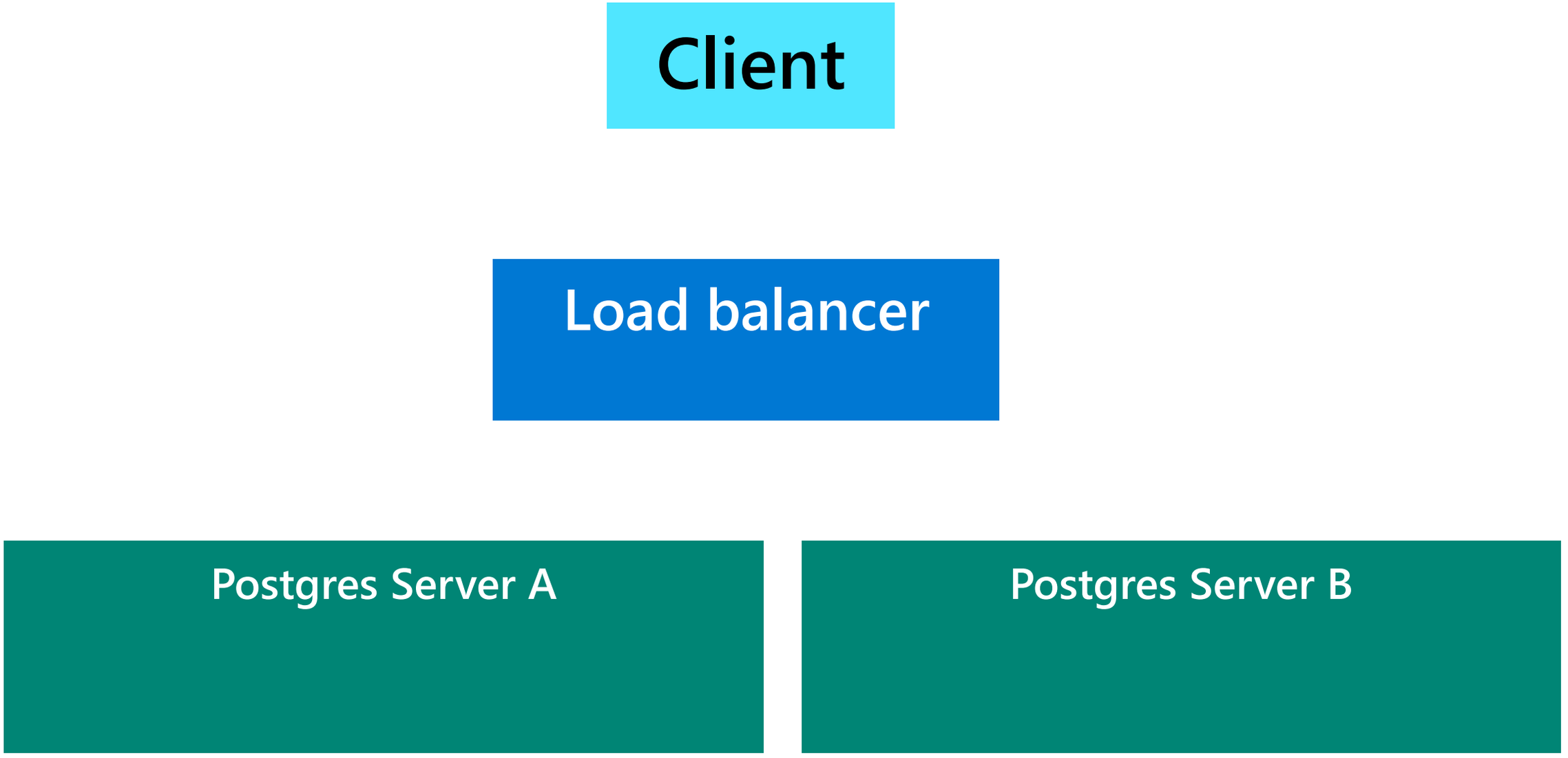
Not so fast (continued)

- Important assumption: Different TCP streams are independent
- Correct: Query on same Postgres session == same TCP stream
- Correct: New session == new TCP stream
- Wrong: Query cancellation request == new TCP stream

The cancellation problem

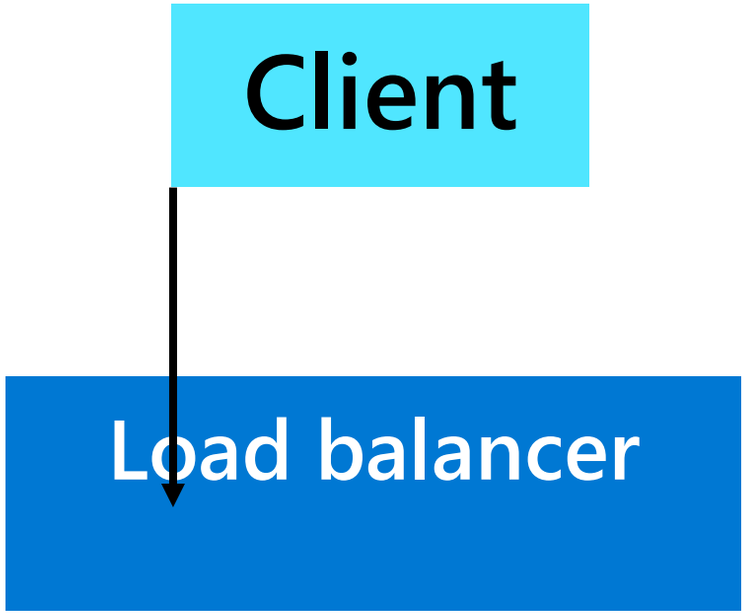
- Query cancellations end up at the wrong server most of the time
- Workaround: Trigger cancellations multiple time at the client side
- Solutions: None exist so far

Idea: Make Postgres servers know about each other



Idea: Make Postgres servers know about each other

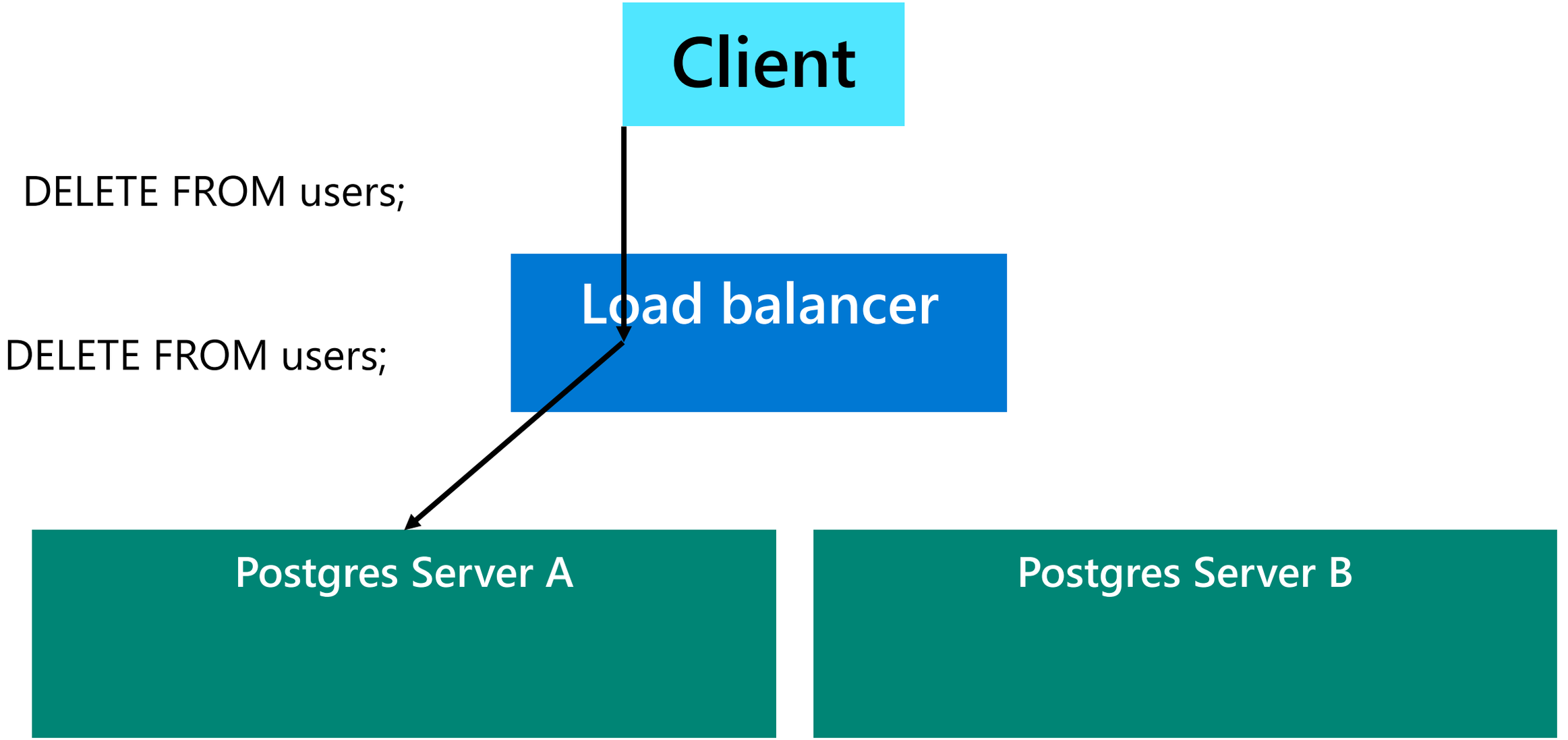
DELETE FROM users;



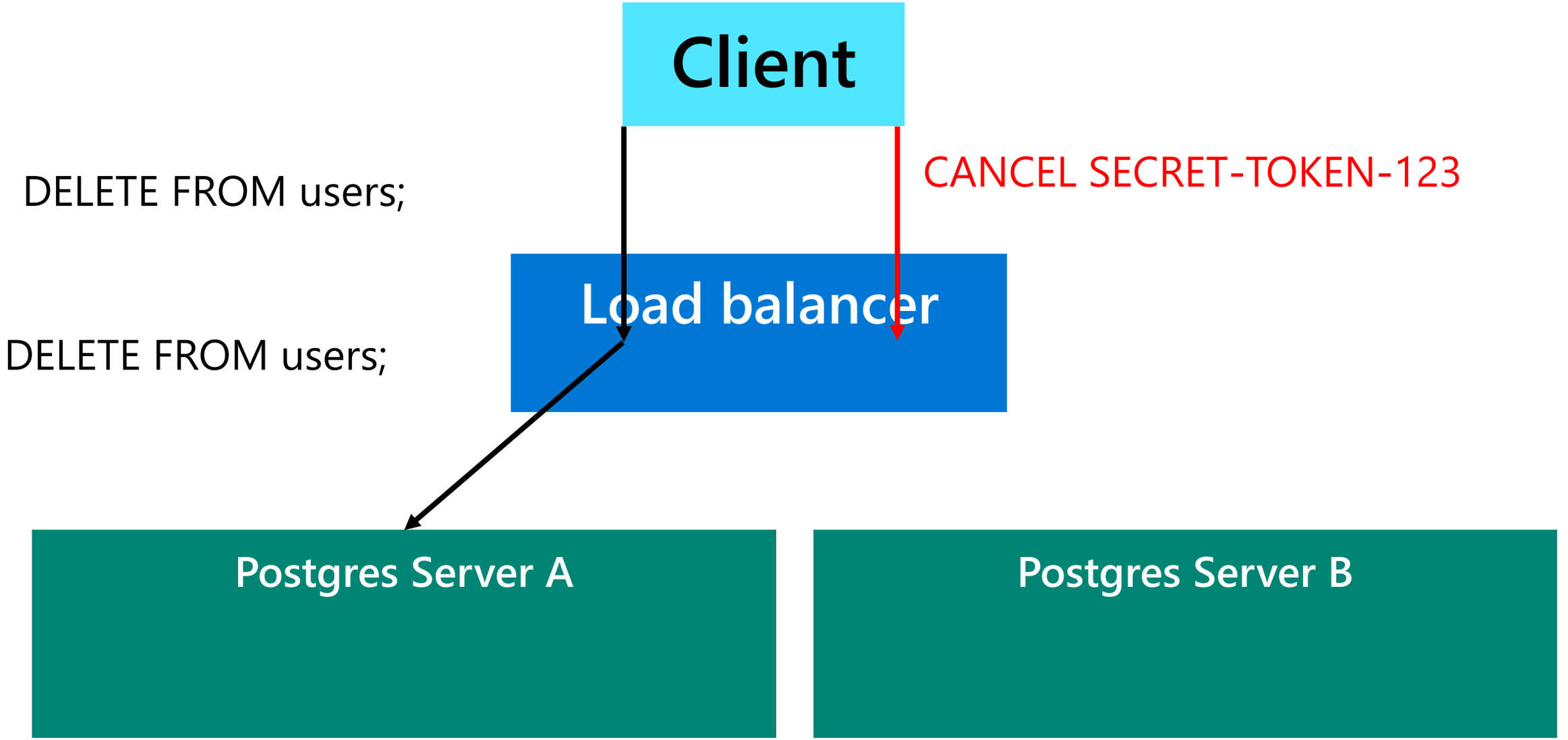
Postgres Server A

Postgres Server B

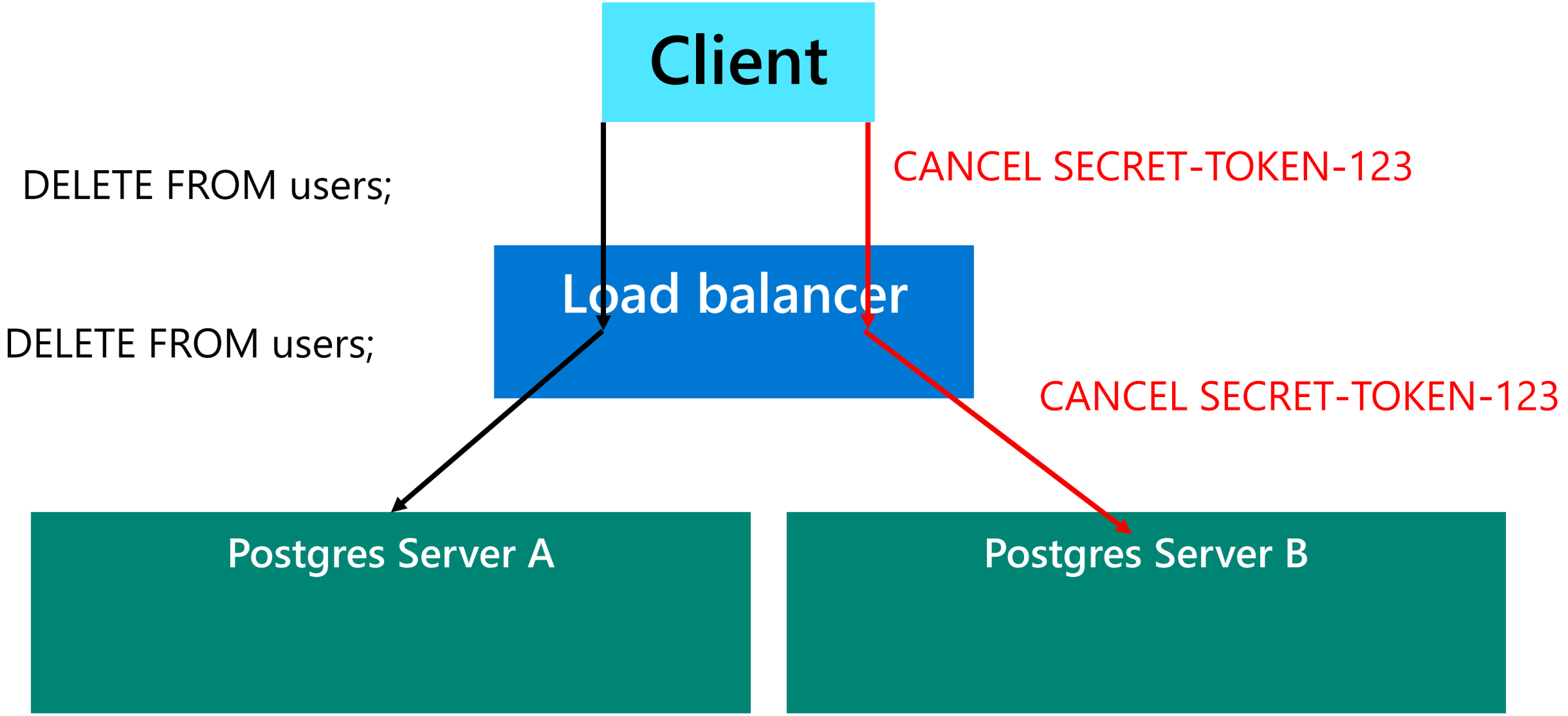
Idea: Make Postgres servers know about each other



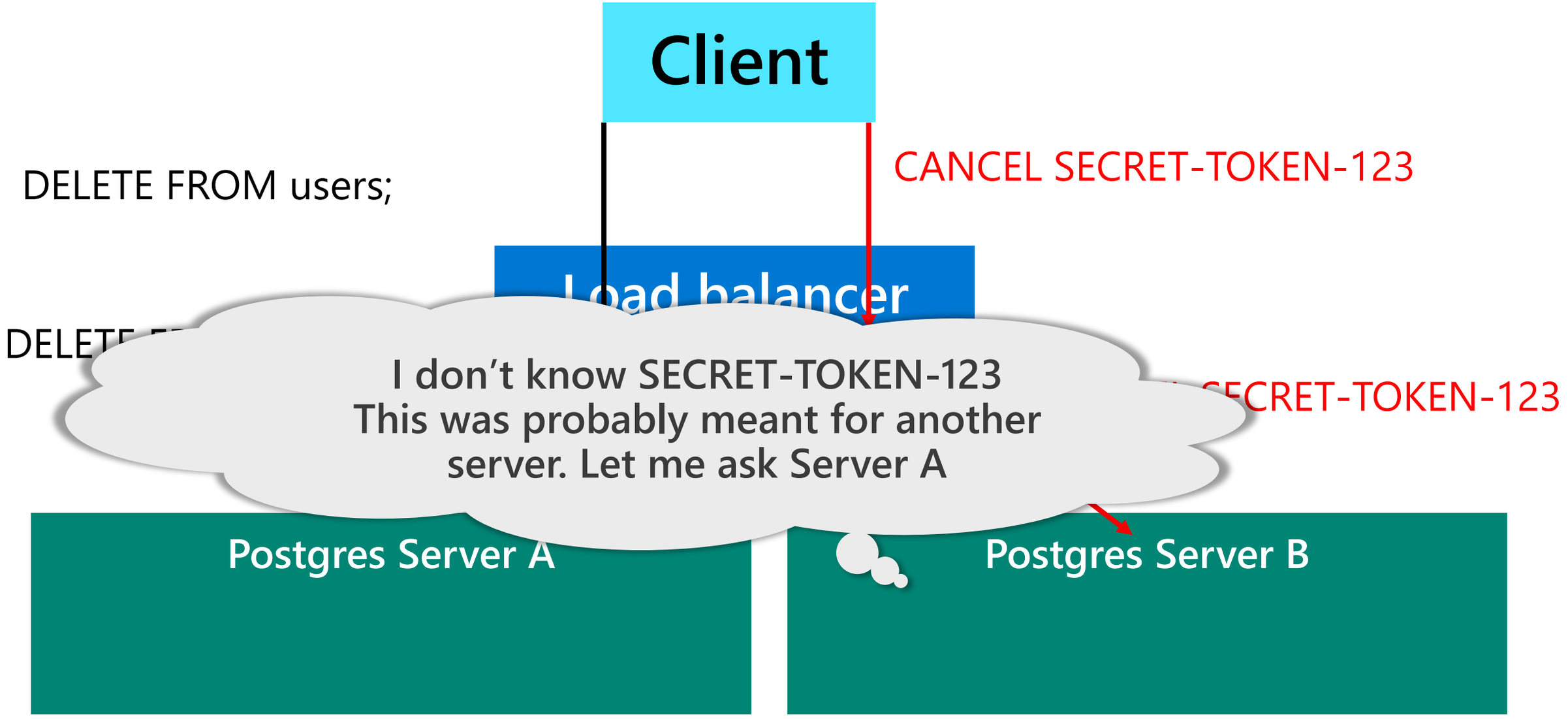
Idea: Make Postgres servers know about each other



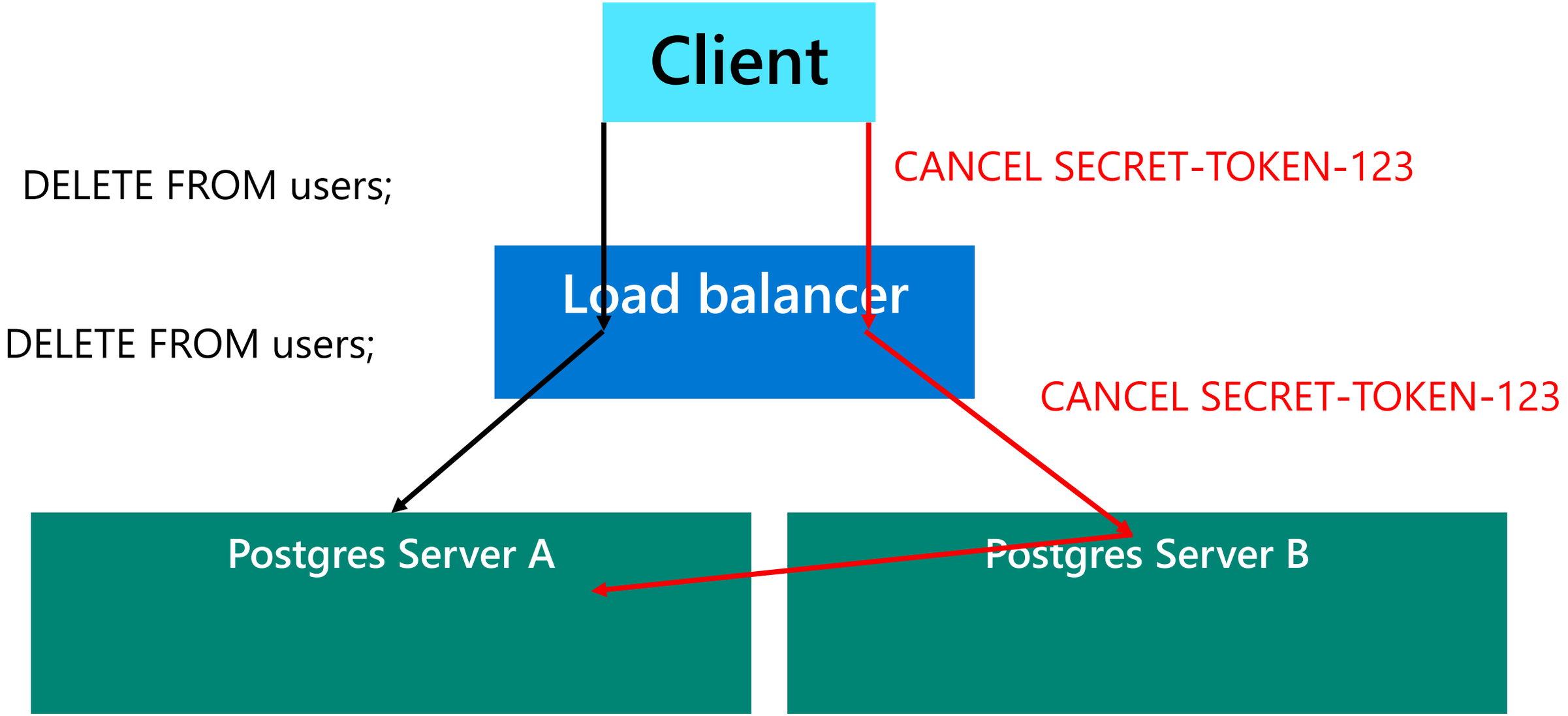
Idea: Make Postgres servers know about each other



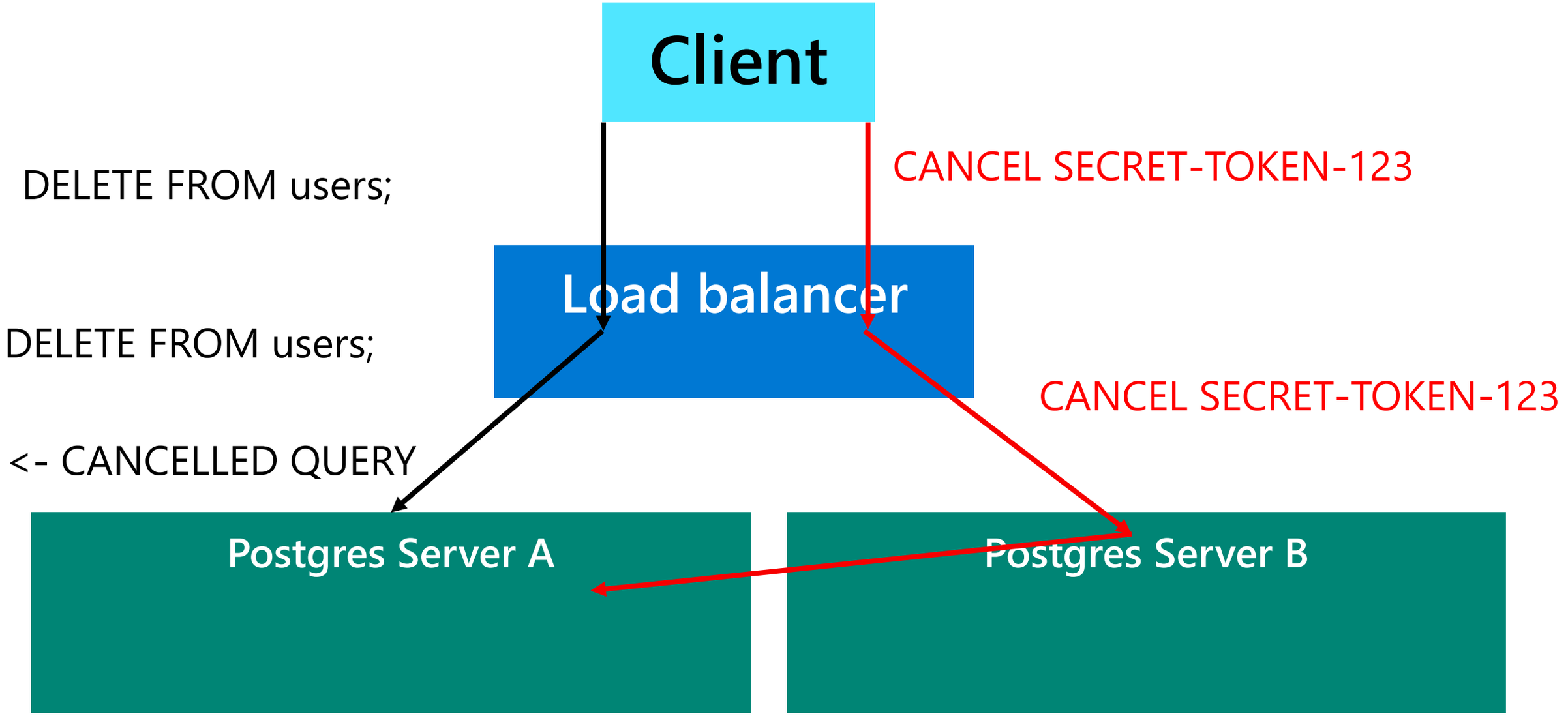
Idea: Make Postgres servers know about each other



Idea: Make Postgres servers know about each other



Idea: Make Postgres servers know about each other



How to make this idea work?

- Postgres does not support this out of the box
- What can we do to make it work?

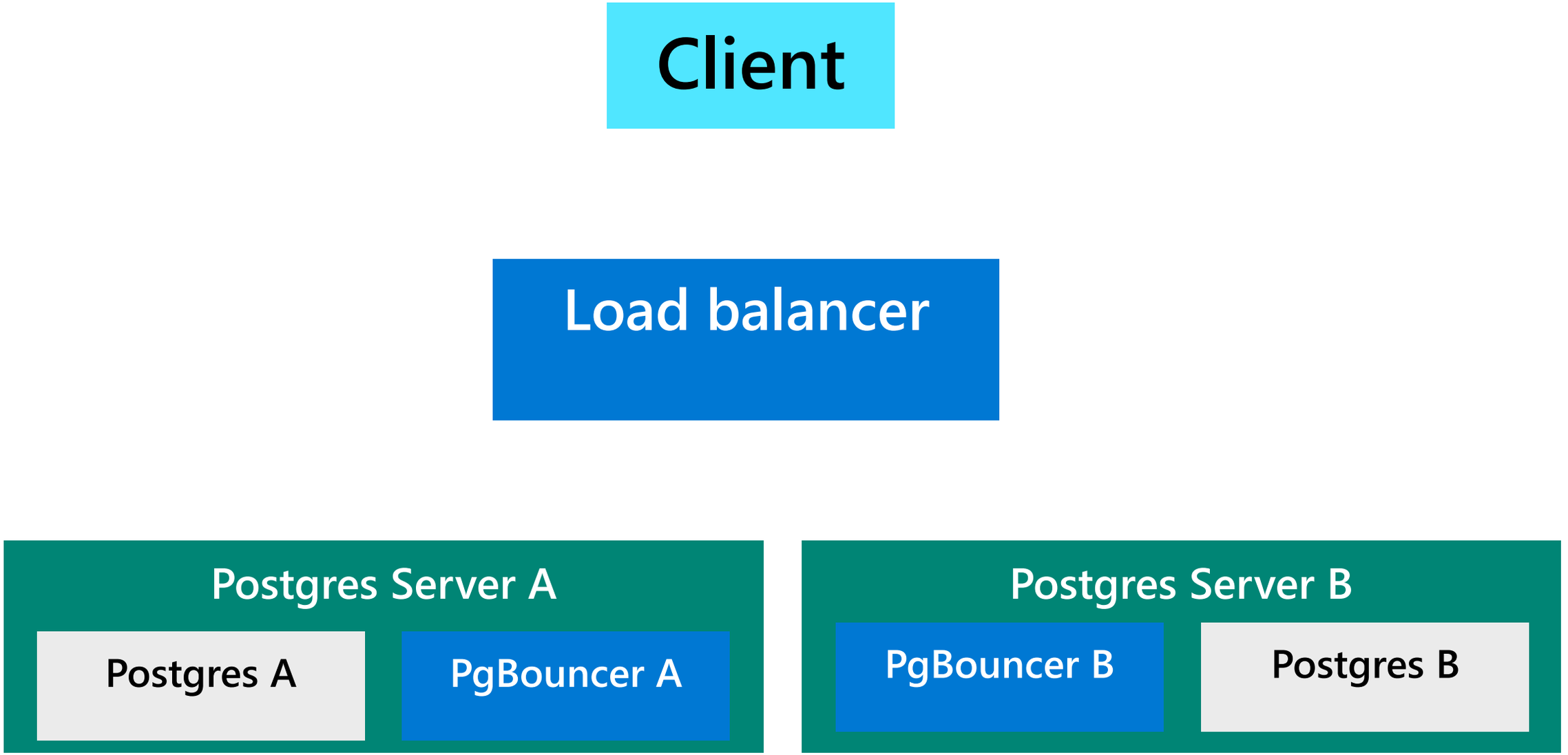
Option 1: Postgres extension

- No extension hooks exist for cancellation related code ☹
- Getting hooks (or built-in functionality) in Postgres will take a long time and won't work on old Postgres versions
- Need to build cancellation forwarding support

Option 2: Modify PgBouncer

- PgBouncer can run in front of any Postgres server version
- PgBouncer already has cancellation forwarding code to forward cancellations to Postgres
- This code only needs to be modified to send to other PgBouncer servers
- PR for this can be found here: https://github.com/pgbouncer/pgbouncer/pull/666

Option 2: Modify PgBouncer



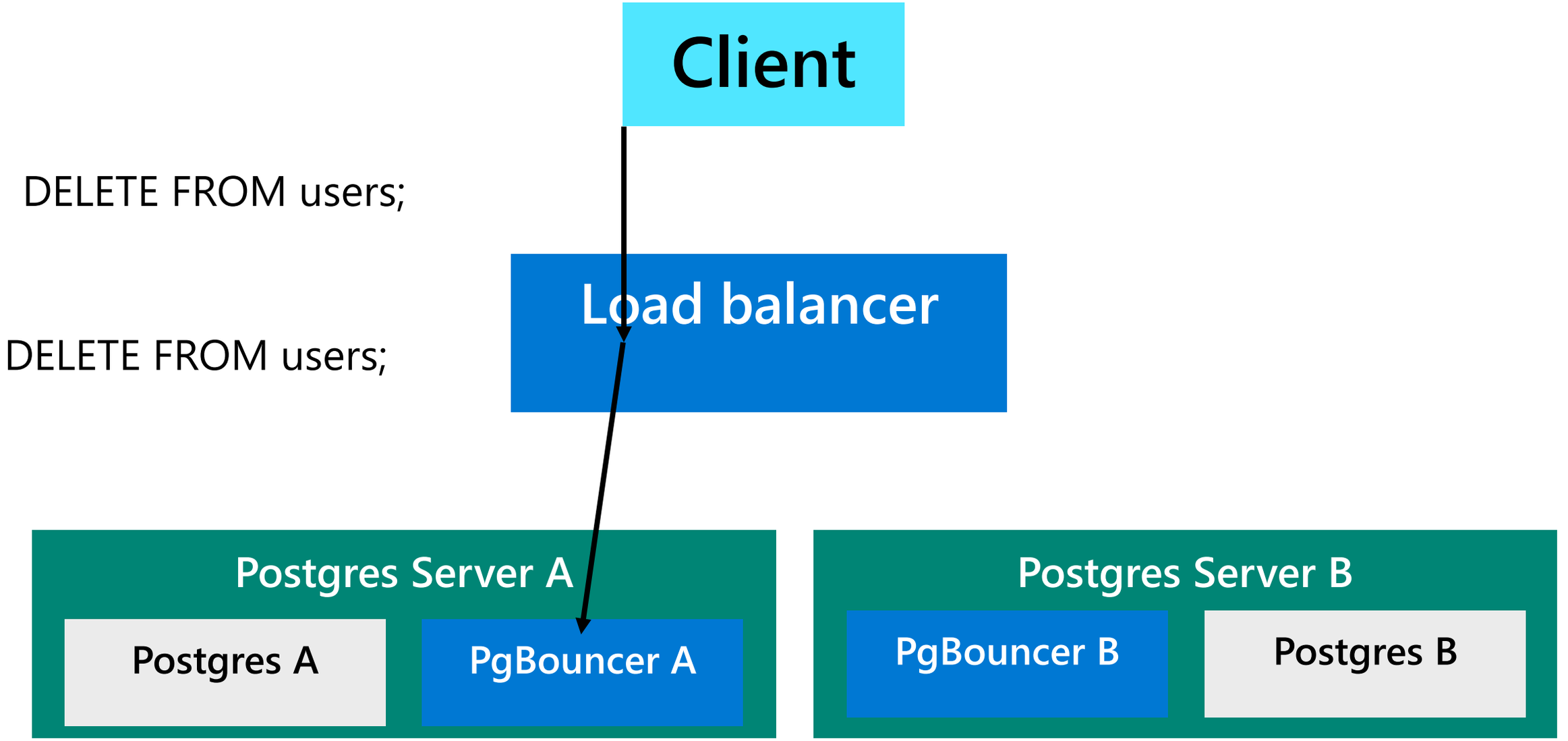
Option 2: Modify PgBouncer

DELETE FROM users;

Option 2: Modify PgBouncer

DELETE FROM users;

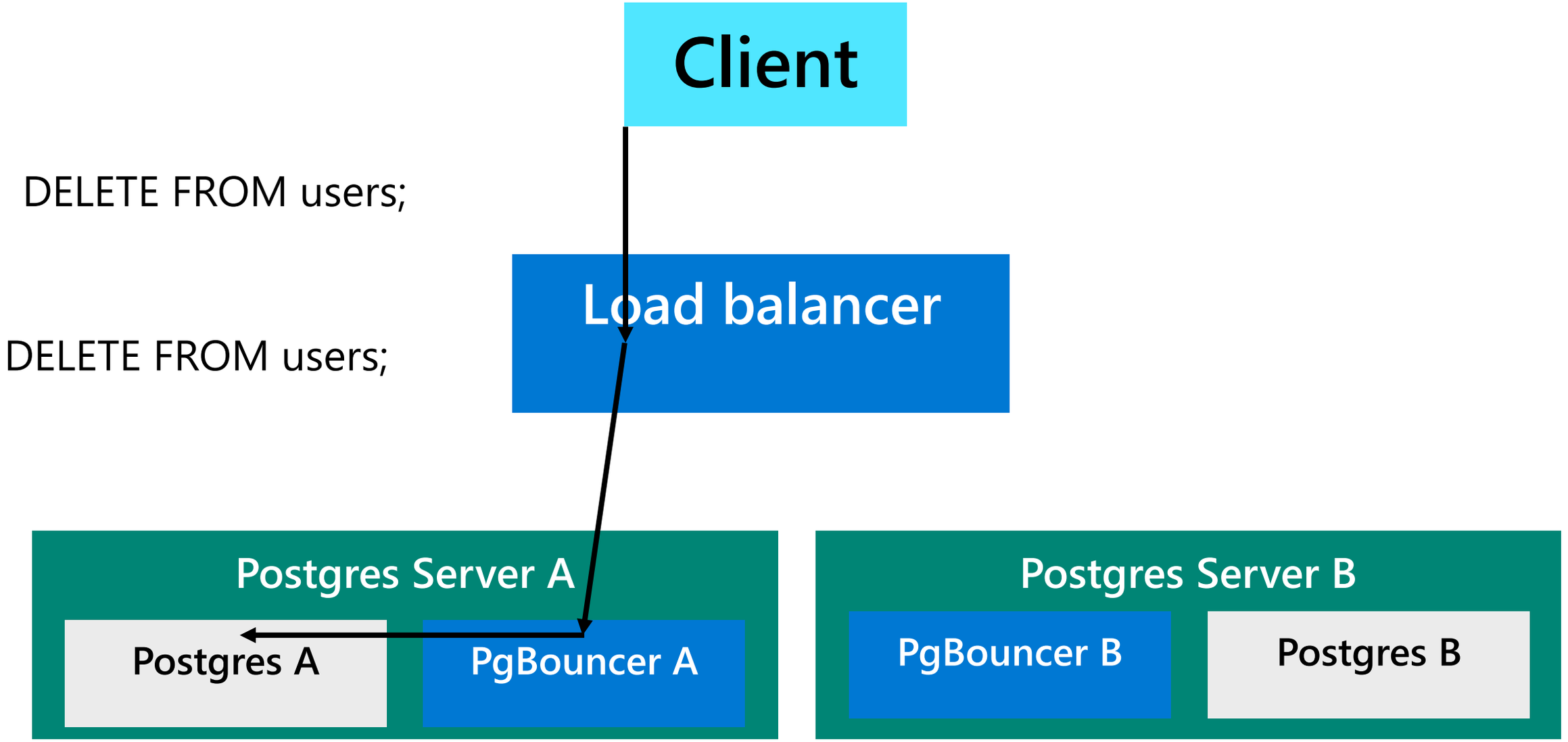
DELETE FROM users;



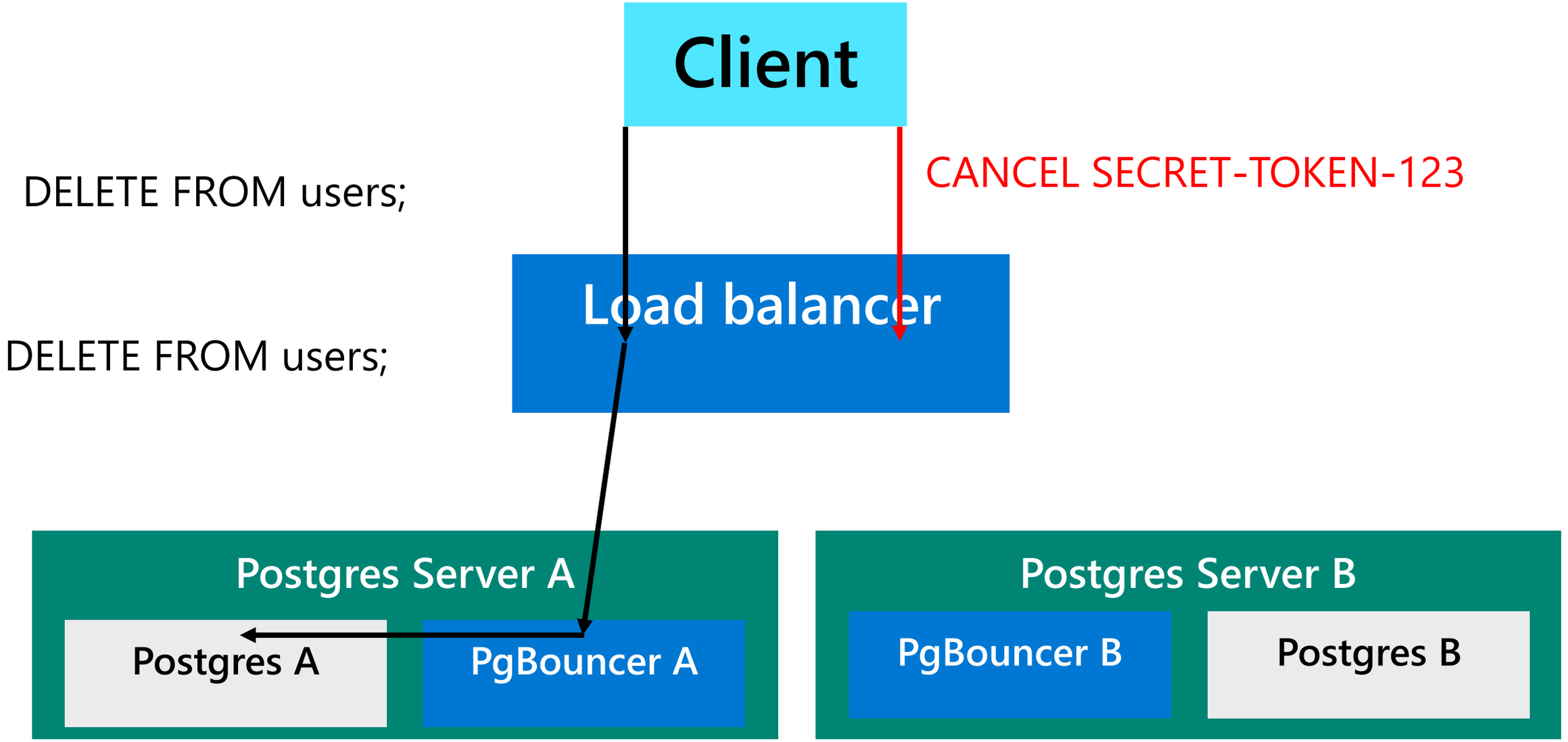
Option 2: Modify PgBouncer

DELETE FROM users;

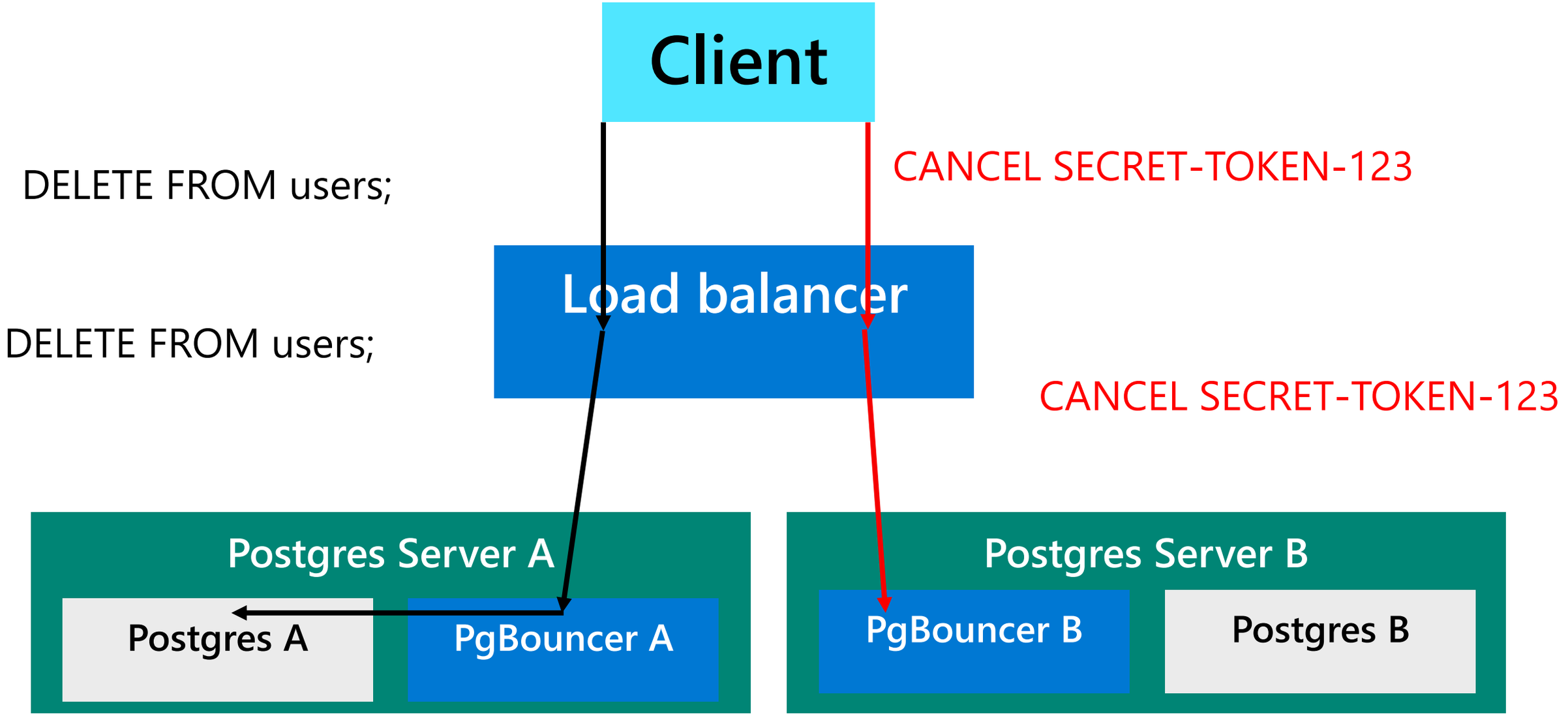
DELETE FROM users;



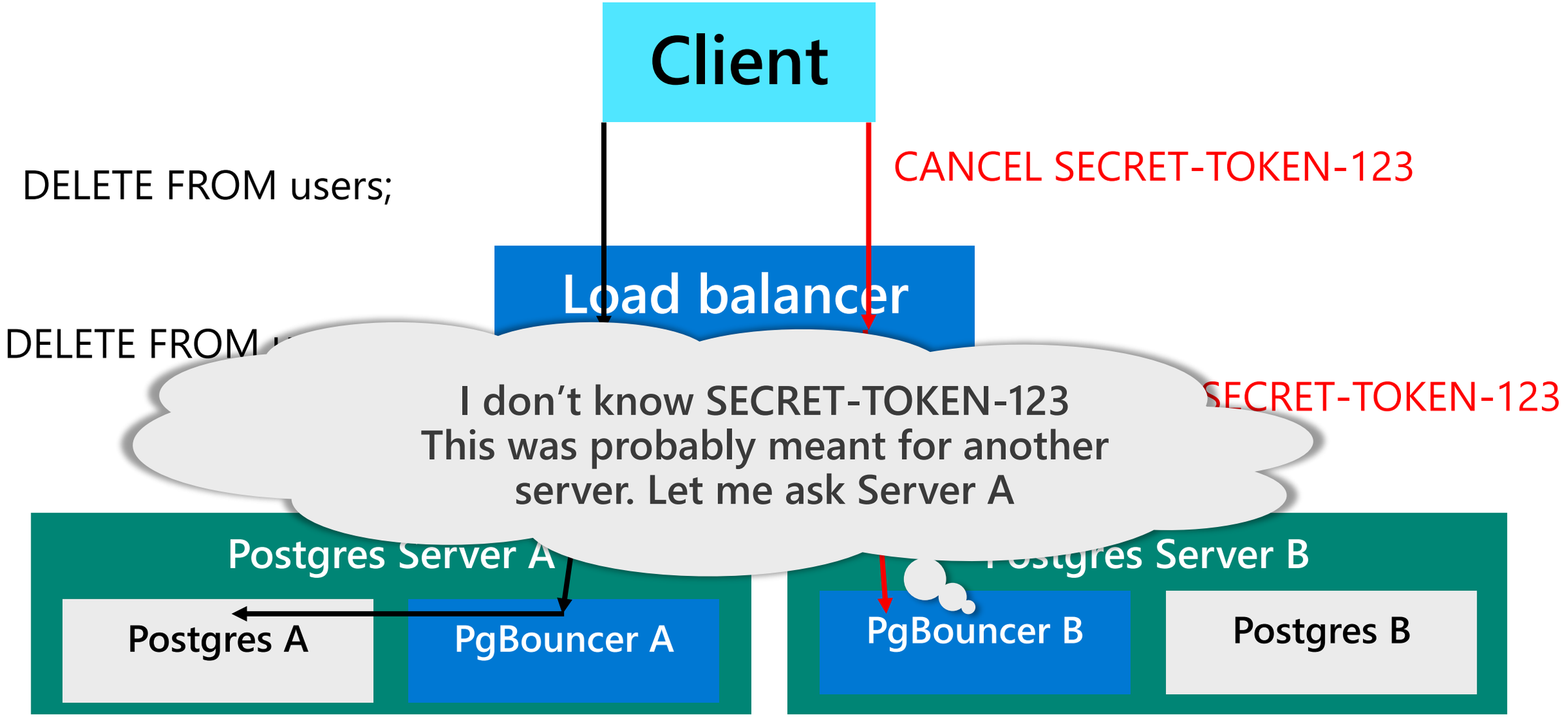
Option 2: Modify PgBouncer



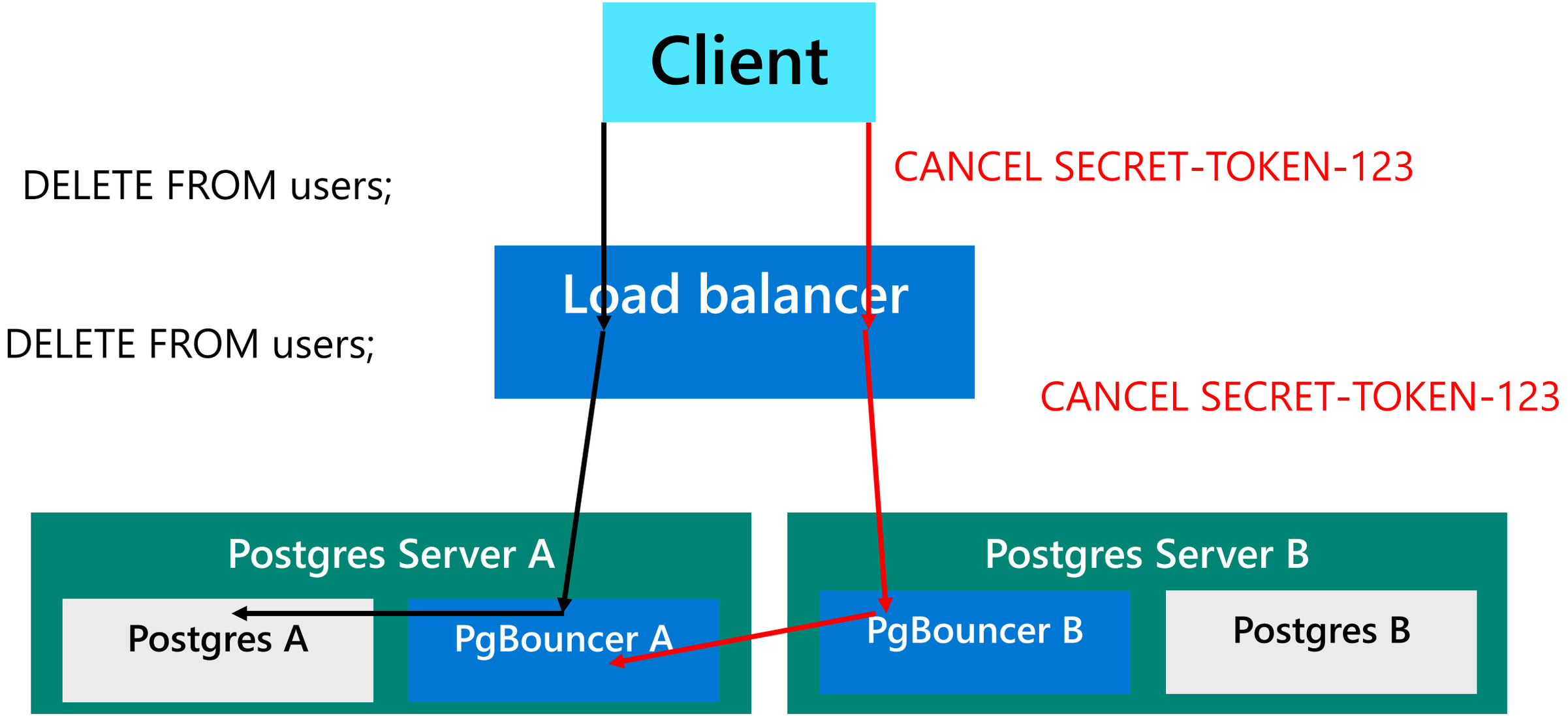
Option 2: Modify PgBouncer



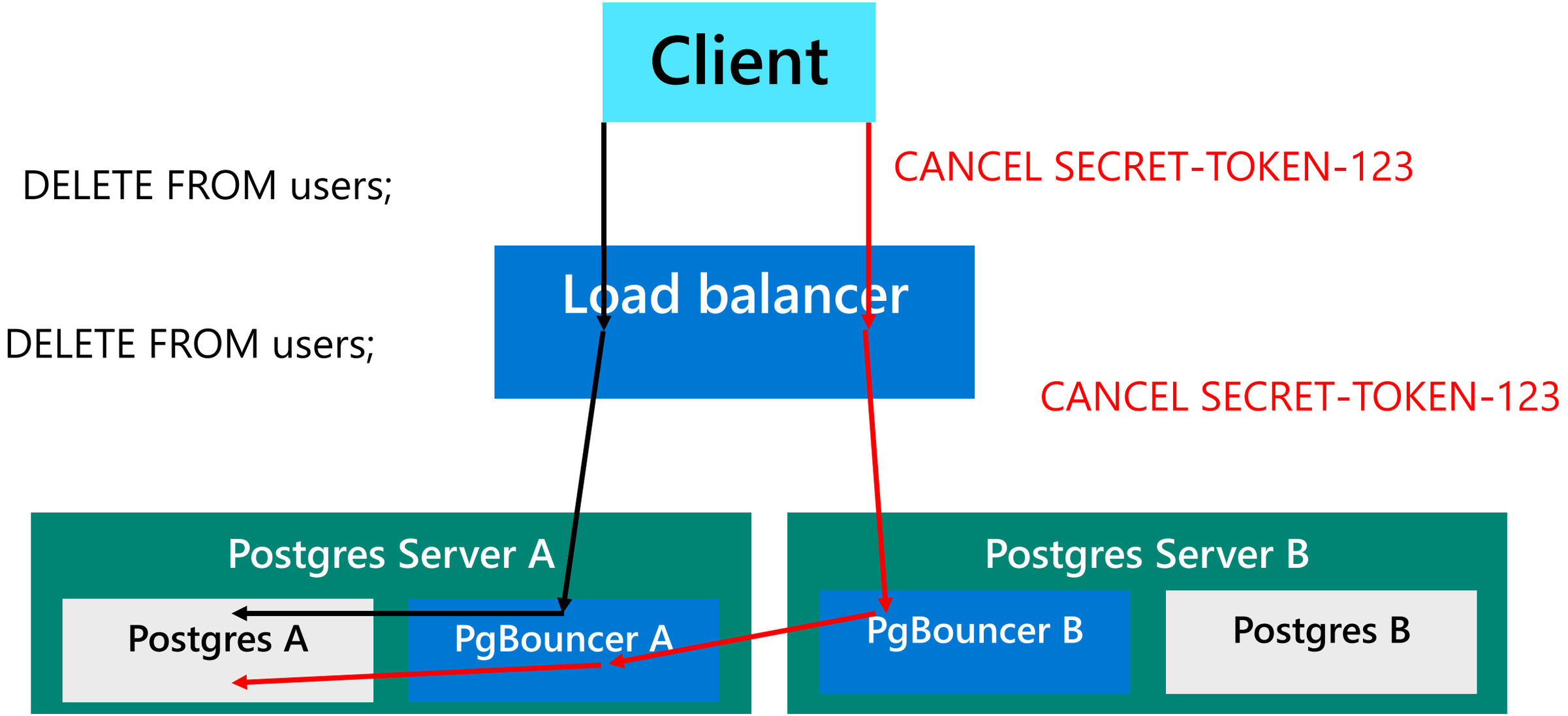
Option 2: Modify PgBouncer



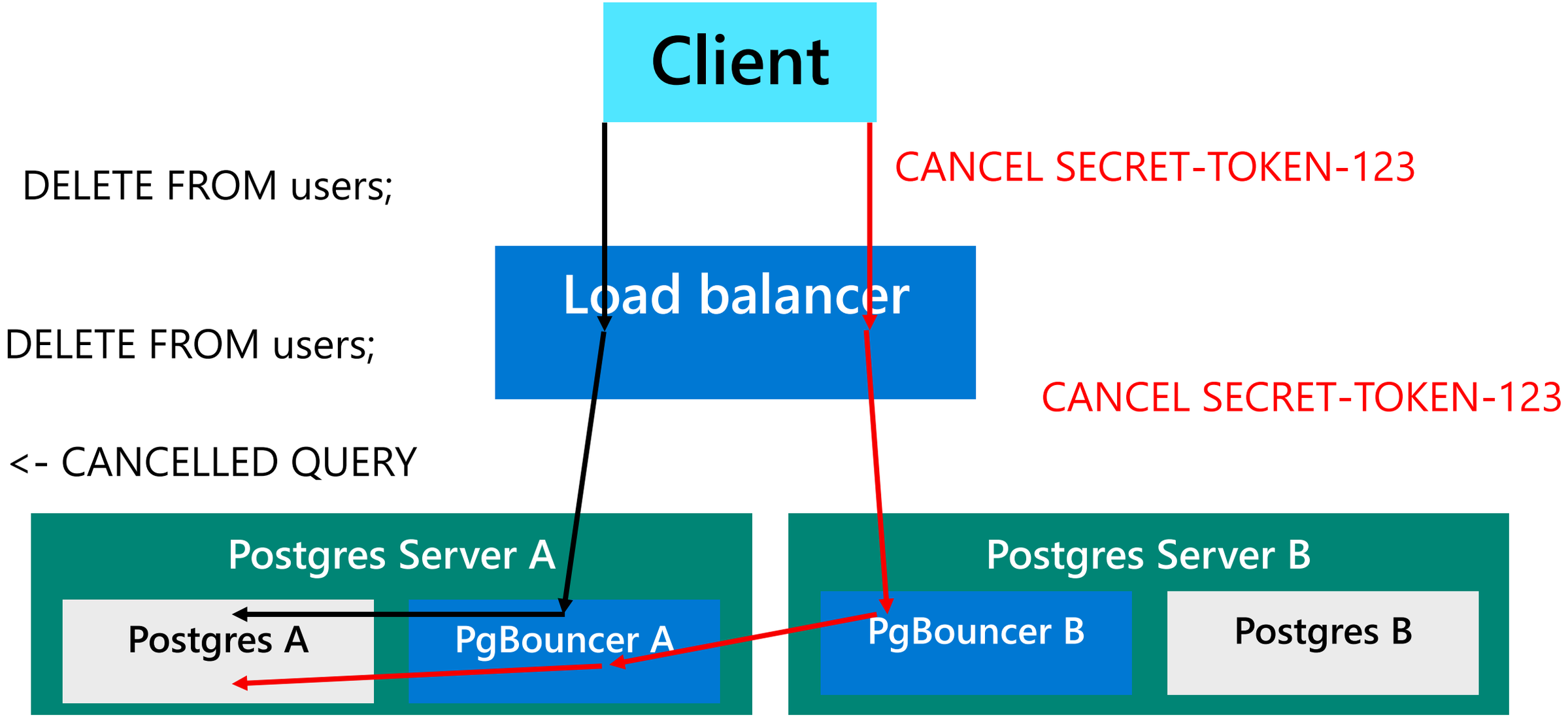
Option 2: Modify PgBouncer



Option 2: Modify PgBouncer



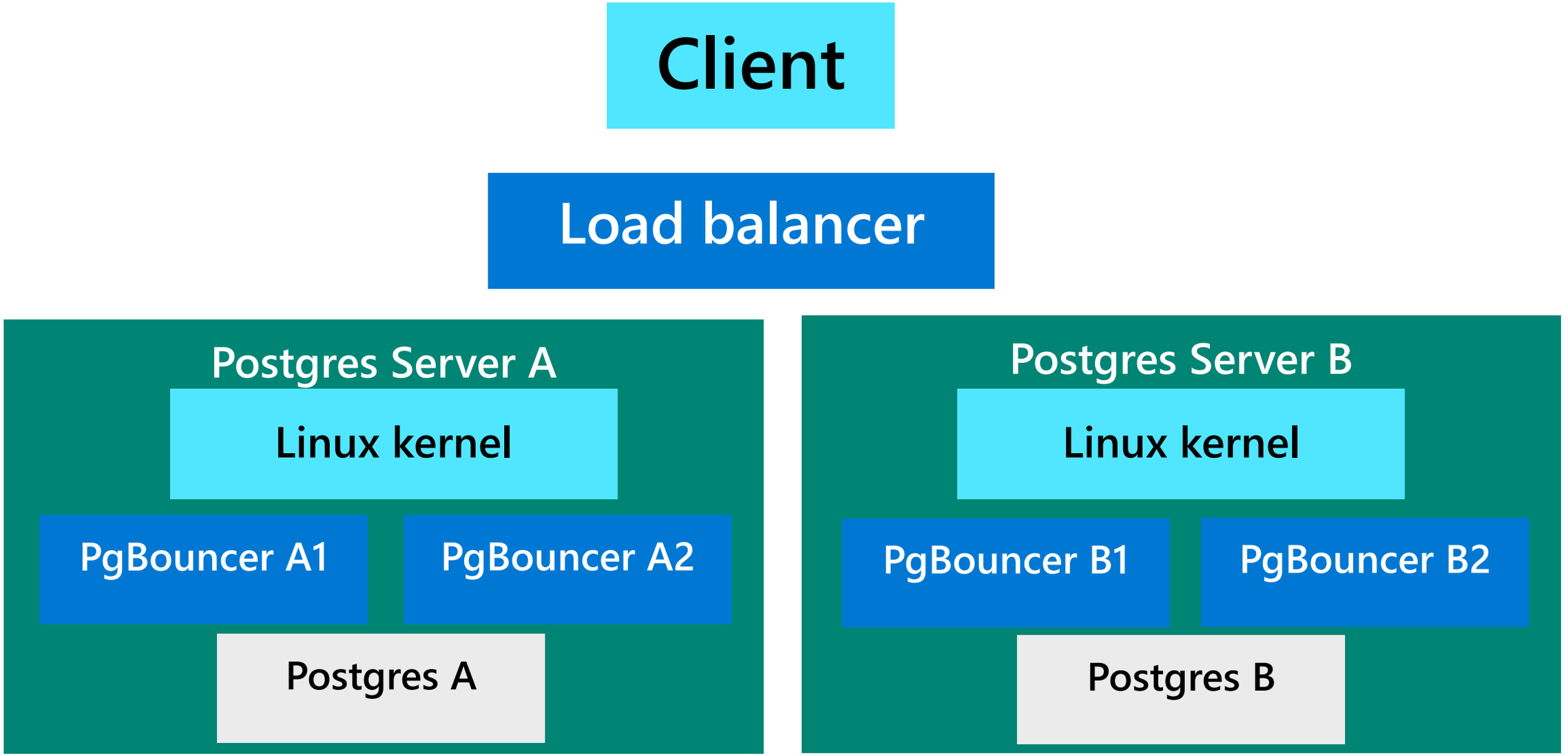
Option 2: Modify PgBouncer



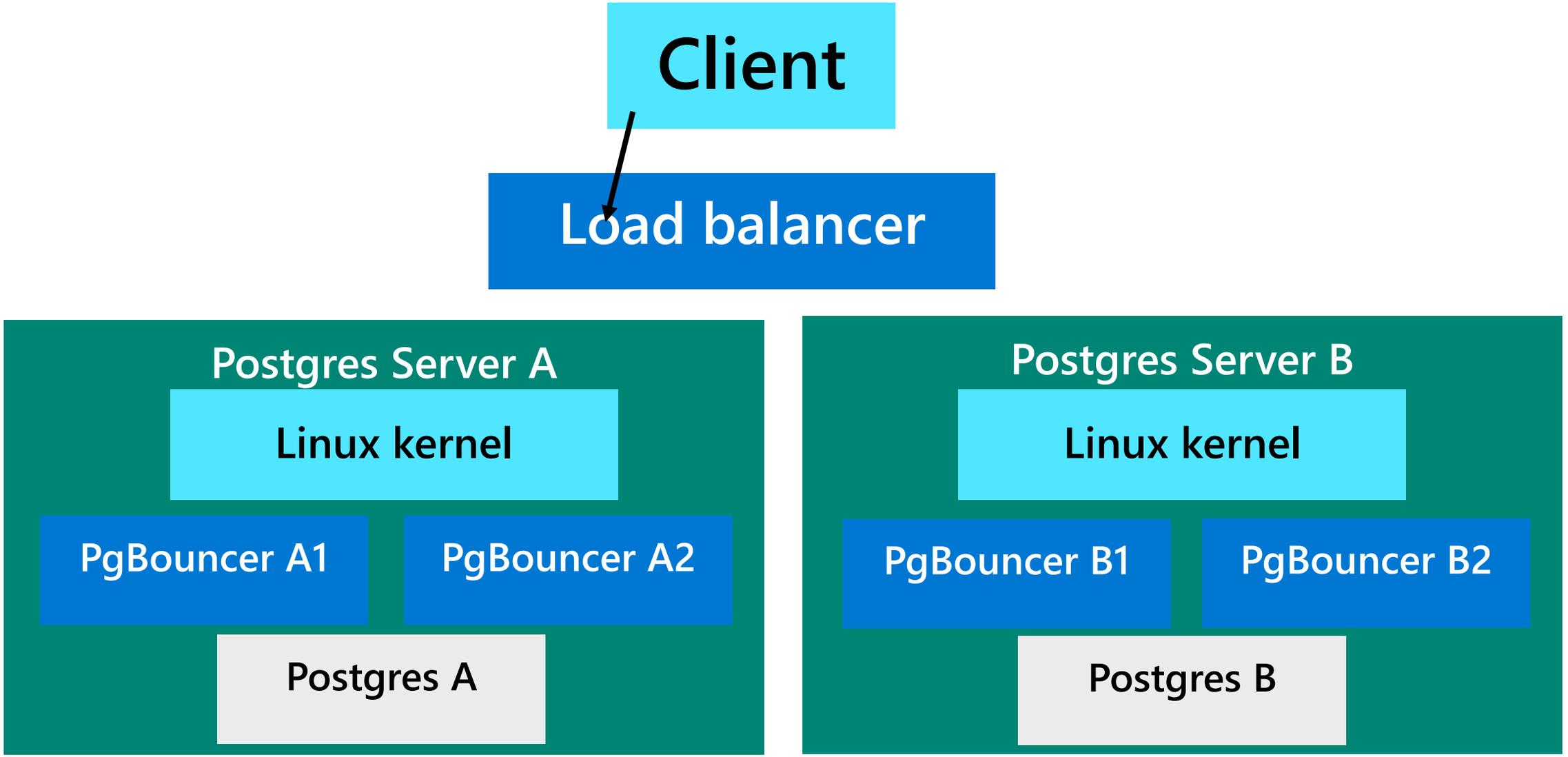
Making PgBouncer performant

- Single-threaded design becomes a bottleneck on high throughput servers
- PgBouncer supports multi-process by using so_reuseport=1
- The Linux kernel then load-balances TCP streams across all processes
- So, what we end up with is a multi-layered load balancer

What we end up with

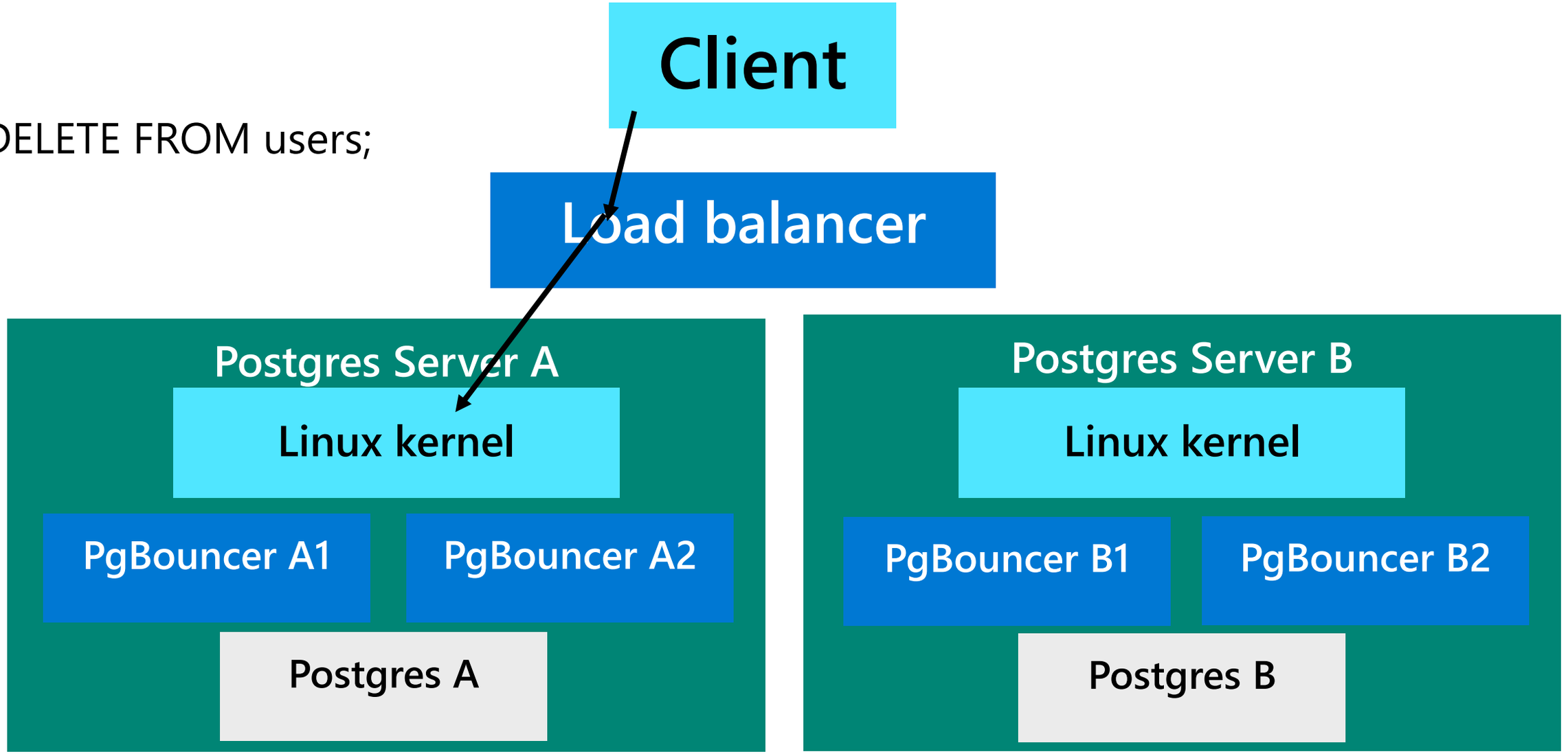


What we end up with



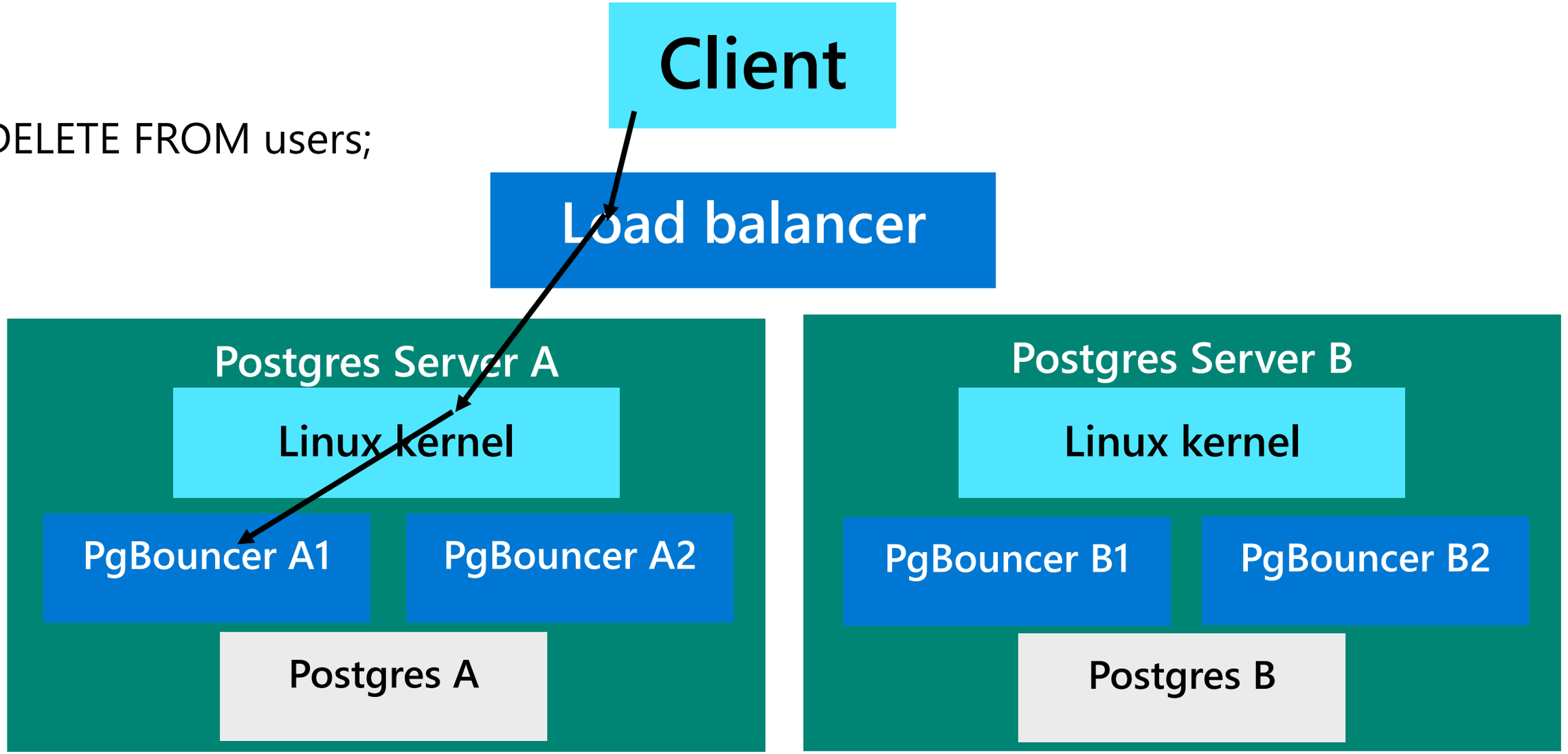
What we end up with

DELETE FROM users;



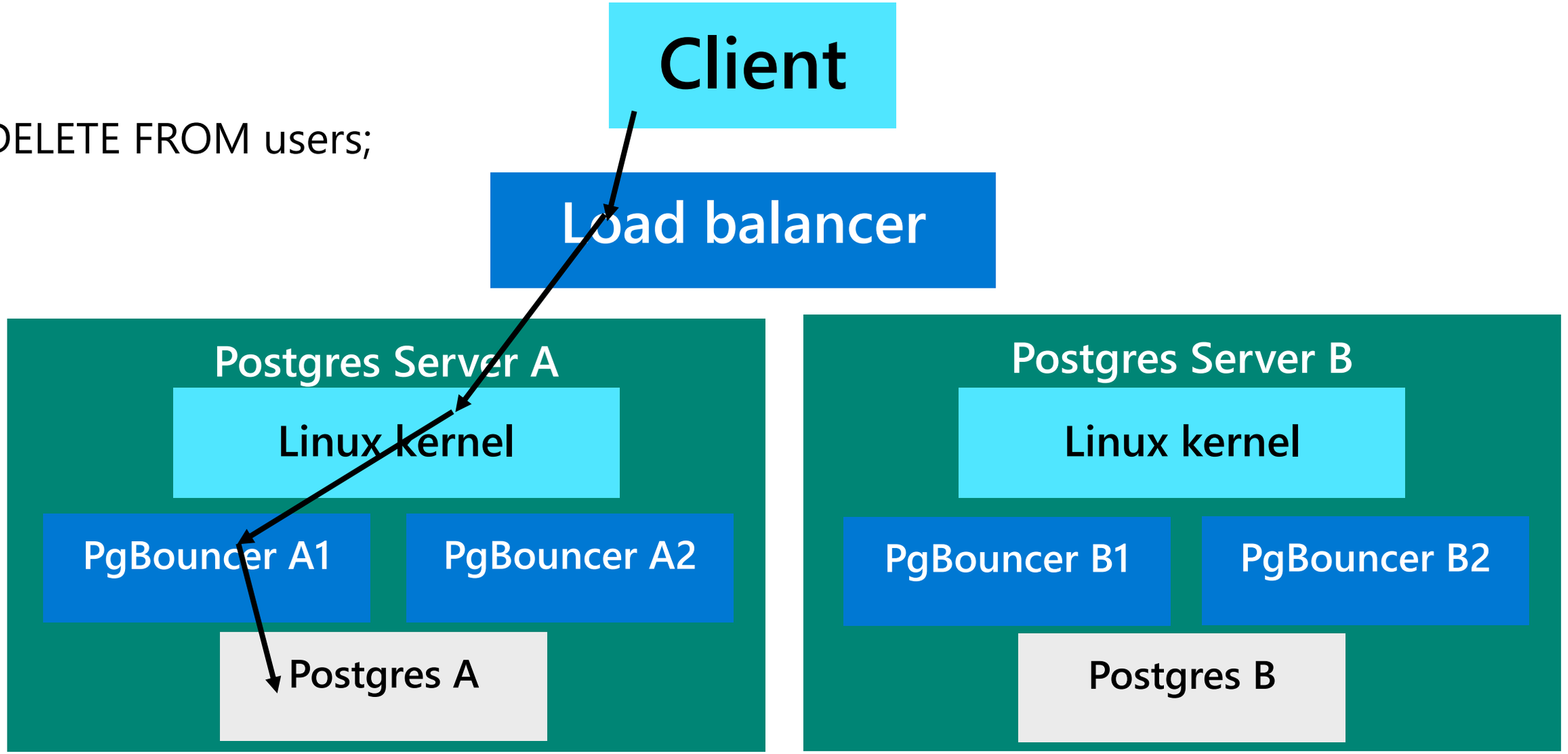
What we end up with

DELETE FROM users;



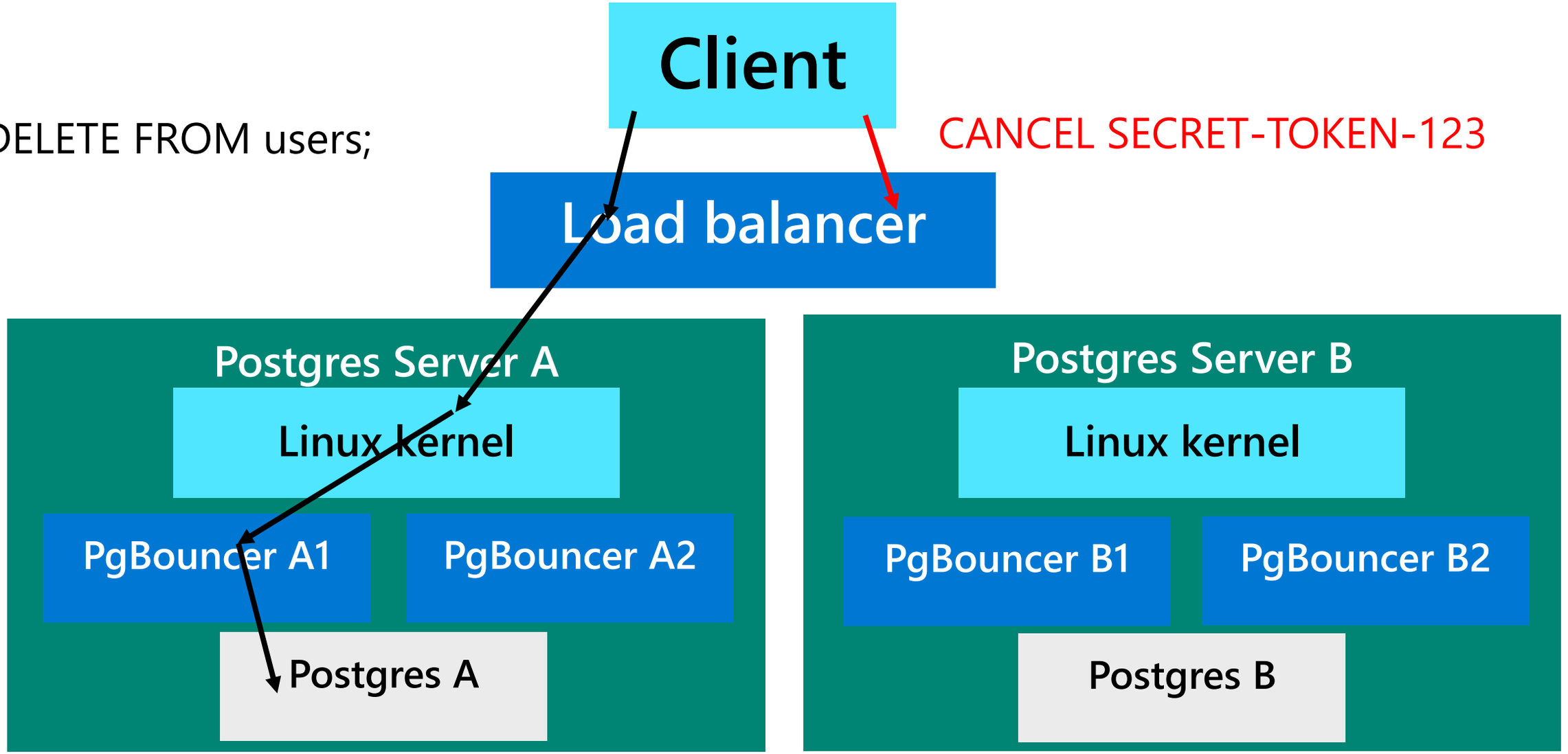
What we end up with

DELETE FROM users;



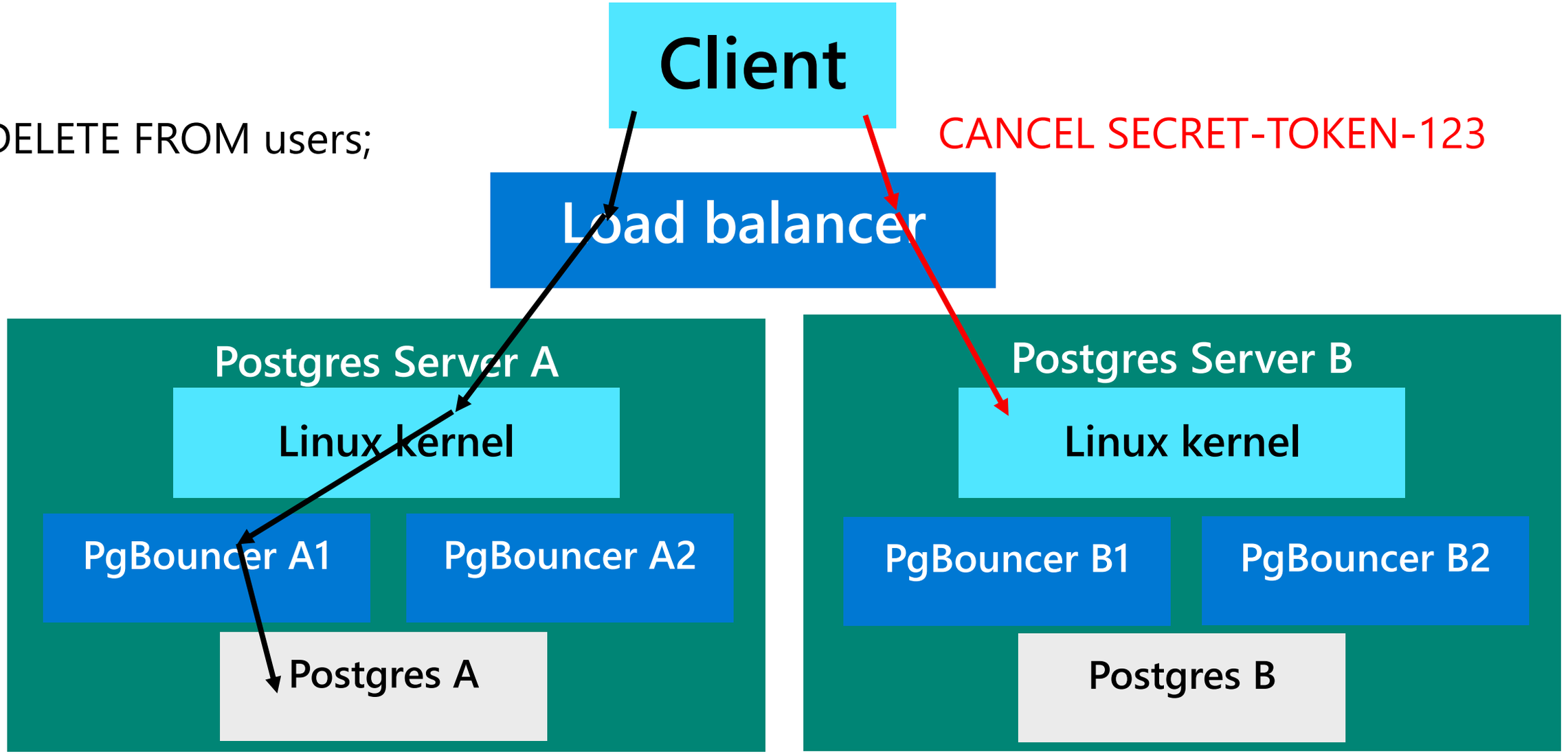
What we end up with

DELETE FROM users;



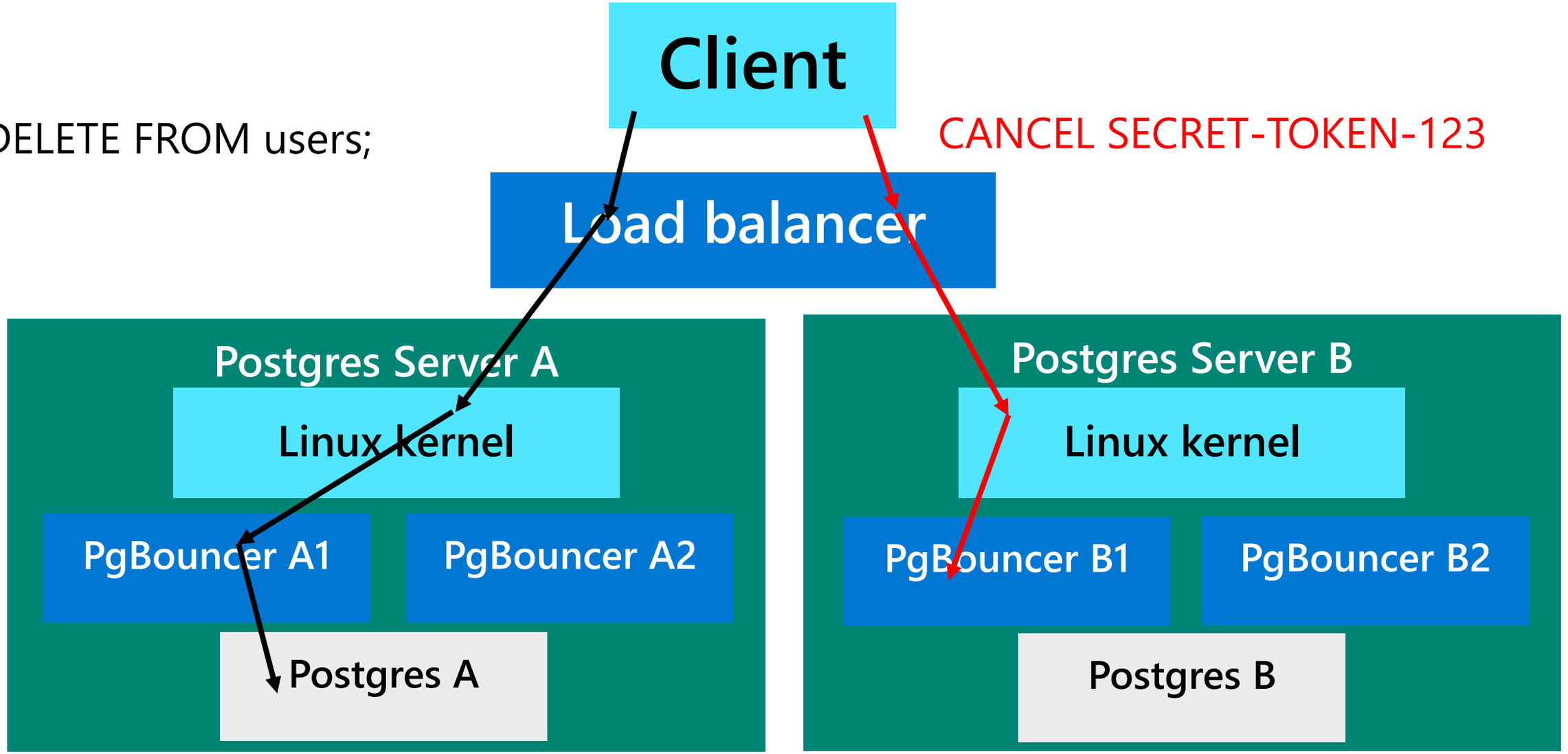
What we end up with

DELETE FROM users;



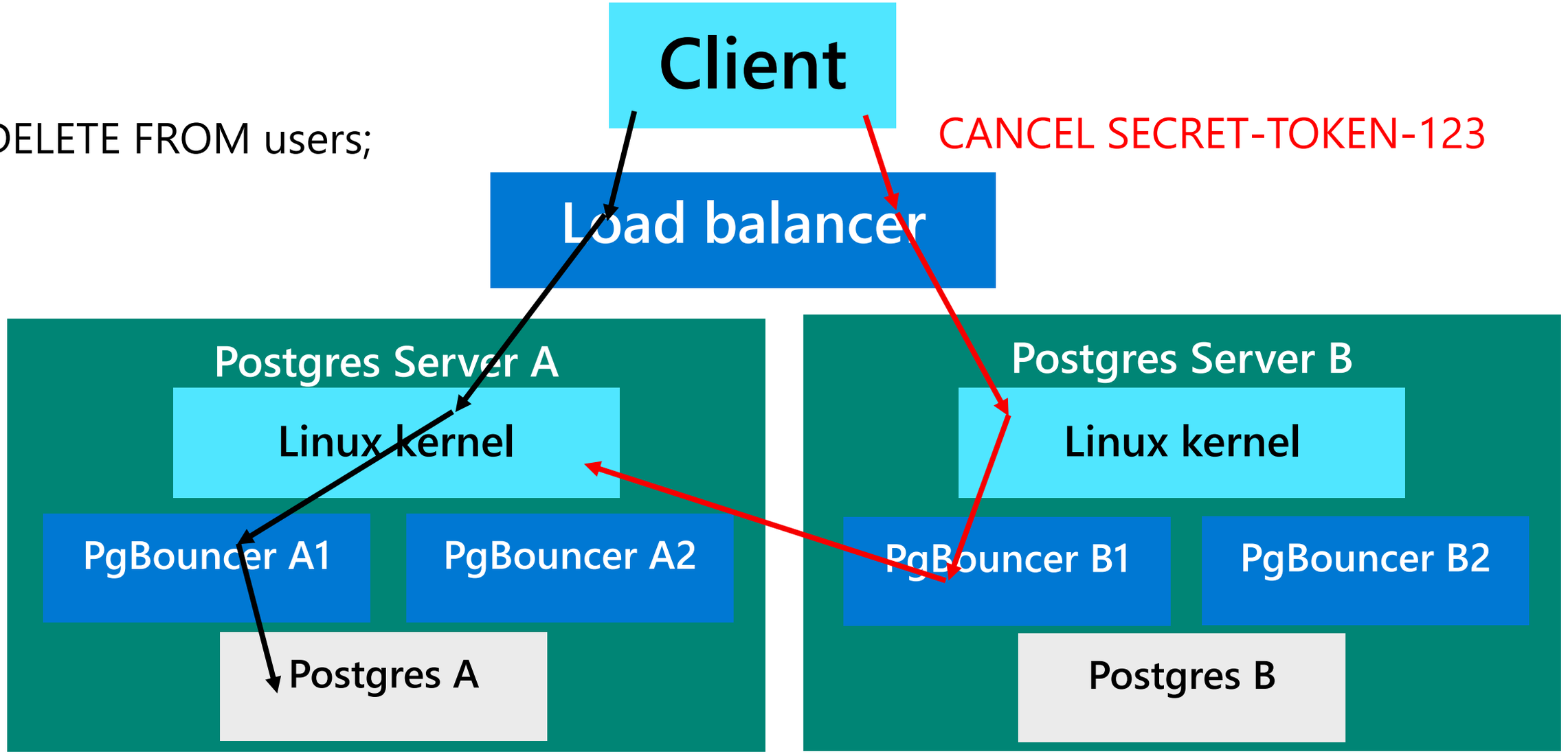
What we end up with

DELETE FROM users;



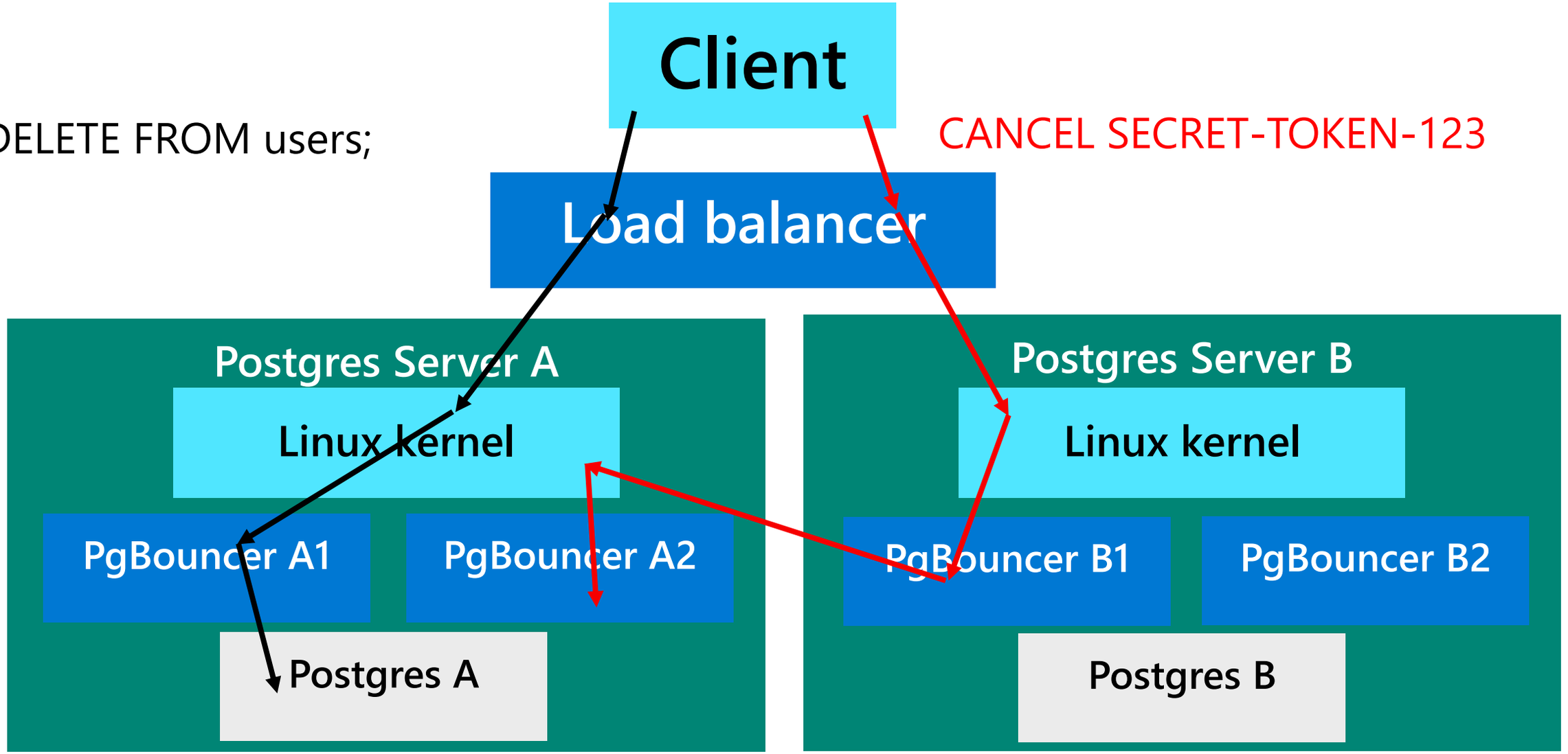
What we end up with

DELETE FROM users;



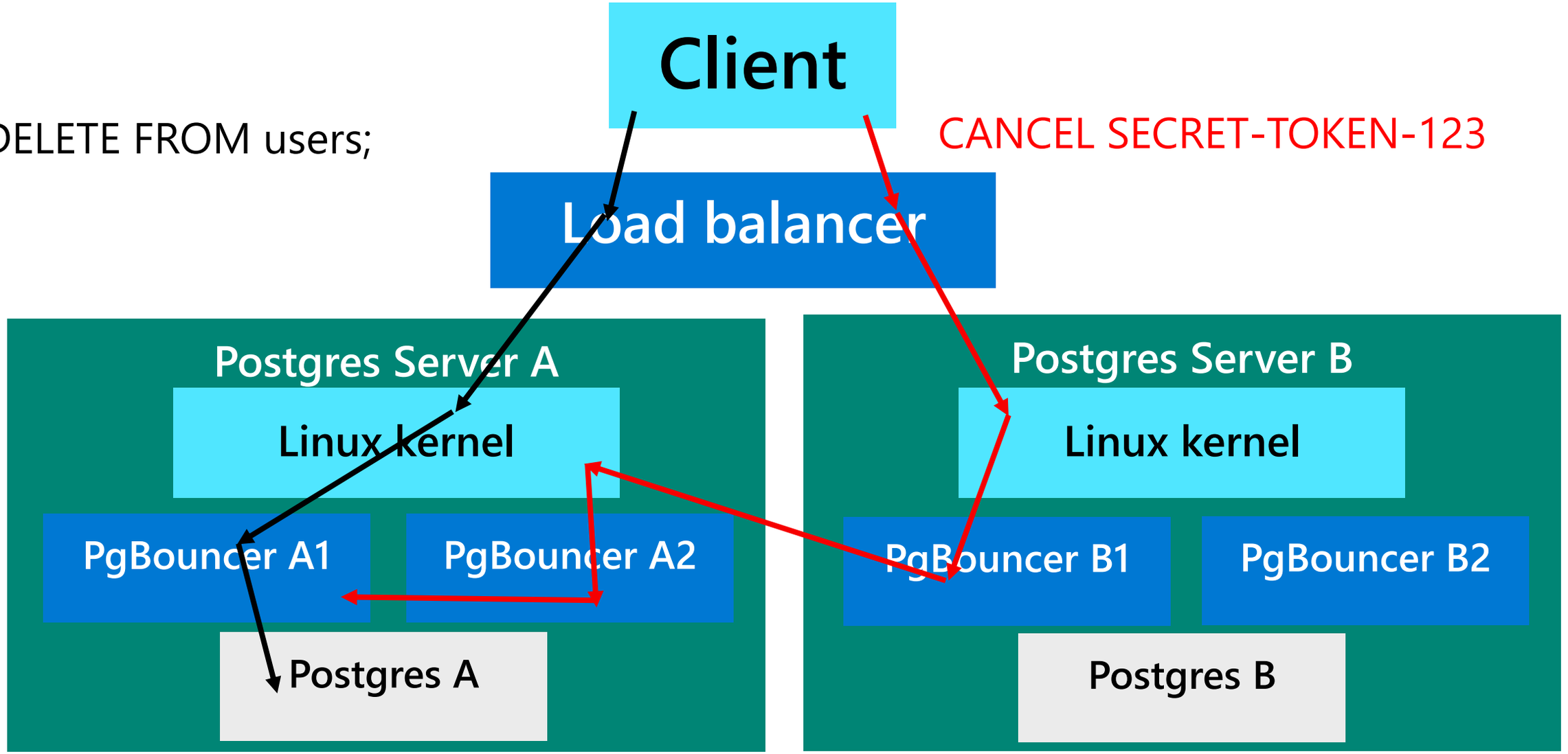
What we end up with

DELETE FROM users;



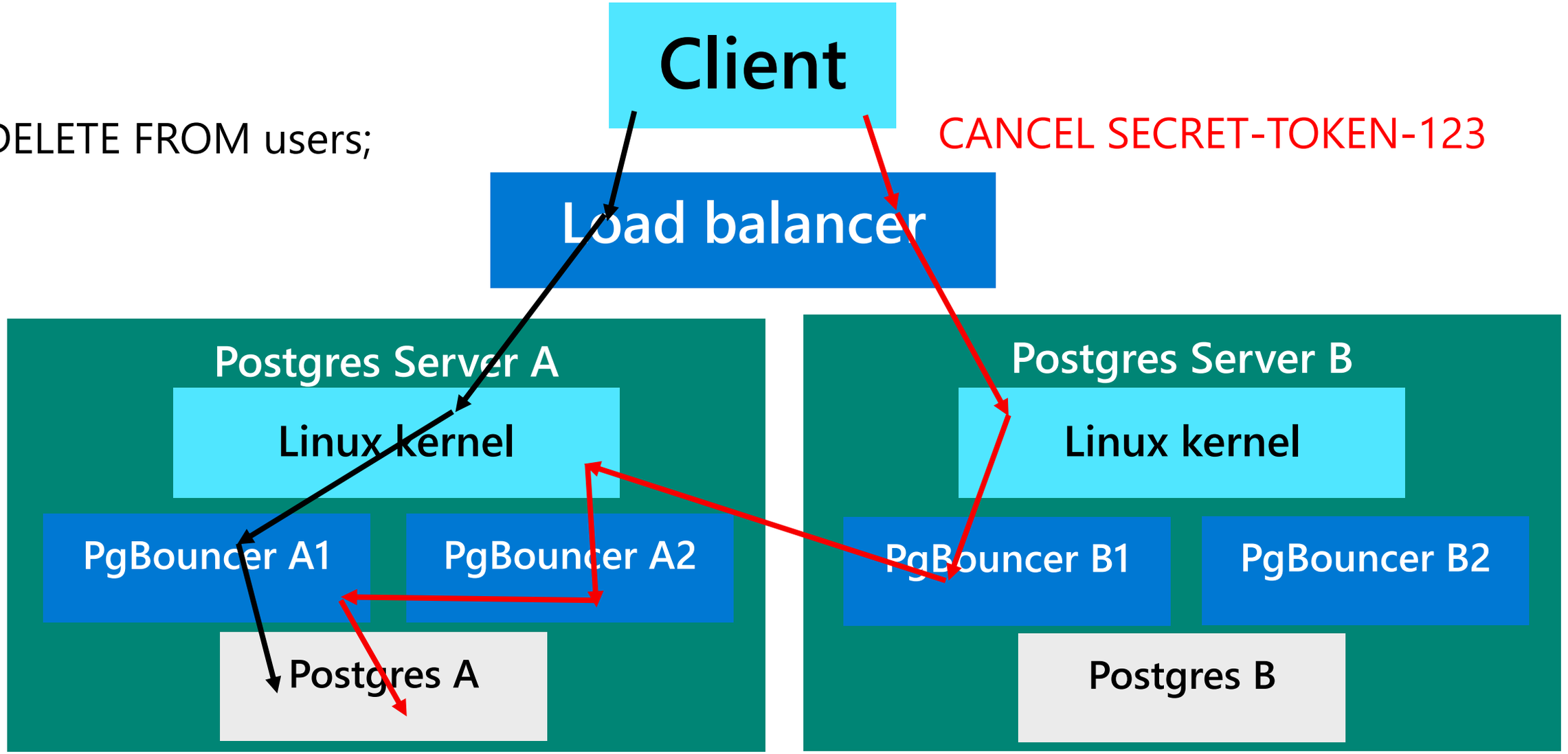
What we end up with

DELETE FROM users;



What we end up with

DELETE FROM users;

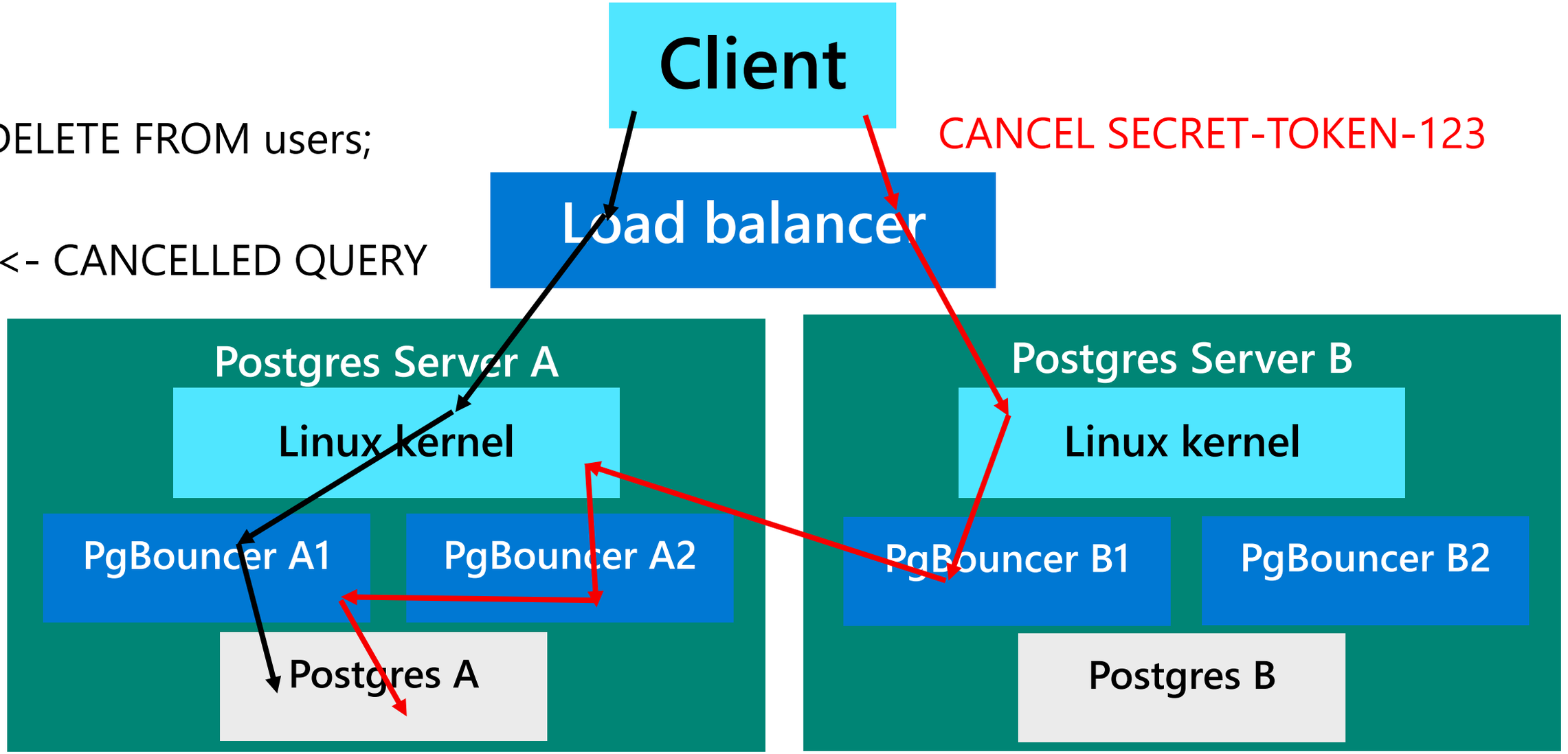


What we end up with

DELETE FROM users;

<- CANCELLED QUERY

CANCEL SECRET-TOKEN-123



The final problem

- To which one should PgBouncer forward it?
- Sending the cancellation to all servers is quite heavy

The final solution

- Encode an identifier in the cancellation token: e.g. SECRET-TOKEN-123-A1

Microsoft Azure

Any questions?