Active Record 8: Resilient By Default

Hartley McGuire - Rails World 2025

✨ Active Record ✨

Hartley McGuire

@skipkayhil

- Rails Issues Team
- Ruby & Rails Infrastructure @Shopify
- https://skipkayhil.github.io/blog

KateSQL

This is a video btw

```
Trilogy::EOFError:
trilogy_query_send:
TRILOGY_CLOSED_CONNECTION
```

500

Internal Server Error

We're sorry, but something went wrong.

If you're the application owner check the logs for more information.

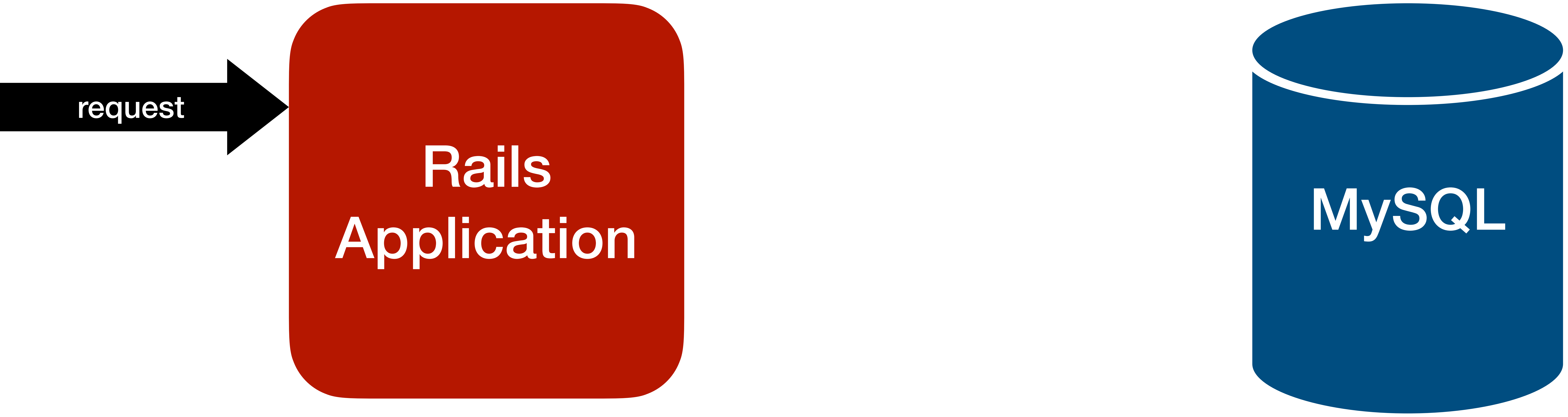
Active Record 8 is the most resilient version of Active Record, *ever.*

Verification

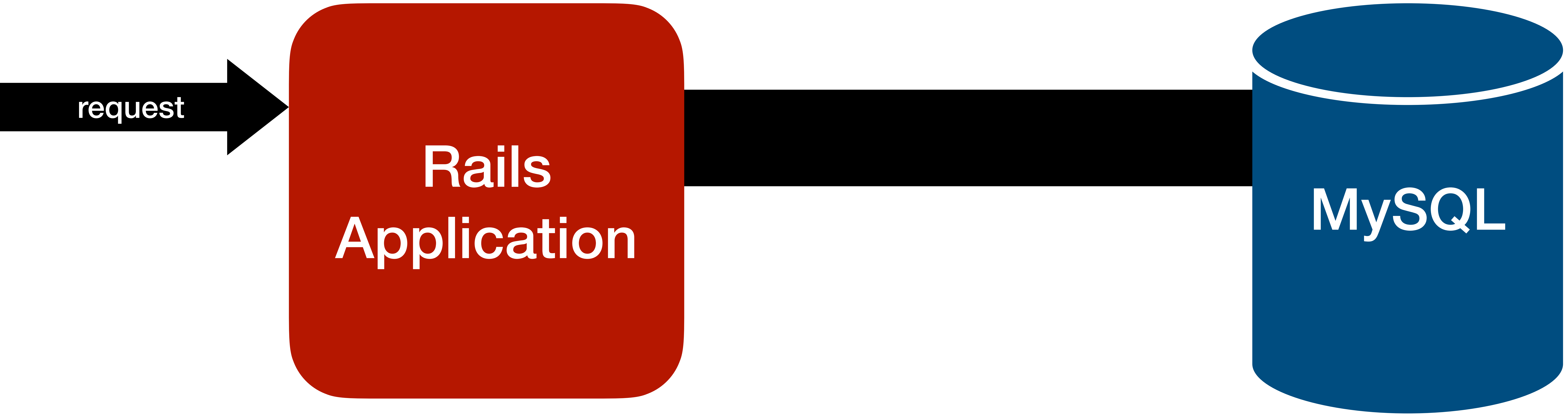
Verification

Rails
Application
MySQL

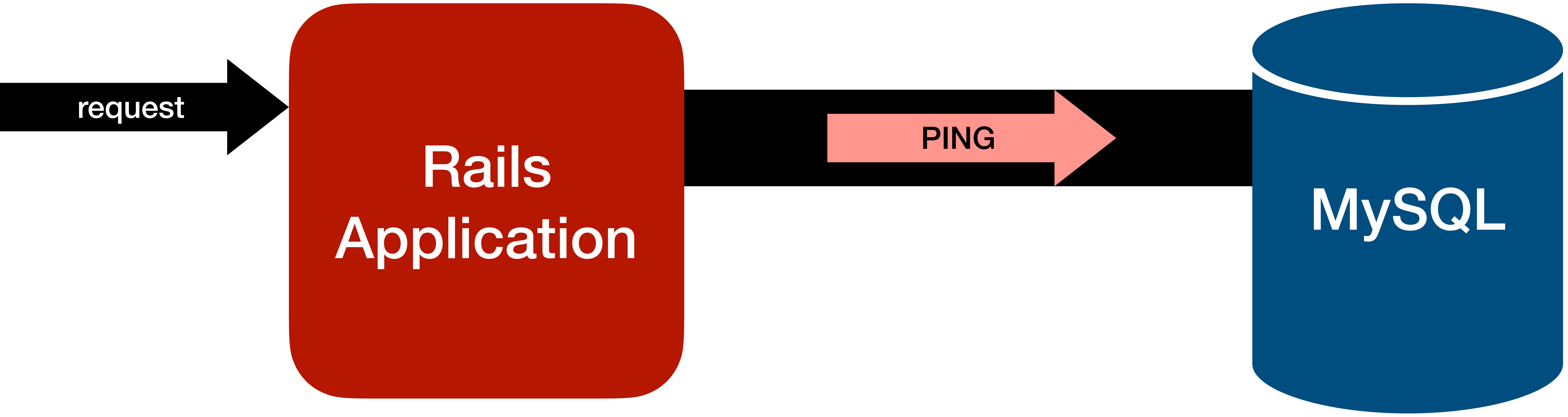
Verification



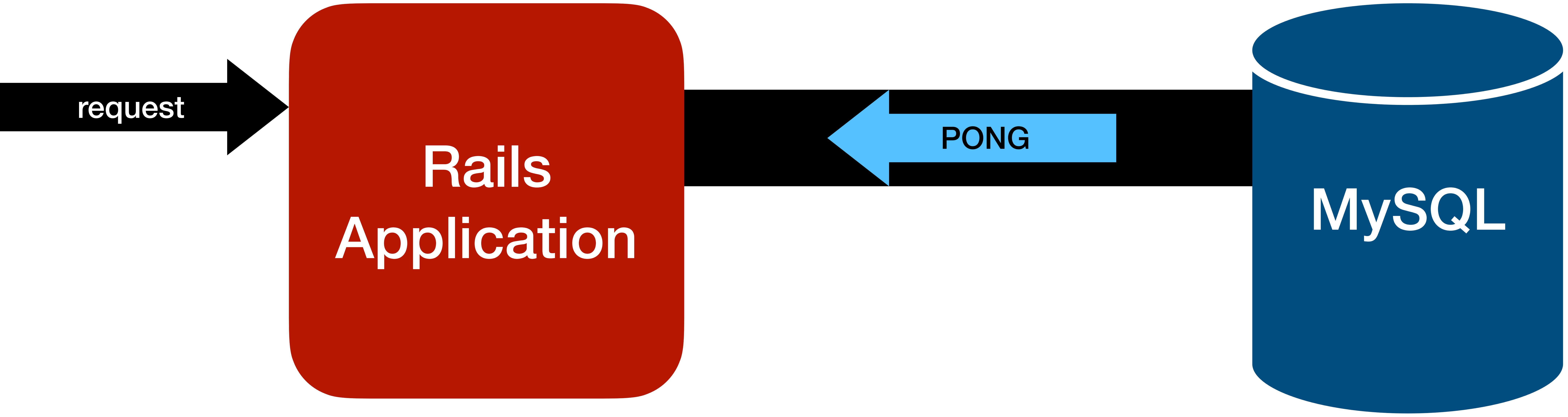
Verification



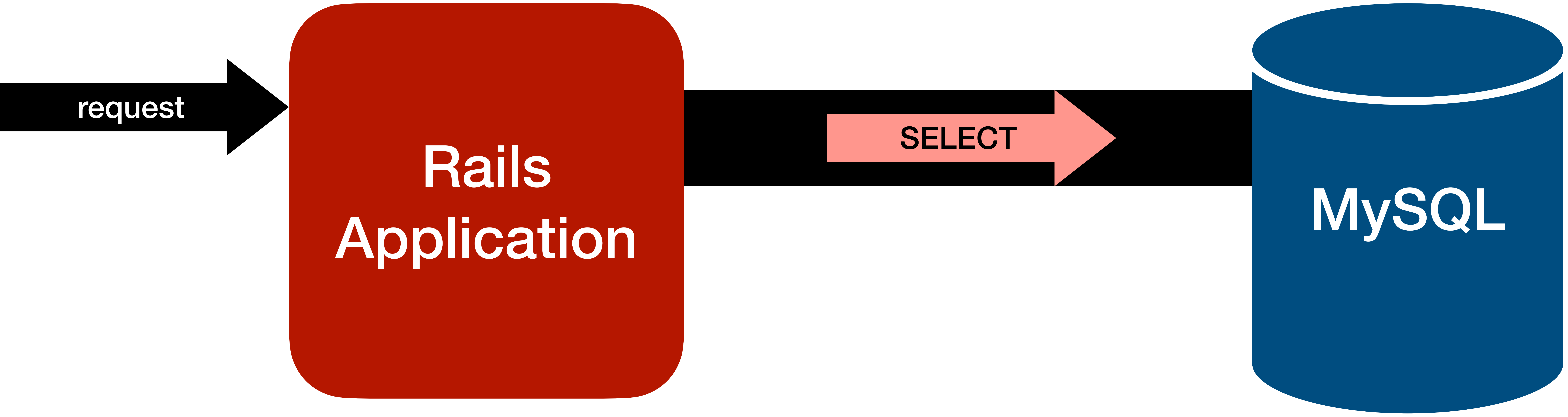
Verification



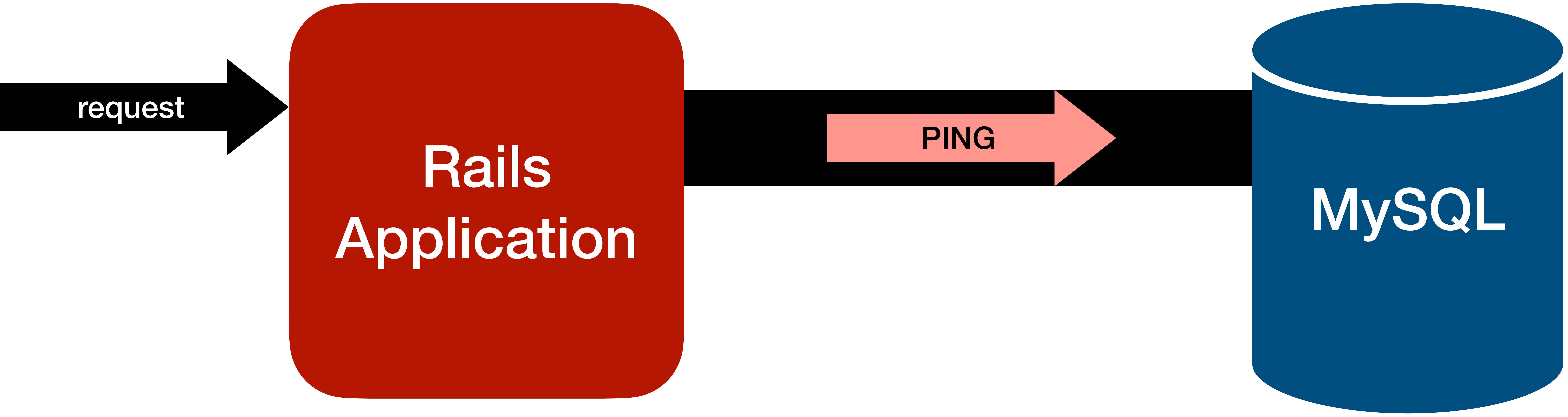
Verification



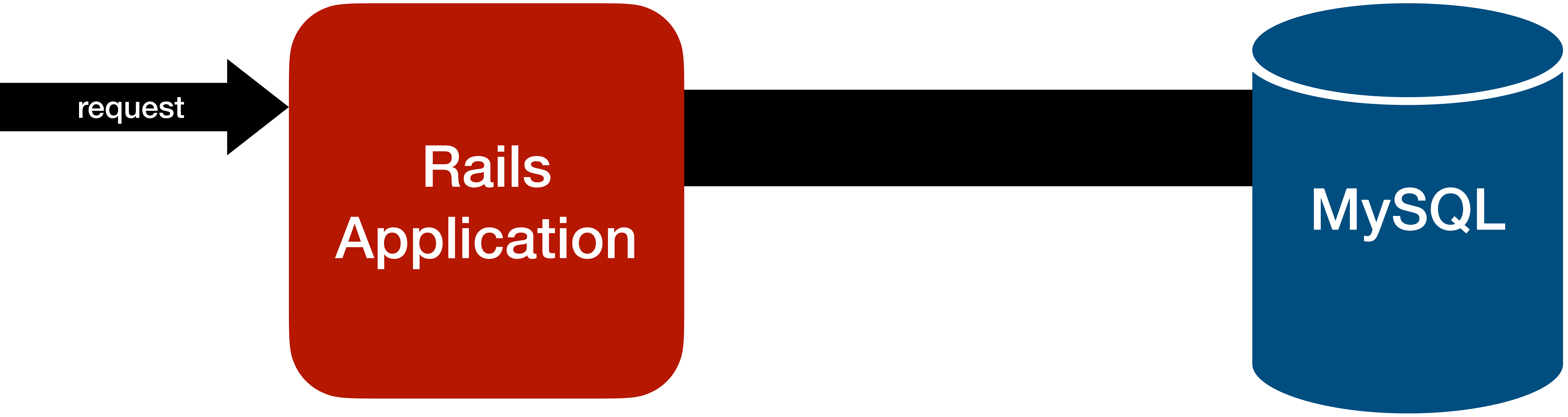
Verification



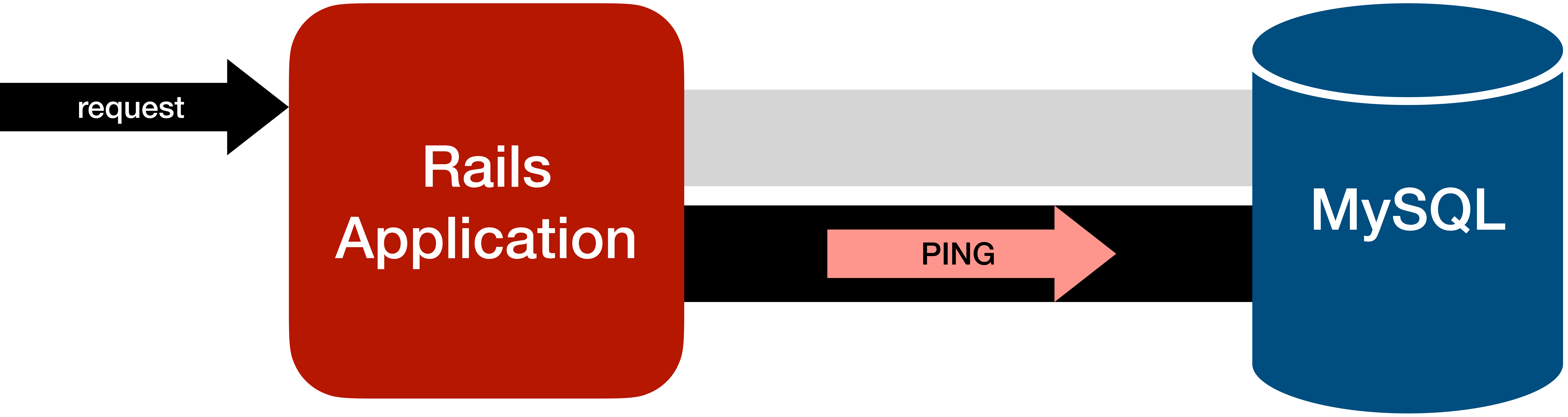
Verification



Verification

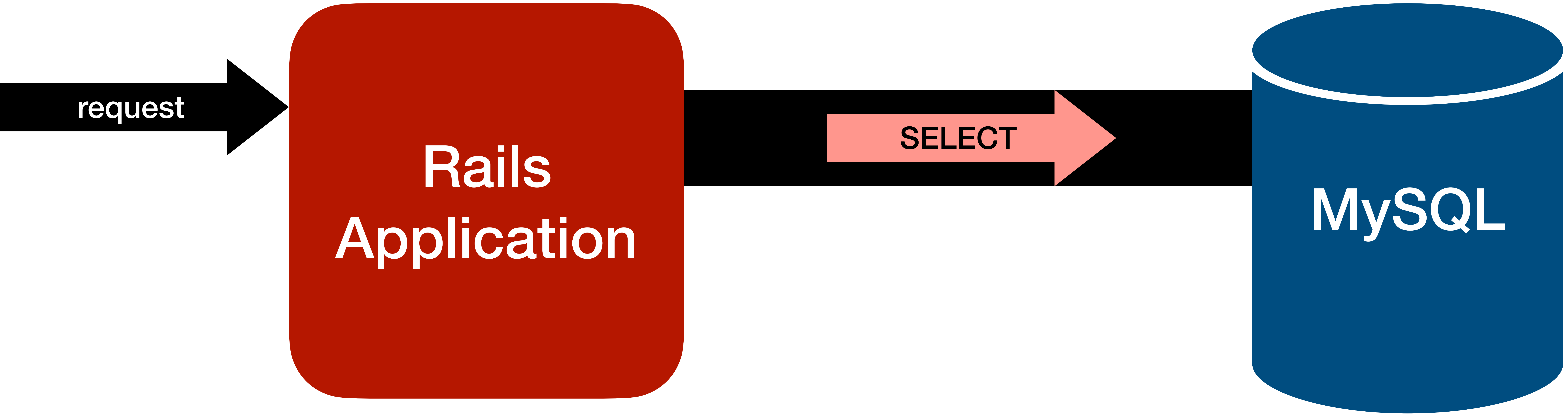


Verification

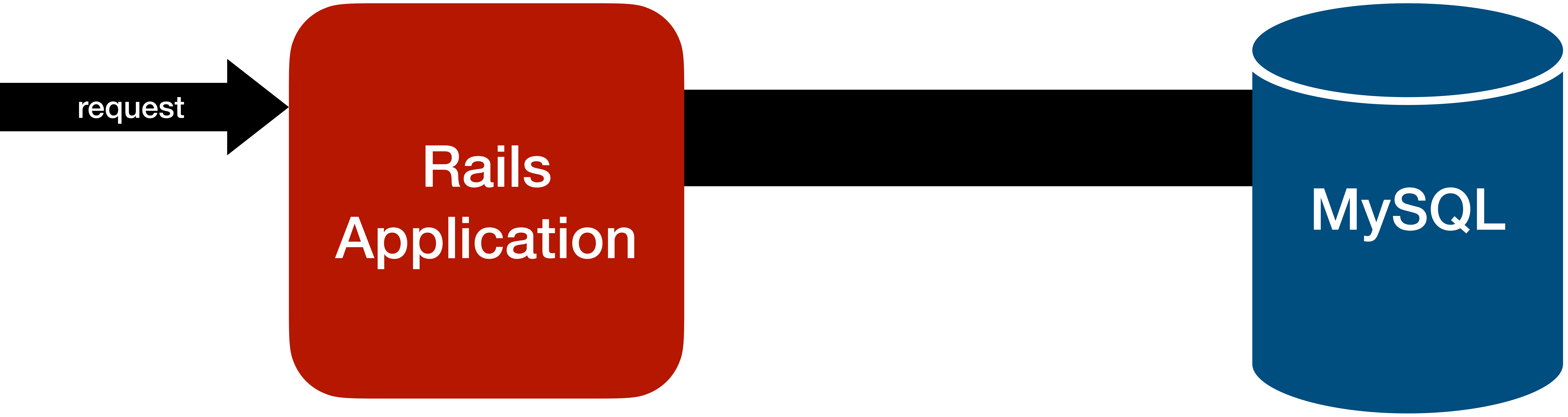


```
From 81c5242f43cb45d97b2a56409f8b39b0dba75ac3 Mon Sep 17 00:00:00 2001
From: Jeremy Kemper <jeremy@bitsweat.net>
Date: Sat, 19 Nov 2005 10:55:11 +0000
Subject: [PATCH]  r3181@asus:  jeremy | 2005-11-19 02:52:24 -0800  Mark
 connections for verification.  Retrieve connection verifies before returning
 a connection.  Verification tests whether the connection is marked then
 reconnects if the connection is inactive.  All active connections are marked
 for verification after a request is handled.  References #428.
```

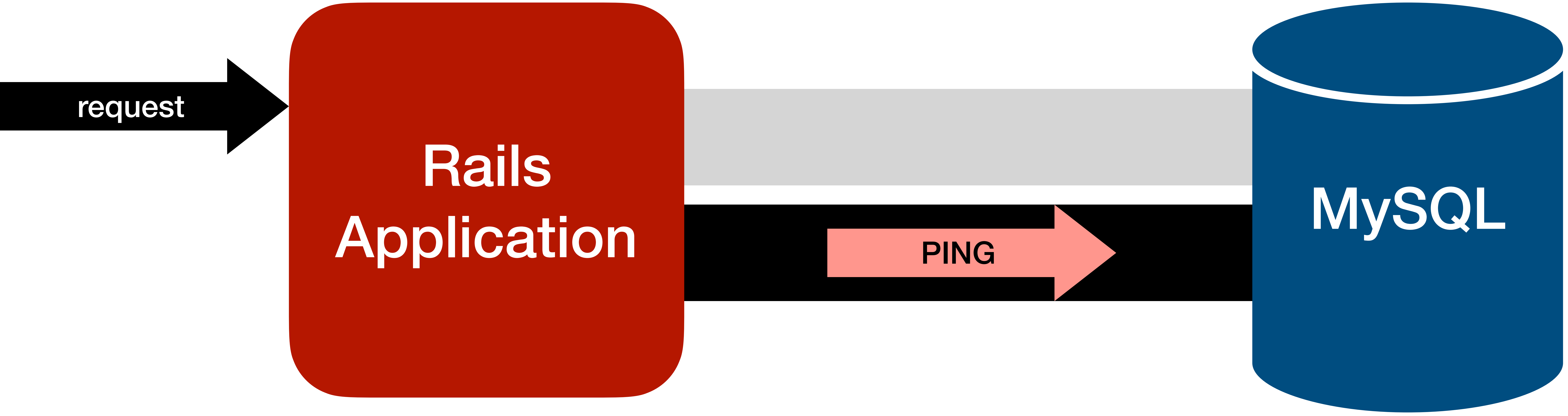
Verification



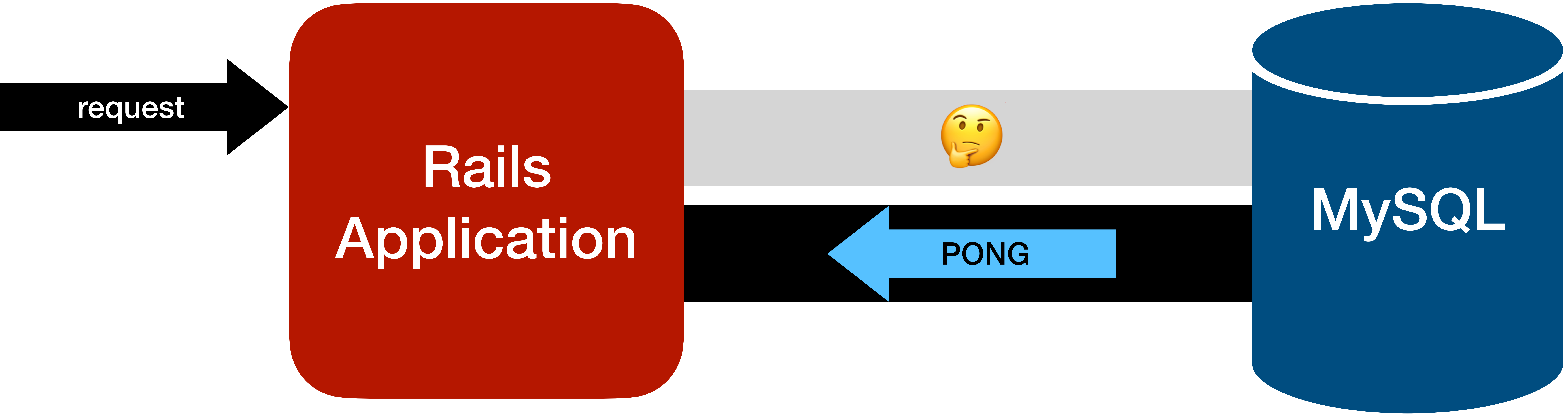
Verification



Verification



Verification



😅

Verification

INSERT
idle
SELECT

Verification

INSERT
idle
SELECT

Verification

INSERT
idle
PING
SELECT

Verification

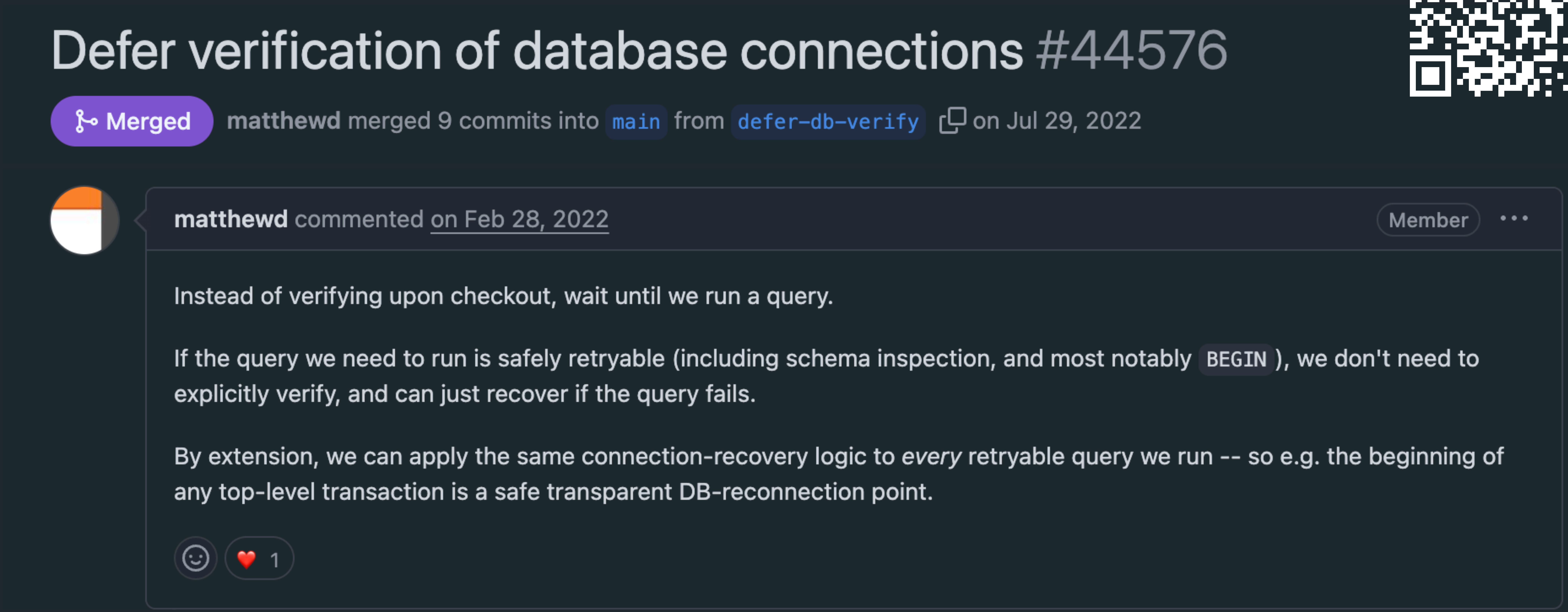
INSERT
idle
PING
SELECT

Verification

INSERT
idle
PING
SELECT

Rails 1.1.0

Rails 7.1.0



Defer verification of database connections #44576

Merged matthewd merged 9 commits into `main` from `defer-db-verify` on Jul 29, 2022

matthewd commented on Feb 28, 2022 · Member

Instead of verifying upon checkout, wait until we run a query.

If the query we need to run is safely retryable (including schema inspection, and most notably `BEGIN`), we don't need to explicitly verify, and can just recover if the query fails.

By extension, we can apply the same connection-recovery logic to *every* retryable query we run -- so e.g. the beginning of any top-level transaction is a safe transparent DB-reconnection point.

❤️ 1

Verification

SELECT
idle
PING
BEGIN
INSERT
COMMIT

Verification

SELECT
idle
PING
BEGIN
INSERT
COMMIT

Verification

SELECT
idle
PING
BEGIN
INSERT
COMMIT

Verification

SELECT
idle
PING
BEGIN
INSERT
COMMIT

Verification

SELECT
idle
PING
BEGIN
INSERT
COMMIT

Verification

SELECT
idle
PING
BEGIN
INSERT
COMMIT

Verification

SELECT
idle
BEGIN
INSERT
COMMIT

Verification

The Summary

- PING to verify-before-checkout
- Are queries retryable? 🤷‍♂️

Rails 7.1

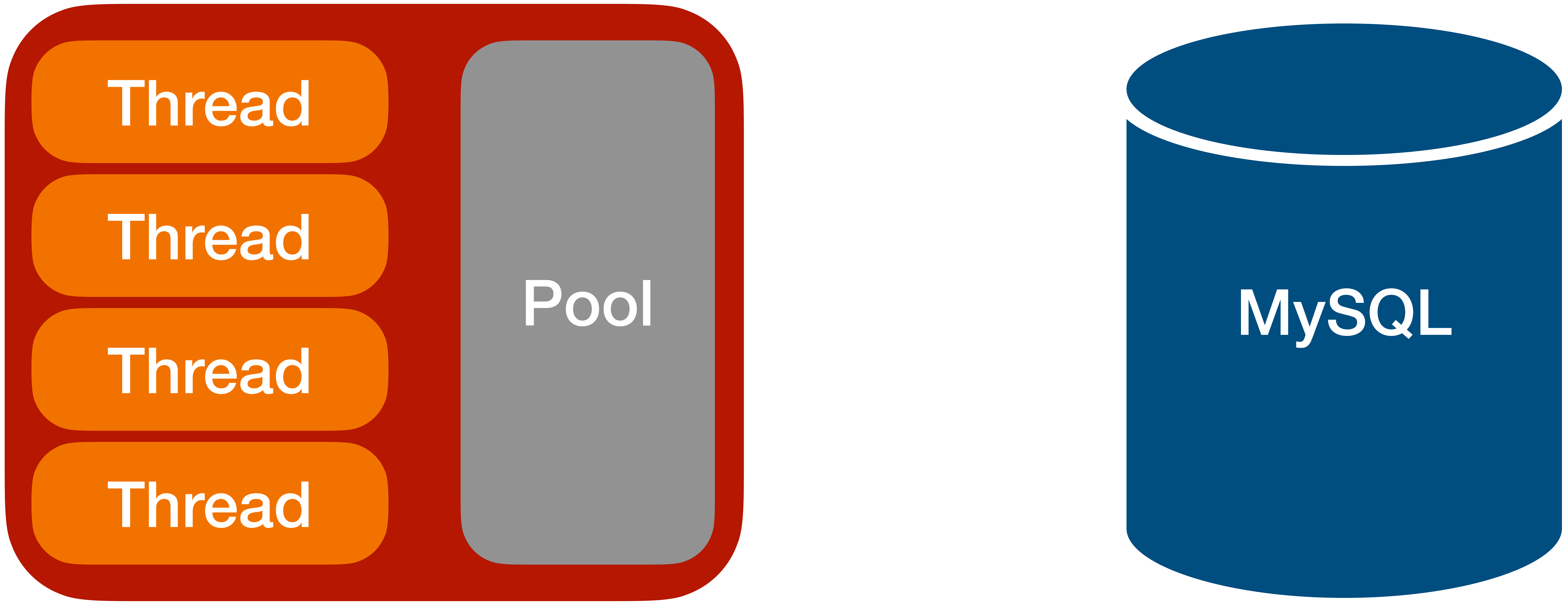
- Are queries retryable? *Sometimes…*
- First query retryable → Skip PING
- Retryable query → Recovery point

Connection Pinning

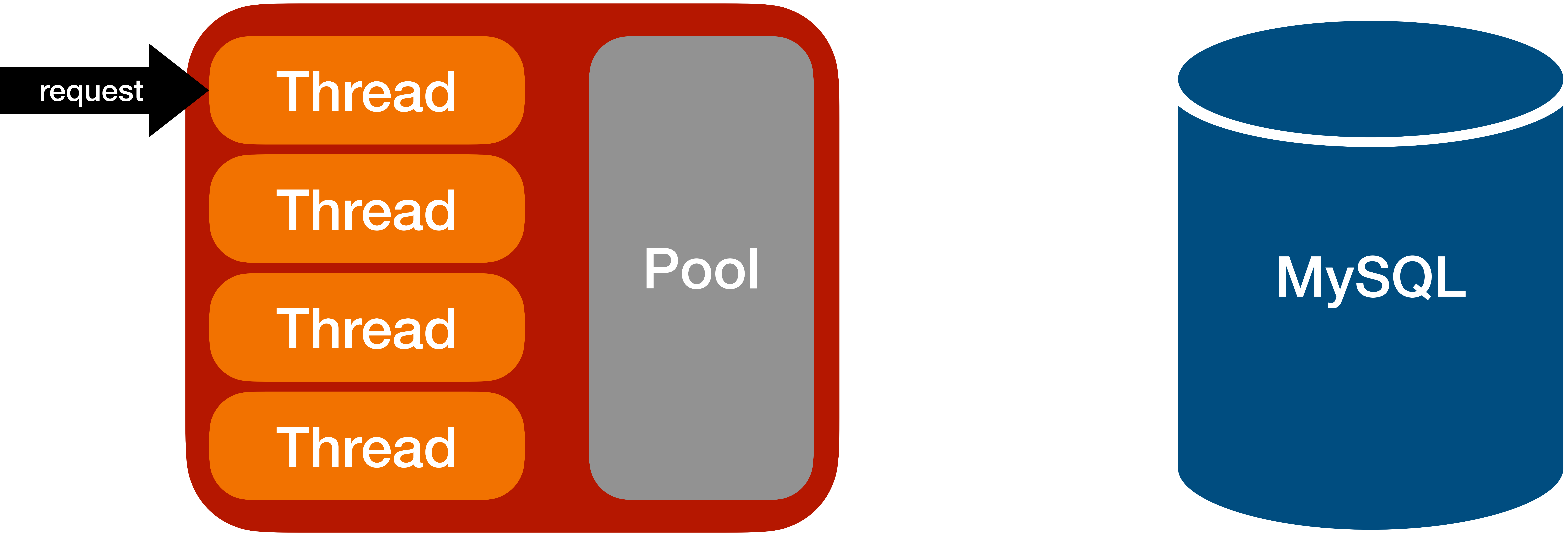
Connection Pinning

Rails Application
MySQL

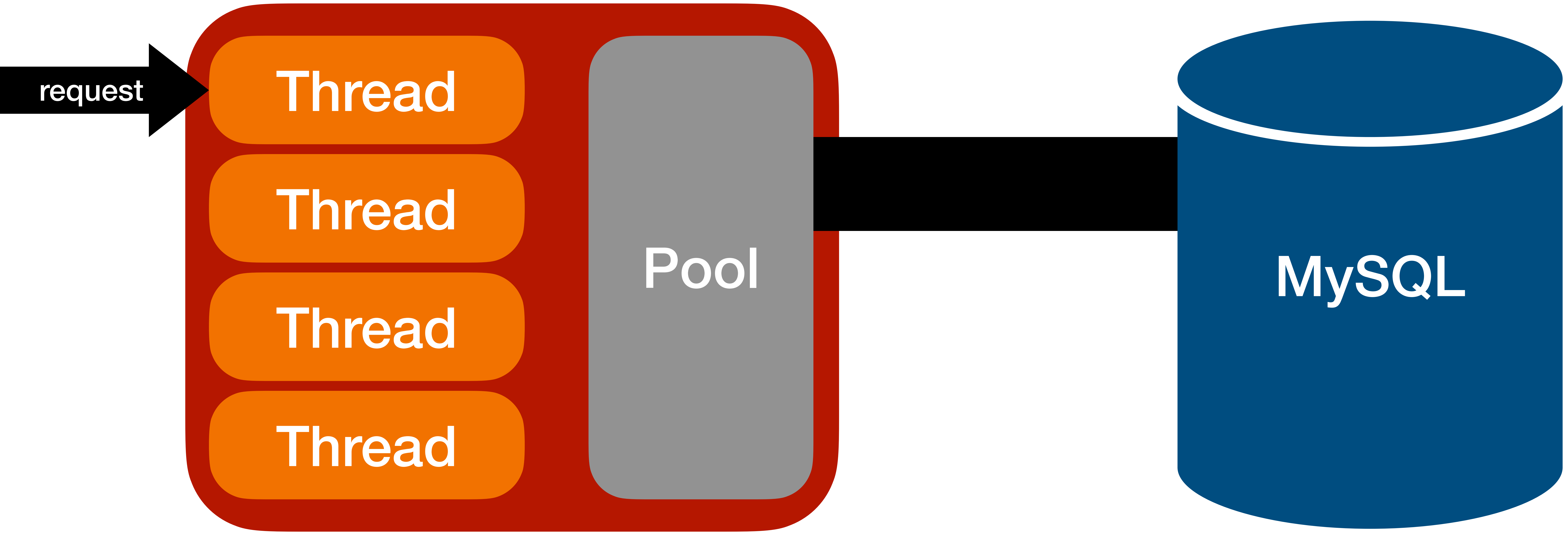
Connection Pinning



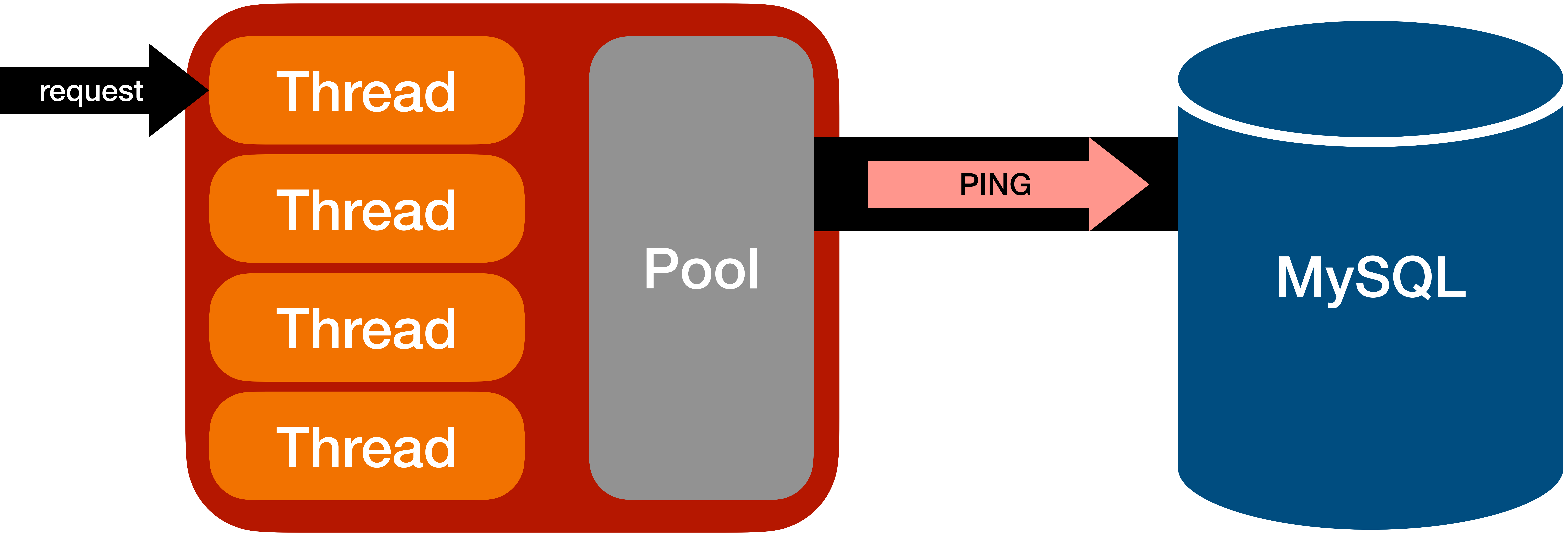
Connection Pinning



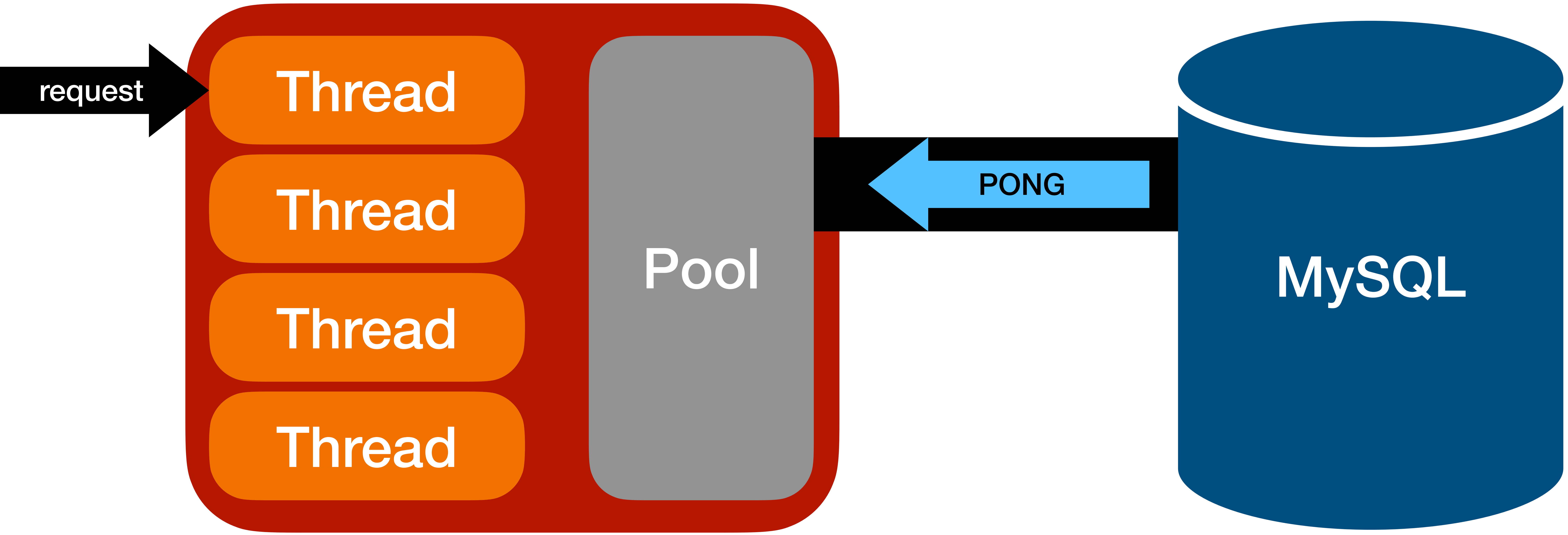
Connection Pinning



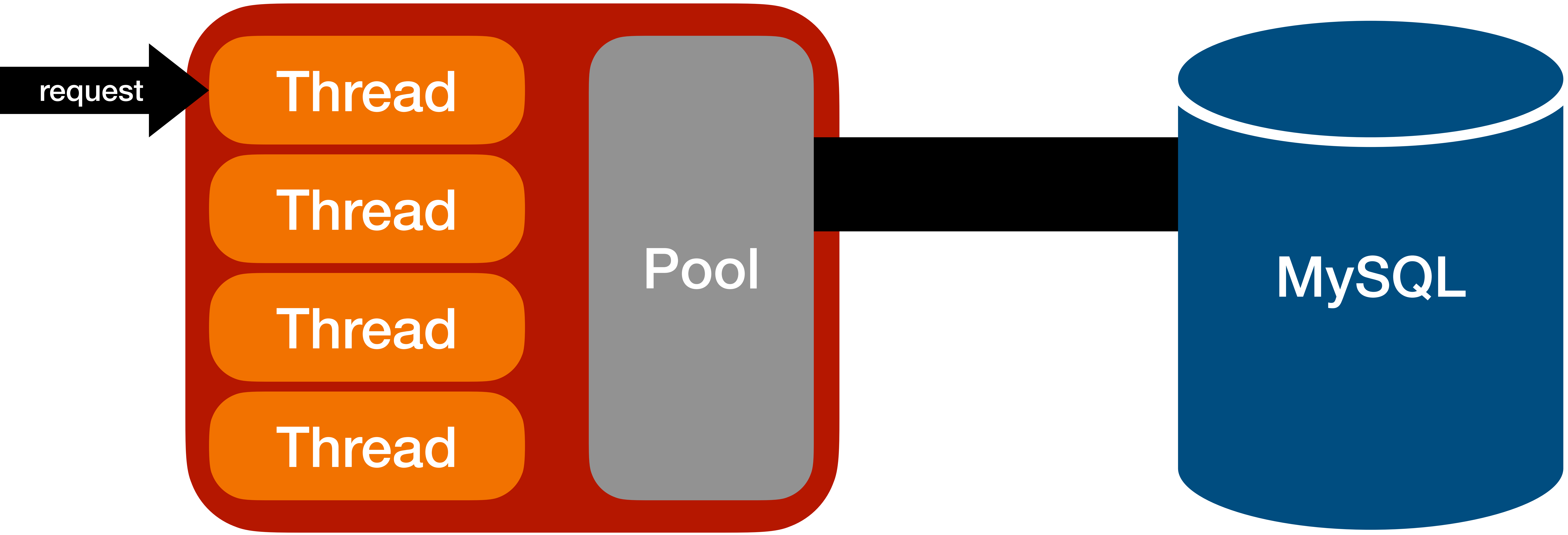
Connection Pinning



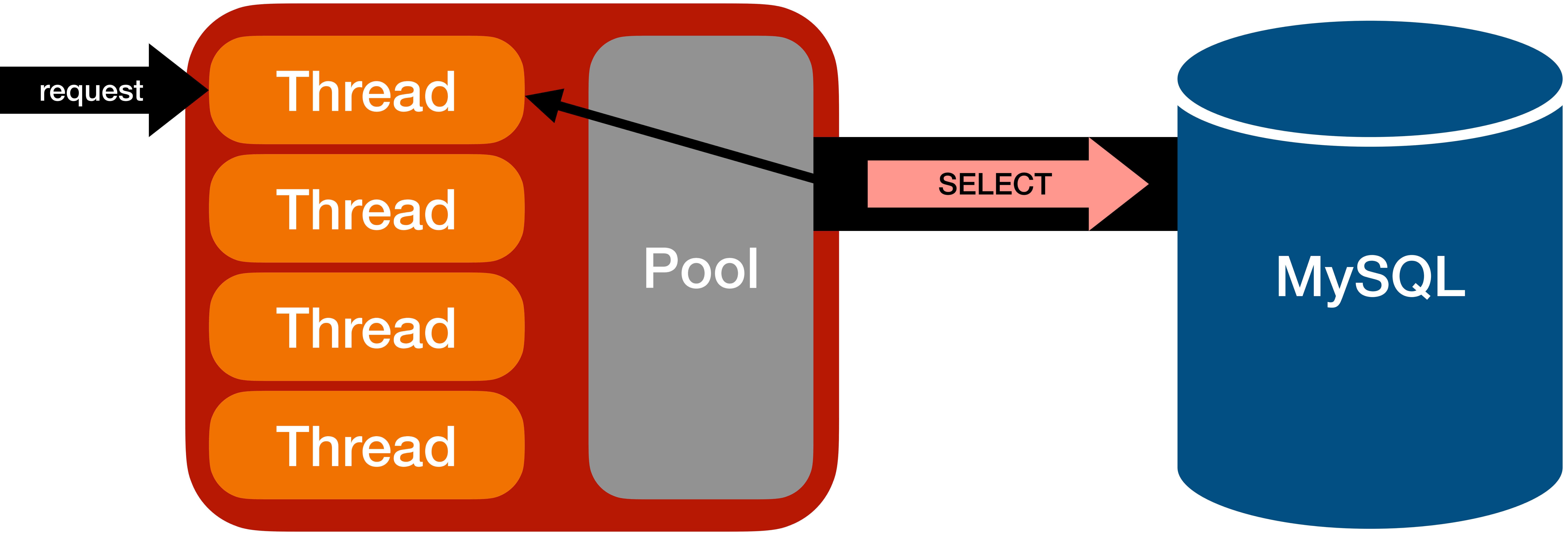
Connection Pinning



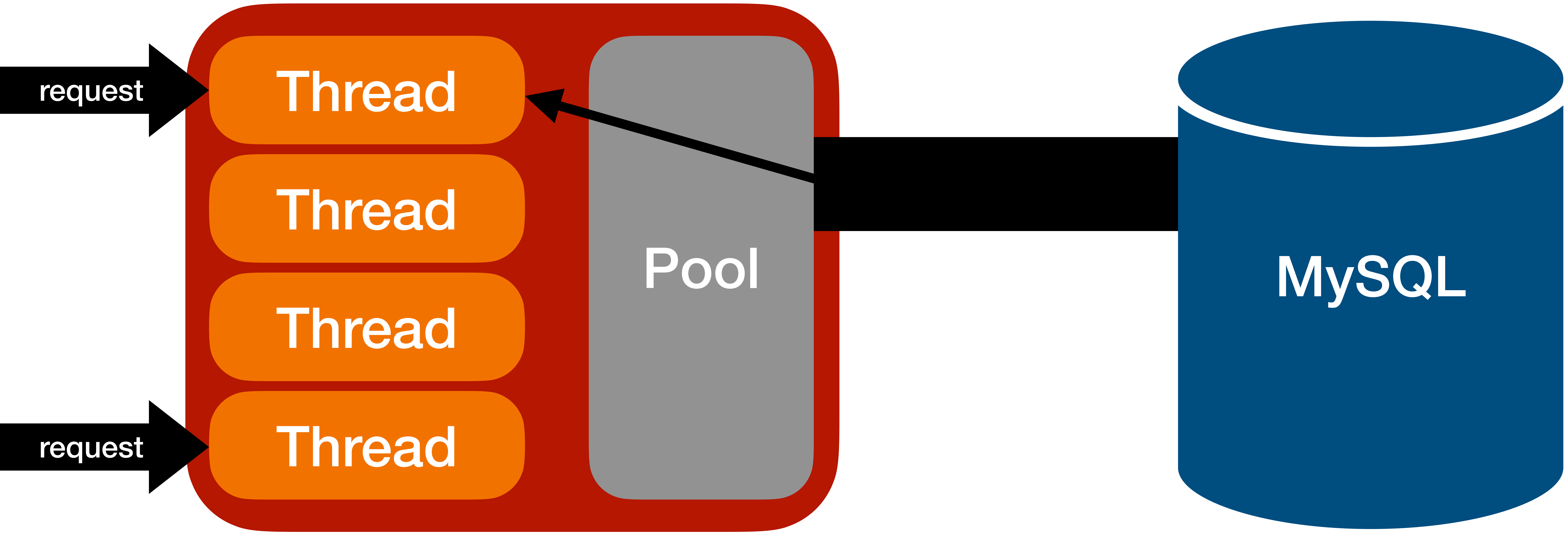
Connection Pinning



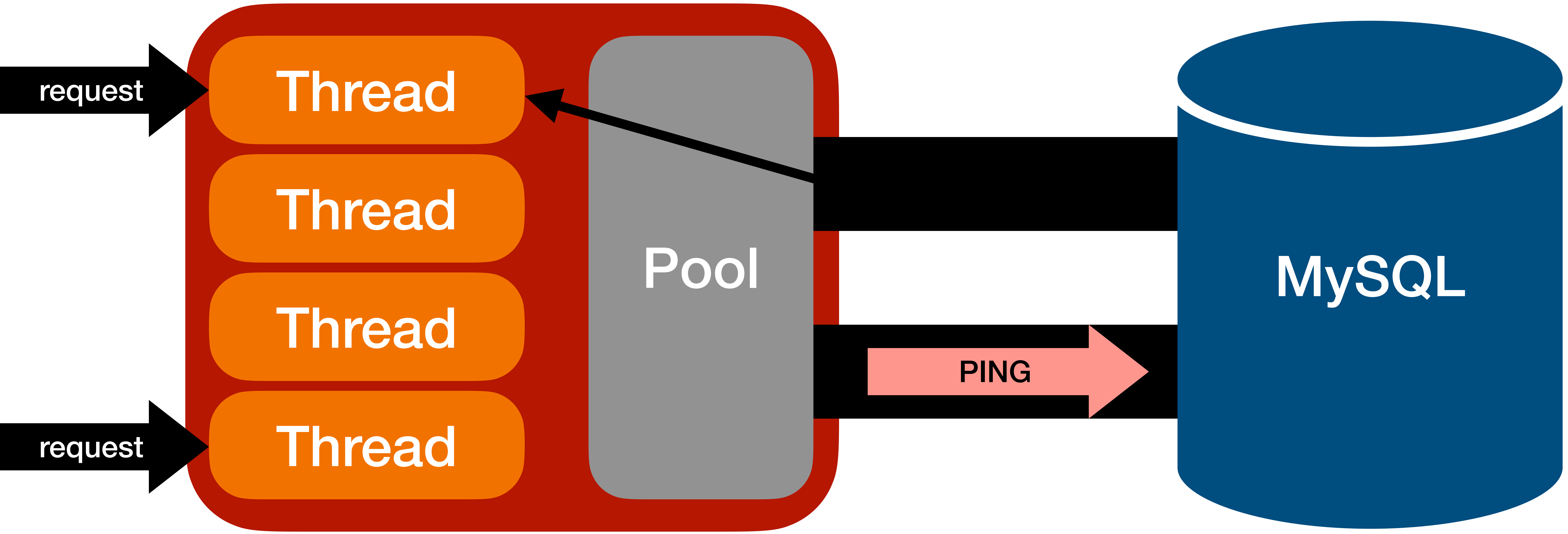
Connection Pinning



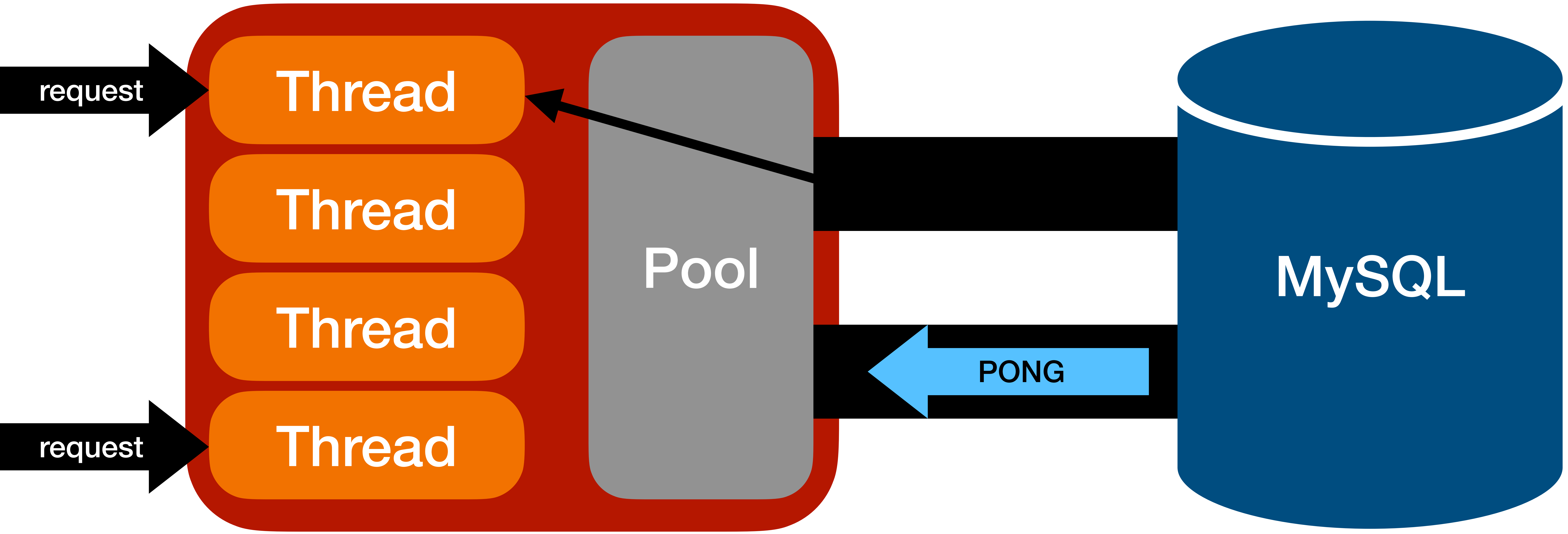
Connection Pinning



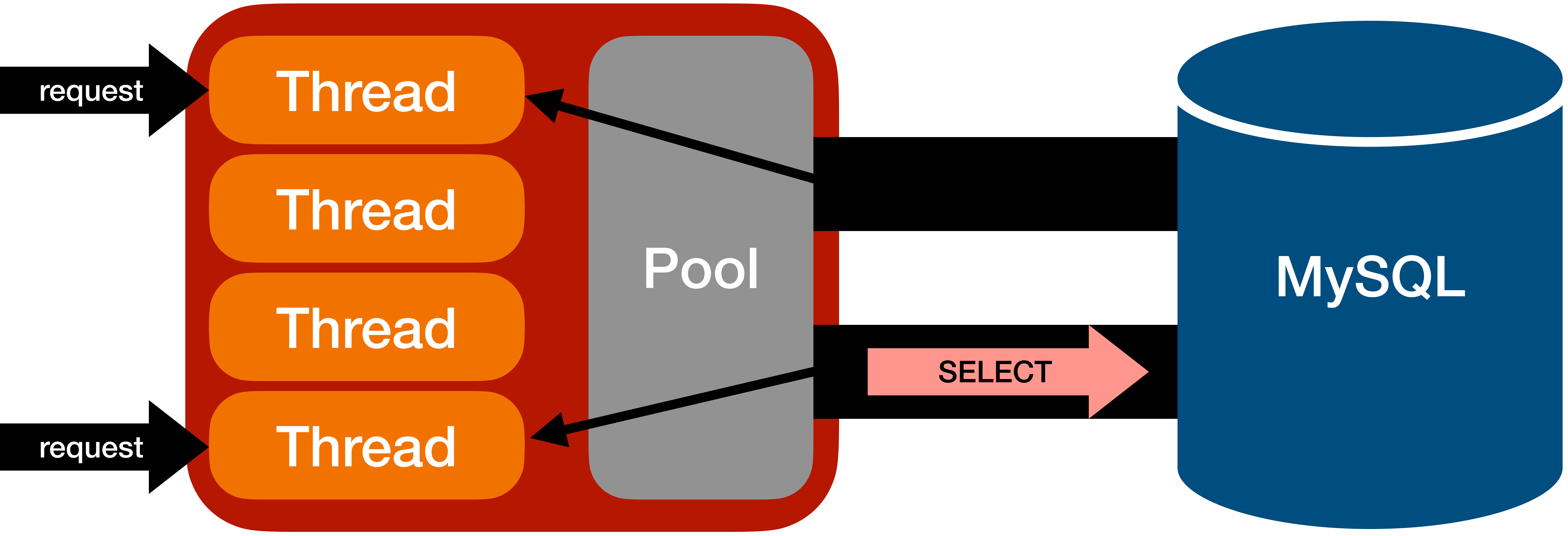
Connection Pinning



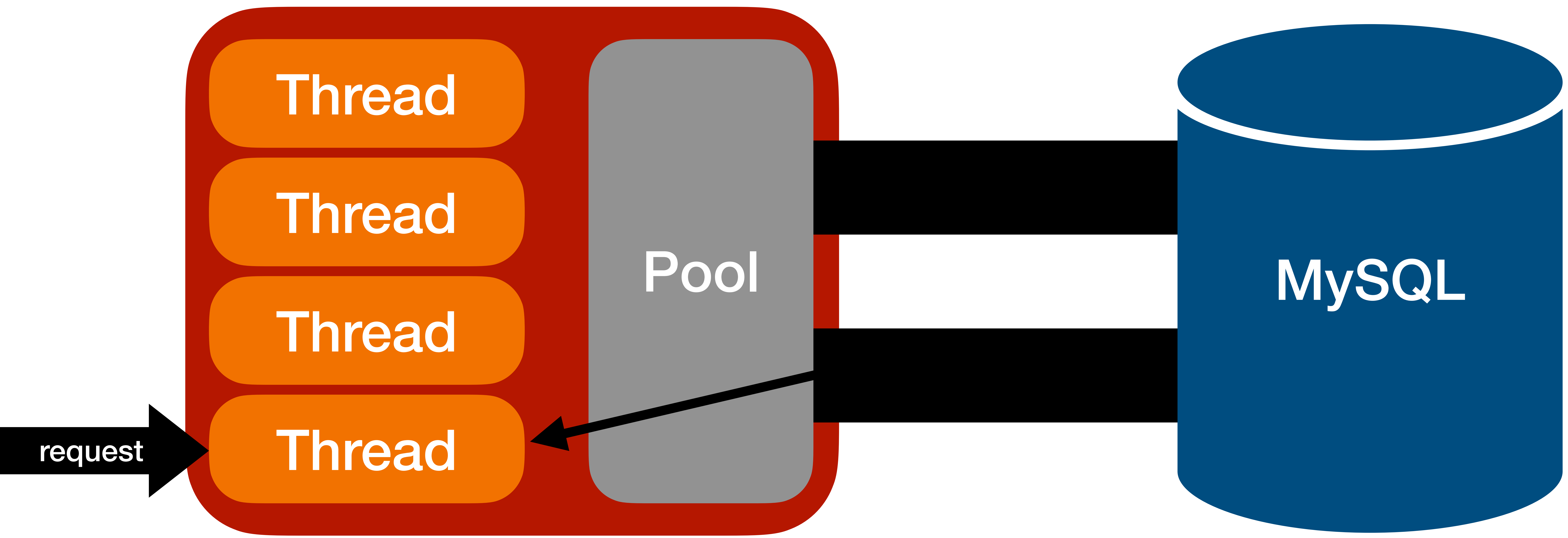
Connection Pinning



Connection Pinning



Connection Pinning



Connection Pinning

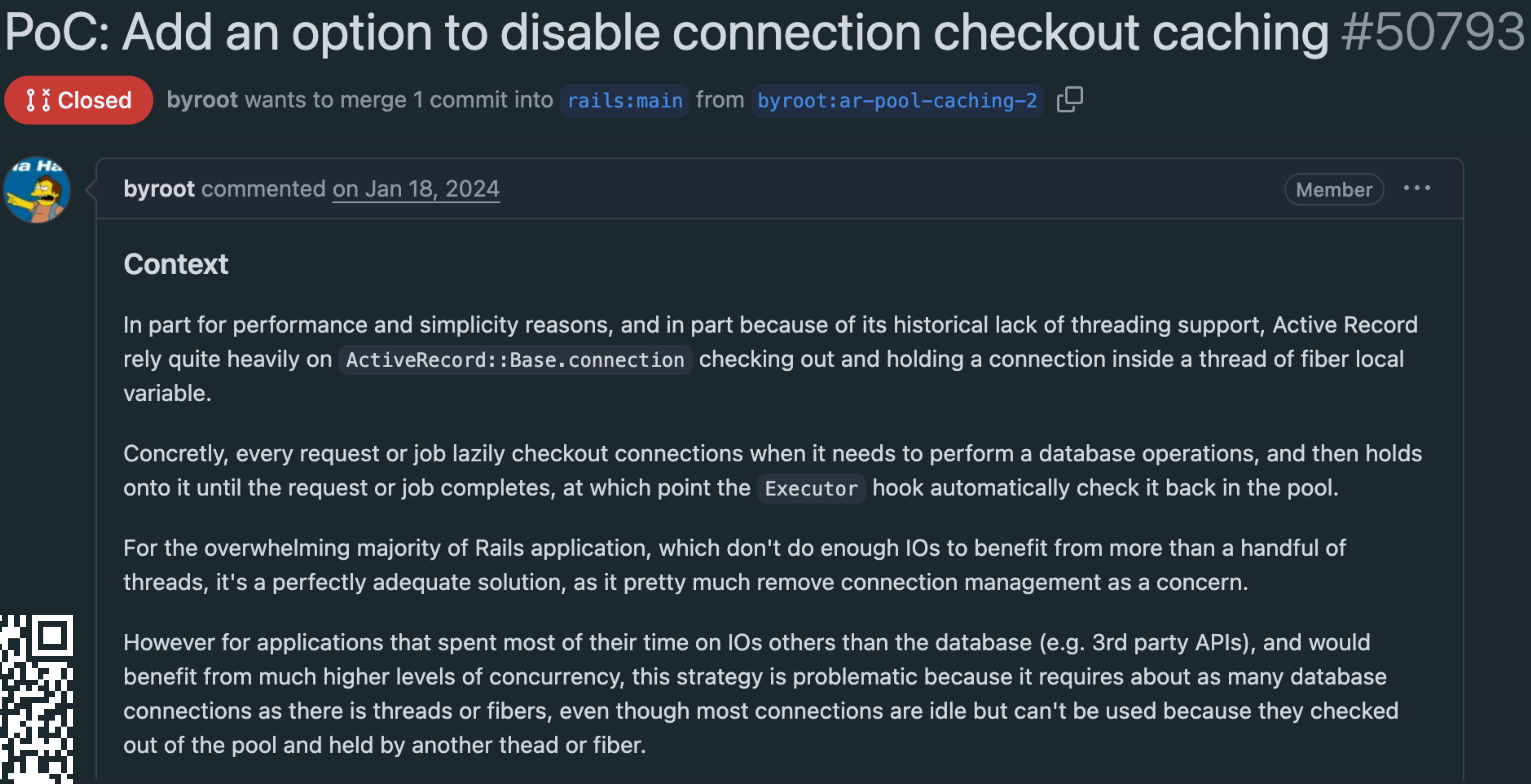
PING
BEGIN
INSERT
COMMIT
BEGIN
INSERT
COMMIT
CHECKOUT
CHECKIN

Connection Pinning

BEGIN
INSERT
COMMIT
BEGIN
INSERT
COMMIT
CHECKOUT
CHECKIN

Rails 7.1.0

Rails 7.2.0



PoC: Add an option to disable connection checkout caching #50793

Closed byroot wants to merge 1 commit into `rails:main` from `byroot:ar-pool-caching-2`

byroot commented on Jan 18, 2024 Member

Context

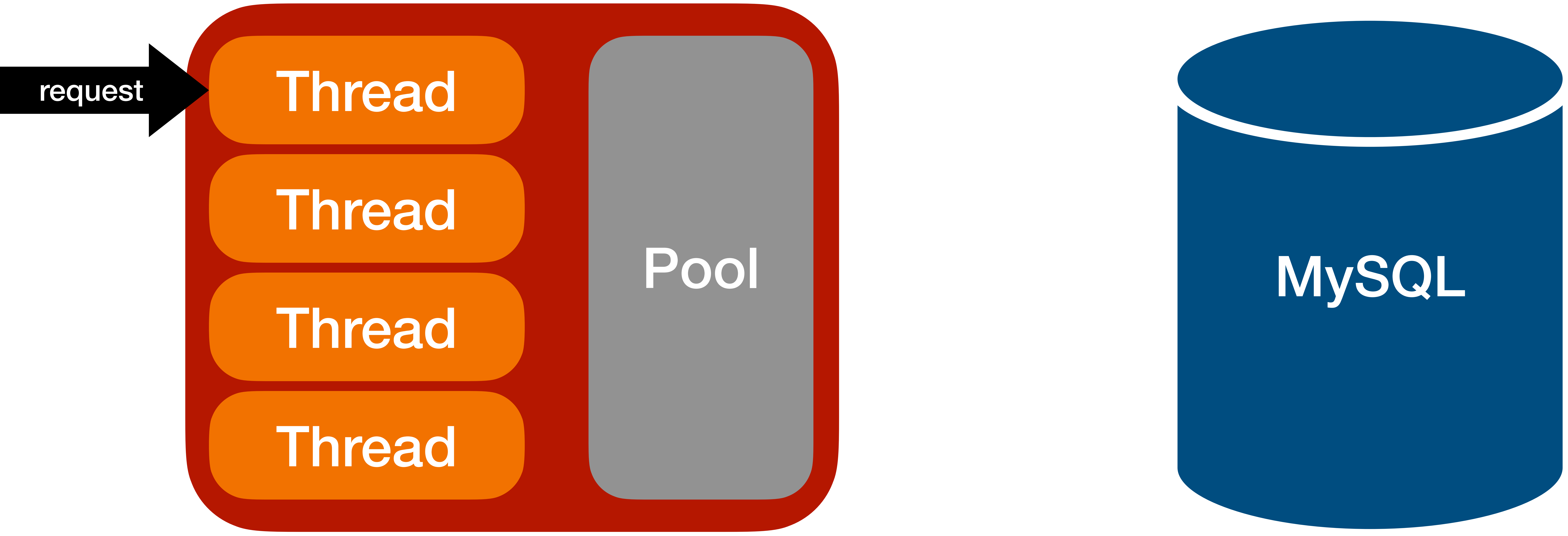
In part for performance and simplicity reasons, and in part because of its historical lack of threading support, Active Record rely quite heavily on `ActiveRecord::Base.connection` checking out and holding a connection inside a thread of fiber local variable.

Concretly, every request or job lazily checkout connections when it needs to perform a database operations, and then holds onto it until the request or job completes, at which point the `Executor` hook automatically check it back in the pool.

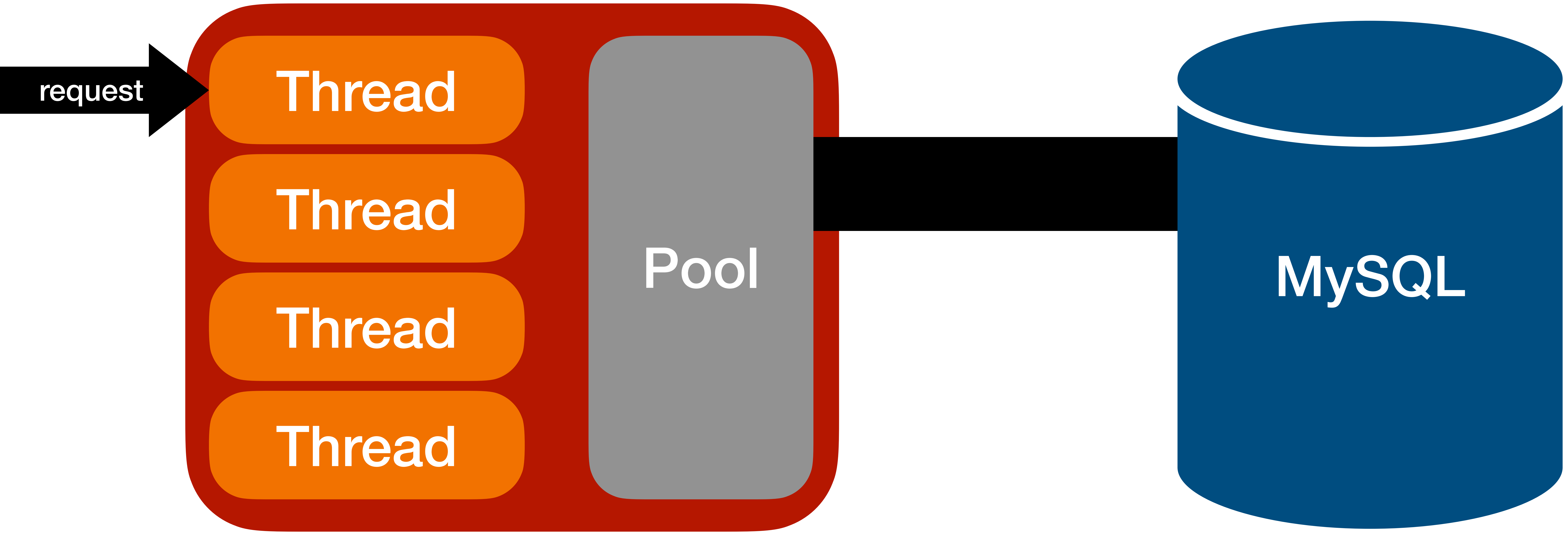
For the overwhelming majority of Rails application, which don't do enough IOs to benefit from more than a handful of threads, it's a perfectly adequate solution, as it pretty much remove connection management as a concern.

However for applications that spent most of their time on IOs others than the database (e.g. 3rd party APIs), and would benefit from much higher levels of concurrency, this strategy is problematic because it requires about as many database connections as there is threads or fibers, even though most connections are idle but can't be used because they checked out of the pool and held by another thead or fiber.

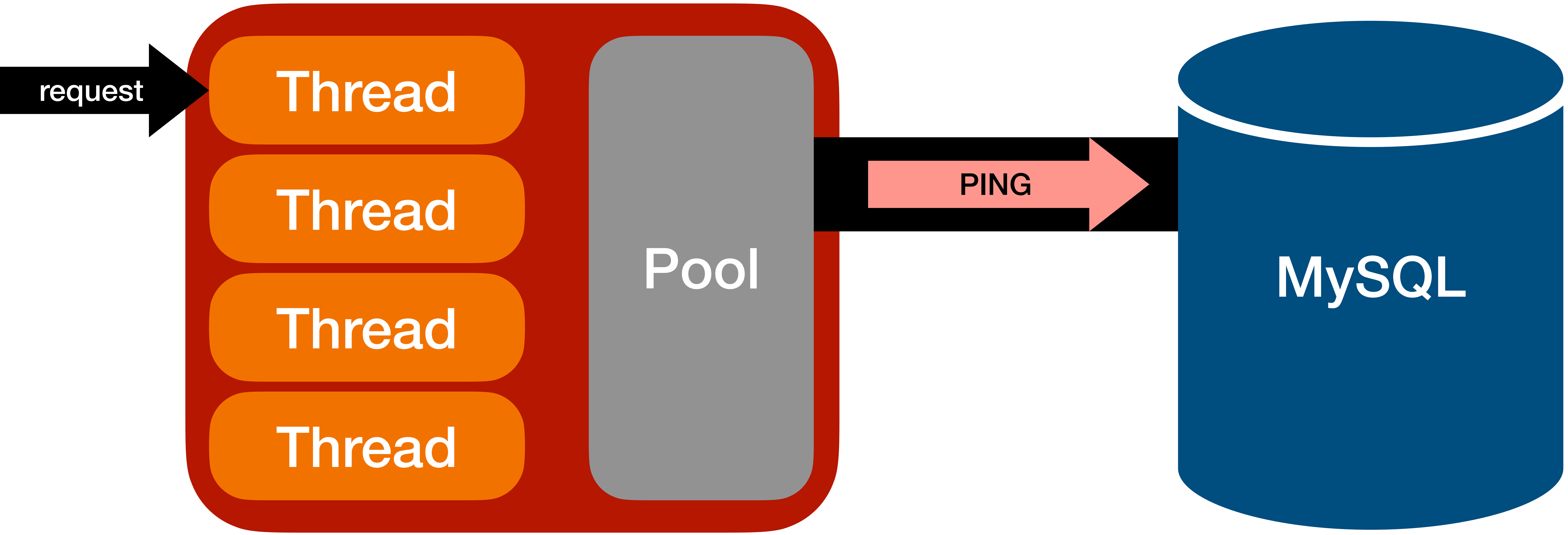
Connection Pinning



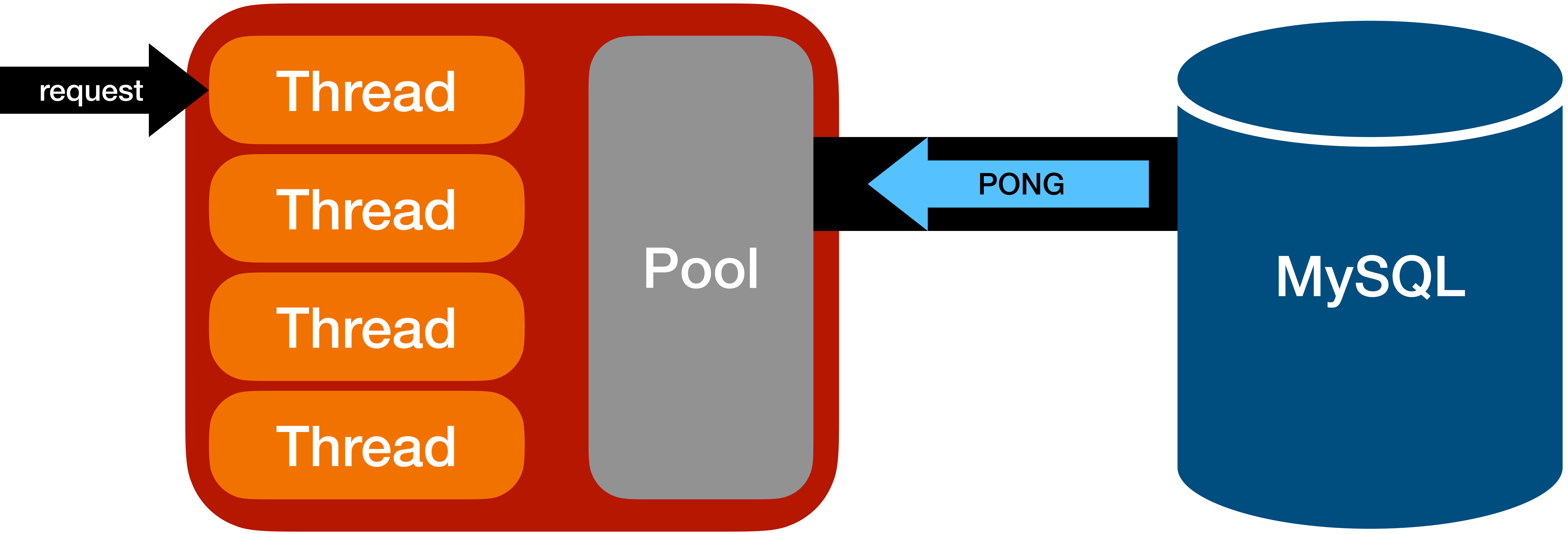
Connection Pinning



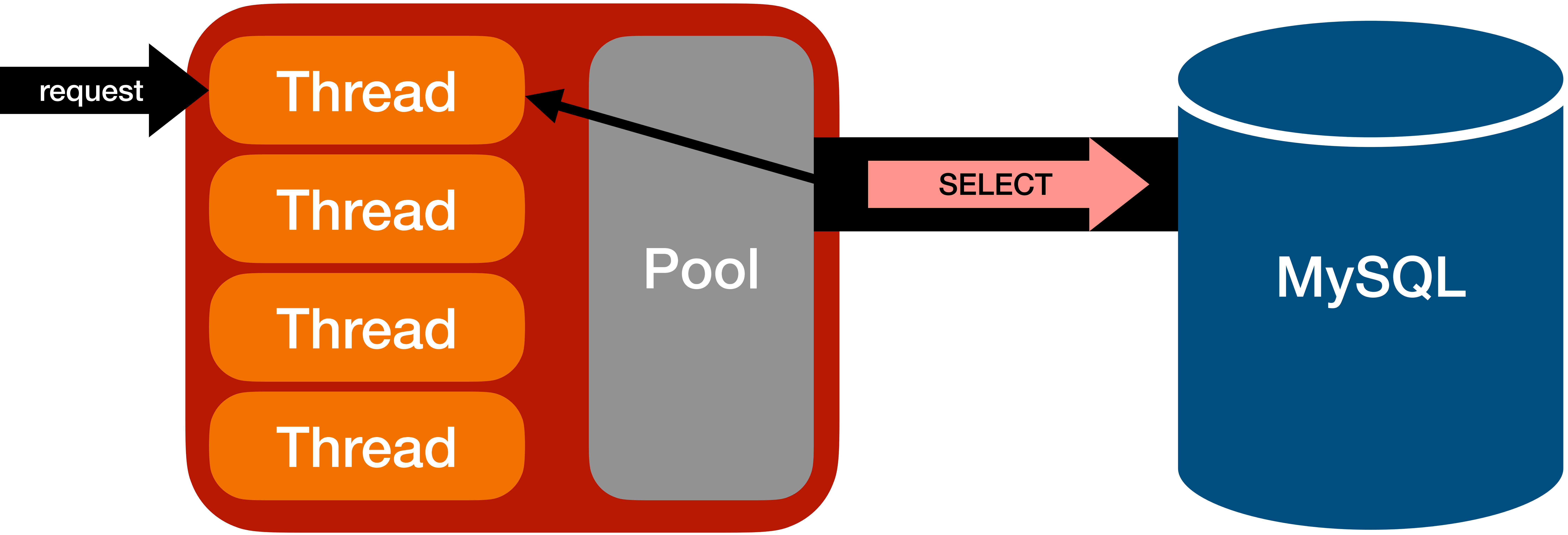
Connection Pinning



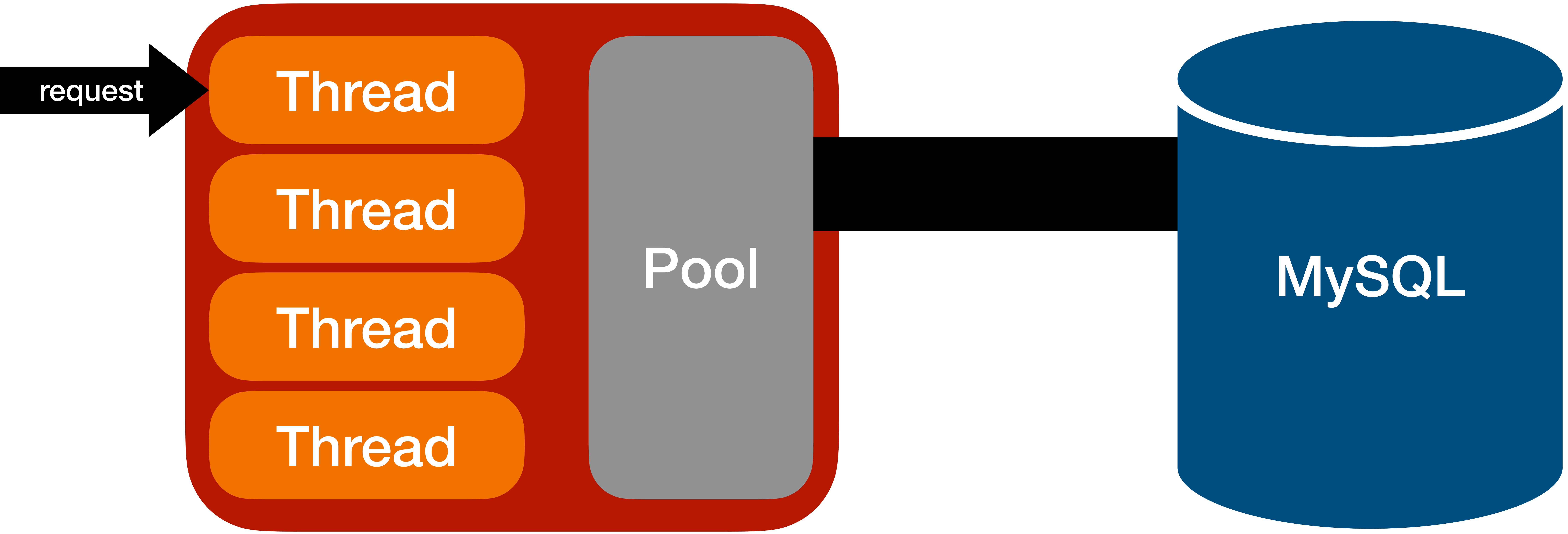
Connection Pinning



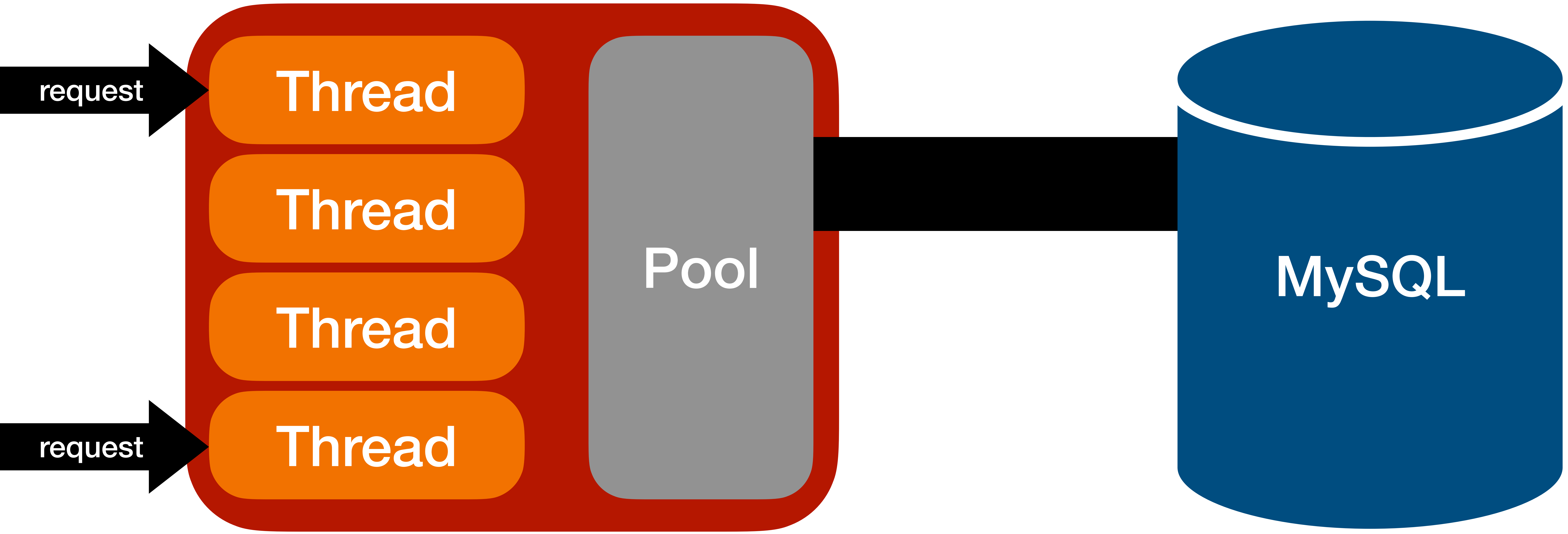
Connection Pinning



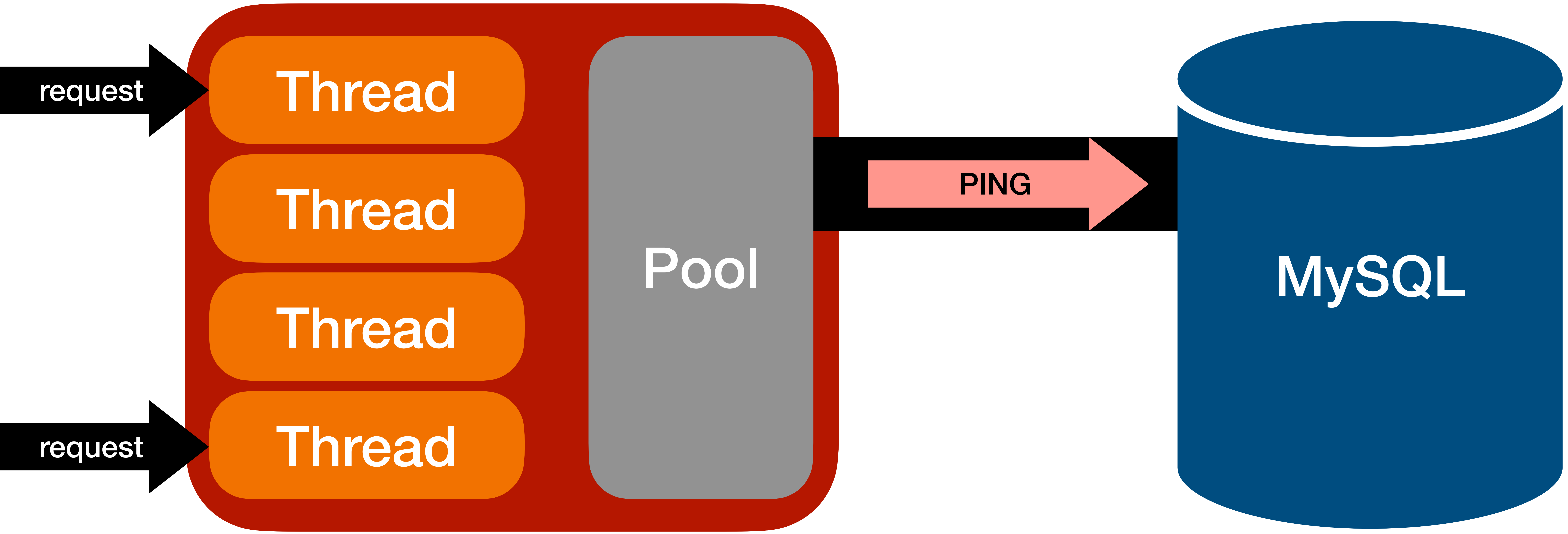
Connection Pinning



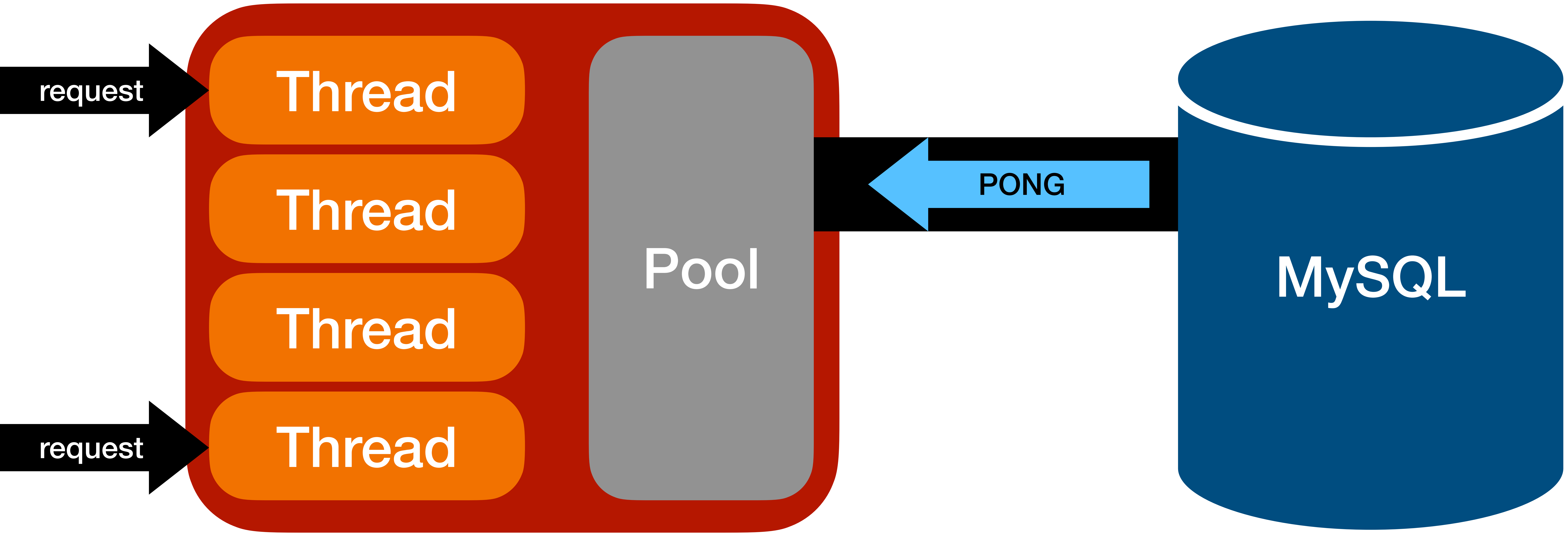
Connection Pinning



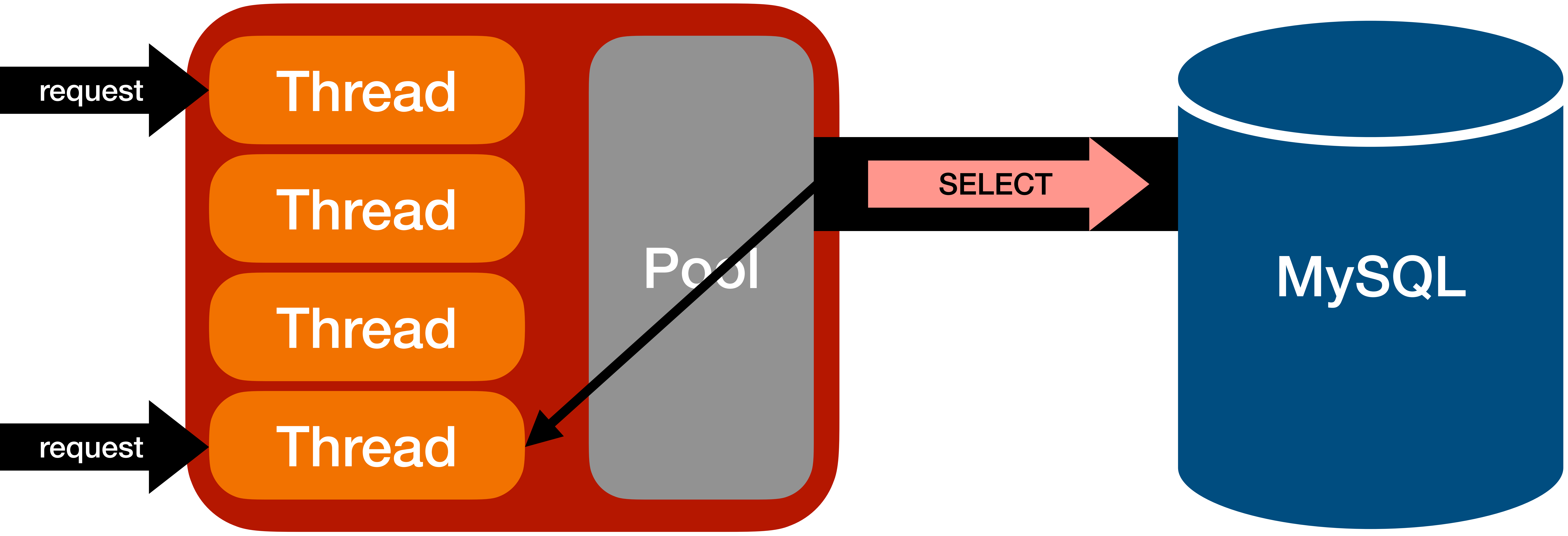
Connection Pinning



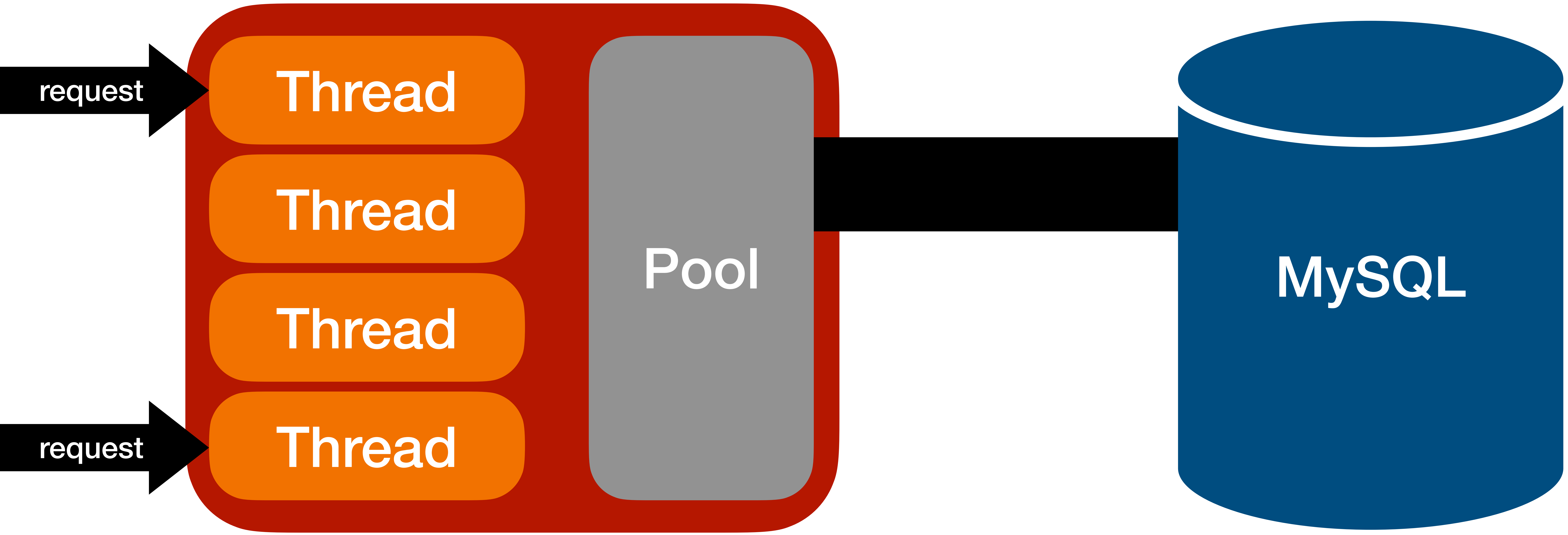
Connection Pinning



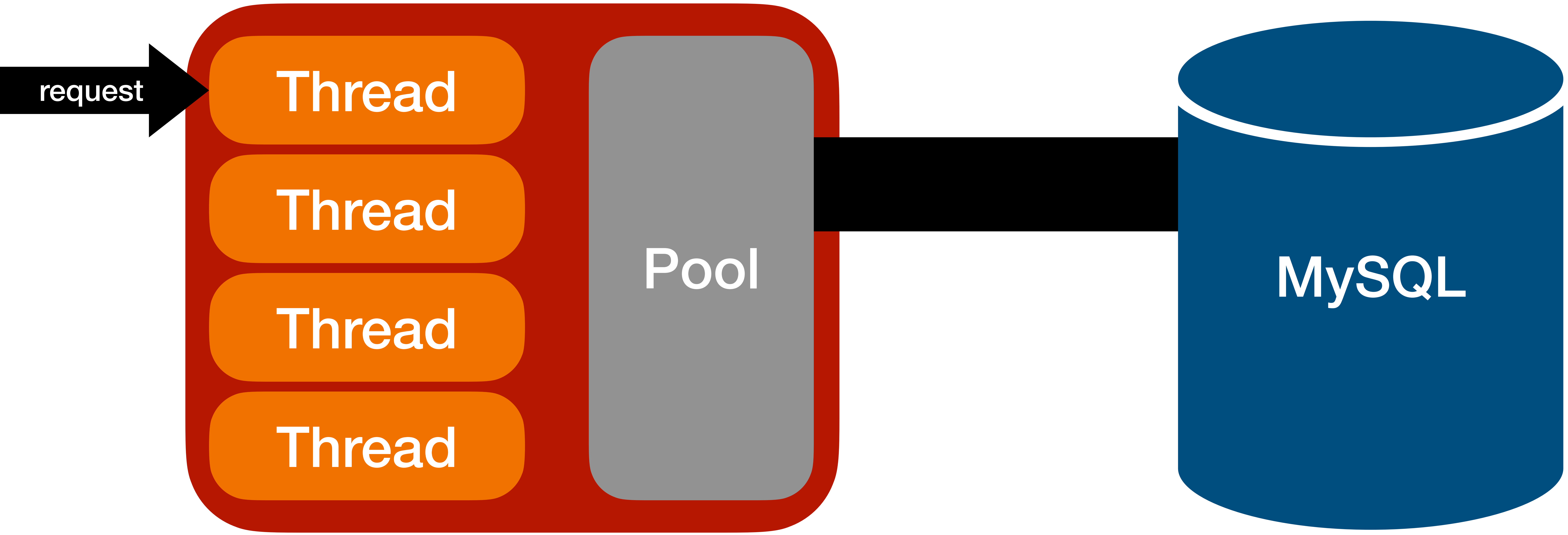
Connection Pinning



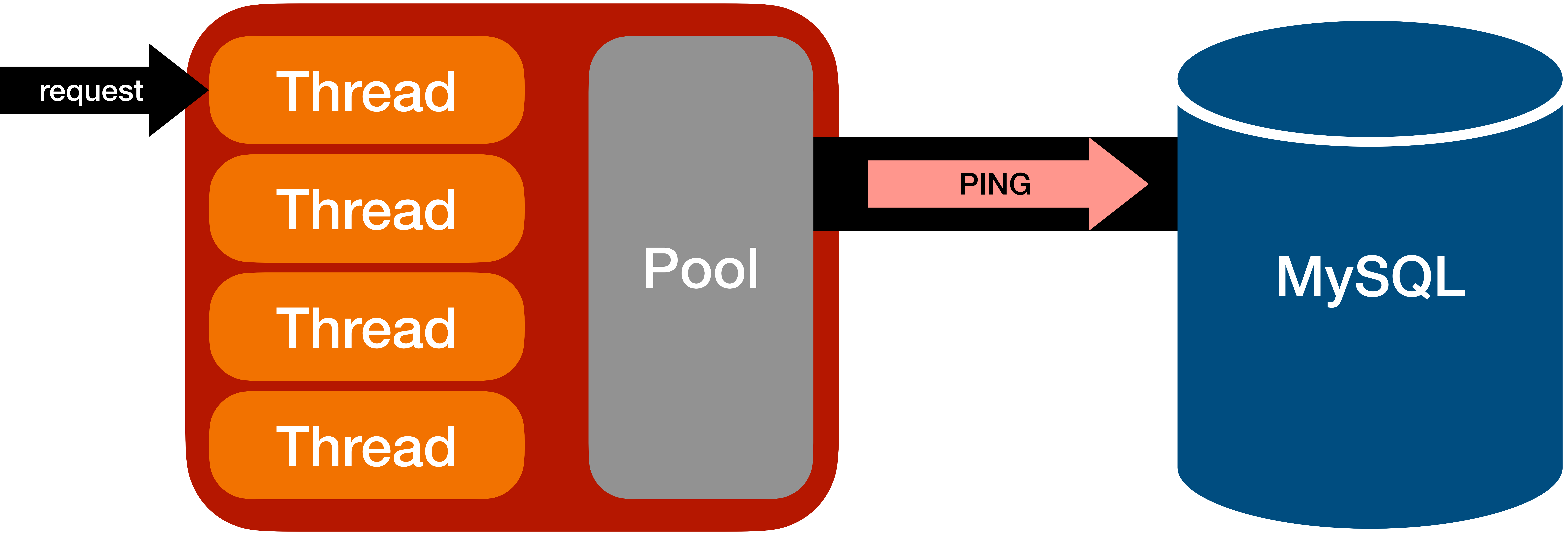
Connection Pinning



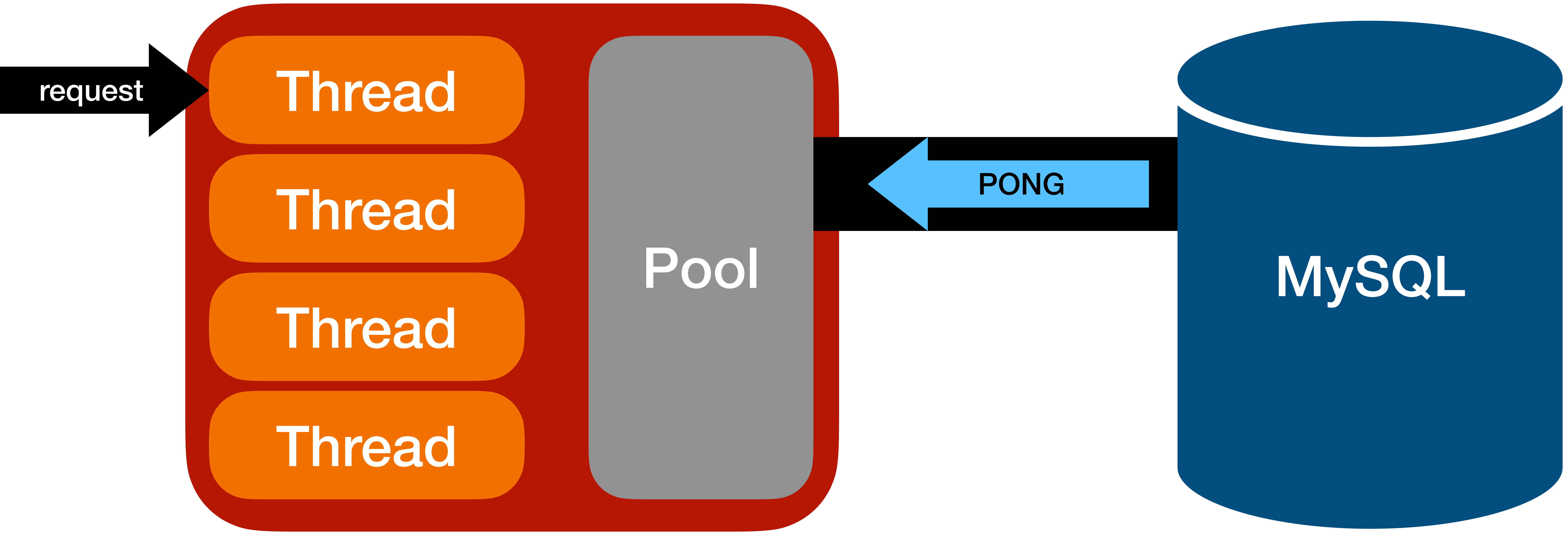
Connection Pinning



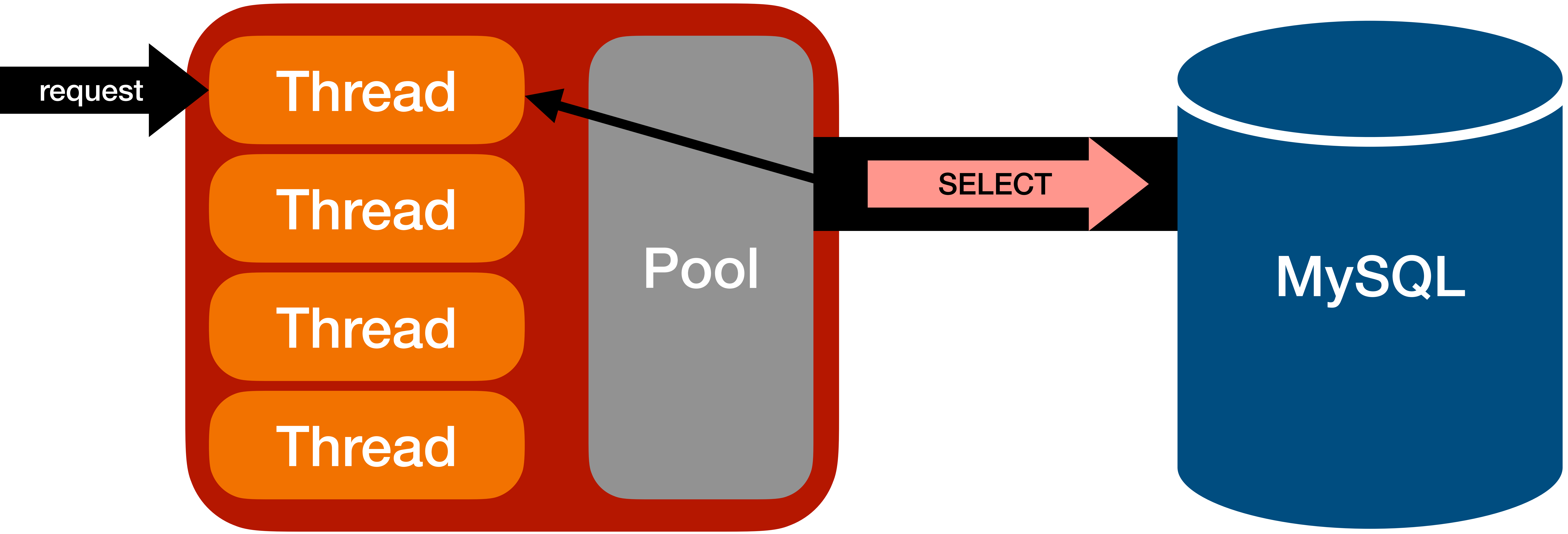
Connection Pinning

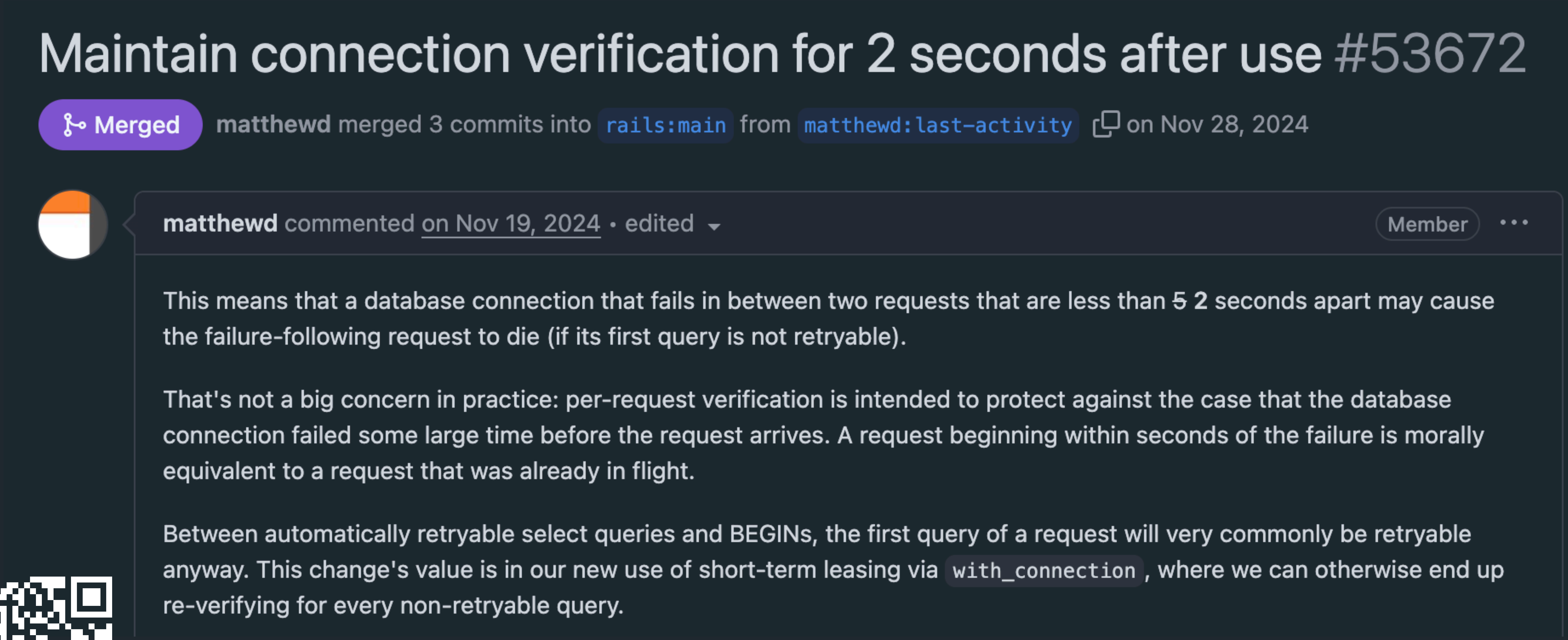


Connection Pinning



Connection Pinning





Maintain connection verification for 2 seconds after use #53672

Merged matthewd merged 3 commits into `rails:main` from `matthewd:last-activity` on Nov 28, 2024

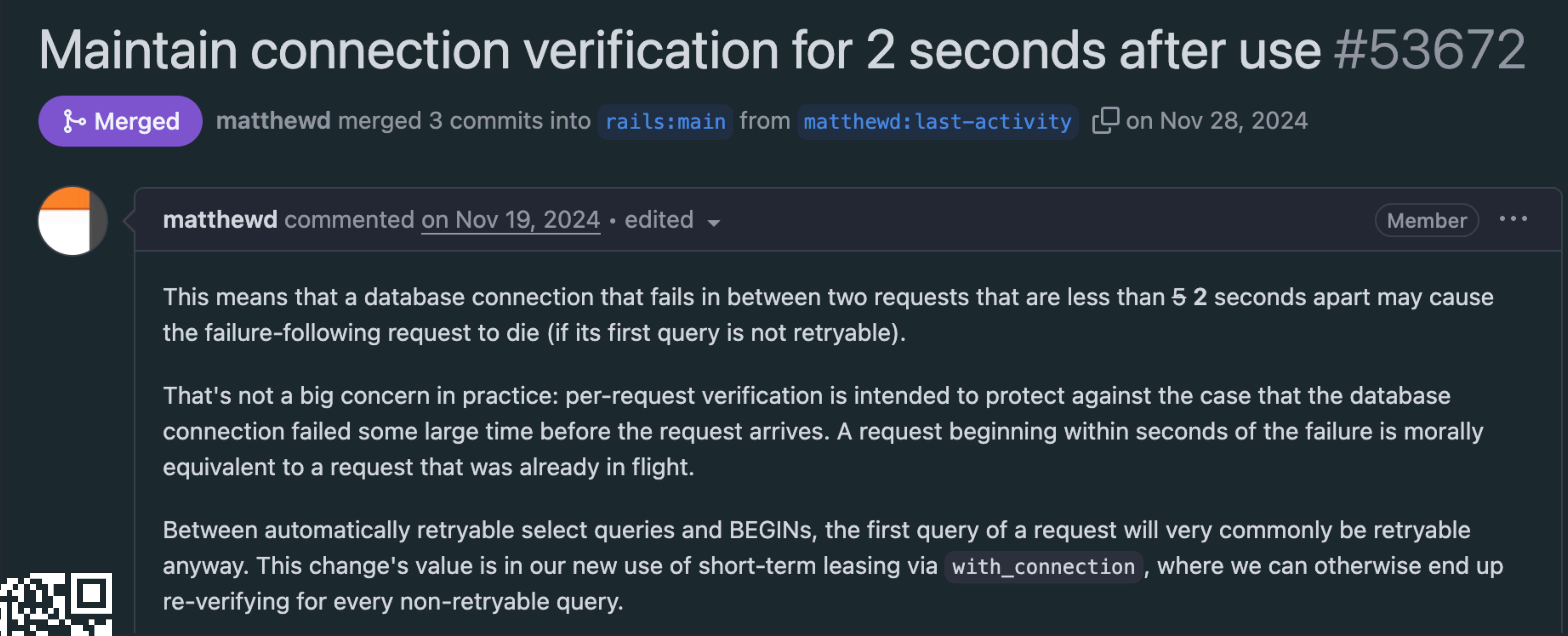
matthewd commented on Nov 19, 2024 • edited Member

This means that a database connection that fails in between two requests that are less than ~~5~~ **2** seconds apart may cause the failure-following request to die (if its first query is not retryable).

That's not a big concern in practice: per-request verification is intended to protect against the case that the database connection failed some large time before the request arrives. A request beginning within seconds of the failure is morally equivalent to a request that was already in flight.

Between automatically retryable select queries and BEGINs, the first query of a request will very commonly be retryable anyway. This change's value is in our new use of short-term leasing via `with_connection`, where we can otherwise end up re-verifying for every non-retryable query.

Included in Rails 8.0.2, 7-2-stable



Maintain connection verification for 2 seconds after use #53672

Merged matthewd merged 3 commits into `rails:main` from `matthewd:last-activity` on Nov 28, 2024

matthewd commented on Nov 19, 2024 • edited Member

This means that a database connection that fails in between two requests that are less than ~~5~~ 2 seconds apart may cause the failure-following request to die (if its first query is not retryable).

That's not a big concern in practice: per-request verification is intended to protect against the case that the database connection failed some large time before the request arrives. A request beginning within seconds of the failure is morally equivalent to a request that was already in flight.

Between automatically retryable select queries and BEGINs, the first query of a request will very commonly be retryable anyway. This change's value is in our new use of short-term leasing via `with_connection`, where we can otherwise end up re-verifying for every non-retryable query.

Connection Pinning

The Summary

- Connection pinned to Thread/Fiber, 1 Verification

Rails 7.2

- Granular checkouts

Rails 8.0.2 / 7-2-stable

- Time based verification

Connection Pinning

The Summary

- Connection pinned to Thread/Fiber, 1 Verification

Rails 7.2

- Granular checkouts

Rails 8.0.2 / 7-2-stable

- Time based verification

Query Retry-ability

“*Idempotence* is the property of certain operations… whereby they can be applied multiple times without changing the result.”

Wikipedia

```sql
SELECT * FROM users WHERE id = 1;
```

```
User.find(1)
```

```
Arel::Node::Select
  projections: [
    Arel::Attributes::Attribute
      relation: Arel::Table(“users”)
      name: “*”
  ]
  source: Arel::Table(“users”)
  wheres: [
    Arel::Nodes::Equality
      left: Arel::Attributes::Attribute
        relation: Arel::Table(“users”)
        name: “id”
      right: ActiveRecord::QueryAttribute(1)
  ]
```

```
Arel::Node::Select
  projections: [
    Arel::Attributes::Attribute
      relation: Arel::Table(“users”)
      name: “*”
  ]
  source: Arel::Table(“users”)
  wheres: [
    Arel::Nodes::Equality
      left: Arel::Attributes::Attribute
        relation: Arel::Table(“users”)
        name: “id”
      right: ActiveRecord::QueryAttribute(1)
  ]
```

```
Arel::Node::Select
  projections: [
    Arel::Attributes::Attribute
      relation: Arel::Table("users")
      name: "*"
  ]
  source: Arel::Table("users")
  wheres: [
    Arel::Nodes::Equality
      left: Arel::Attributes::Attribute
        relation: Arel::Table("users")
        name: "id"
      right: ActiveRecord::QueryAttribute(1)
  ]
```

SELECT

```
Arel::Node::Select
  projections: [
    Arel::Attributes::Attribute
      relation: Arel::Table(“users”)
      name: “*”
  ]
  source: Arel::Table(“users”)
  wheres: [
    Arel::Nodes::Equality
      left: Arel::Attributes::Attribute
        relation: Arel::Table(“users”)
        name: “id”
      right: ActiveRecord::QueryAttribute(1)
  ]
```

```
SELECT “users”
```

```
Arel::Node::Select
  projections: [
    Arel::Attributes::Attribute
      relation: Arel::Table("users")
      name: "*"
  ]
  source: Arel::Table("users")
  wheres: [
    Arel::Nodes::Equality
      left: Arel::Attributes::Attribute
        relation: Arel::Table("users")
        name: "id"
      right: ActiveRecord::QueryAttribute(1)
  ]
```

```
SELECT "users".*
```

```
Arel::Node::Select
  projections: [
    Arel::Attributes::Attribute
      relation: Arel::Table(“users”)
      name: “*”
  ]
  source: Arel::Table(“users”)
  wheres: [
    Arel::Nodes::Equality
      left: Arel::Attributes::Attribute
        relation: Arel::Table(“users”)
        name: “id”
      right: ActiveRecord::QueryAttribute(1)
  ]
```

```
SELECT “users”.* FROM “users”
```

```
Arel::Node::Select
  projections: [
    Arel::Attributes::Attribute
      relation: Arel::Table(“users”)
      name: “*”
  ]
  source: Arel::Table(“users”)
  wheres: [
    Arel::Nodes::Equality
      left: Arel::Attributes::Attribute
        relation: Arel::Table(“users”)
        name: “id”
      right: ActiveRecord::QueryAttribute(1)
  ]
```

```
SELECT “users”.* FROM “users” WHERE “users”
```

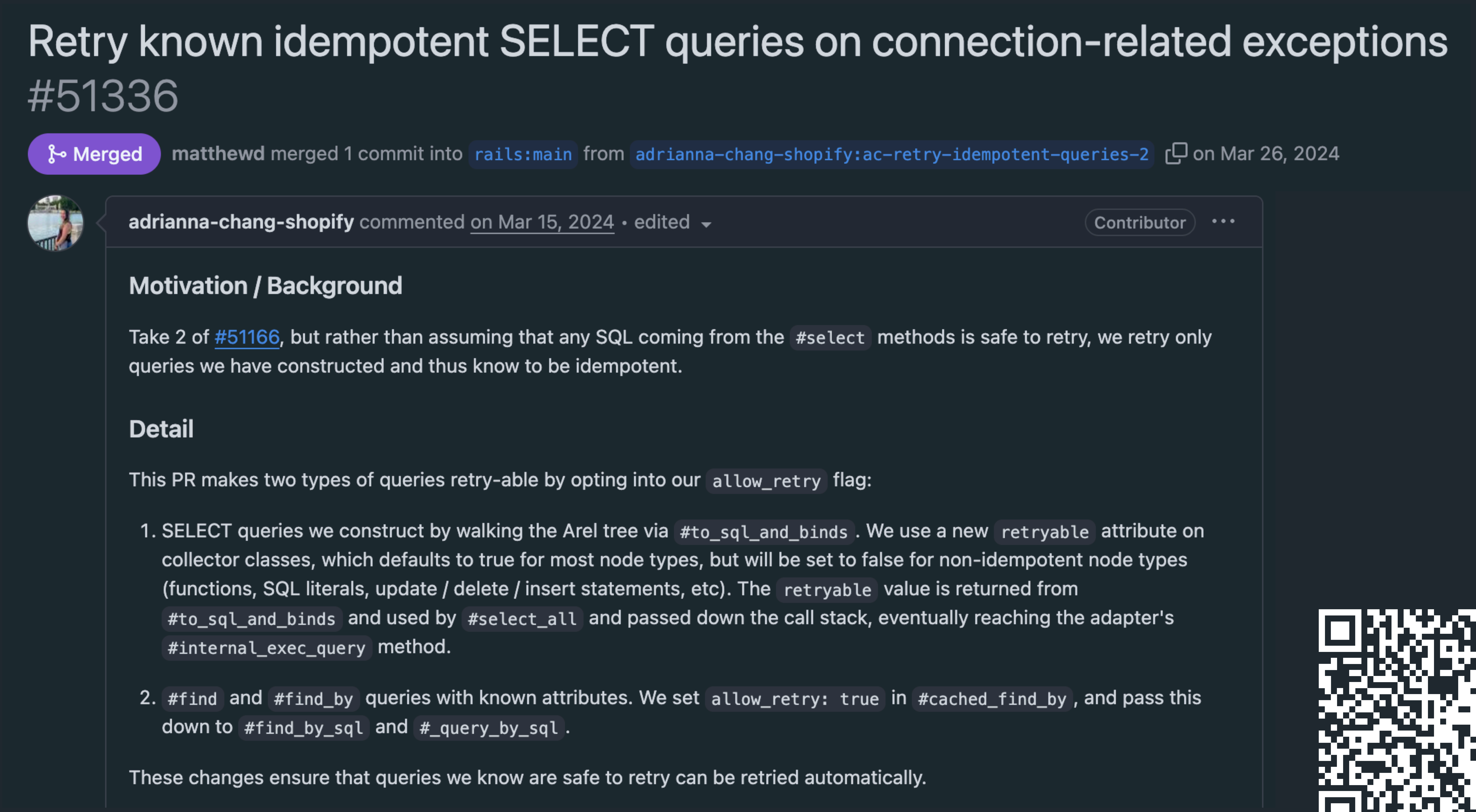
```
Arel::Node::Select
  projections: [
    Arel::Attributes::Attribute
      relation: Arel::Table("users")
      name: "*"
  ]
  source: Arel::Table("users")
  wheres: [
    Arel::Nodes::Equality
      left: Arel::Attributes::Attribute
        relation: Arel::Table("users")
        name: "id"
      right: ActiveRecord::QueryAttribute(1)
  ]
```

```
SELECT "users".* FROM "users" WHERE "users".id
```

```
Arel::Node::Select
  projections: [
    Arel::Attributes::Attribute
      relation: Arel::Table(“users”)
      name: “*”
  ]
  source: Arel::Table(“users”)
  wheres: [
    Arel::Nodes::Equality
      left: Arel::Attributes::Attribute
        relation: Arel::Table(“users”)
        name: “id”
      right: ActiveRecord::QueryAttribute(1)
  ]
```

```
SELECT “users”.* FROM “users” WHERE “users”.id =
Arel::Node::Select
  projections: [
    Arel::Attributes::Attribute
      relation: Arel::Table(“users”)
      name: “*”
  ]
  source: Arel::Table(“users”)
  wheres: [
    Arel::Nodes::Equality
      left: Arel::Attributes::Attribute
        relation: Arel::Table(“users”)
        name: “id”
      right: ActiveRecord::QueryAttribute(1)
  ]
```

```
SELECT “users”.* FROM “users” WHERE “users”.id = 1
```



Retry known idempotent SELECT queries on connection-related exceptions #51336

Merged matthewd merged 1 commit into `rails:main` from `adrianna-chang-shopify:ac-retry-idempotent-queries-2` on Mar 26, 2024

adrianna-chang-shopify commented on Mar 15, 2024 • edited Contributor

Motivation / Background

Take 2 of #51166, but rather than assuming that any SQL coming from the `#select` methods is safe to retry, we retry only queries we have constructed and thus know to be idempotent.

Detail

This PR makes two types of queries retry-able by opting into our `allow_retry` flag:

1. SELECT queries we construct by walking the Arel tree via `#to_sql_and_binds`. We use a new `retryable` attribute on collector classes, which defaults to true for most node types, but will be set to false for non-idempotent node types (functions, SQL literals, update / delete / insert statements, etc). The `retryable` value is returned from `#to_sql_and_binds` and used by `#select_all` and passed down the call stack, eventually reaching the adapter's `#internal_exec_query` method.
2. `#find` and `#find_by` queries with known attributes. We set `allow_retry: true` in `#cached_find_by`, and pass this down to `#find_by_sql` and `#_query_by_sql`.

These changes ensure that queries we know are safe to retry can be retried automatically.

```
User.where(“modify()”)
```

```
Arel::Node::Select
  projections: [
    Arel::Attributes::Attribute
      relation: Arel::Table(“users”)
      name: “*”
  ]
  source: Arel::Table(“users”)
  wheres: [
    Arel::Nodes::SqlLiteral(“modify()”)
  ]
```

Retryable

```
Arel::Node::Select
  projections: [
    Arel::Attributes::Attribute
      relation: Arel::Table(“users”)
      name: “*”
  ]
  source: Arel::Table(“users”)
  wheres: [
    Arel::Nodes::SqlLiteral(“modify()”)
  ]
```

Retryable

```
Arel::Node::Select
  projections: [
    Arel::Attributes::Attribute
      relation: Arel::Table(“users”)
      name: “*”
  ]
  source: Arel::Table(“users”)
  wheres: [
    Arel::Nodes::SqlLiteral(“modify()”)
  ]
```

SELECT

Retryable

```
Arel::Node::Select
  projections: [
    Arel::Attributes::Attribute
      relation: Arel::Table(“users”)
      name: “*”
  ]
  source: Arel::Table(“users”)
  wheres: [
    Arel::Nodes::SqlLiteral(“modify()”)
  ]
```

```
SELECT “users”
```

Retryable

```
Arel::Node::Select
  projections: [
    Arel::Attributes::Attribute
      relation: Arel::Table("users")
      name: "*"
  ]
  source: Arel::Table("users")
  wheres: [
    Arel::Nodes::SqlLiteral("modify()")
  ]
```

```
SELECT "users".*
```

Retryable

```
Arel::Node::Select
  projections: [
    Arel::Attributes::Attribute
      relation: Arel::Table(“users”)
      name: “*”
  ]
  source: Arel::Table(“users”)
  wheres: [
    Arel::Nodes::SqlLiteral(“modify()”)
  ]
```

```
SELECT “users”.* FROM “users”
```

NOT Retryable

```
Arel::Node::Select
  projections: [
    Arel::Attributes::Attribute
      relation: Arel::Table(“users”)
      name: “*”
  ]
  source: Arel::Table(“users”)
  wheres: [
    Arel::Nodes::SqlLiteral(“modify()”)
  ]
```

```
SELECT “users”.* FROM “users” WHERE modify()
```

Query Retry-ability

PING
SELECT
BEGIN
INSERT
COMMIT
SELECT
SELECT
CHECKOUT
CHECKIN

Query Retry-ability

SELECT
BEGIN
INSERT
COMMIT
SELECT
SELECT
CHECKOUT
CHECKIN

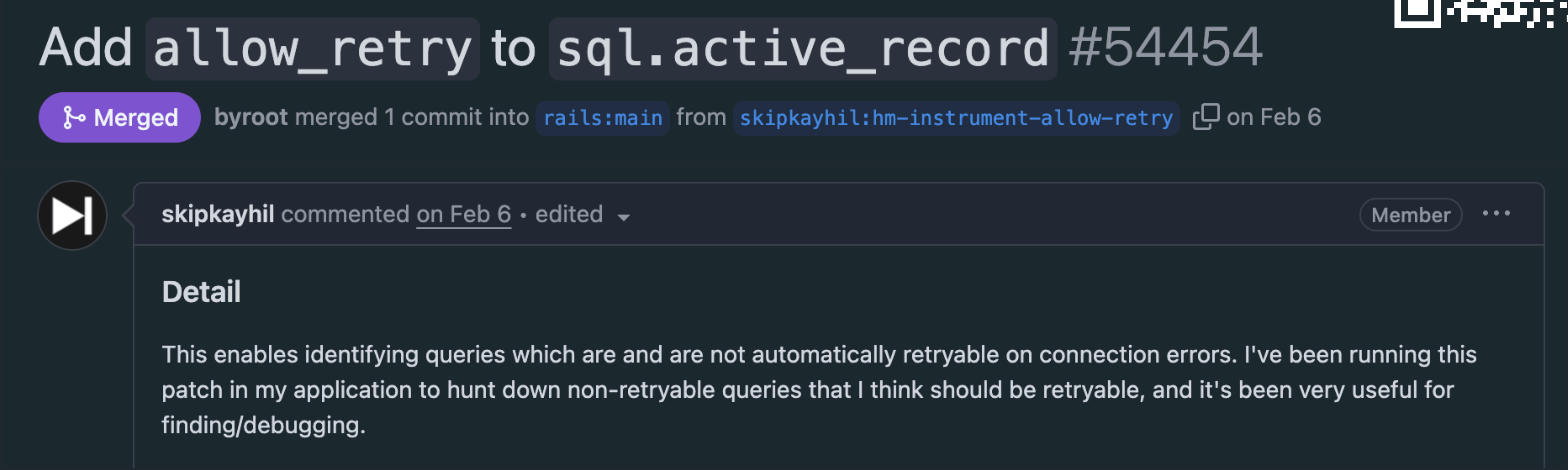
Resilient By Default

Verification Timeout		8.0
Granular Checkouts	Automatically Retry SELECTs	7.2
Deferred Connection Verification		7.1

Active Record 8 is the most resilient version of Active Record, *ever.*

Active Record 8.1 is the most resilient version of Active Record, *ever.*

```
Trilogy::EOFError:
trilogy_query_send:
TRILOGY_CLOSED_CONNECTION
```



Add `allow_retry` to `sql.active_record` #54454

Merged byroot merged 1 commit into `rails:main` from `skipkayhil:hm-instrument-allow-retry` on Feb 6

skipkayhil commented on Feb 6 • edited Member

Detail

This enables identifying queries which are and are not automatically retryable on connection errors. I've been running this patch in my application to hunt down non-retryable queries that I think should be retryable, and it's been very useful for finding/debugging.

```
class NonRetryableQueries < ActiveSupport::LogSubscriber
  def sql(event)
    return if event.payload[:allow_retry]

    sql   = payload[:sql]

    debug “FIXME: #{sql}”
  end

  attach_to :active_record
end
```

```
SELECT column_name
FROM information_schema.statistics
WHERE index_name = 'PRIMARY'
  AND table_schema = database()
  AND table_name = `users`
ORDER BY seq_in_index
```

```
User.exists?
# => SELECT 1 AS one FROM “users” LIMIT 1
```

```
User.exists?
# => SELECT 1 AS one FROM “users” LIMIT 1
```

⚠️ Warning: Sharp Knife ⚠️

```
Arel.sql(“1 AS ONE”)
Arel.sql(“1 AS ONE”, retryable: true)
```

```
User.where(
  Arel.sql(“type = ‘admin’”, retryable: true)
)
```

Resilient By Default

	Retry Observability	EVEN MORE Retries	8.1
Verification Timeout			8.0
Granular Checkouts	Automatically Retry SELECTs		7.2
Deferred Connection Verification			7.1

Active Record 8.1 is the most resilient version of Active Record, *ever.*