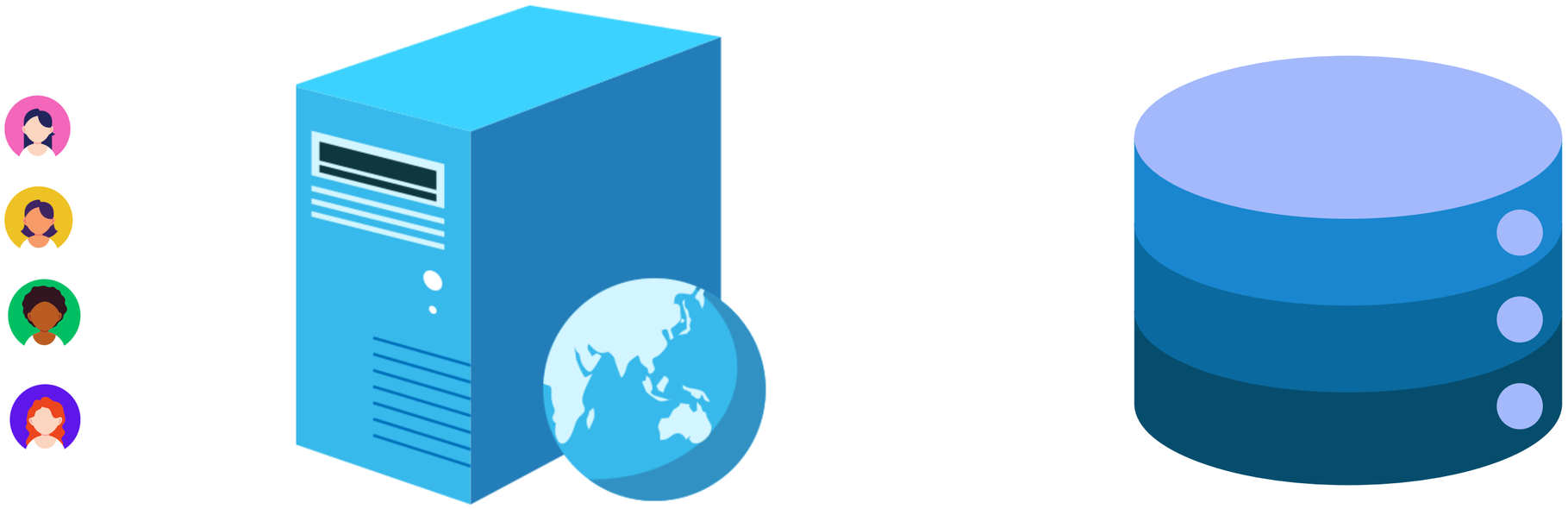
Database Connections Models



Application Server

Database

Communication Models Between Application Server and Database :

1- Connection Per User Request

2- Connection Pools

User

Application Server

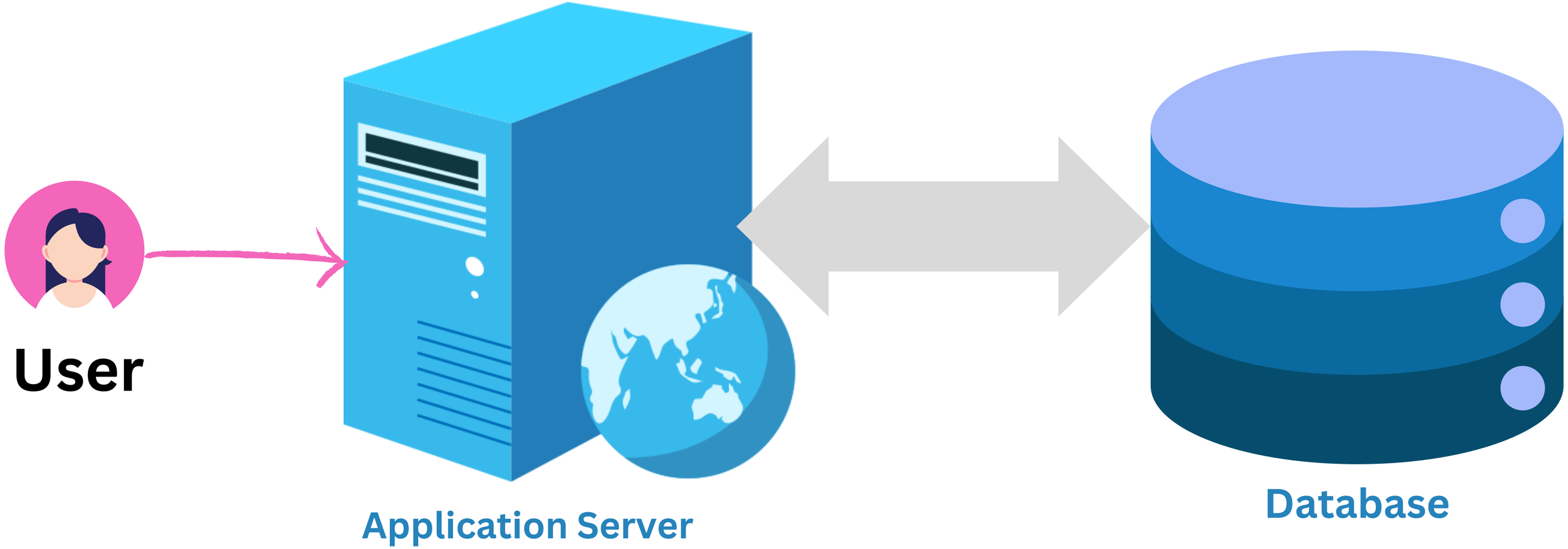
Database

Connection Per User Request Model

Connection Per User Request Model

Step 1 :

User Send API Request To Application Server To Get Data from Database



Connection Per User Request Model

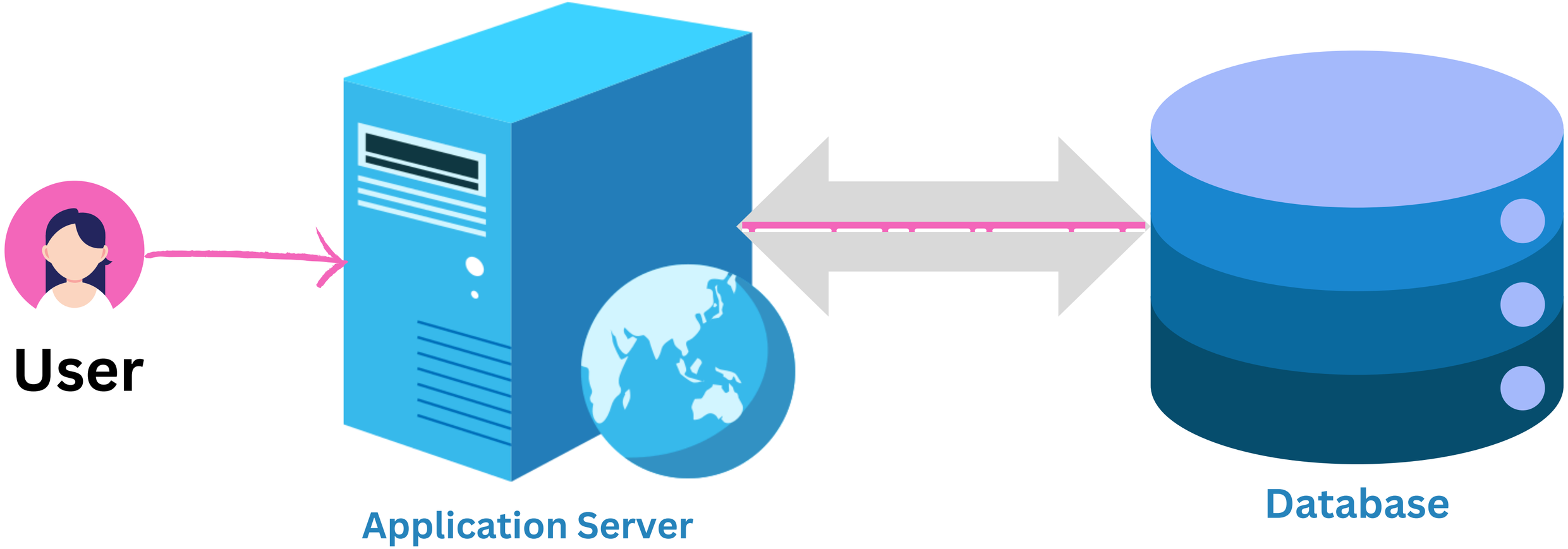
Step 2 :

Application Server open new Physical TCP Connection

Then Transfer Data From Database

Then Close The Connection

The Connection is Used Only With Single API Request



Connection Per User Request Model

Step 2 :

Application Server open new Physical TCP Connection

Then Transfer Data From Database

Then Close The Connection

The Connection is Used Only With Single API Request

User

Application Server

Database

Connection Per User Request Model

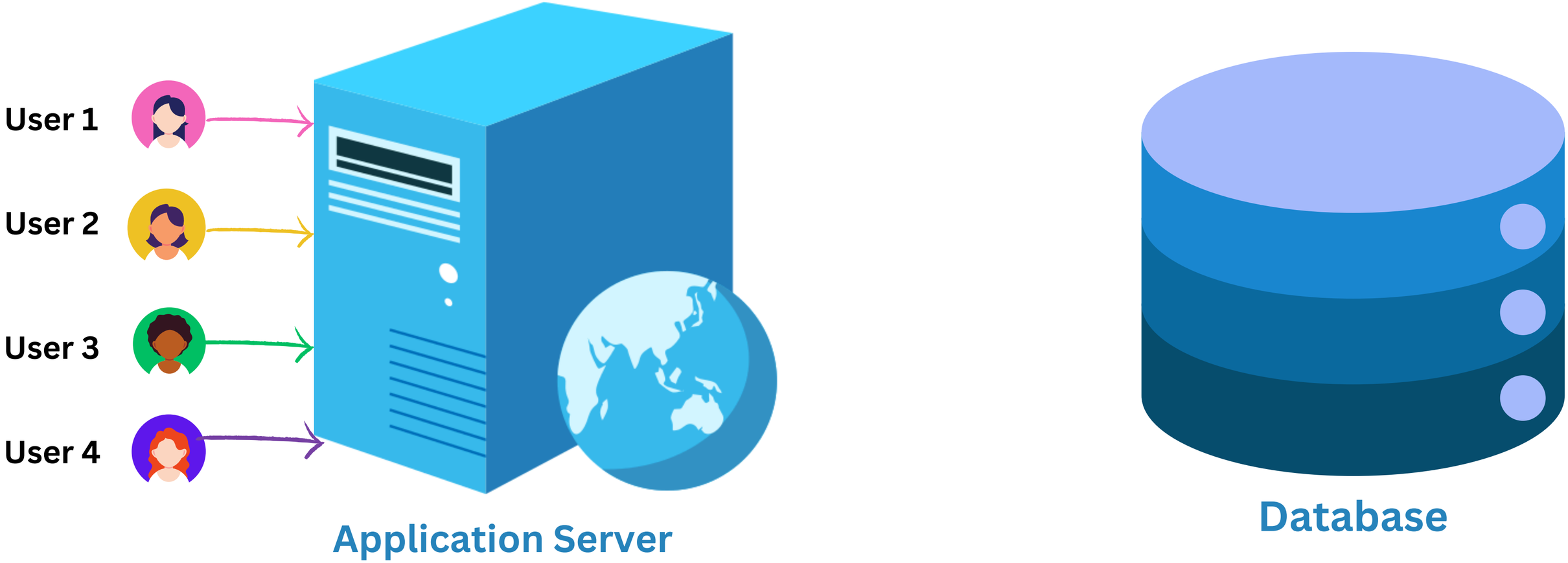
Step 2 :

Application Server open new Physical TCP Connection

Then Transfer Data From Database

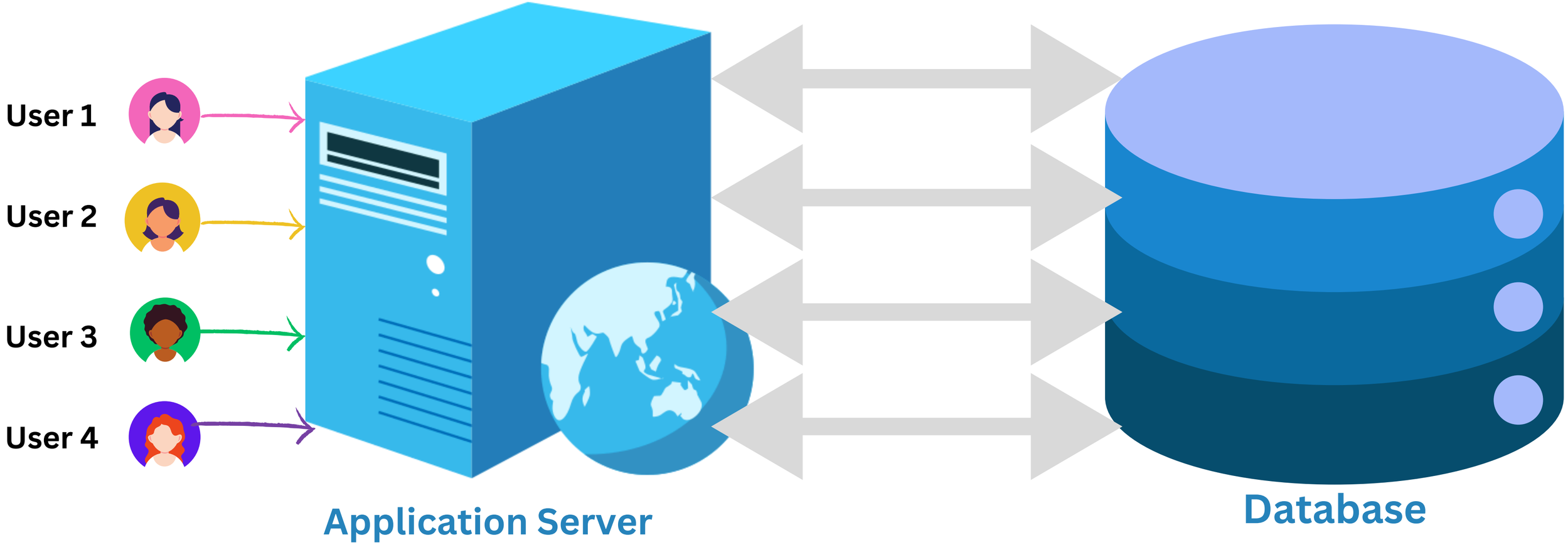
Then Close The Connection

The Connection is Used Only With Single API Request



Connection Per User Request Model

Example : 4 Concurrent Users

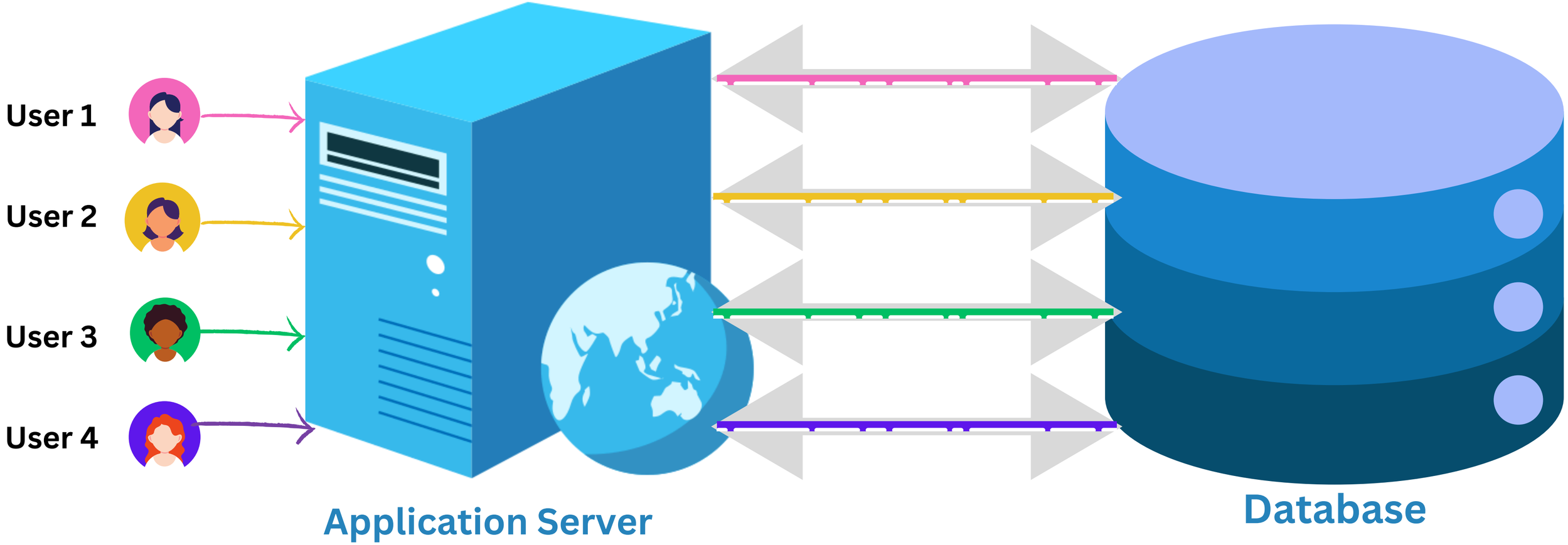


Connection Per User Request Model

Every API Request open new Connection
Then Every Connection is Closed after Transfer Data is Done Fot its Request

number of concurrent users = number of connections

4 Users = 4 Connections



Connection Per User Request Model

Every API Request open new Connection
Then Every Connection is Closed after Transfer Data is Done Fot its Request

number of concurrent users = number of connections

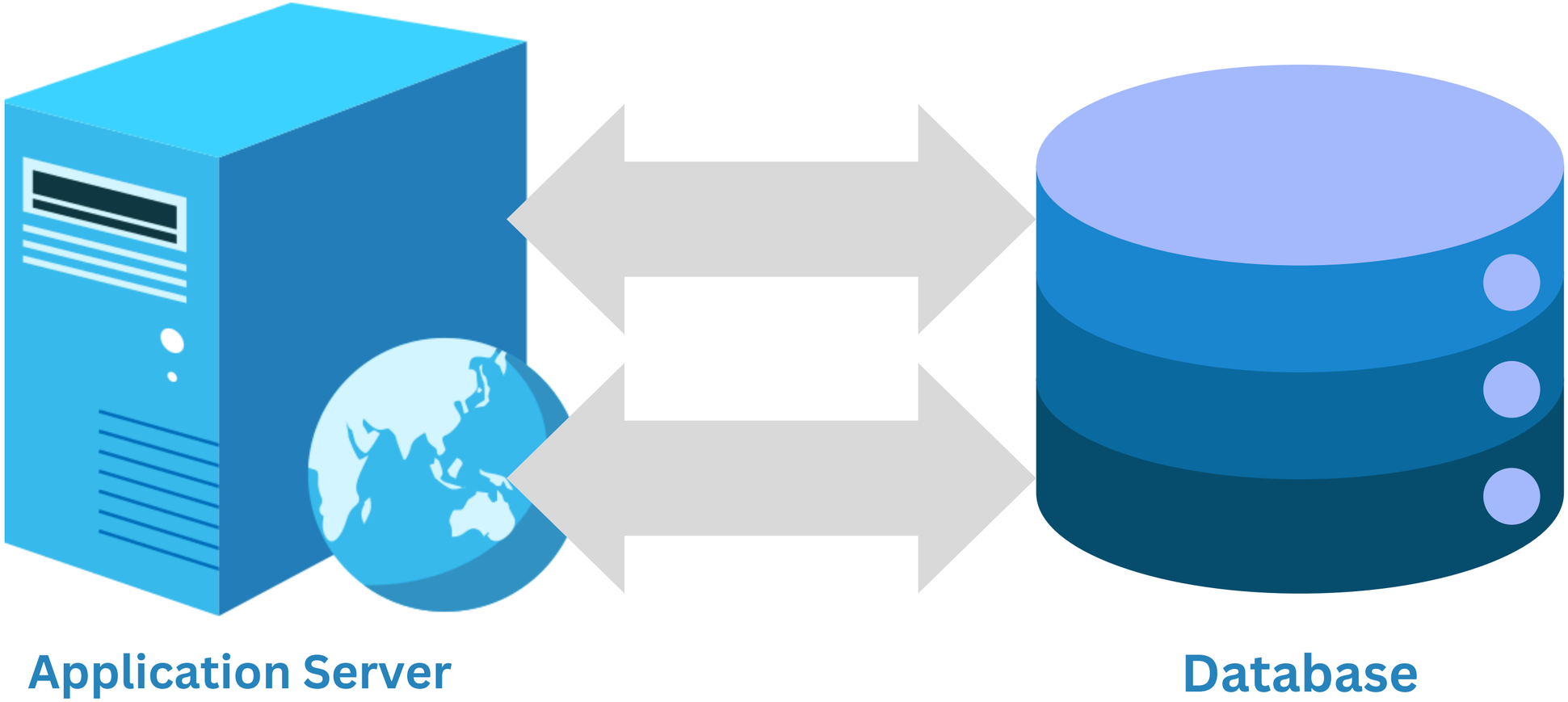
4 Users = 4 Connections

Connection Per User Request Model

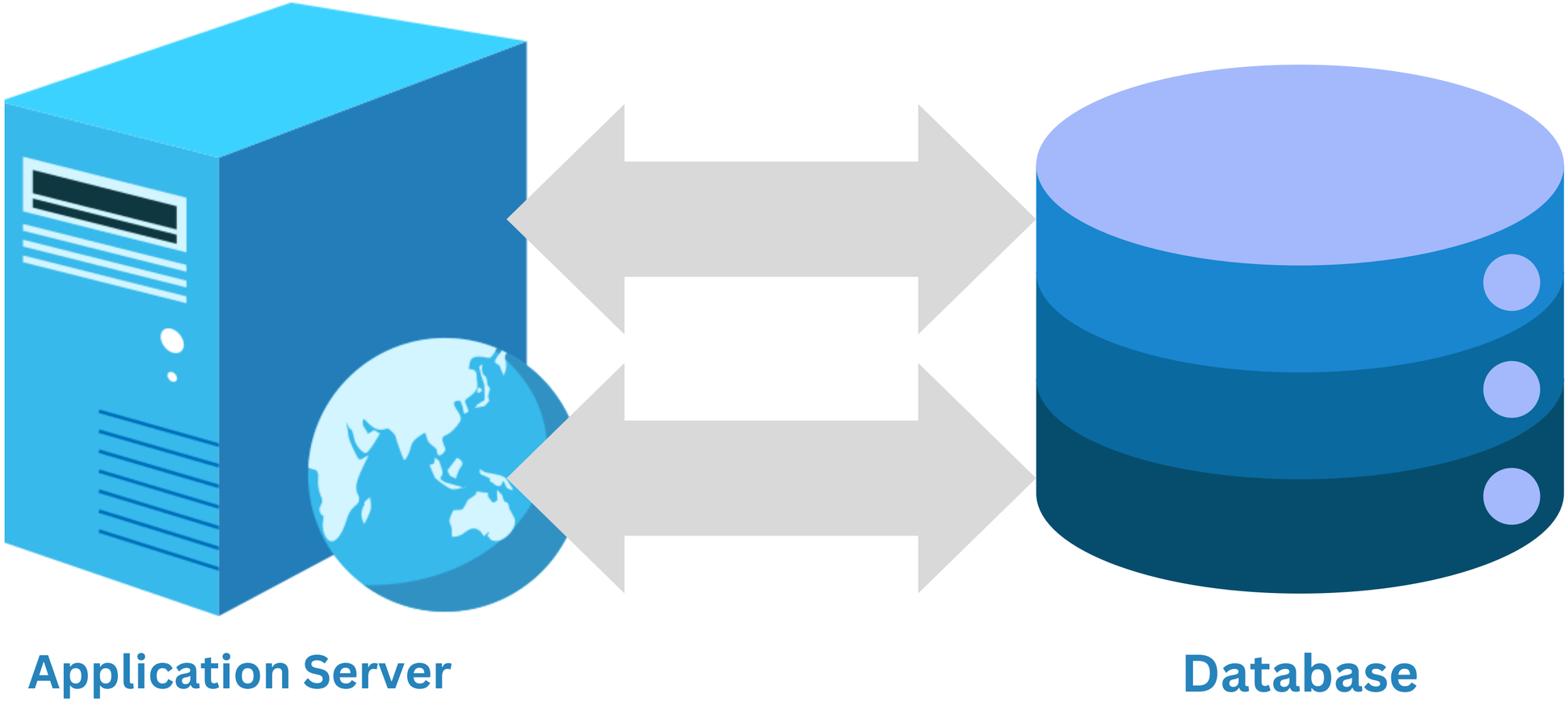
Every API Request open new Connection
Then Every Connection is Closed after Transfer Data is Done Fot its Request

number of concurrent users = number of connections

4 Users = 4 Connections



Connection Pools Model



Connection Pools Model

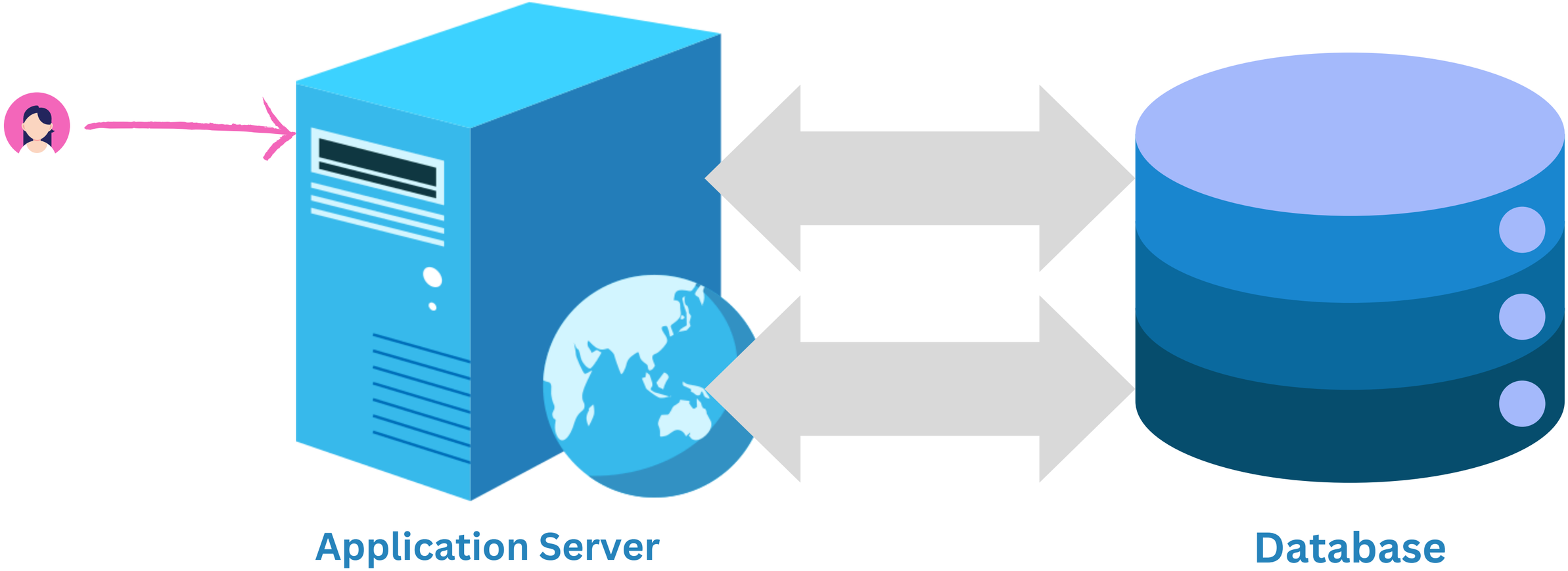
step 1 :

There are already a set of opening connections

that exist and always open between application server and database

usually this connections are created when application server start-up

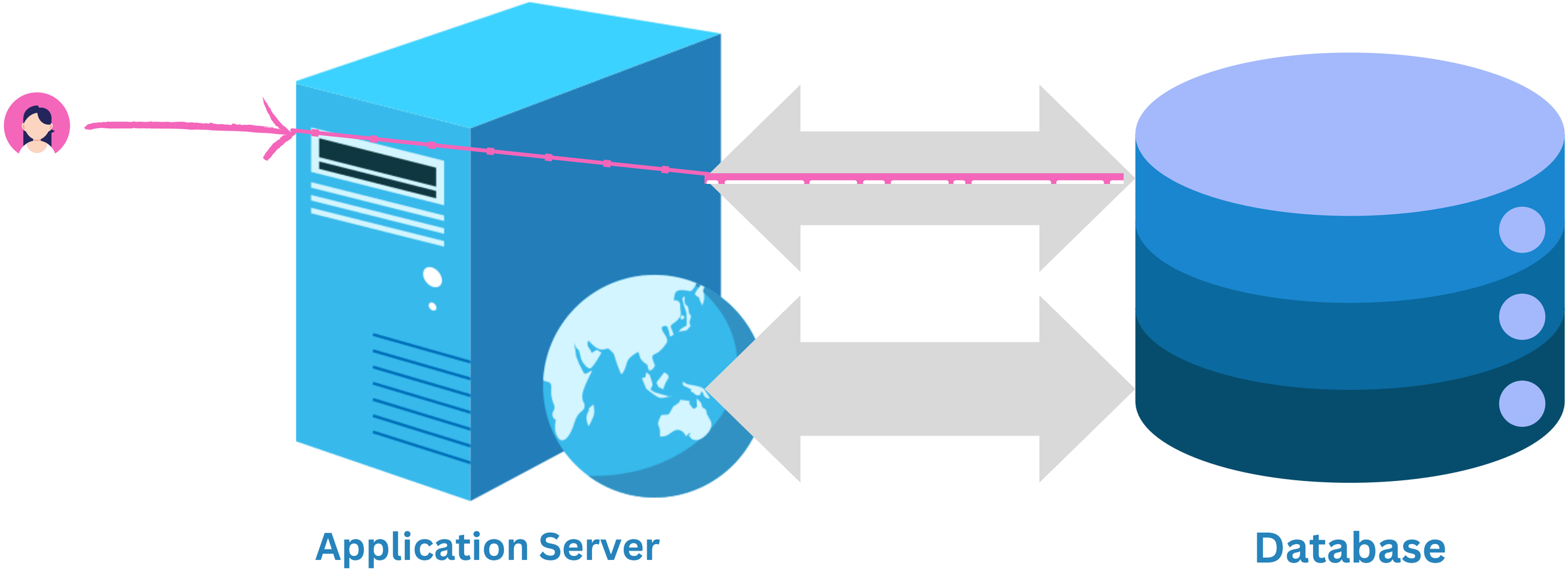
we will explain min pool size and max pool size in details in next sections



Connection Pools Model

step 2 :

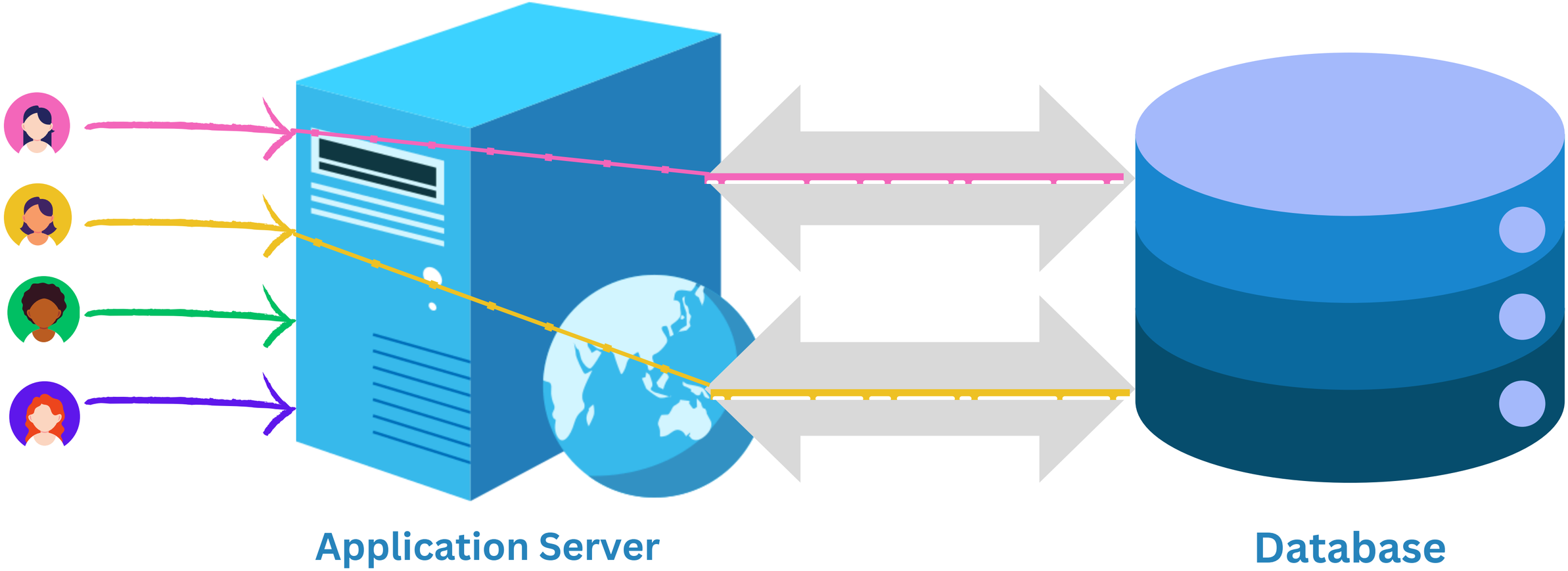
user sent request to appplication server get data from database



Connection Pools Model

step 3 :

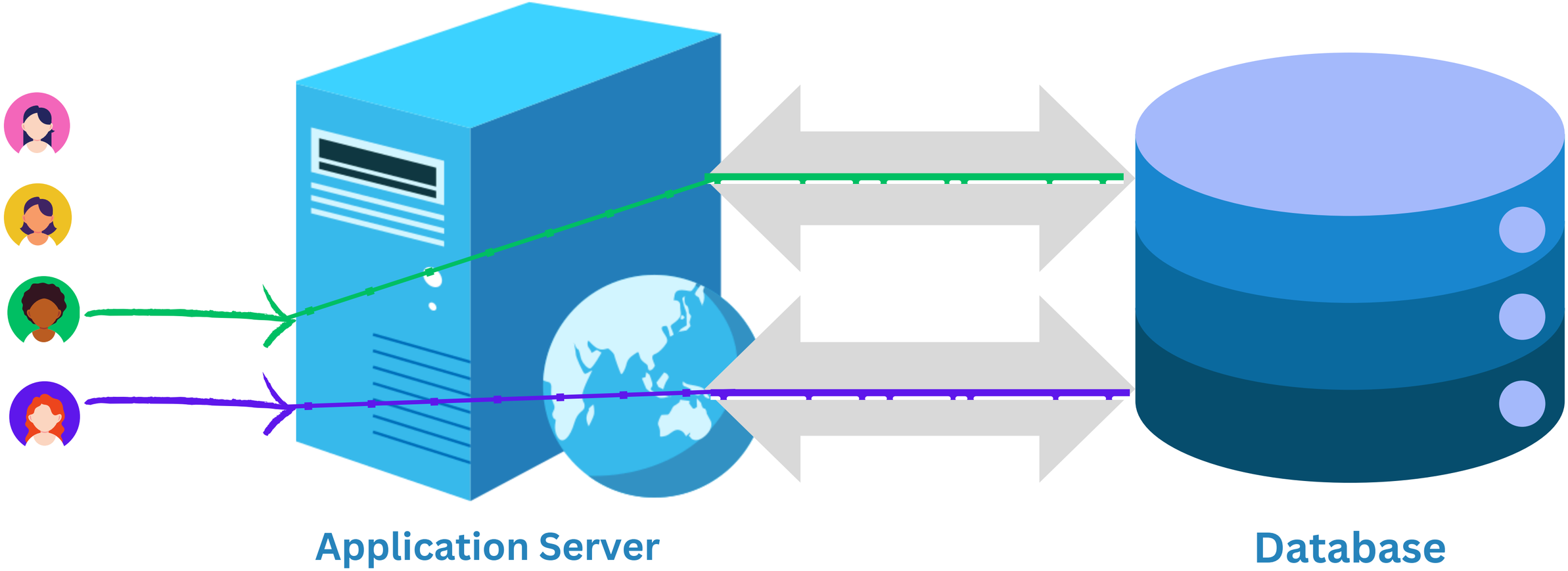
application server will reuse one of the available opening connection



Connection Pools Model

step 3 :

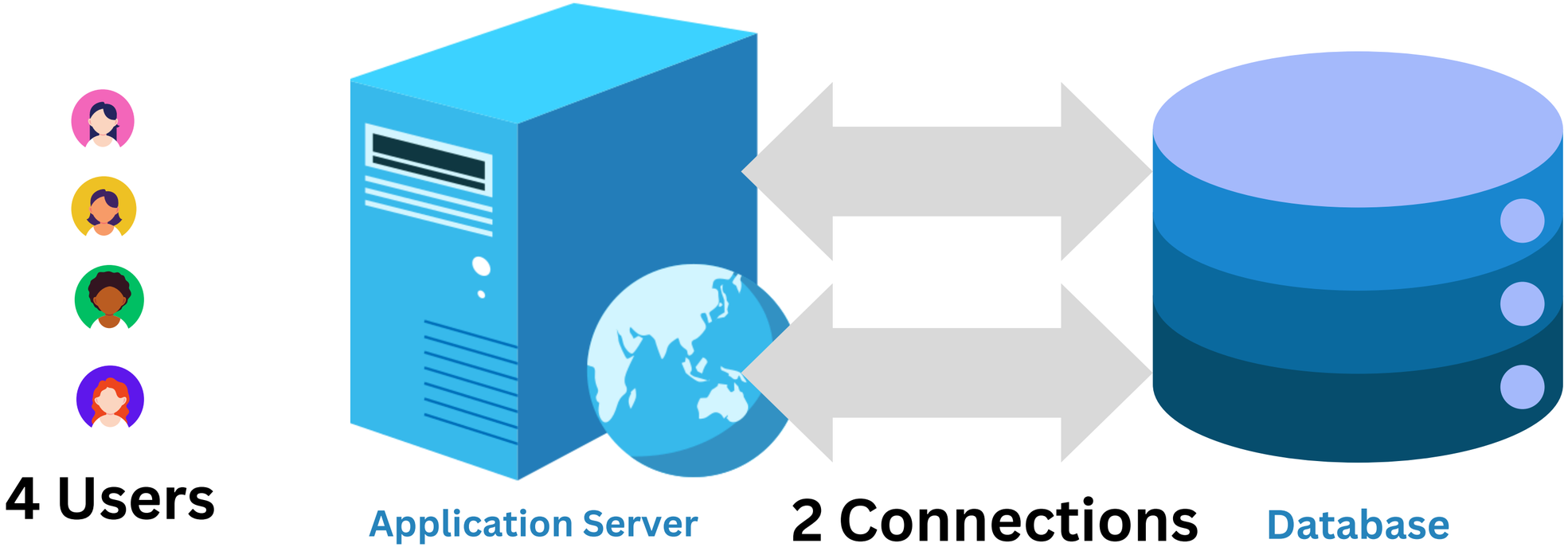
application server will reuse one of the available opening connection



Connection Pools Model

step 3 :

application server will reuse one of the available opening connection



Connection Pools Model

number of concurrent users > number of connections

in connection per user request model :

Opening new physical tcp connection for every api request is very time/memory consuming.

but in connections pools model :

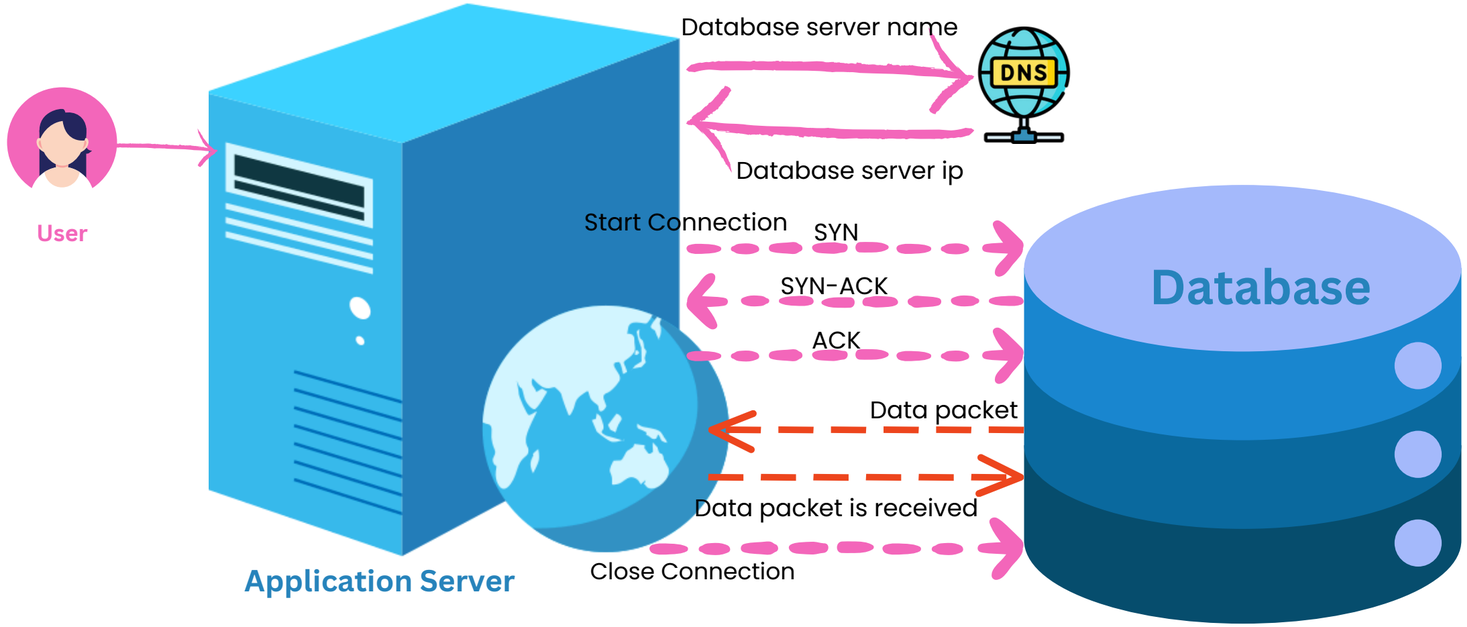
we keep the connections opened and reuse them on the future api requests

Connection Per User Request

When application server open new physical tcp connection
to get some data from database ,
this operation is not done with single netwok round trip ,
but in reality need more than netwok round trips

Connection Per User Request

When application server open new physical tcp connection
to get some data from database ,
this operation is not done with single netwok round trip ,
but in reality need more than netwok round trips

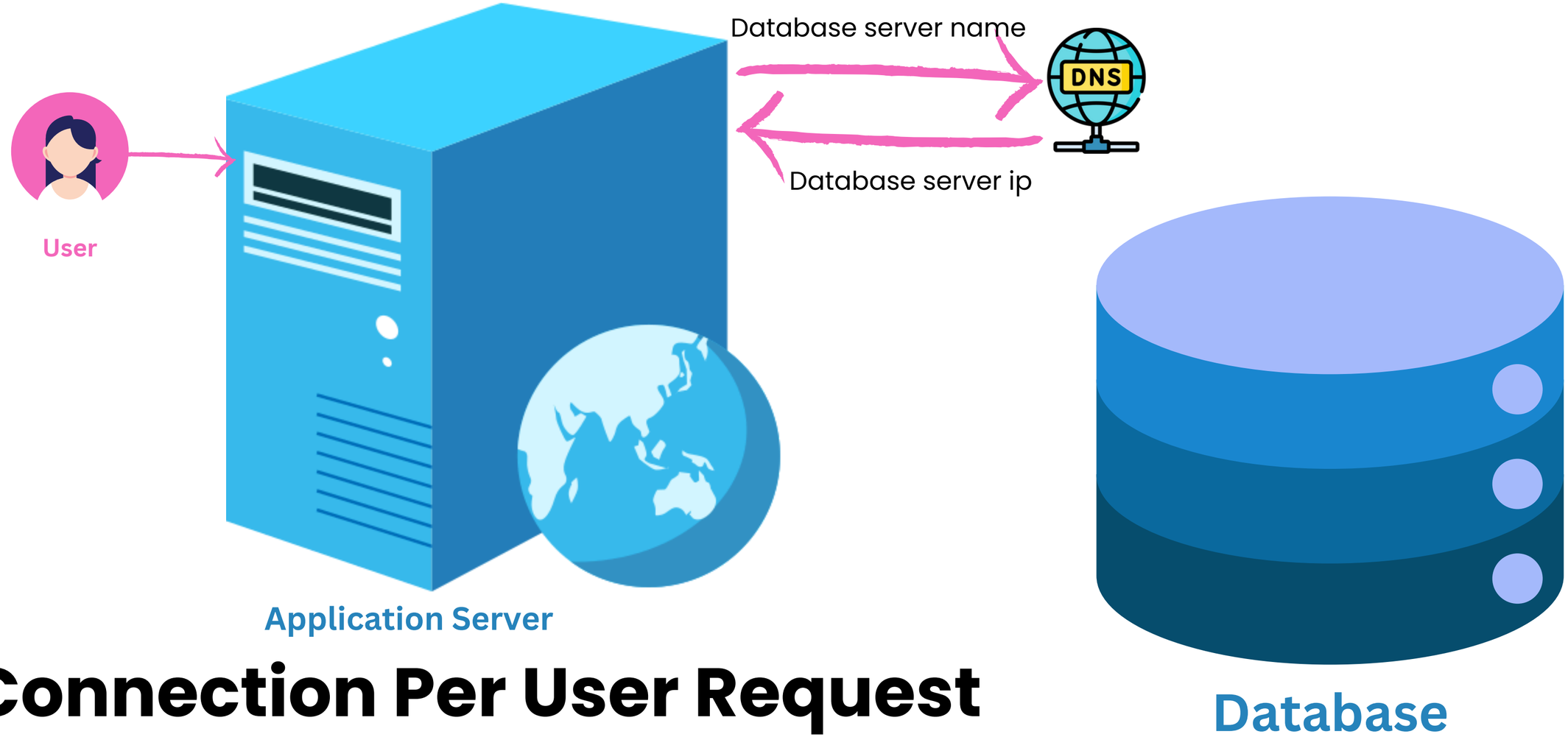


Connection Per User Request

New Connection = Apply All TCP Protocol Steps From Scratch

Main Netwok Round Trips (we will explain main steps only for simplicity):

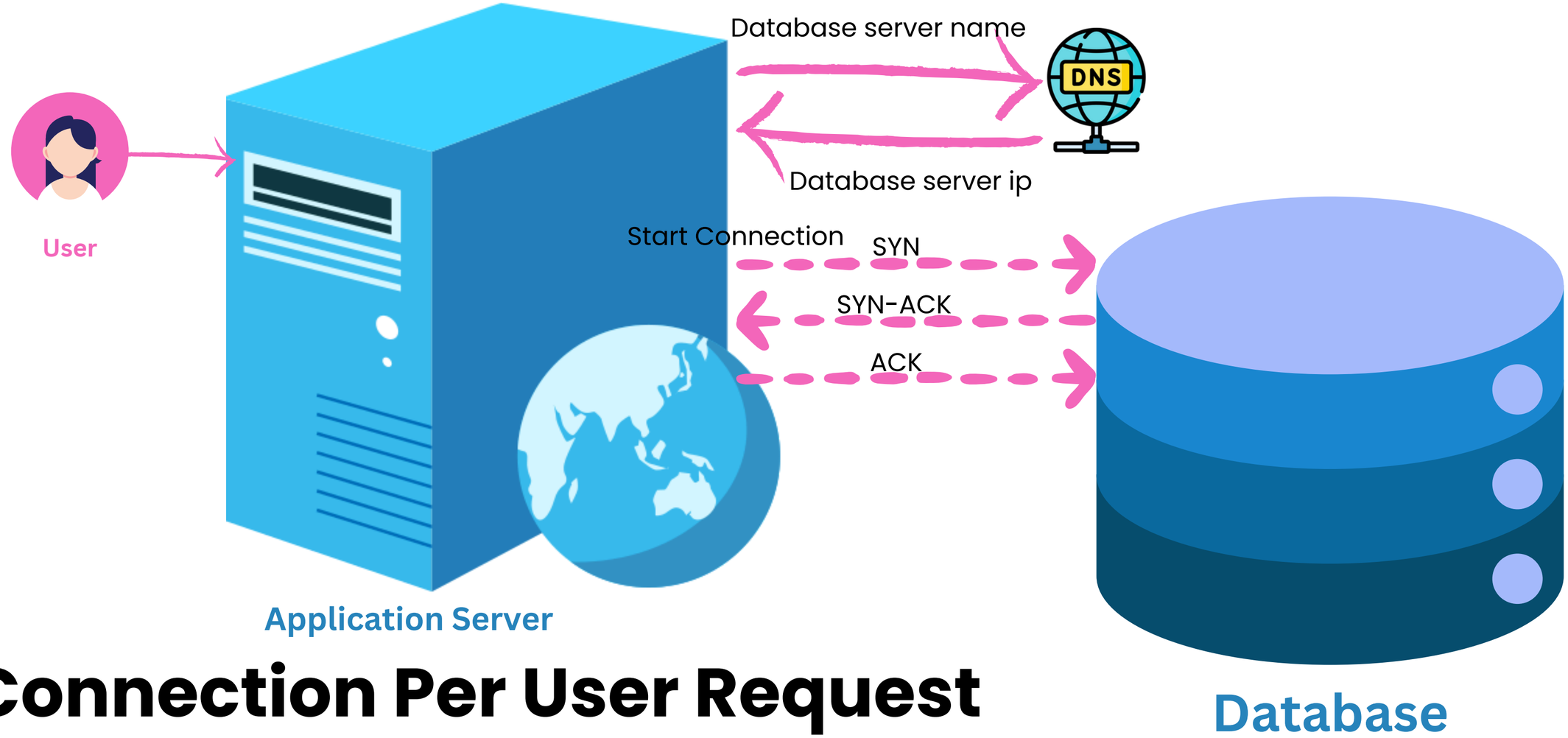
1- DNS Resolution
2- TCP Handshke and Authentication
3- Transfer Data
4- Close The Connection



Connection Per User Request

Main Netwok Round Trips :

1- DNS Resolution

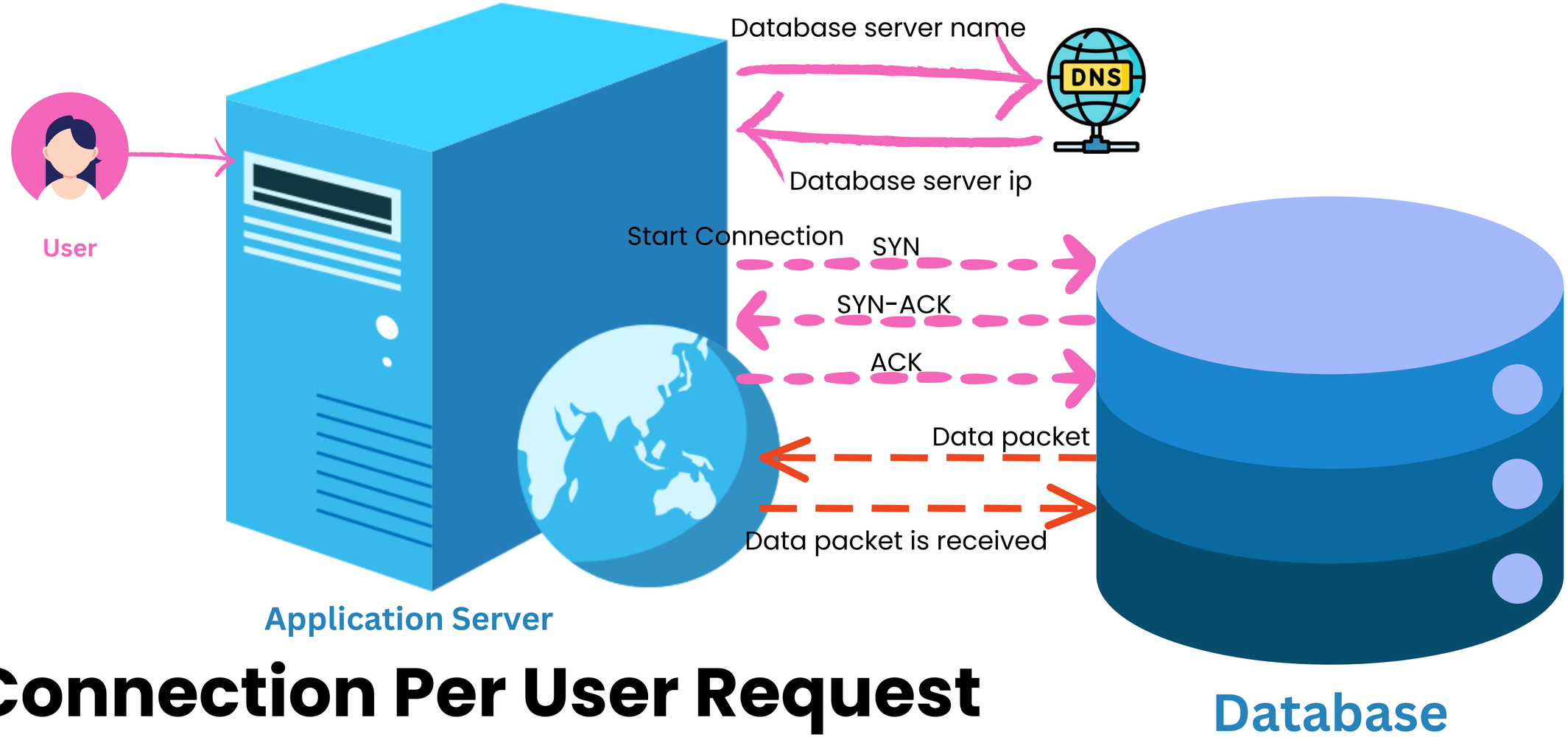


Connection Per User Request

Main Netwok Round Trips :

1- DNS Resolution

2- TCP Handshke and Authentication



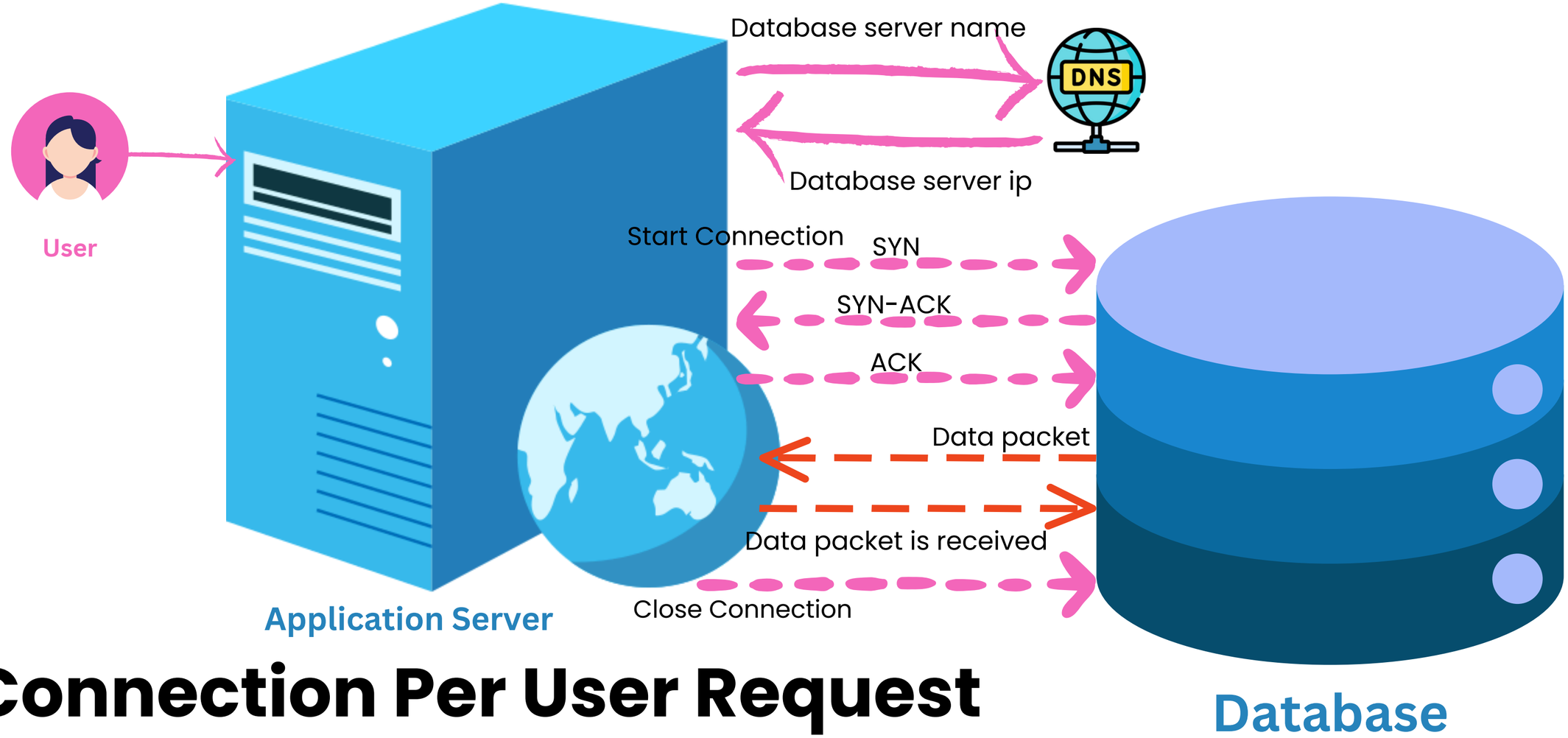
Connection Per User Request

Main Netwok Round Trips :

1- DNS Resolution

2- TCP Handshke and Authentication

3- Transfer Data



Connection Per User Request

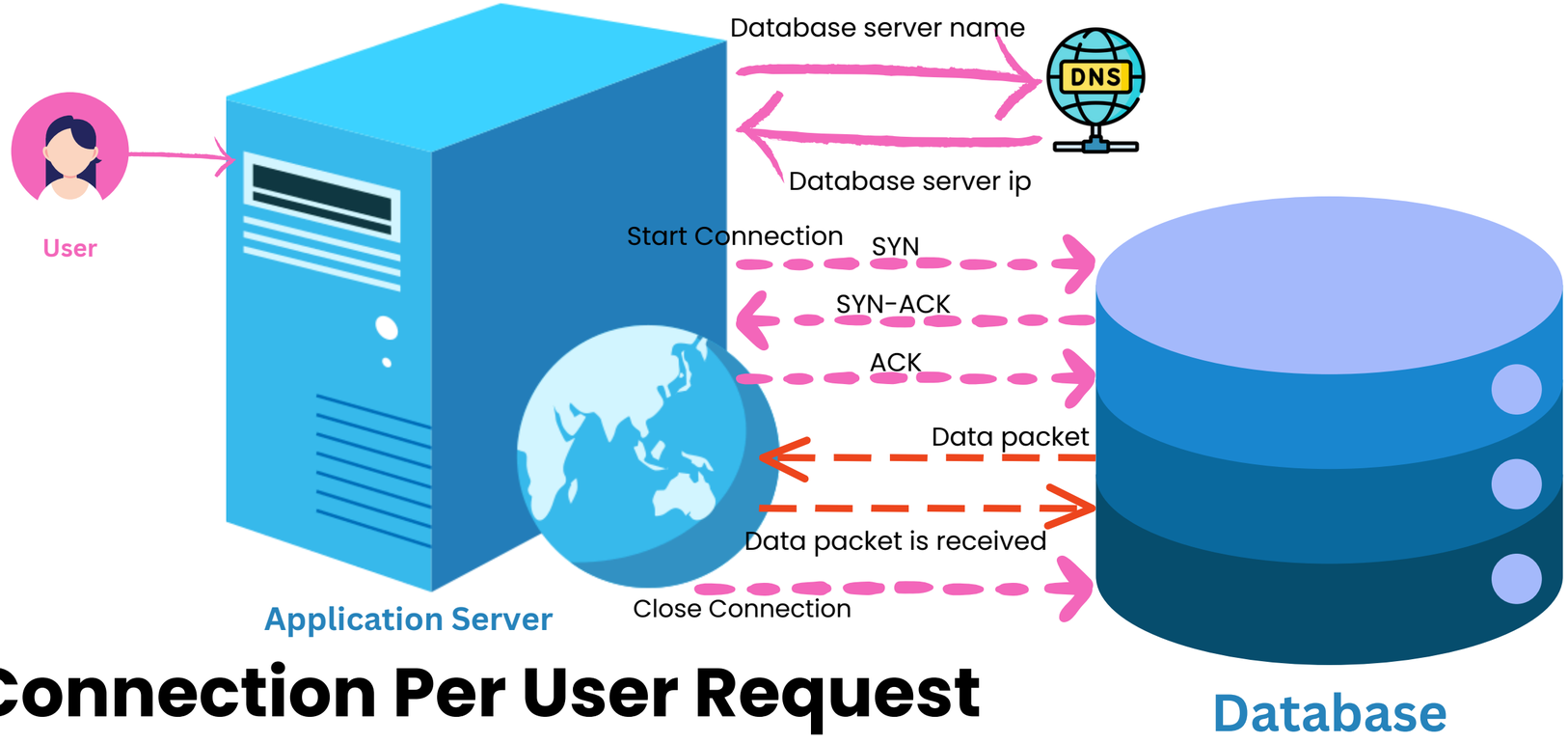
Main Netwok Round Trips :

1- DNS Resolution

2- TCP Handshke and Authentication

3- Transfer Data

4- Close The Connection

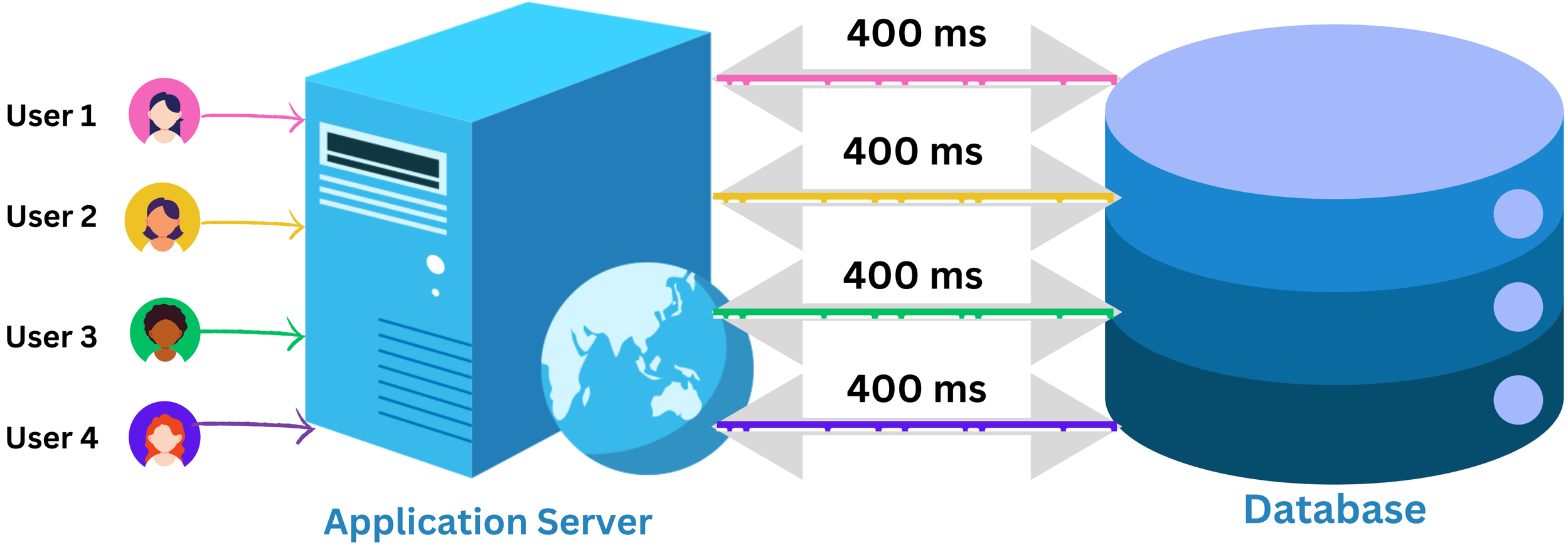


Connection Per User Request

Main Netwok Round Trips : (assuming every trip take 50 ms)

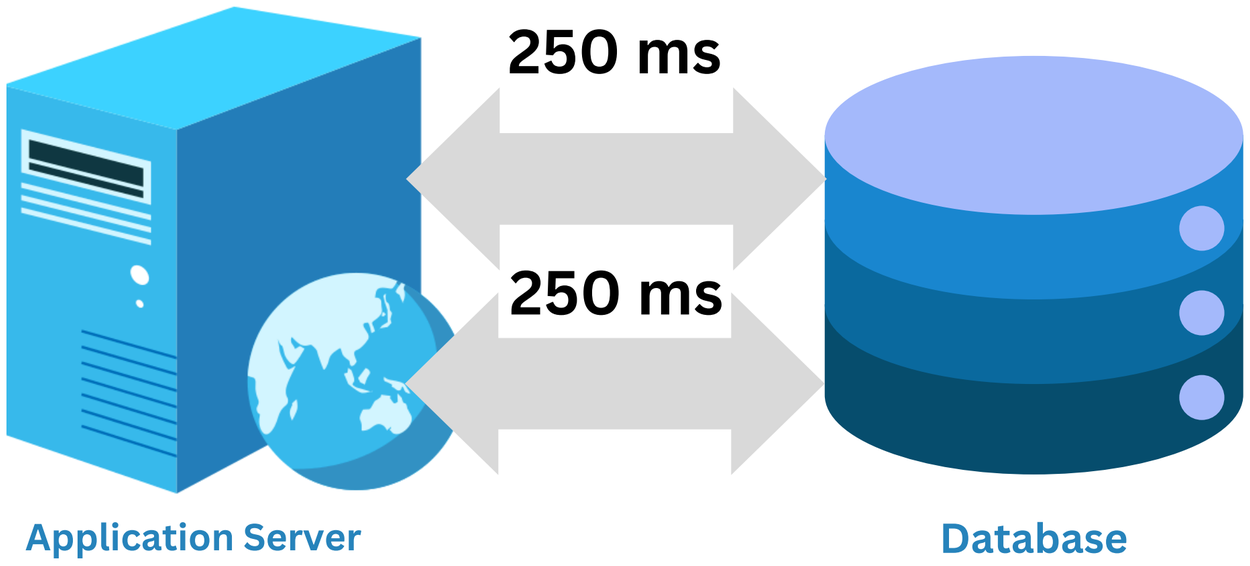
1- DNS Resolution (50ms + 50 ms = 100ms)
2- TCP Handshke and Authentication (50ms + 50 ms + 50ms = 150ms)
3- Transfer Data (50 ms + 50 ms = 100ms)
4- Close The Connection (50 ms)

Total time = 400 ms



Connection Per User Request Model

Average response time = 400 ms



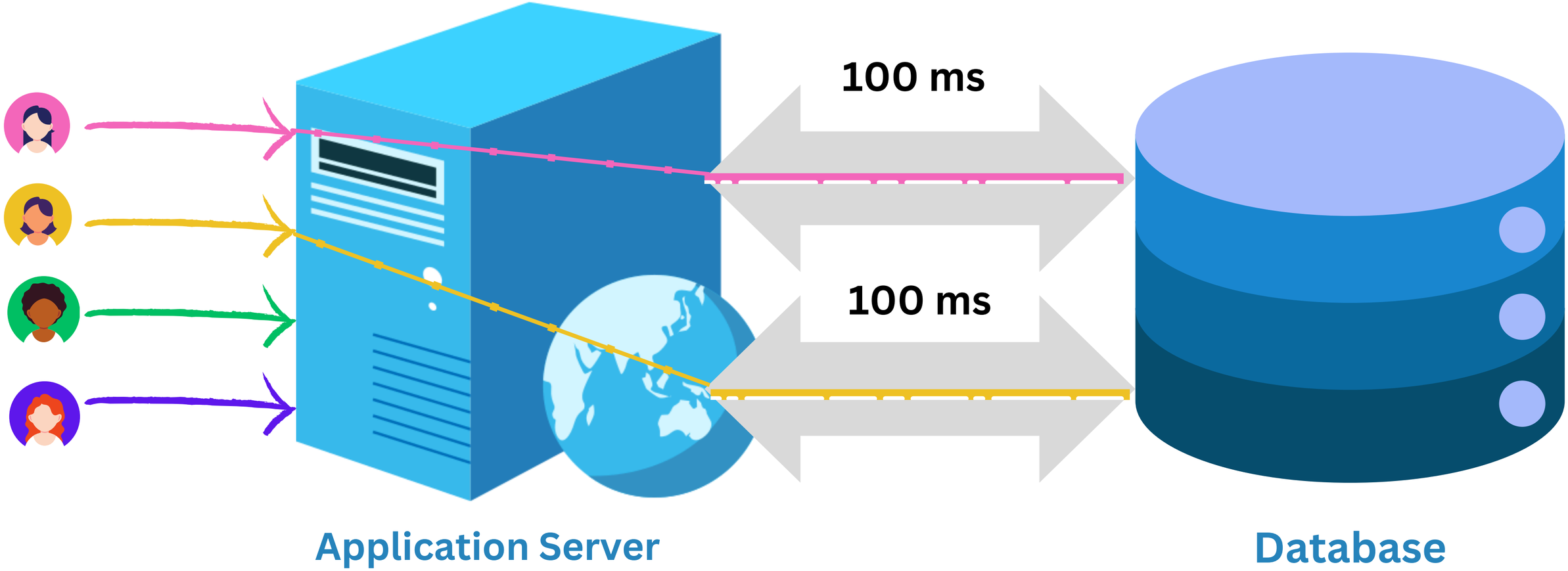
Connection Pools

First Step :

when the application server start-up it automatically open a set of connections with database and keep then opening to serve the incoming user requests

1- DNS Resolution (50ms + 50 ms = 100ms)
2- TCP Handshke and Authentication (50ms + 50 ms + 50ms = 150ms)

Total time = 250 ms

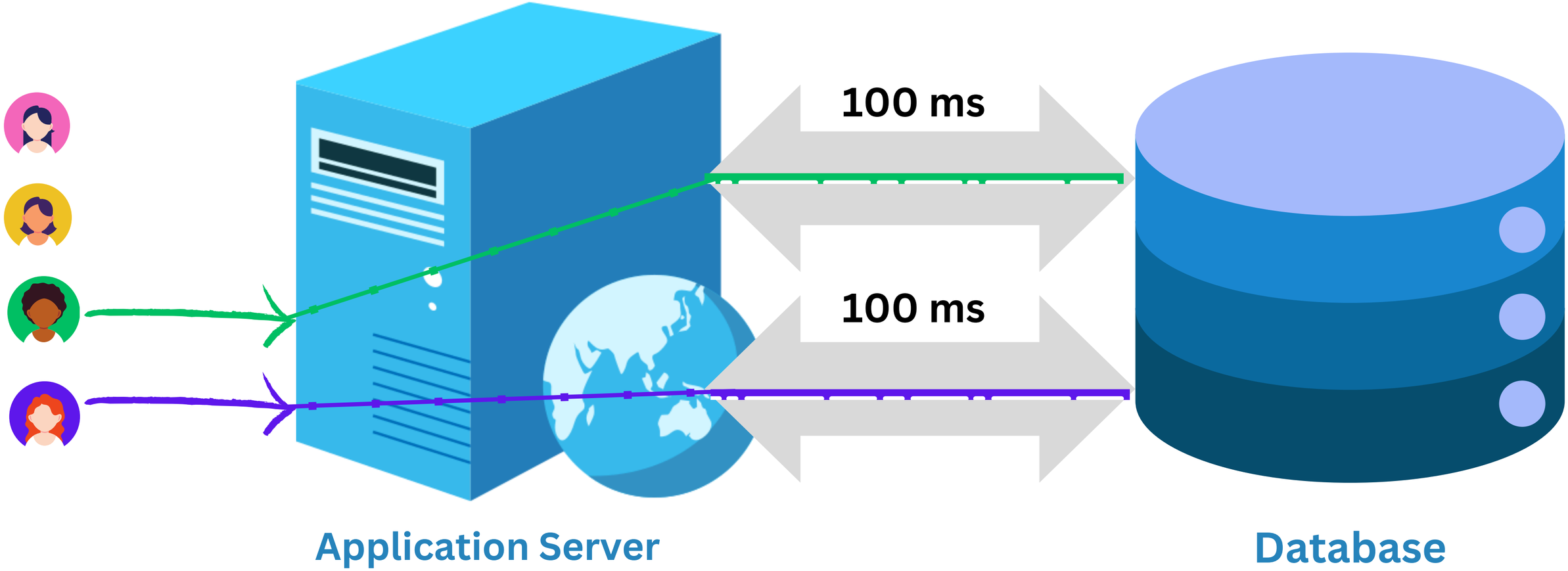


Connection Pools Model

Average response time = 100 ms

Min Pool Size = 2 , Max Pool Size = 2

we will discuss the min and max pool size in next sections with advance examples when every one of them has different value

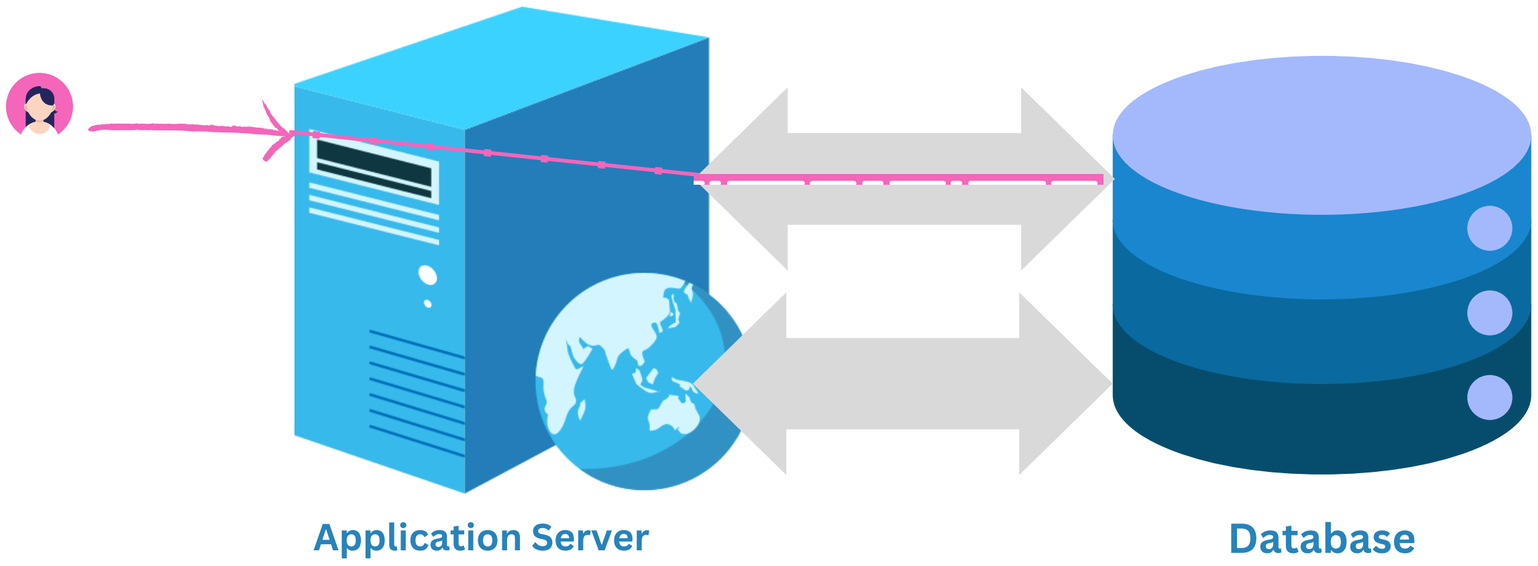


Connection Pools Model

Average response time = 100 ms

Min Pool Size = 2 , Max Pool Size = 2

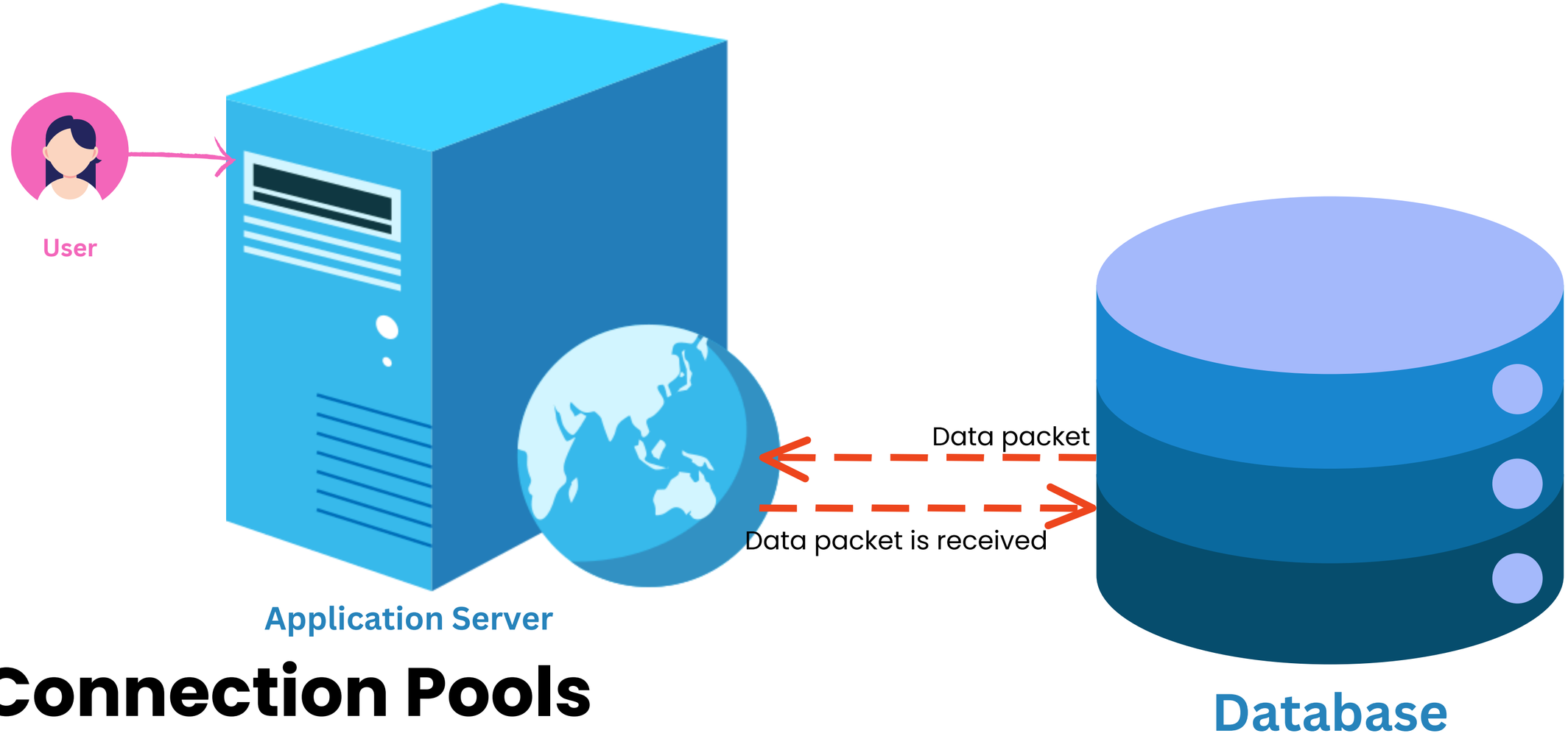
we will discuss the min and max pool size in next sections with advance examples when every one of them has different value



Connection Pools

application server not open new physical tcp connection from scratch

but Reuse one of the available connections

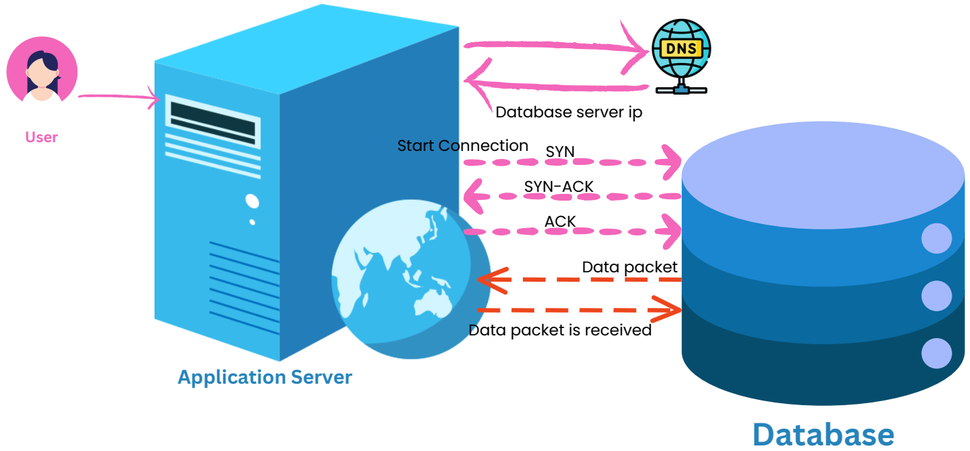


Connection Pools

Main Netwok Round Trips : (assuming every trip take 50 ms)

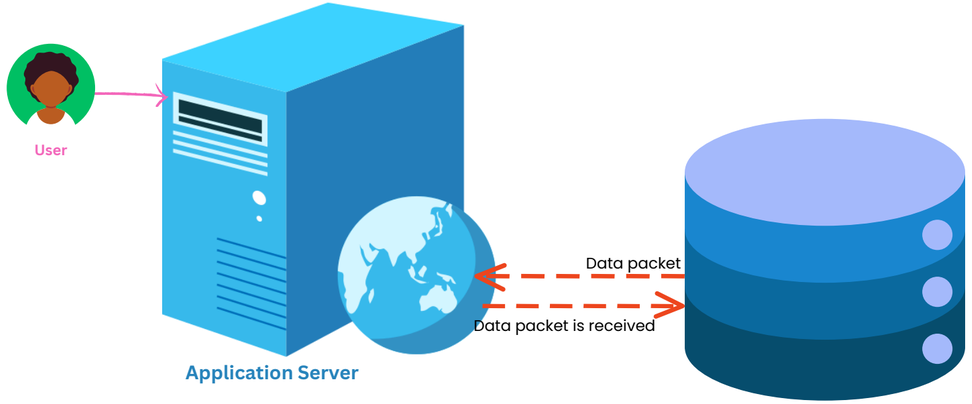
1- Transfer Data (50 ms + 50 ms)

Connection Pools



Connection Pools
First Request

(400 ms)



Connection Pools
Second Request

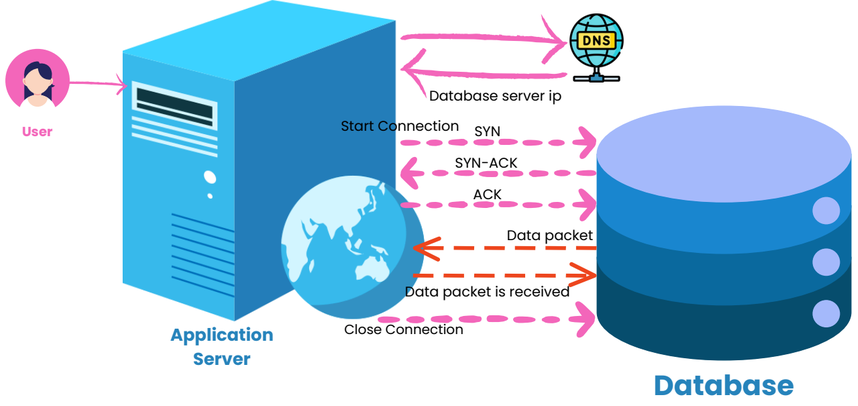
(100 ms)

For Request 1 :

Apply All TCP Network Steps
(Except Close The Connection)

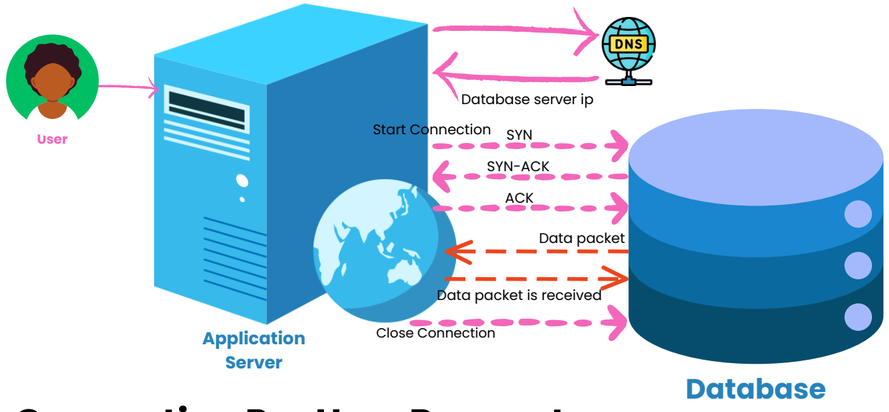
For Request 2 :

Not Apply All TCP Network Steps
Reuse The connection of request 1
Start Transfer Data Directly



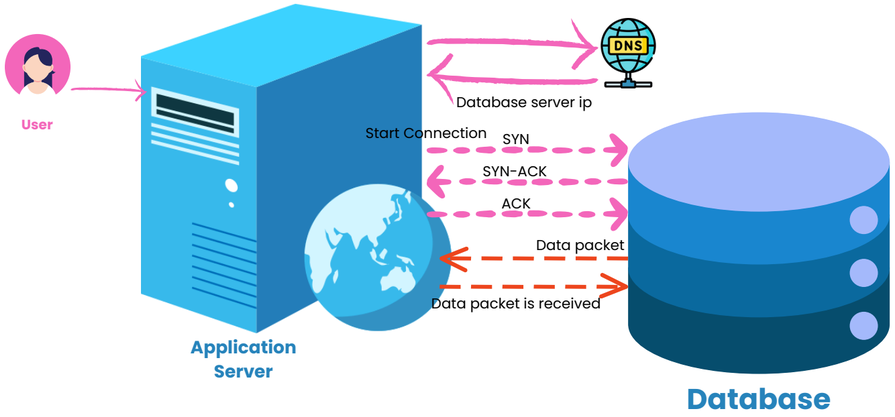
Connection Per User Request
First Request

(400 ms)



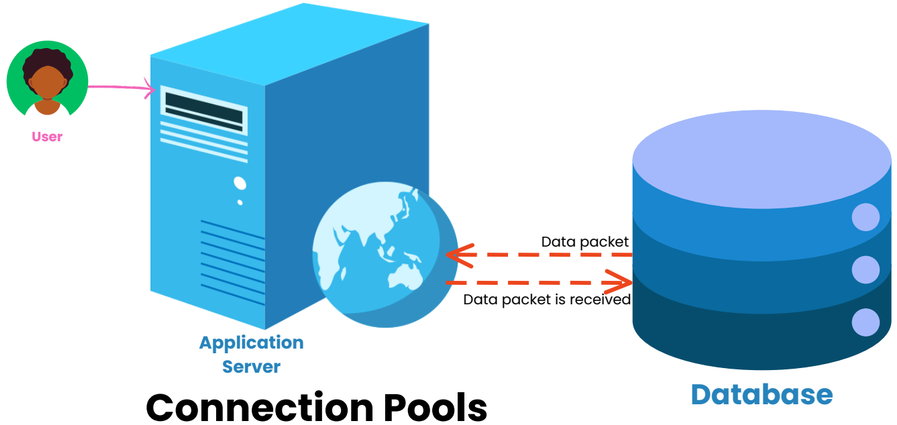
Connection Per User Request
Second Request

(400 ms)



Connection Pools
First Request

(400 ms)



Connection Pools
Second Request

(100 ms)

Is that mean connections pools are always better than Connection per User Request ?

usually yes , but sometimes no

When to use connections pools ?

1- Short Life time queries

when the application send simple queries to database
that take a small amount of time

2- High-Concurrency Applications

when the application has a huge number of conccurent users at the same time
and your devops or SRE team has metrics about number of users

When to use connection per user request ?

1- Long Life time queries

when the application send complex queries to database that take a long amount of time

2- Low-Concurrency Applications

when the application has a very low number of users

Postgres and Ef core example

1- Connection per user request

```
1 reference
public static void AddPostgresDatabaseContext(this IServiceCollection services, IConfiguration configuration)
{
    services.AddDbContext<PostgresDatabaseContext>((serviceProvider, options) =>
    {
        var postgresOptions = serviceProvider
                              .GetRequiredService<IOptions<PostgresqlDatabaseConfigurationOptions>>().Value;

        options.UseNpgsql(postgresOptions.ConnectionString);
    });

}
```

Postgres and Ef core example

2- Connections Pools

```
1 reference
public static void AddPostgresDatabaseContext(this IServiceCollection services, IConfiguration configuration)
{
    services.AddDbContextPool<PostgresDatabaseContext>((serviceProvider, options) =>
    {
        var postgresOptions = serviceProvider
                              .GetRequiredService<IOptions<PostgresqlDatabaseConfigurationOptions>>().Value;

        options.UseNpgsql(postgresOptions.ConnectionString);
    });

}
```

appsettings.json

Schema: <No Schema Selected>

```
1
2
3   [redacted]:Minimum Pool Size=10;Maximum Pool Size=100;"
4
5
6
```

Postgres and Ef core example

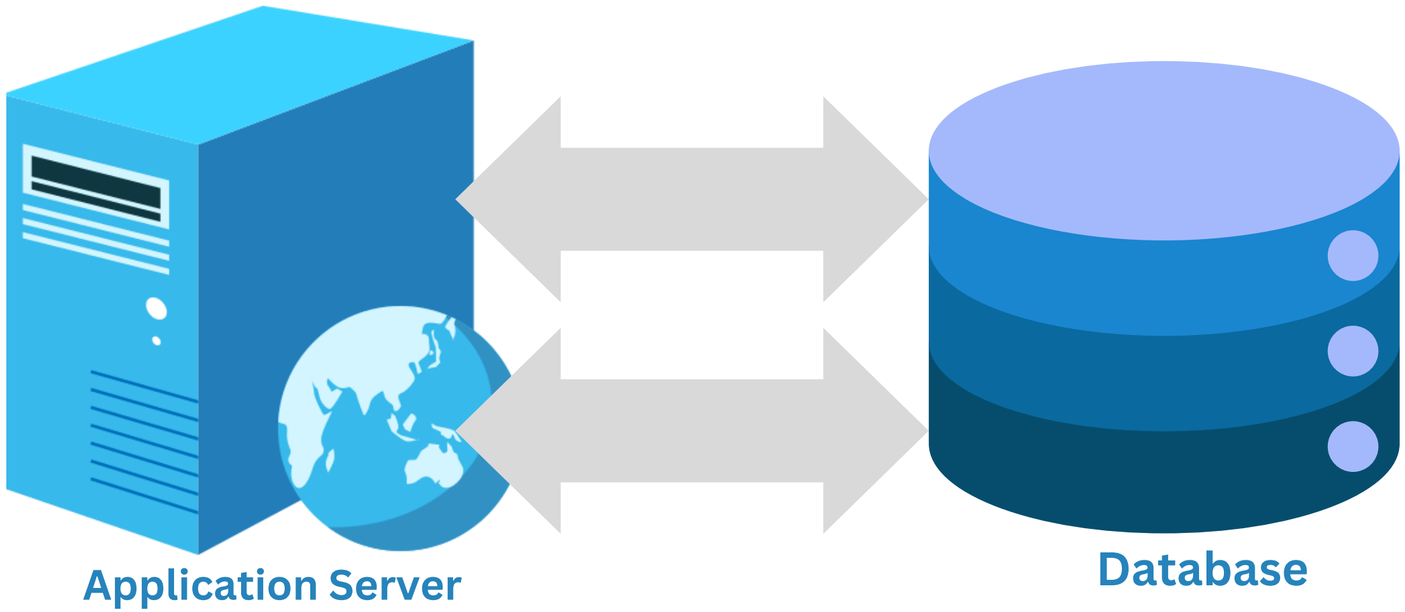
Connection String Parameters | Npgsql Documentation
Click here to visit the page.

2- Connections Pools

Pooling

Parameter	Description	Default
Pooling	Whether connection pooling should be used.	true
Minimum Pool Size	The minimum connection pool size.	0
Maximum Pool Size	The maximum connection pool size.	100 since 3.1, 20 previously
Connection Idle Lifetime	The time (in seconds) to wait before closing idle connections in the pool if the count of all connections exceeds `Minimum Pool Size`. Introduced in 3.1.	300
Connection Pruning Interval	How many seconds the pool waits before attempting to prune idle connections that are beyond idle lifetime (see `Connection Idle Lifetime`). Introduced in 3.1.	10
Connection Lifetime	The total maximum lifetime of connections (in seconds). Connections which have exceeded this value will be destroyed instead of returned from the pool. This is useful in clustered configurations to force load balancing between a running server and a server just brought online.	0 (disabled)

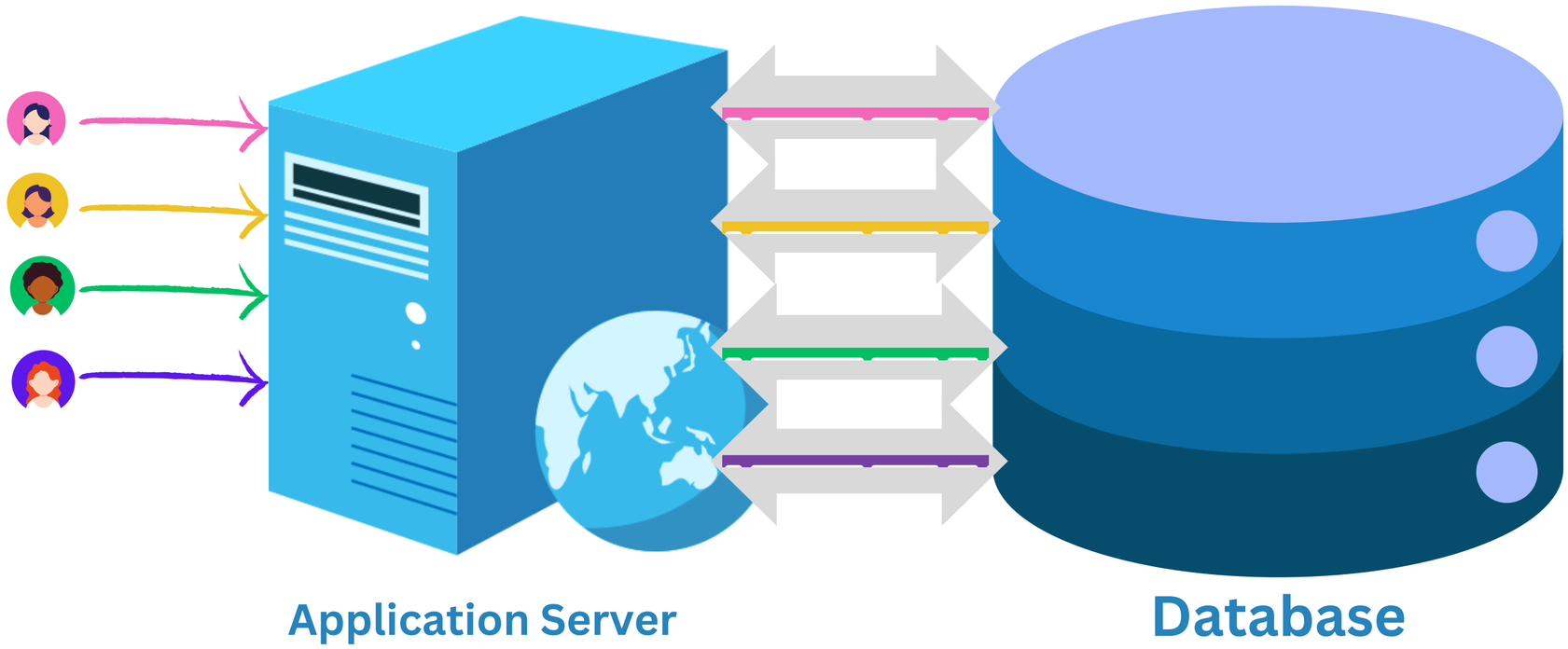
Ex : Min Pool Size = 2



Min Pool Size :

1- Min Number of connections that must be opened between appliccation server and database.
2- whether there are api requests from users or not.
3- they are opened to be ready when application server receive api requests.
4- usually opend when when the application server start-up.

Ex : Max Pool Size = 4



Max Pool Size :

Max Number of connections between appliccation server and database.

Full Example :

Min Pool Size = 2

Max Pool Size = 4

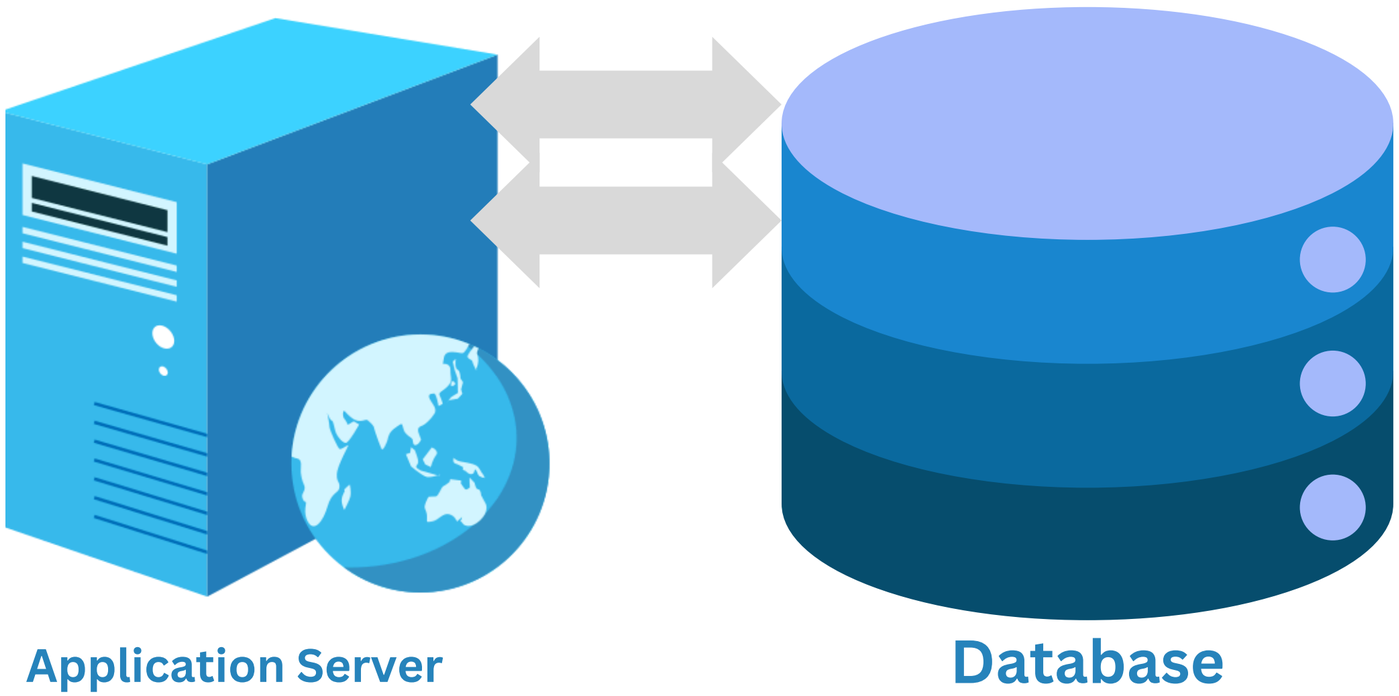
Number Of Concurrent Users = 5

Application Server

Database

Full Example : Step 1

When Application Server Start-Up Will Open 2 Connections With Database and Keep Them Always Opening
and This Because Min Pool Size = 2



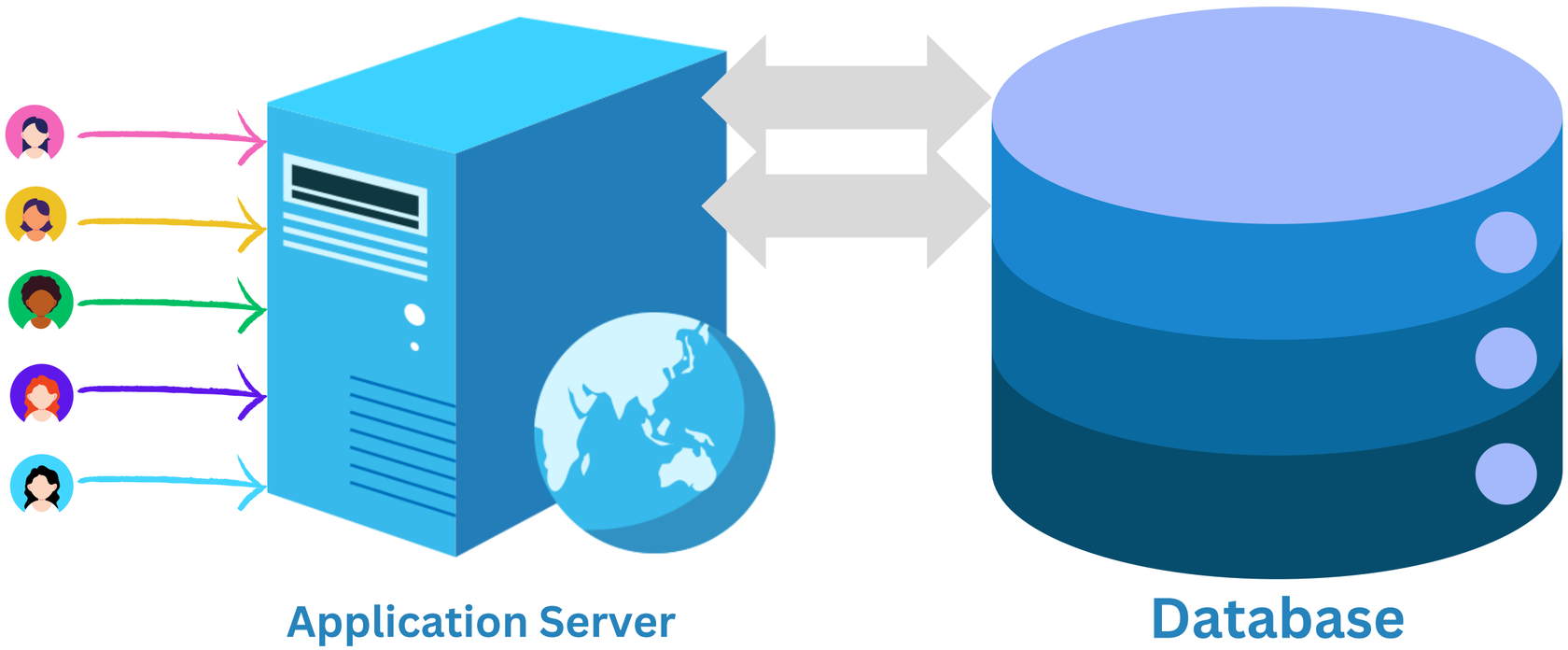
Min Pool Size = 2

Current Connections = 2

Max Pool Size = 4

Full Example : Step 2

5 Conncurrant Users Send 5 Api Requests At The Same Time



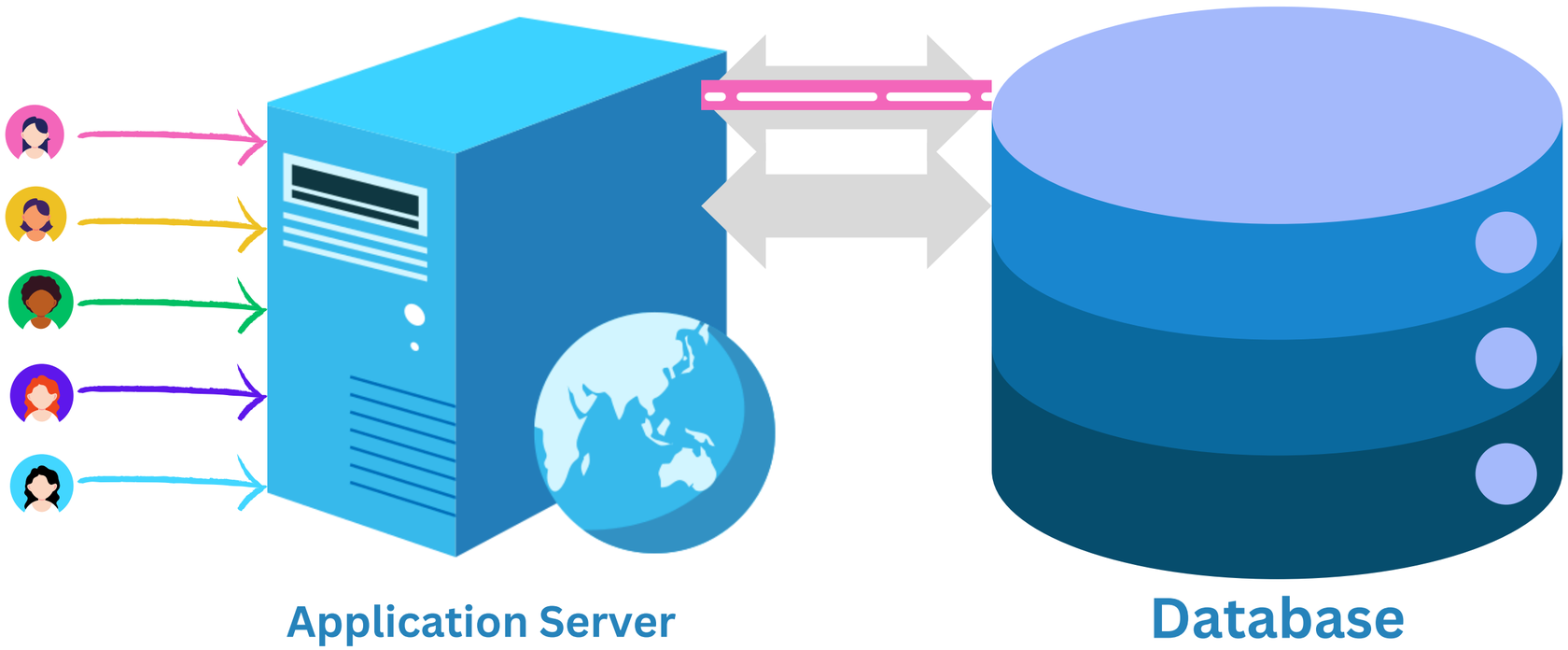
Min Pool Size = 2

Current Connections = 2

Max Pool Size = 4

Full Example : Step 3

User 1 Will use connection 1



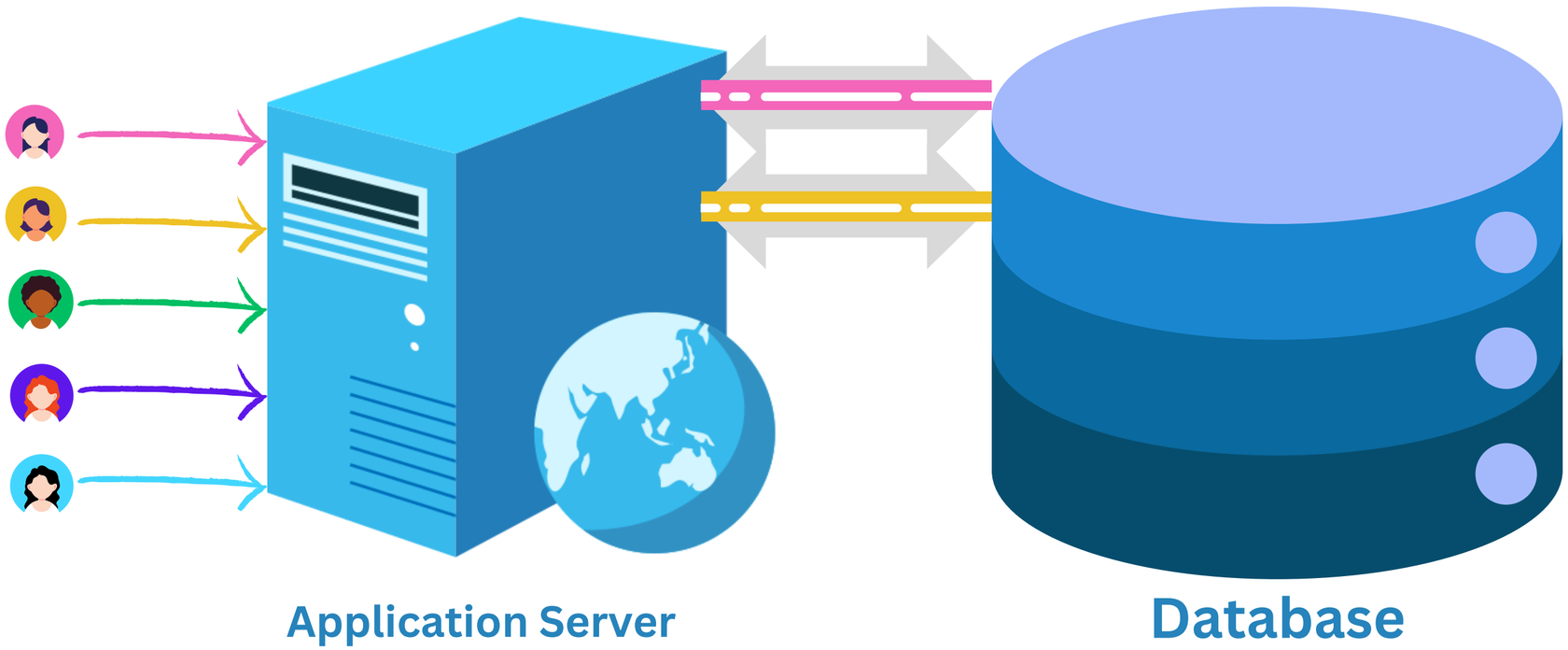
Min Pool Size = 2

Current Connections = 2

Max Pool Size = 4

Full Example : Step 4

User 2 Will use connection 2



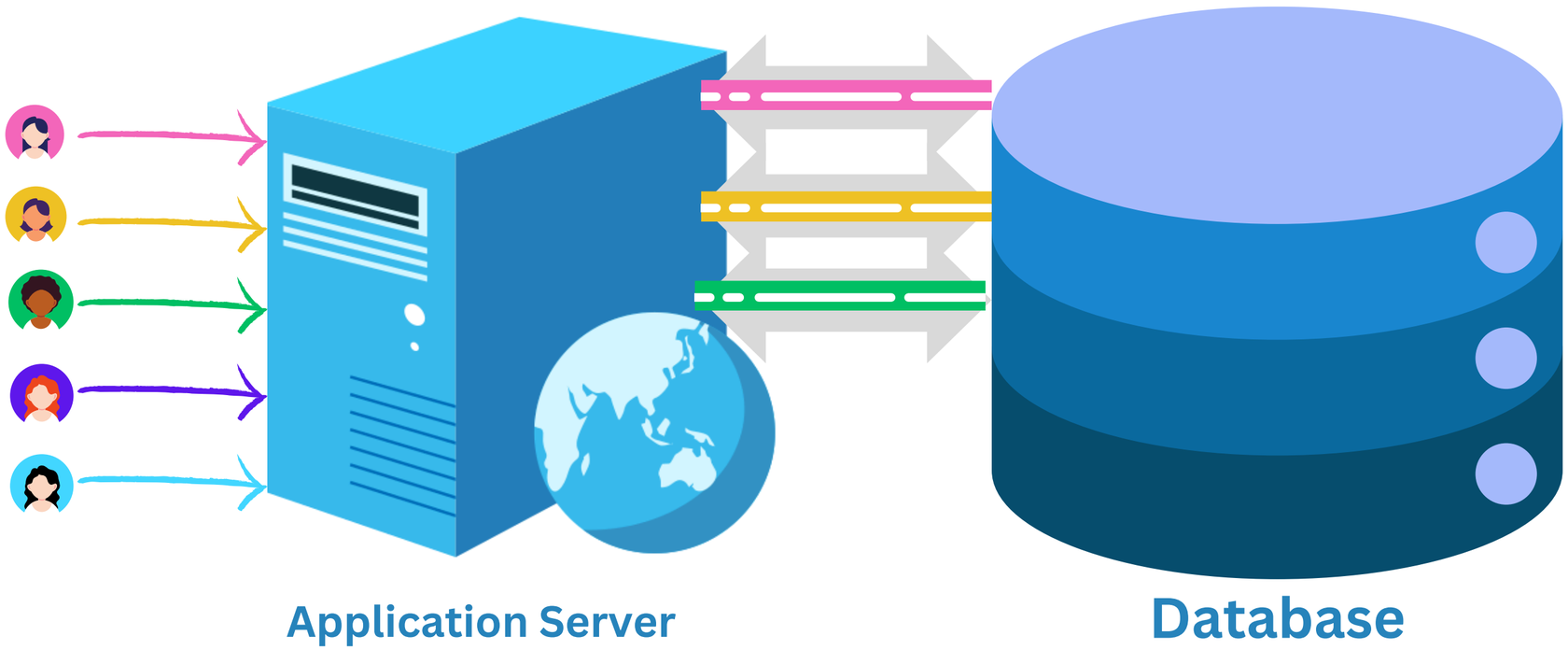
Min Pool Size = 2

Current Connections = 2

Max Pool Size = 4

Full Example : Step 5

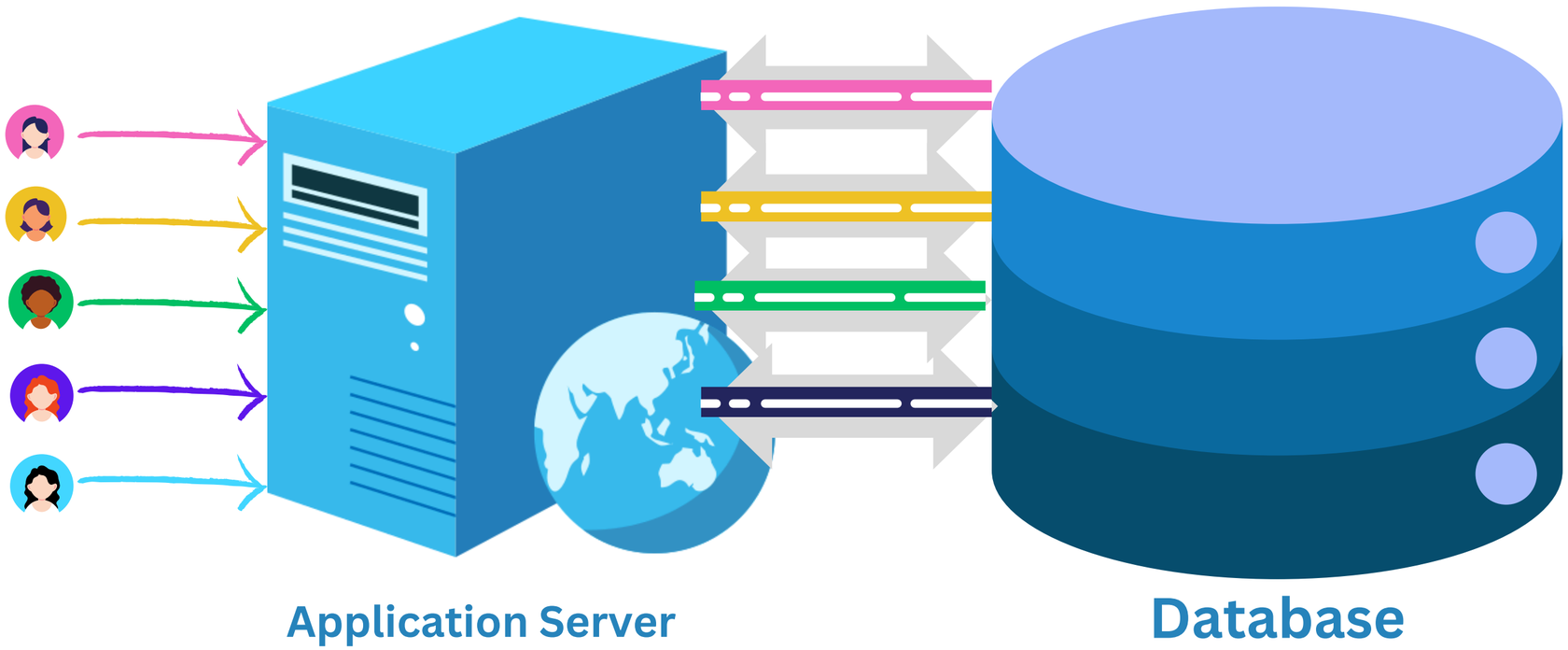
User 3 will not find any available connections
But current connections < max pool size
so application server will open new connection 3 for user 3



Min Pool Size = 2
Current Connections = 3
Max Pool Size = 4

Full Example : Step 6

User 4 will not find any available connections
But current connections < max pool size
so application server will open new connection 4 for user 4



Min Pool Size = 2
Current Connections = 4
Max Pool Size = 4

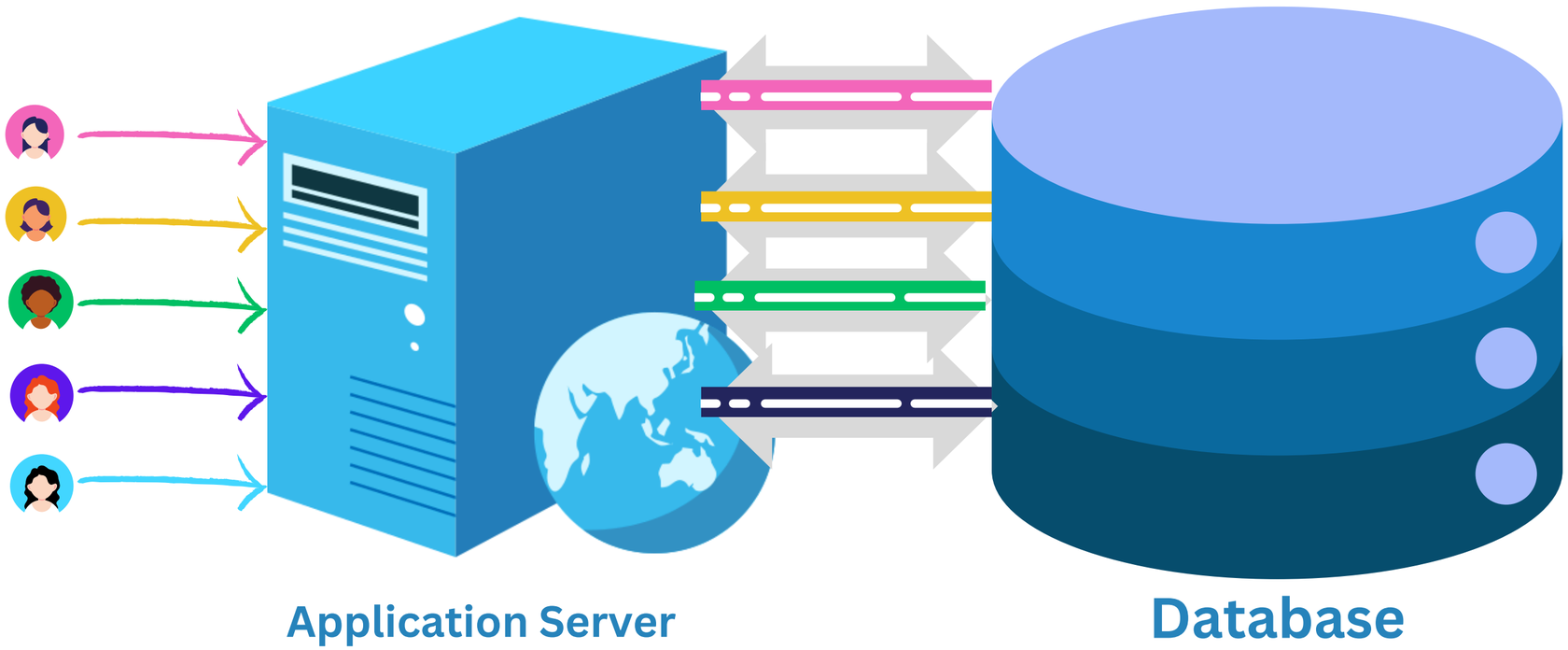
Full Example : Step 7

User 5 will not find any available connections

But current connections = max pool size

so application server will not open new connection 5 for user 5

But application server will wait until one of the current connections will be free or available and then use it



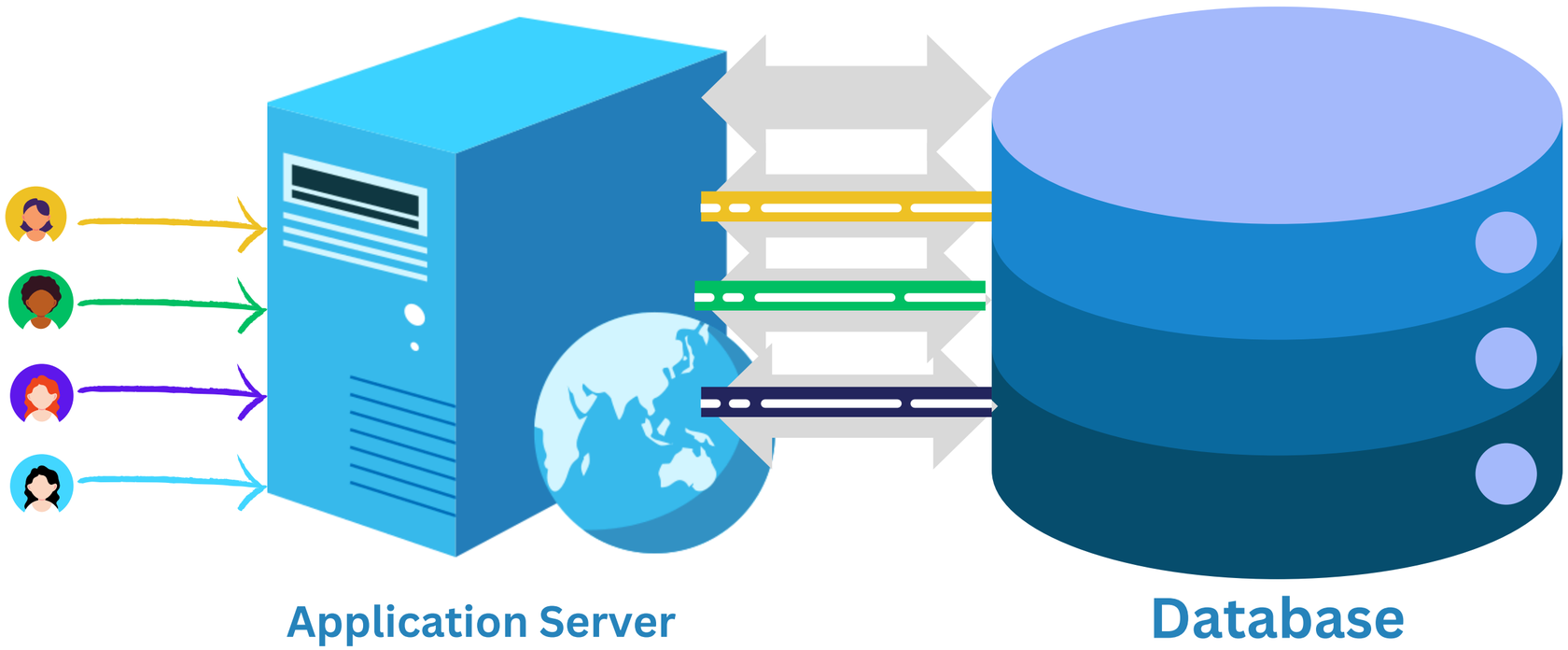
Min Pool Size = 2

Current Connections = 4

Max Pool Size = 4

Full Example : Step 8

Assume user 1 get his response and connection 1 now is free



Min Pool Size = 2

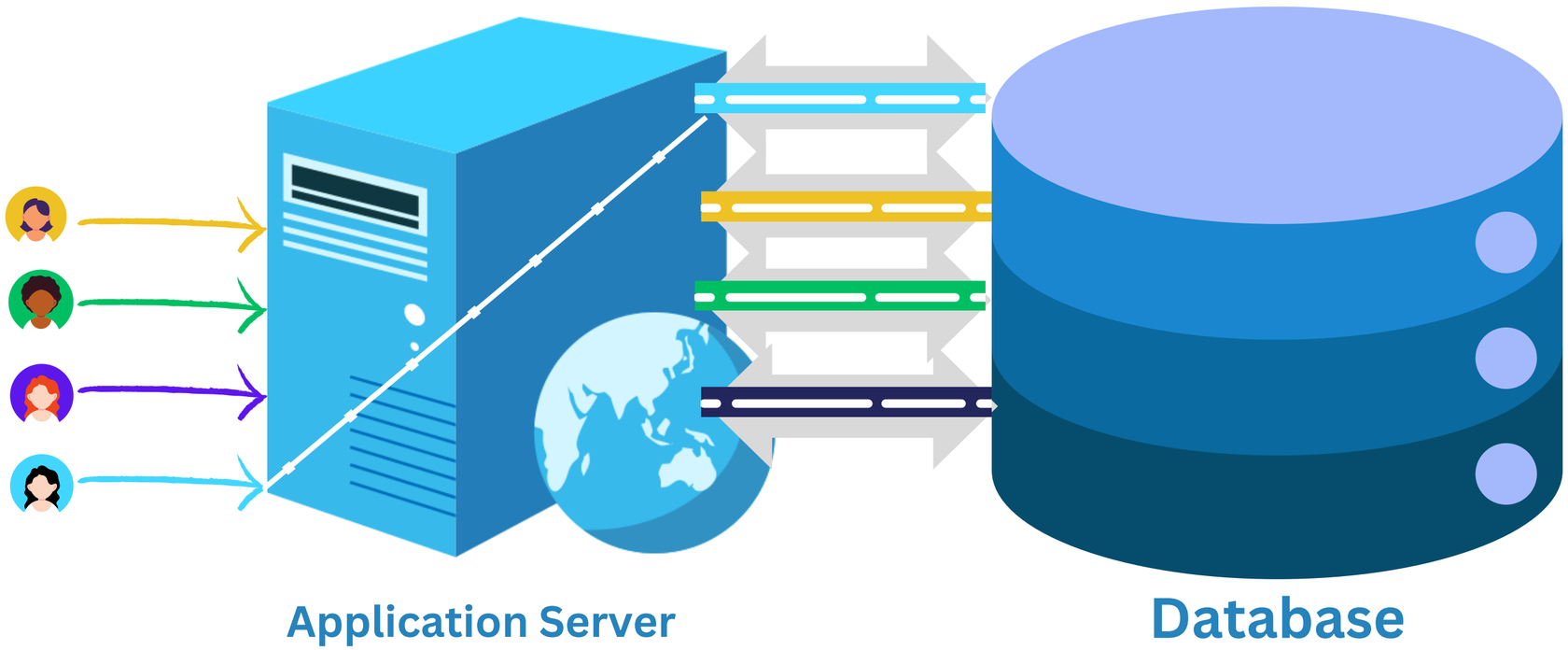
Current Connections = 4

Max Pool Size = 4

Full Example : Step 9

Assume user 1 get his response and connection 1 now is free

so user 5 will use connection 1



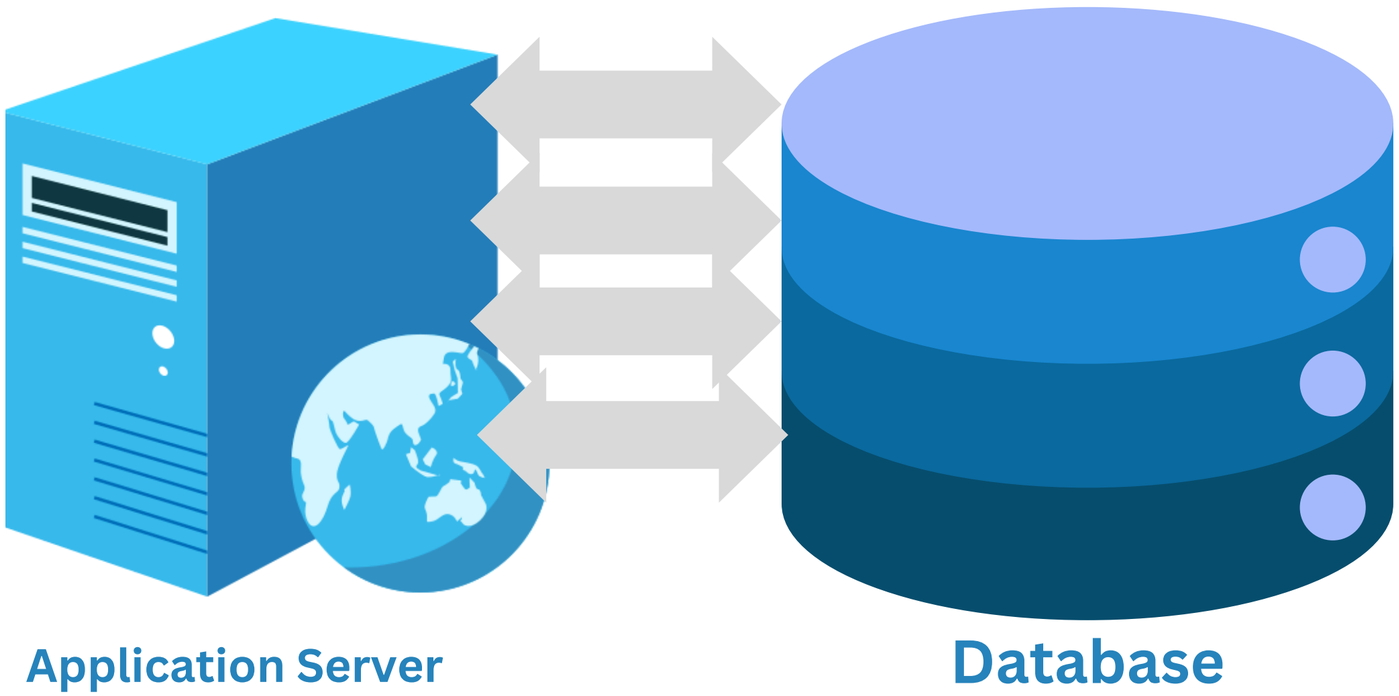
Min Pool Size = 2

Current Connections = 4

Max Pool Size = 4

Full Example : Step 10

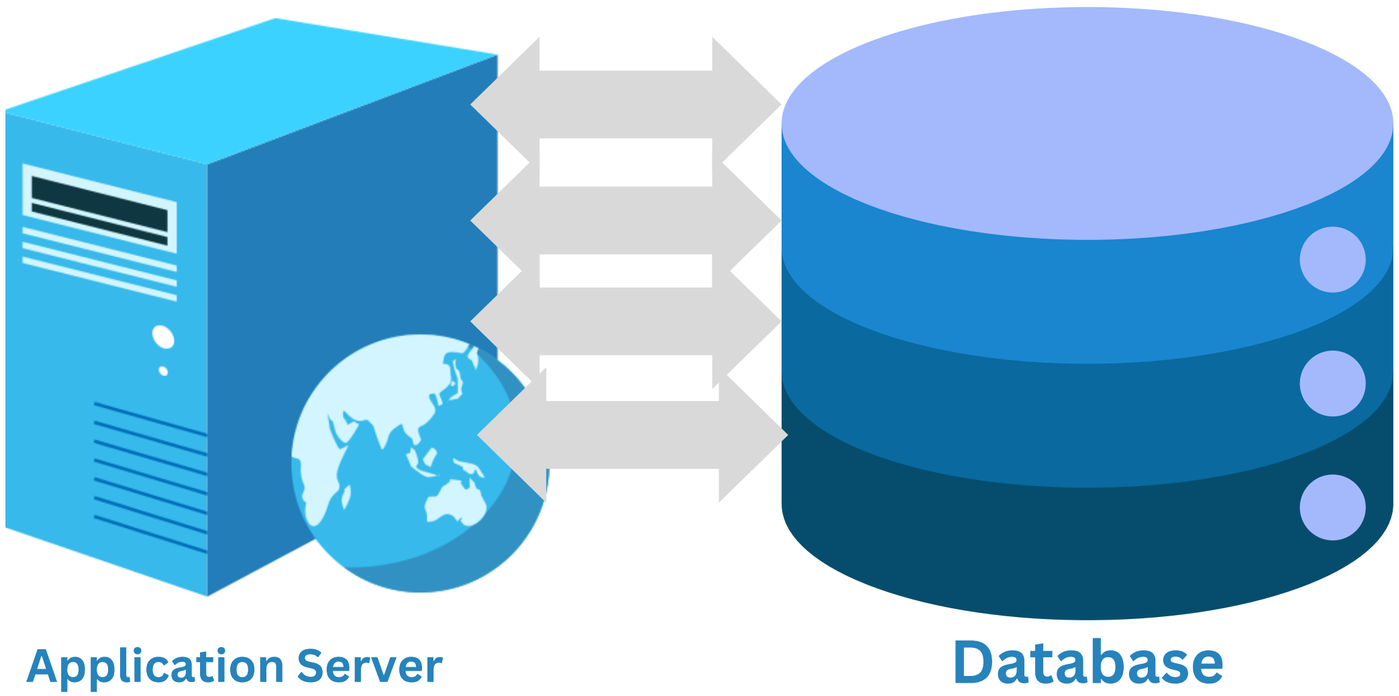
so now all 5 users get thier responses ,
and now there will be 4 current free connections



Min Pool Size = 2
Current Connections = 4
Max Pool Size = 4
Connection Idle Lifetime = 30 sec

Full Example : Step 11

assume no users send api for 40 sec



Min Pool Size = 2

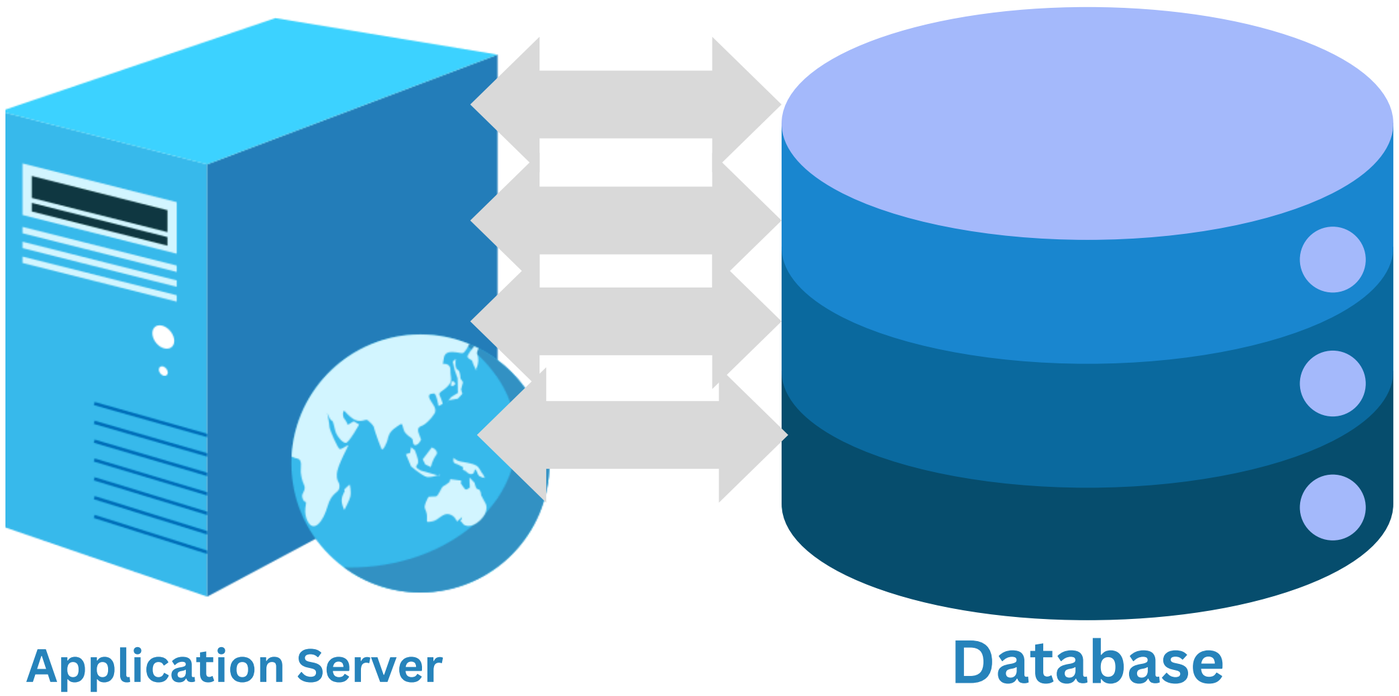
Current Connections = 4

Max Pool Size = 4

Connection Idle Lifetime = 30 sec

Full Example : Step 12

after 29 sec , there will be also 4 free current connections



Min Pool Size = 2

Current Connections = 4

Max Pool Size = 4

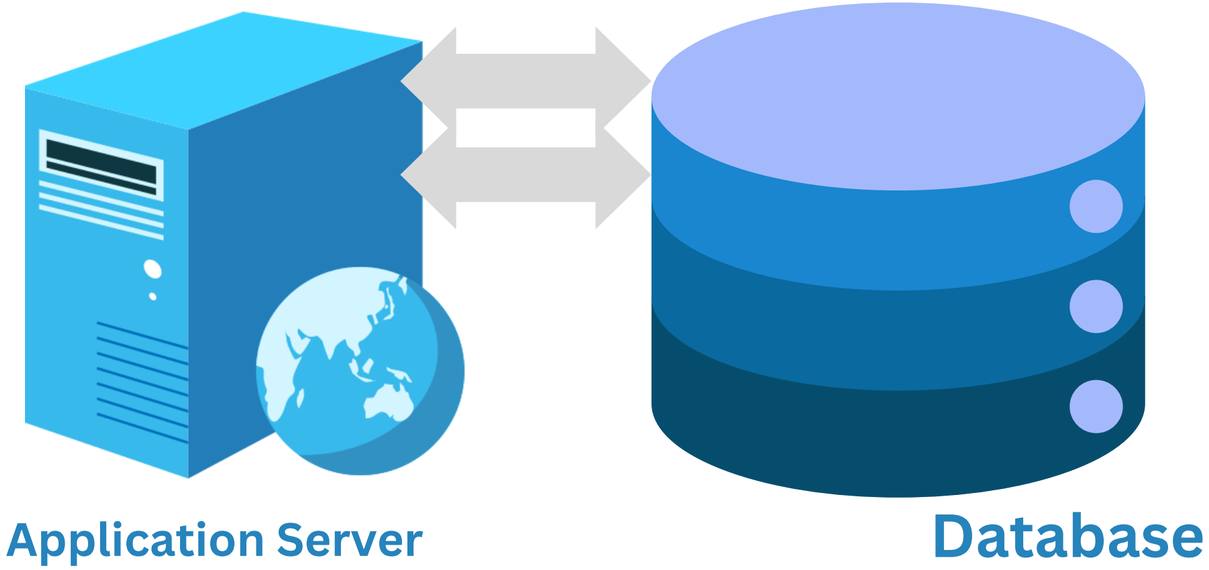
Connection Idle Lifetime = 30 sec

Full Example : Step 11

after 30 sec , the application server will close connection 3 , 4 and keep connections 1 , 2
because we set Connection Idle Lifetime = 30 sec
so if the connection is not used by any user for 30 sec ,
the connection will be closed.

note : why application server keep connection 1 , 2 alive and not close them ?

because min pool size = 2 , so application server will need always to keep 2 connections alive



Min Pool Size = 2
Current Connections = 2
Max Pool Size = 4
Connection Idle Lifetime = 30 sec

How to calculate or choose a good min and max pool size ?

1- Get the average of conccurant users from SRE Team :

Assume it = **1000** conccurant users at every second

2- Get the average response time for single query from database

Assume it = **100 ms** or **.1 sec**

3- Get the average response time for api user request from your non-functional requirement docs

Assume it = **1 sec**

4- Calculate number of queries that can be serve by single connection :

average response time of api / average response of database query =

1 sec / .1 sec = 10 queries

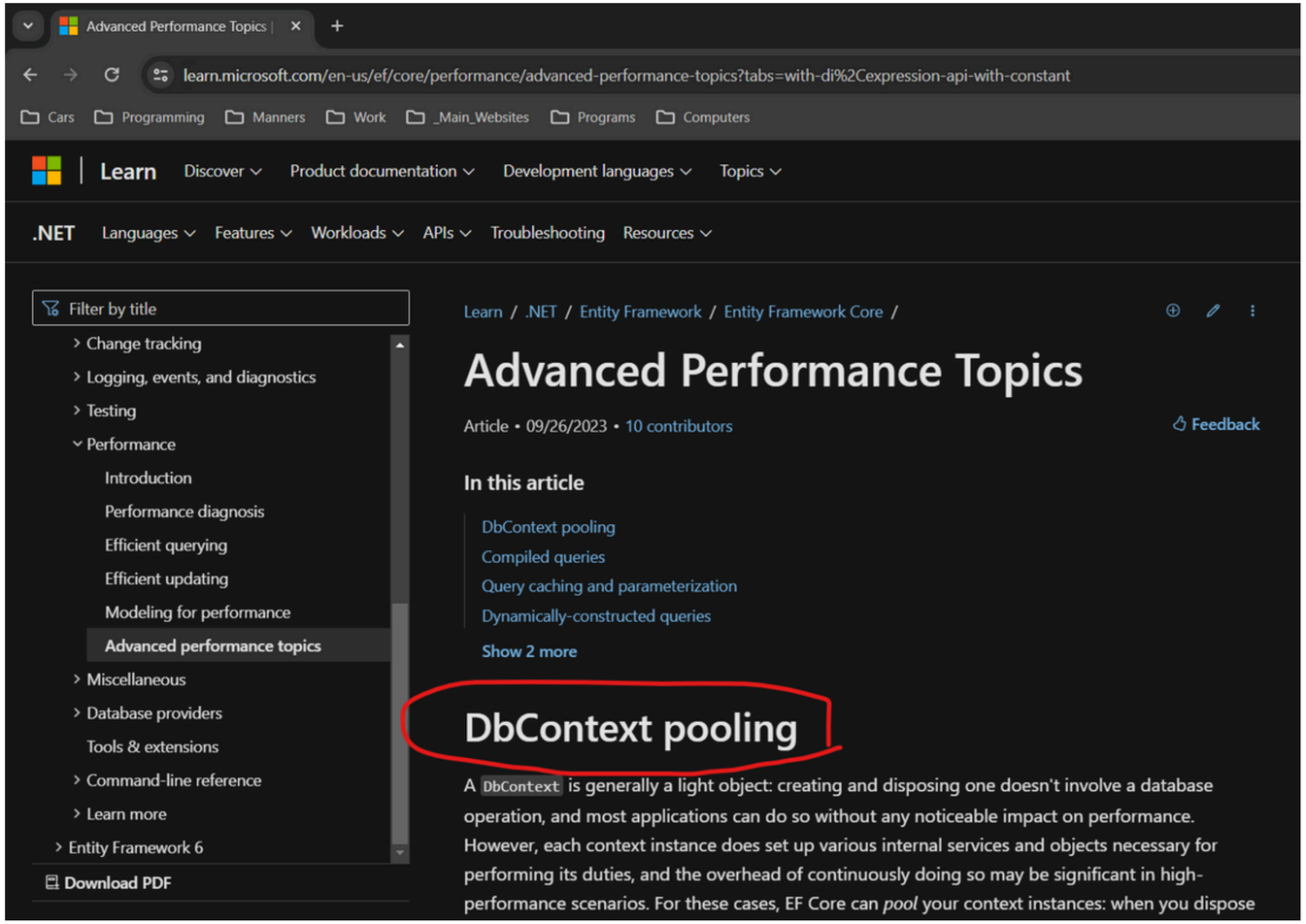
5- Calculate number of needed connections :

number of conccurant users / number of queries by connection =

1000 user / 10 queries = 100 connections

Choosing the right and suitable communication model is very critical and make very difference impact on the performance and throughput of the system

Choosing the right and suitable communication model is very critical and make very difference impact on the performance and throughput of the system



Choosing the right and suitable communication model is very critical and make very difference impact on the performance and throughput of the system

redis.io/kb/doc/1mebipyp1e/performance-tuning-best-practices

Cars | Programming | Manners | Work | _Main_Websites | Programs | Computers

Connection pooling

It is always recommended to use connection pooling. Otherwise, each request will open a new connection. This exposes many possible momentary problems that could prevent the opening of the connection. In addition, opening and closing connections cause additional overhead.

A few connections will serve all the requests using connection pooling and will not close after each request. This eliminates the problems above and will ensure better performance because no time will be wasted on opening and closing connections.

Redis Enterprise keeps connections open forever unless clients close them or don't reply to TCP keep-alive messages. Idle connections that do not answer keepalives for 5 consecutive minutes are closed: Redis Enterprise assumes that clients closed their connections. Verifying the number of active connections is possible using the `CLIENT LIST` command.

If using a pool, you may need to increase the minimum number of connections in the pool so no new thread creation will delay operations on the client. If not using a pool, evaluate using it.

Examples of clients supporting connection pooling:

- redis-py
- jedis

Not all the clients make the connection pooling feature available: StackExchange, as an example, multiplexes a

Performance Tuning Best Practices

This document resumes the causes of slowness when using a Redis database. Discover the...

redis.io

Choosing the right and suitable communication model is very critical and make very difference impact on the performance and throughput of the system

13 Common RabbitMQ Mistakes and How to Avoid Them

Last updated: 2023-08-08

We have been hosting RabbitMQ for many years, and have probably seen way more configuration mistakes than anybody else. Here is a list that will help you avoid common RabbitMQ mistakes!

Please read Part 1 RabbitMQ Best Practice for general best practices and 'dos and don'ts' tips for RabbtitMQ.

Update August 2023: RabbitMQ strongly advice you to use Quorum Queues in favour of classic mirrored queues. Read: Reasons you should switch to Quorum Queues.

1. **Don't open and close connections or channels repeatedly.**
   Have long-lived connections if possible, and use channels for each task. The handshake process for an AMQP connection is quite complex and requires at least 7 TCP packets (more if TLS is used). Channels can be opened and closed more frequently if needed. Even channels should be long-lived if possible, e.g., reuse the same channel per thread for publishing. Don't open a channel each time you are publishing. If you can't have long-lived connections, then make sure to gracefully close the connection. **Best practice is to reuse connections and multiplex a connection between threads with channels.**
   - AMQP connections: 7 TCP packets
   - AMQP channel: 2 TCP packets
   - AMQP publish: 1 TCP packets (more for larger messages)
   - AMQP close channel: 2 TCP packets
   - AMQP close connection: 2 TCP packets
   - Total 14-19 packets (+ Acks)
2. **Don't use too many connections or channels.**
   Try to keep the connection/channel count low. Use separate connections to publish and consume. Ideally, you should have one connection per process, and then use one channel per thread in your application.

LOVISA JOHANSSON
Developer

FREE EBOOK
"The Optimal RabbitMQ Guide"
Download your copy

Posts from @CloudAMQP

Part 1: RabbitMQ Best Practices

We have been working with RabbitMQ a long time, and we have probably seen way more configuration mistakes than anybody else. We know how to configure for optimal performance and how to get the most stable...

CloudAMQP

13 Common RabbitMQ Mistakes and How to Avoid Them

We have been hosting RabbitMQ for many years, and we have probably seen way more configuration mistakes than anybody else. Here is a list, that will help you avoid common RabbitMQ mistake!

CloudAMQP