Using Entity Framework Core in Your Controllers

Kevin Dockx
Architect

@KevinDockx | www.kevindockx.com

Coming Up

Introducing the repository pattern

Learning about async code

Reading, creating, updating and deleting resources via Entity Framework Core

Using AutoMapper

Introducing the Repository Pattern

No repository pattern	vs	Repository pattern
Code duplication More error-prone code Harder to test the consuming class		

The Repository Pattern

An abstraction that reduces complexity and aims to make the code, safe for the repository implementation, persistence ignorant

Introducing the Repository Pattern

No repository pattern	vs	Repository pattern
Code duplication		No duplication
More error-prone code		Less error-prone code
Harder to test the consuming class		Better testability of the consuming class

Persistence Ignorant

Switching out the persistence technology is not the main purpose.
Choosing the best one for each repository method is.

Demo

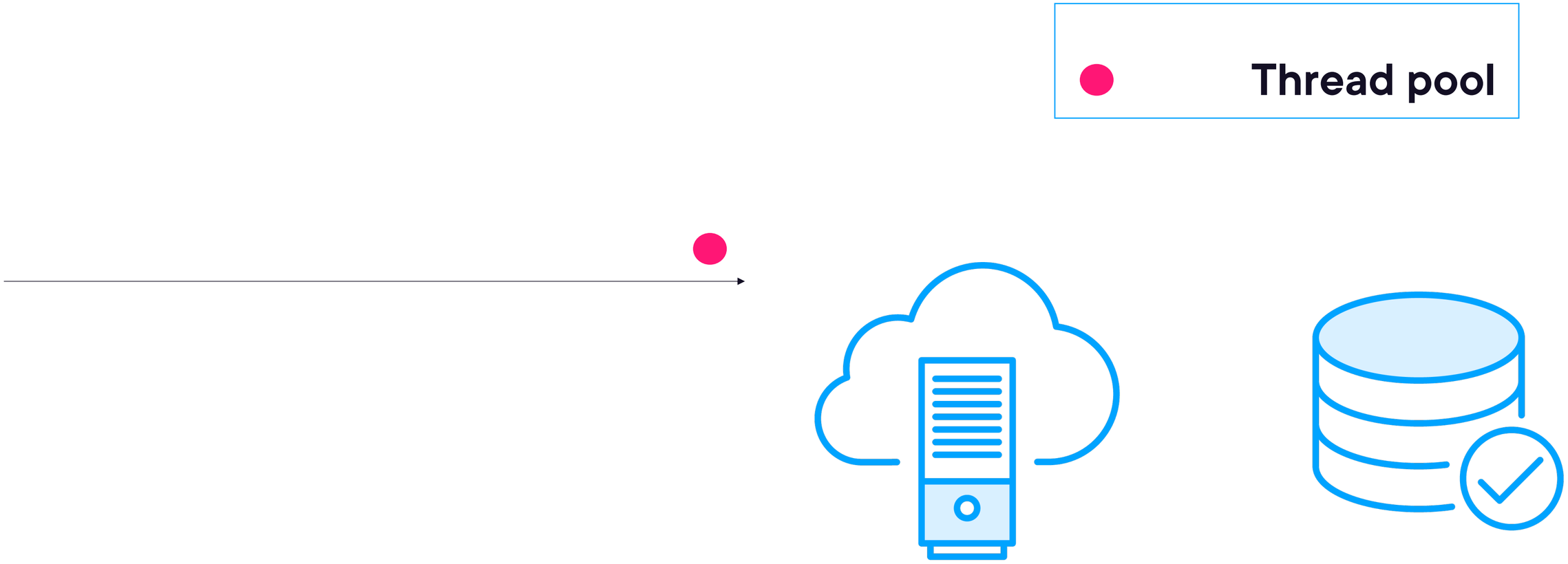
Introducing the repository pattern (part 1)

The Purpose of Async Code

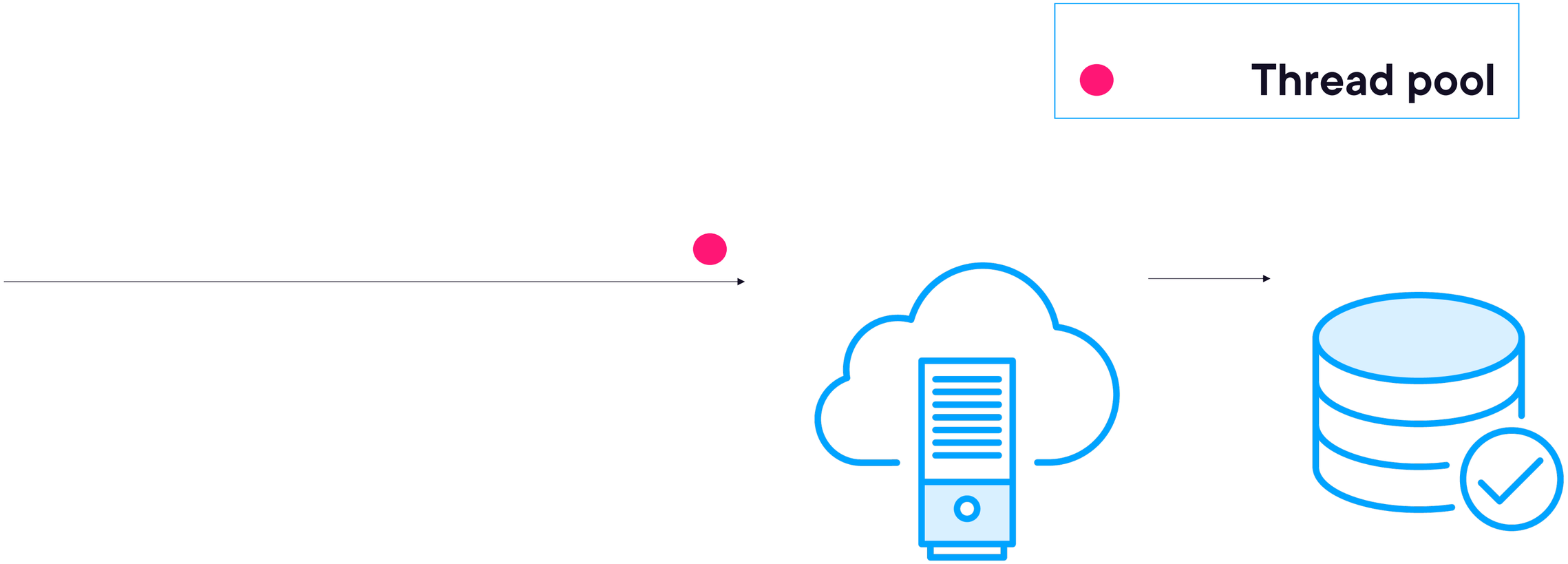
Freeing up threads so they can be used for other tasks, which improves the scalability of your application

Handling Synchronous Requests

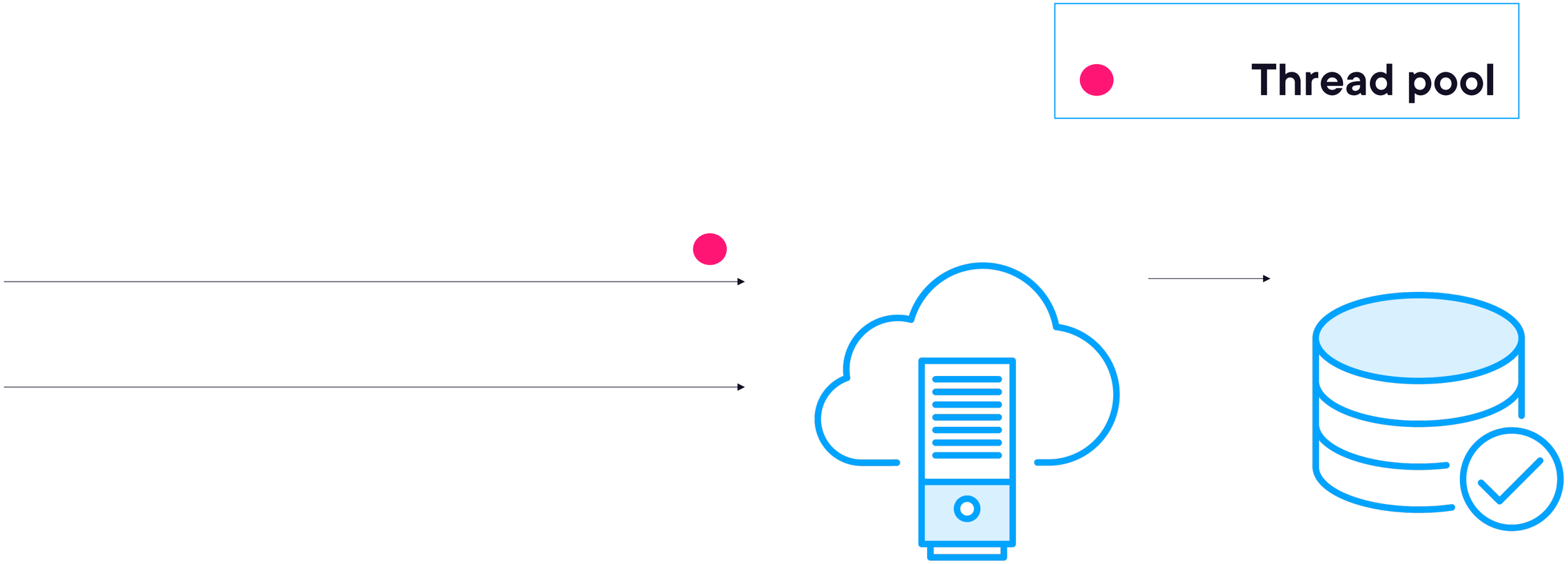
Handling Synchronous Requests



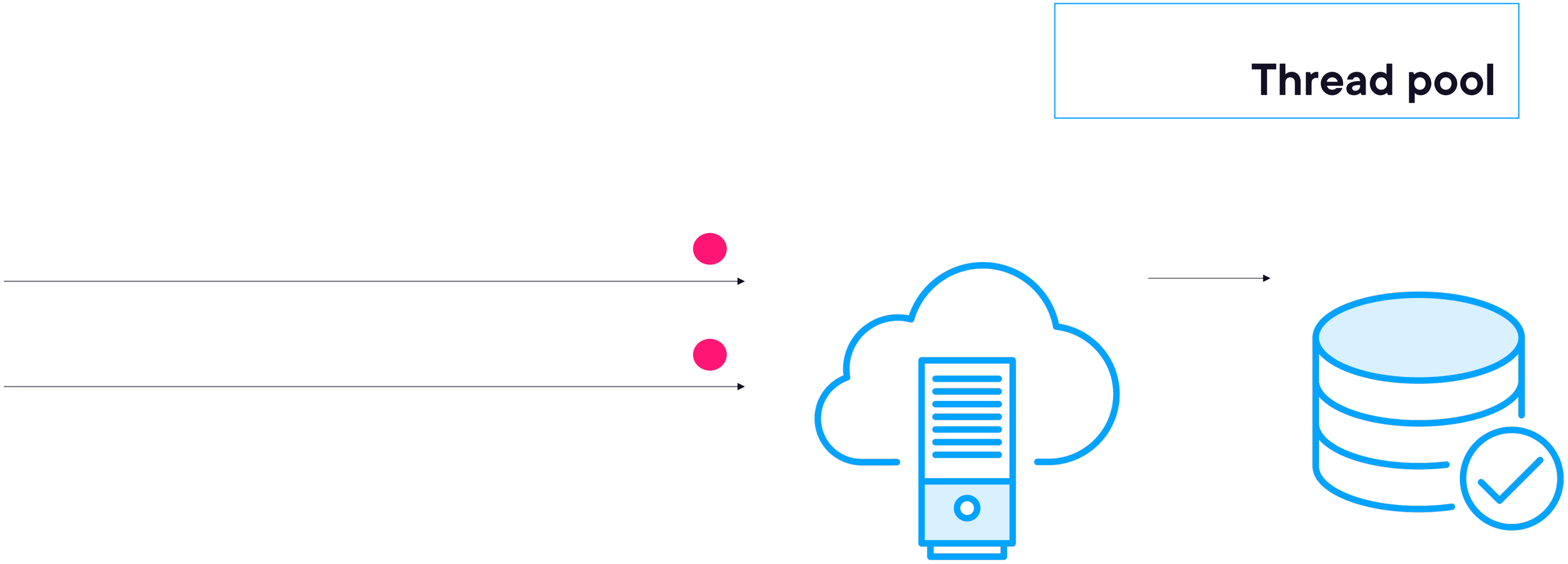
Handling Synchronous Requests



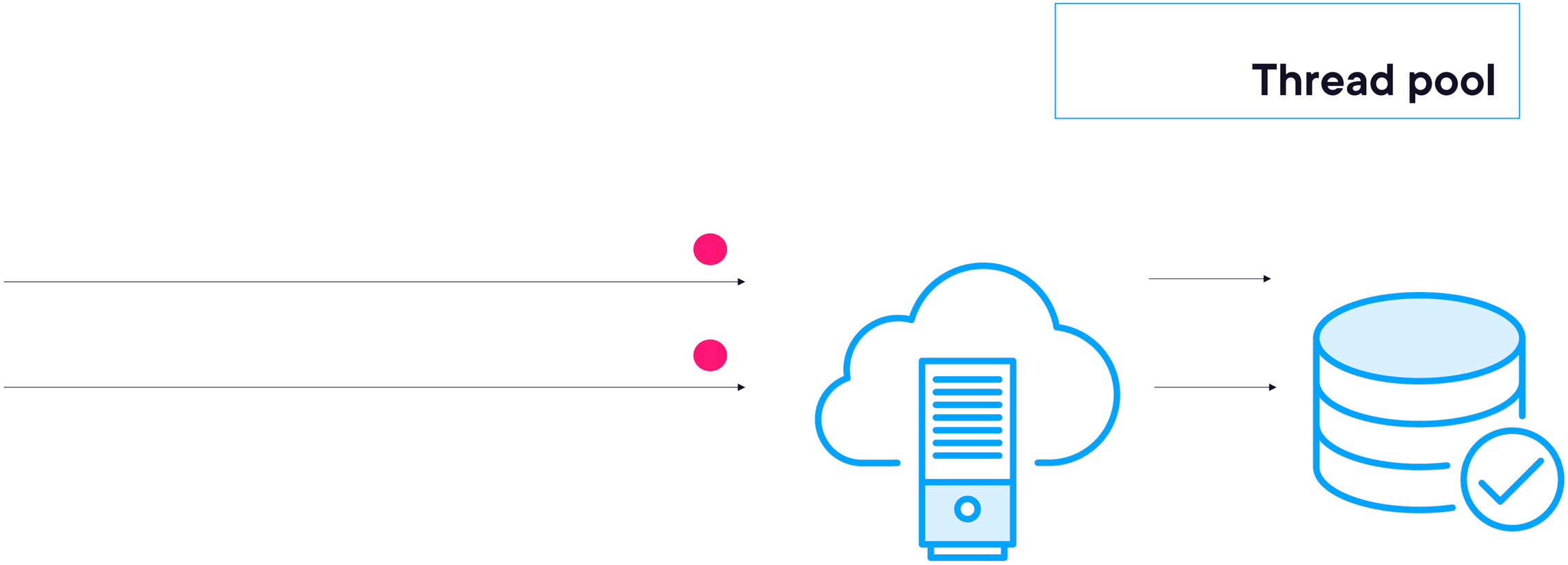
Handling Synchronous Requests



Handling Synchronous Requests



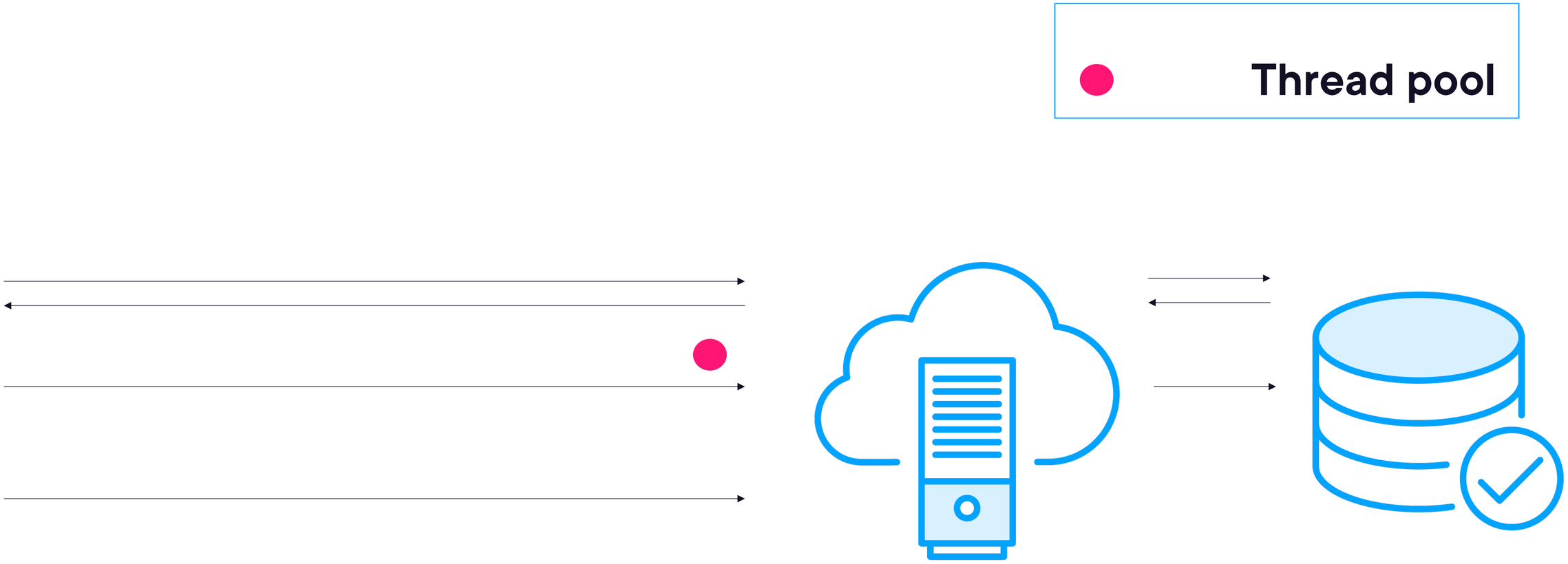
Handling Synchronous Requests



Handling Synchronous Requests

Handling Synchronous Requests

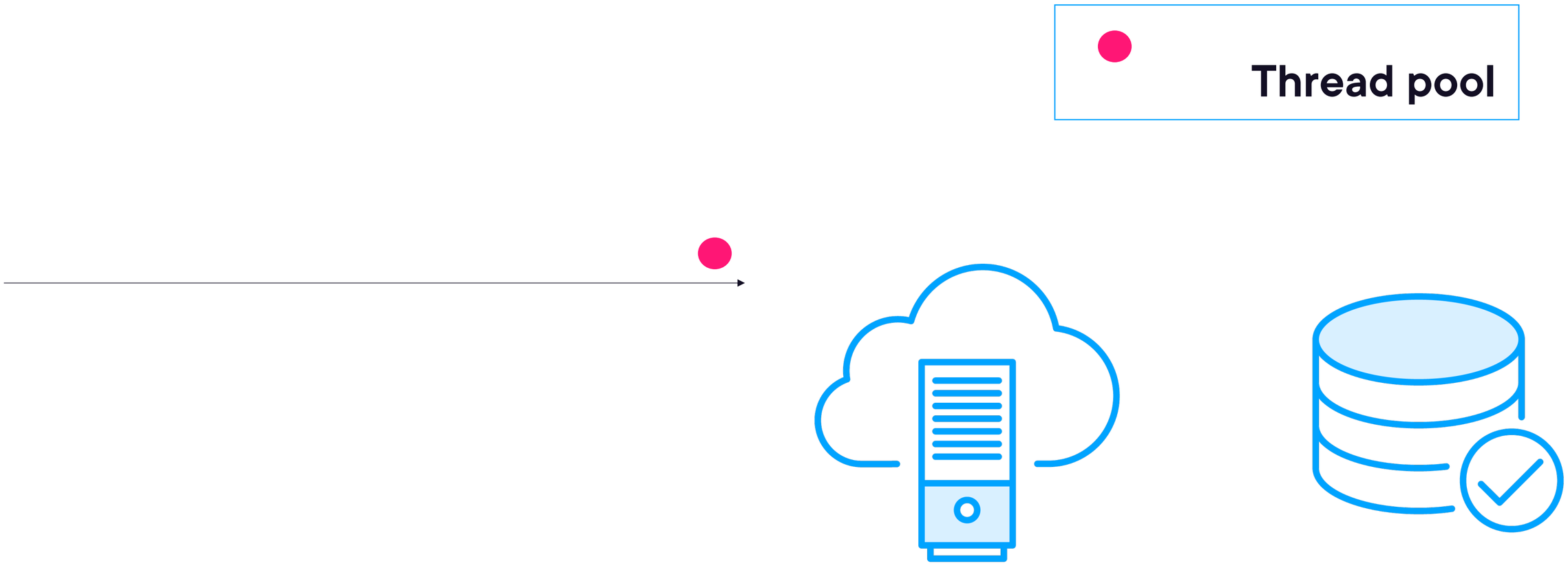
Handling Synchronous Requests



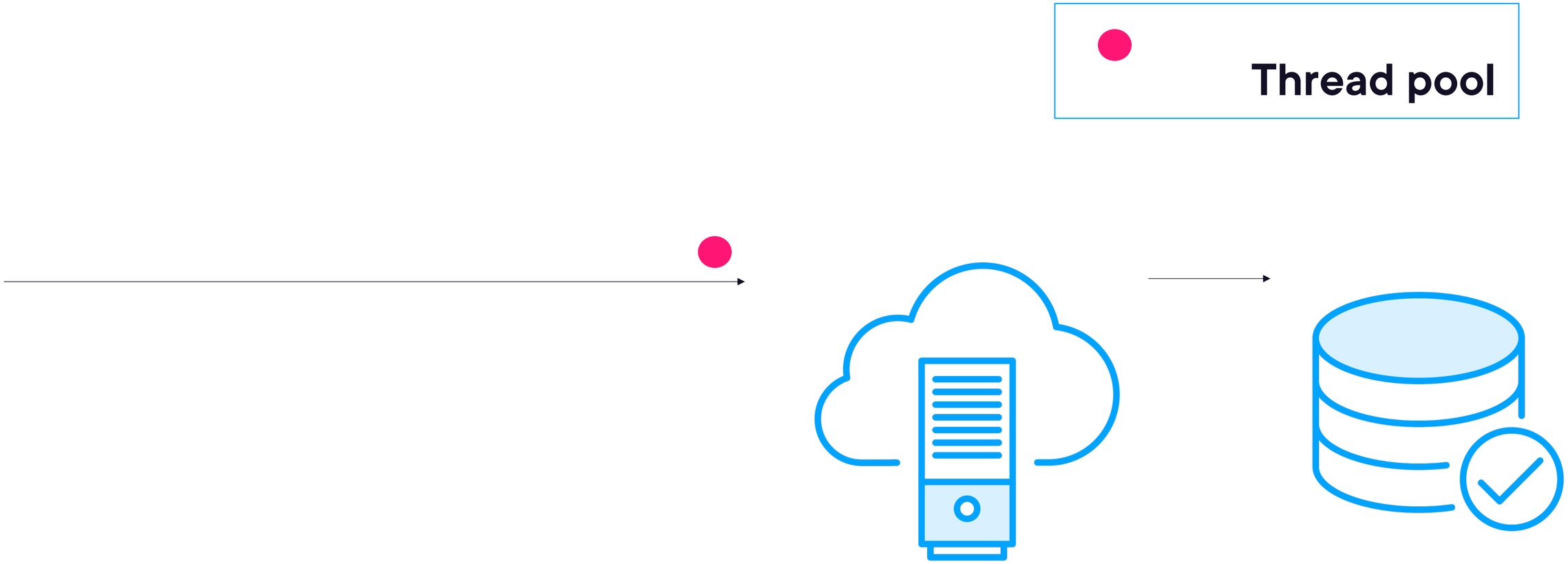
Handling Synchronous Requests

Handling Asynchronous Requests

Handling Asynchronous Requests



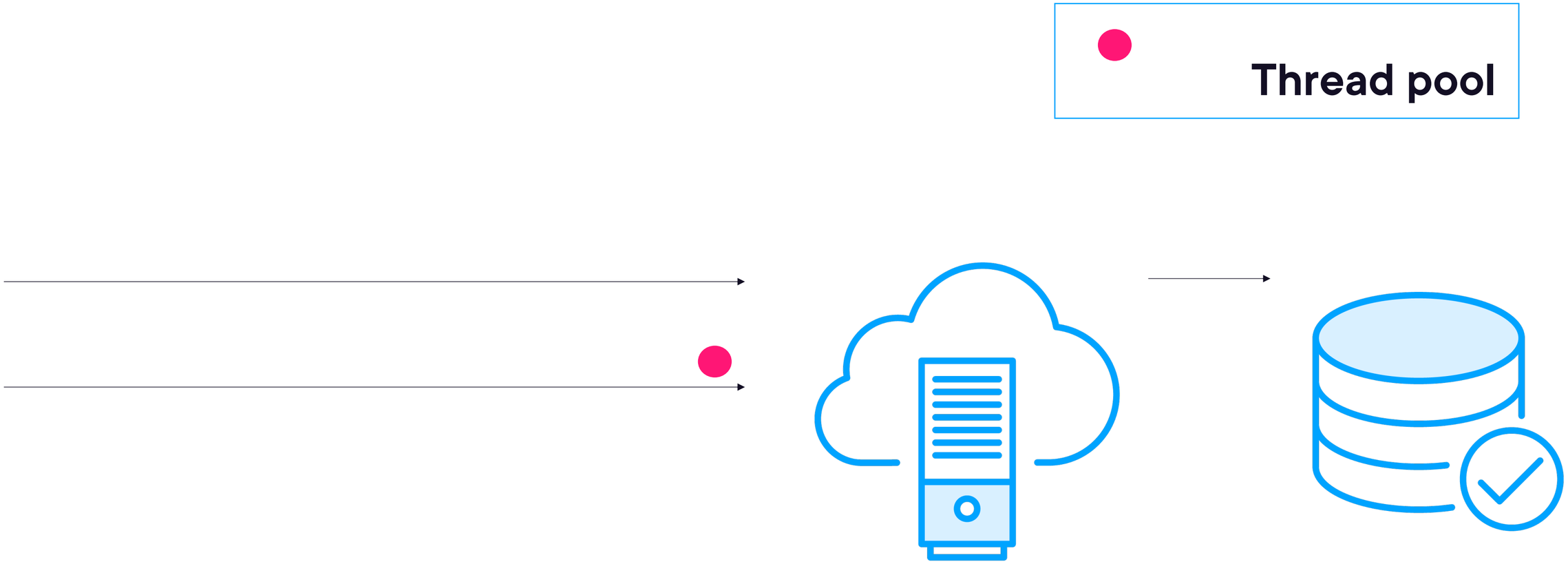
Handling Asynchronous Requests



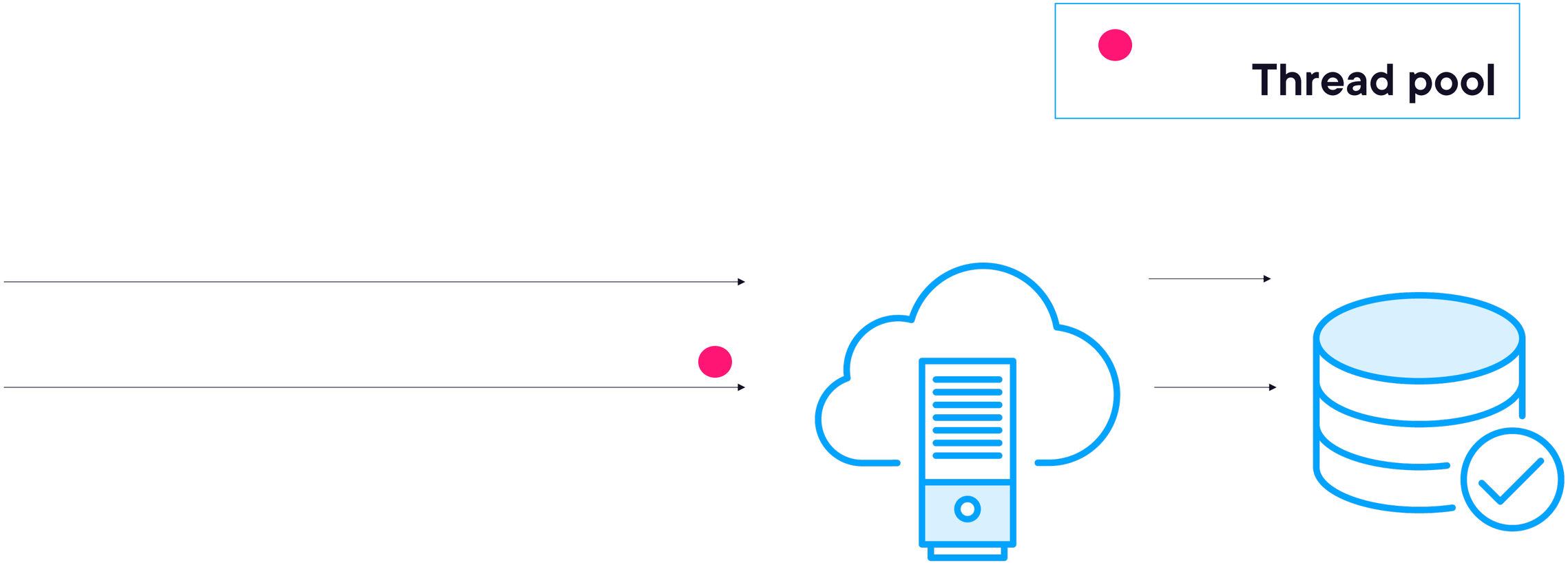
Handling Asynchronous Requests

Handling Asynchronous Requests

Handling Asynchronous Requests



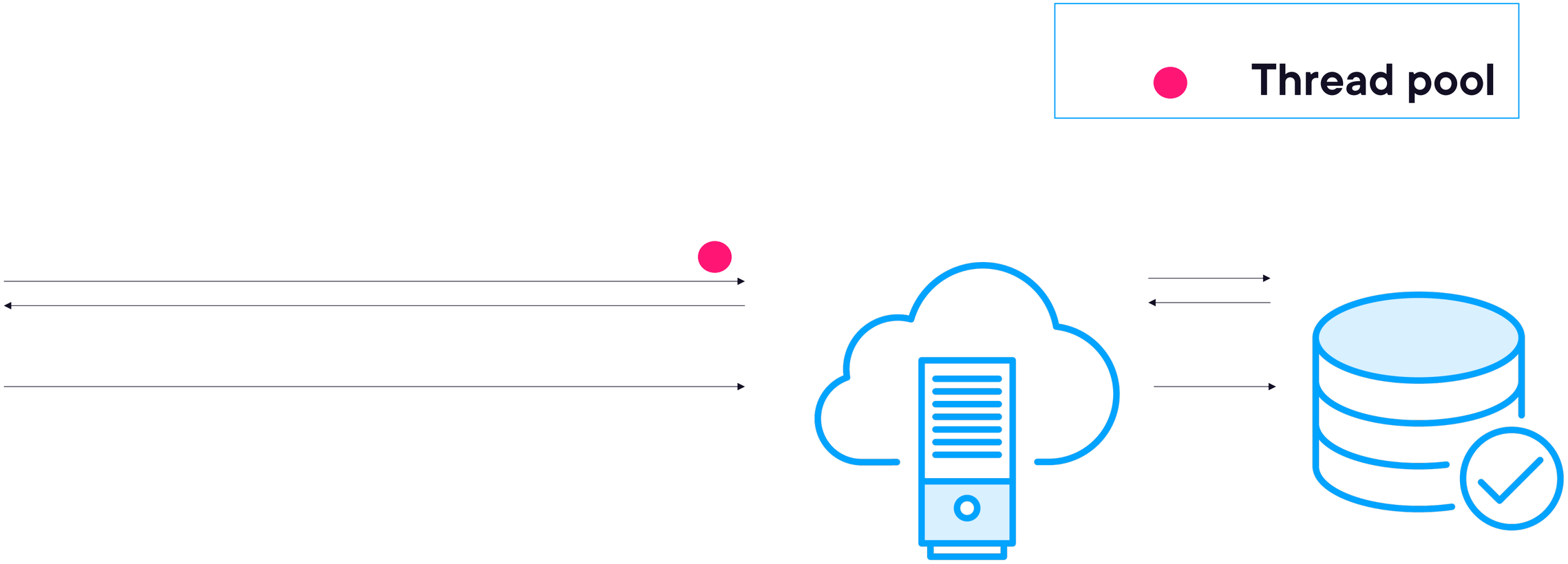
Handling Asynchronous Requests



Handling Asynchronous Requests

Handling Asynchronous Requests

Handling Asynchronous Requests



Handling Asynchronous Requests

Handling Asynchronous Requests

Handling Asynchronous Requests

Demo

Introducing the repository pattern (part 2)

Demo

Returning data from the repository when requesting resources (part 1)

Demo

Using AutoMapper to map between entities and DTOs

Demo

Returning data from the repository when requesting resources (part 2)

Demo

Creating a resource

Demo

Updating a resource

Demo

Partially updating a resource

Demo

Deleting a resource

Summary

The repository pattern is an abstraction that reduces complexity and aims to make the code, safe for the repository implementation, persistence ignorant

Summary

Using async code for I/O operations ensures threads can be freed up faster, resulting in improved scalability

Using AutoMapper greatly reduces error-prone mapping code

Up Next:

Searching, Filtering and Paging Resources