Querying Data with Repositories

Andrew Morgan
INDEPENDENT CONSULTANT
@mogronalol

Overview

Overview of the repository pattern

Spring Data repositories

- CRUD repository
- Derived queries
- Pagination and sorting
- Custom repositories

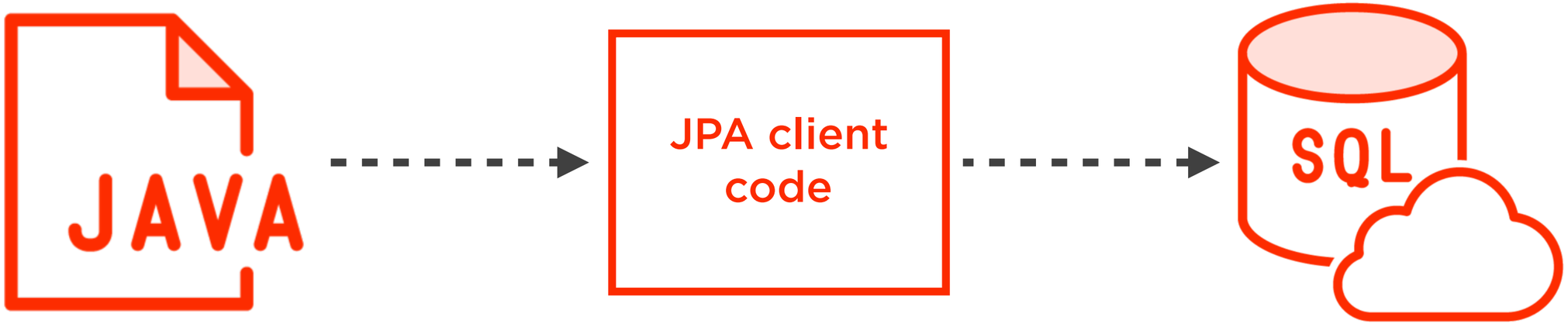
Swapping modules

Repository pattern

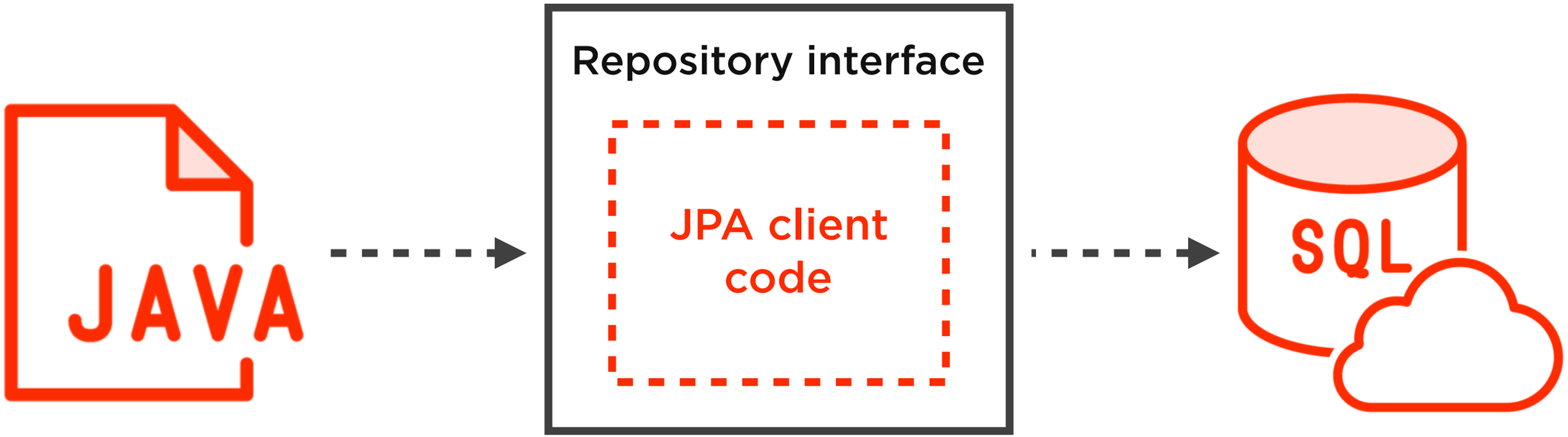
A persistence ignorant data access abstraction

Repository Pattern Overview

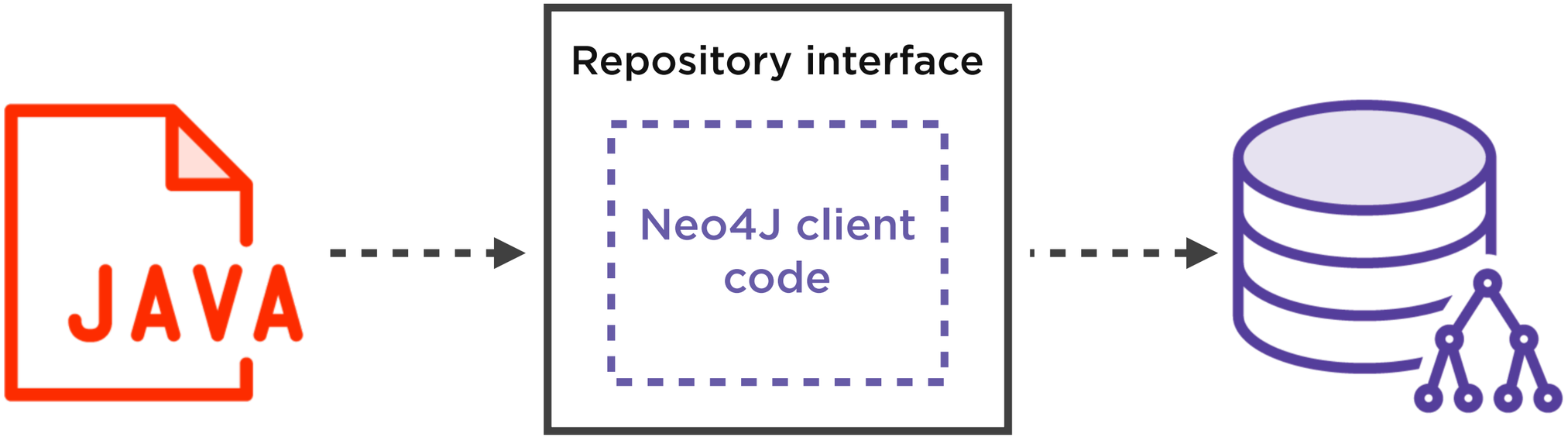
Repository Pattern Overview



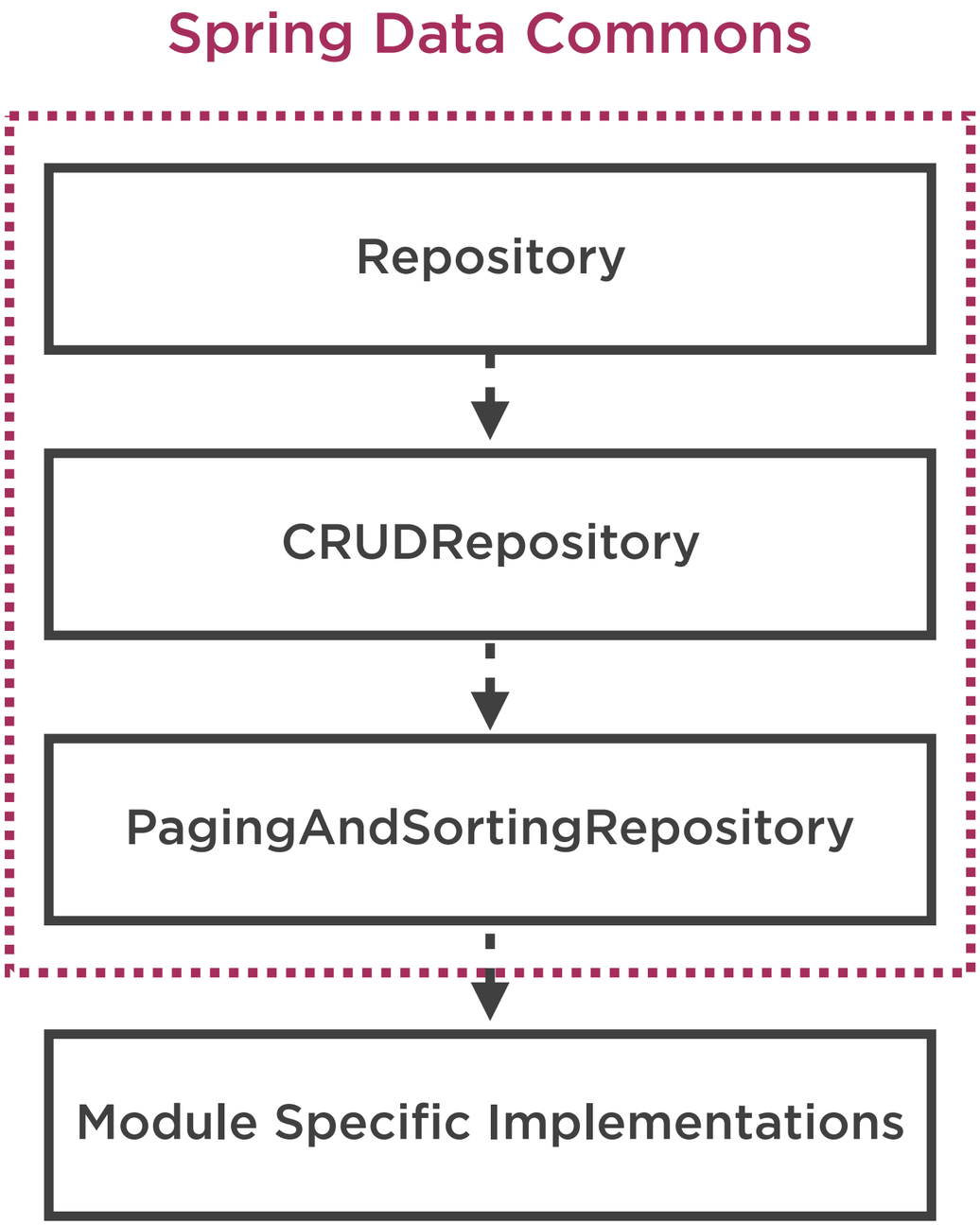
Repository Pattern Overview



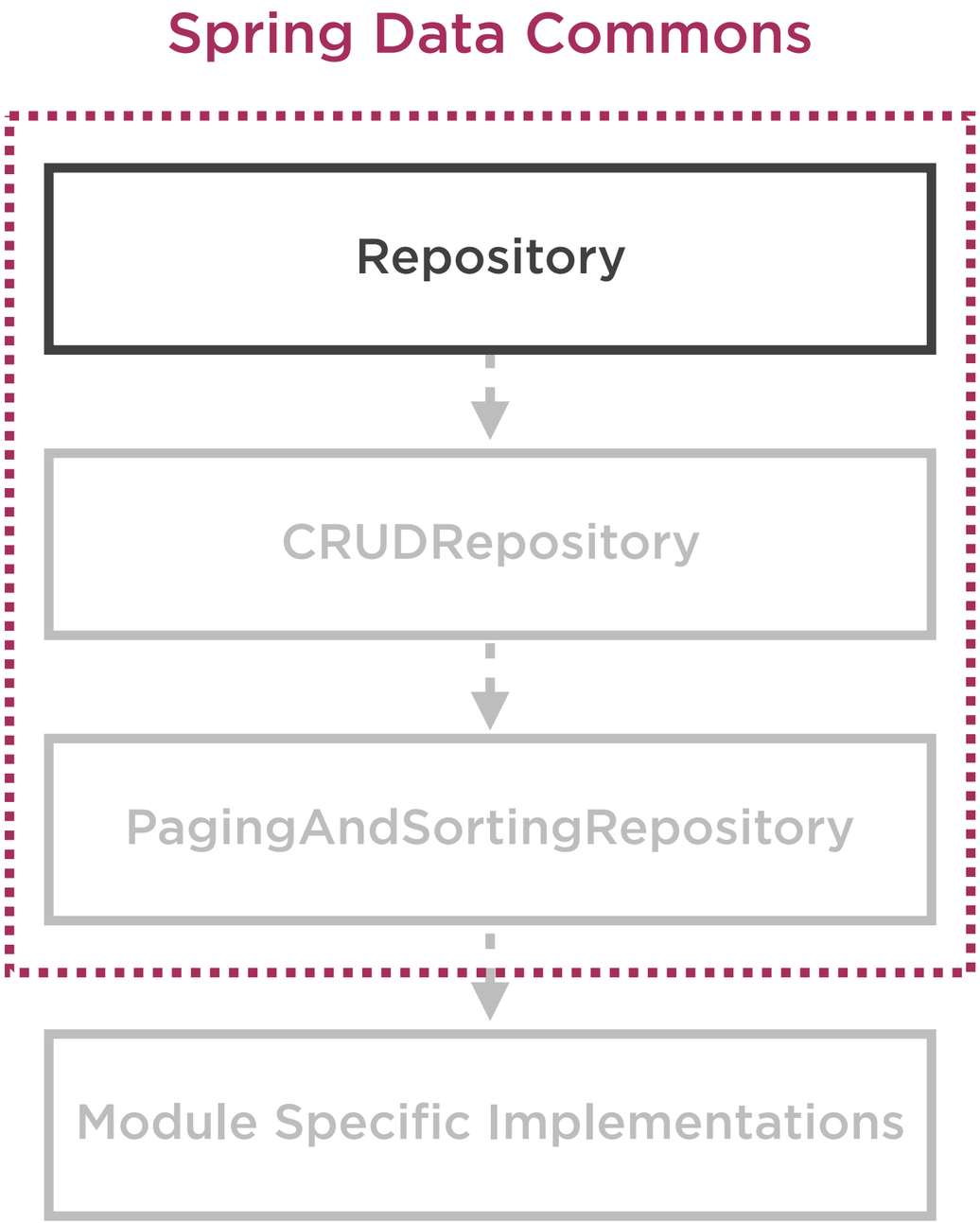
Repository Pattern Overview



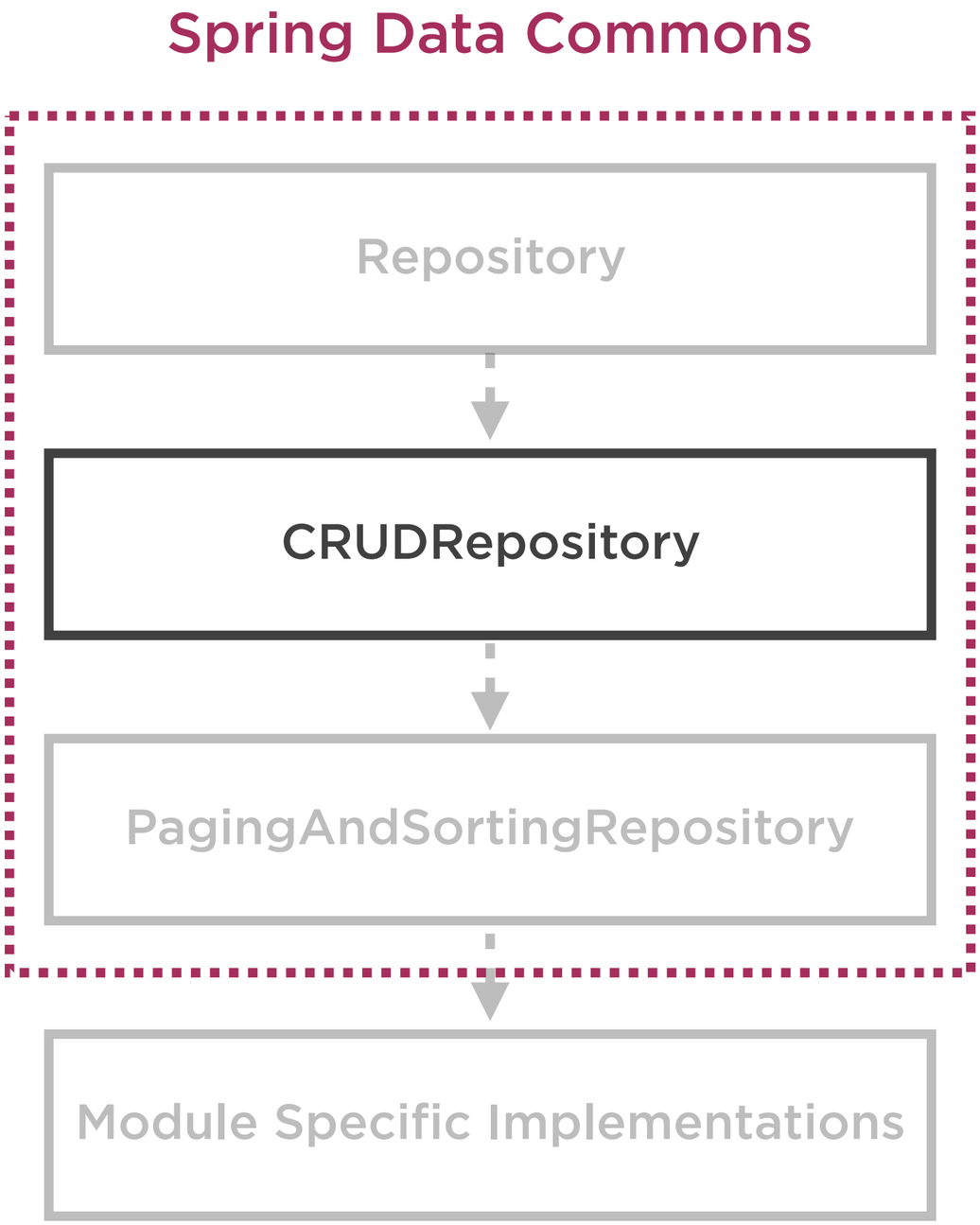
Spring Data Repository Hierarchy



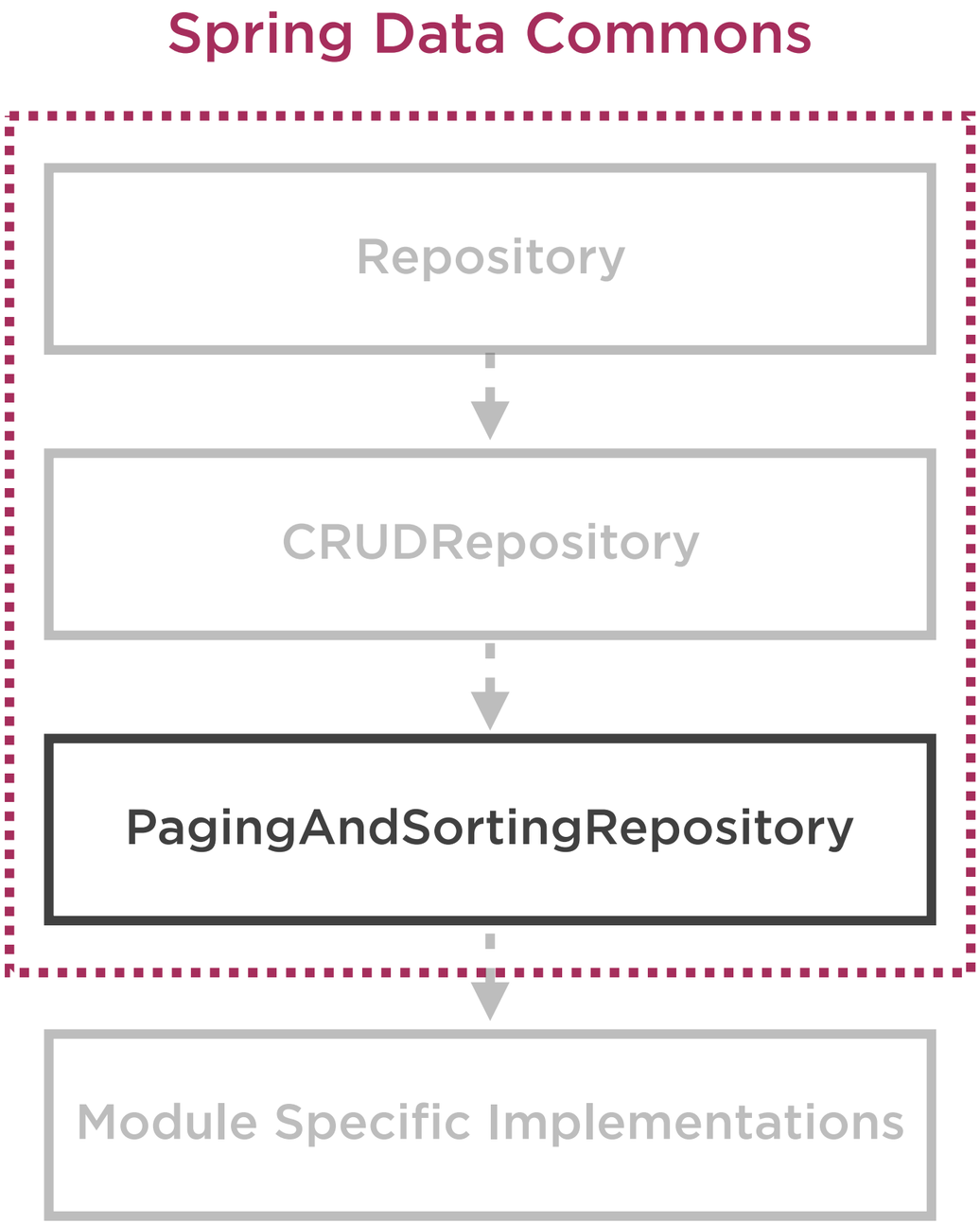
Spring Data Repository Hierarchy



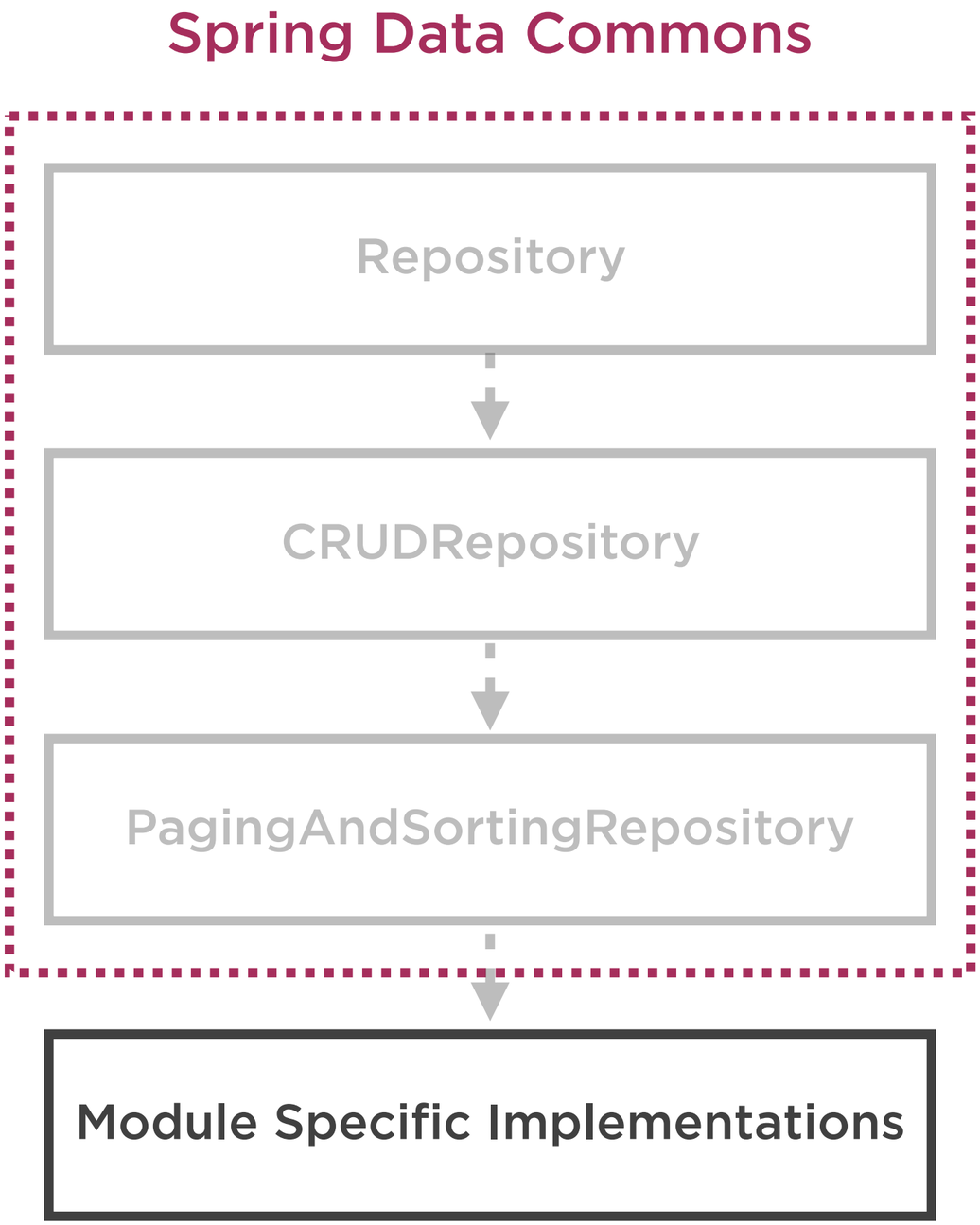
Spring Data Repository Hierarchy



Spring Data Repository Hierarchy



Spring Data Repository Hierarchy



Root Repository Interface

```
public interface Repository<T, ID> { }
```

Root Repository Interface

```
public interface Repository<T, ID> { }
```

Type of entity to persist

Root Repository Interface

ID type for the entity

```
public interface Repository<T, ID> { }
```

Type of entity to persist

```java
public interface CrudRepository
  <T, ID> extends Repository<T, ID>

    <S extends T> S save(S entity);

    Optional<T> findById(ID id);

    void delete(T entity);

     // more methods
}
```

```java
public interface CrudRepository
  <T, ID> extends Repository<T, ID>

    <S extends T> S save(S entity);

    Optional<T> findById(ID id);

    void delete(T entity);

     // more methods
}
```

◀ **Creating or updating**

```java
public interface CrudRepository
  <T, ID> extends Repository<T, ID>

    <S extends T> S save(S entity);

    Optional<T> findById(ID id);

    void delete(T entity);

     // more methods
}
```

- Creating or updating
- **Reading**

```java
public interface CrudRepository
  <T, ID> extends Repository<T, ID>

    <S extends T> S save(S entity);

    Optional<T> findById(ID id);

    void delete(T entity);

     // more methods
}
```

- Creating or updating
- Reading
- **Deleting**

Demo

Creating our first CRUDRepository

Querying the database using the repository

Query Boilerplate

```
List<Customer> findByName(String name);
```

Query Boilerplate

```
List<Customer> findByName(String name);
```

```
entityManager.createNativeQuery(
  "SELECT * FROM customers WHERE name = 'Andrew' = ?")
    .setParameter(1, origin)
    .getResultList();
```

Deriving Queries

```
List<Customer> findByName(String name);
```

Deriving Queries

```
List<Customer> findByName(String name);
```

SELECT

Deriving Queries

WHERE NAME =

```
List<Customer> findByName(String name);
```

SELECT

Deriving Queries

WHERE NAME =

```
List<Customer> findByName(String name);
```

SELECT

VALUE

Deriving Queries

Map results to me

WHERE NAME =

```
List<Customer> findByName(String name);
```

SELECT

VALUE

Derived Query Examples

Example	Description
`List<Customer> findByFirstNameAndLastName(String name, String lastName);`	And queries
`List<Customer> findByFirstNameIgnoreCase(String name, String lastName);`	Case insensitivity
`List<Customer> findByFirstNameOrderByCreated(String name);`	Ordering
`void deleteByName(String name)`	Deletion
`List<Customer> findByAgeGreaterThan(int age);`	Less than, greater than, etc.

Demo

Creating some derived queries

Verifying they function with a test

Pagination and Sorting

Pagination and Sorting

Pagination and Sorting

Pagination and Sorting

Pagination and Sorting

```
SELECT * FROM orders ORDER BY order_date
LIMIT 50 OFFSET 50
```

Pagination and Sorting

```
SELECT * FROM orders ORDER BY order_date
LIMIT 50 OFFSET 50
```

Paging and Sorting Dissected

```
Page<Flight> findByName(String name, Pageable pageable, Sort sort);
```

Paging and Sorting Dissected

Combine with
derived queries

```
Page<Flight> findByName(String name, Pageable pageable, Sort sort);
```

Paging and Sorting Dissected

Combine with
derived queries

```
Page<Flight> findByName(String name, Pageable pageable, Sort sort);
```

```
PageRequest.of(page, pageSize);
```

Paging and Sorting Dissected

Combine with derived queries

```
Sort.by("column");
```

```
Page<Flight> findByName(String name, Pageable pageable, Sort sort);
```

```
PageRequest.of(page, pageSize);
```

PagingAndSortingRepository

```
public interface PagingAndSortingRepository<T, ID> extends
    CrudRepository<T, ID> {

      Iterable<T> findAll(Sort sort);

     Page<T> findAll(Pageable pageable);
}
```

PagingAndSortingRepository

```
public interface PagingAndSortingRepository<T, ID> extends
    CrudRepository<T, ID> {

      Iterable<T> findAll(Sort sort);

      Page<T> findAll(Pageable pageable);
}
```

Default sort method

PagingAndSortingRepository

```
public interface PagingAndSortingRepository<T, ID> extends
    CrudRepository<T, ID> {

      Iterable<T> findAll(Sort sort);

      Page<T> findAll(Pageable pageable);
}
```

Default sort method

Default page method

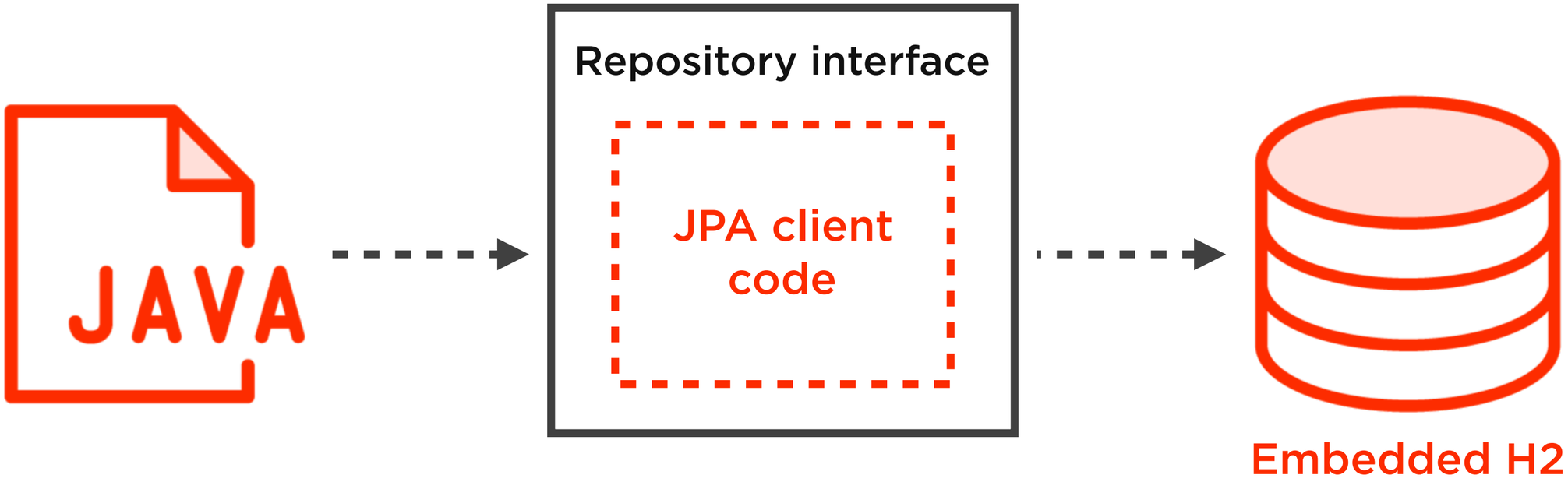
Demo

Working with the PagingAndSortingRepository

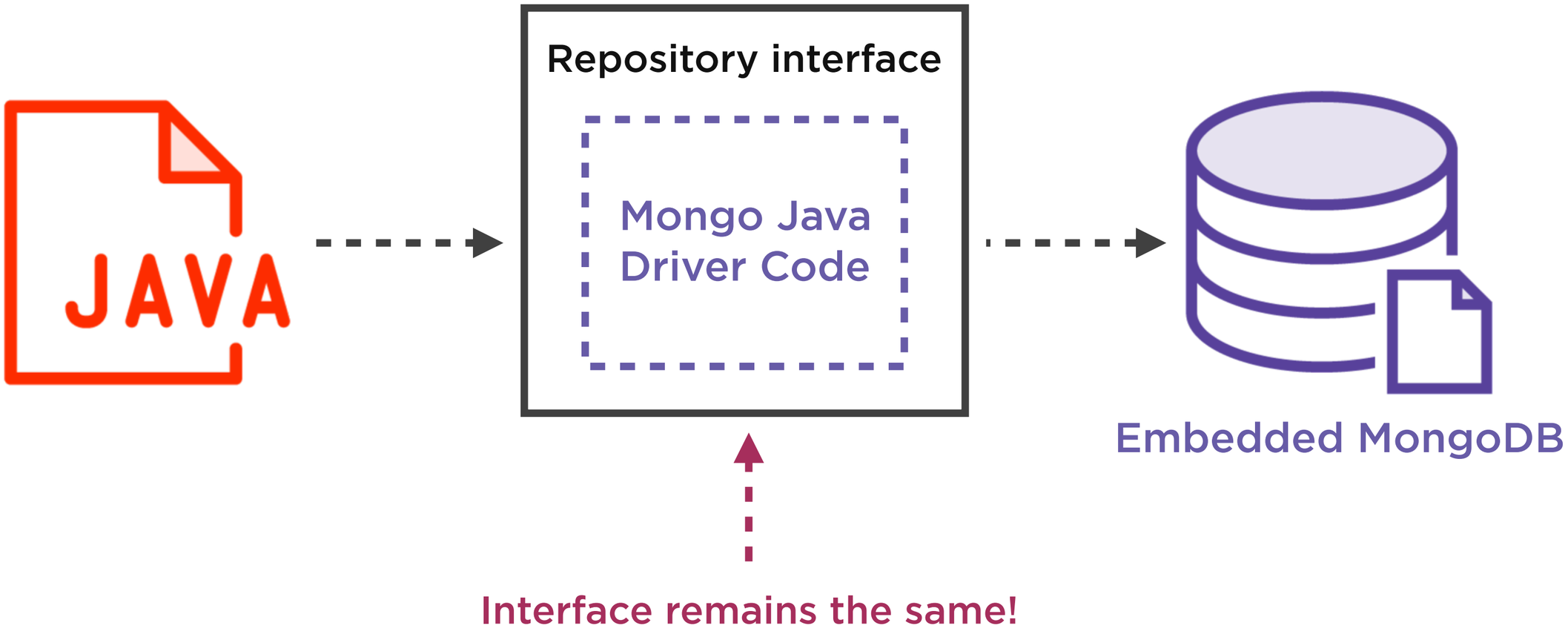
Using the page and sort abstractions

Enriching our derived queries with paging and sorting

Our Application So Far



Migrating to MongoDB



Demo

Switching to MongoDB

- Changing Maven dependencies
- Switching to embedded Mongo
- Remapping our entity
- Verifying our tests still pass

Custom Implementation Gotcha

```
public interface UserRepo extends CrudRepository<User, Long> {

}
```

Custom Implementation Gotcha

```
public interface UserRepo extends CrudRepository<User, Long> {

  List<User> cannotImplementMe(String foo);

}
```

Custom Implementation the Right Way

```
public interface CustomRepository {


}
```

Custom Implementation The Right Way

```
public interface CustomRepository {

  List<User> canImplementMe(String foo);

}
```

Custom Implementation Composition

```
public interface UserRepo extends CrudRepository<User, Long>,
  CustomRepository {


}
```

Demo

Create a custom repository interface

Implement our repository

Verify it functions with a test

Summary

The repository interface provides us with a data access abstraction

We rarely have to write our own queries, reducing boilerplate

- CRUD is provided
- Advanced queries are derived
- Pagination & sorting are abstracted

We can fall back to creating custom implementations of we need to