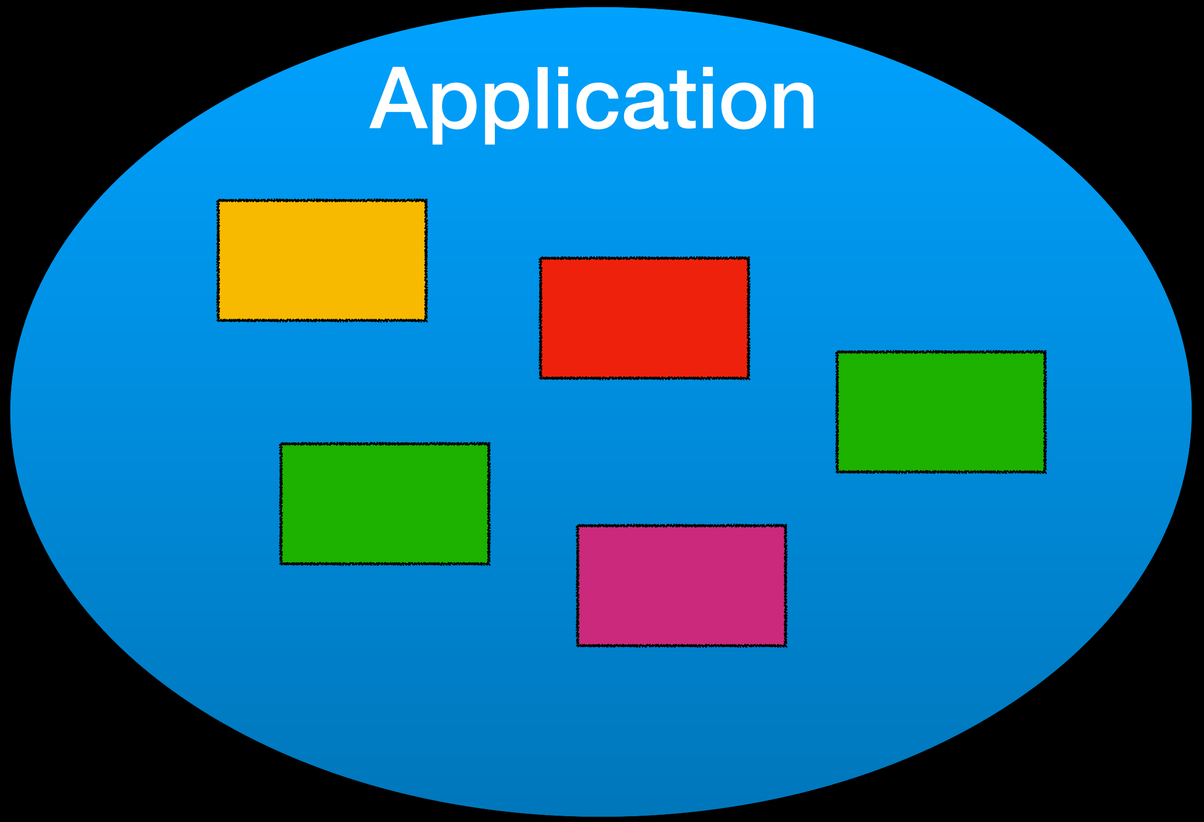
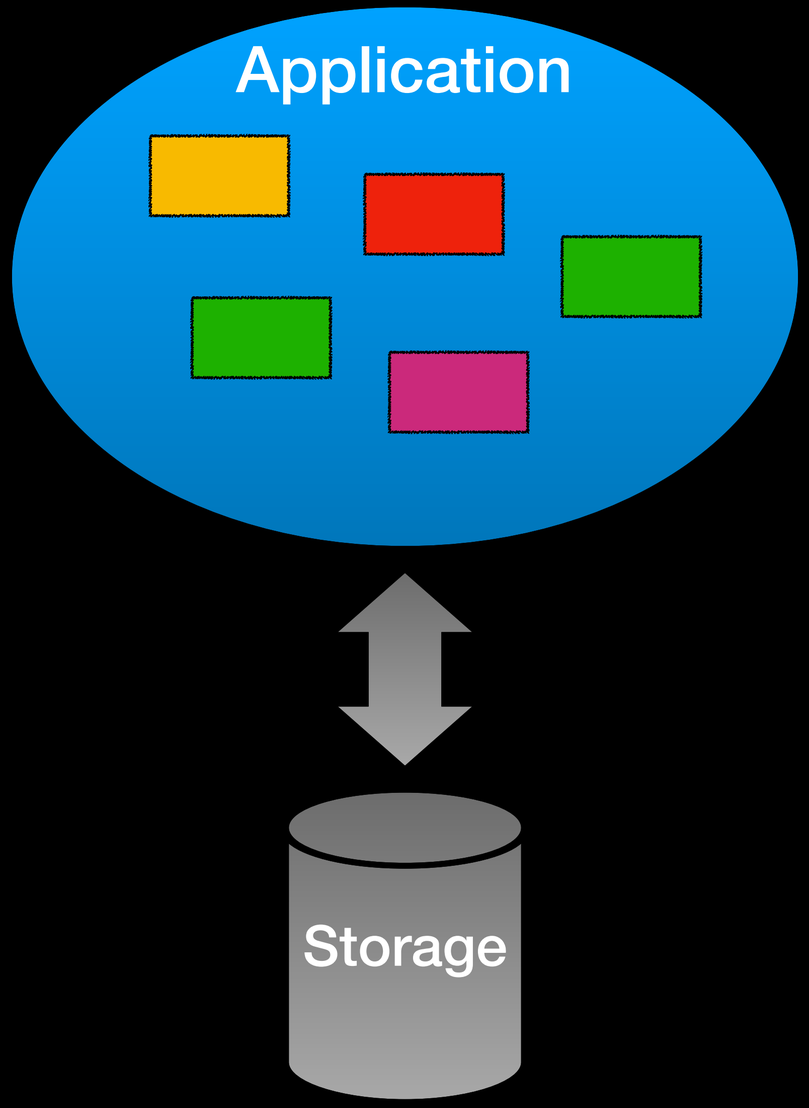
lambda DAYS
27-28 MAY 2024
KRAKÓW | POLAND

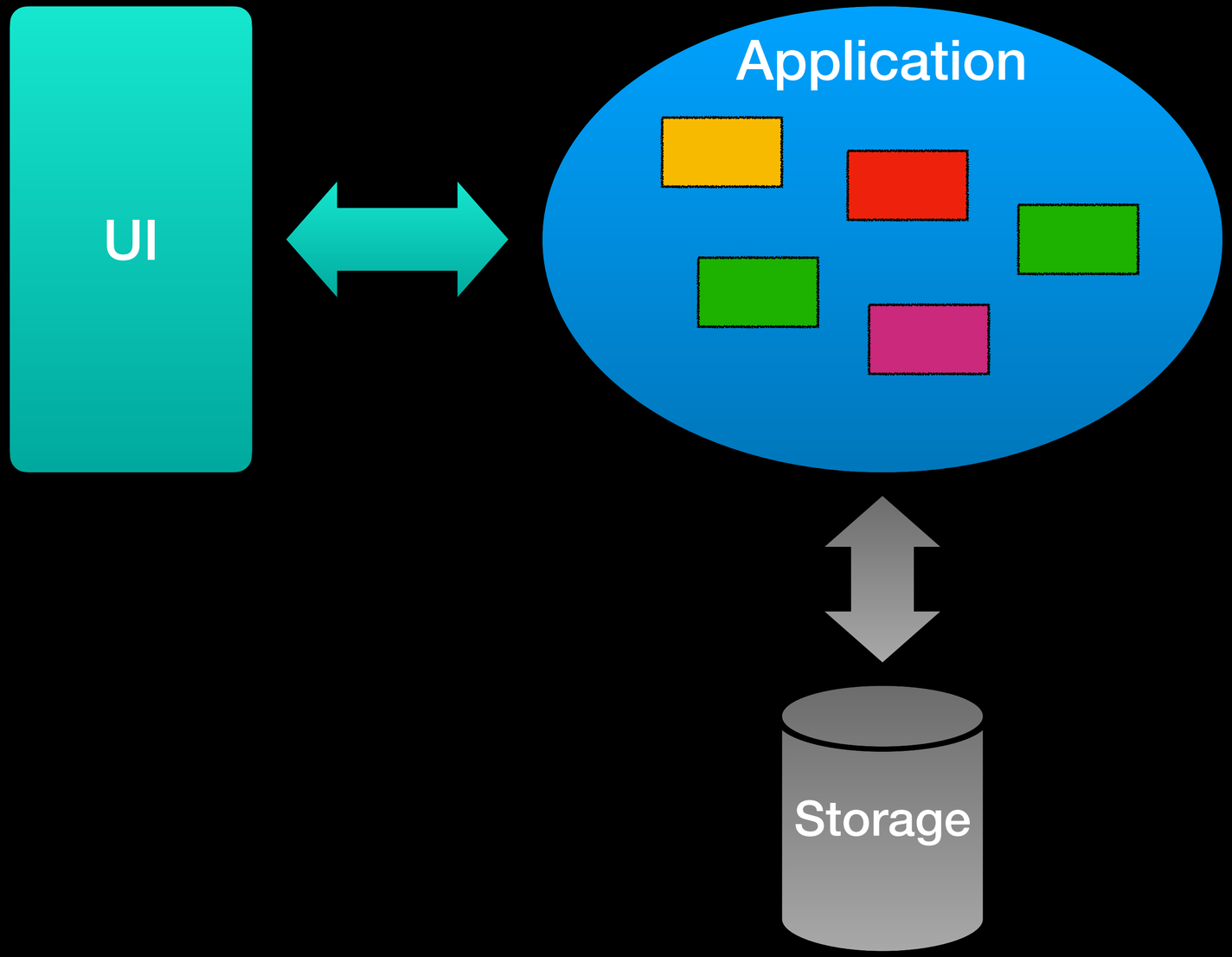
Marcus Crestani
Active Group GmbH

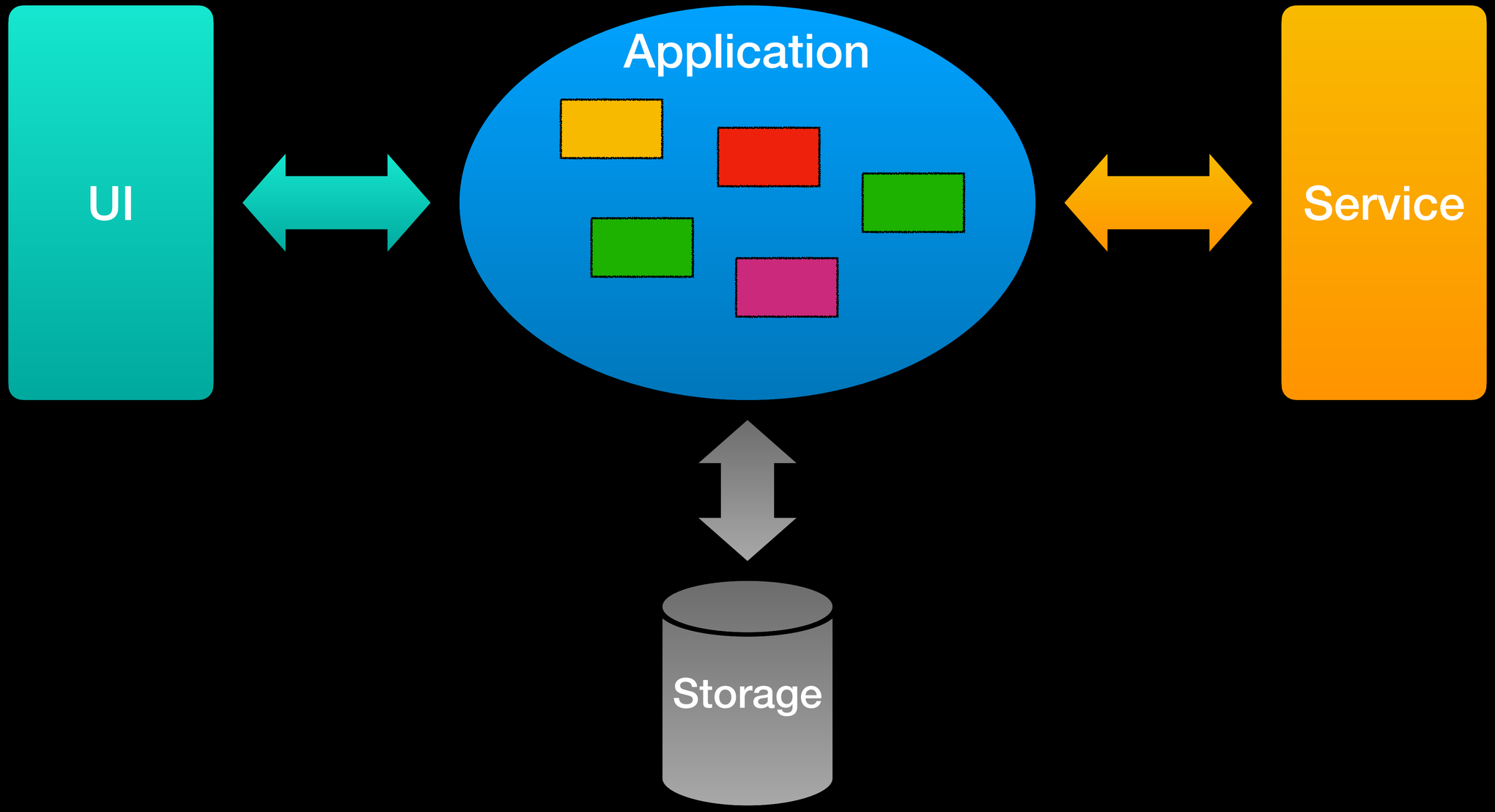
Bidirectional Data Transformations

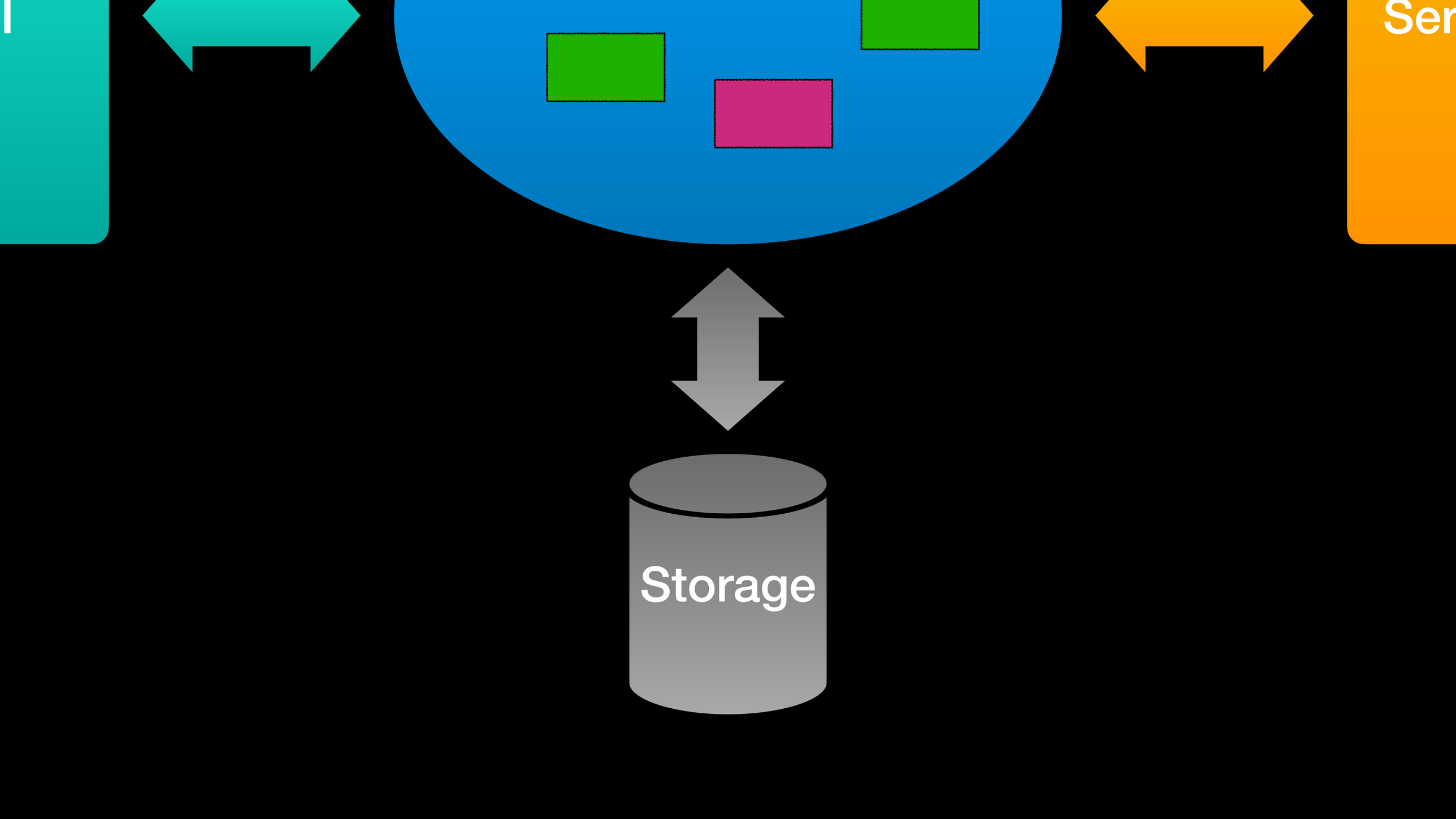
@ctive group

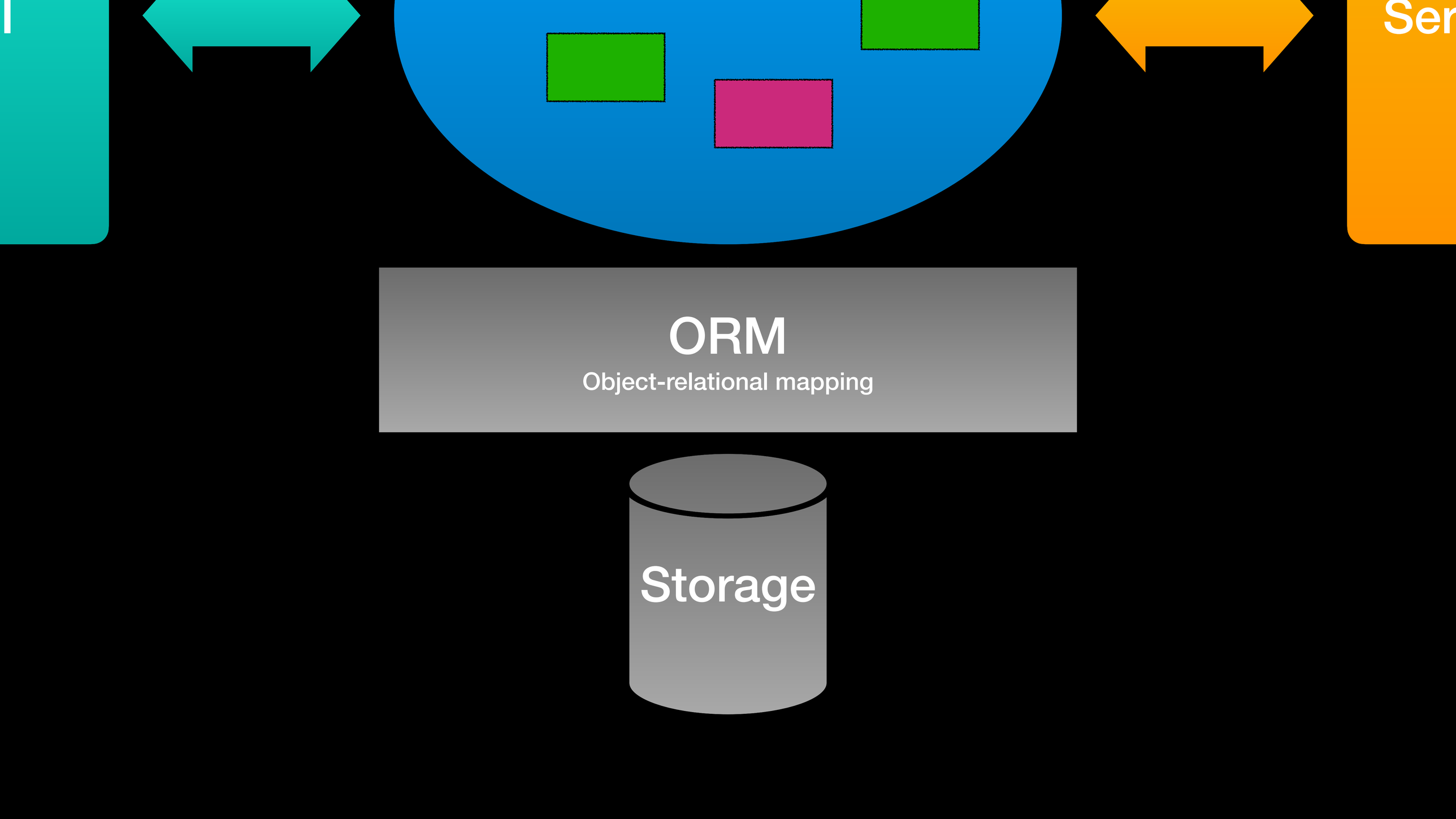

Application

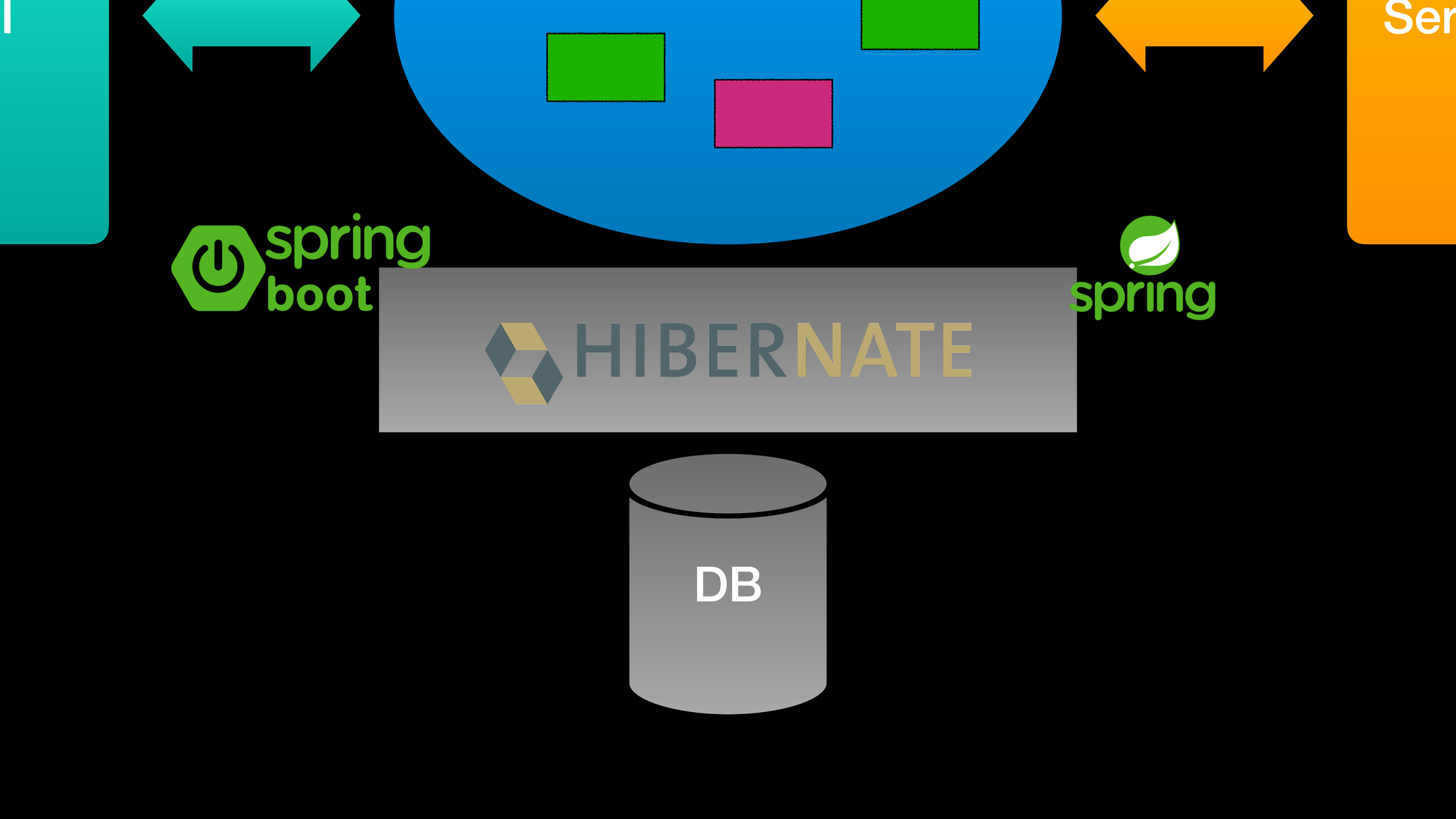

Application
Storage


UI
Application
Storage


UI
Application
Service
Storage


Storage


ORM
Object-relational mapping
Storage


spring
boot
HIBERNATE
spring
DB

DB

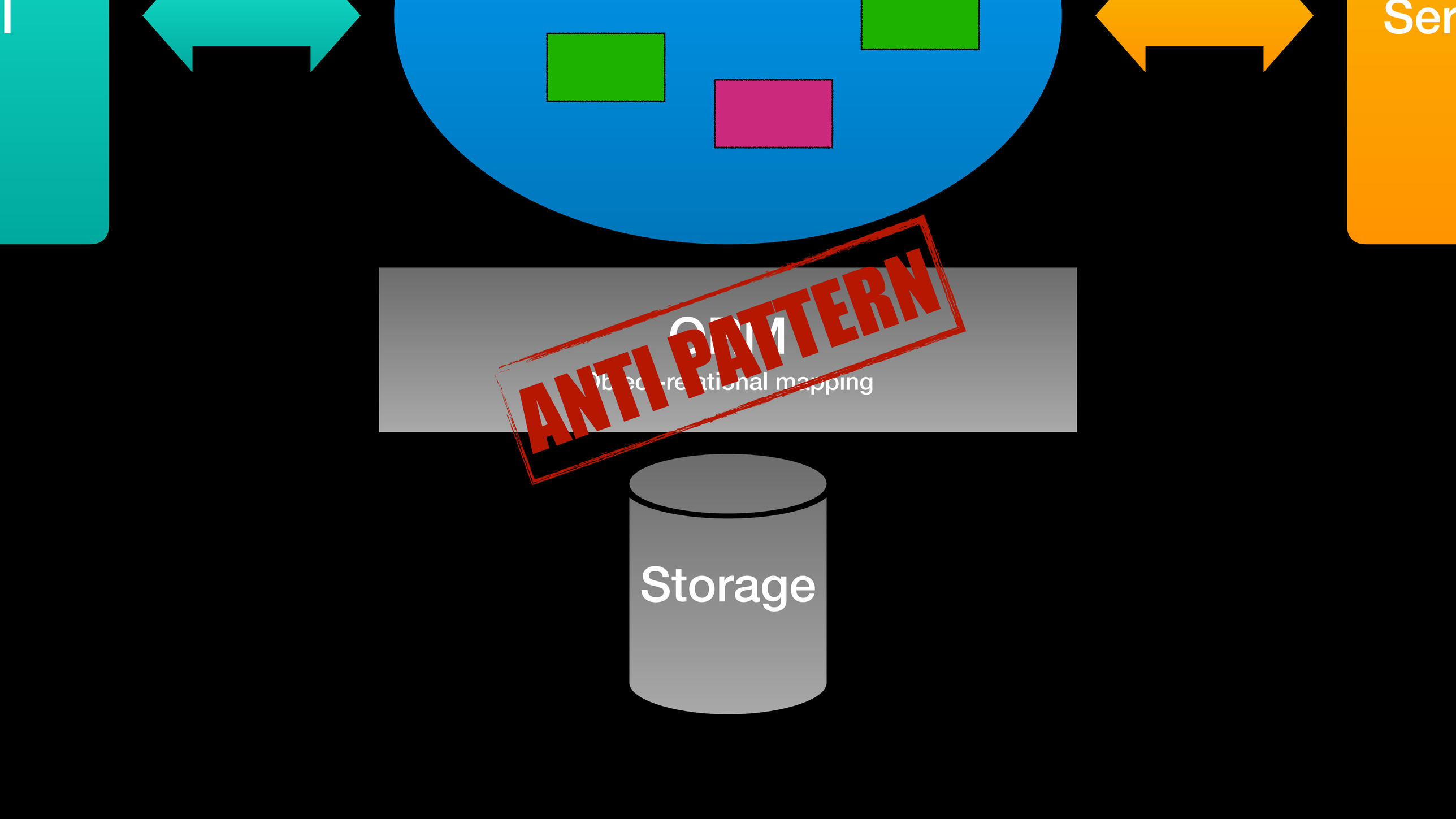
```java
@Entity
public class User {
  @Id
  @GeneratedValue
  private Long id;
  private String name;
  private String password;
  private List<Role> roles;

  // standard constructors, getters, and setters
}

@Entity
public class Role {
  @Id
  @GeneratedValue
  private Long id;
  private String name;

  // standard constructors, getters, and setters
}
```

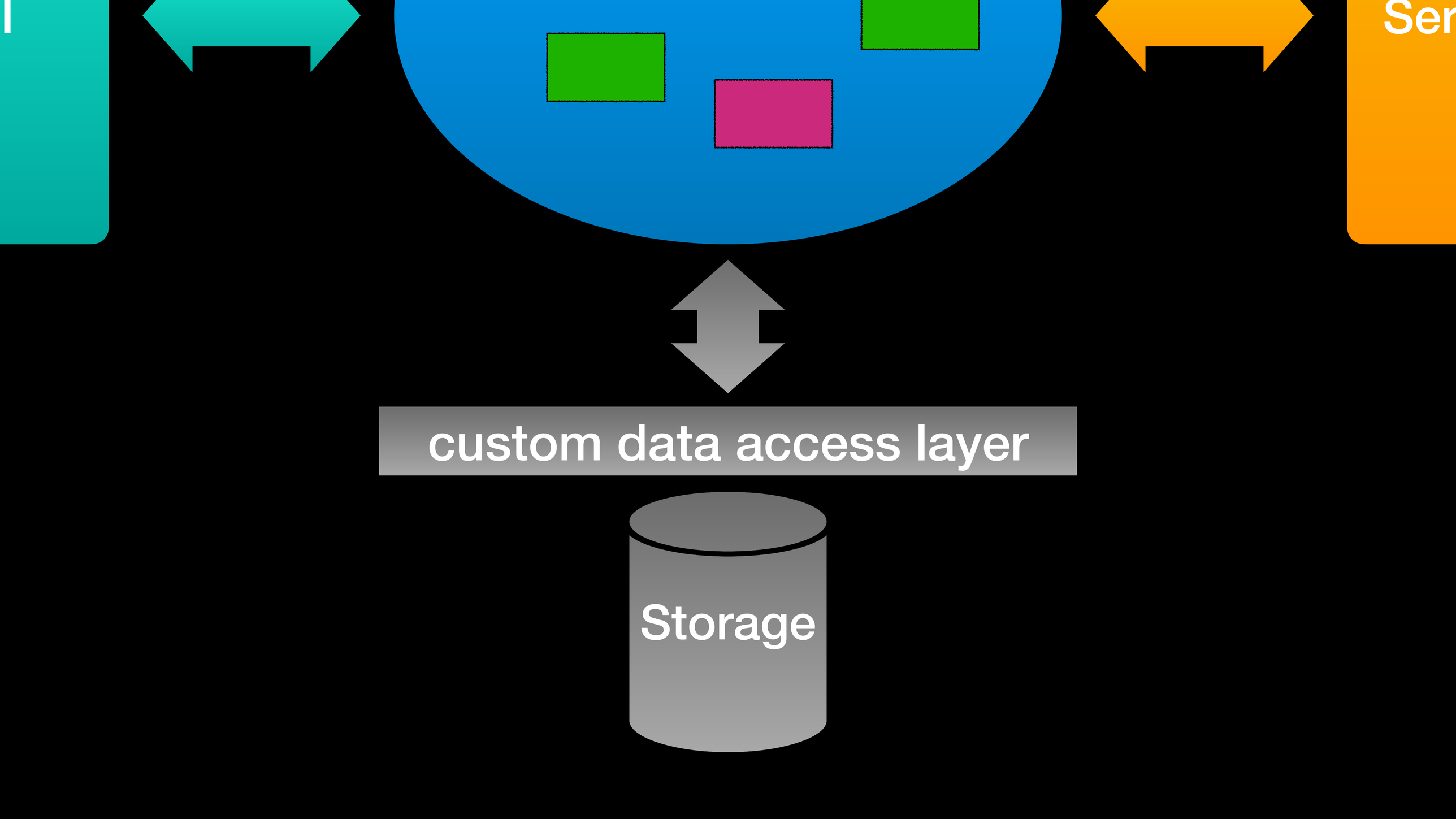
ORM

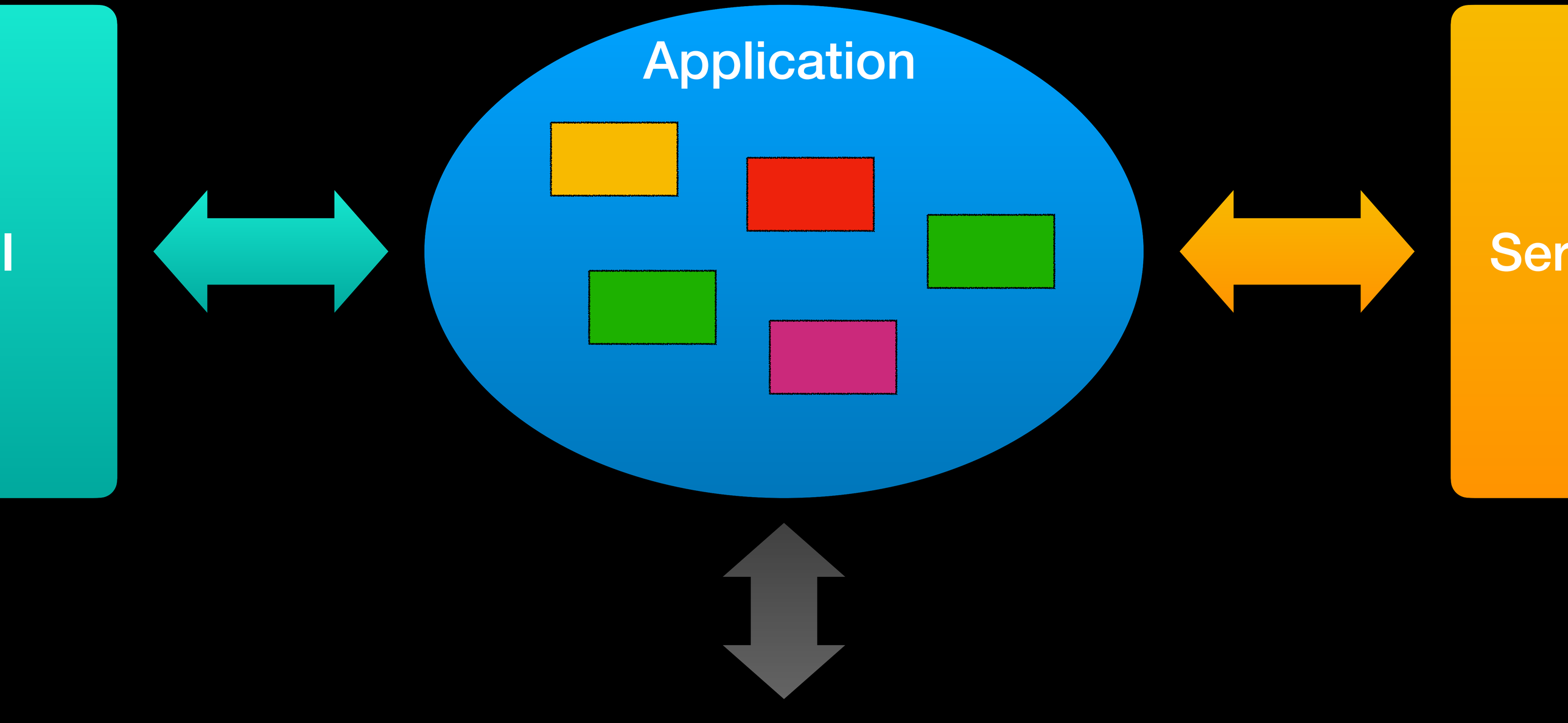
- short development time
- 🤷

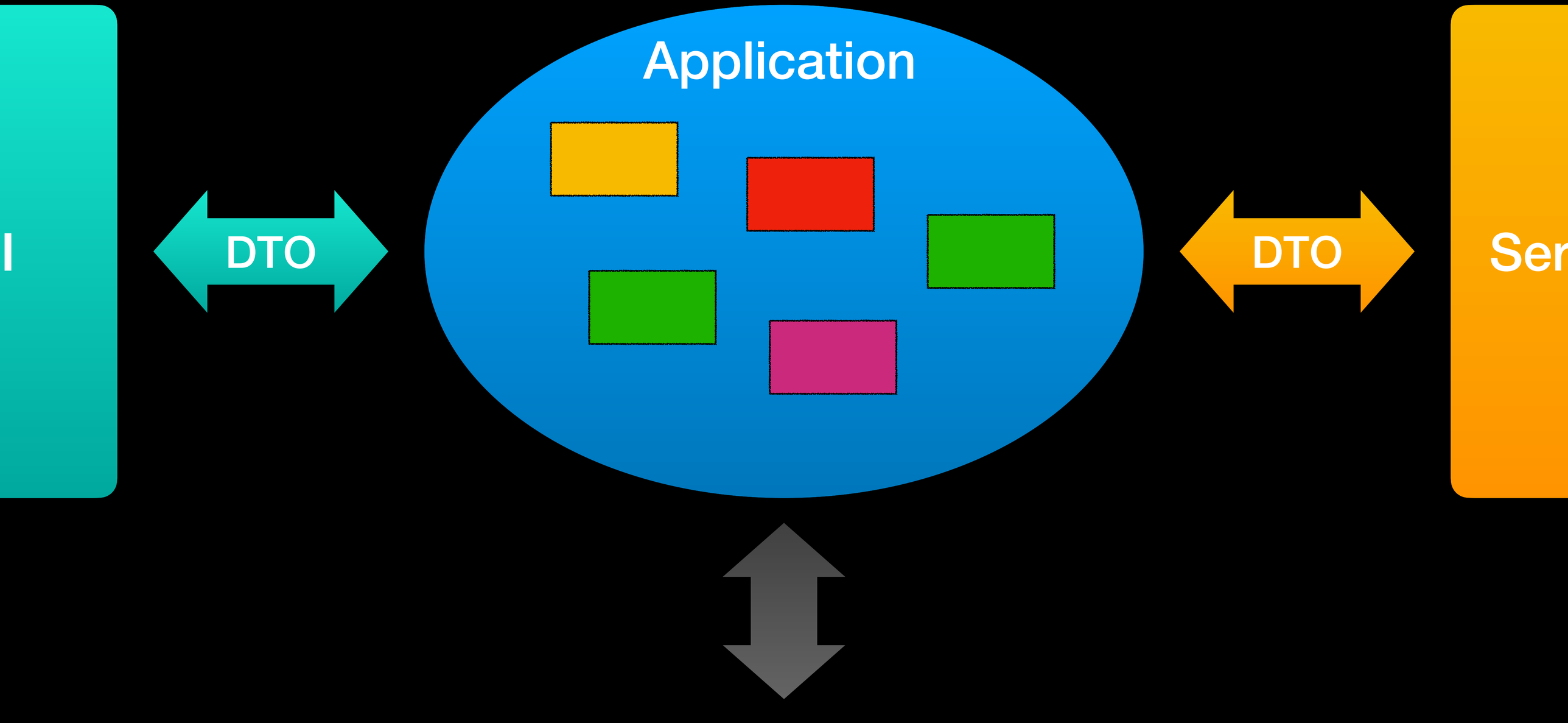
ORM

- inadequate abstraction
- relations leak through
- incorrect abstraction
- SQL leaks through
- high coupling
- difficult to test
- inefficient


ANTI PATTERN
Storage

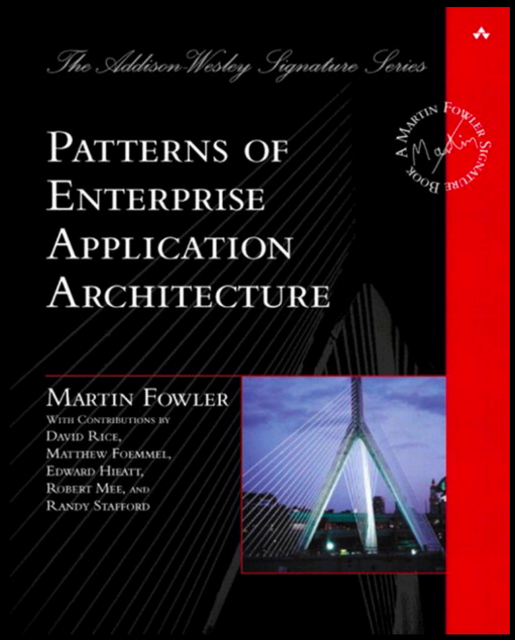

custom data access layer
Storage


Application


l
DTO
Application
DTO
Ser

Data Transfer Object

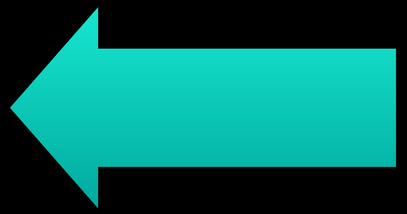
- reduce round trips
- encapsulation of serialization logic
- decouples from the domain model



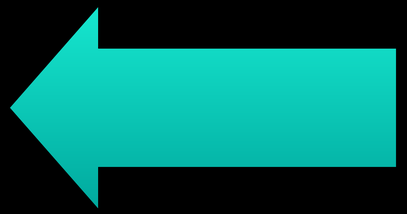
```java
public class User {
    private String name;
    private String password;
    private List<Role> roles;
    // standard constructors, getters, and setters
}

public class Role {
  private String name;
  // standard constructors, getters, and setters
}
```

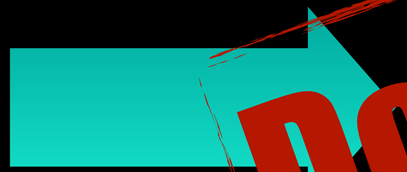
```java
public class UserDTO {
  private String name;
  private String password;
  private List<String> roleNames;
  // standard constructors, getters, and setters
}
```

```
class Mapper {
  public UserDTO toDto(User user) {
    String name = user.getName();
    String password = user.getPassword();
    List<String> roles = user
      .getRoles()
      .stream()
      .map(role -> role.getName())
      .collect(toList());
    return new UserDTO(name, password, roles);
  }

  public User toUser(UserDTO userDTO) {
    String name = userDTO.getName();
    String password = userDTO.getPassword();
    List<Role> roles = userDTO
      .getRoleNames()
      .stream()
      .map(roleName -> new Role(roleName))
      .collect(toList());
    return new User(name, password, roles);
  }
}
```



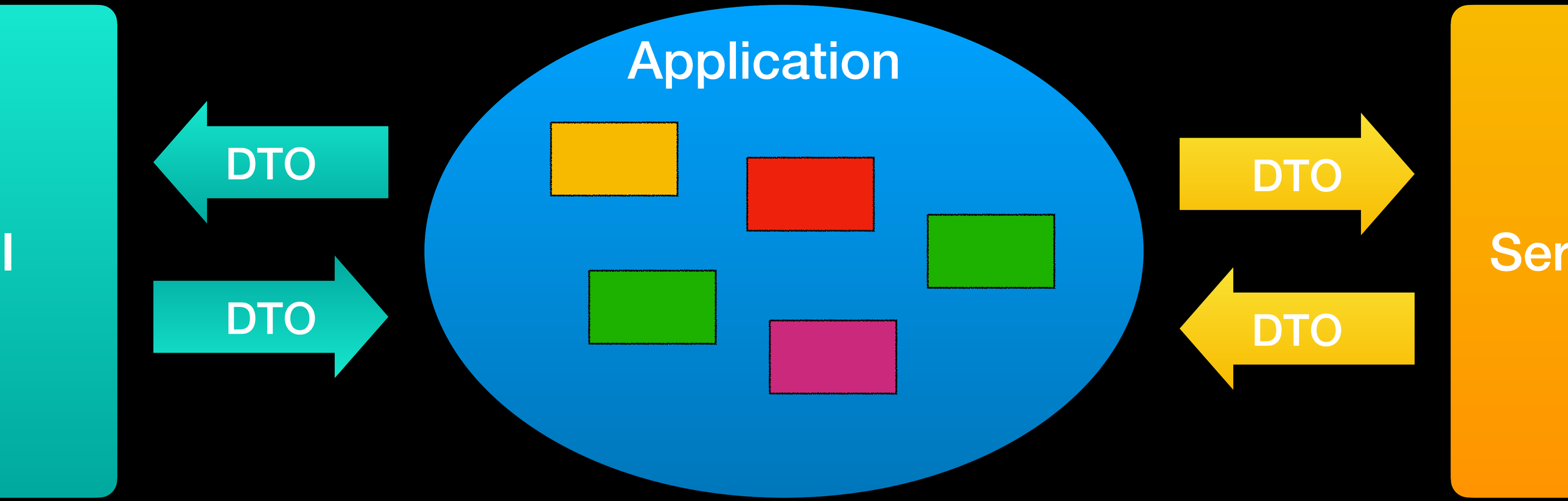
```java
class Mapper {
  public UserDTO toDto(User user) {
    String name = user.getName();
    String password = user.getPassword();
    List<String> roles = user
      .getRoles()
      .stream()
      .map(role -> role.getName())
      .collect(toList());
    return new UserDTO(name, password, roles);
  }

  public User toUser(UserDTO userDTO) {
    String name = userDTO.getName();
    String password = userDTO.getPassword();
    List<Role> roles = userDTO
      .getRoleNames()
      .stream()
      .map(roleName -> new Role(roleName))
      .collect(toList());
    return new User(name, password, roles);
  }
}
```





```
class Mapper {
  public UserDTO toDto(User user) {
    String name = user.getName();
    String password = user.getPassword();
    List<String> roles = user
      .getRoles()
      .stream()
      .map(role -> role.getName())
      .collect(toList());
    return new UserDTO(name, password, roles);
  }

  public User toUser(UserDTO userDTO) {
    String name = userDTO.getName();
    String password = userDTO.getPassword();
    List<Role> roles = userDTO
      .getRoleNames()
      .stream()
      .map(roleName -> new Role(roleName))
      .collect(toList());
    return new User(name, password, roles);
  }
}
```

Don't repeat yourself


Application
DTO
DTO
DTO
DTO

FPFTW

functional programming for the win

Lenses

- F#: Aether
  `https://xyncro.tech/aether`
- Scala: Monocle
  `https://www.optics.dev/Monocle`
- Haskell: Control.Lens
  `https://hackage.haskell.org/package/lens`
- Clojure: Active.Clojure.Lens
  `https://github.com/active-group/active-clojure#lenses`

```
(define-record-type User
  make-user
  user?
  [name user-name
   password user-password
   roles user-roles])

(define-record-type Role
  make-role
  role?
  [name role-name])

(def mike
  (make-user "Mike" "s3cr37" [(make-role "Manager")]))
```

```
(define-record-type User
  make-user
  user?
  [name user-name
   password user-password
   roles user-roles])

(define-record-type Role
  make-role
  role?
  [name role-name])

(def mike
  (make-user "Mike" "s3cr37" [(make-role "Manager")]))
```

Lenses

```
(def mike
  (make-user "Mike" "s3cr37" [(make-role "Manager")]))
```

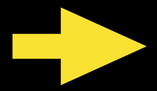
`user-password`

Lenses

```clojure
(def mike
  (make-user "Mike" "s3cr37" [(make-role "Manager")]))
```

user-password

```clojure
(lens/yank mike user-password)
```



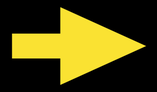
```clojure
"s3cr37"
```

Lenses

```clojure
(def mike
  (make-user "Mike" "s3cr37" [(make-role "Manager")]))
```

user-password

```clojure
(lens/shove mike user-password "passw0rd")
```



```clojure
(make-user "Mike" "passw0rd" [(make-role "Manager")])
```

Lenses

```clojure
(def mike
  (make-user "Mike" "s3cr37" [(make-role "Manager")]))
```

```clojure
(lens/>> user-roles (lens/at-index 0) role-name)
```

Lenses

```clojure
(def mike
  (make-user "Mike" "s3cr37" [(make-role "Manager")]))
```

```clojure
(lens/>> user-roles (lens/at-index 0) role-name)
```

Lenses

```clojure
(def mike
  (make-user "Mike" "s3cr37" [(make-role "Manager")]))
```

```clojure
(lens/>> user-roles (lens/at-index 0) role-name)
```

Lenses

```clojure
(def mike
  (make-user "Mike" "s3cr37" [(make-role "Manager")]))
```

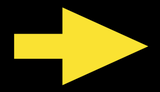
```clojure
(lens/>> user-roles (lens/at-index 0) role-name)
```

Lenses

```clojure
(def mike
  (make-user "Mike" "s3cr37" [(make-role "Manager")]))
```

```clojure
(def first-role-name
  (lens/>> user-roles (lens/at-index 0) role-name))
```

```clojure
(lens/yank mike first-role-name)
```



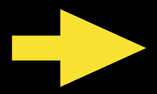
```clojure
"Manager"
```

Lenses

```clojure
(def mike
  (make-user "Mike" "s3cr37" [(make-role "Manager")]))
```

```clojure
(def first-role-name
  (lens/>> user-roles (lens/at-index 0) role-name))
```

```clojure
(lens/shove mike first-role-name "Admin")
```

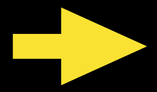


```clojure
(make-user "Mike" "s3cr37" [(make-role "Admin")])
```

Lenses

- yank

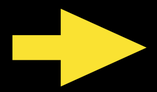
```
(first-role-name mike)
```



```
"Manager"
```

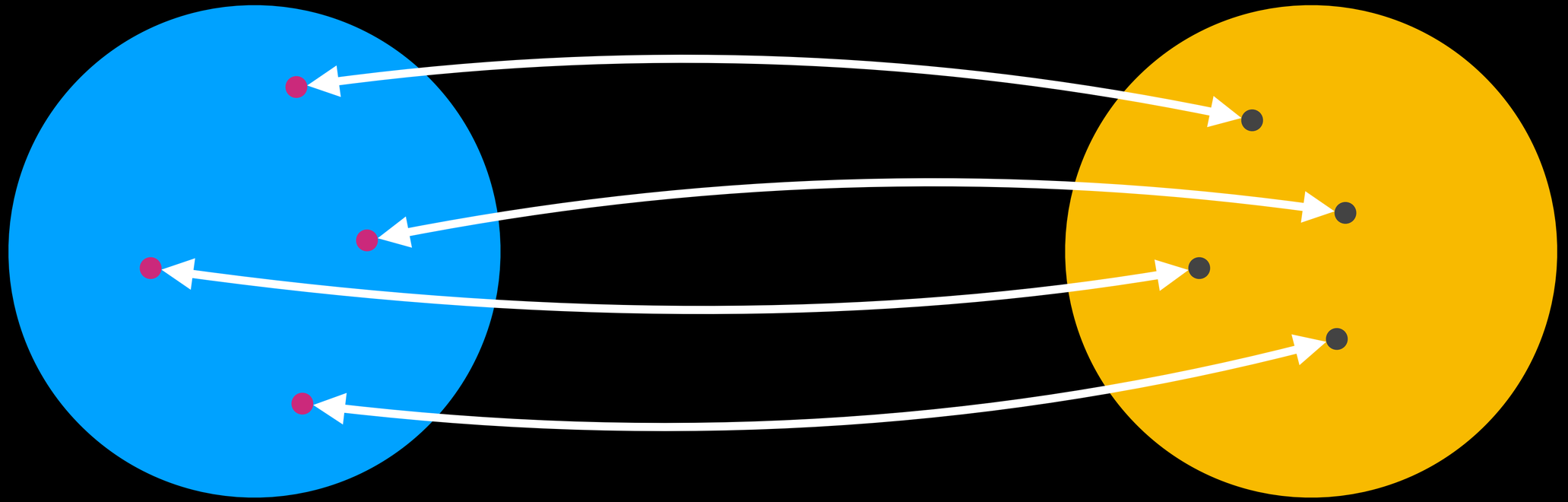
- shove

```
(first-role-name mike "Admin")
```



```
(make-user "Mike" "s3cr37" [(make-role "Admin")])
```

Projections



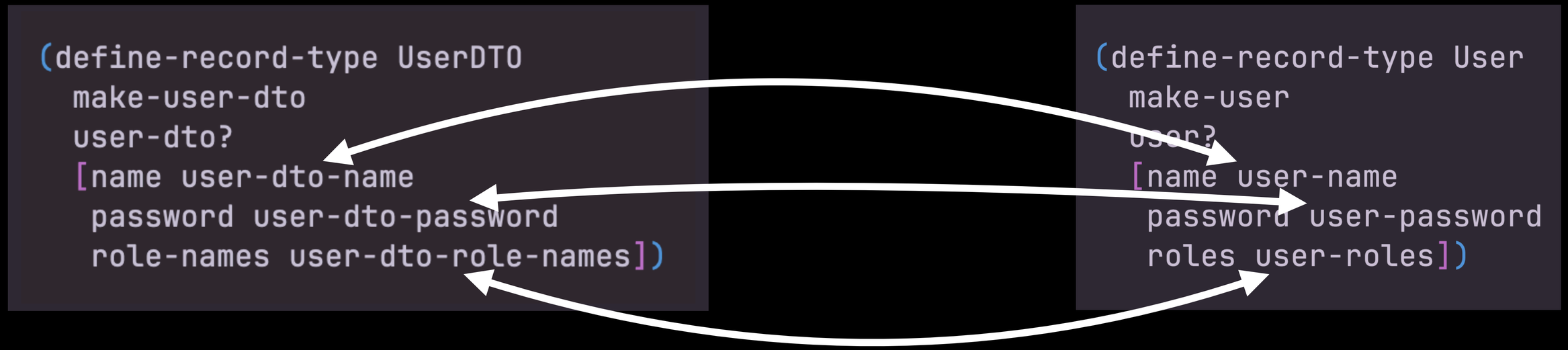
bijective mapping

Projections

```
(define-record-type UserDTO
  make-user-dto
  user-dto?
  [name user-dto-name
   password user-dto-password
   role-names user-dto-role-names])
```

```
(define-record-type User
  make-user
  user?
  [name user-name
   password user-password
   roles user-roles])
```

Projections



Projections

```
{user-dto-name       user-name,
 user-dto-password   user-password,
 user-dto-role-names (lens/>> user-roles
                             (lens/mapl role-name))}
```

```
(define-record-type UserDTO
  make-user-dto
  user-dto?
  [name user-dto-name
   password user-dto-password
   role-names user-dto-role-names])
```

```
(define-record-type User
  make-user
  user?
  [name user-name
   password user-password
   roles user-roles])
```

Projections

```
{user-dto-name        user-name,
 user-dto-password    user-password,
 user-dto-role-names  (lens/>> user-roles
                              (lens/mapl role-name))}
```

```
(define-record-type UserDTO
  make-user-dto
  user-dto?
  [name user-dto-name
   password user-dto-password
   role-names user-dto-role-names])
```

```
(define-record-type User
  make-user
  user?
  [name user-name
   password user-password
   roles user-roles])

(define-record-type Role
  make-role
  role?
  [name role-name])
```

Bidirectional Transformations

```
(def user->user-dto
  (lens/projection (make-user-dto nil nil nil)
                   {user-dto-name       user-name,
                    user-dto-password   user-password,
                    user-dto-role-names (lens/>> user-roles
                                                 (lens/mapl role-name))}))

(def user-dto->user
  (lens/invert user->user-dto (make-user nil nil nil)))
```

Bidirectional Transformations

```
(def user->user-dto
  (lens/projection (make-user-dto nil nil nil)
                   {user-dto-name       user-name,
                    user-dto-password   user-password,
                    user-dto-role-names (lens/>> user-roles
                                                 (lens/mapl role-name))}))

(def user-dto->user
  (lens/invert user->user-dto (make-user nil nil nil)))
```

Projection lenses

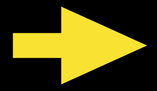
```
(define-record-type UserDTO
  {:projection-lens into-user-dto-projection-lens}
  make-user-dto
  user-dto?
  [name user-dto-name
   password user-dto-password
   roles user-dto-roles])
```

Projection lenses

```clojure
(define-record-type UserDTO
  {:projection-lens into-user-dto-projection-lens}
  make-user-dto
  user-dto?
  [name user-dto-name
   password user-dto-password
   roles user-dto-roles])

(def edn->user-dto
  (into-user-dto-projection-lens :name
                                 :password
                                 :role-names))

(def user-dto->edn
  (lens/invert edn->user-dto))
```

Projection lenses

```
(user-dto->edn (make-user-dto "Mike" "s3cr37" ["Admin"]))
```



```
{:name     "Mike",
 :password "s3cr37",
 :roles    ["Manager"]}
```

Projection lenses

```
(make-user-dto "Mike" "s3cr37" ["Admin"]))
```

```
{:name     "Mike",
 :password "s3cr37",
 :roles    ["Manager"]}
```

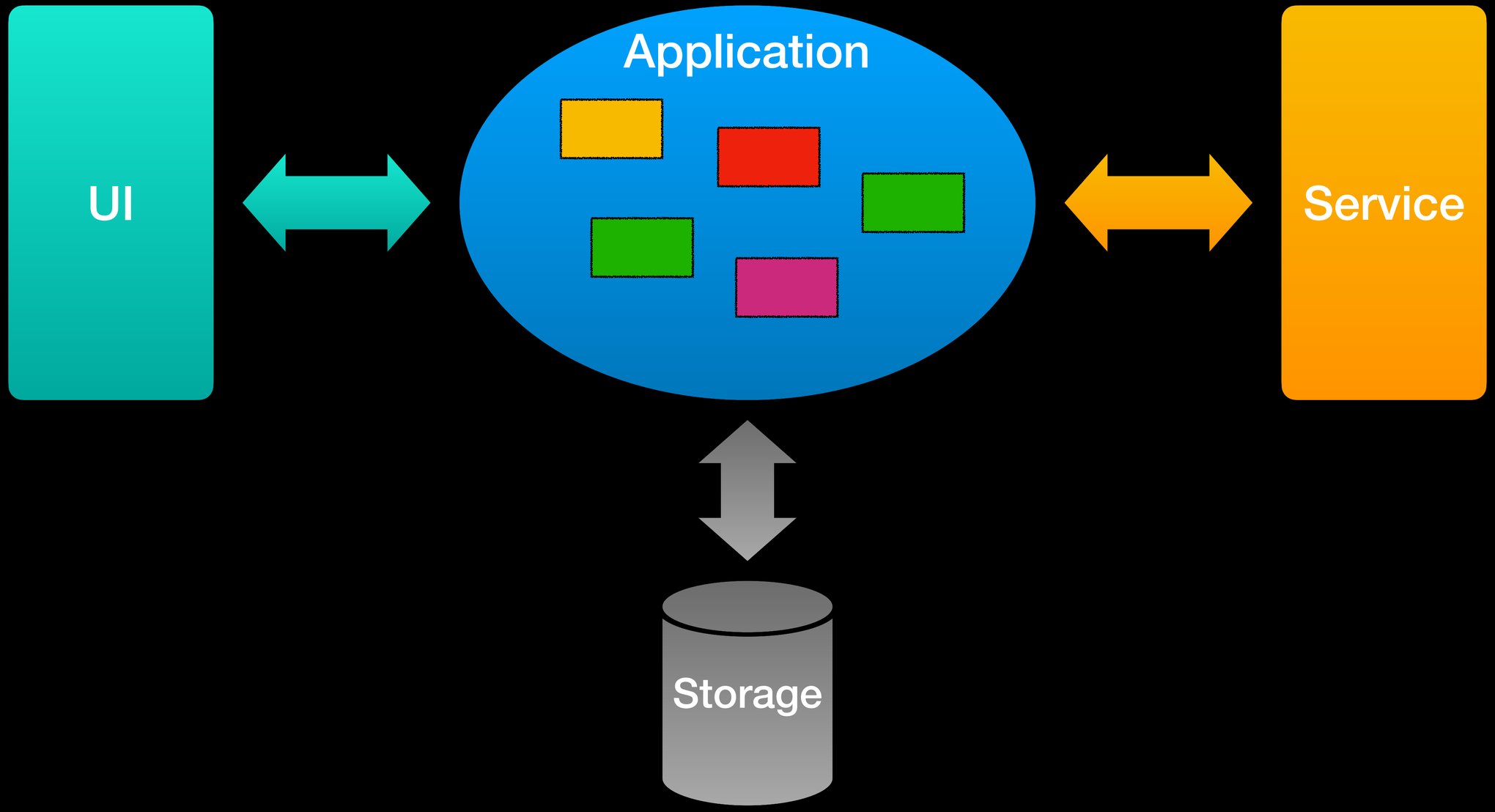
Composability

- Concatenation

```clojure
(def user->edn
  (lens/>> user->user-dto user-dto->edn))
```

- Mixed, recursive data definitions

```clojure
(def tree->edn
  (lens/union-vector [tree-node? tree-node->edn]
                     [tree-leaf? tree-leaf->edn]))
```


UI
Application
Service
Storage

@active group

https://active-group.de

Marcus Crestani
(he/him)

- Software Development
- Consulting
- Trainings

- Functional Programming
- Functional Software Architecture
- https://funktionale-programmierung.de
- https://bobkonf.de

Links

- `https://github.com/active-group/active-clojure`
  Library with various basic utilities for programming with Clojure, i.e. the lenses
- `https://funktionale-programmierung.de/2023/02/28/projection-lenses.html`
  Blog posting on projection lenses (German)