BANCA BT TRANSILVANIA®

Hexagonal Architecture

MARIAN GRADEA

… most programmers spend the first 5 years of their career mastering complexity, and the rest of their lives learning simplicity.”

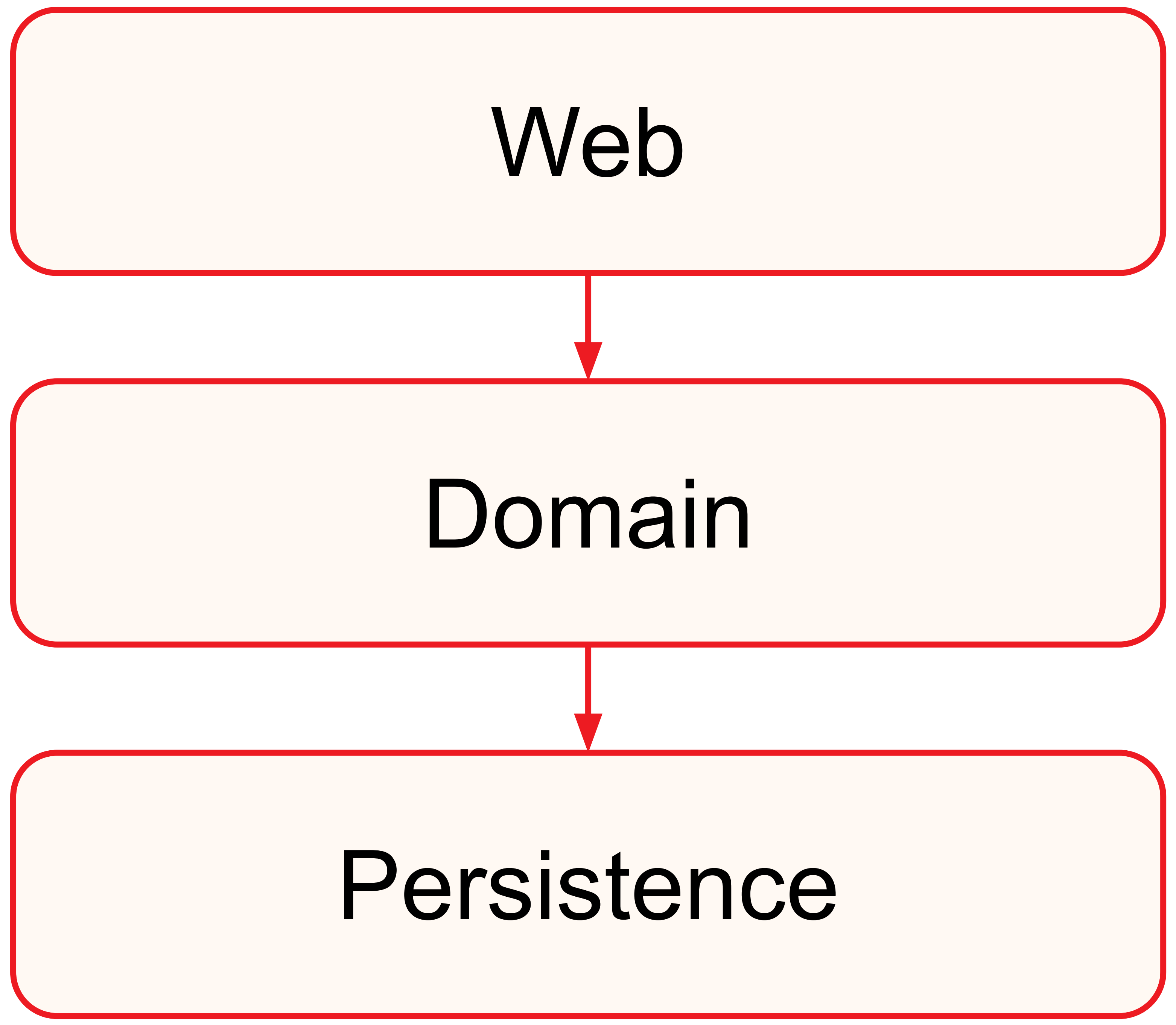
BUZZ ANDERSEN (DEC 30, 2009)

Agenda

1. Layered Architecture

What it promises

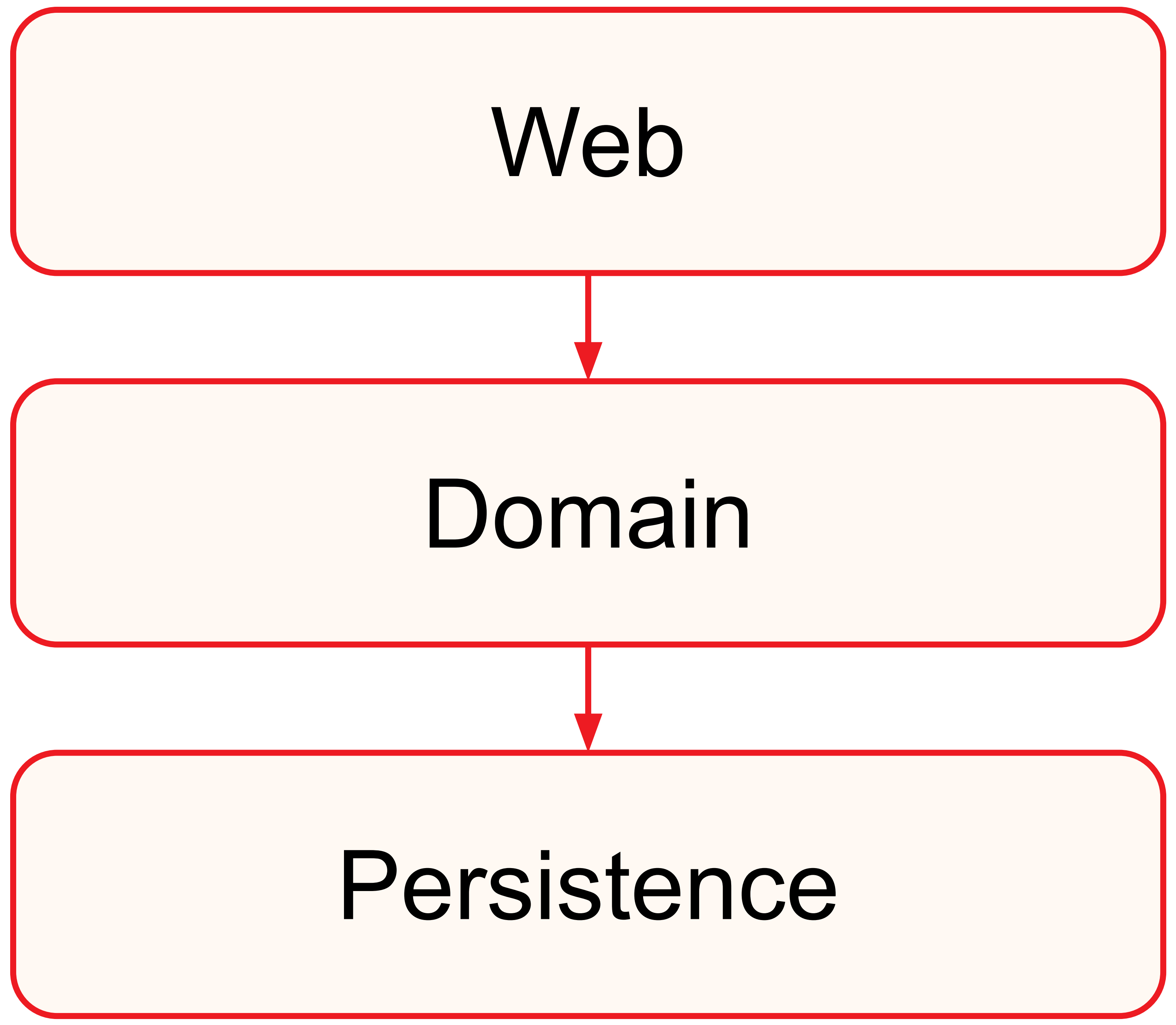
- Ease of development
- Testability



1. Layered Architecture

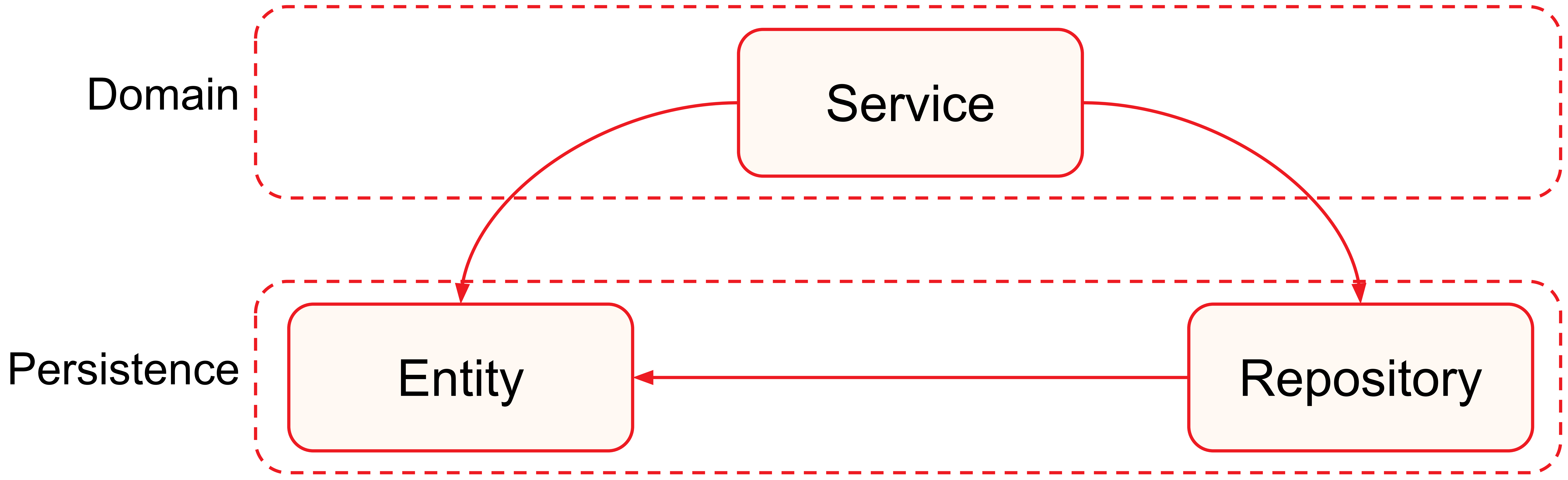
What's wrong with it

- It promotes database-driven design
- It's prone to shortcuts
- It grows hard to test
- It hides the use cases
- It makes parallel work difficult



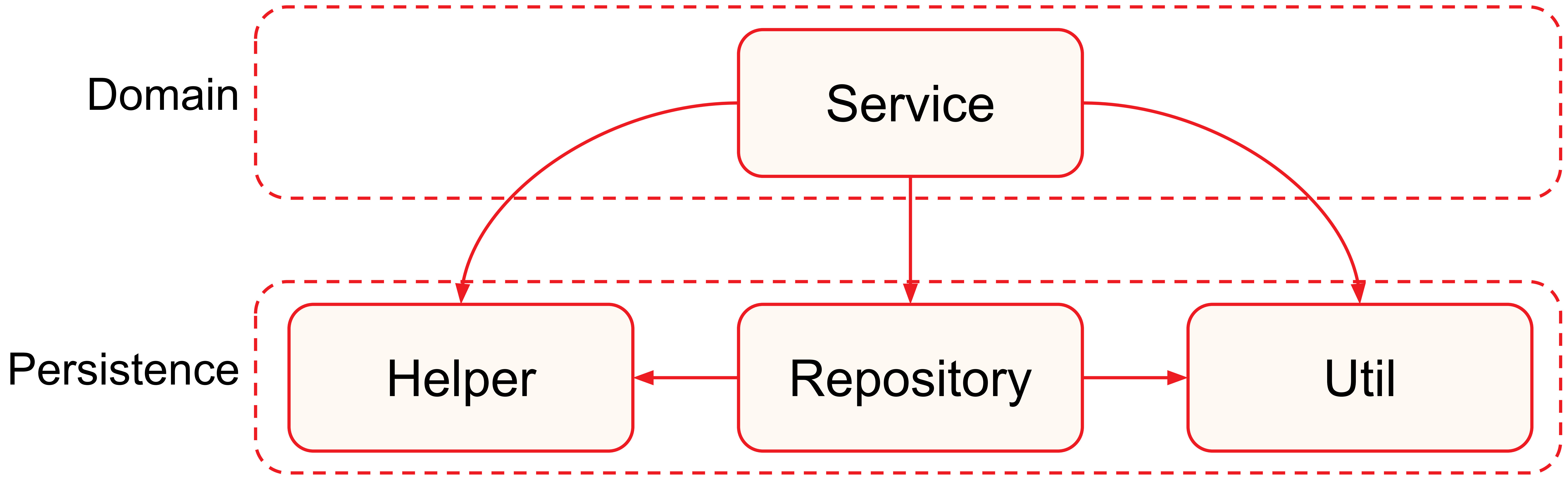
1. Layered Architecture

Promotes database-driven design



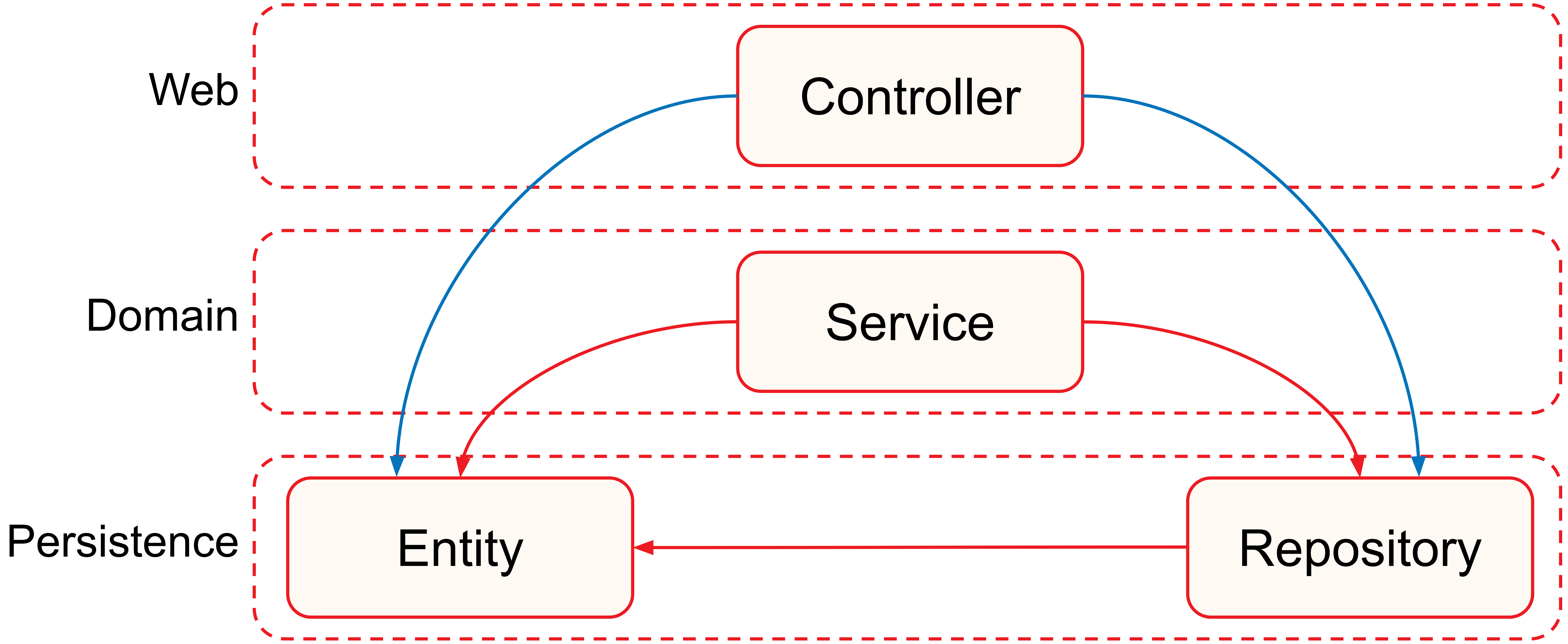
1. Layered Architecture

It’s prone to shortcuts



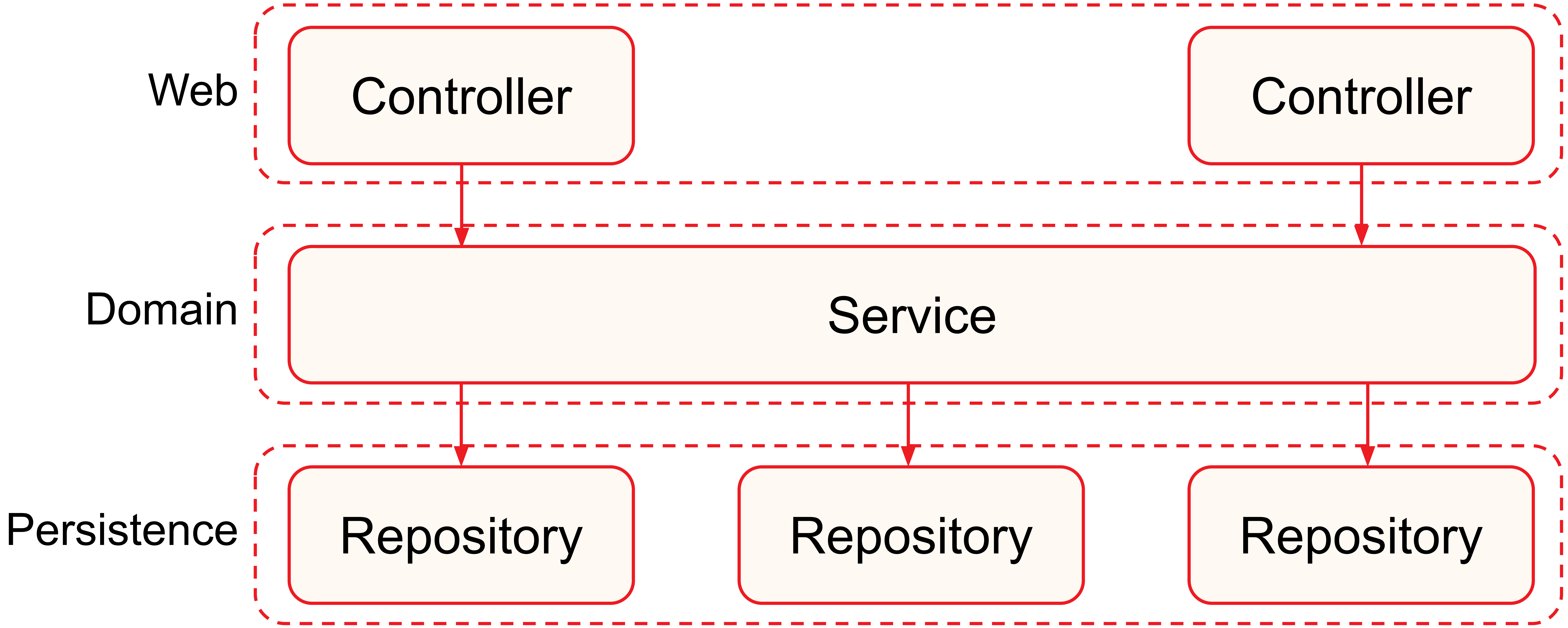
1. Layered Architecture

Grows hard to test



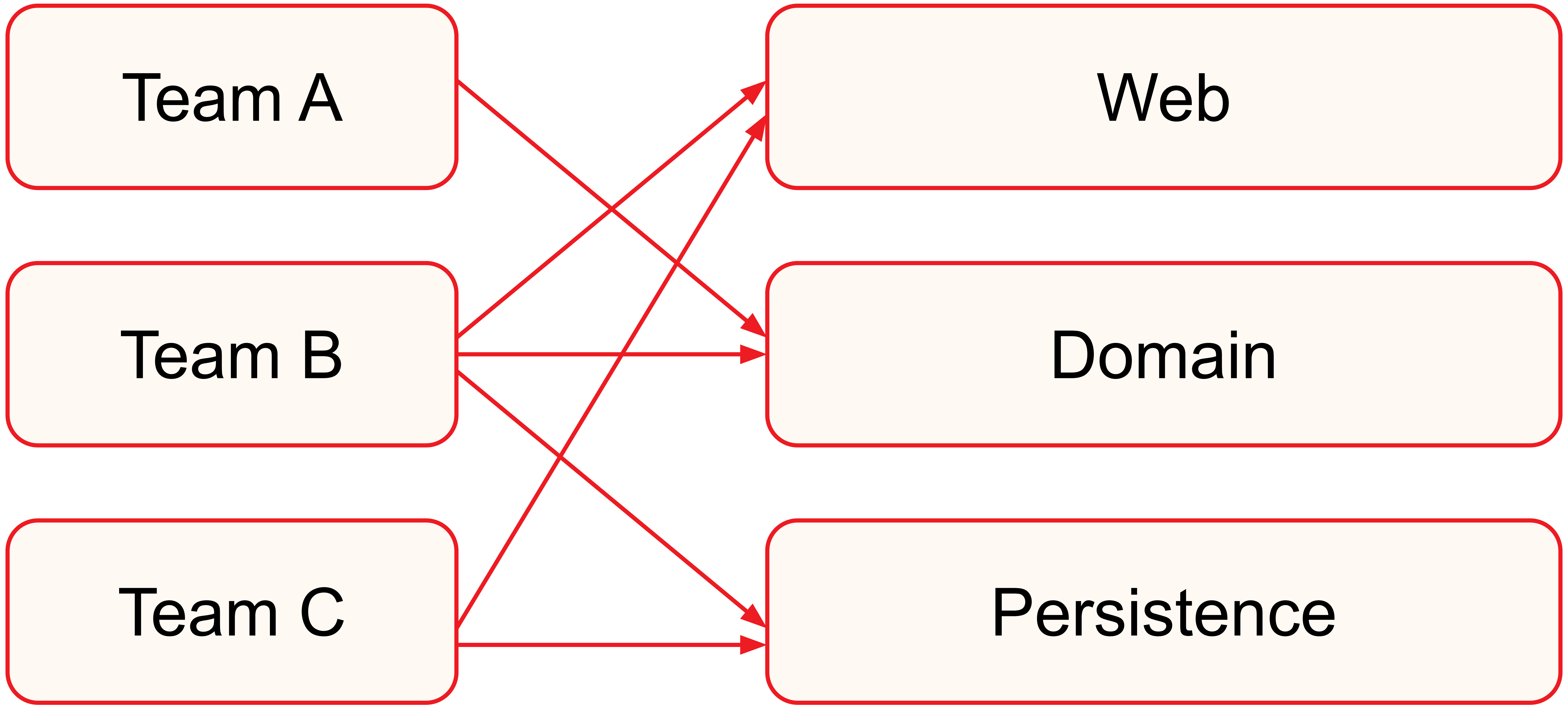
1. Layered Architecture

Hides the use cases



1. Layered Architecture

Makes parallel work difficult



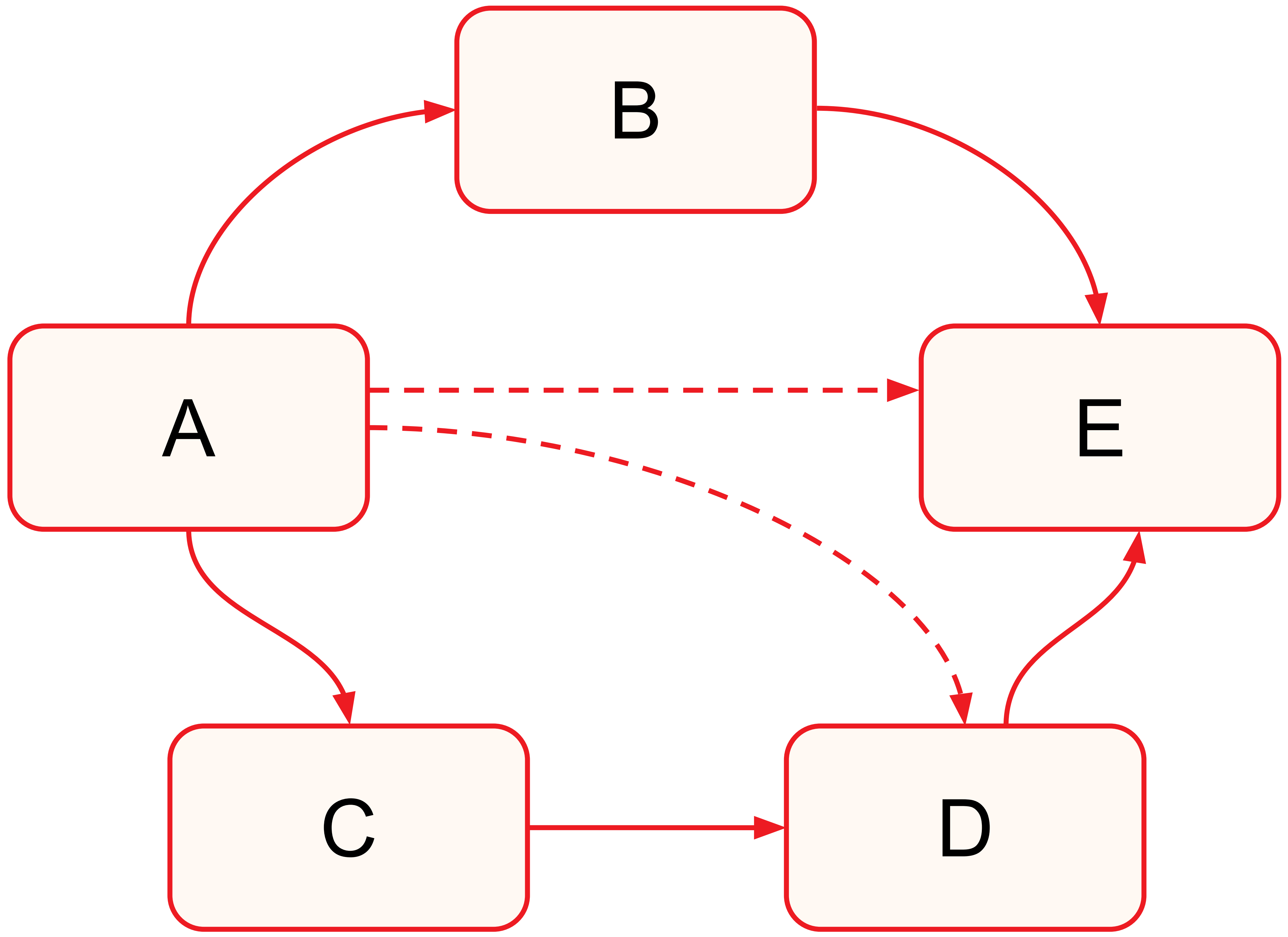
2. Inverting Dependencies

The single responsibility principle

- A component should do only one thing, and do it right.

-or-

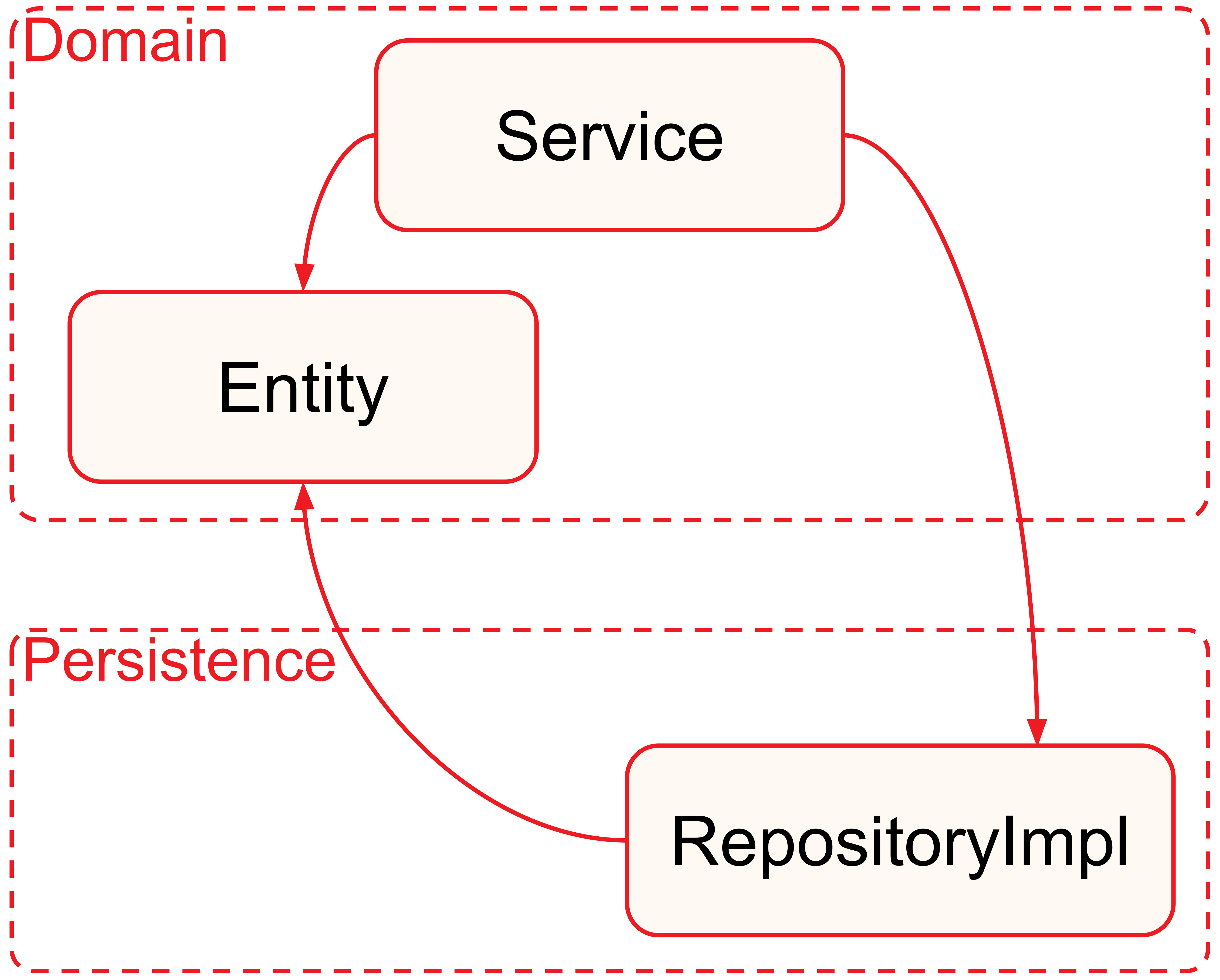
- A component should have only one reason to change.


$$\begin{tikzcd}
 & B \arrow[dr] & \\
A \arrow[ur] \arrow[rr, dashed] \arrow[d] \arrow[drr, dashed, bend left] & & E \\
C \arrow[rr] & & D \arrow[u]
\end{tikzcd}$$

2. Inverting Dependencies

The dependency inversion principle

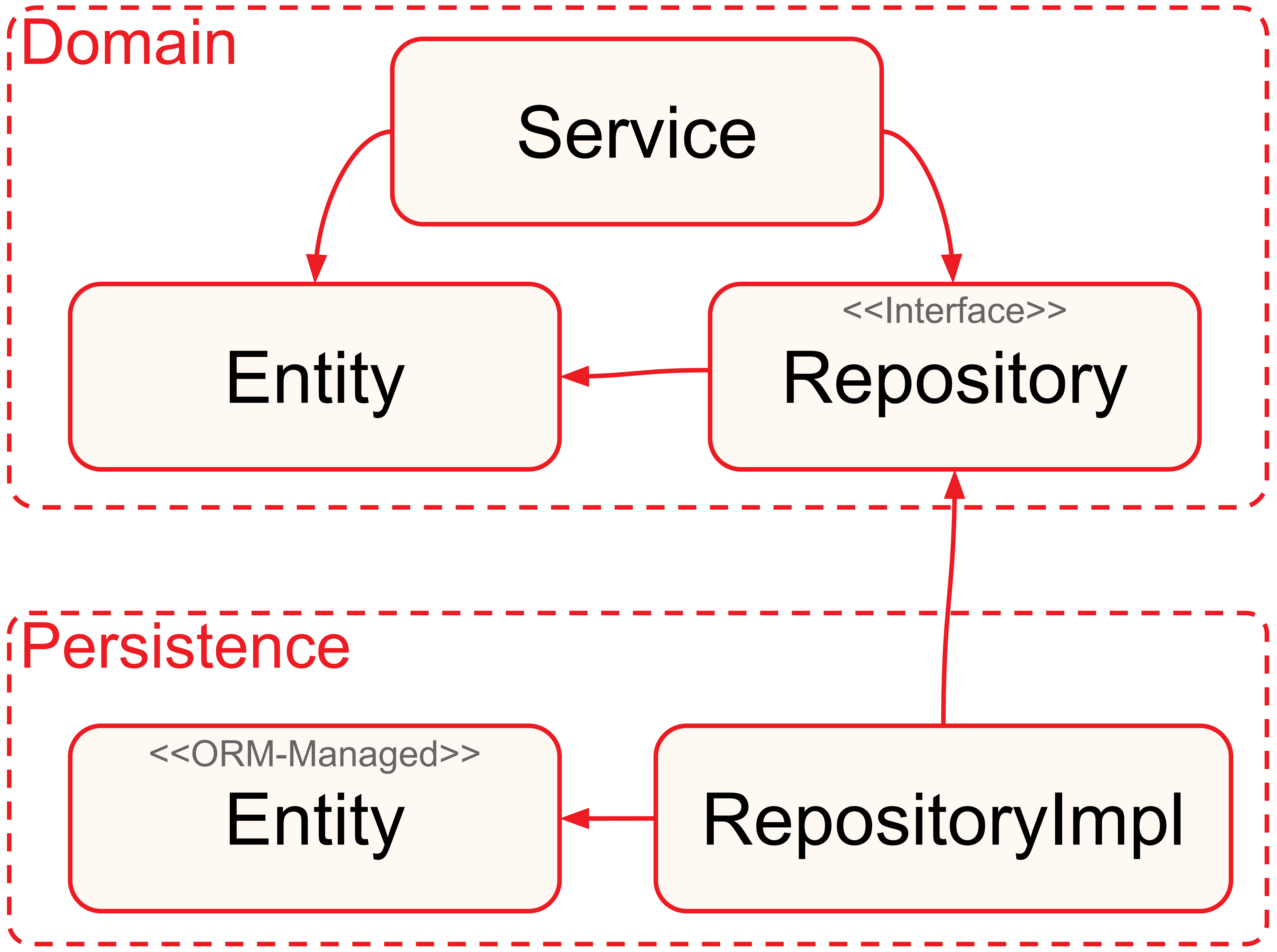
- The direction of any dependency in the codebase can be inverted.



2. Inverting Dependencies

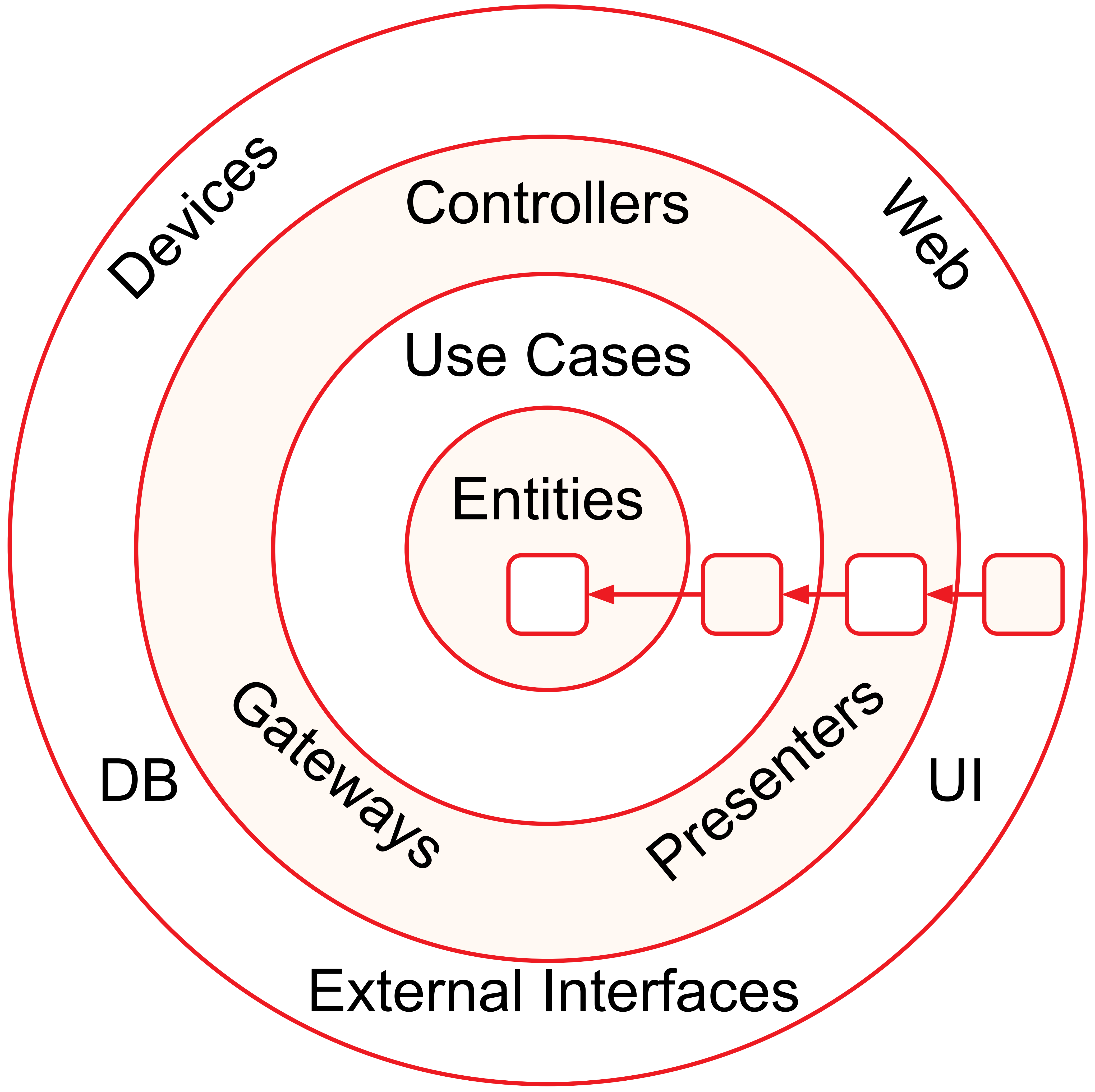
The dependency inversion principle

- The direction of any dependency in the codebase can be inverted.



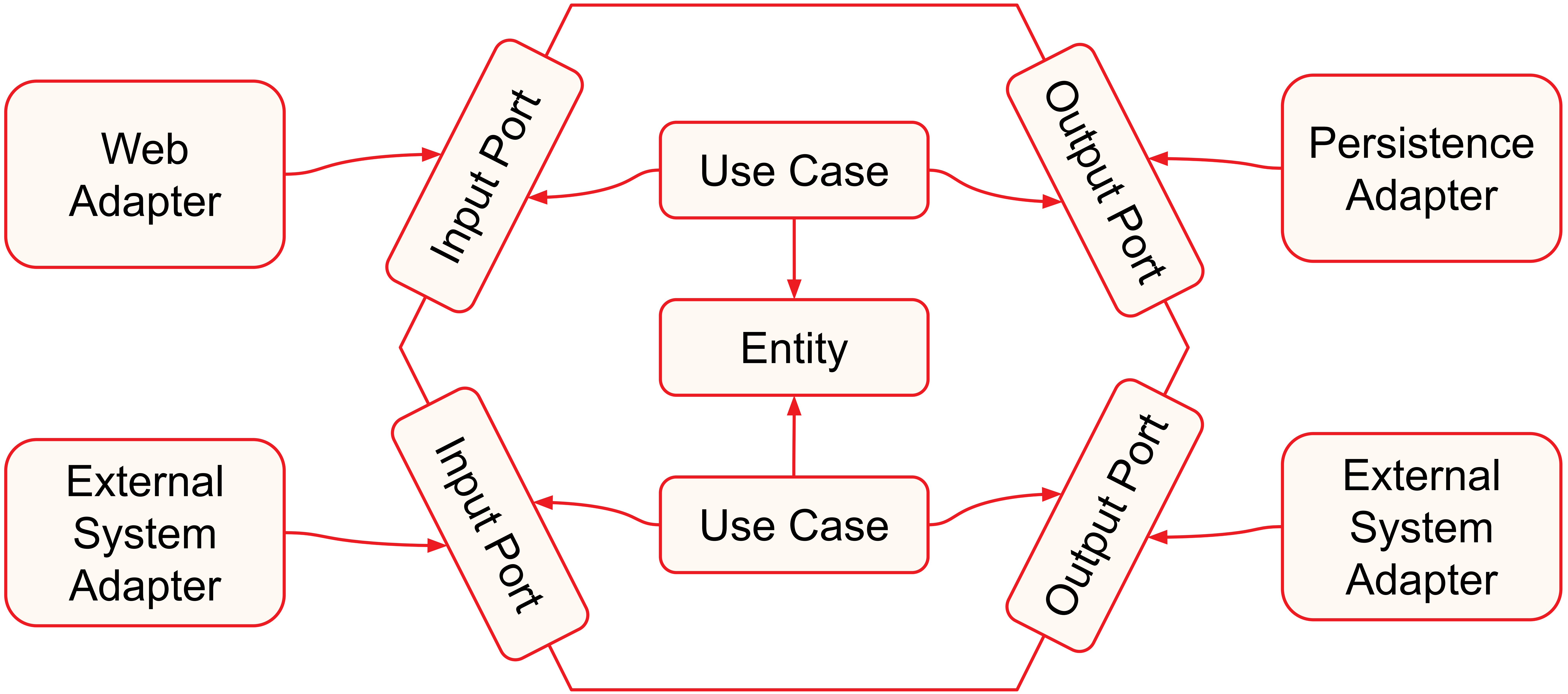
2. Inverting Dependencies

Clean architecture



2. Inverting Dependencies

Clean architecture



3. Organizing Code

Organizing by layer

```
ro.btrl.hexar
├── domain
│   ├── Account
│   ├── AccountRepository
│   ├── AccountService
│   ├── AccountServiceImpl
├── persistence
│   └── AccountRepositoryImpl
└── web
    └── AccountController
```

Organizing by feature

```
ro.btrl.hexar
└── account
    ├── Account
    ├── AccountController
    ├── AccountRepository
    ├── AccountRepositoryImpl
    └── AccountService
```

3. Organizing Code

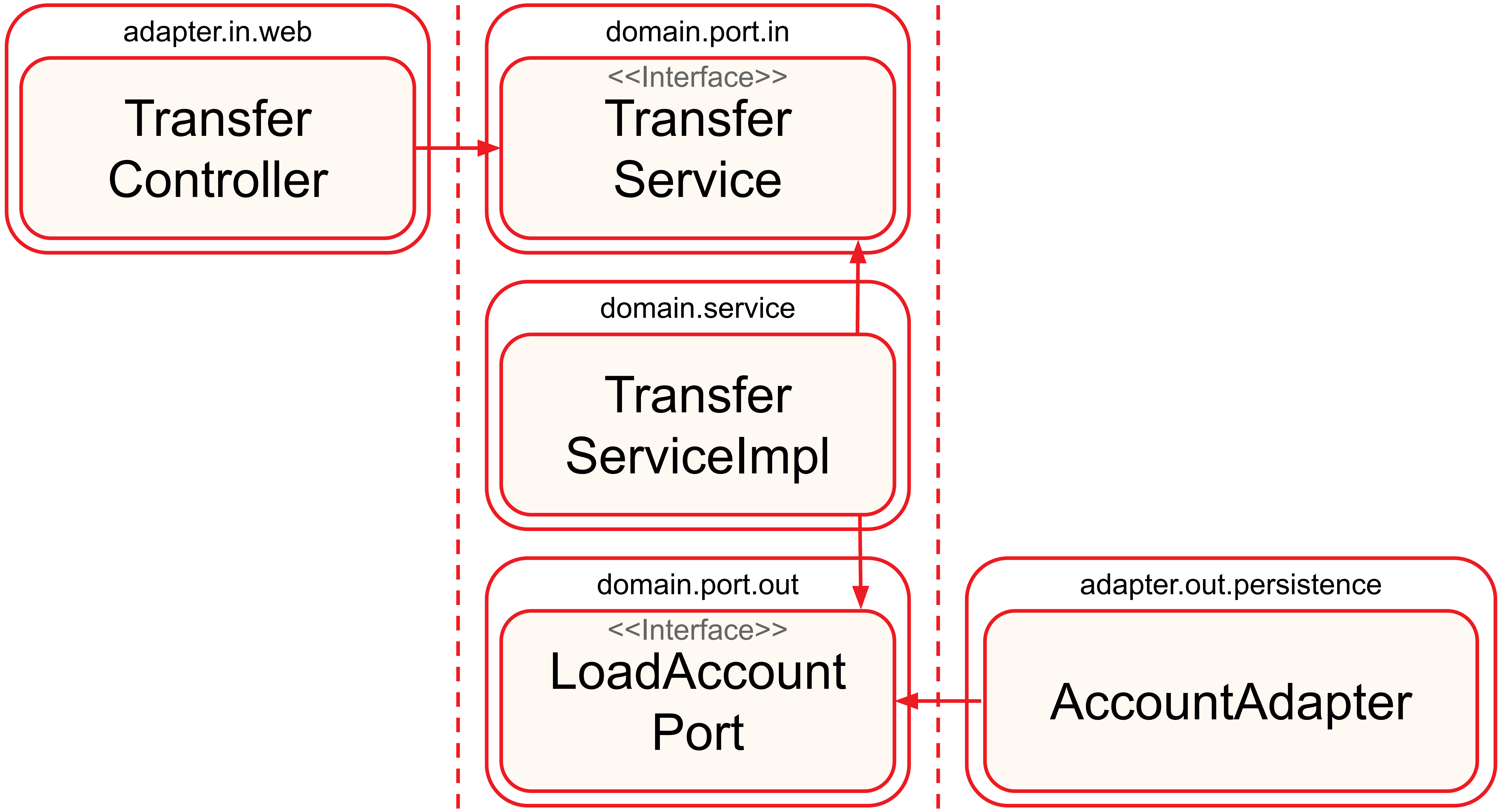
Expressive package structure

```
ro.btrl.hexar
├── adapter
│   ├── in.web
│   │   ├── controller
│   │   │   └── AccountController
│   │   ├── dto
│   │   │   └── AccountDto
│   │   └── mapper
│   │       └── AccountDtoMapper
│   └── out.persistence
│       ├── entity
│       │   └── AccountEntity
│       ├── mapper
│       │   └── AccountEntityMapper
│       ├── repository
│       │   └── AccountRepository
...
```

```
...
│           └── AccountAdapter
└── domain
    ├── model
    │   └── Account
    ├── port
    │   ├── in
    │   │   └── AccountService
    │   └── out
    │       ├── LoadAccountPort
    │       └── LoadAllAccountsPort
    └── service
        └── AccountServiceImpl
```

3. Organizing Code

The role of dependency injection



4. Implementing a Use Case

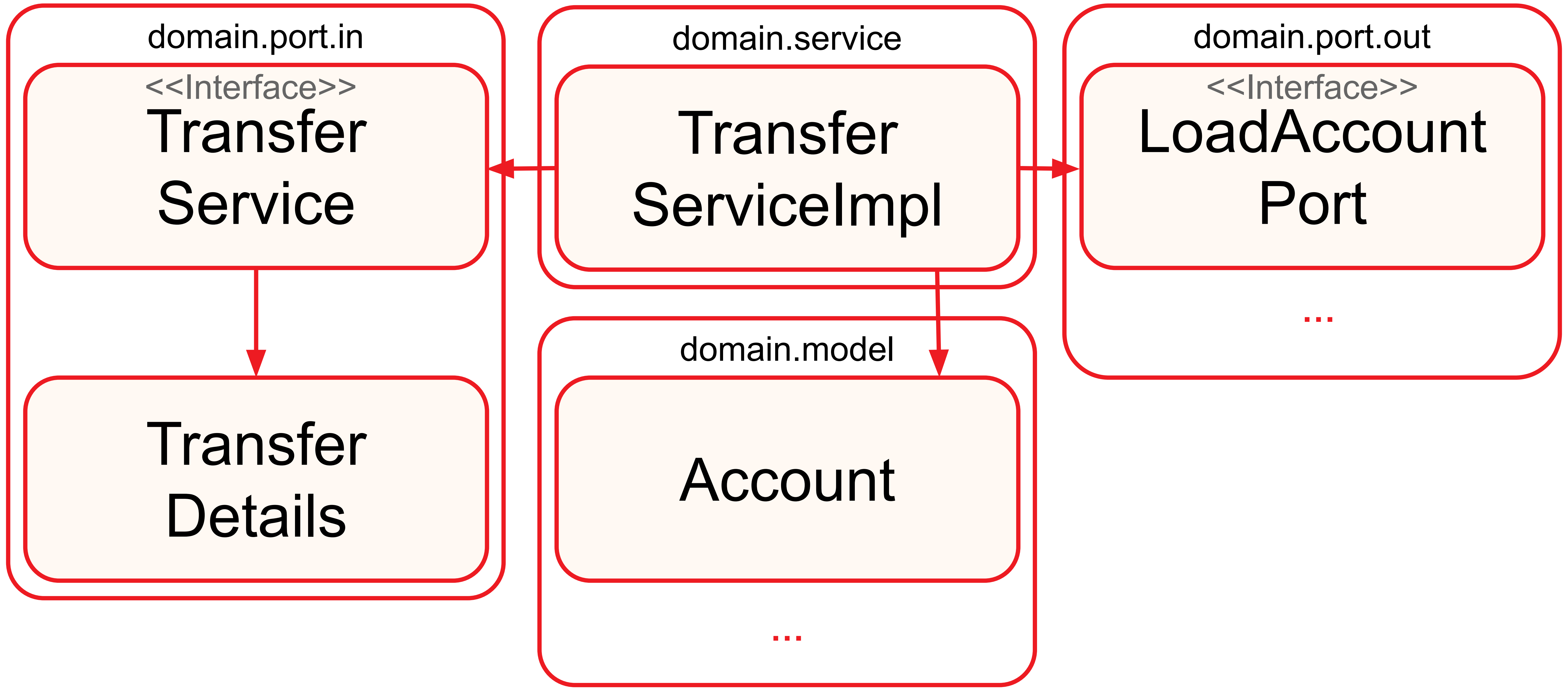
The domain model

```java
package ro.btrl.hexar.domain.model;
...
public class Account {

  private Long id;
  private Long balance;
  private List<Transaction> transactions;

  public Long calculateBalance() {
    ...
    final var depositBalance = transactions.stream()
                     .filter(t -> t.getTargetAccountId().equals(id))
                     .map(Transaction::getAmount)
                     .reduce(0L, Long::sum);
    ...
    return balance + depositBalance - withdrawalBalance;
  }
}
```

4. Implementing a Use Case

A use case responsibilities

- Take input
- Validate business rules
- Manipulate model state
- Return output



4. Implementing a Use Case

Validate input

```java
package ro.btrl.hexar.domain.port.in;
...
@Getter
@RequiredArgsConstructor
public class TransferDetails {

  @NotNull
  private final Long sourceAccountId;

  @NotNull
  private final Long targetAccountId;

  @NotNull
  @PositiveOrZero
  private final Long amount;
}
```

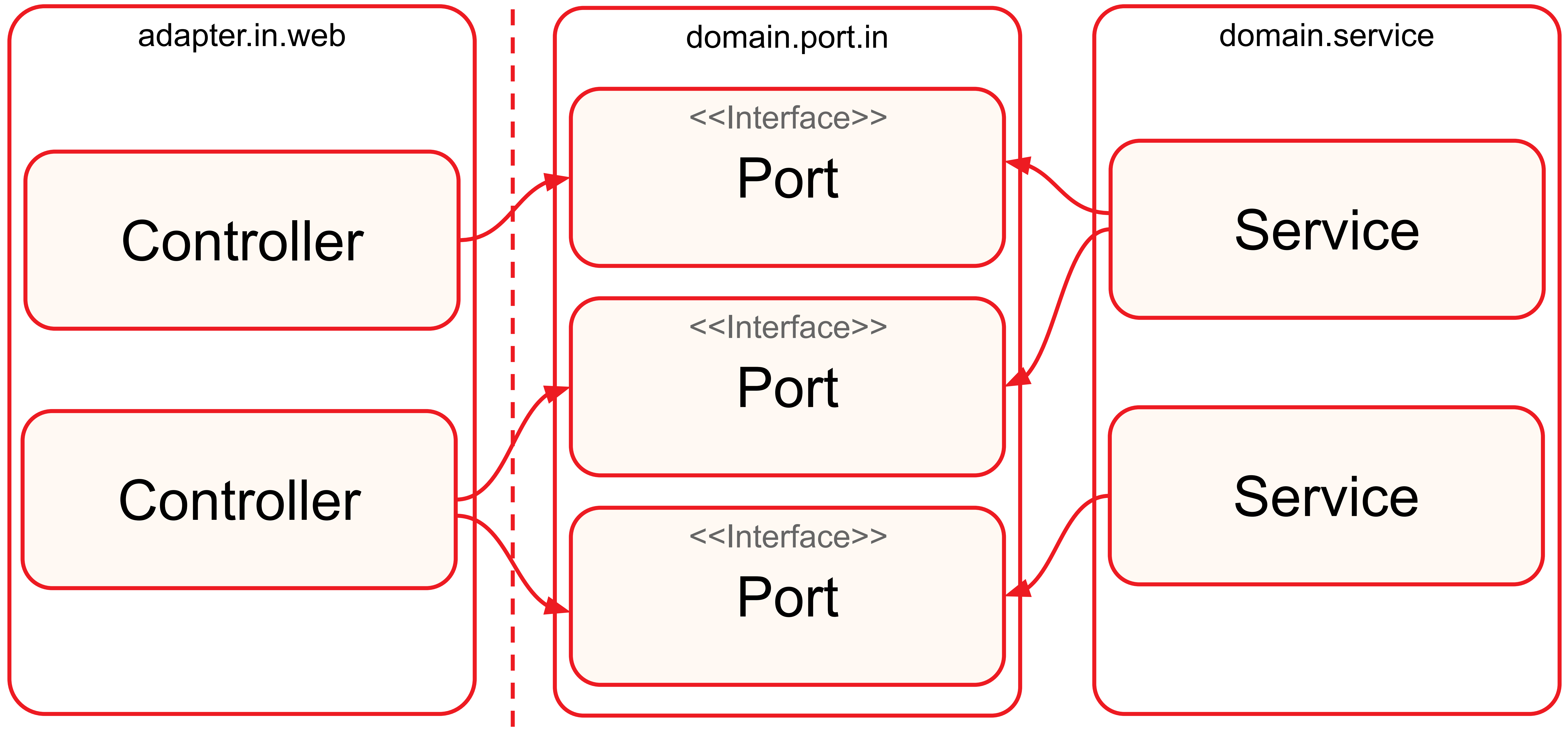
4. Implementing a Use Case

A use case implementation

```java
package ro.btrl.hexar.domain.service;
...
@Component
@RequiredArgsConstructor
@Transactional
public class TransferServiceImpl implements TransferService {
  private final LoadAccountPort loadAccountPort;
  private final CreateTransactionPort createTransactionPort;
  @Override
  public void transfer(TransferDetails transferDetails) {
    // Validate source and target accounts' ids
    // Get source and target accounts
    // Check source account's balance
    // Create transaction model
    ...
    createTransactionPort.createTransaction(transaction);
  }
}
```

5. Implementing the Adapters

Web adapter



5. Implementing the Adapters

Web adapter responsibilities

- Map HTTP request to Java objects
- Perform authorization checks
- Validate input
- Map input to the input model of the use case
- Call the use case
- Map output of the use case back to HTTP
- Return HTTP response

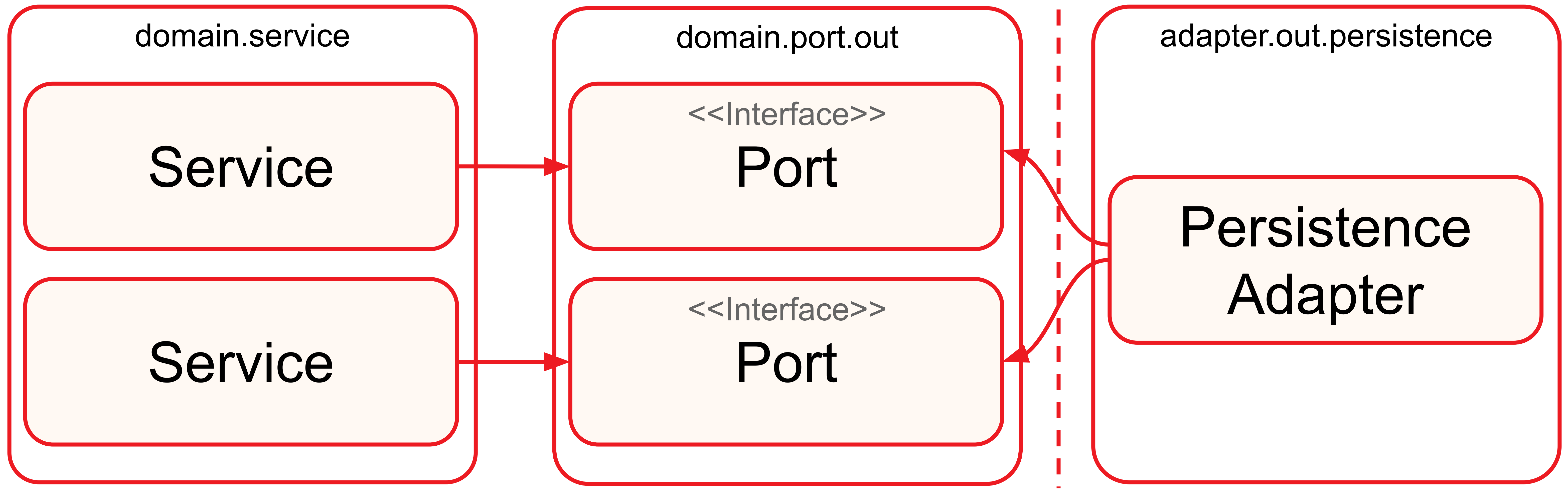
5. Implementing the Adapters

Web adapter

```java
package ro.btrl.hexar.adapter.in.web;
...
@RestController
@RequestMapping("transfers")
@RequiredArgsConstructor
public class TransferController {

  private final TransferService transferService;

  @PostMapping(path = "execute/{sourceAccountId}/{targetAccountId}/{amount}")
  void transfer(@PathVariable("sourceAccountId") Long sourceAccountId,
                @PathVariable("targetAccountId") Long targetAccountId,
                @PathVariable("amount") Long amount) {
      final var transferDetails =
              new TransferDetails(sourceAccountId, targetAccountId, amount);
      transferService.transfer(transferDetails);
  }
}
```

5. Implementing the Adapters

Persistence adapter



5. Implementing the Adapters

Persistence adapter responsibilities

- Take input
- Map input into database format
- Send input to the database
- Map database output into application format
- Return output

5. Implementing the Adapters

Persistence adapter implementation (1)

```
package ro.btrl.hexar.adapter.out.persistence.repository;
...
interface AccountRepository extends JpaRepository<AccountEntity, Long> {
}
```

```
package ro.btrl.hexar.adapter.out.persistence.repository;
...
interface TransactionRepository extends JpaRepository<TransactionEntity, Long> {

  @Query("""
          SELECT t FROM TransactionEntity t
          WHERE t.sourceAccountId = :accountId
          OR t.targetAccountId = :accountId
          AND t.processed = false
         """)
  List<TransactionEntity> getAllTransactionsForAccountId(Long accountId);
}
```

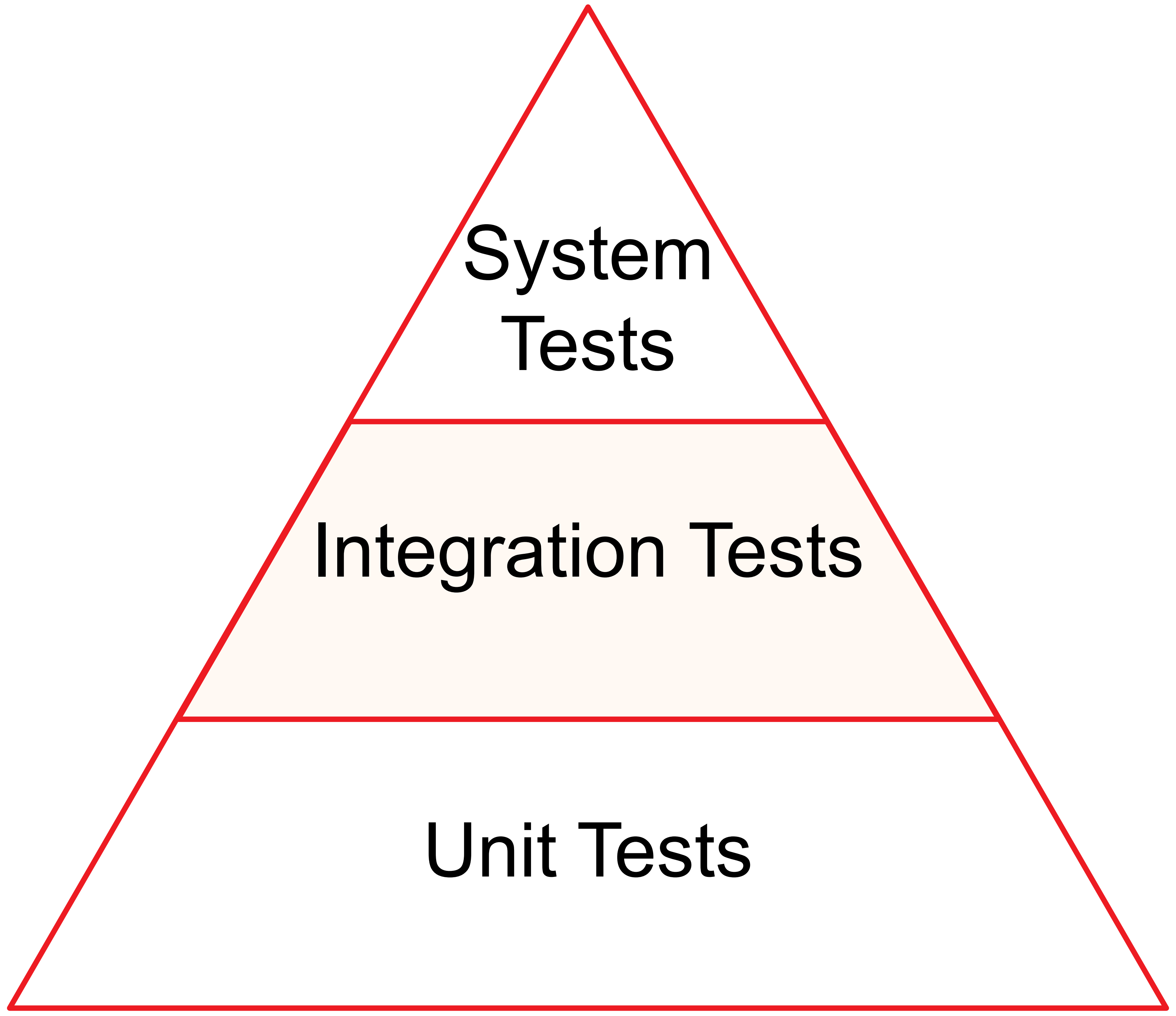
5. Implementing the Adapters

Persistence adapter implementation (2)

```java
package ro.btrl.hexar.adapter.out.persistence;
...
@Component
@RequiredArgsConstructor
class AccountAdapter implements LoadAccountPort {

  private final AccountEntityMapper accountEntityMapper;
  private final TransactionEntityMapper transactionEntityMapper;
  private final AccountRepository accountRepository;
  private final TransactionRepository transactionRepository;
  @Override
  public Account loadAccount(Long accountId) {
    // Load account and transactions
    ...
    return account;
  }
}
```

6. Testing Architecture Elements

The test pyramid



6. Testing Architecture Elements

Unit testing a domain entity

```java
package ro.btrl.hexar.domain.model;
...
class AccountTest {
    @Test
    void calculateBalanceSuccess() {
        Account account = new Account();
        Transaction transactionOut = new Transaction(1L, 2L, 10L, LocalDateTime.now());
        Transaction transactionIn = new Transaction(2L, 1L, 10L, LocalDateTime.now());
        account.setId(1L);
        account.setBalance(20L);
        account.setTransactions(List.of(transactionOut, transactionIn));

        Long balance = account.calculateBalance();

        assertThat(balance).isEqualTo(20L);
    }
}
```

6. Testing Architecture Elements

Unit testing a use case

```java
package ro.btrl.hexar.domain.service;
...
@ExtendWith(MockitoExtension.class)
class TransferServiceImplTest {
    @Mock private LoadAccountPort loadAccountPort;
    @Mock private CreateTransactionPort createTransactionPort;
    @InjectMocks private TransferServiceImpl transferService;
    @Test
    void transferSuccess() {
        when(loadAccountPort.loadAccount(1L)).thenReturn(createAccount(1L));
        when(loadAccountPort.loadAccount(2L)).thenReturn(createAccount(2L));
        doNothing().when(createTransactionPort).createTransaction(any(Transaction.class));
        TransferDetails transferDetails = new TransferDetails(1L, 2L, 10L);
        transferService.transfer(transferDetails);
        verifyNoMoreInteractions(createTransactionPort);
    }
...
}
```

6. Testing Architecture Elements

Integration tests for persistence adapter

```java
package ro.btrl.hexar.adapter.out.persistence;
...
@DataJpaTest
@Import({AccountAdapterPort.class, AccountMapperImpl.class, TransactionMapperImpl.class})
class AccountAdapterTest {

    @Autowired
    private AccountAdapterPort accountAdapterPort;

    @Test
    @Sql("data.sql")
    void loadAccount() {
        Account account = accountAdapterPort.loadAccount(1001L);

        assertThat(account.getTransactions()).hasSize(2);
        assertThat(account.calculateBalance()).isEqualTo(95L);
    }
}
```

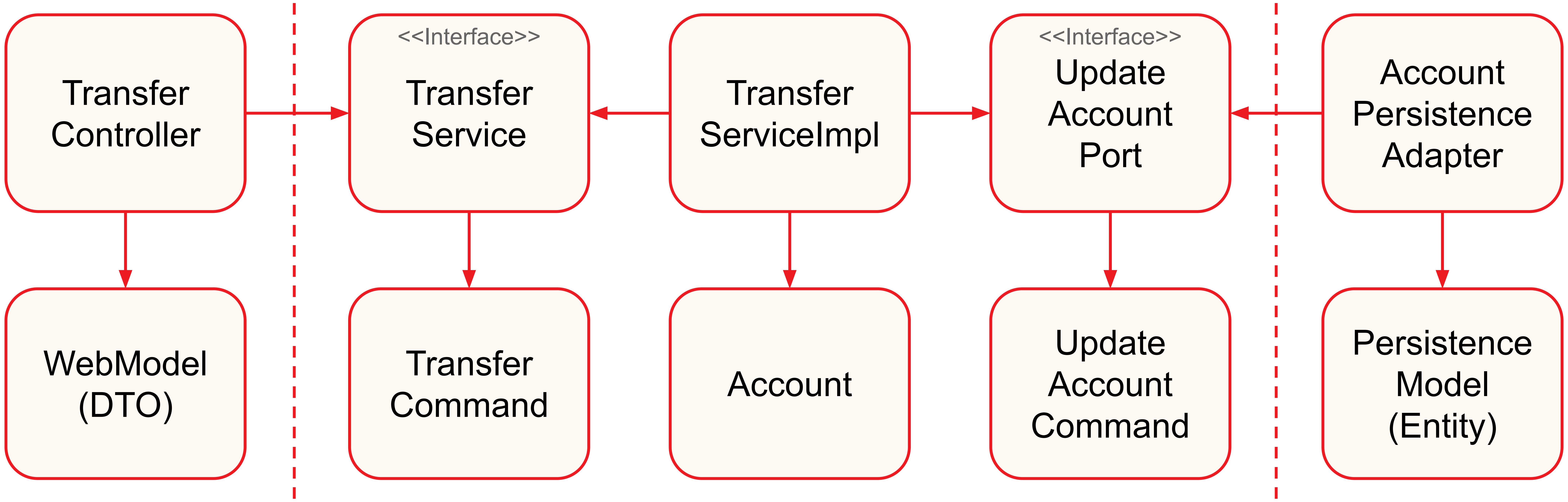
6. Testing Architecture Elements

System tests for main paths

```java
package ro.btrl.hexar.system;
...
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
class TransferSystemTest {
    @Autowired private TestRestTemplate restTemplate;
    @Test
    @Sql("data.sql")
    void transferSuccess() {
        HttpHeaders headers = new HttpHeaders();
        headers.add("Content-Type", "application/json");
        HttpEntity<Void> request = new HttpEntity<>(null, headers);
        var response = restTemplate.exchange(
                "/transfers/execute/{sourceAccountId}/{targetAccountId}/{amount}",
                HttpMethod.POST, request, Object.class, 101, 102, 10);
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
    }
}
```

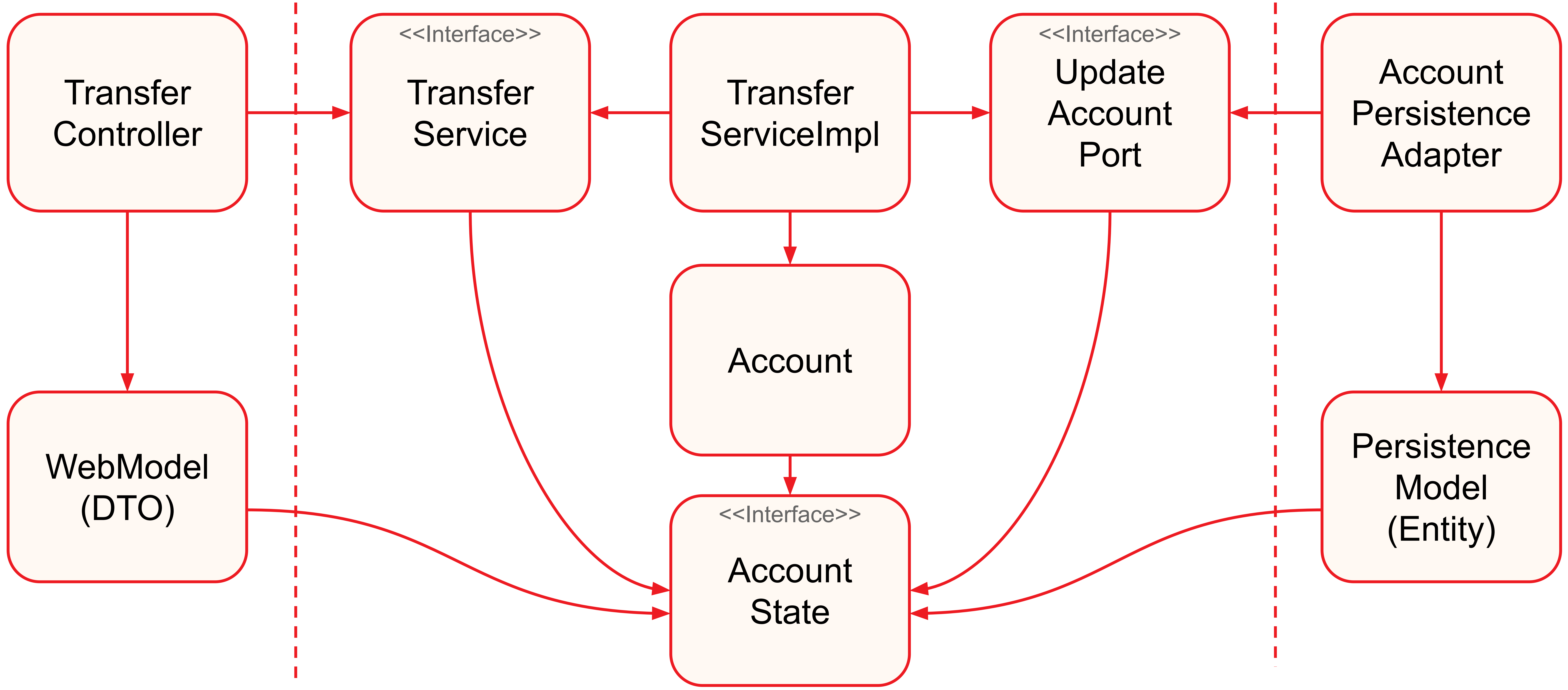
7. Mapping Between Boundaries

Full mapping strategy



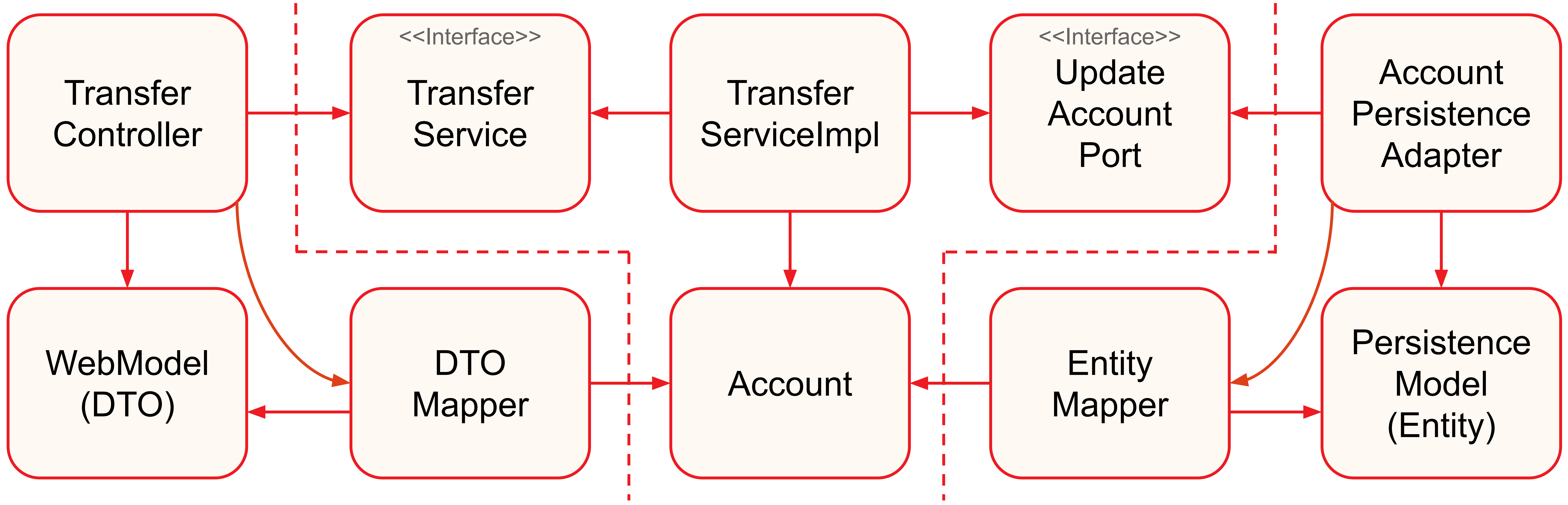
7. Mapping Between Boundaries

One way mapping strategy (1)



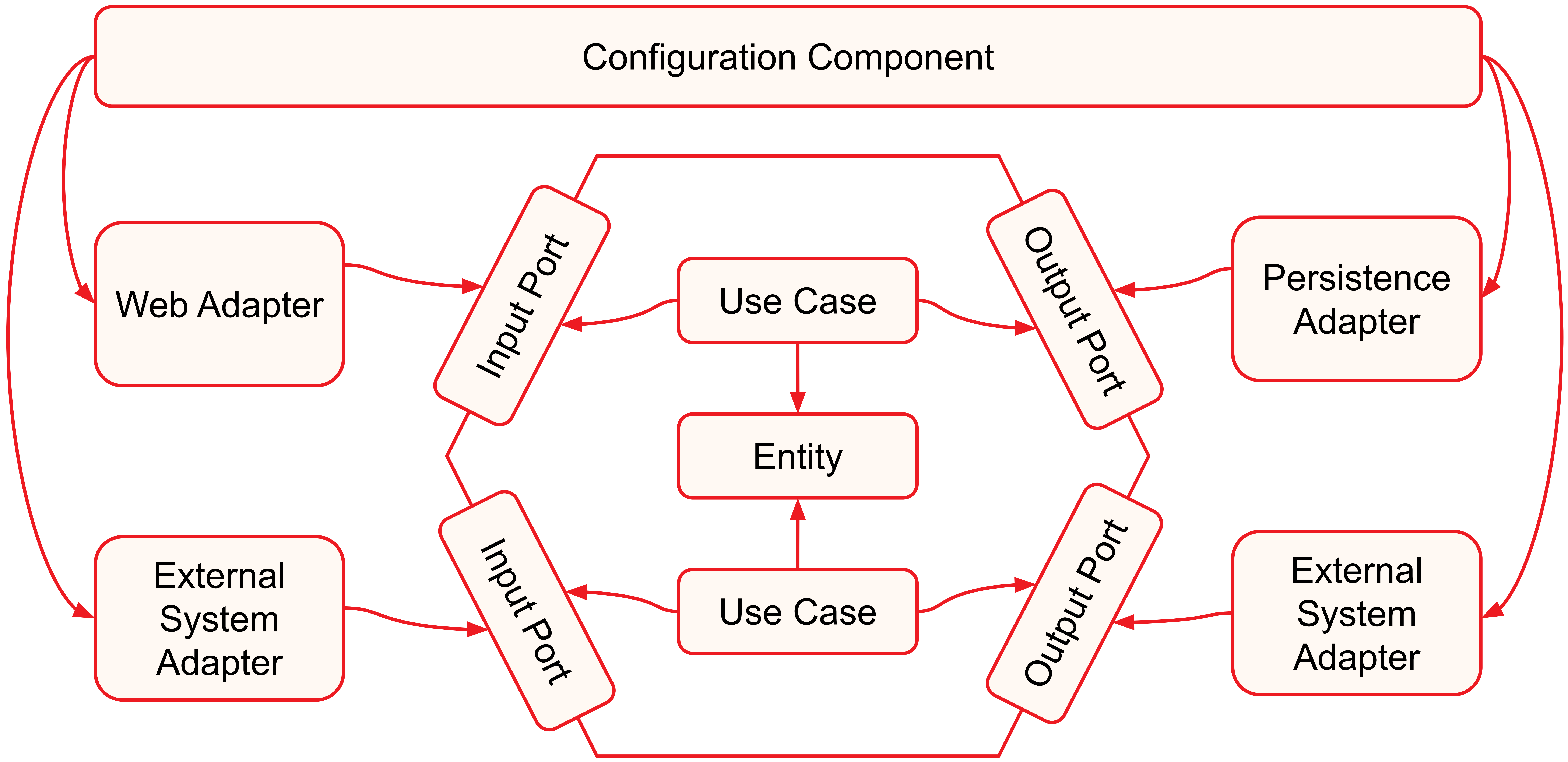
7. Mapping Between Boundaries

One way mapping strategy (2)



8. Assembling the Application

The configuration component



8. Assembling the Application

Assembling via plain code

```java
package ro.btrl.hexar;

public class HexarApplication {

  public static void main(String[] args) {
    var accountRepository = new AccountRepositoryImpl();
    var transactionRepository = new TransactionRepositoryImpl();
    var loadAccountPort = new AccountAdapter(accountRepository, transactionRepository);
    var createTransactionPort = new TransactionAdapter(transactionRepository);
    var transferService = new TransferServiceImpl(loadAccountPort, createTransactionPort);
    var transferController = new TransferController(transferService);
    startProcessingWebRequests(transferController);
  }
}
```

8. Assembling the Application

Assembling via Spring’s classpath scanning

```
...
@Component
@RequiredArgsConstructor
class AccountAdapter implements LoadAccountPort, LoadAllAccountsPort {

  private final AccountMapper accountMapper;
  private final TransactionMapper transactionMapper;
  private final AccountRepository accountRepository;
  private final TransactionRepository transactionRepository;

  @Override
  public Account loadAccount(Long accountId) {
    ...
    return account;
  }
  ...
}
```

8. Assembling the Application

Assembling via Spring's Java config

```
...
@Configuration
@EnableJpaRepositories
class PersistenceAdapterConfiguration {
  @Bean
  AccountAdapter accountAdapter(
      AccountEntityMapper accountEntityMapper,
      TransactionEntityMapper transactionEntityMapper,
      AccountRepository accountRepository,
      TransactionRepository transactionRepository){
    return new AccountAdapter(accountEntityMapper, transactionEntityMapper,
                              accountRepository, transactionRepository);
  }
  @Bean
  AccountDtoMapper accountDtoMapper(){
    return new AccountDtoMapper();
  }
}
```

9. Conclusions

It’s all relative

- There is no right way to implement the Hexagonal Architecture
- Avoid taking shortcuts

10. Resources

Nobody has time to read

- Get Your Hands Dirty on Clean Architecture

  A hands-on guide to creating clean web applications with code examples in Java

  - *Tom Hombergs*

- Clean Architecture

  A Craftsman's Guide to Software Structure and Design

  - *Robert C. Martin, Kevlin Henney*

- Clean Code

  A Handbook of Agile Software Craftsmanship

  - *Dean Wampler, Robert C. Martin*

Q&A

BANCA BT TRANSILVANIA®