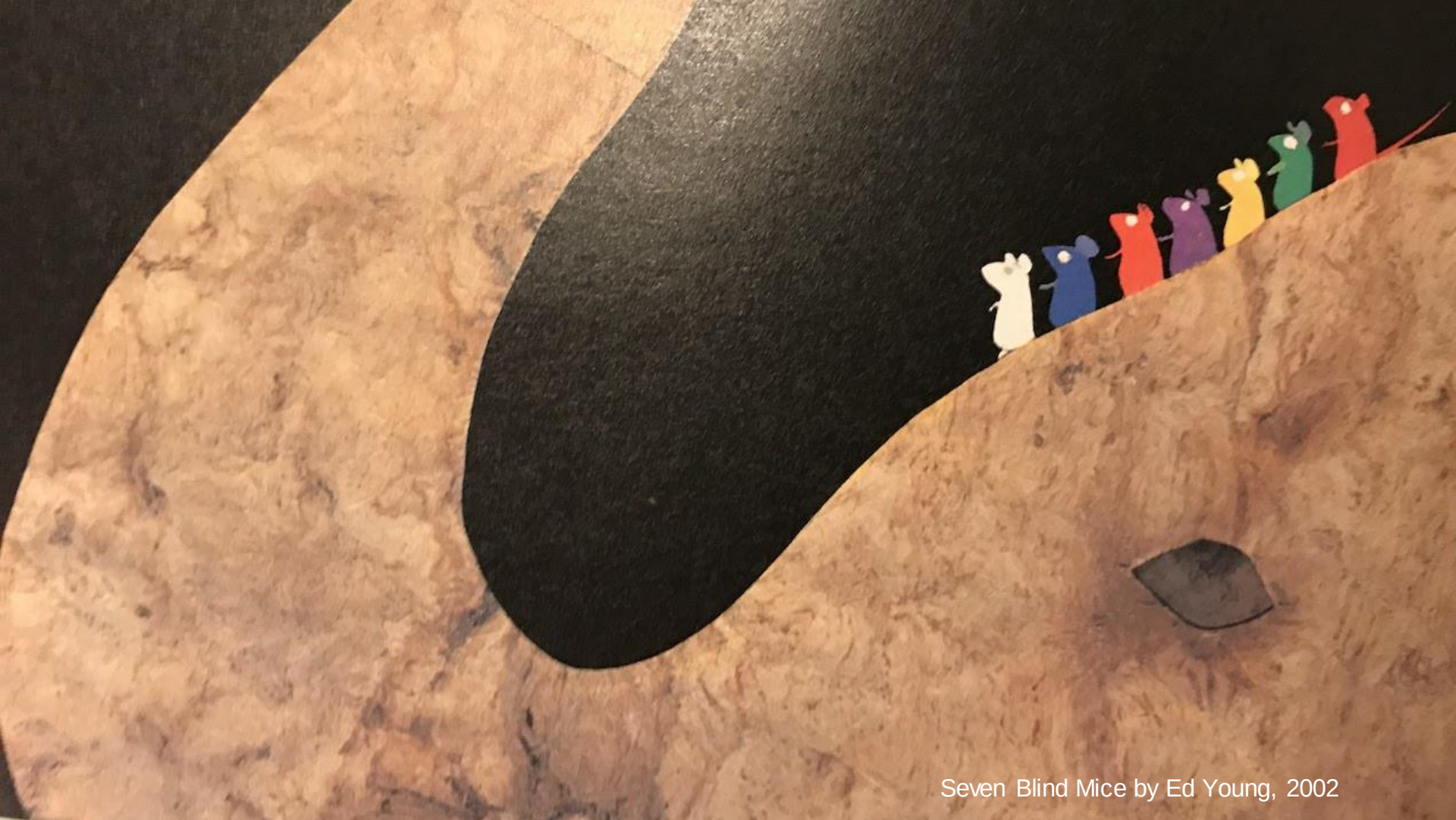


Seven Blind Mice by Ed Young, 2002

Command Query Responsibility Segregation

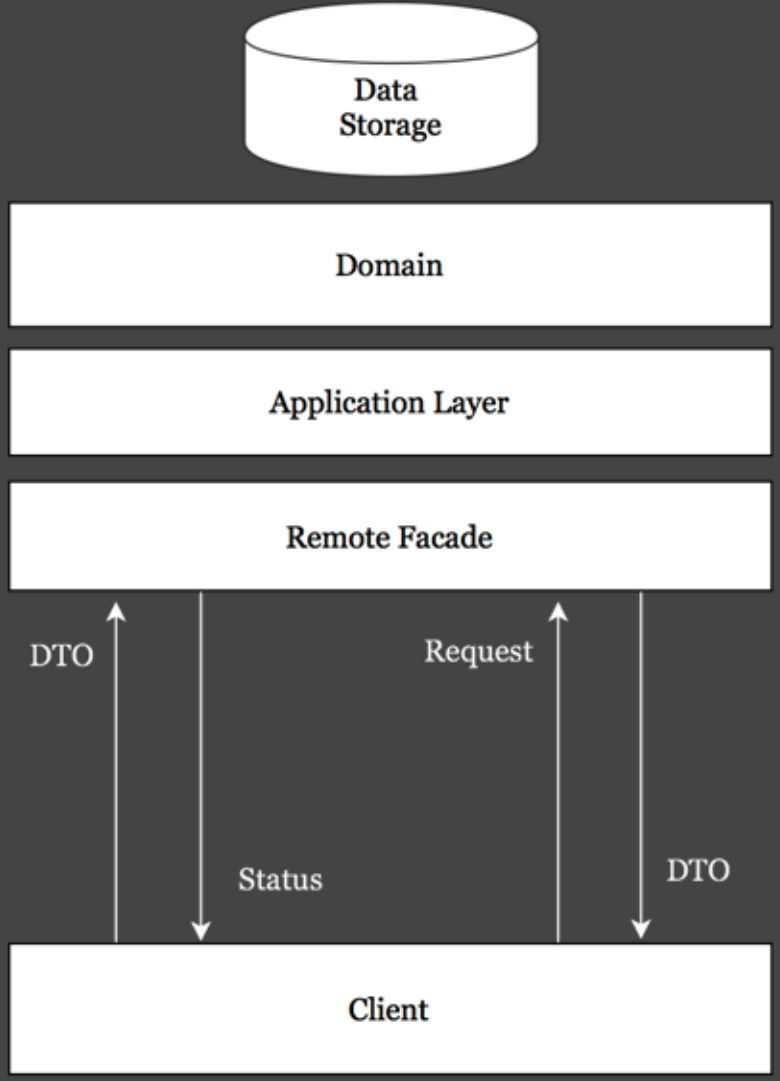
Andriy Drozdyuk

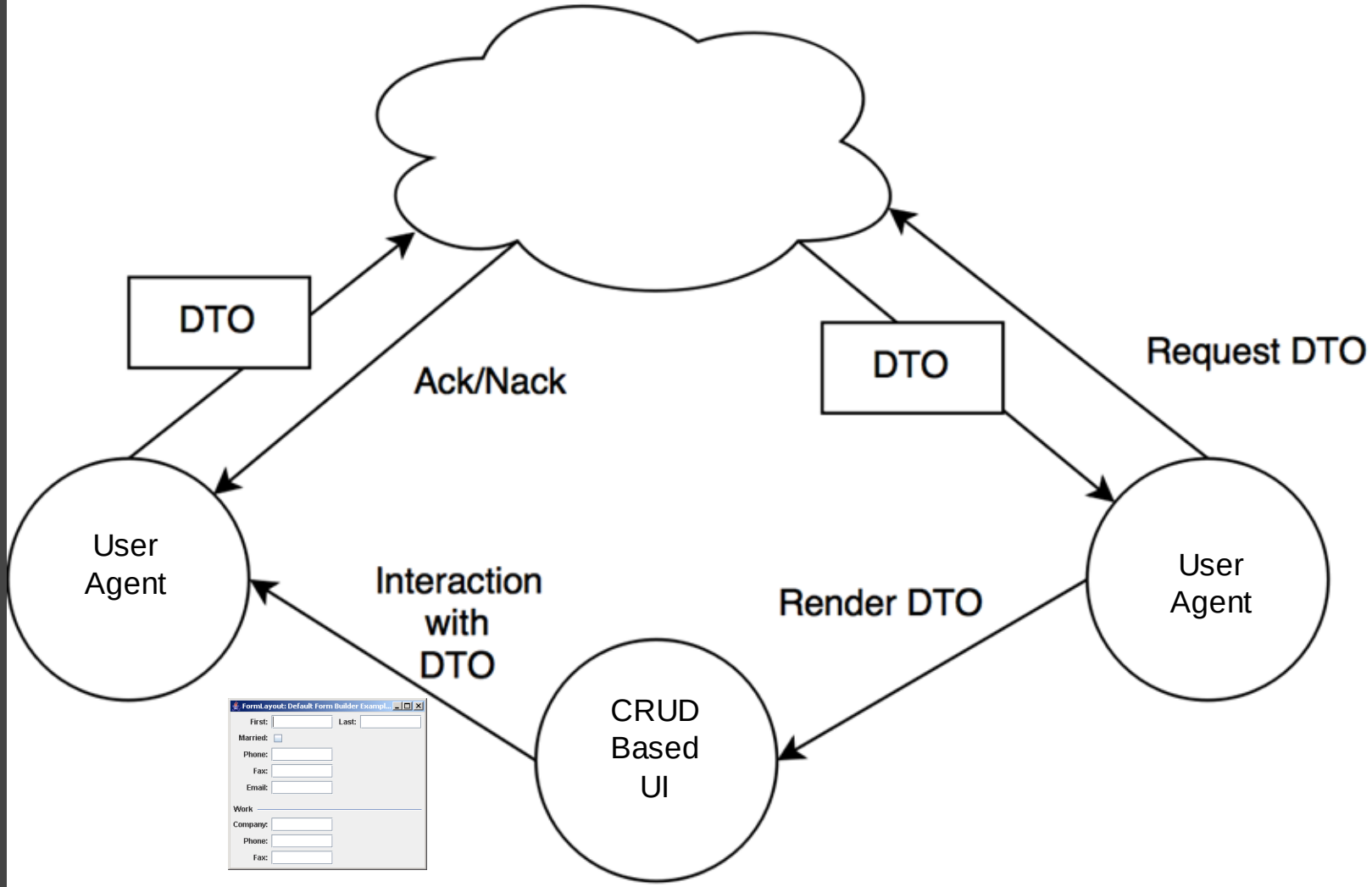
Outline

1. Stereotypical Architecture (SA) ..
2. Why? ..
3. Why Not? .....
4. Task Based UI vs. CRUD Based UI .....
5. CQRS .....
6. Command Side ...
7. Query Side ...
8. Command + Query ..

Length: 60 minutes

Stereotypical Architecture


Data
Storage
Domain
Application Layer
Remote Facade
DTO
Request
Status
DTO
Client


DTO
Ack/Nack
DTO
Request DTO
User Agent
Interaction with DTO
Render DTO
User Agent
CRUD Based UI
FormLayout: Default Form Builder Exampl...
First:
Last:
Married:
Phone:
Fax:
Email:
Work
Company:
Phone:
Fax:

Why?

Training

1. Any junior developer can be taught it
2. Very generic
3. Widely known
4. Removes the need to think about architecture

Tooling

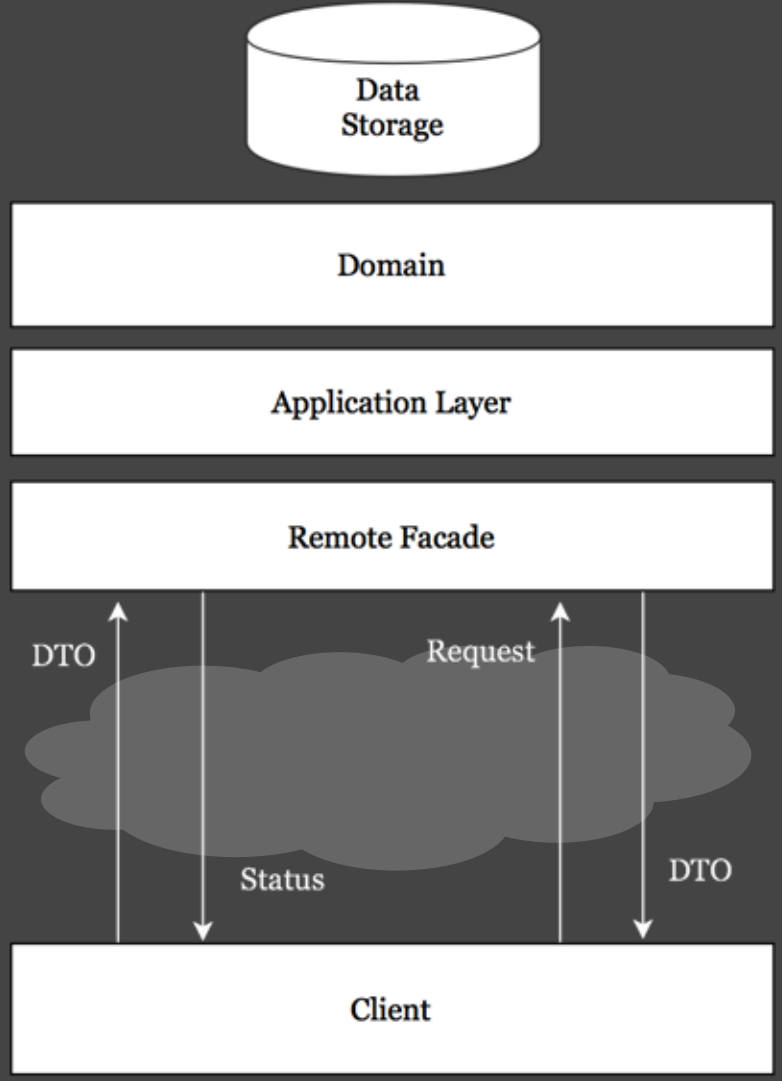
1. Frameworks
2. ORM
3. Mapping utilities

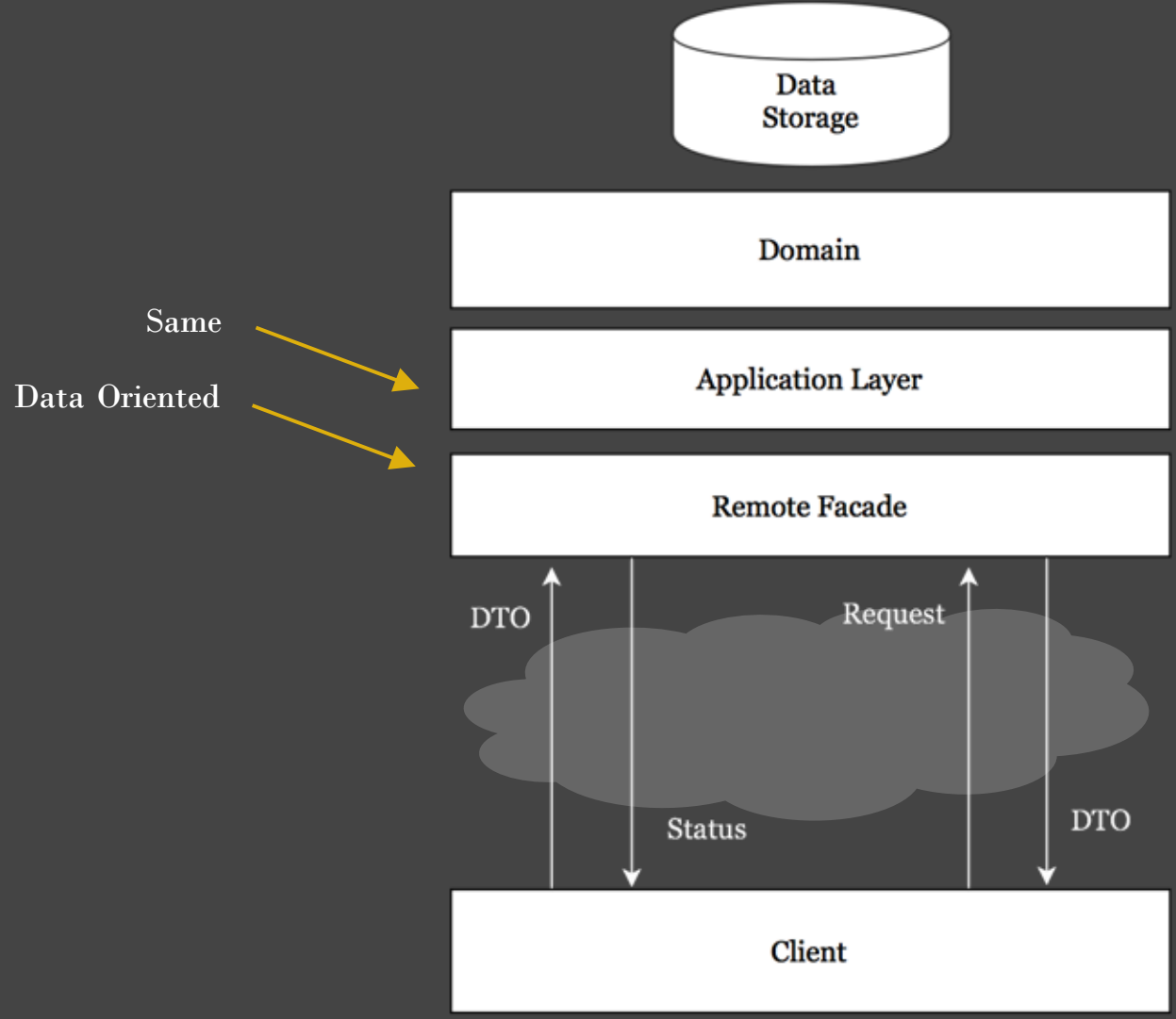
Why Not?

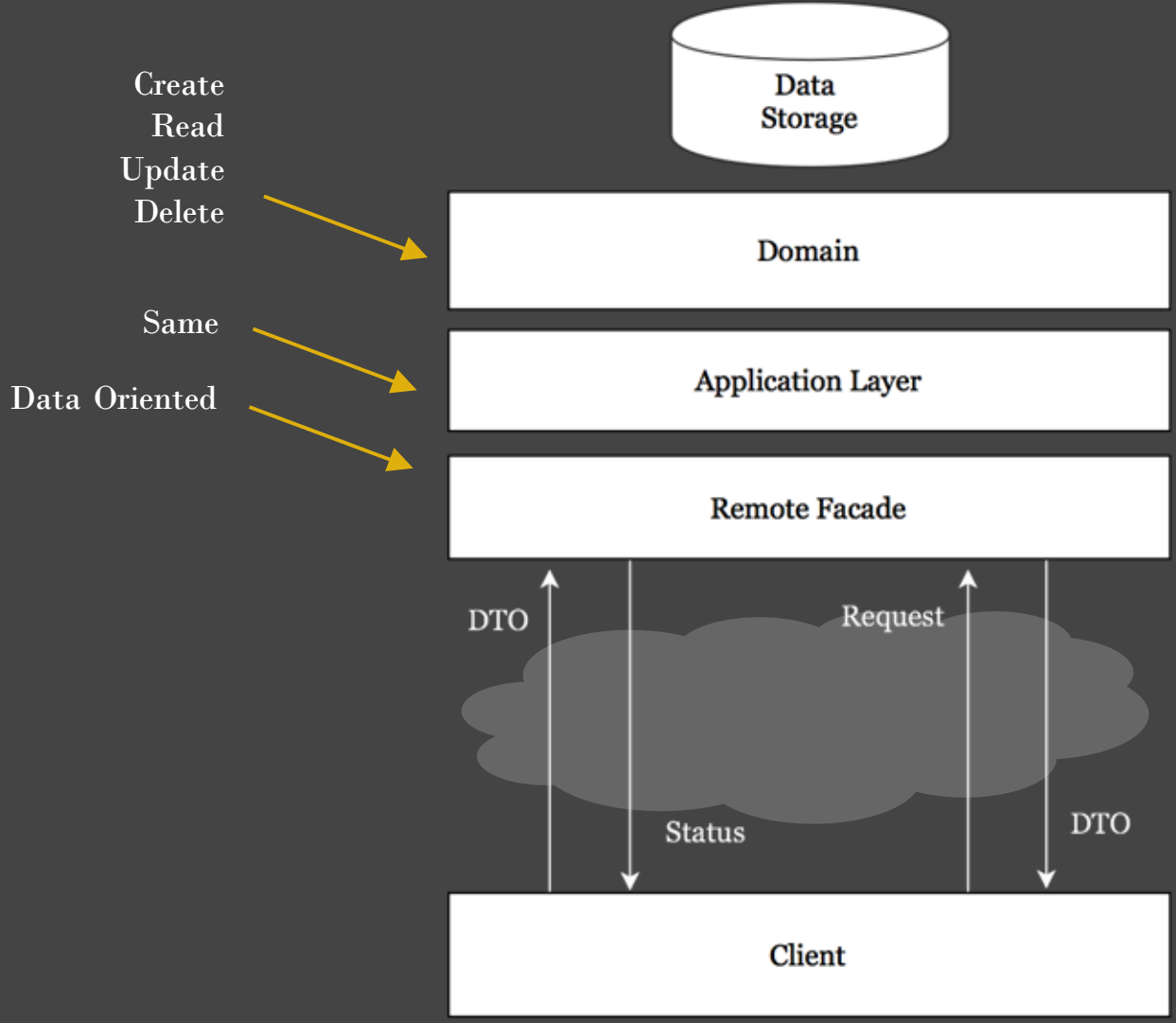
Downsides

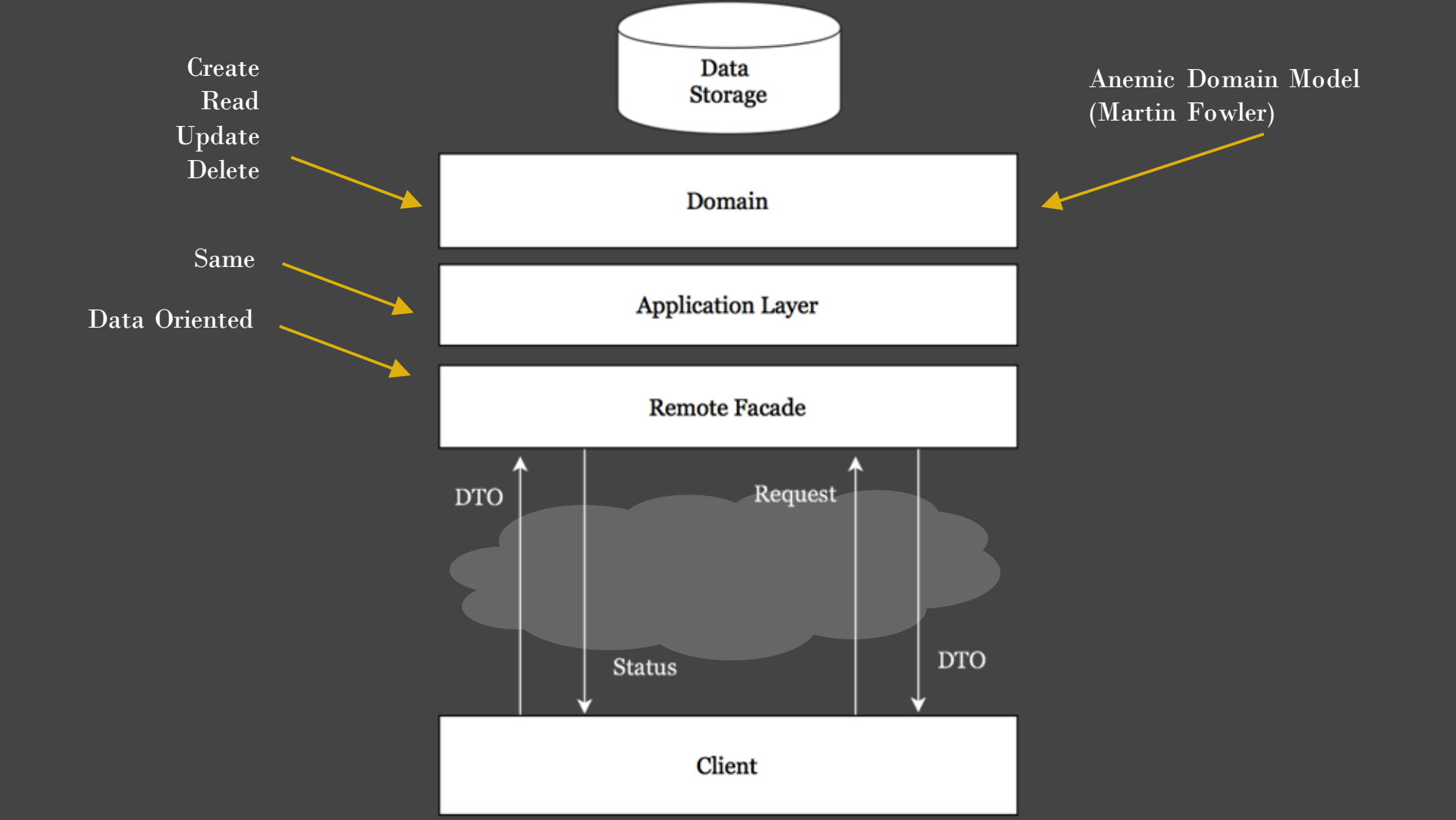
1. Impossible to do Domain Driven Design

Data Oriented



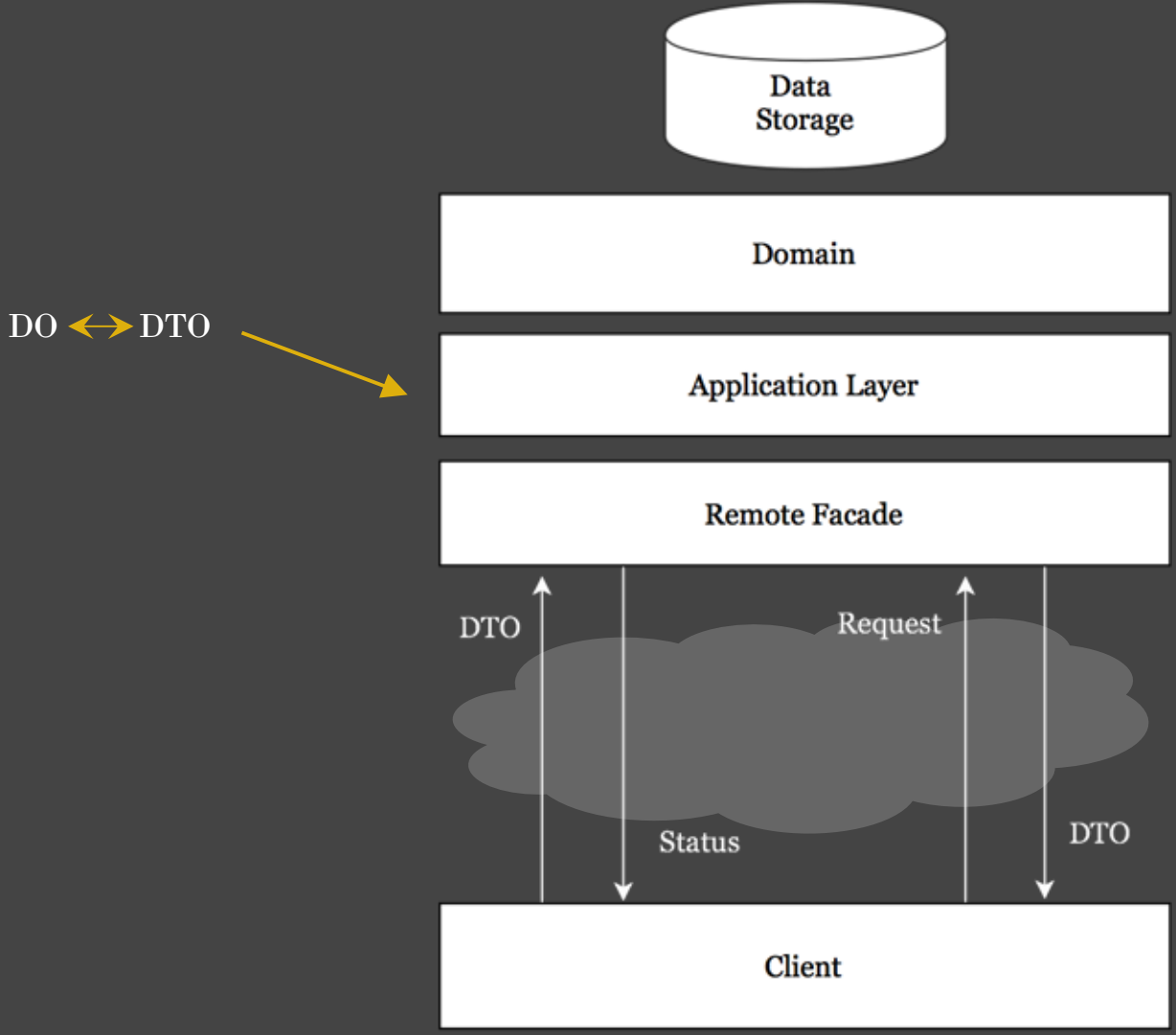

Data
Storage
Domain
Same
Application Layer
Data Oriented
Remote Facade
DTO
Request
Status
DTO
Client

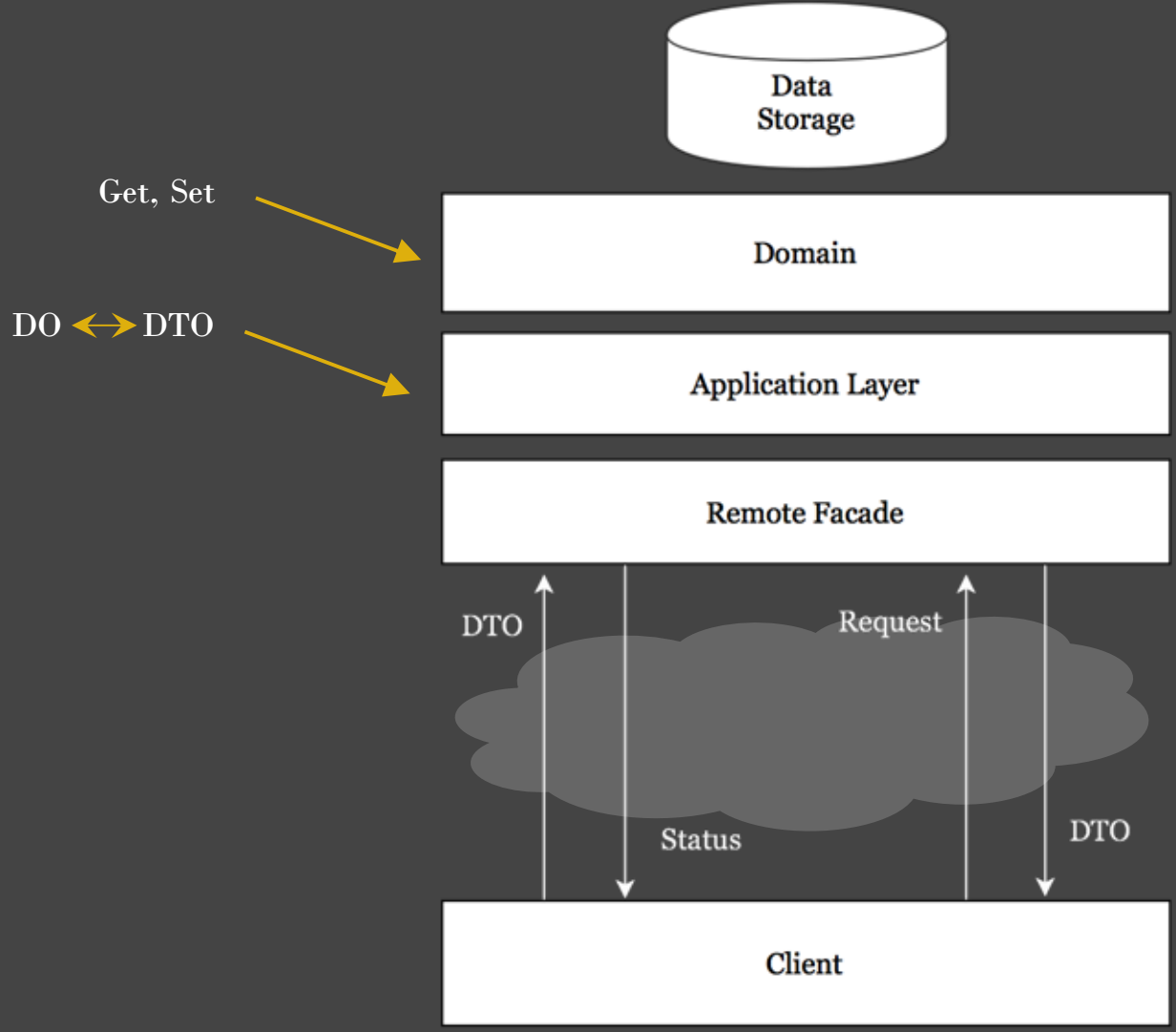

Create
Read
Update
Delete
Same
Data Oriented
Data Storage
Domain
Application Layer
Remote Facade
DTO
Request
Status
DTO
Client

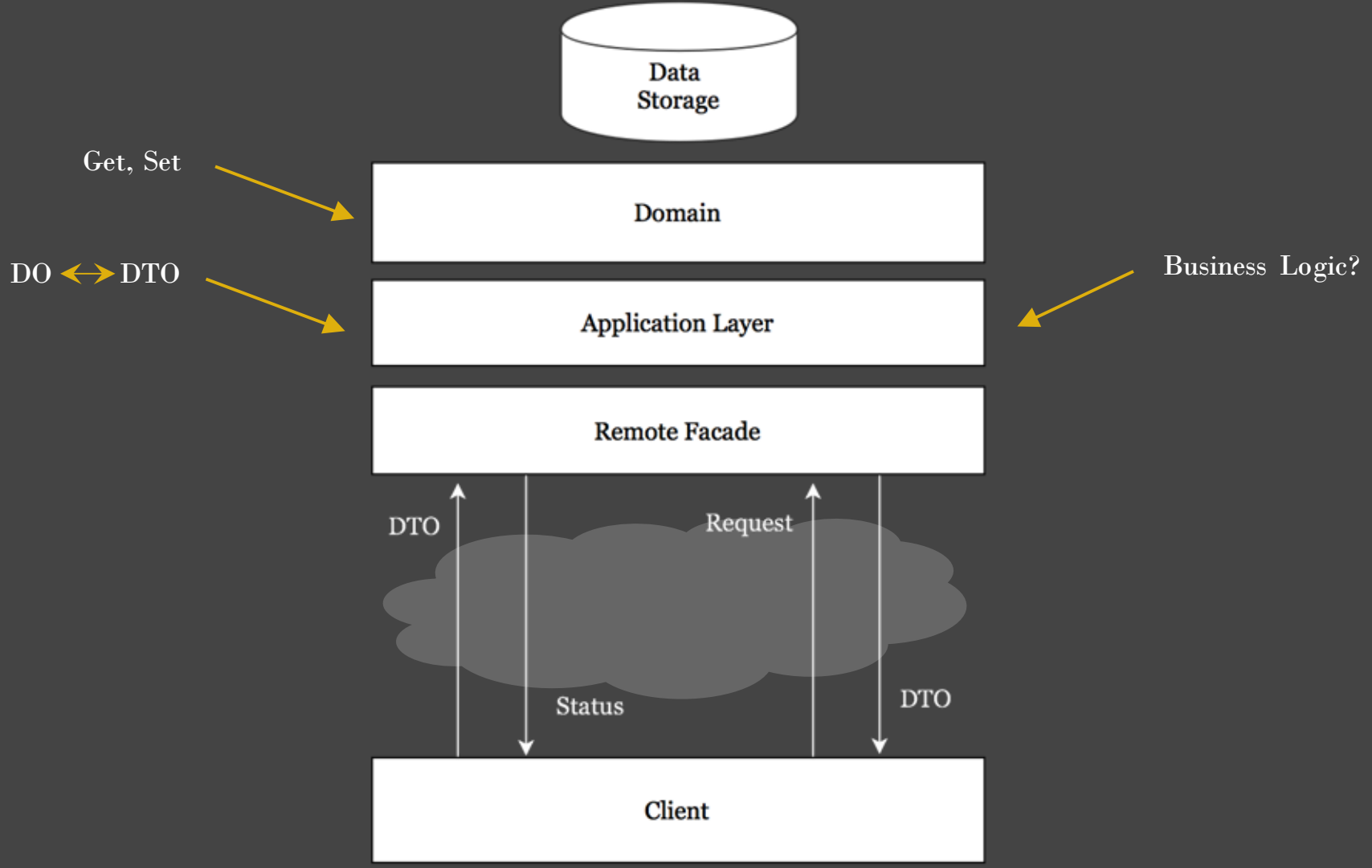

Create
Read
Update
Delete
Data
Storage
Anemic Domain Model
(Martin Fowler)
Domain
Same
Application Layer
Data Oriented
Remote Facade
DTO
Request
Status
DTO
Client

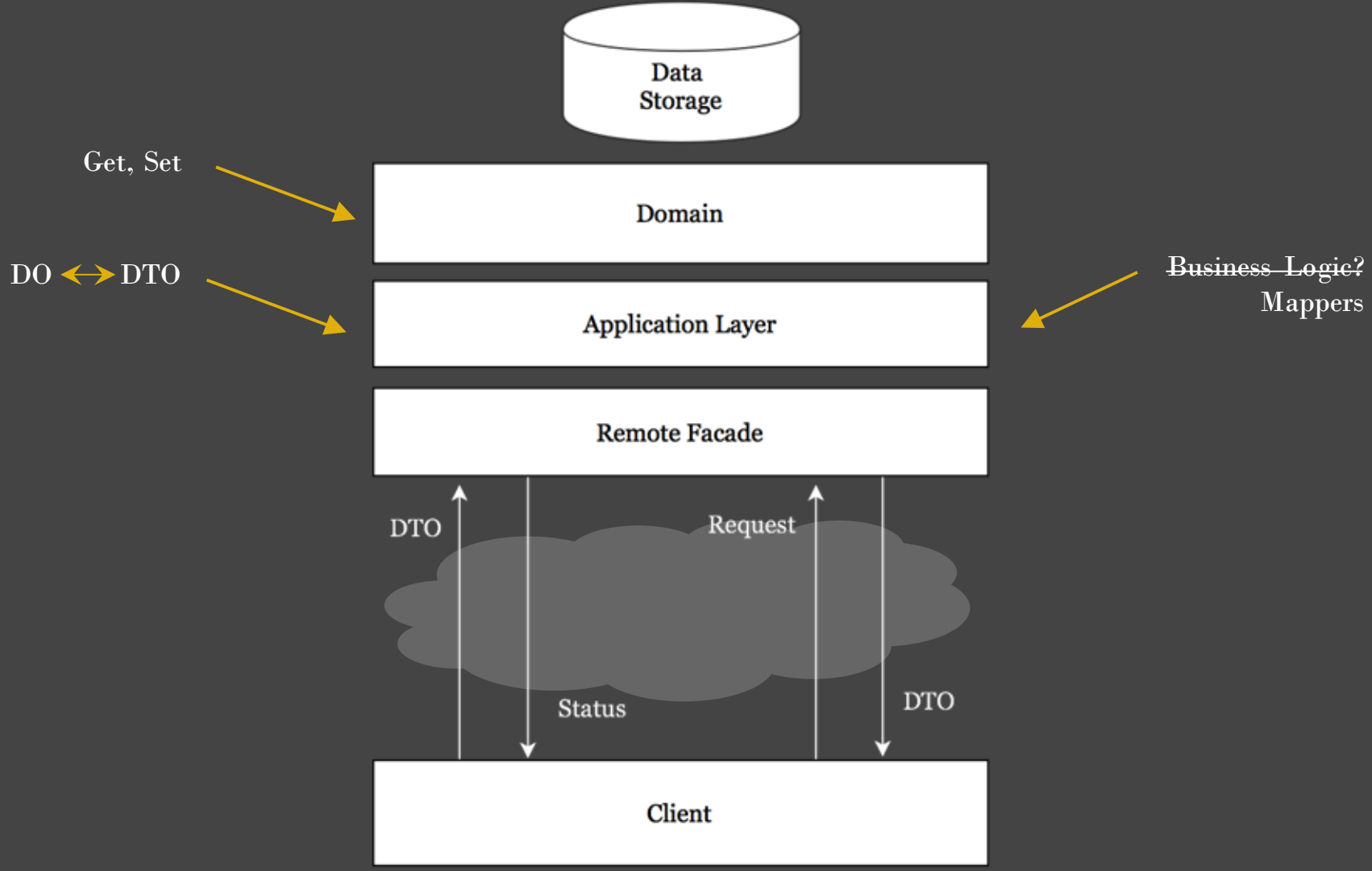
Anemic Domain Model

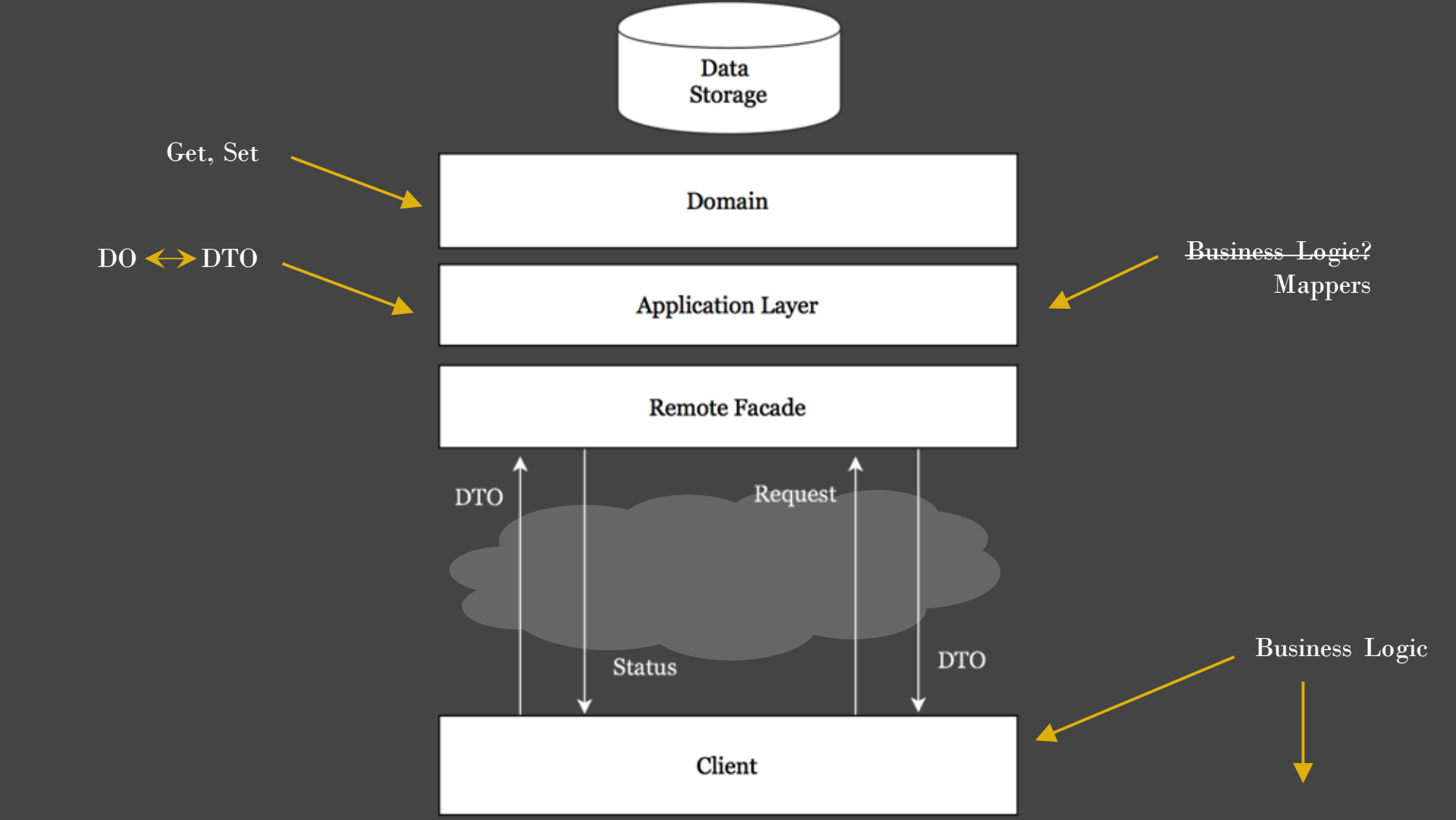
[T]here is hardly any behavior on these objects, making them little more than bags of getters and setters. … Instead there are a set of service objects which capture all the domain logic. These services live on top of the domain model and use the domain model for data. (Martin Fowler, 2003)


Data Storage
Domain
DO ↔ DTO
Application Layer
Remote Facade
DTO
Request
Status
DTO
Client


Data Storage
Get, Set
Domain
DO ↔ DTO
Application Layer
Remote Facade
DTO
Request
Status
DTO
Client


Data Storage
Get, Set
Domain
DO ↔ DTO
Application Layer
Business Logic?
Remote Facade
DTO
Request
Status
DTO
Client


Data Storage
Get, Set
Domain
DO ↔ DTO
Application Layer
~~Business Logic?~~
Mappers
Remote Facade
DTO
Request
Status
DTO
Client


Data Storage
Get, Set
Domain
DO ↔ DTO
Application Layer
~~Business Logic?~~
Mappers
Remote Facade
DTO
Request
Status
DTO
Business Logic
Client

This is far worse than the creation of an anemic model, this is the creation of a glorified excel spreadsheet.
-Greg Young

Scaling Characteristics

1. Bottlenecks
    a. Data storage
    b. Relational Databases: not horizontal, vertical expensive
2. Most systems don't need to scale

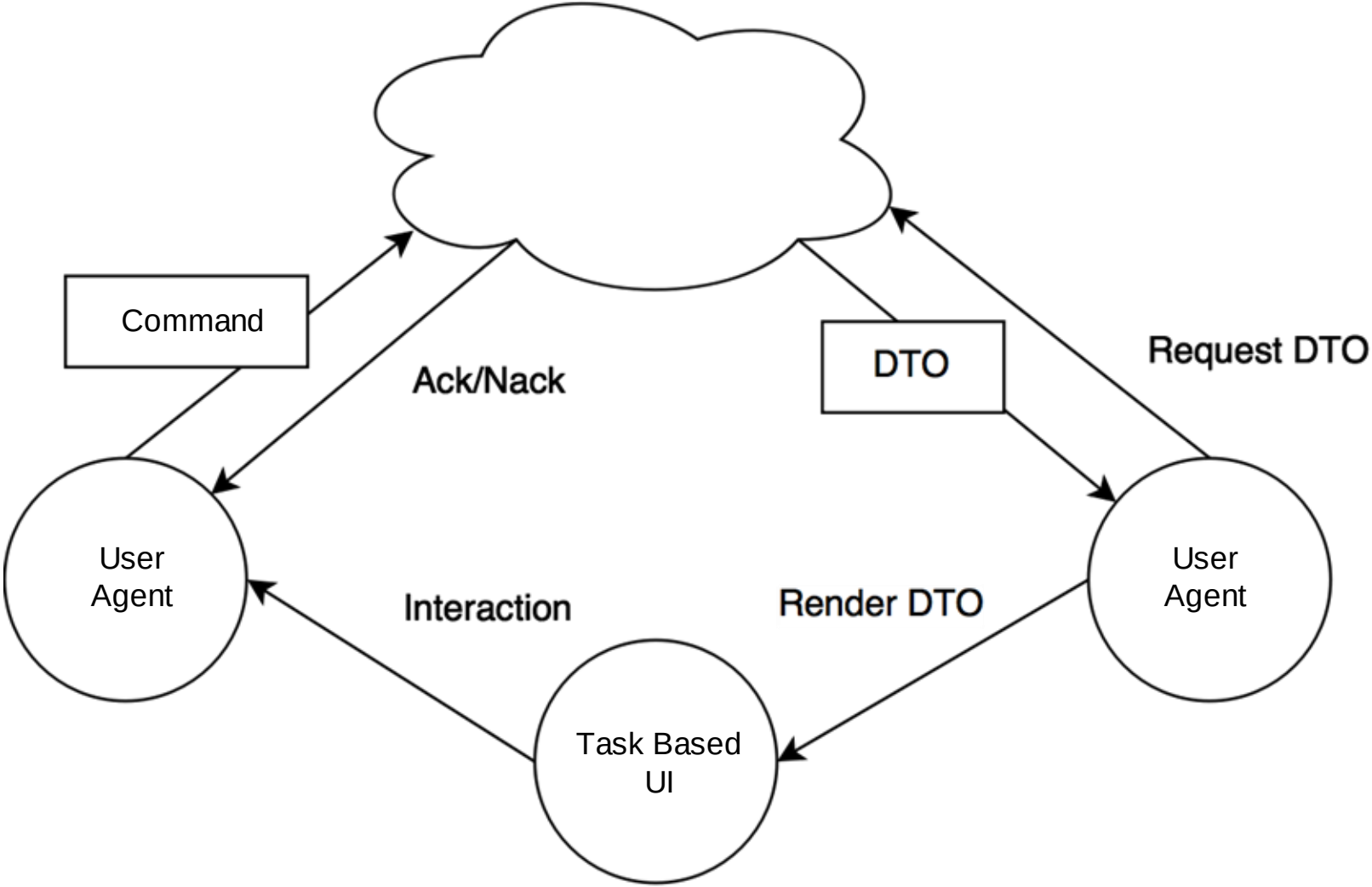
Task Based UI
vs.
CRUD Based UI

Problems with SA CRUD Based UI

1. Loses “intent” of the user
2. Data centric DTOs
3. Domain without verbs
4. Behavior lives in the client

Task Based UI

1. Bring intention of the user forward


Command
Ack/Nack
DTO
Request DTO
User Agent
Interaction
Render DTO
User Agent
Task Based UI

Commands

1. Command = operation name + data required
2. Imperative tone
3. Only carry necessary information

Operation name + data required

```
class ReserveSeat(flightId:GUID, seat:Int)
```

Imperative tone

`CompleteSale`

vs:

`SaleOccured`

Only required information

```
CompleteSale(orderId:GUID, amount:Price)
```

vs.

```
class SaleDTO {
  orderId: GUID,
  amount: Price,
  cleared: Bool,
  isDeleted: False,
  preferred: False
}
```

Good or Bad?

ChangeAddress

Good or Bad?

ChangeAddress

Try “Correct Address”, “Relocate Customer”

Good or Bad?

CreateUser

Good or Bad?

CreateUser

Try “RegisterUser”

Good or Bad?

DeleteCourse

Good or Bad?

DeleteCourse

Try “UnenrollStudent”

Commands

1. Not difficult but unfamiliar
2. Seen as a lot of work

CQRS

Command Query Responsibility Segregation

Split model in two

1. One containing Commands
2. One containing Queries

```
trait CustomerService {
  def makePreferred(customer_id: Int)
  def getCustomer(customer_id:Int): Customer
  def getCustomersWithName(name: String): Set[Customer]
  def getPreferredCustomers(): Set[Customer]
  def changeCustomerLocale(customer_id:Int, locale:Locale)
  def createCustomer(customer: Customer)
  def editCustomerDetails(details: CustomerDetails)
}
```

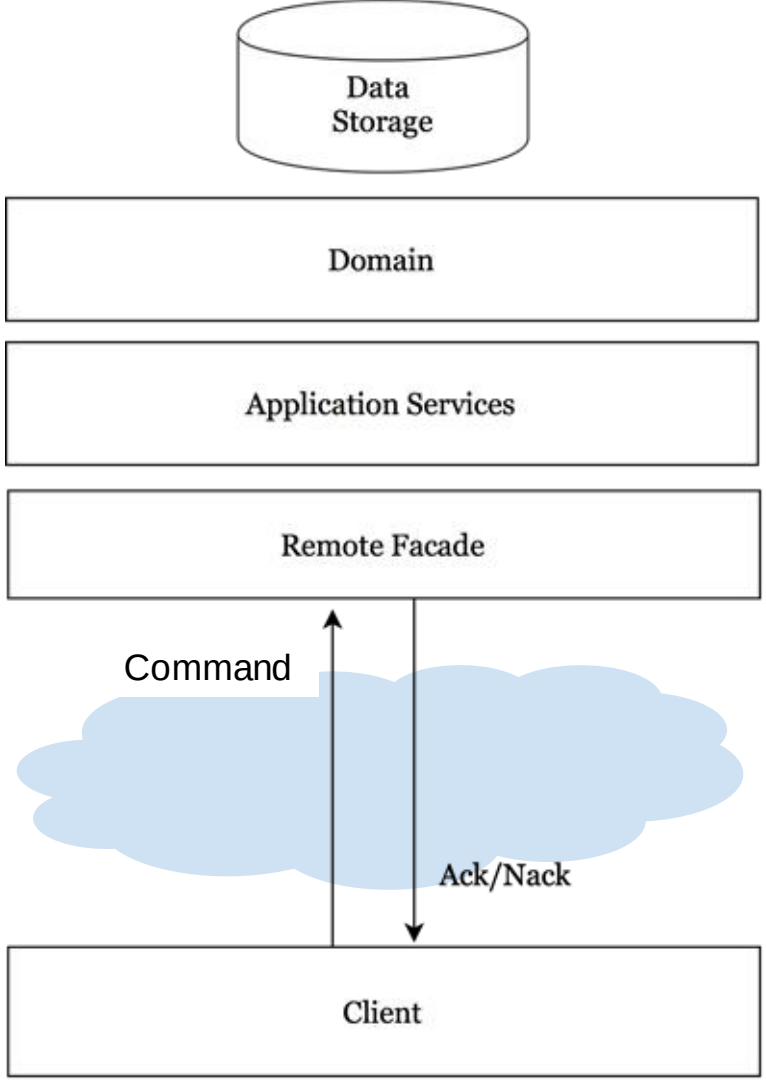
```
trait CustomerService {
  def makePreferred(customer_id: Int)
  def getCustomer(customer_id:Int): Customer
  def getCustomersWithName(name: String): Set[Customer]
  def getPreferredCustomers(): Set[Customer]
  def changeCustomerLocale(customer_id:Int, locale:Locale)
  def createCustomer(customer: Customer)
  def editCustomerDetails(details: CustomerDetails)
}

trait CustomerWriteService {
  def makePreferred(customer_id: Int)
  def changeCustomerLocale(customer_id:Int, locale:Locale)
  def createCustomer(customer: Customer)
  def editCustomerDetails(details: CustomerDetails)
}

trait CustomerReadService {
  def getCustomer(customer_id:Int): Customer
  def getCustomersWithName(name: String): Set[Customer]
  def getPreferredCustomers(): Set[Customer]
}
```

	Command	*Query*
1. Consistency	Consistent	Eventual
2. Data Storage	3NF	1NF
3. Scalability	Few transactions	Many requests

It is not possible to create an optimal solution for searching, reporting, and processing transactions utilizing a single model

-Greg Young

Command Side


Data
Storage
Domain
Application Services
Remote Facade
Command
Ack/Nack
Client

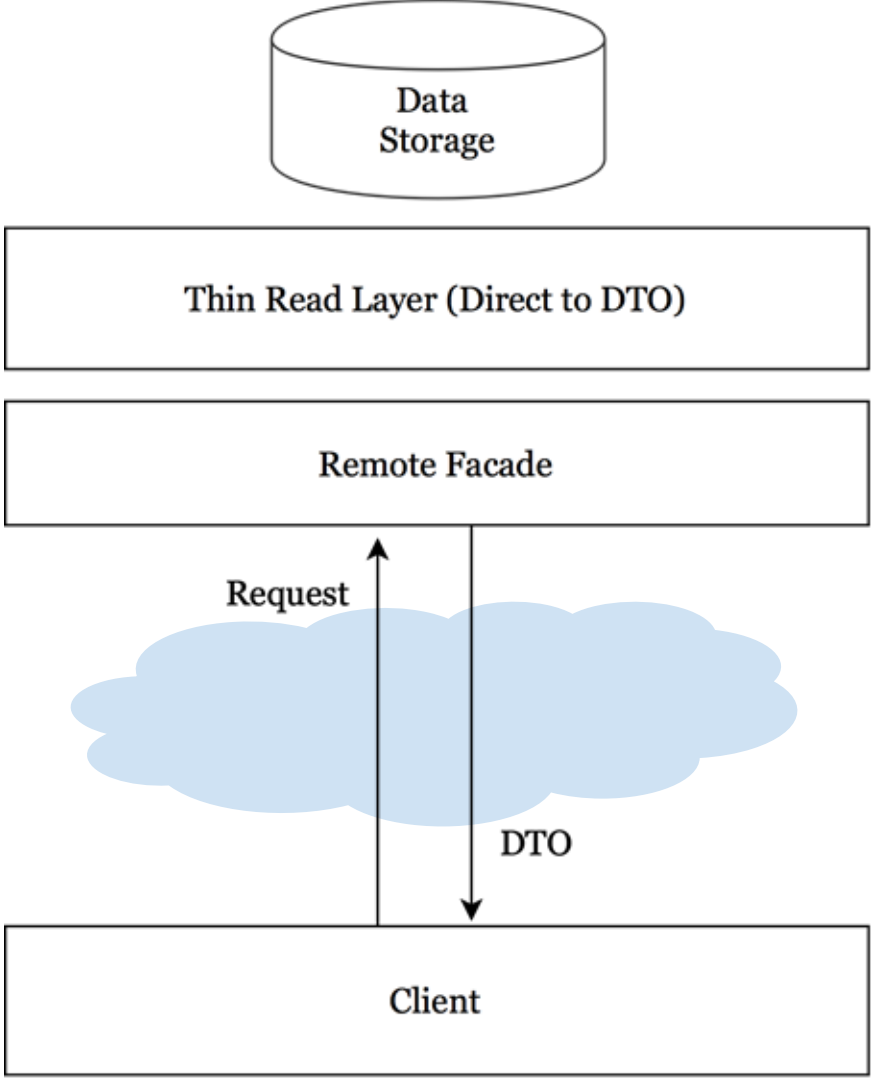
Problems with Commands in SA

1. Repositories cluttered with read methods (paging, sorting)
2. Getters on Domain Objects expose internal state
3. Prefetching not needed
4. Loading multiple ARs blurs their boundaries

Benefits

1. Domain focused on processing commands
2. Domain Objects no longer expose state
3. Repositories with few query methods (i.e. getById)
4. Aggregate boundaries are preserved

Query Side


Data
Storage
Thin Read Layer (Direct to DTO)
Remote Facade
Request
DTO
Client

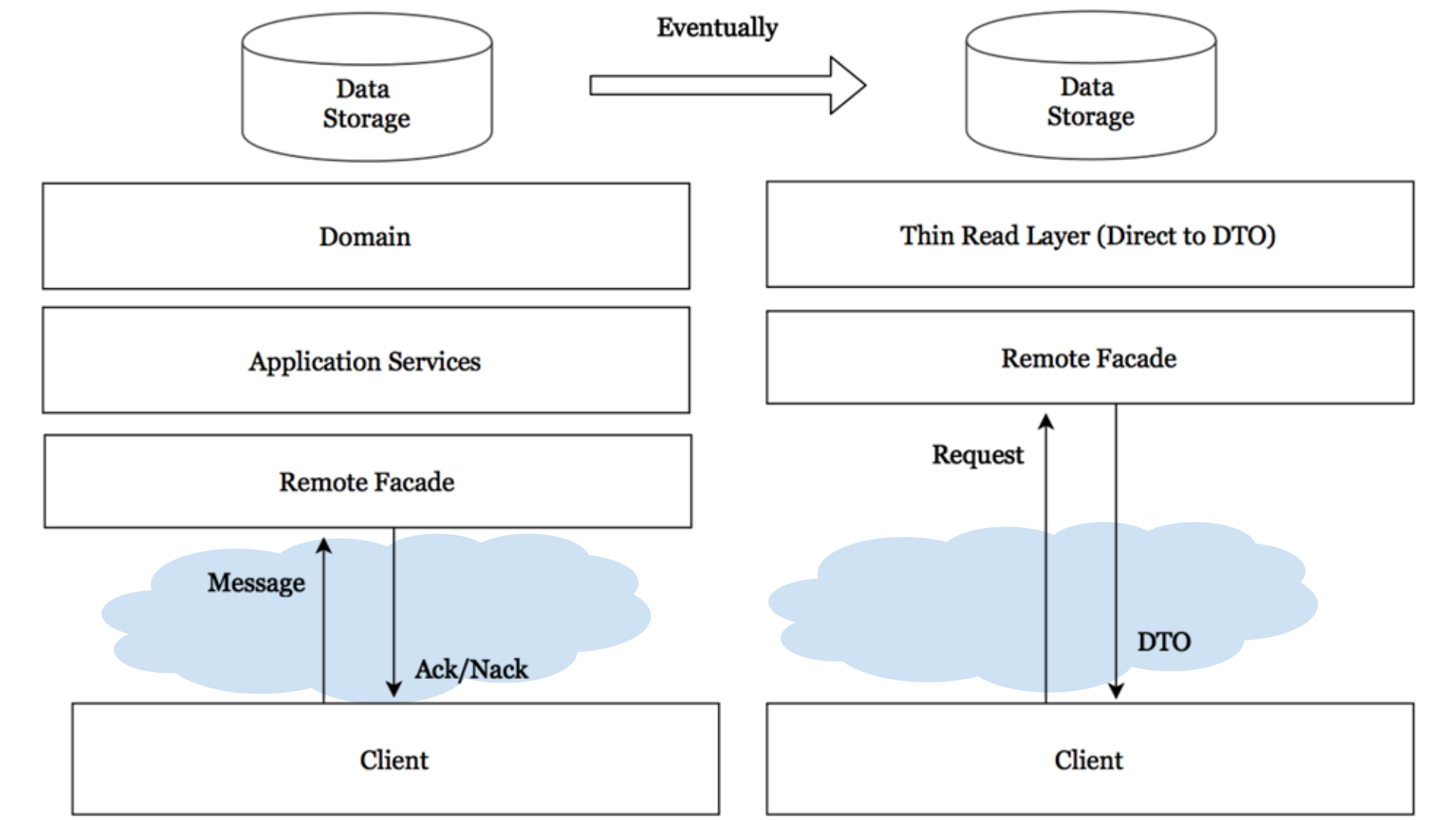
Problems with Queries in SA

1. Repositories need read methods (paging, sorting)
2. Domain Objects need getters
3. Prefetching is needed
4. Loading multiple ARs causes non-optimal querying
5. Optimizing queries is hard

Benefits

1. Does not suffer from impedance mismatch
2. Queries can be optimized, fine-tuned
3. Developers only have to understand data model
4. Developers don't need to know ORM
5. Domain not affected by query requirements & can be specialized

Command + Query


Data Storage
Eventually
Data Storage
Domain
Application Services
Remote Facade
Message
Ack/Nack
Client
Thin Read Layer (Direct to DTO)
Remote Facade
Request
DTO
Client

Separate Models Costs & Benefits

1. Relatively easy to separate Queries and Commands
2. Lowers cost of optimization (querying)
3. Domain model - less conceptual overhead
4. Worst case: `Cost(CQRS) = Cost(SA)`
   (occurs when Query Side uses domain underneath)

Look at

1. Domain Events
2. Event Sourcing
3. Actors

References

1. CQRS Documents by Greg Young
   https://cqrs.wordpress.com/documents/
2. Eric Evans, Domain-Driven Design, 2004.
3. Fighting bottlenecks with CQRS
   http://verraes.net/2013/12/fighting-bottlenecks-with-cqrs/
4. Microsoft CQRS
   https://www.microsoftpressstore.com/articles/article.aspx?p=2248809
5. CQRS Pocket Guide
   http://cqrs.wikidot.com/

The End

this slide left intentionally blank

Extra Slides

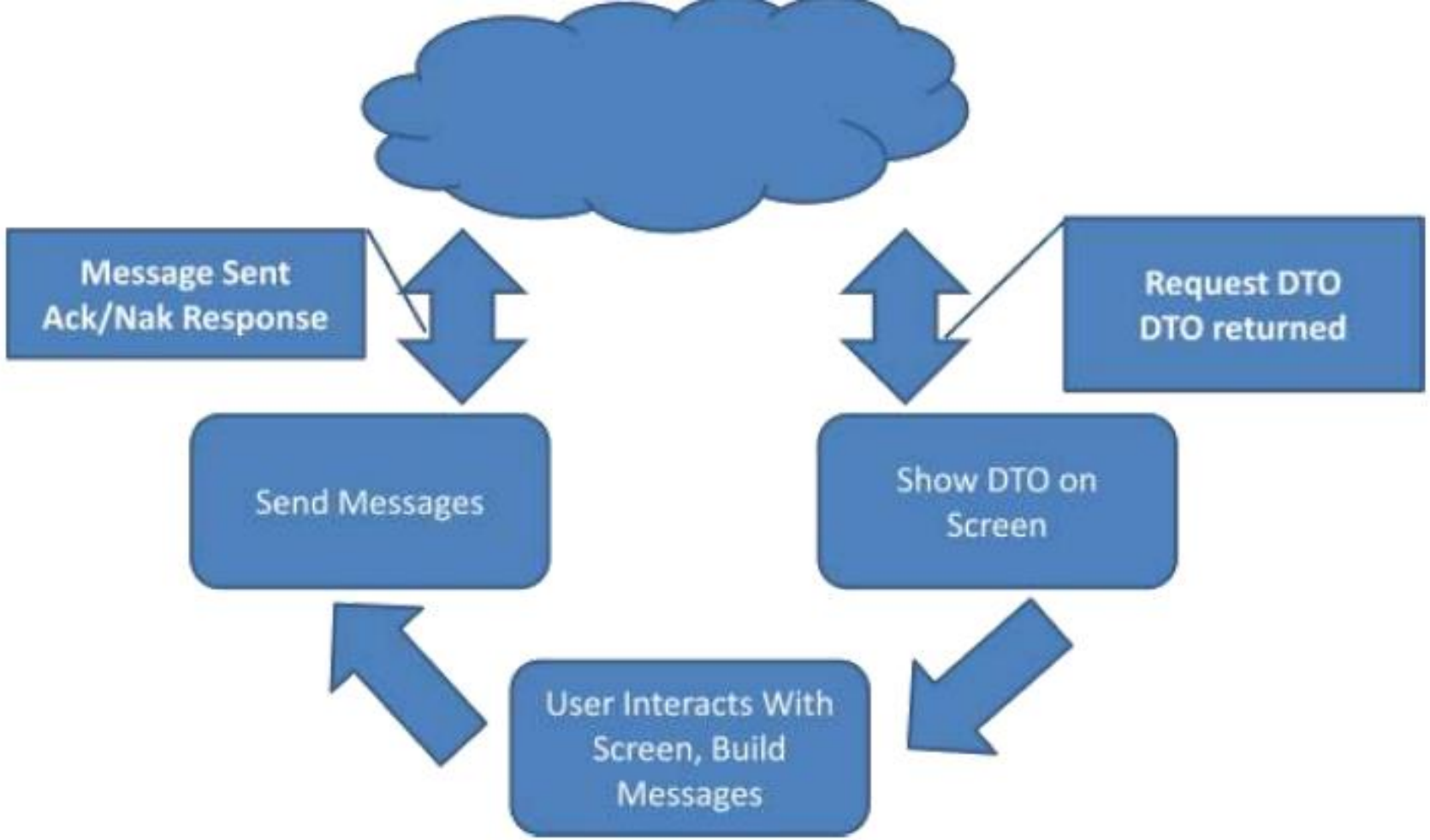


Figure 5 Behavioral Interface

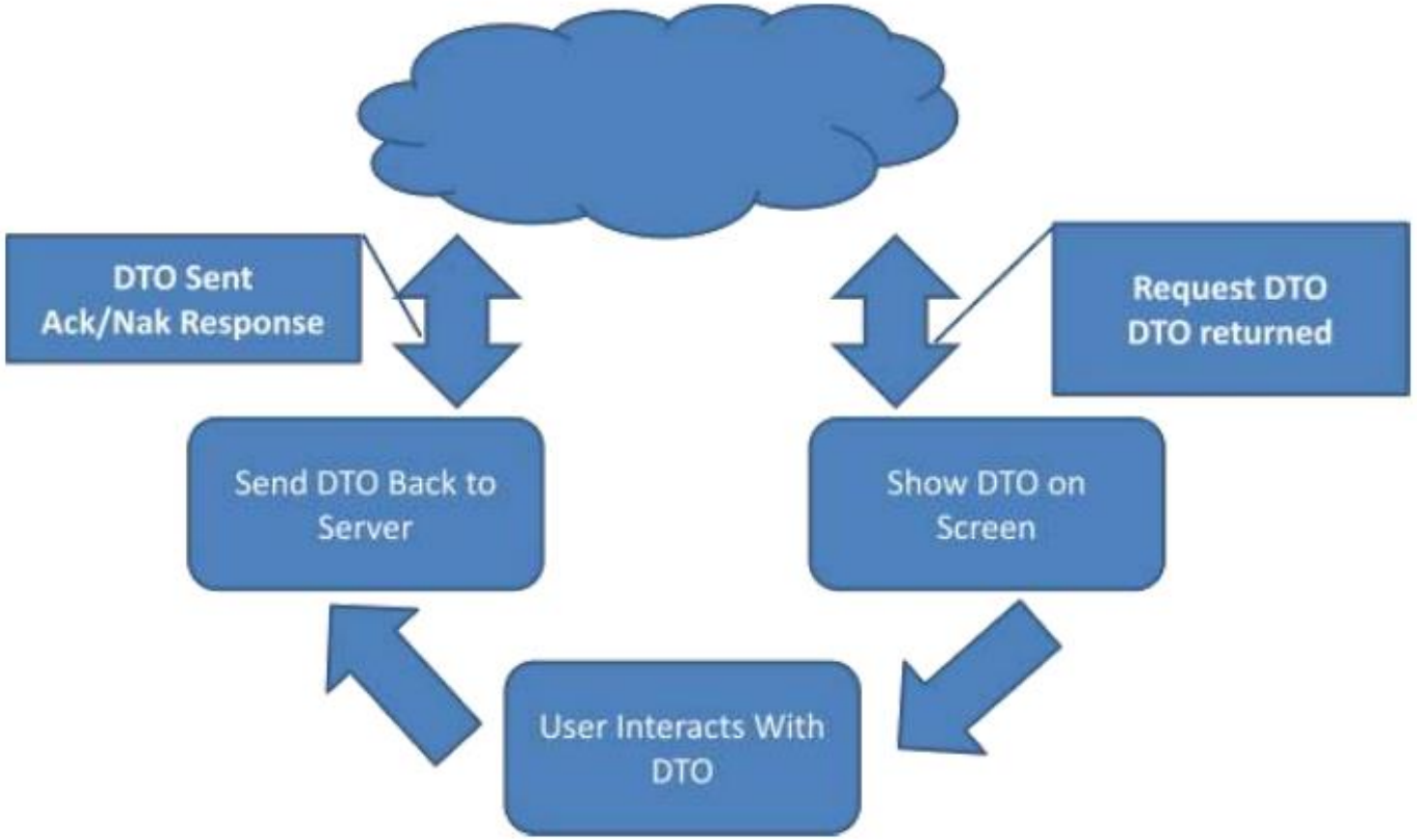
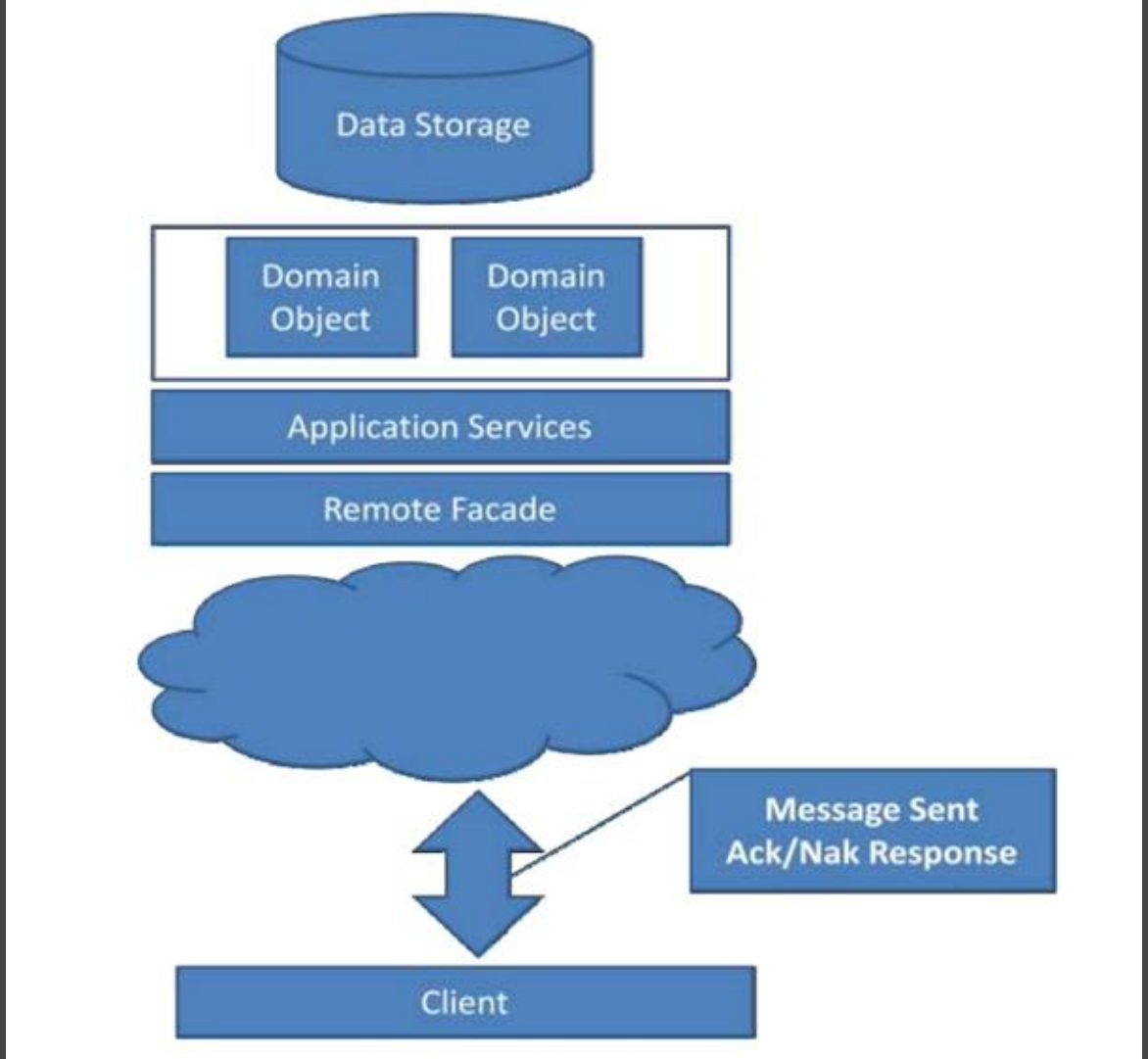
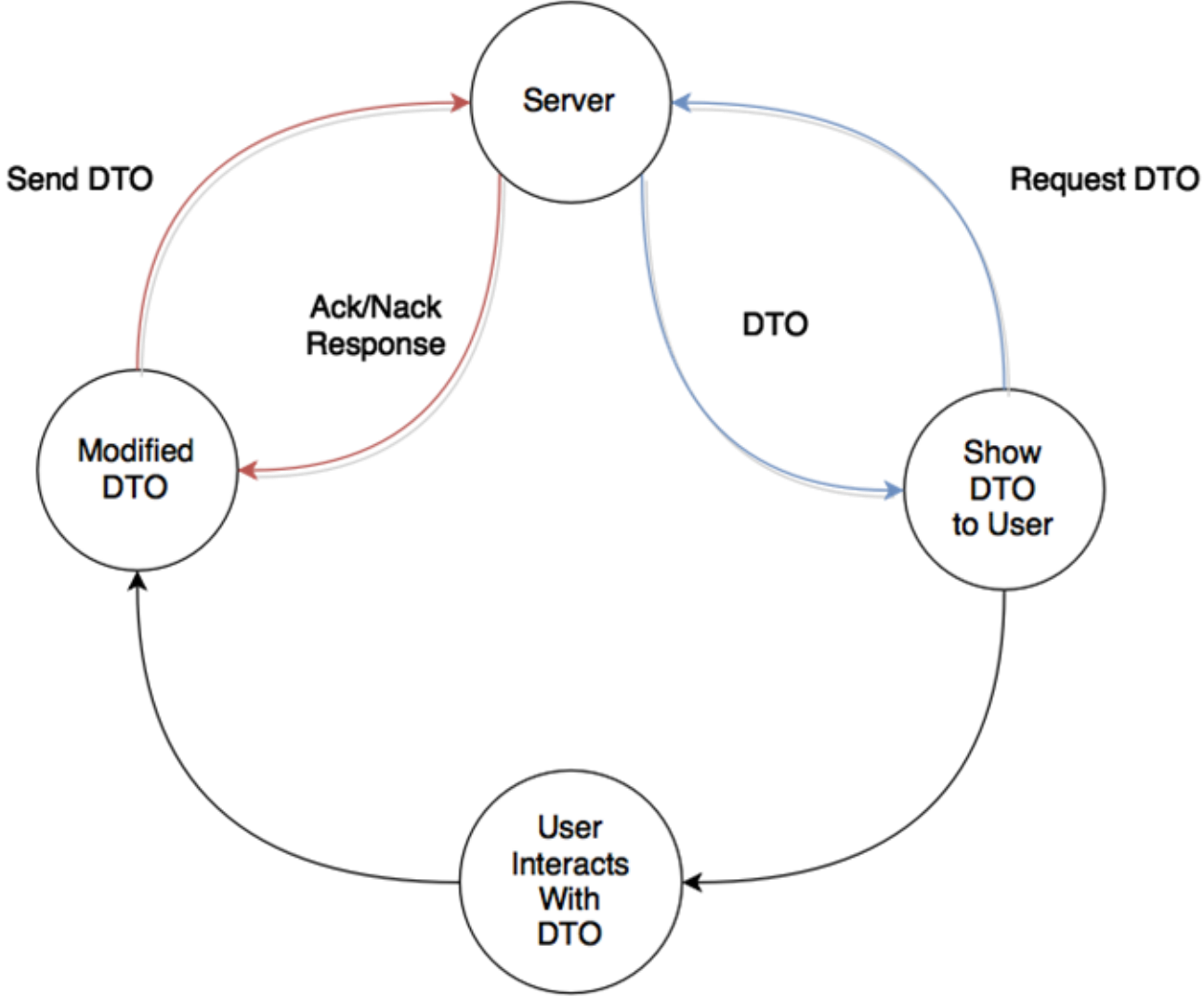
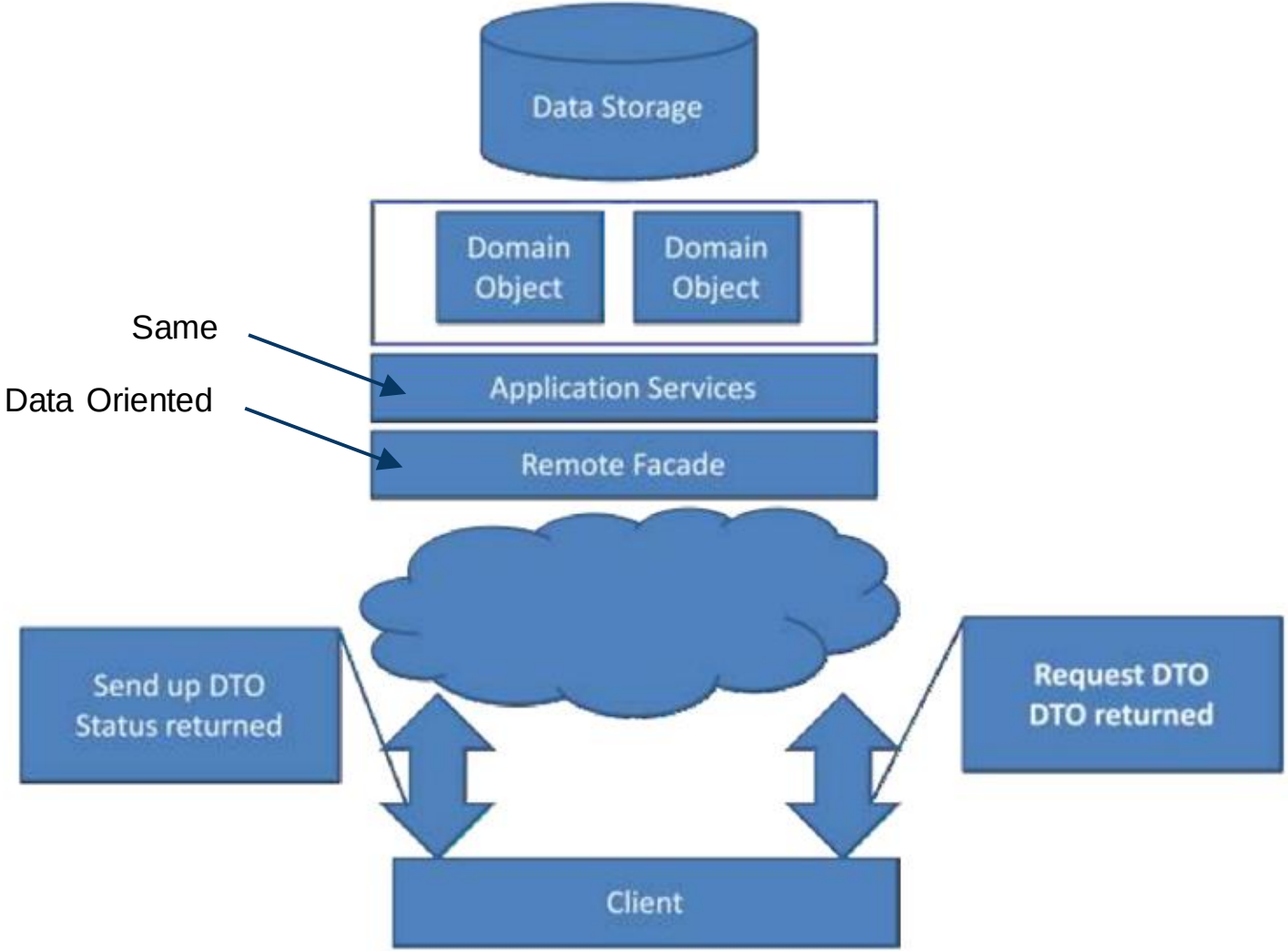
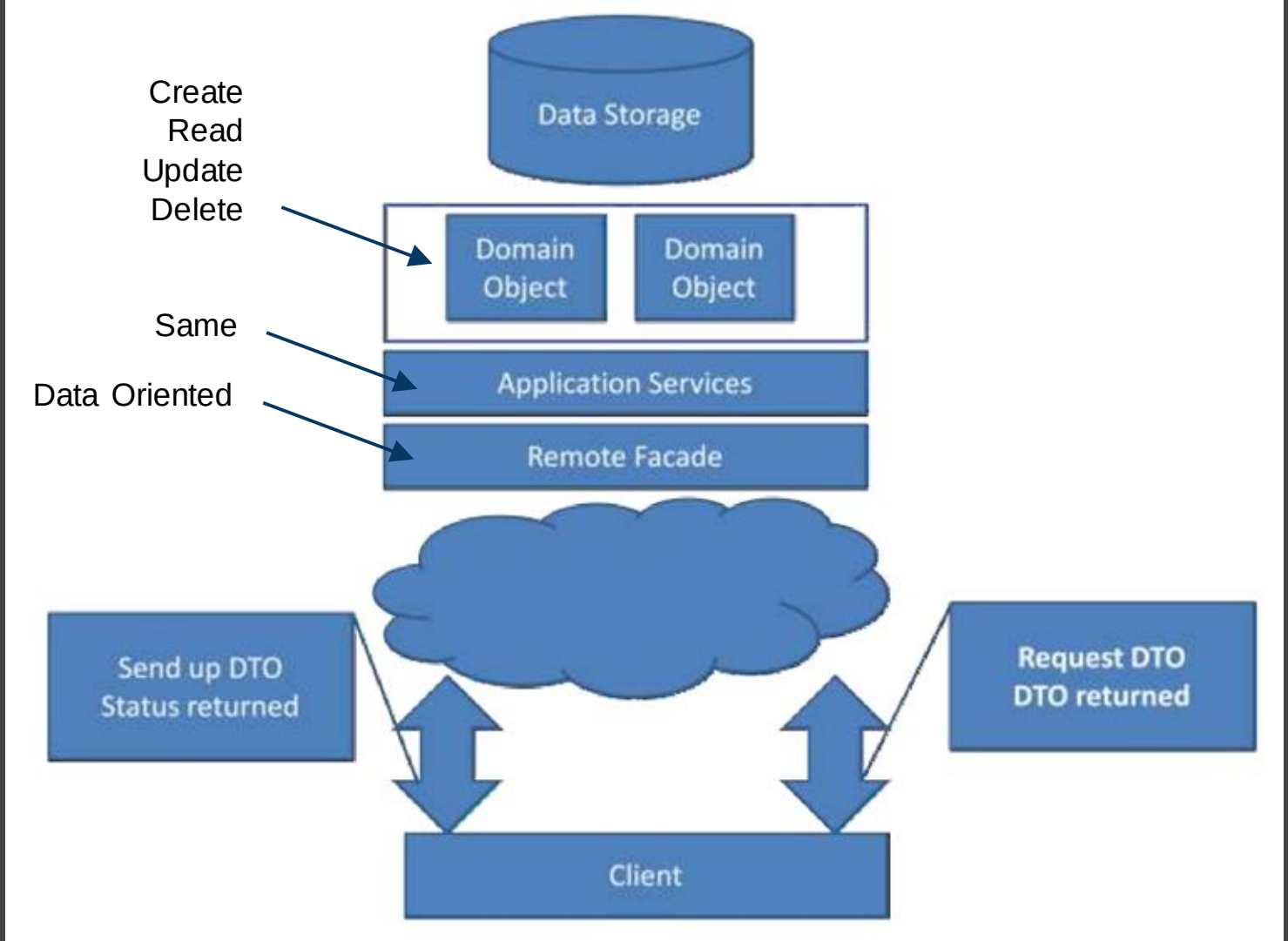


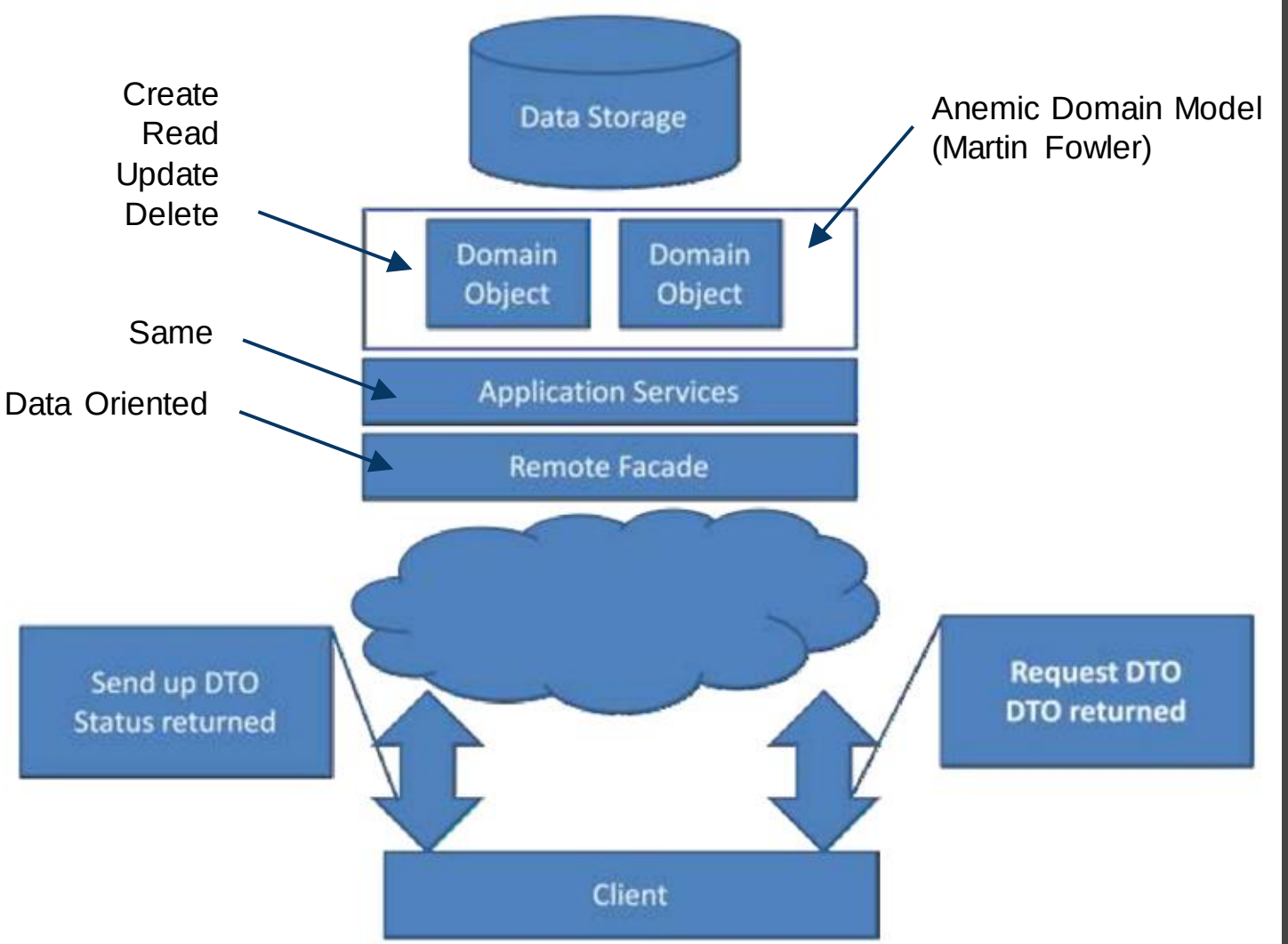
Figure 4 Interaction in a DTO Up/Down Architecture

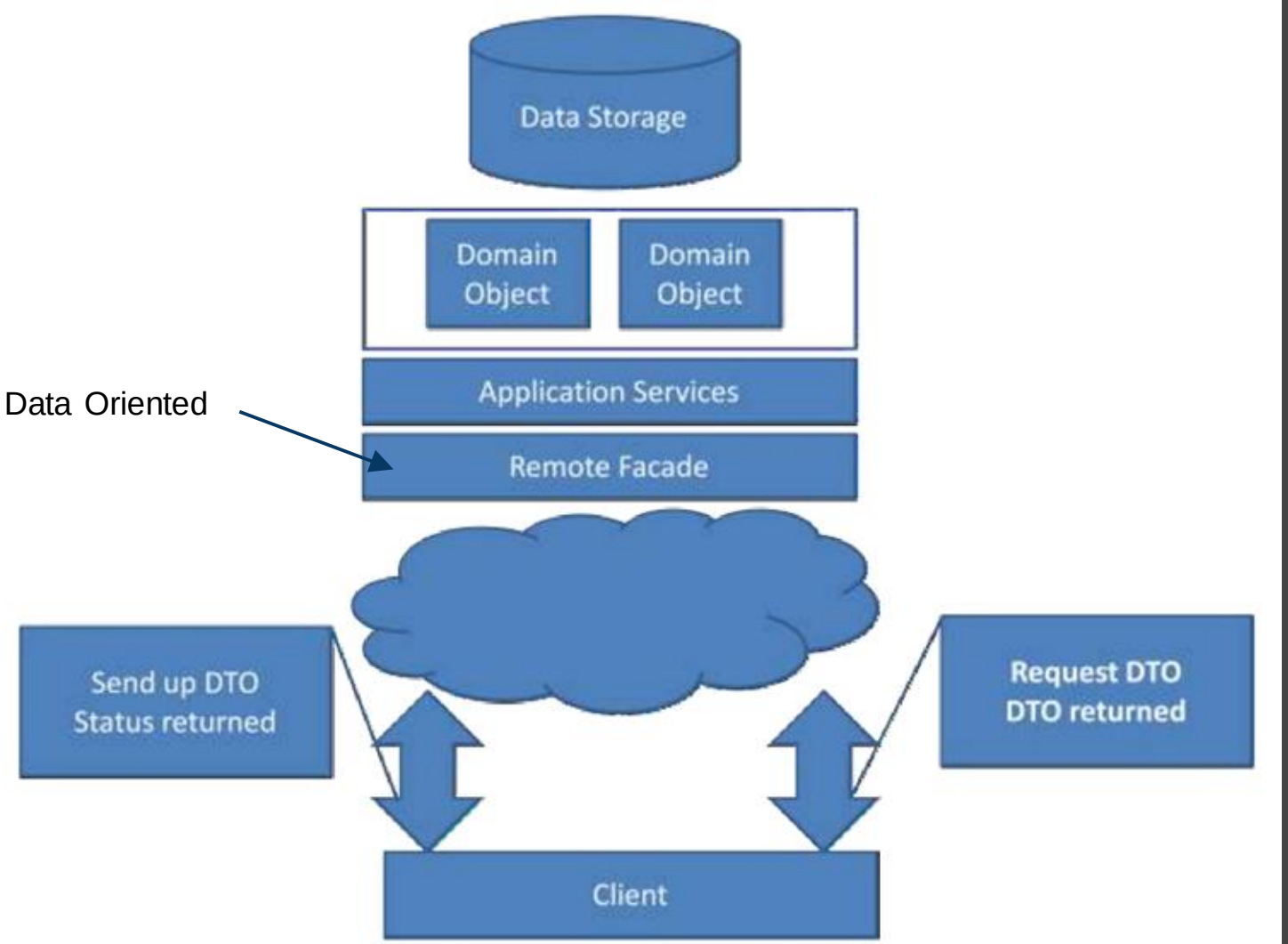

Data Storage
Domain Object
Domain Object
Application Services
Remote Facade
Message Sent
Ack/Nak Response
Client


Server
Send DTO
Request DTO
Ack/Nack
Response
DTO
Modified
DTO
Show
DTO
to User
User
Interacts
With
DTO

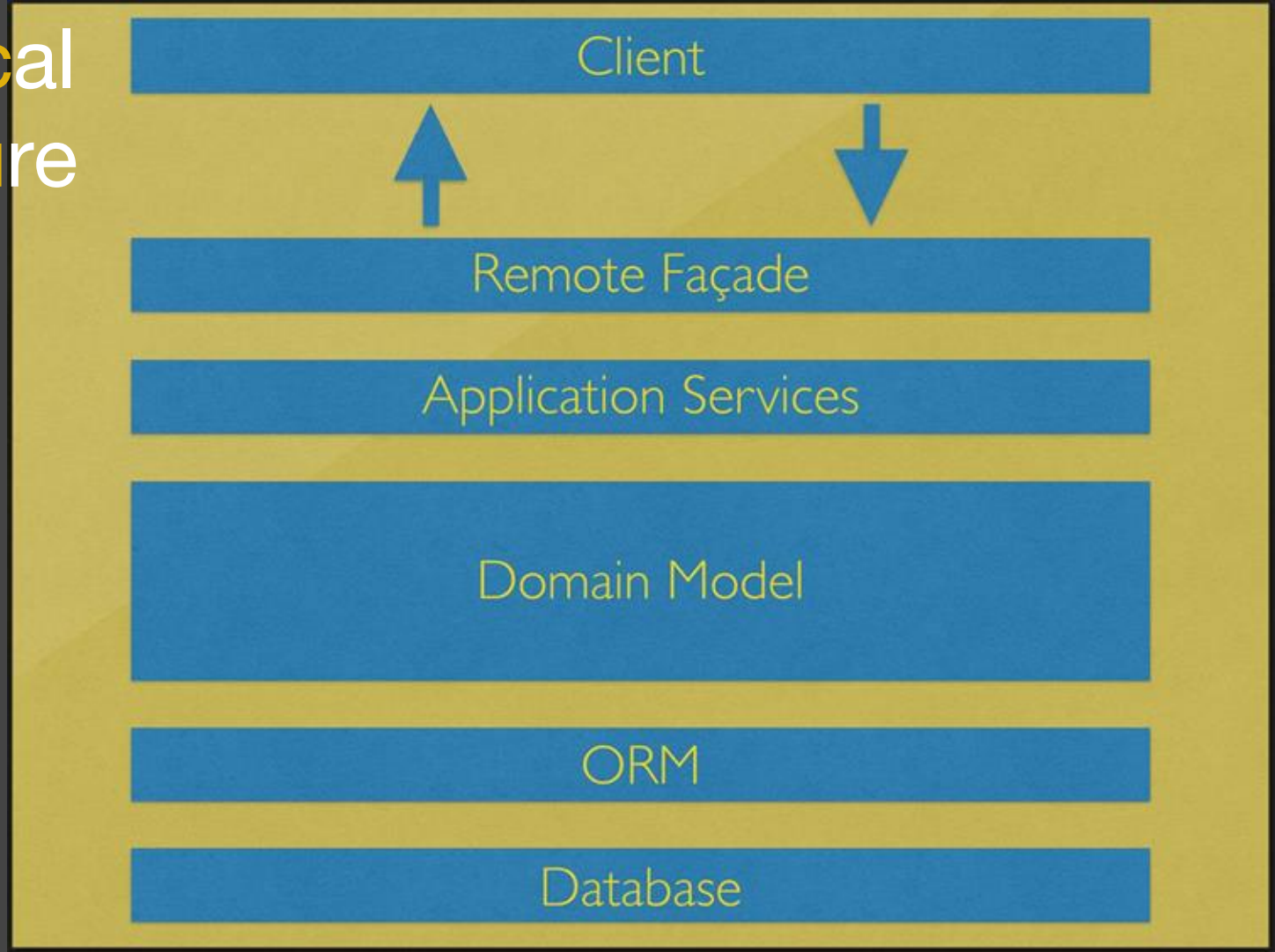

Data Storage
Domain Object
Domain Object
Same
Application Services
Data Oriented
Remote Facade
Send up DTO
Status returned
Request DTO
DTO returned
Client


Create
Read
Update
Delete
Same
Data Oriented
Data Storage
Domain Object
Domain Object
Application Services
Remote Facade
Send up DTO
Status returned
Request DTO
DTO returned
Client


Create
Read
Update
Delete
Same
Data Oriented
Data Storage
Anemic Domain Model
(Martin Fowler)
Domain Object
Domain Object
Application Services
Remote Facade
Send up DTO
Status returned
Request DTO
DTO returned
Client


Data Storage
Domain Object
Domain Object
Application Services
Remote Facade
Data Oriented
Send up DTO
Status returned
Request DTO
DTO returned
Client

Prototypical Architecture



CustomerService

void MakeCustomerPreferred(CustomerId)

Customer GetCustomer(CustomerId)

CustomerSet GetCustomersWithName(Name)

CustomerSet GetPreferredCustomers()

void ChangeCustomerLocale(CustomerId, NewLocale)

void CreateCustomer(Customer)

void EditCustomerDetails(CustomerDetails)

CustomerWriteService

void MakeCustomerPreferred(CustomerId)

void ChangeCustomerLocale(CustomerId, NewLocale)

void CreateCustomer(Customer)

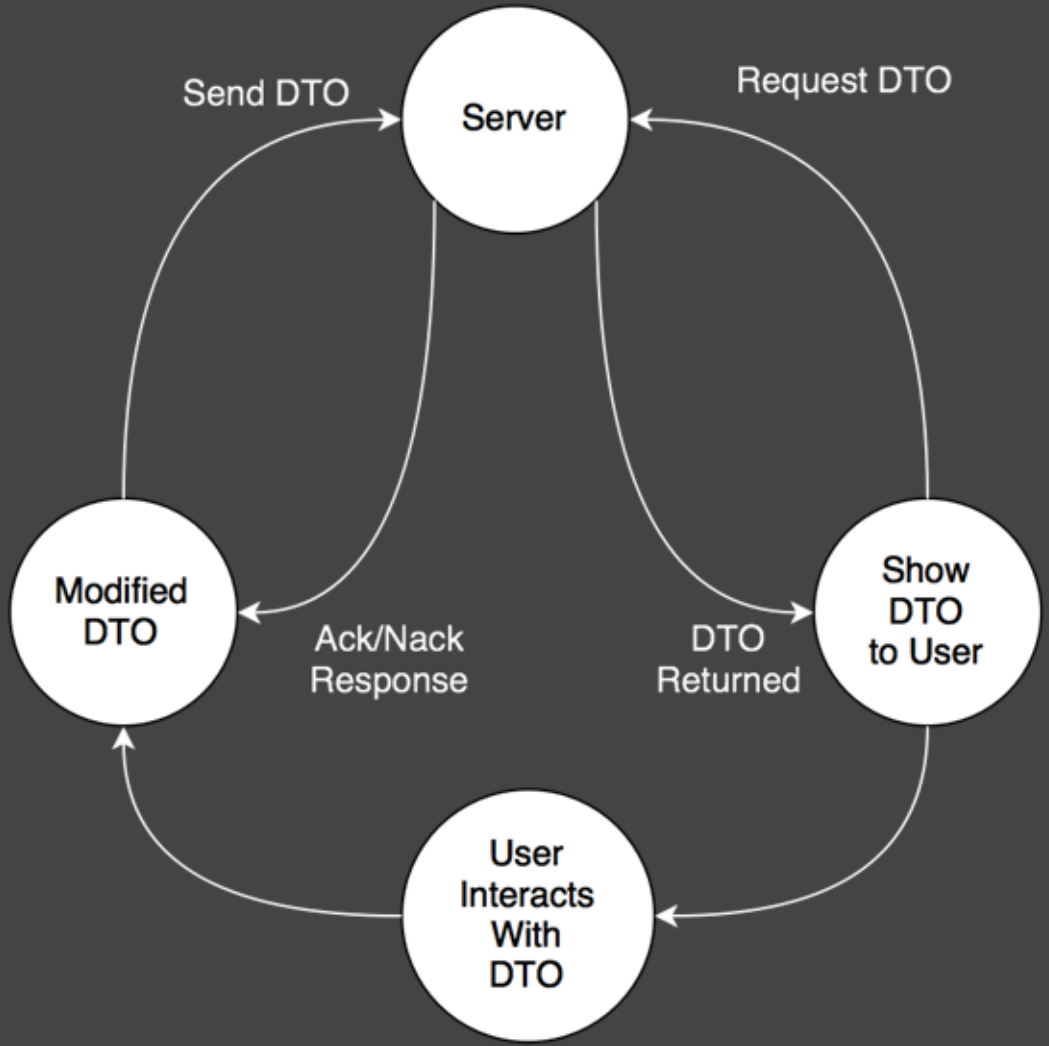
void EditCustomerDetails(CustomerDetails)

CustomerReadService

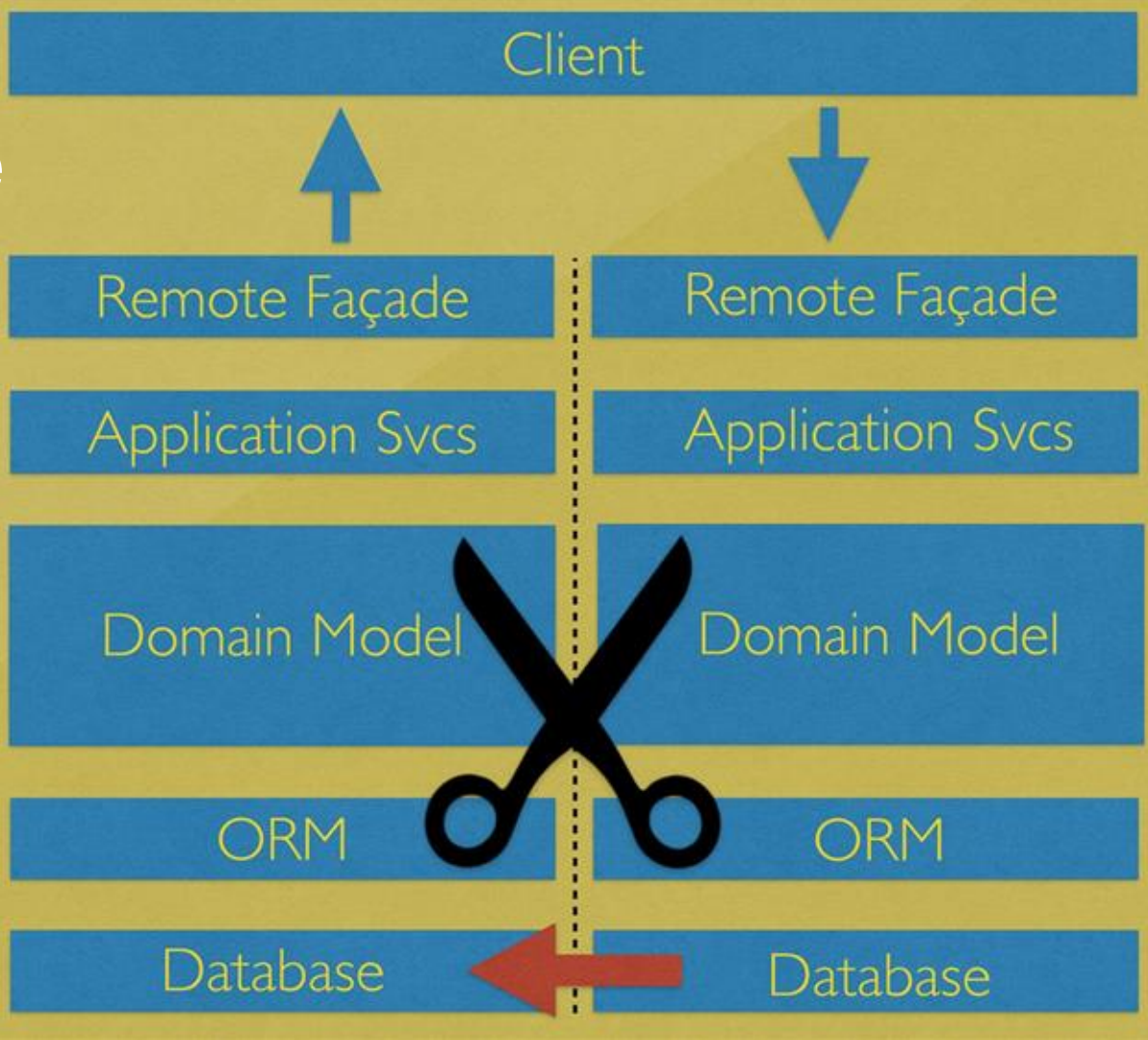
Customer GetCustomer(CustomerId)

CustomerSet GetCustomersWithName(Name)

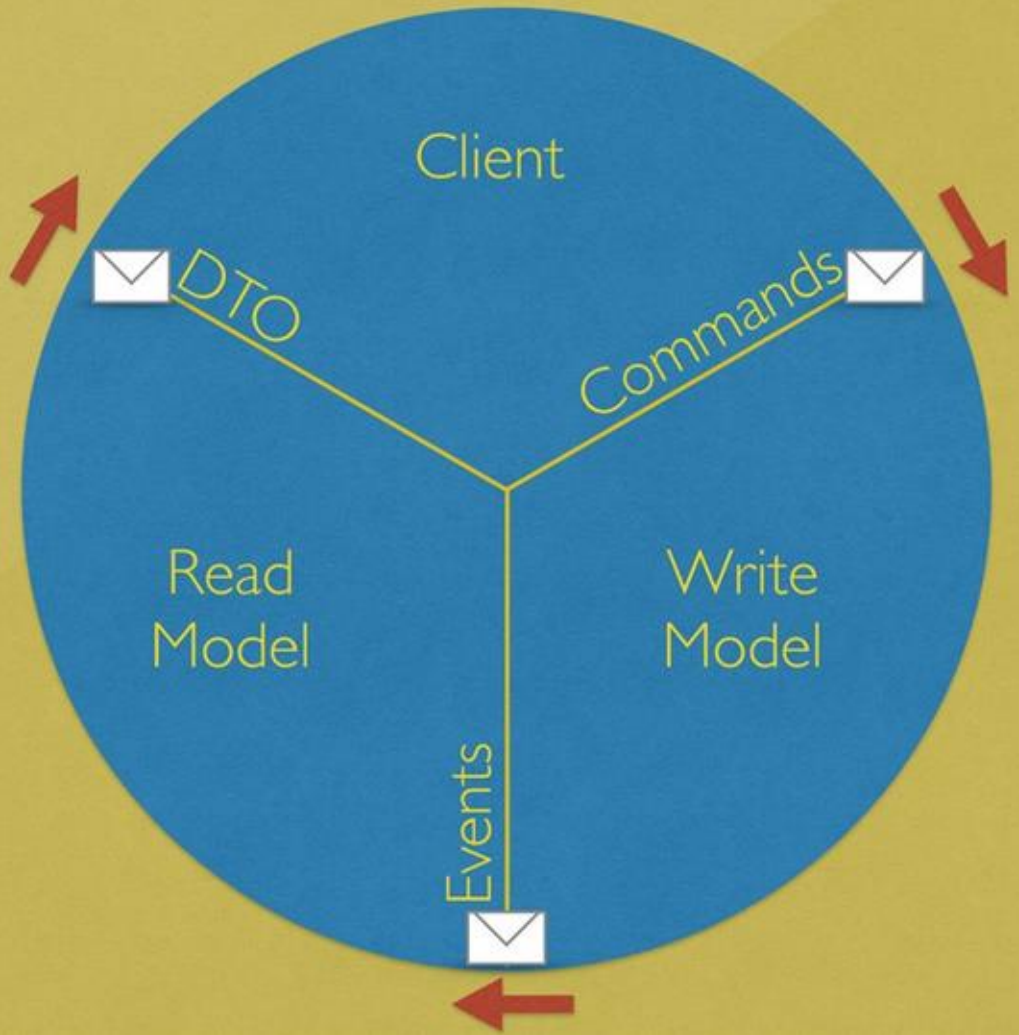
CustomerSet GetPreferredCustomers()


Server
Send DTO
Request DTO
Modified DTO
Ack/Nack Response
DTO Returned
Show DTO to User
User Interacts With DTO

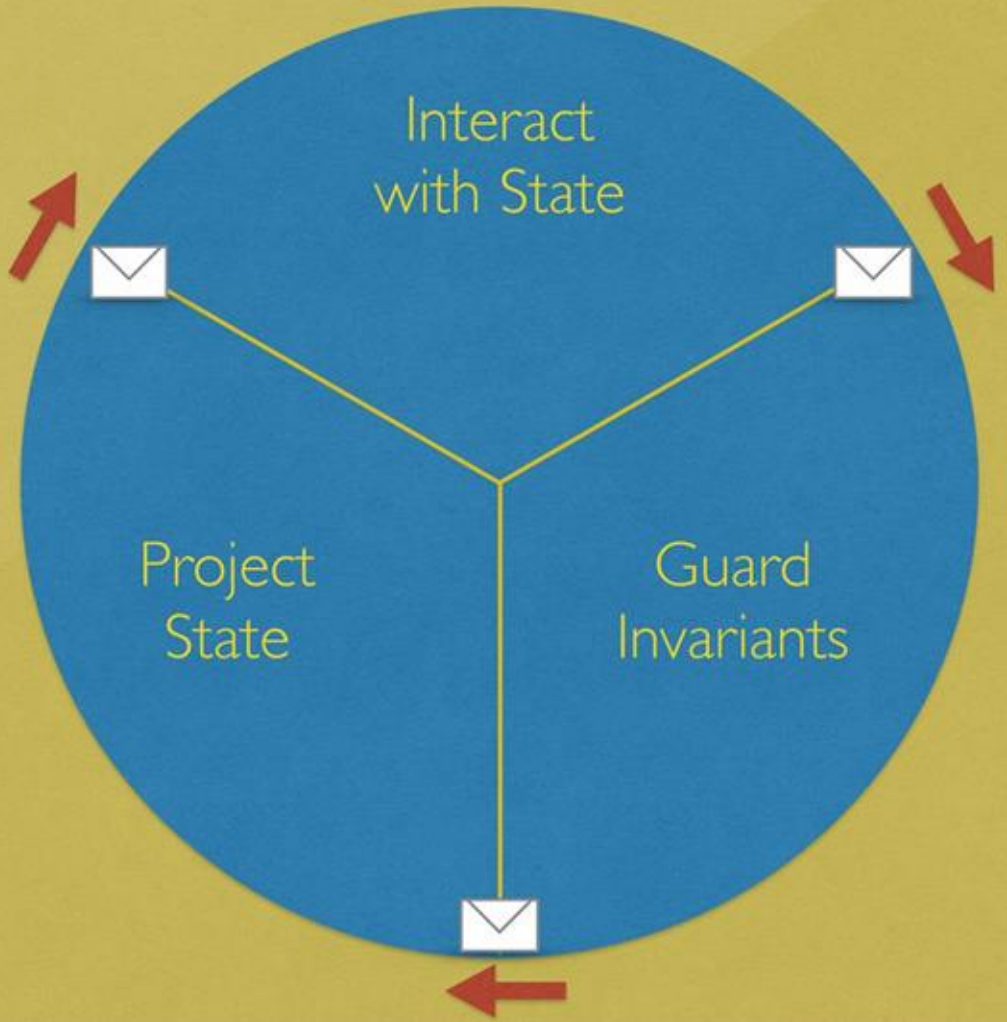
CQRS Architecture



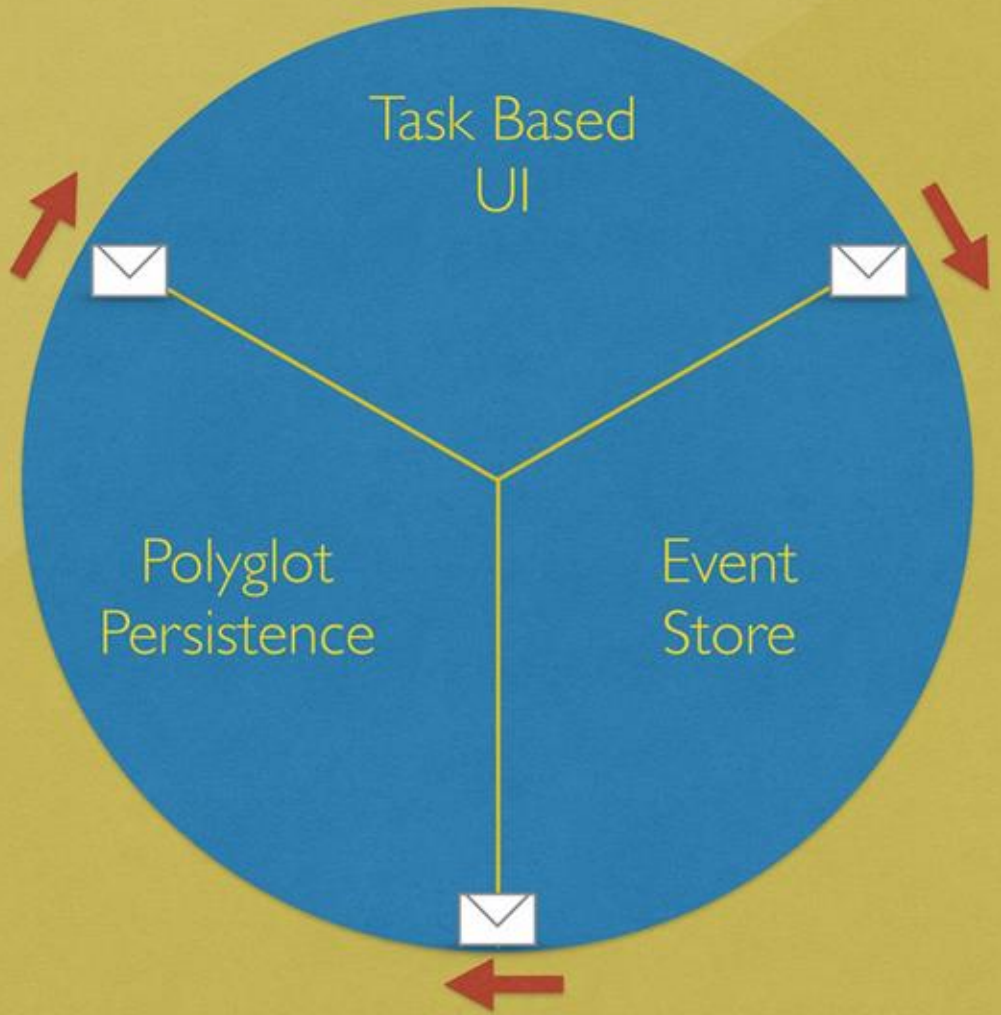
DTOs
Commands
Events

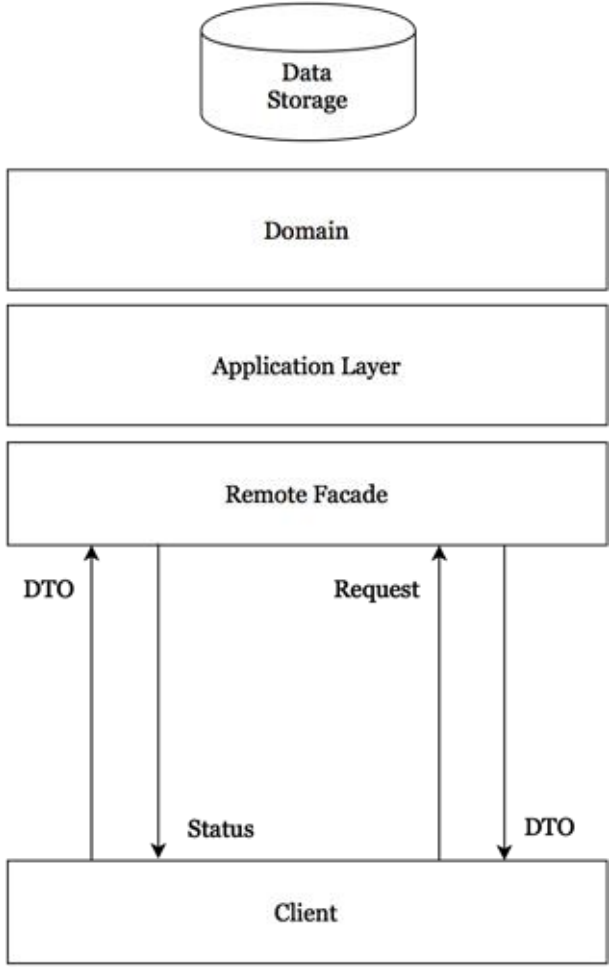


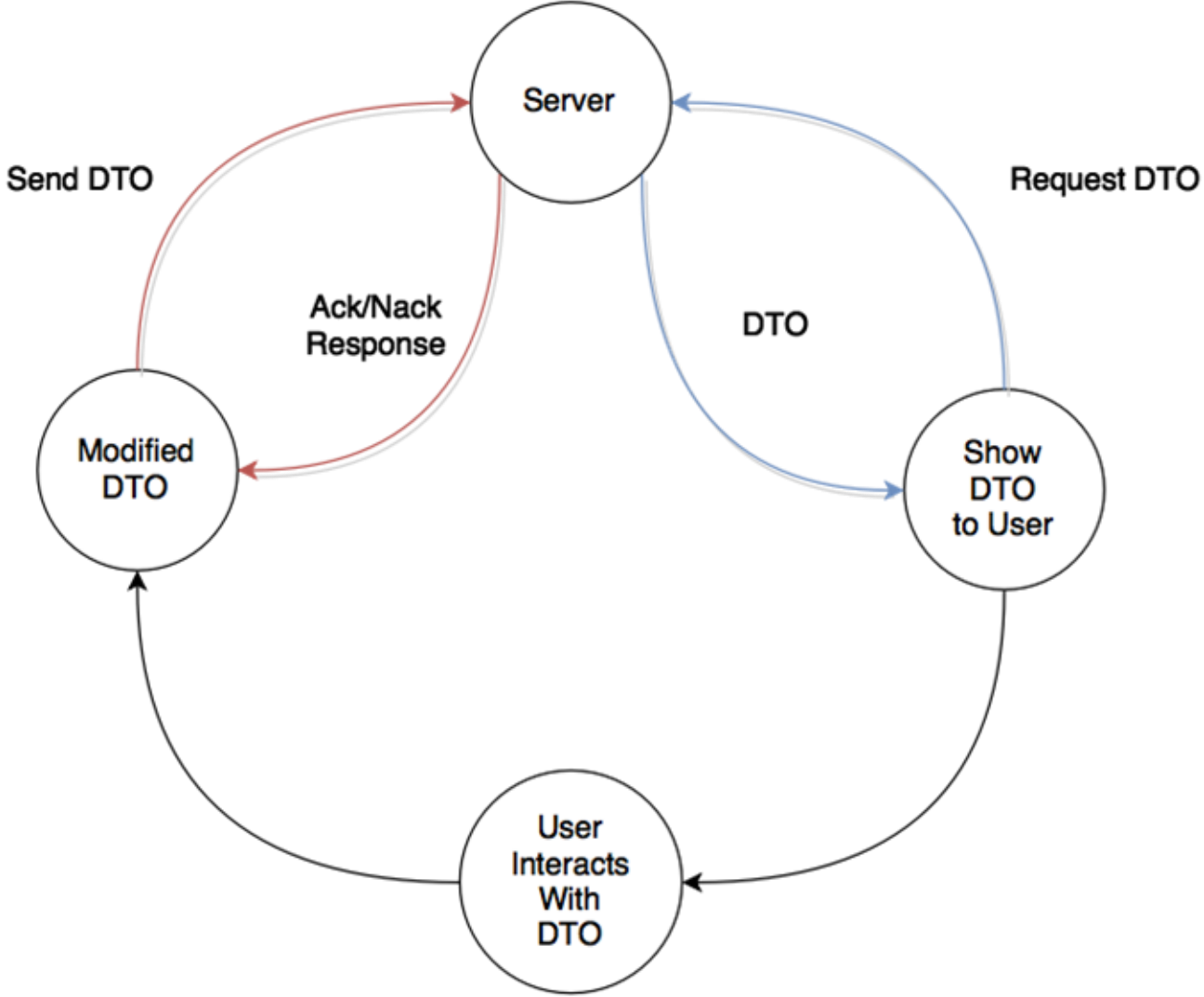
State & Invariants

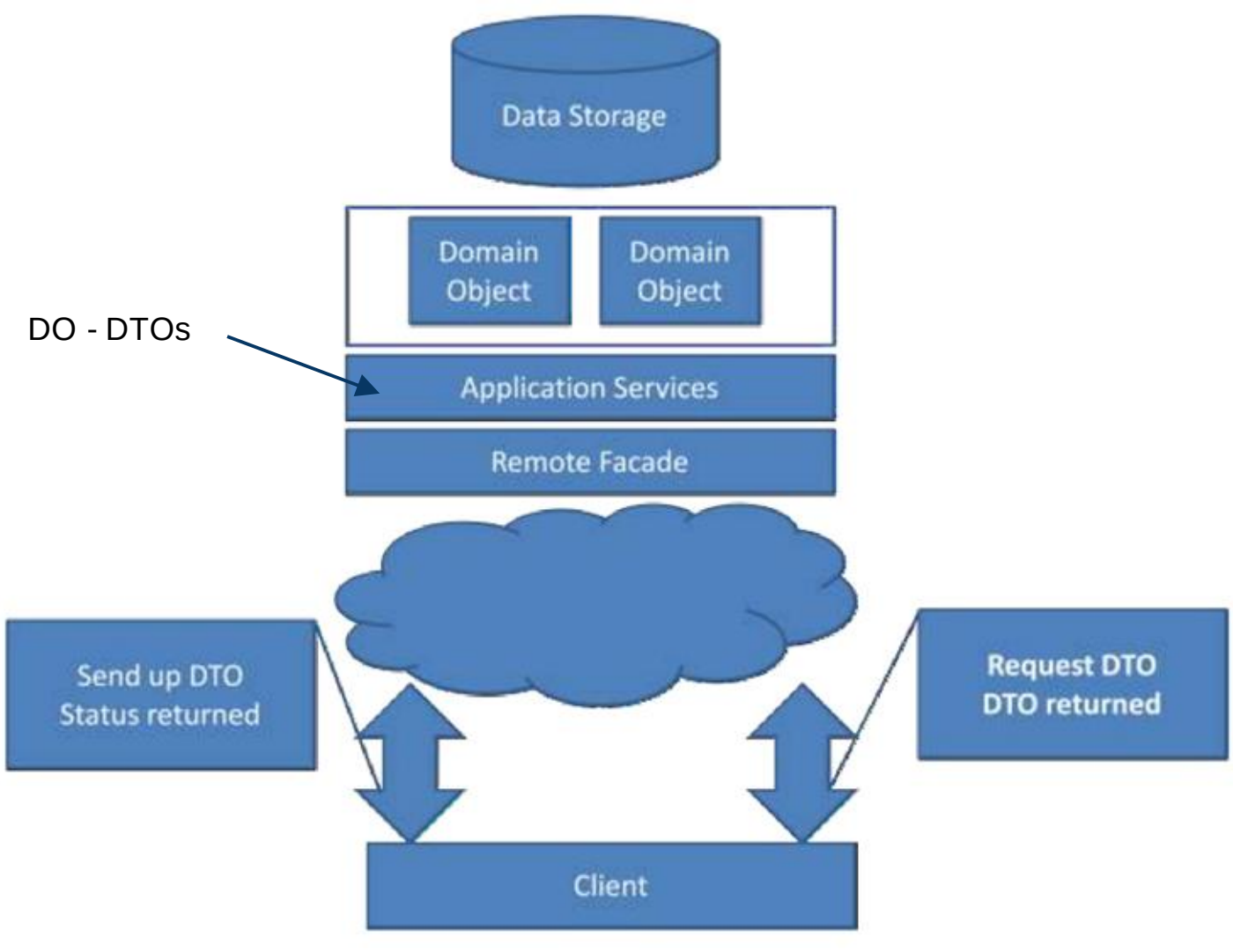


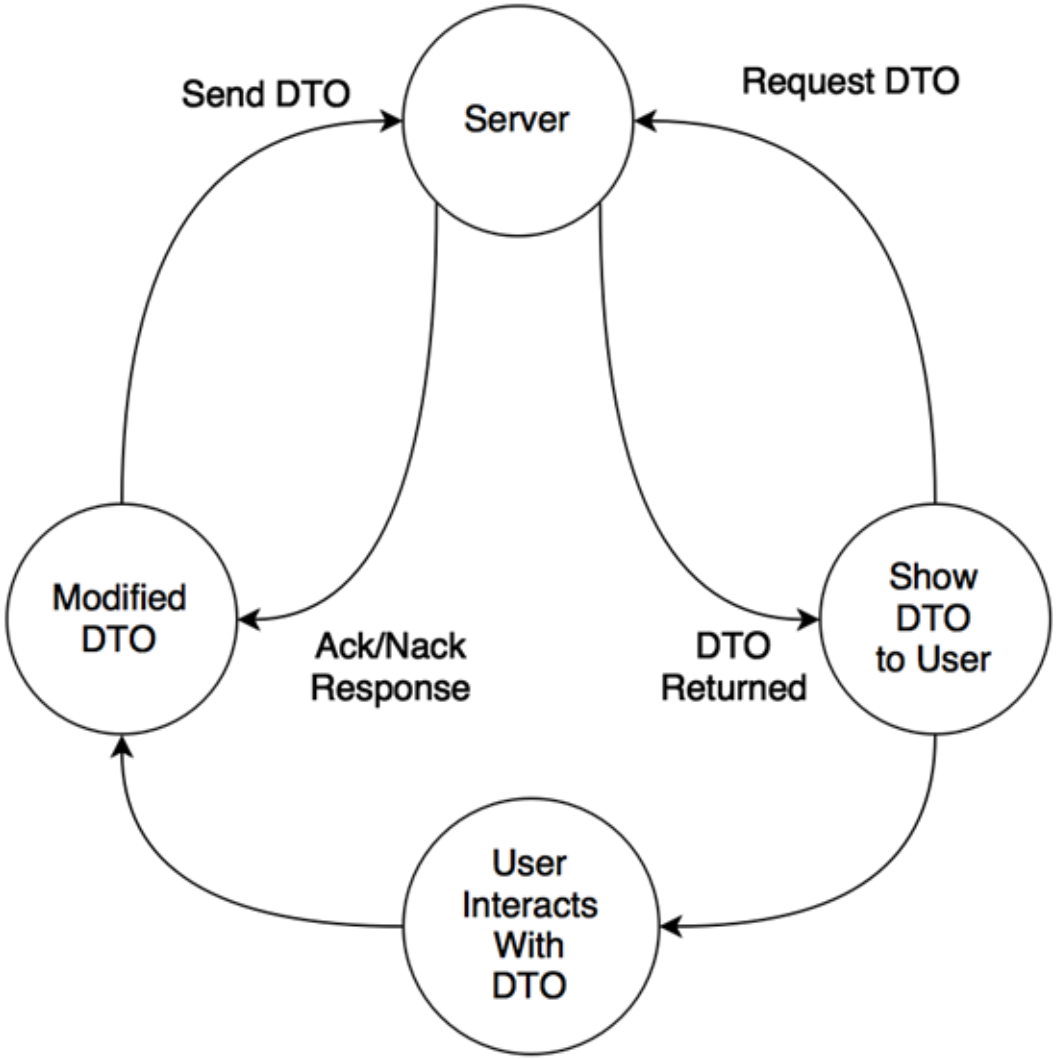
Task Based UI

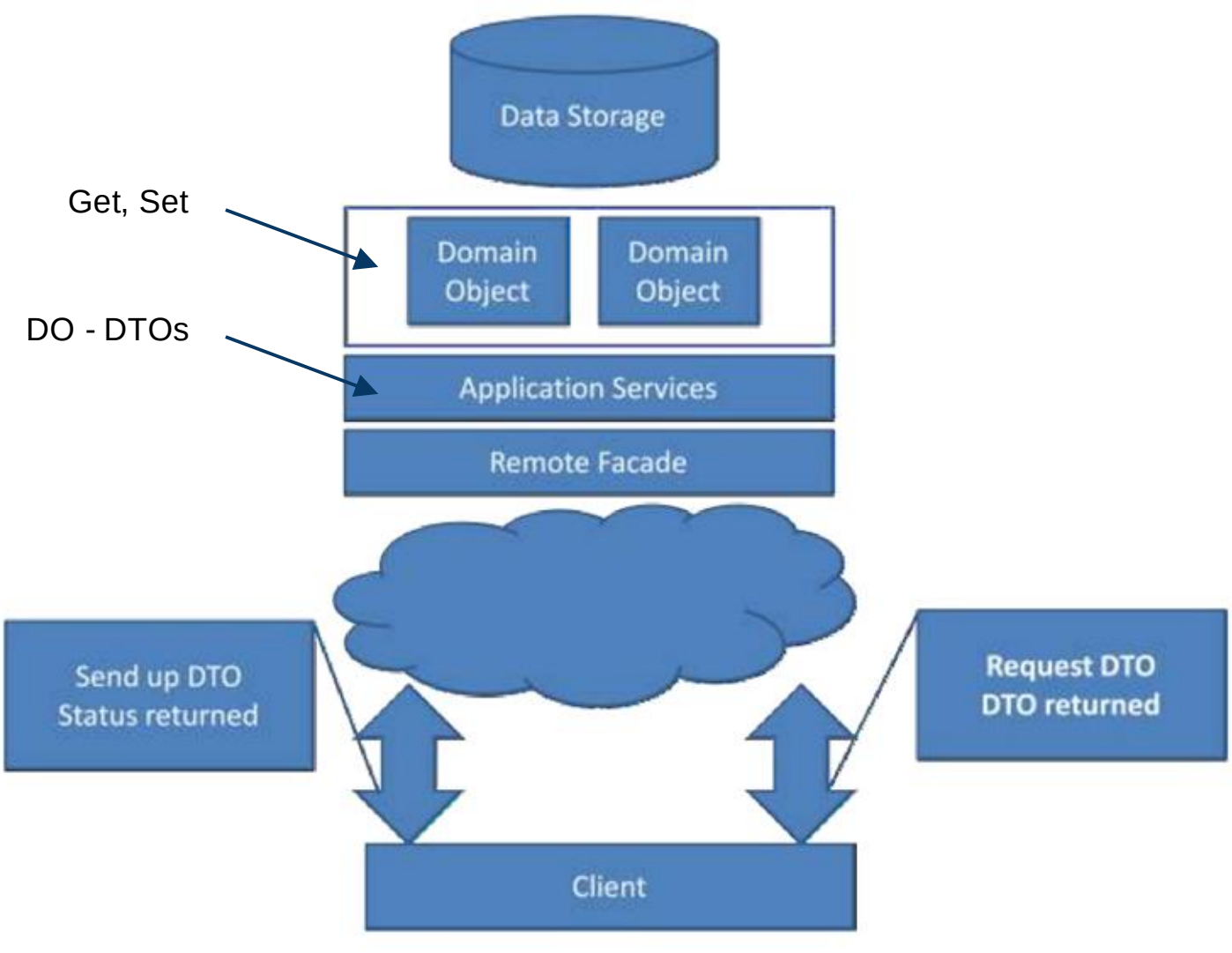


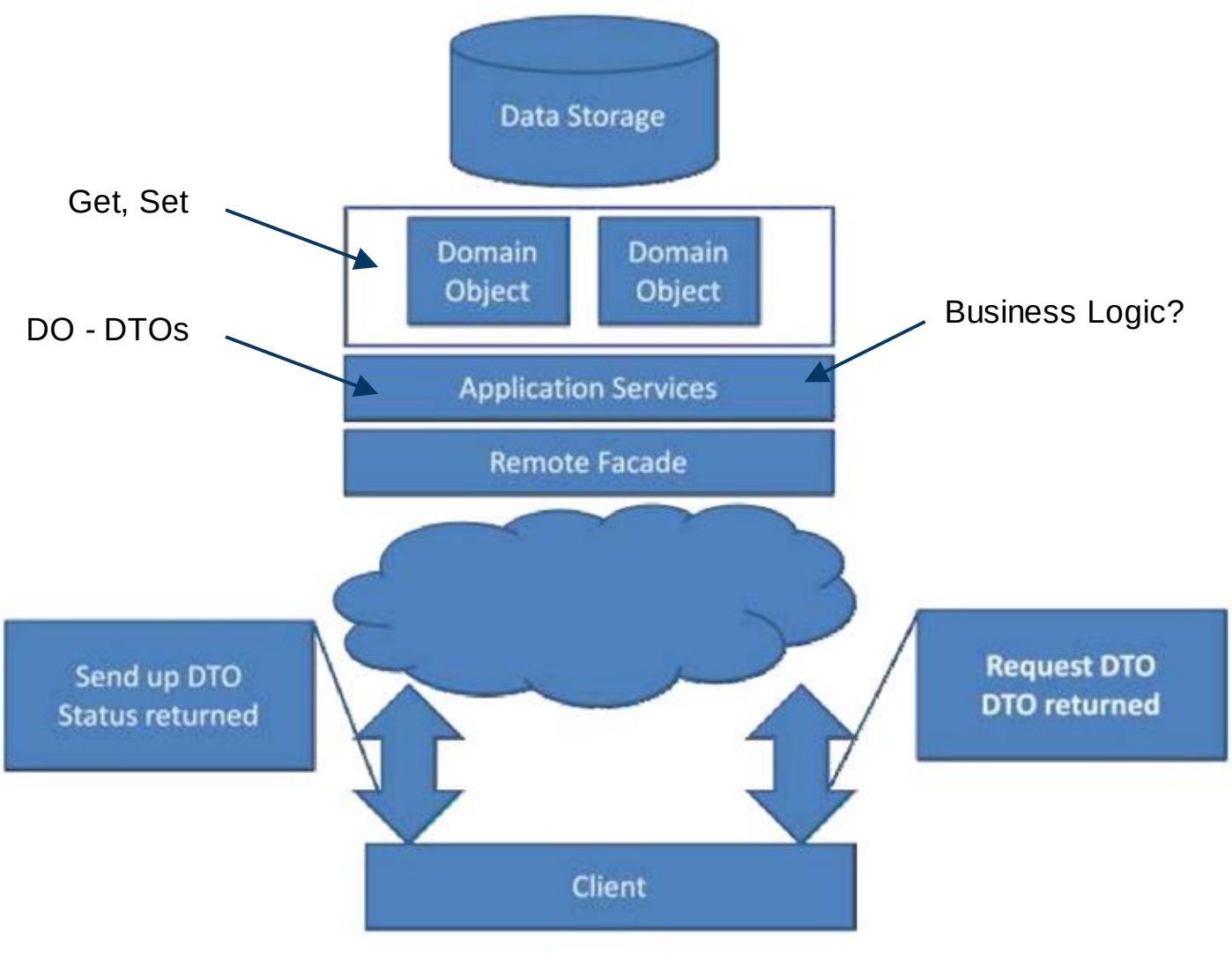

Data
Storage
Domain
Application Layer
Remote Facade
DTO
Request
Status
DTO
Client

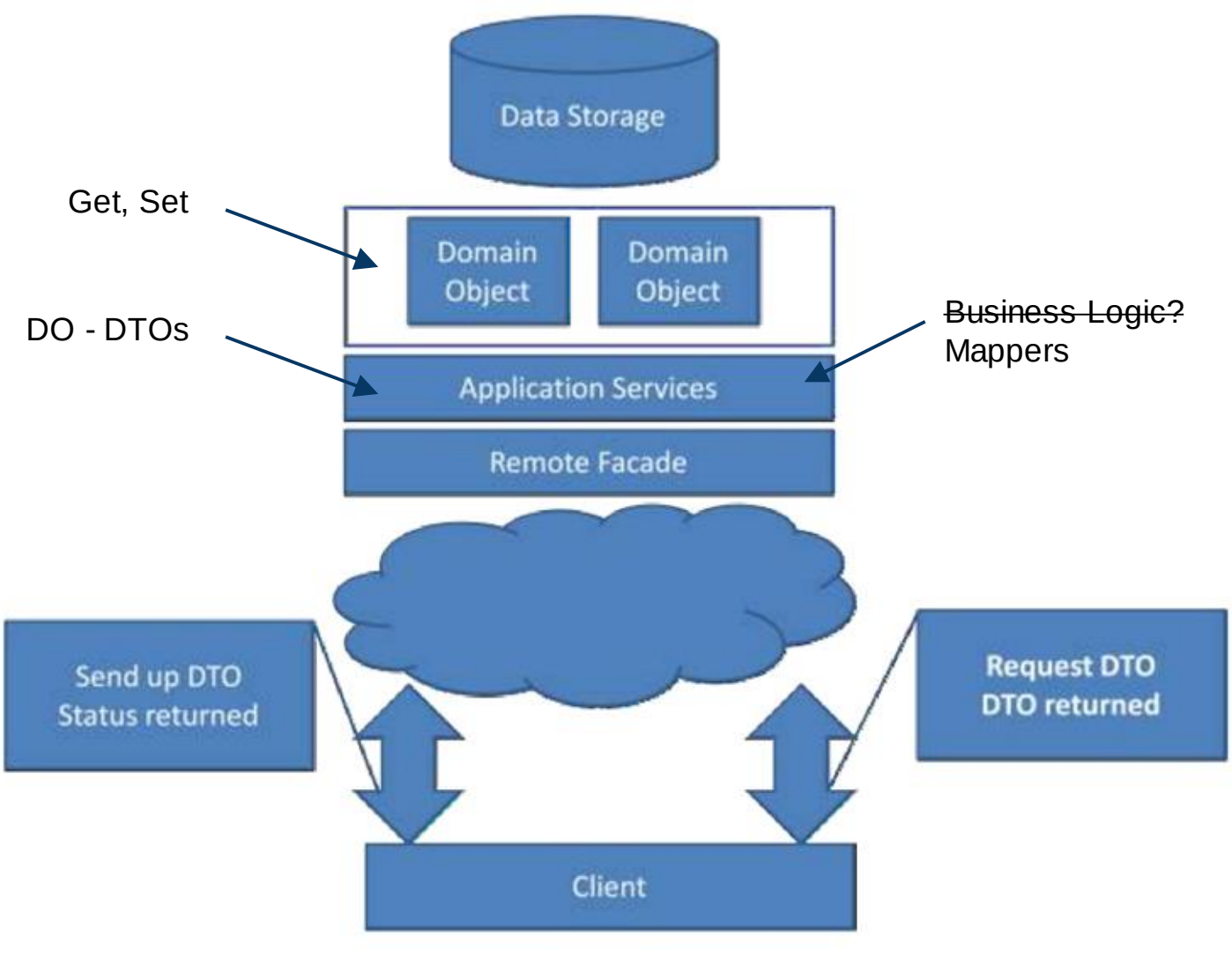

Server
Send DTO
Request DTO
Ack/Nack
Response
DTO
Modified
DTO
Show
DTO
to User
User
Interacts
With
DTO

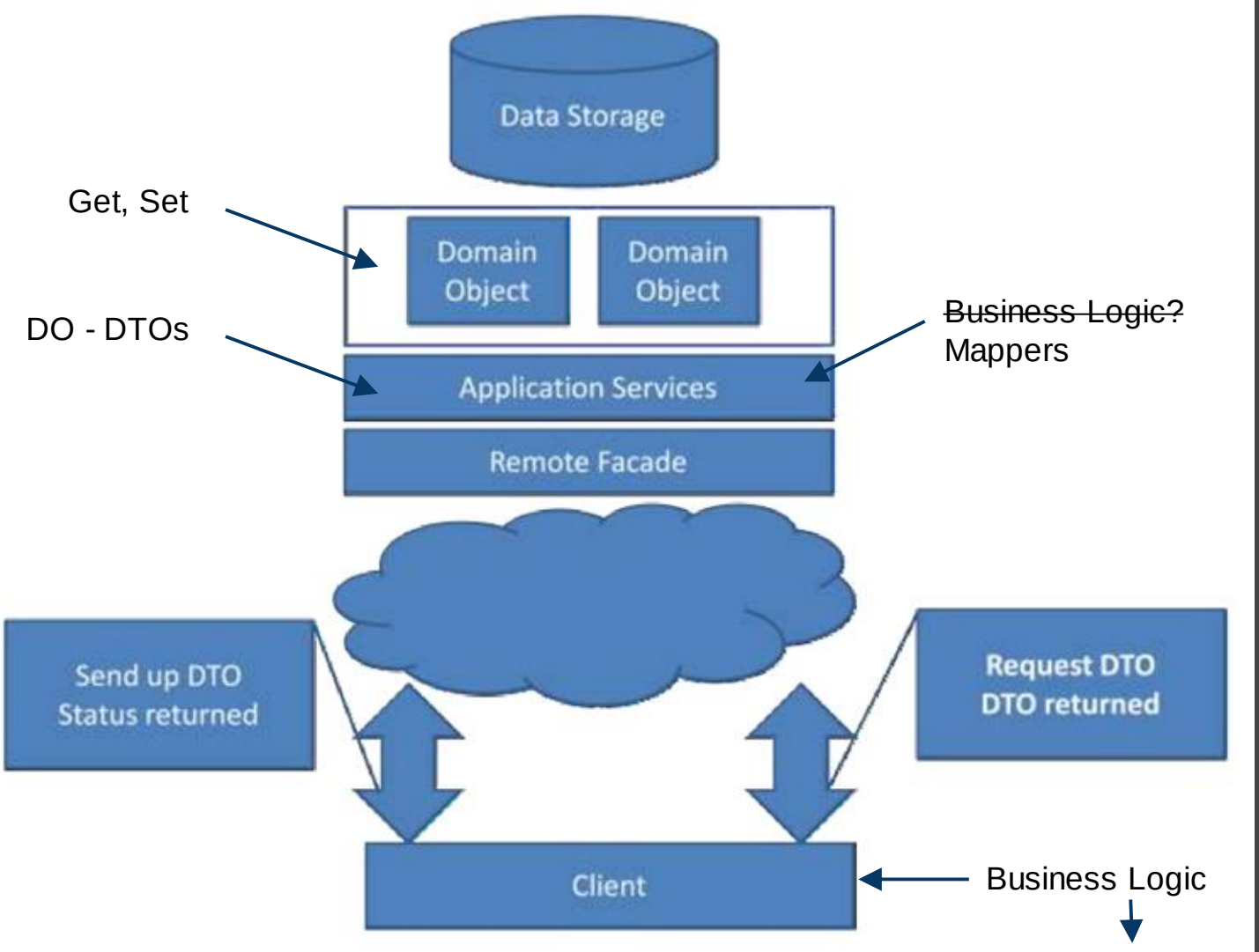

Data Storage
Domain Object
Domain Object
DO - DTOs
Application Services
Remote Facade
Send up DTO
Status returned
Request DTO
DTO returned
Client

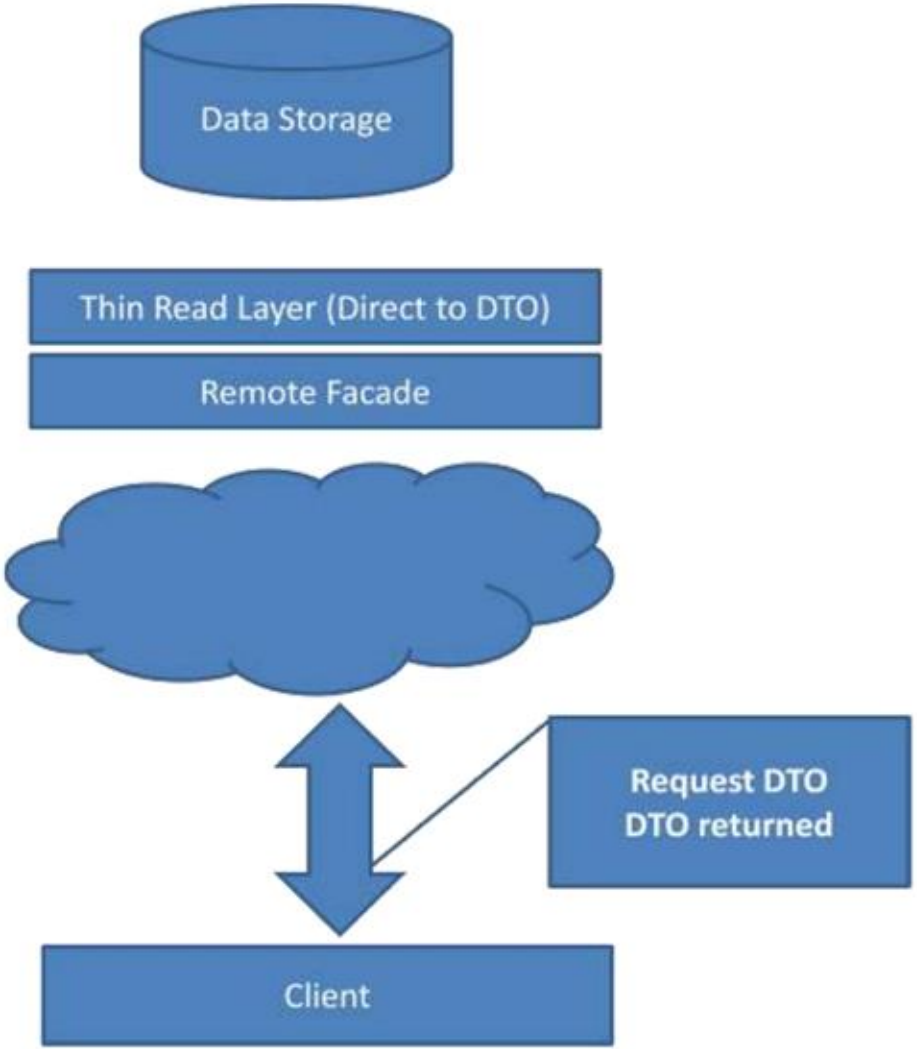

Server
Send DTO
Request DTO
Modified DTO
Ack/Nack Response
DTO Returned
Show DTO to User
User Interacts With DTO

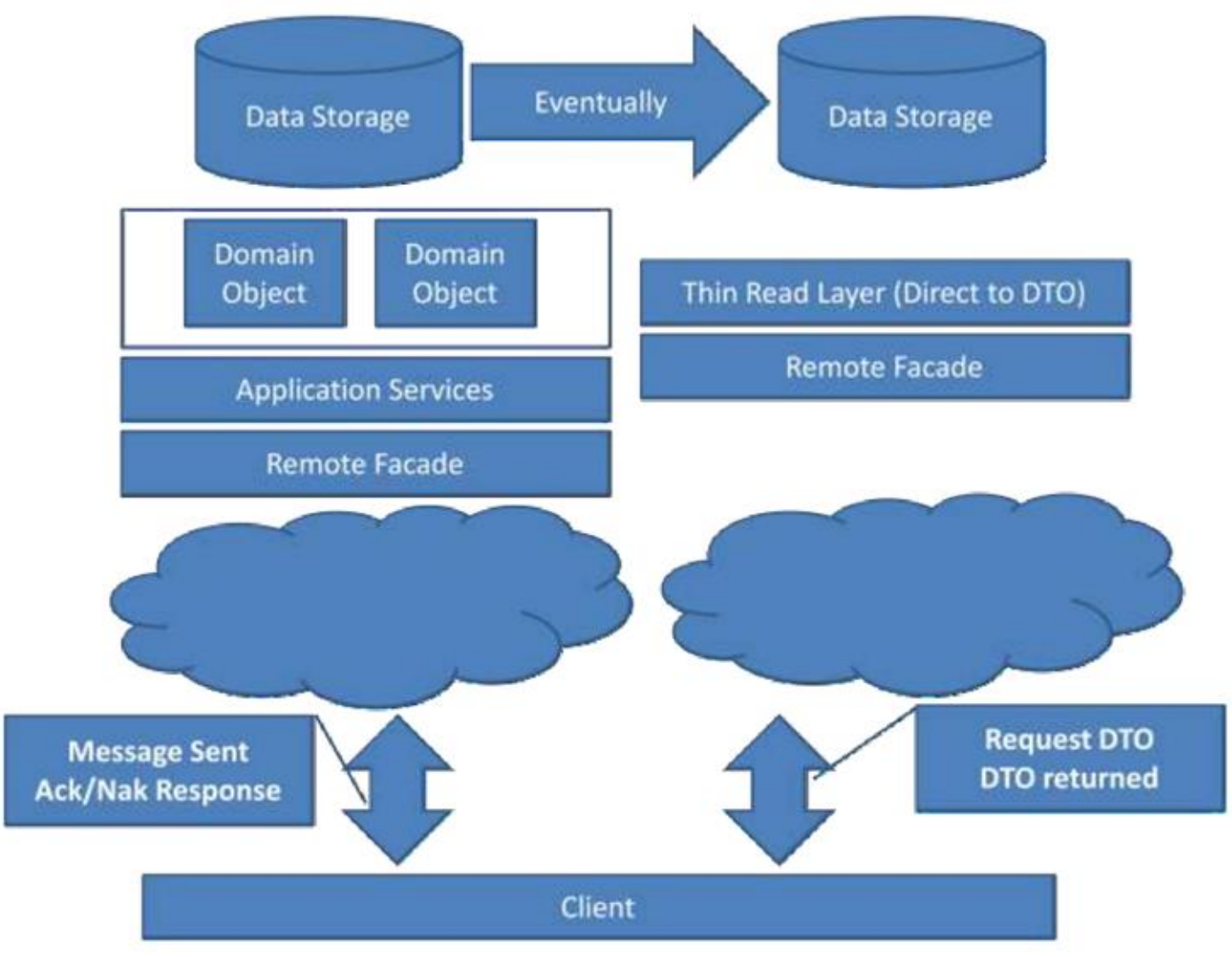

Data Storage
Get, Set
Domain Object
Domain Object
DO - DTOs
Application Services
Remote Facade
Send up DTO
Status returned
Request DTO
DTO returned
Client

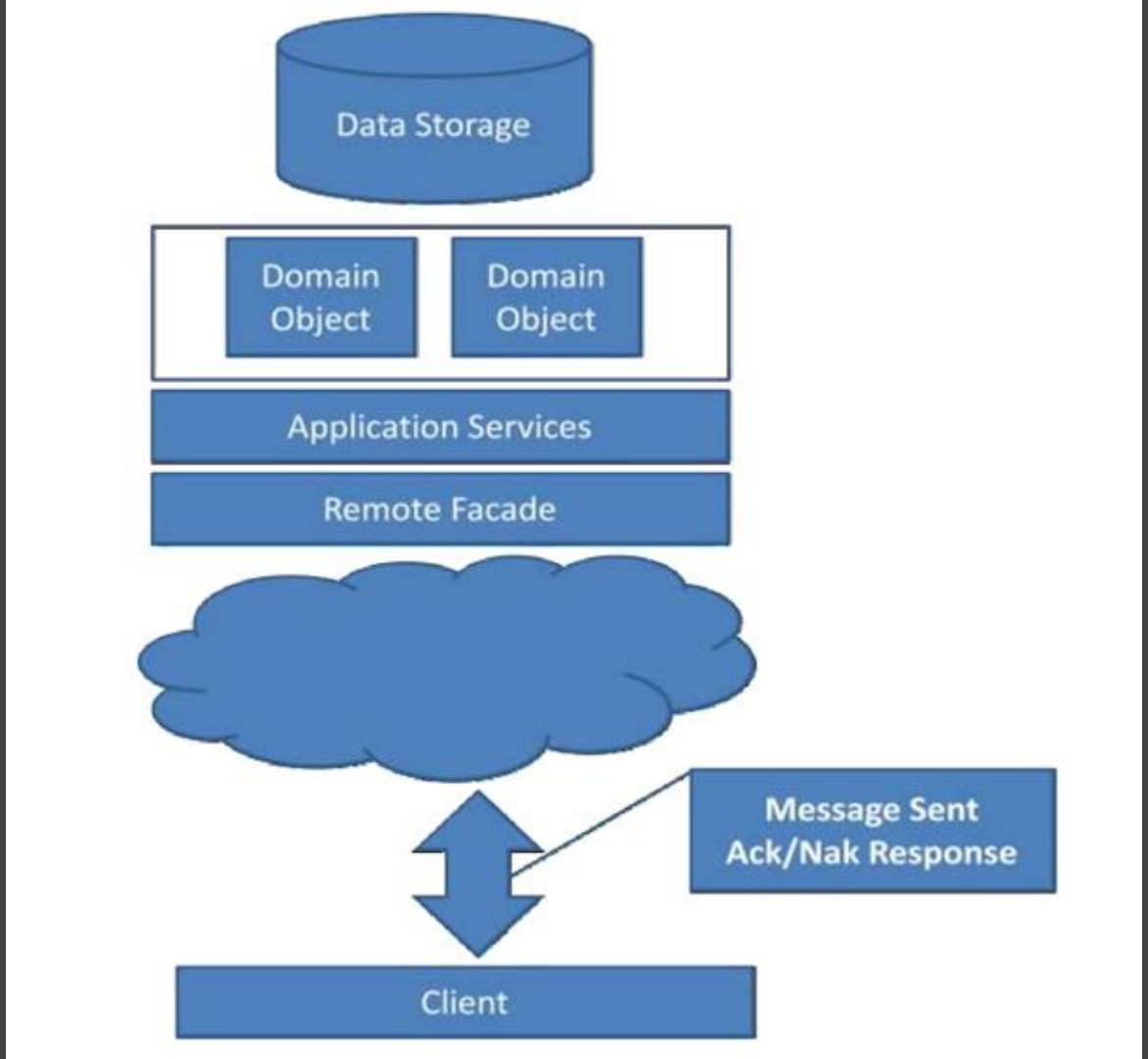

Data Storage
Get, Set
Domain Object
Domain Object
DO - DTOs
Business Logic?
Application Services
Remote Facade
Send up DTO
Status returned
Request DTO
DTO returned
Client

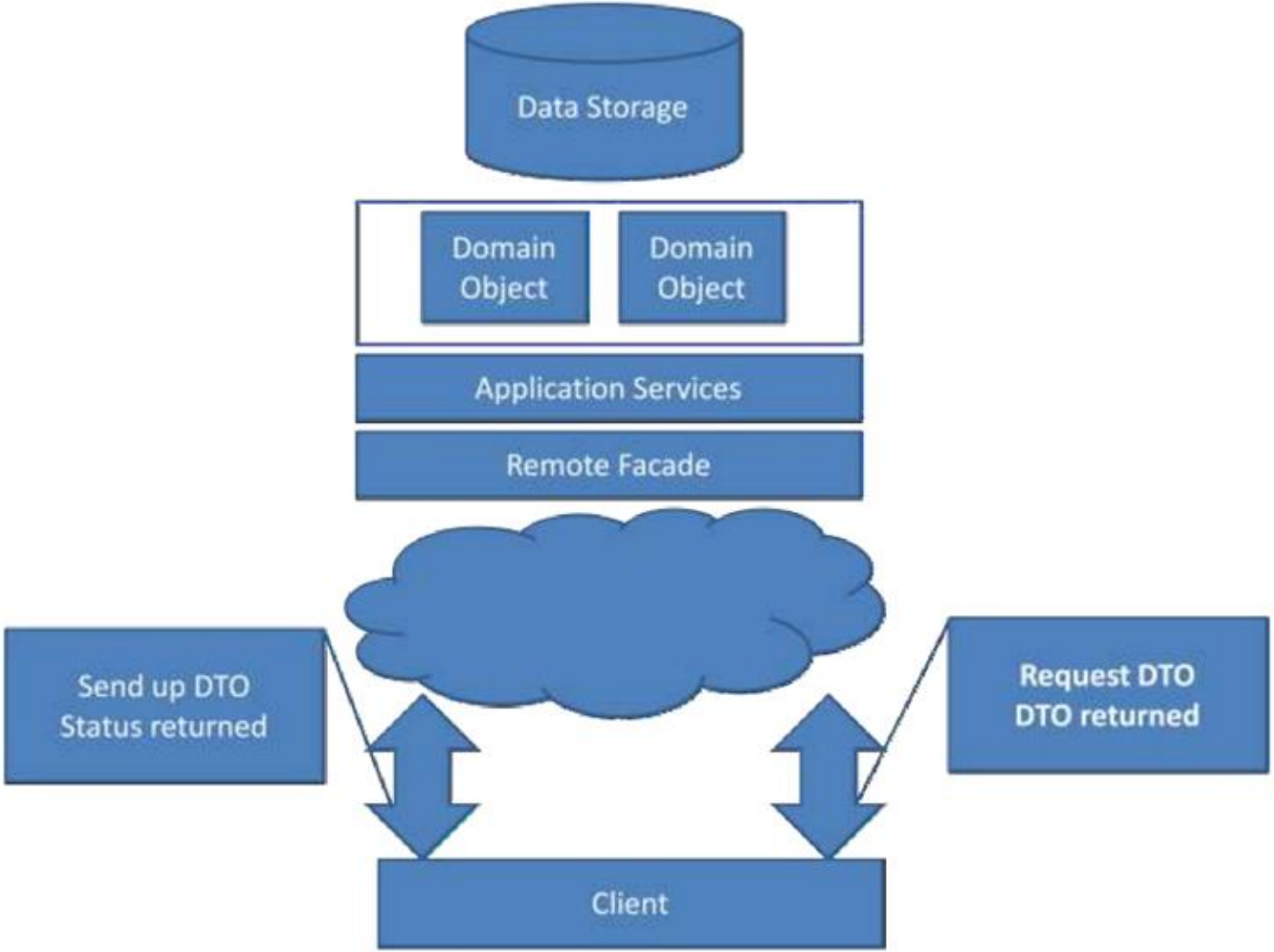

Data Storage
Get, Set
Domain Object
Domain Object
DO - DTOs
~~Business Logic?~~
Mappers
Application Services
Remote Facade
Send up DTO
Status returned
Request DTO
DTO returned
Client


Data Storage
Get, Set
Domain Object
Domain Object
DO - DTOs
~~Business Logic?~~
Mappers
Application Services
Remote Facade
Send up DTO
Status returned
Request DTO
DTO returned
Client
Business Logic


Data Storage
Thin Read Layer (Direct to DTO)
Remote Facade
Request DTO
DTO returned
Client


Data Storage
Eventually
Data Storage
Domain Object
Domain Object
Application Services
Remote Facade
Thin Read Layer (Direct to DTO)
Remote Facade
Message Sent
Ack/Nak Response
Request DTO
DTO returned
Client


Data Storage
Domain Object
Domain Object
Application Services
Remote Facade
Message Sent
Ack/Nak Response
Client


Data Storage
Domain Object
Domain Object
Application Services
Remote Facade
Send up DTO
Status returned
Request DTO
DTO returned
Client