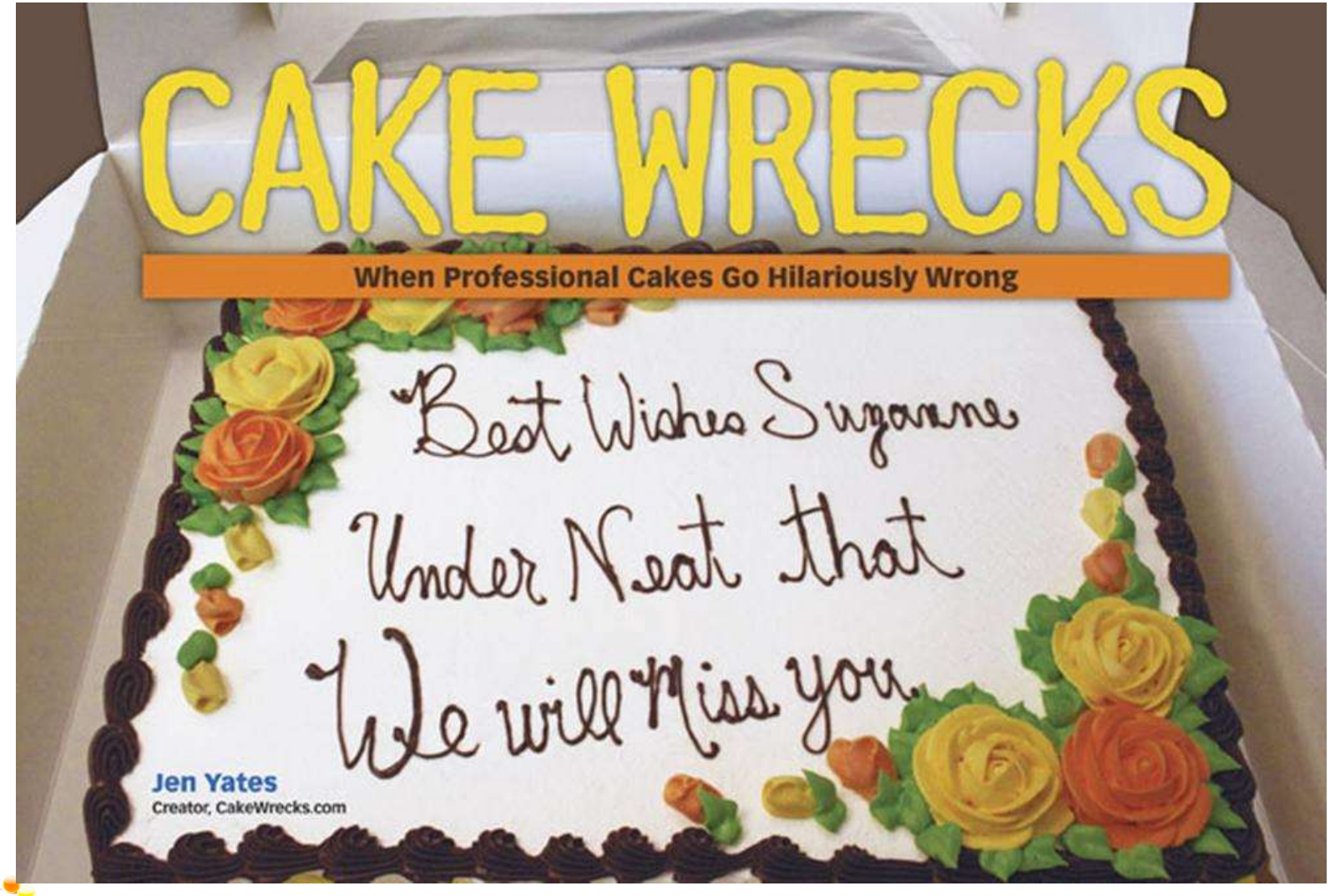
Big upfront architecture documents

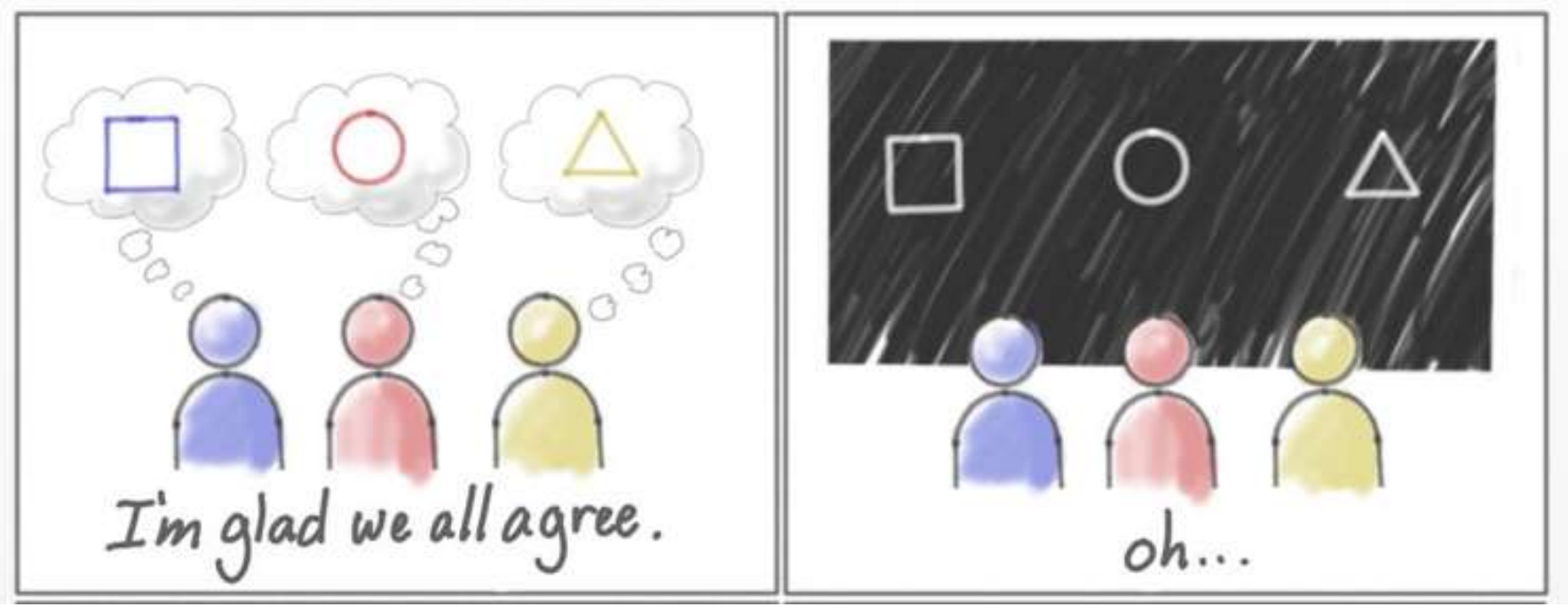
- *We can both read the same document, but have a different understanding of it.*

I am glad we all agree



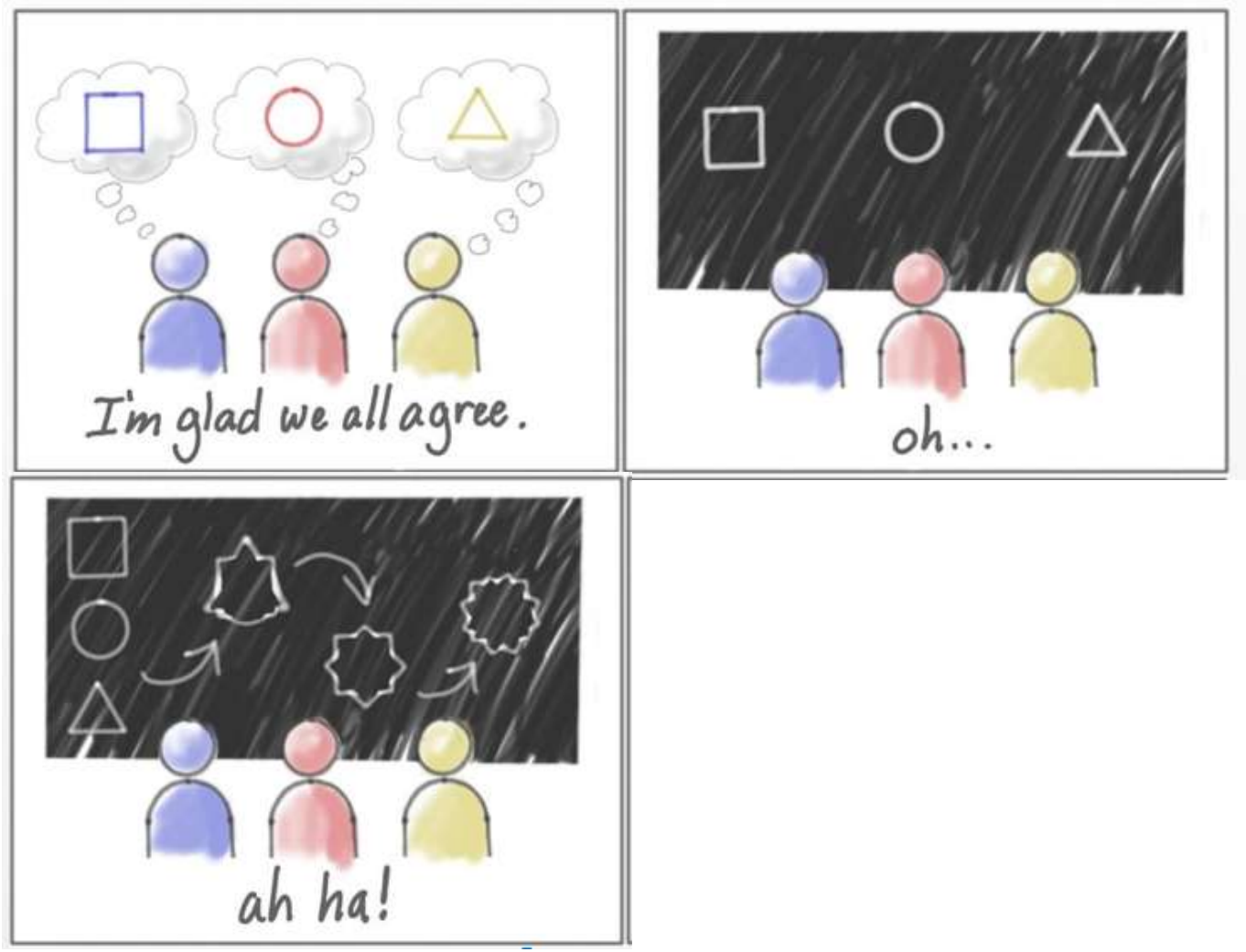
I am glad we all agree

Creating shared understanding



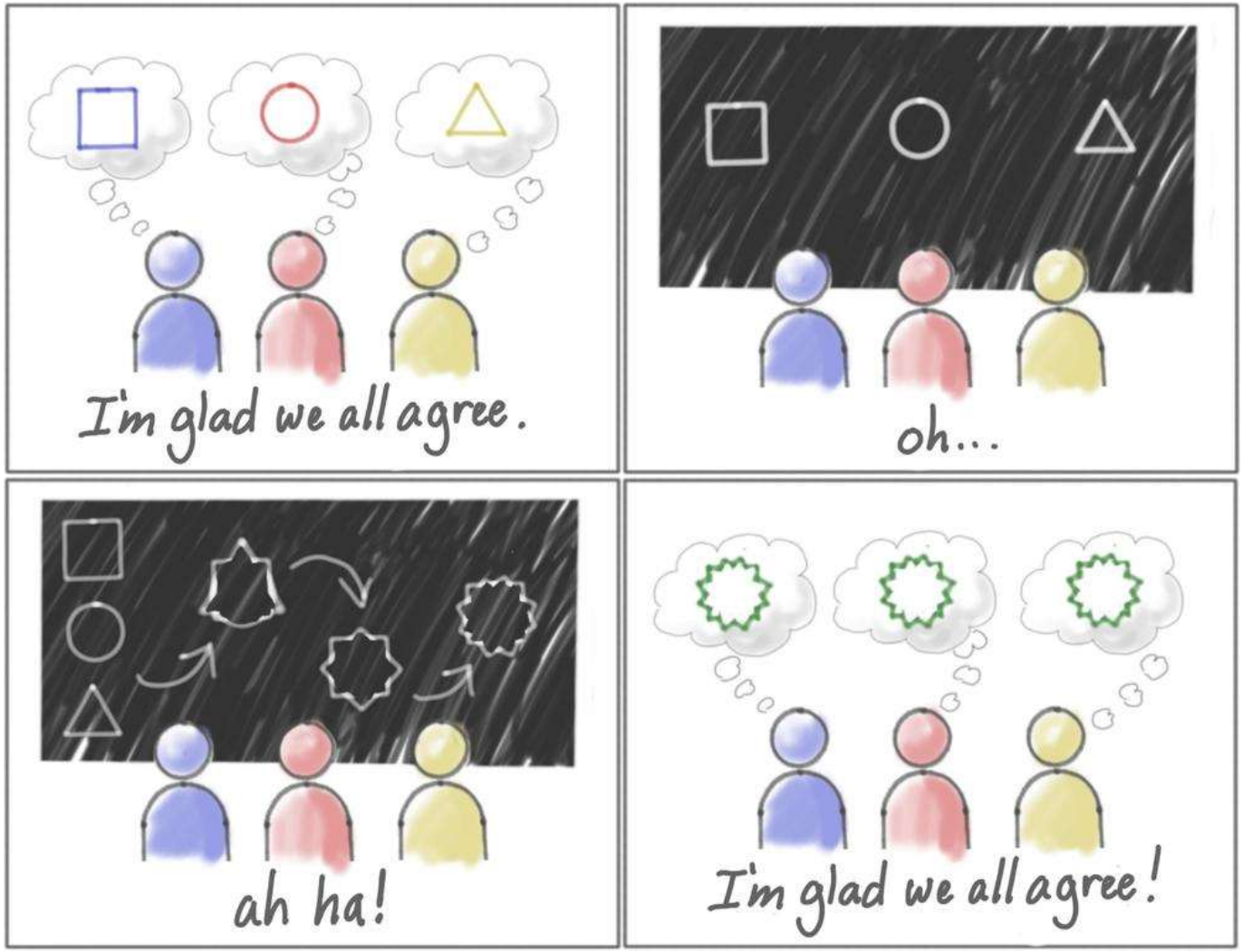
When we externalize our thinking with pictures we detect differences

Creating shared understanding



When we combine and refine, we arrive at something better

Creating shared understanding



Afterwards, when we say the same thing, we actually mean it

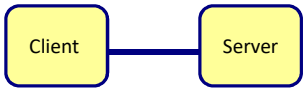
Lesson 2

APPLICATION ARCHITECTURE

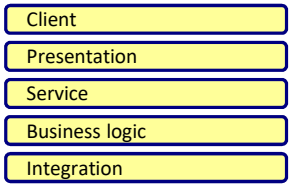
Application Architecture

ARCHITECTURAL STYLES

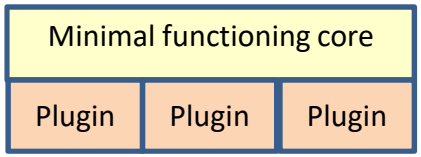
Architecture styles



Client-server

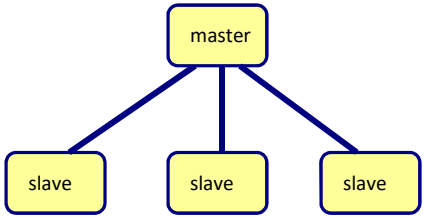


Layering



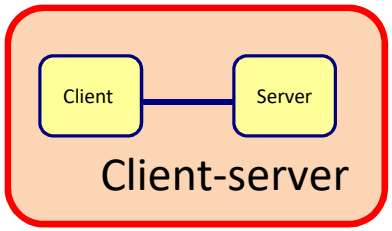
Microkernel

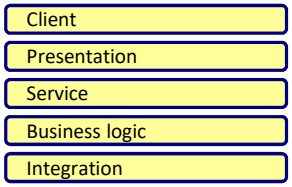
Pipe-and-Filter



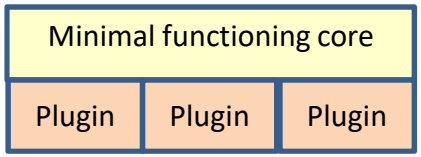
Master-Slave

Client-server



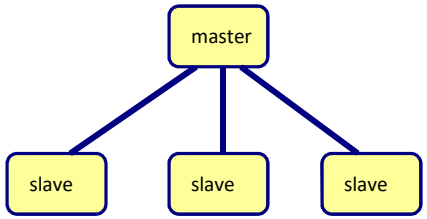


Layering



Microkernel

Pipe-and-Filter

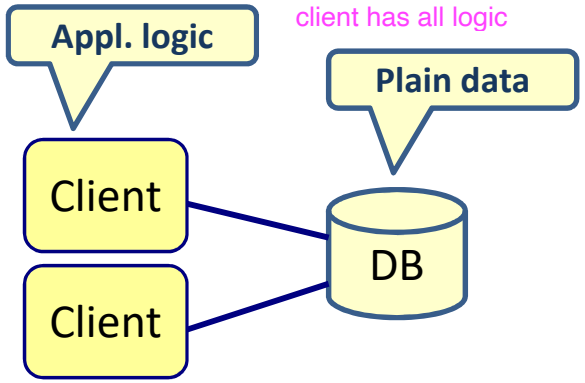


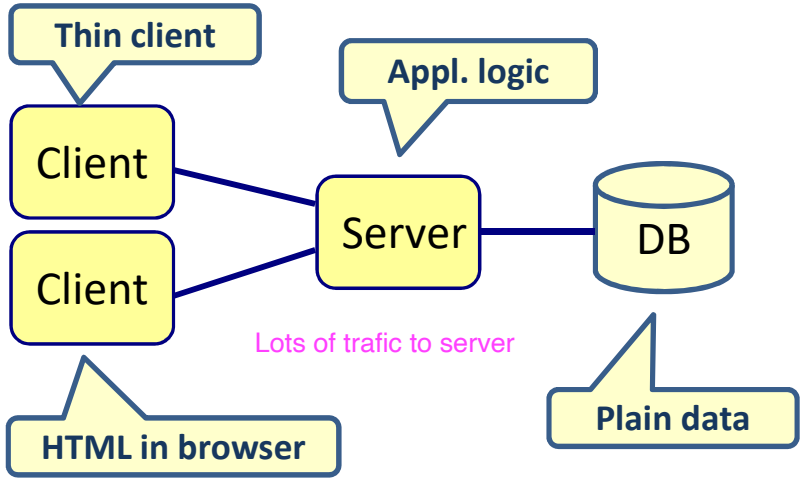
Master-Slave

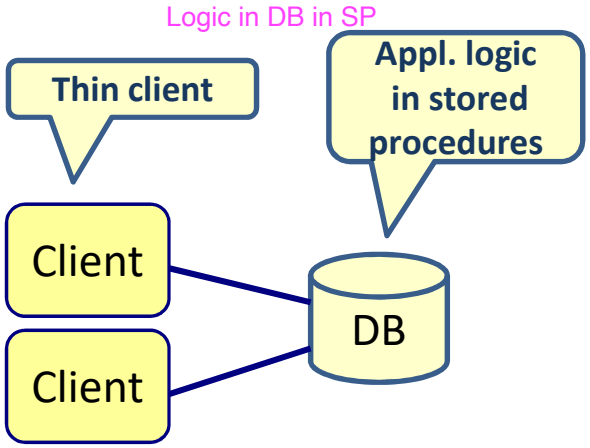
Client-Server

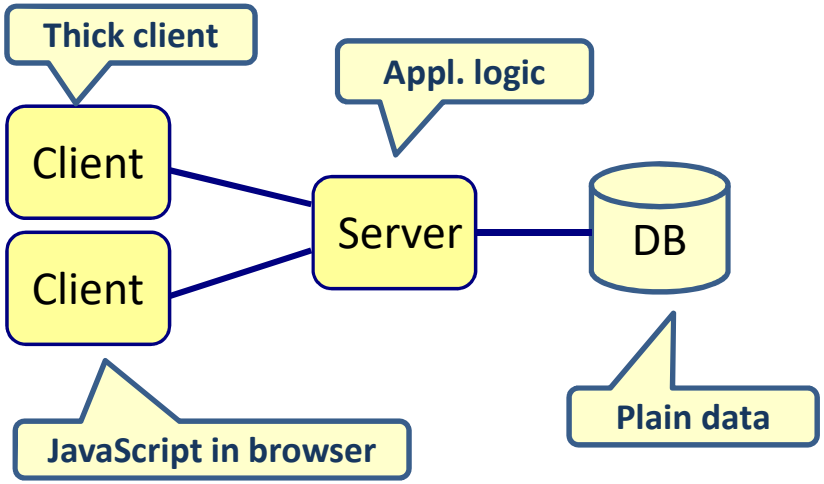
- Multiple clients per server
- Thin-client/Thick client Where to put the logic
- Stateless / stateful server Where to put the State
- Multiple tiers layer is logical and tier is phisical division
- Requests typically handled in separate threads

Client-server architectures







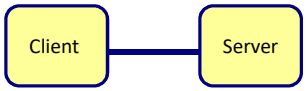


Client-server

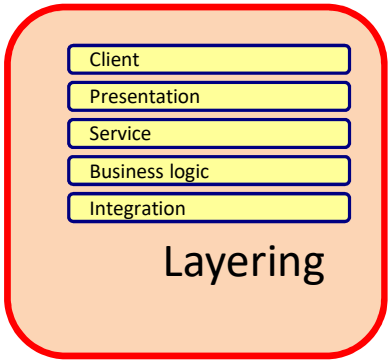
- Benefits
  - Easy maintenance
    - Application logic in one place (server)
  - Supports many different clients
- Drawbacks
  - Performance can become an issue

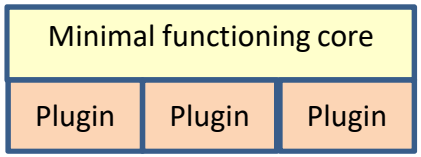
Due to distributed systems

Layering



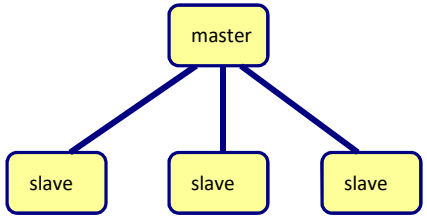
Client-server





Microkernel

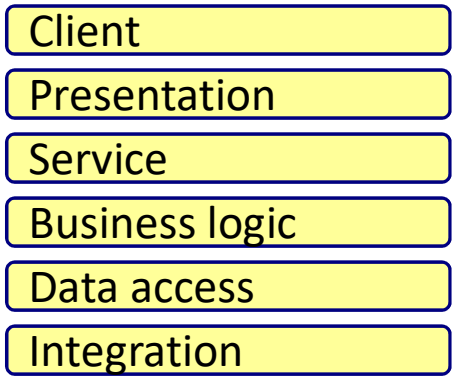
Pipe-and-Filter



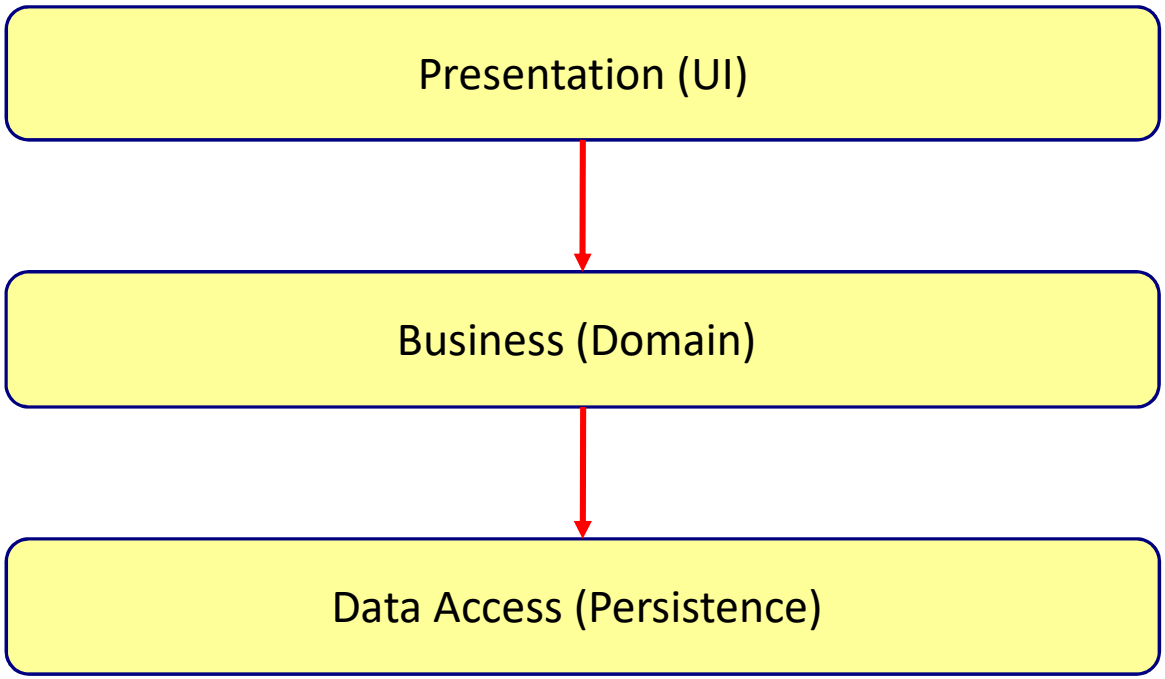
Master-Slave

Layering

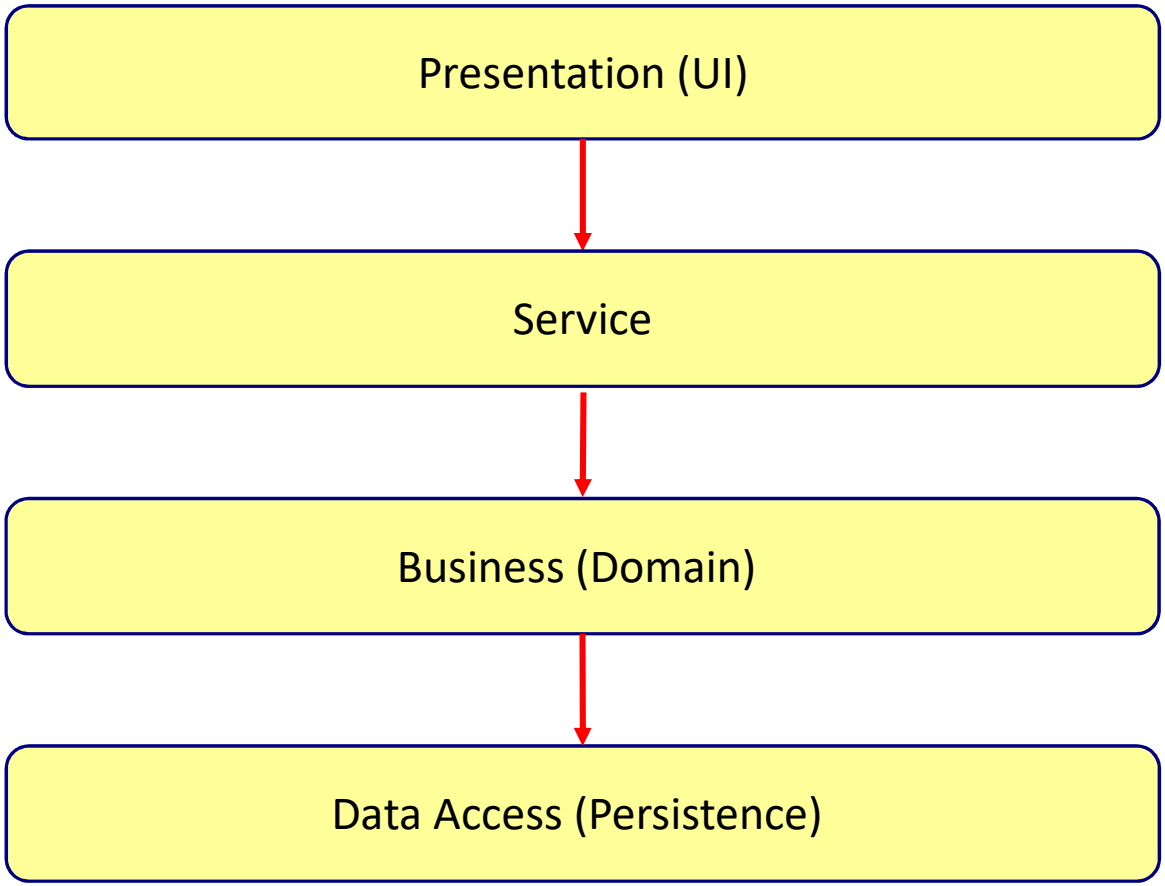
- Separation of concern seperate thing and make them independent
- Layers are independent
- Layers can be distributed
- Layers use different techniques



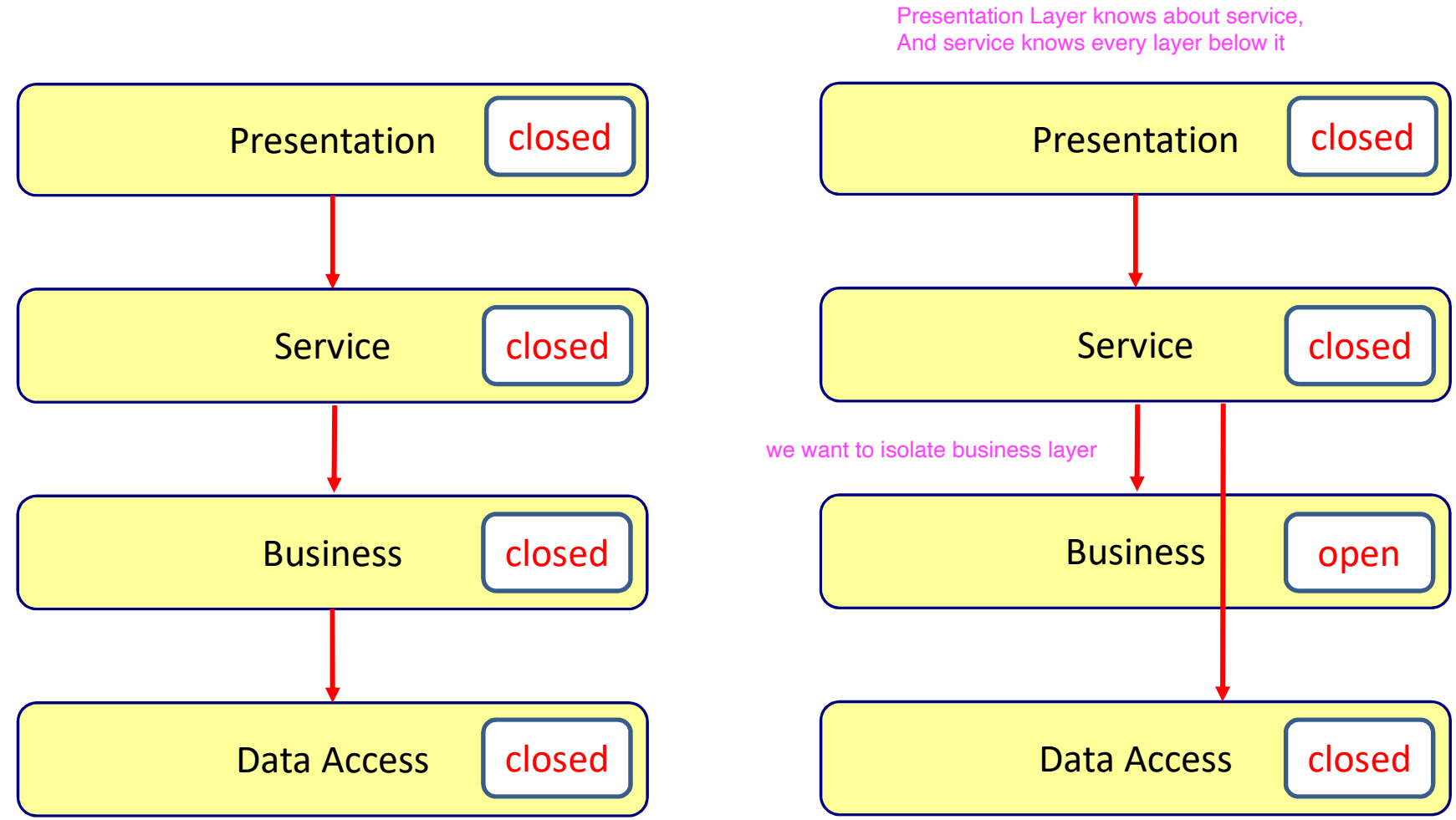
3 layered architecture



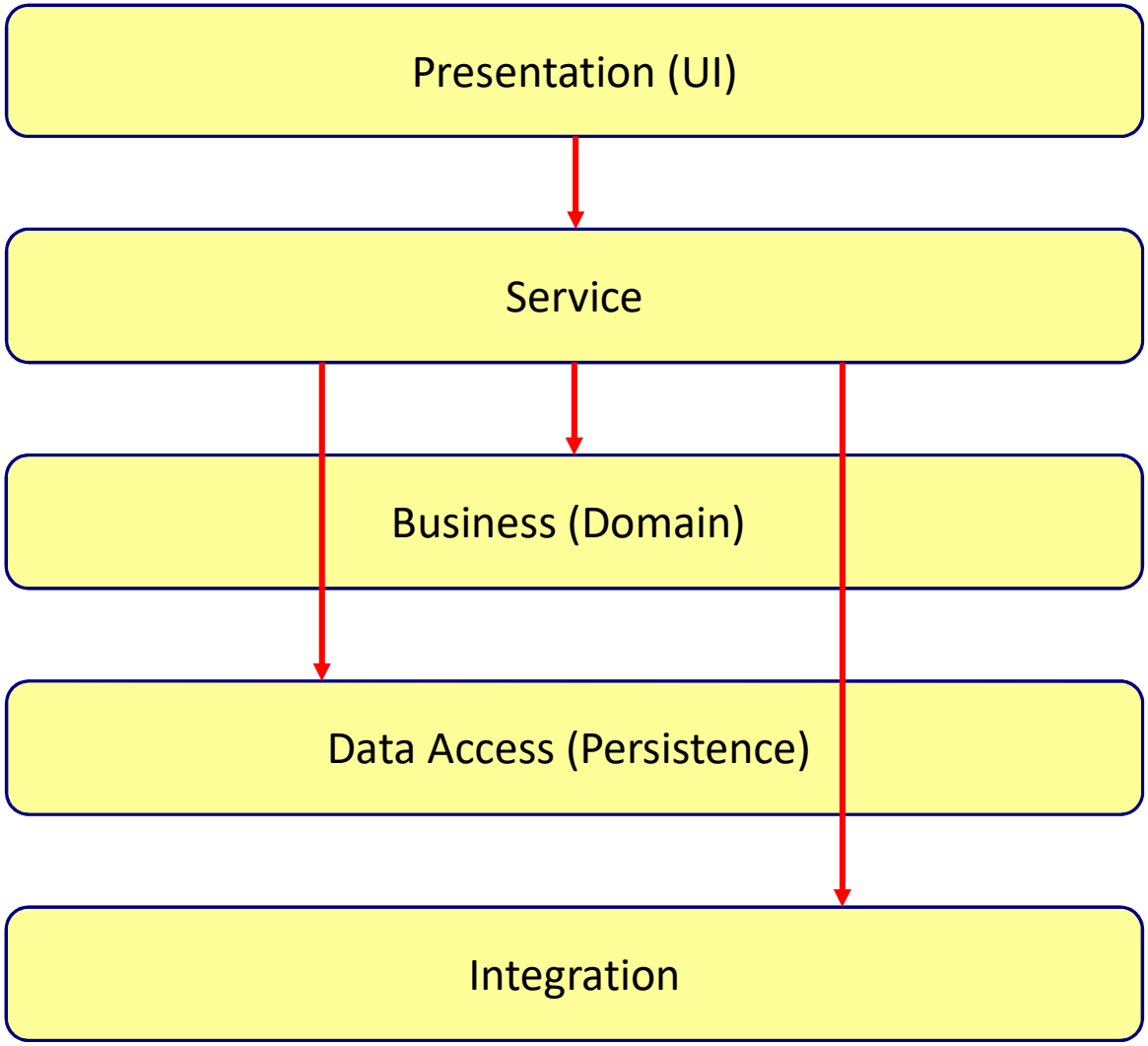
4 layered architecture



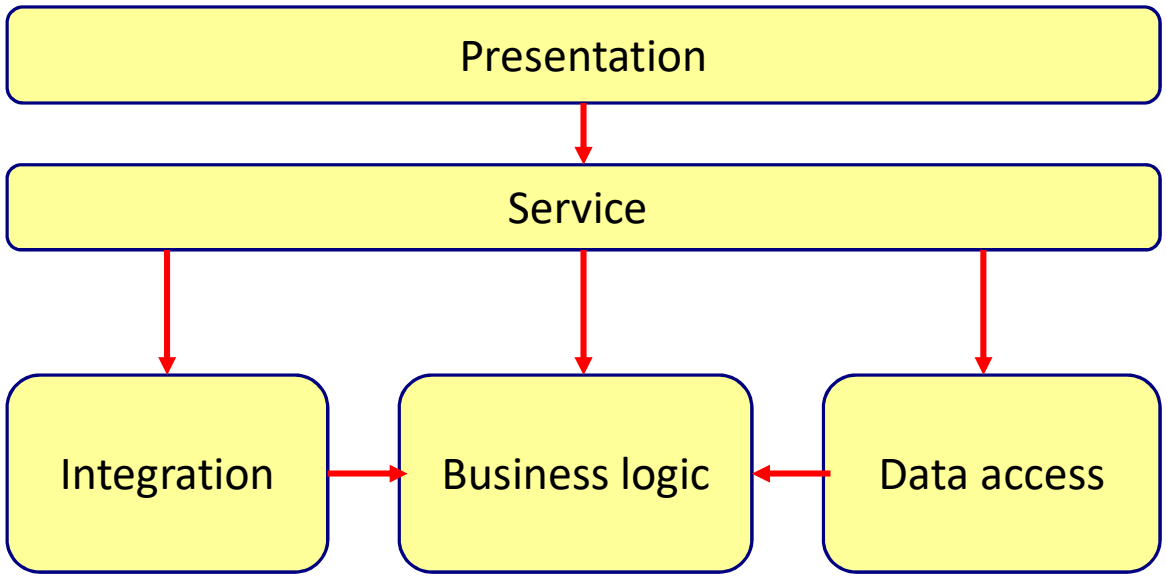
Open and closed layers



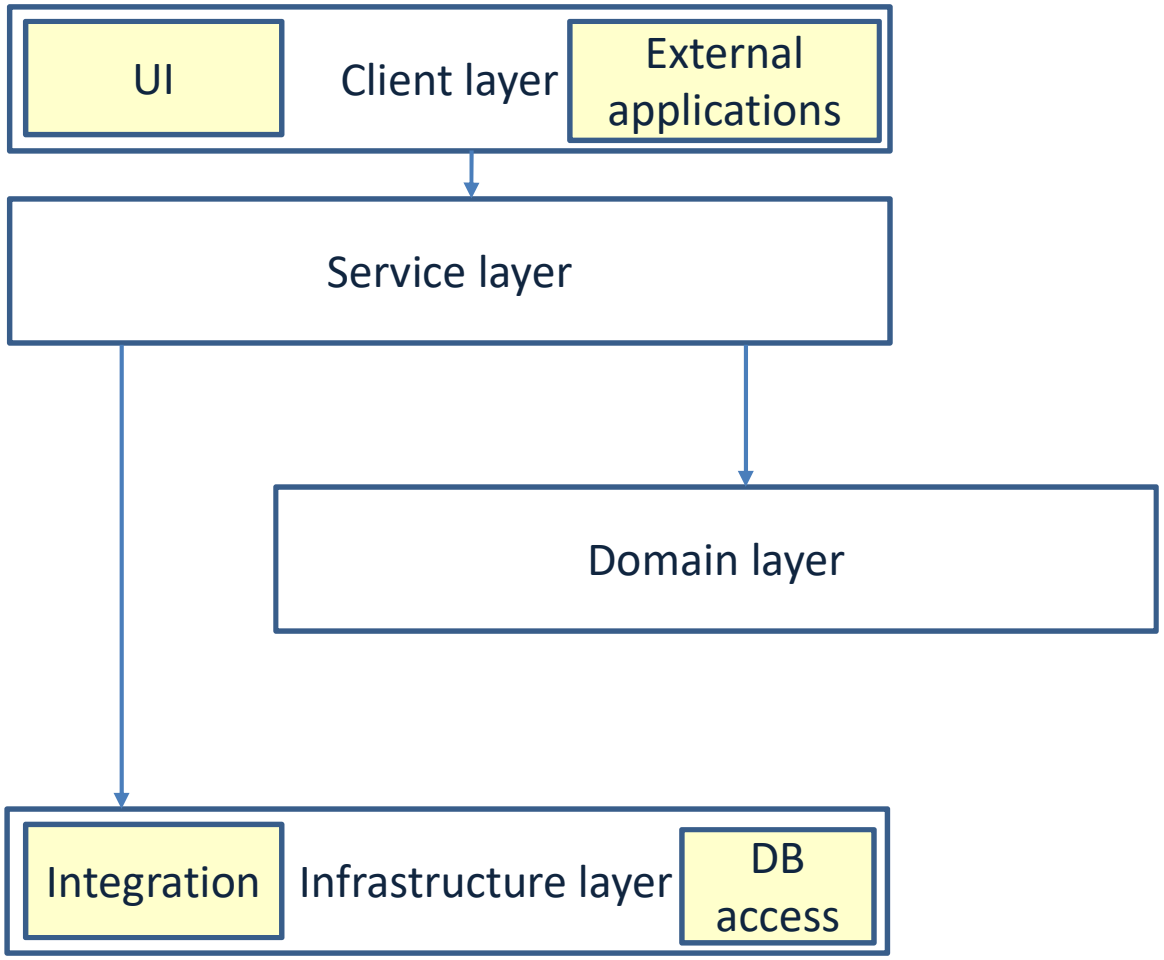
5 layered architecture



Layered architecture



Layered architecture

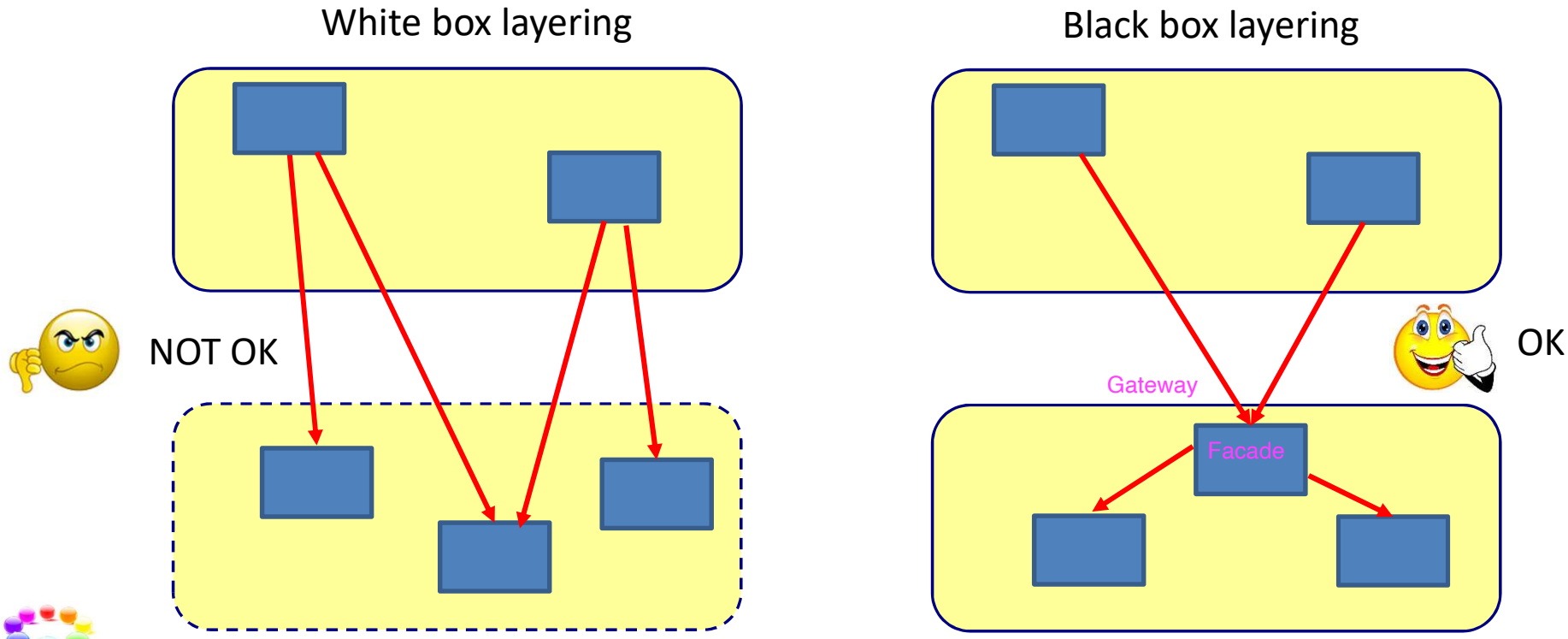


Layering

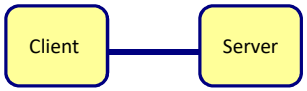
- Benefits
  - Layers can be distributed
  - Separation if concern
    - Different skills required in each layer
    - Easy to modify no need to change anywhere else
    - Easy to test
- Drawbacks
  - Development effort can increase
  - Performance can become an issue

Layering anti patterns

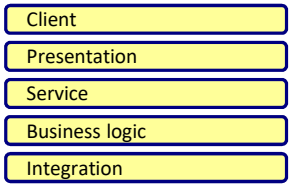
- Too much layers
- No logic in layers
- No encapsulation of layers



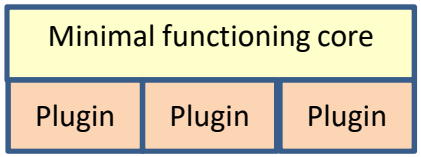
Pipe and Filter



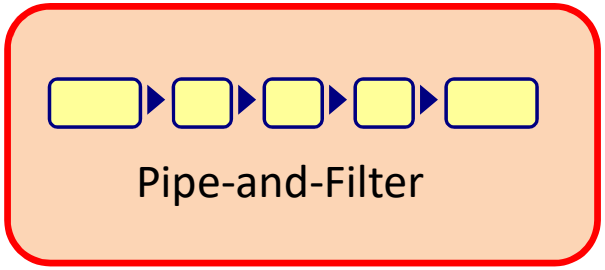
Client-server

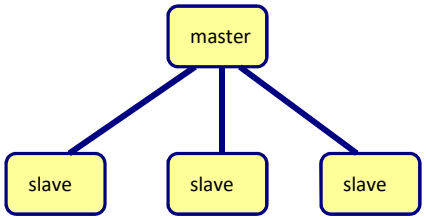


Layering



Microkernel

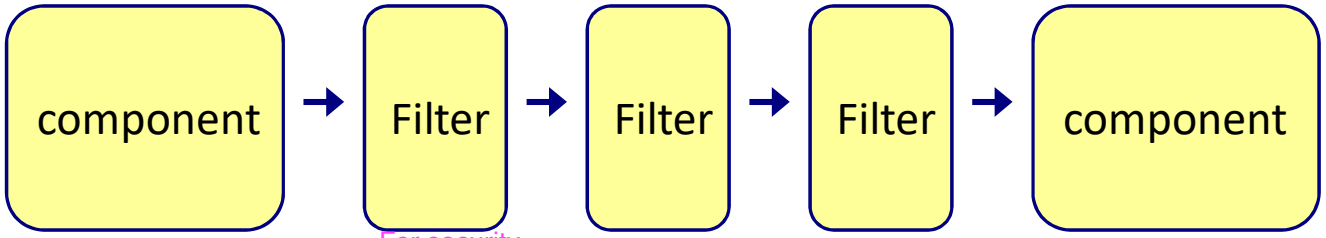




Master-Slave

Pipe and Filter

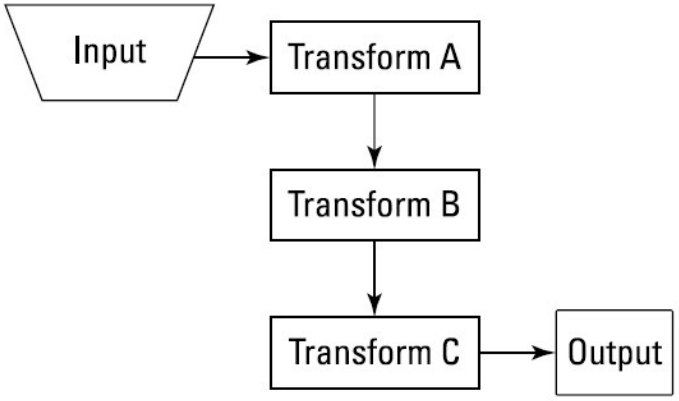
One app is gonna talk with another app



You can add as many as you want
independent
you can apply where ever
change the order

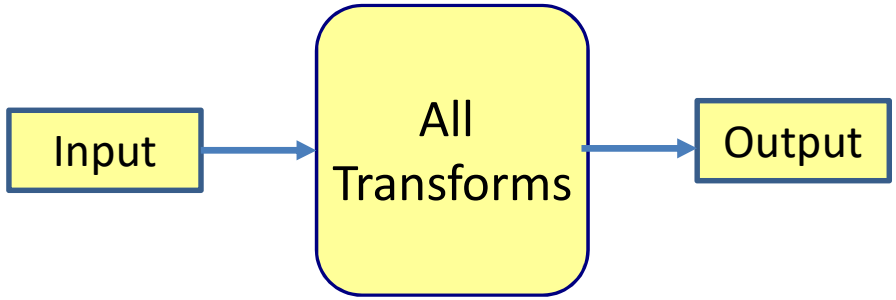
Analyzing an Image Stream

for every thing that have sequence we can apply Filter

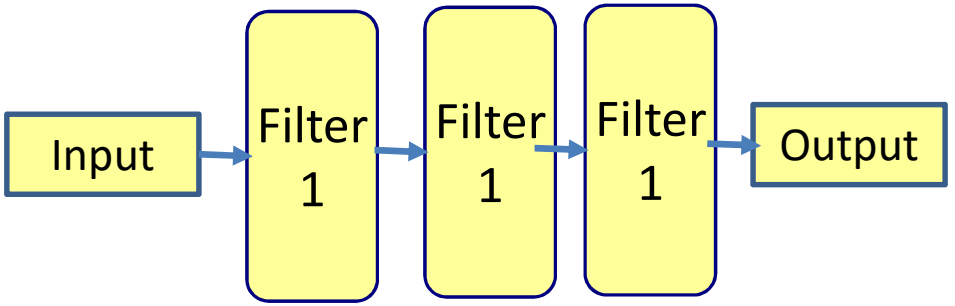


Handler, can be chained
If any of them could handle, will do, otherwise pass to the next handler
When one handler, did, request will come out from chain

One big transformation



Pipe and Filter

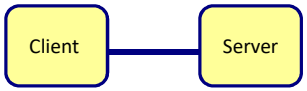


Pipe and Filter

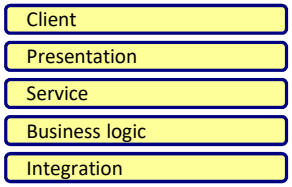
- Benefits
  - Filters are independent
  - Filters are reusable
  - Order of filters can change
  - Easy to add new filters
  - Filters can work in parallel
- Drawbacks
  - Works only for sequential processing
  - Sharing state between filters is difficult

They shouldn't share any state
They are not independent any more

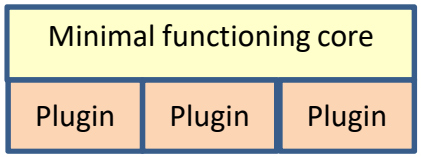
Master slave



Client-server

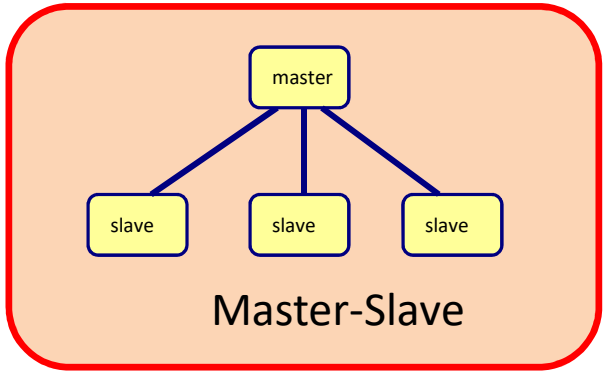


Layering



Microkernel

Pipe-and-Filter



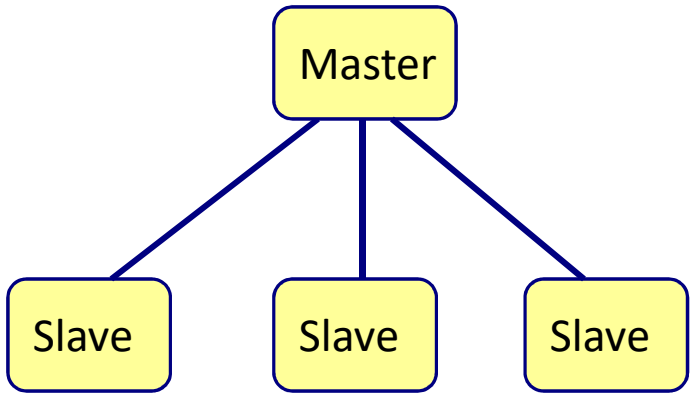
Master-Slave

Some thing that takes a lot of time, can be seperated in pieces to be faster

Master-Slave

Used in DBs

- Master organizes work into distinct subtasks
- Subtasks are allocated to isolated slaves
- Slaves report their result to the master
- Master integrates results

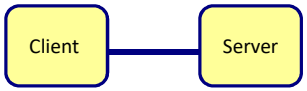


Master slave

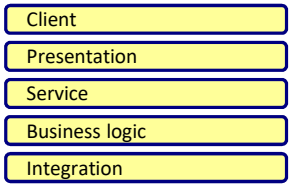
- Benefits
  - Separation of coordination and actual work
  - Master has complete control
  - Slaves are independent
    - No shared state
  - Easy to add new slaves
  - Slaves can work in parallel => Fast
  - Slaves can be duplicated for fault tolerance
- Drawbacks
  - Problem must be decomposable
  - Master is single point of failure

Microkernel

Like a framework

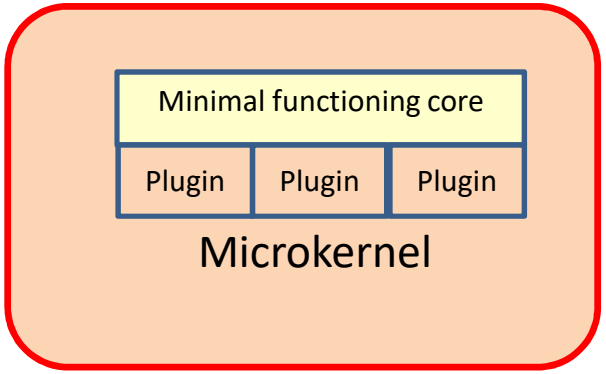


Client-server

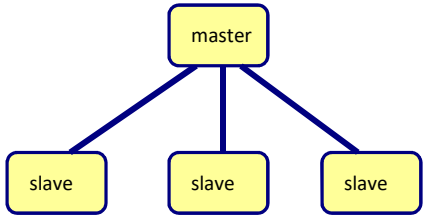


Layering

Pipe-and-Filter



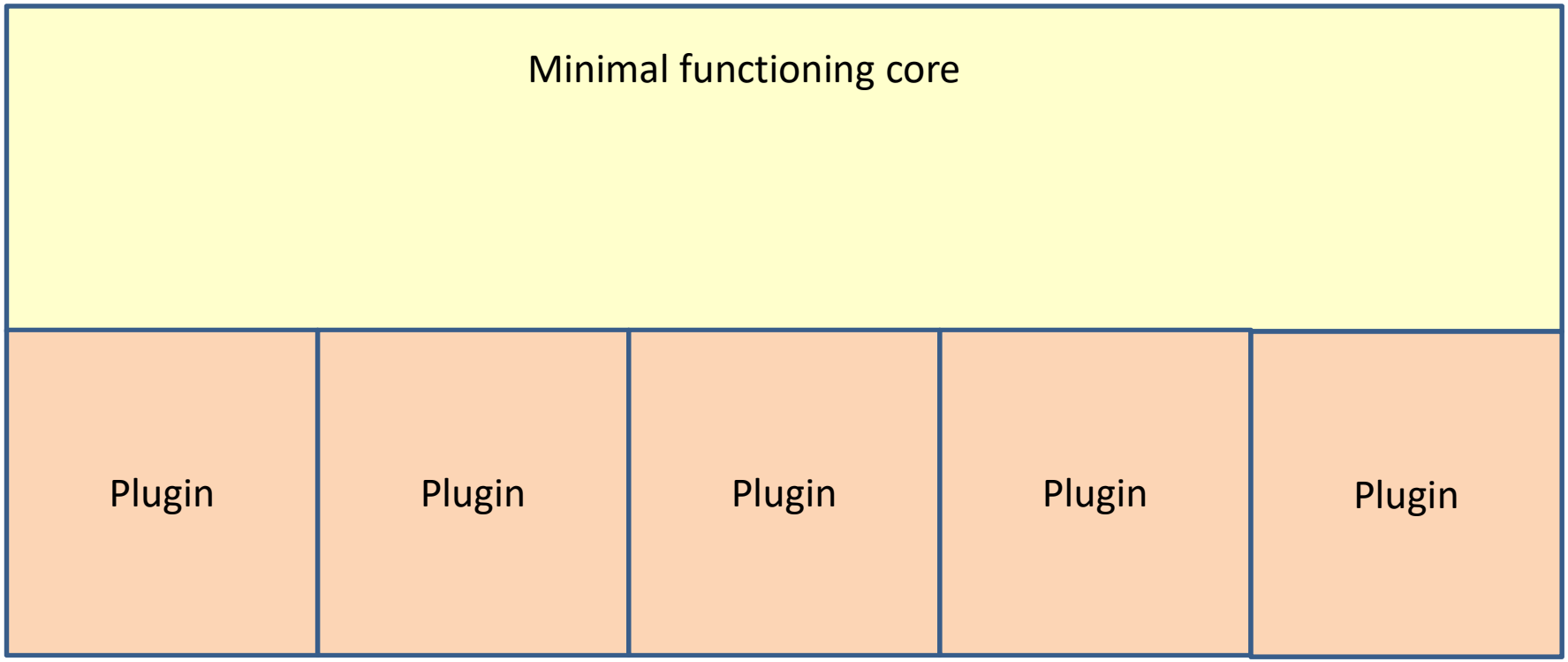
Microkernel



Master-Slave

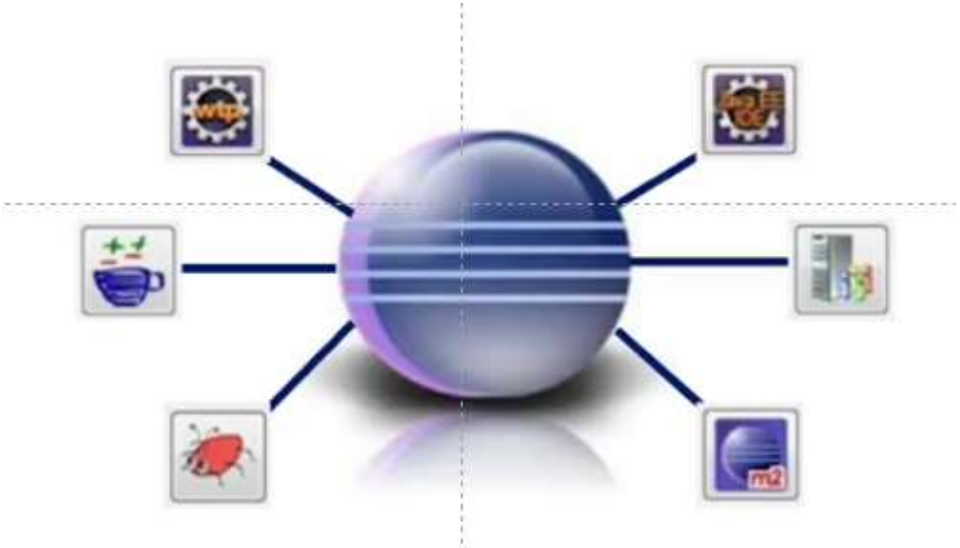
Microkernel

- Plugin application/framework



Microkernel examples

- Eclipse
  - With plugins



- Operating system
  - With drivers

Microkernal

- Benefits
  - Natural for product based apps
  - Extensibility
  - Flexibility Add, with out changing functionality
  - Separation of concern
- Drawbacks
  - Complexity

Main point

- Most architecture styles separate different concerns so that the system becomes:
  - More modular
  - More loosely coupled
  - More flexible
  - Easier to understand and to change
- Harmony exists in diversity

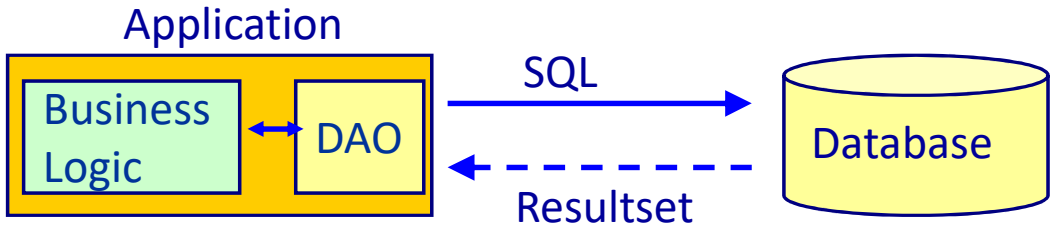
ARCHITECTURE PATTERNS

Data Access Object (DAO)

- Object that knows how to access the database
- Contains all database related logic
- Also called repository

encapsulated class that has all the logic to talk to db

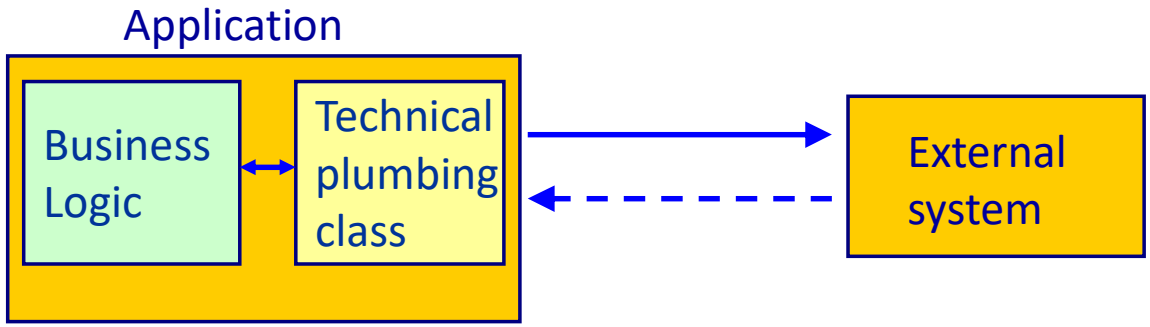
One DAO for every domain class



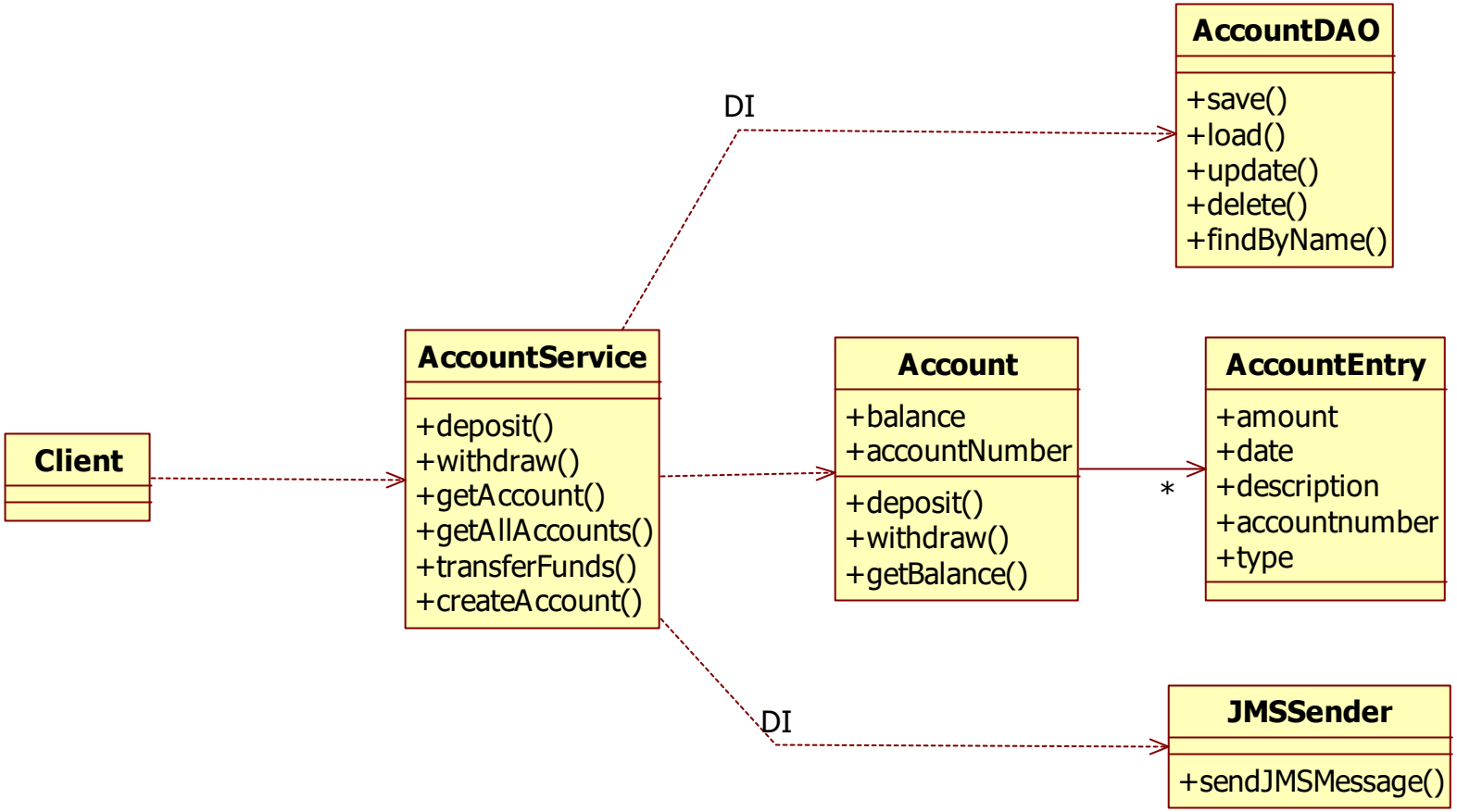
Technical plumbing classes

- Single responsibility
  - Web service
  - Remote calls
  - Messaging
  - Email
  - Logging

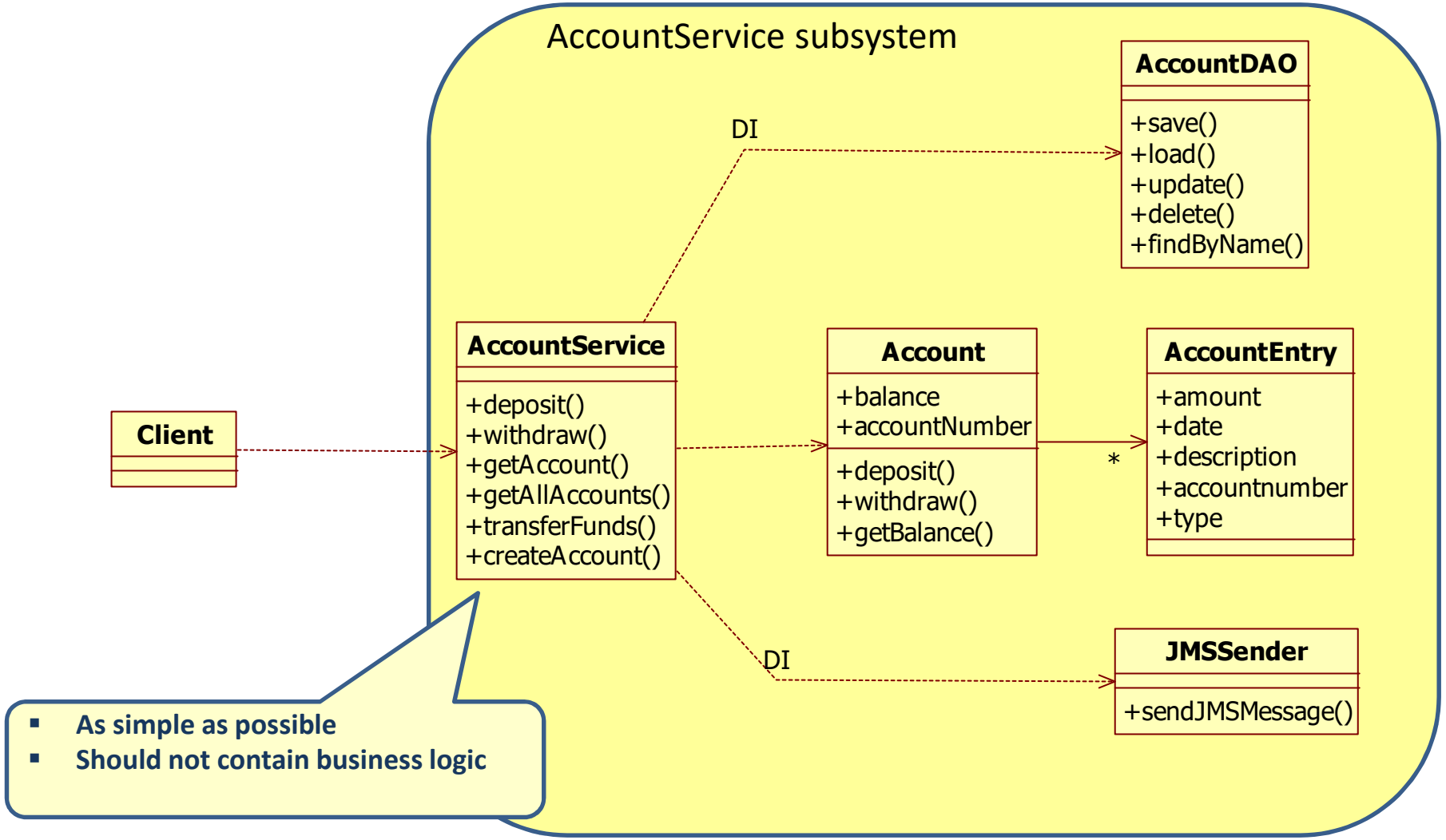
encapsulated class that has all the logic to talk with external syste\m



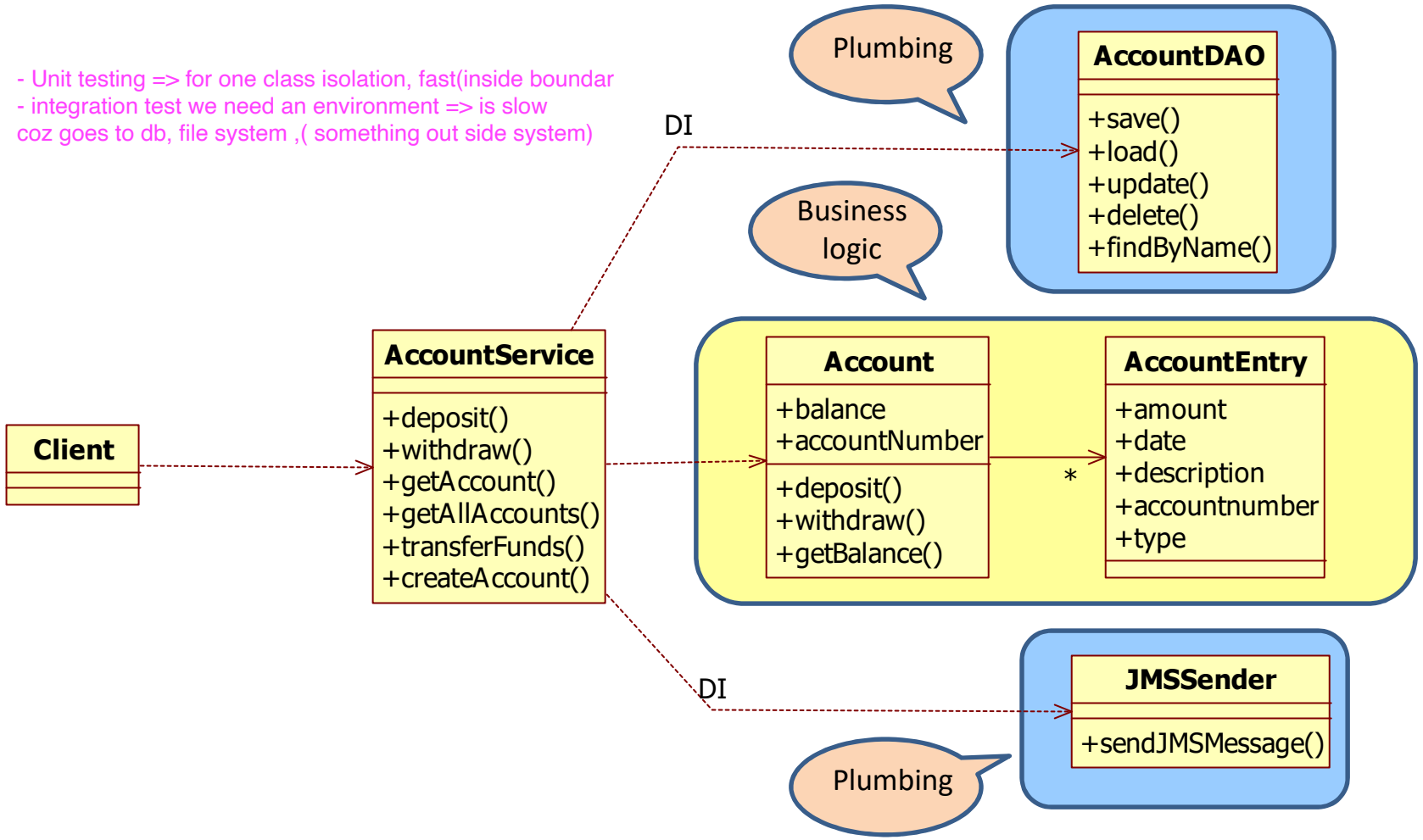
Service Object



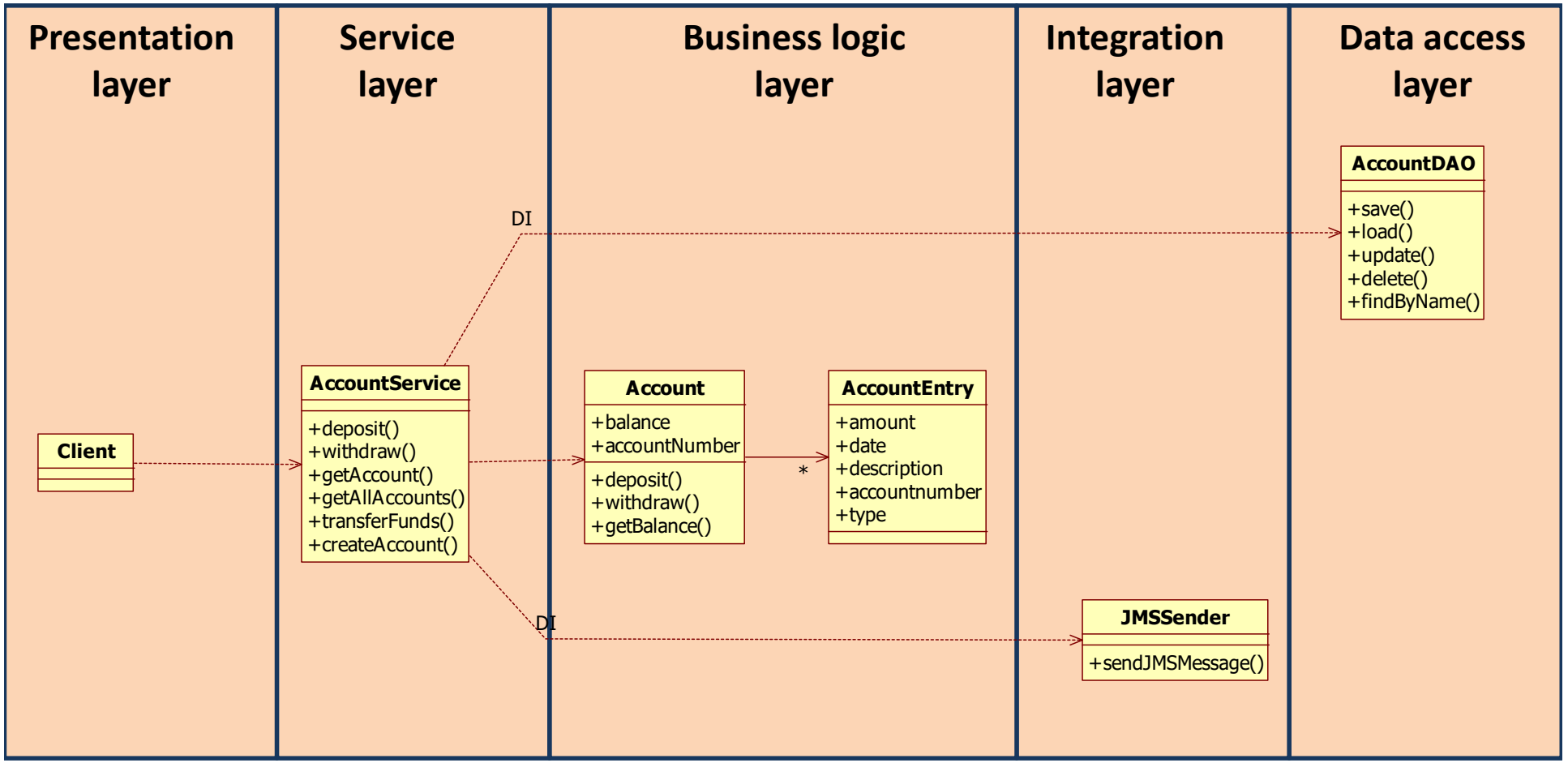
Entry of a complex subsystem



Separation of concern

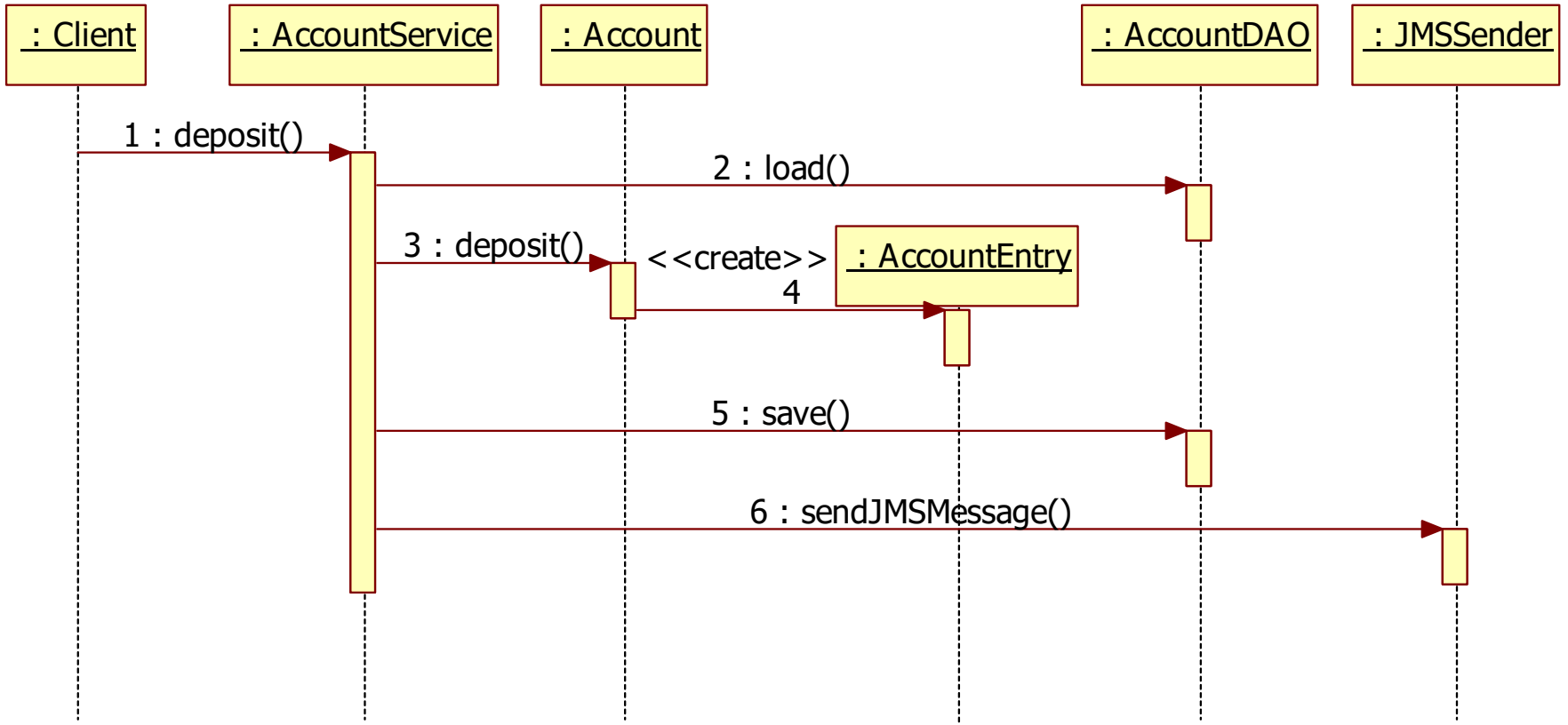


Application layers



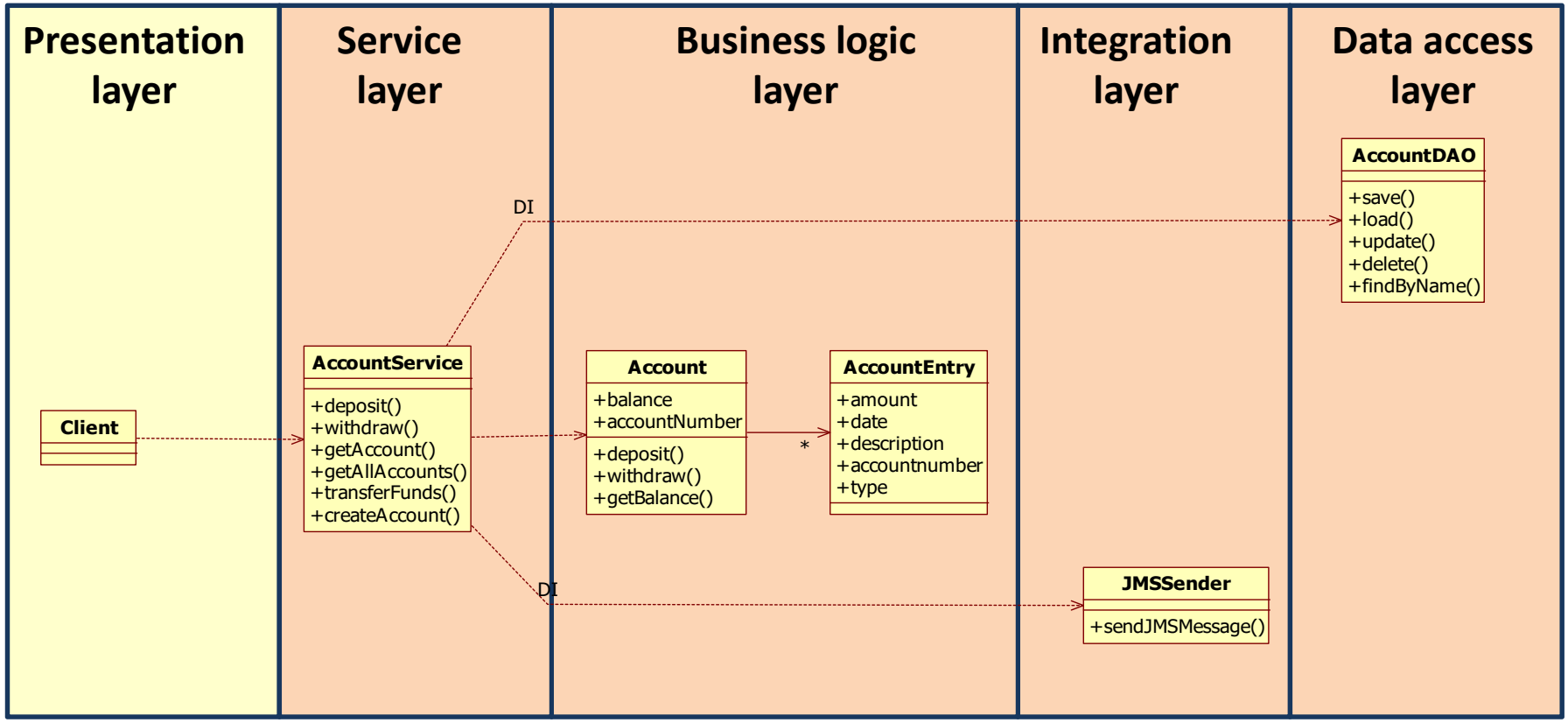
Service object

Service class is only class talking with Plumping



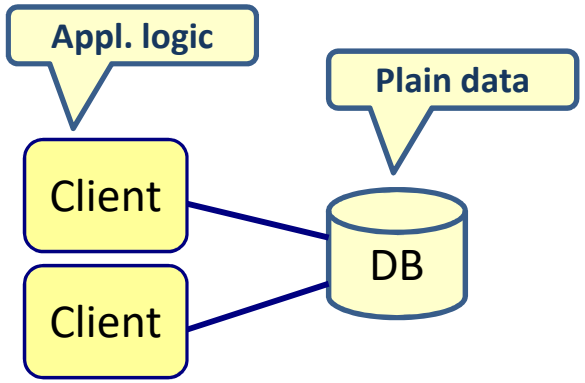
Main point

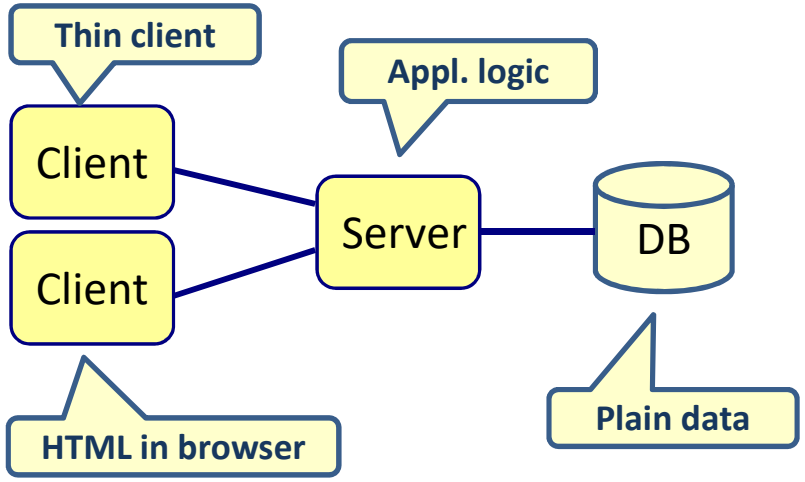
- The domain classes are never aware of technical “plumbing” classes. This gives many different advantages.
- By diving deep into pure consciousness, one gains support of all the laws of nature without needing to know or to be aware of all different details of your life and your world.

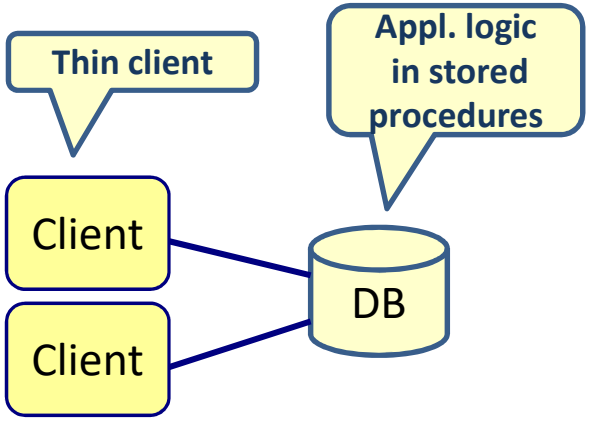


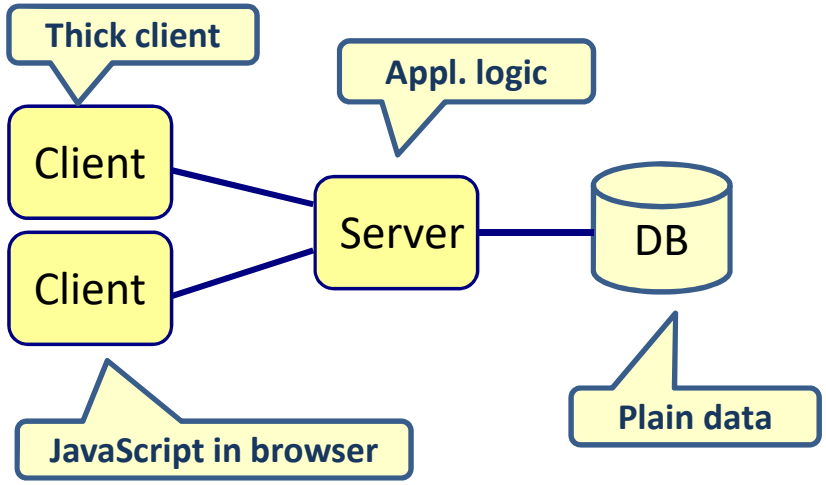
PRESENTATION LAYER

Client-server architectures

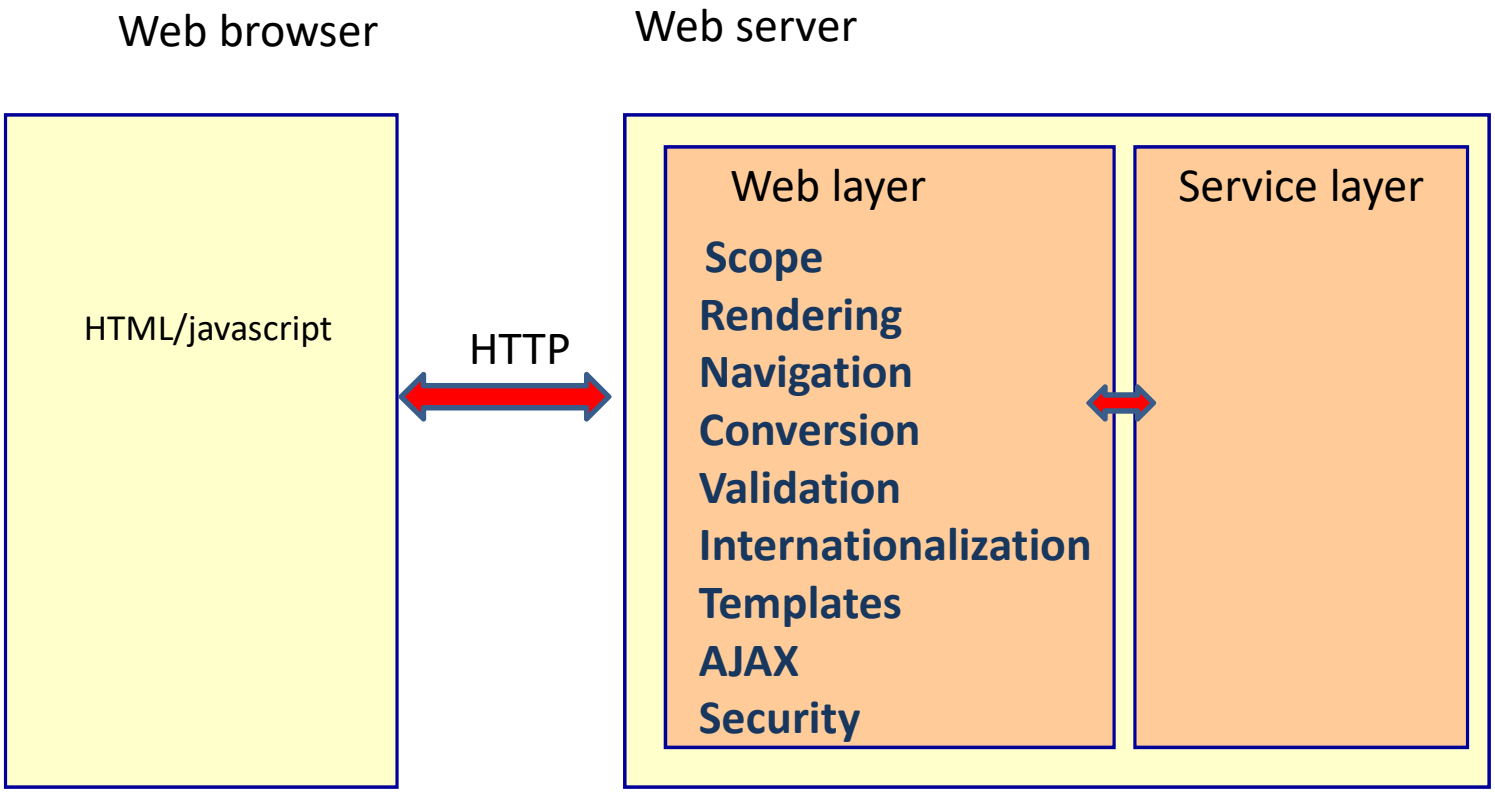






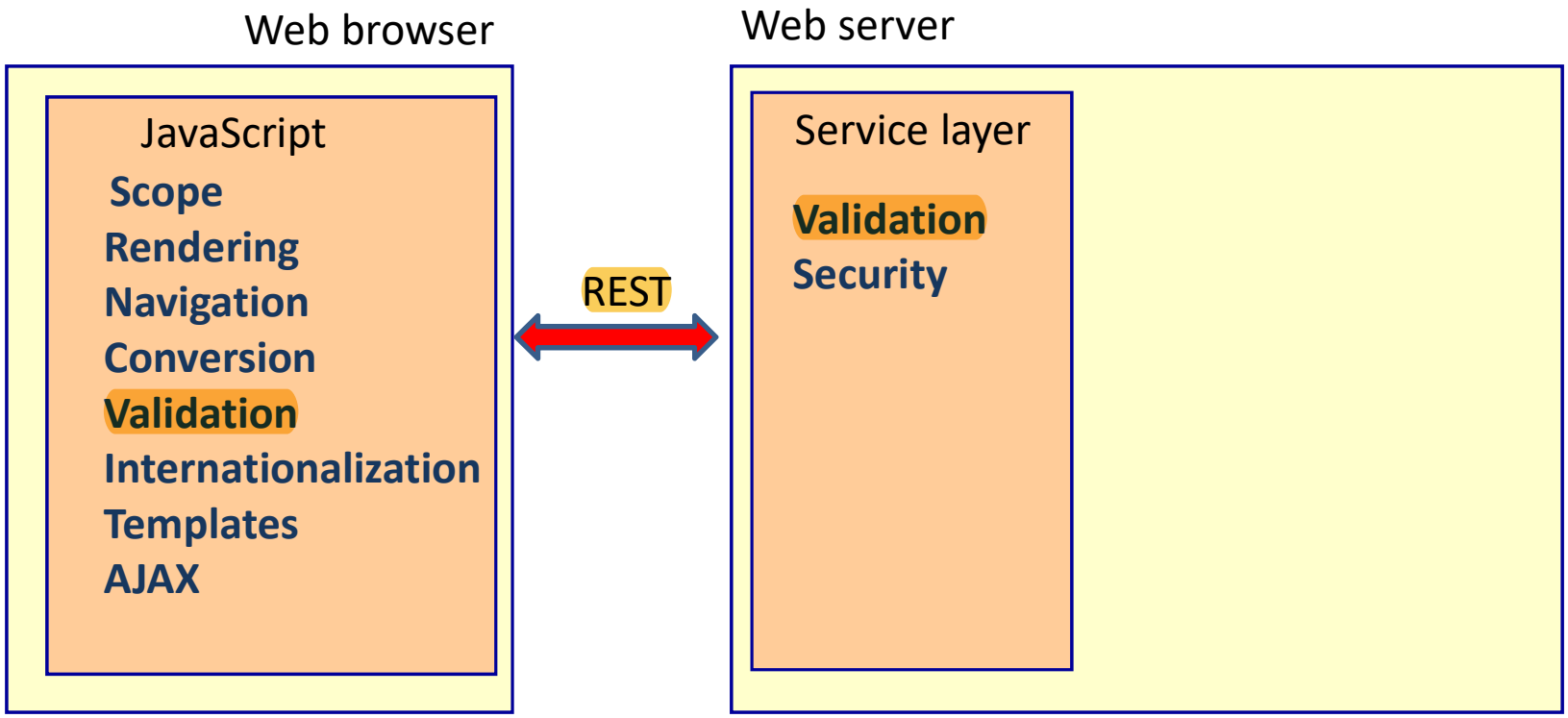


Server side web framework



- Every action is executed on the server

Client side web framework



- You only go to the server if you need to.

Server centric versus client centric

- Remove a stock from the watch list:
  - Sever centric: send a request to the server and execute on the server
  - Client centric: execute within the browser

Server side web framework

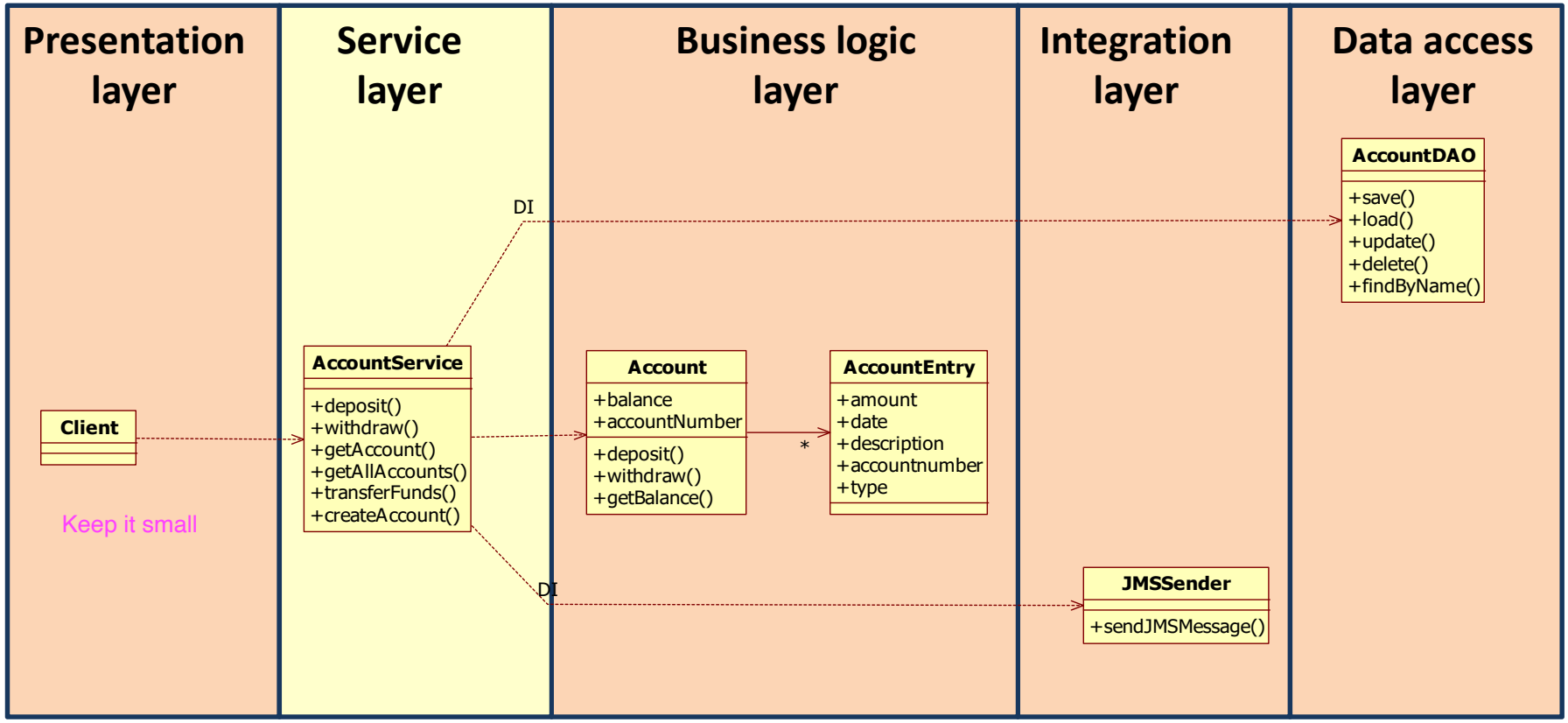
- Example: SpringMVC, ASP.net

- Advantages
  - Stable frameworks
- Disadvantages
  - More network calls
  - Less reactive
  - Less scalable
    - If your store session state

Client side web framework

- Example: Angular, React

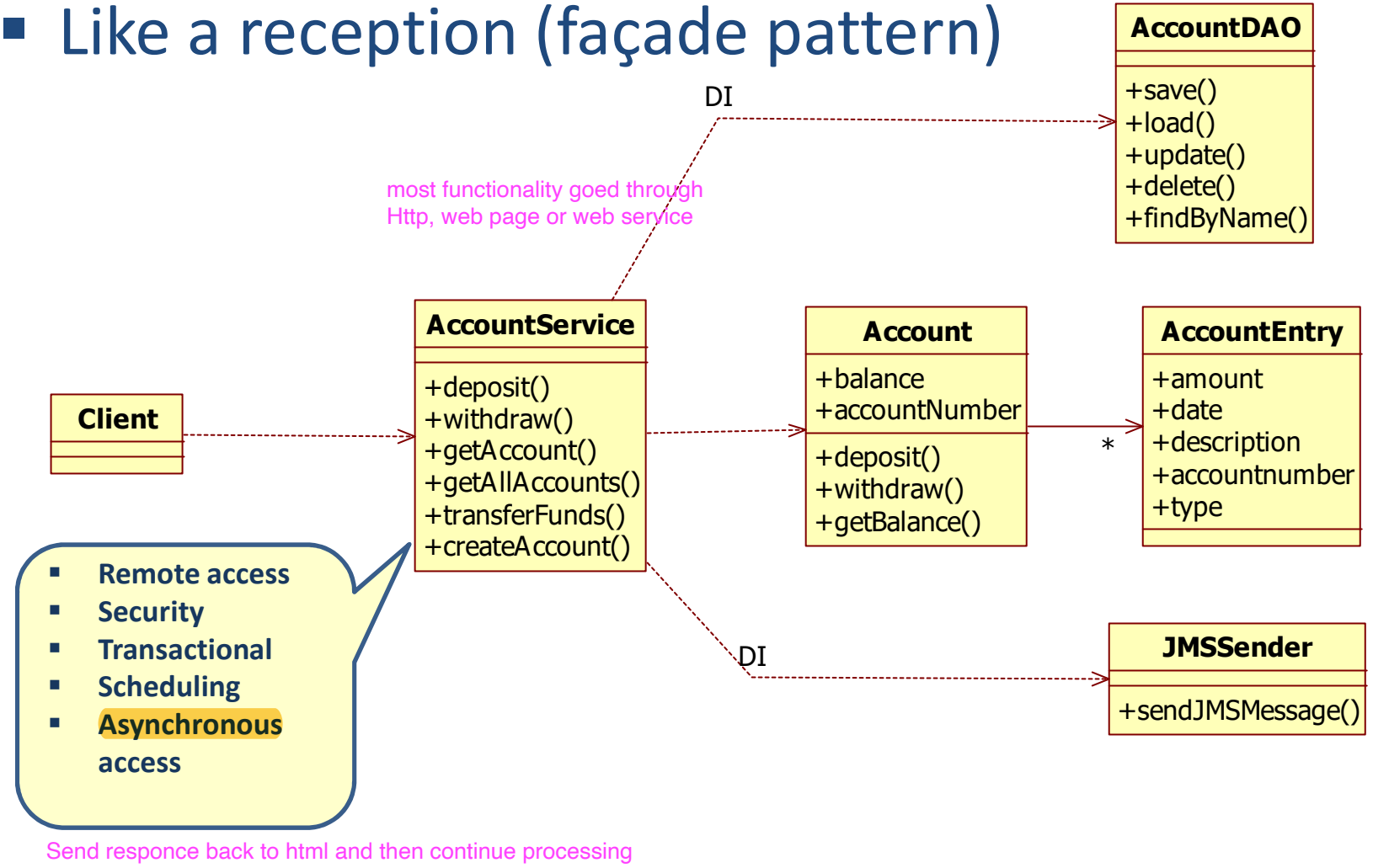
- Advantages
  - Less network calls
  - More reactive
  - Very scalable
    - The state is stored on the client application(browser)
- Disadvantages
  - Frameworks change very frequently

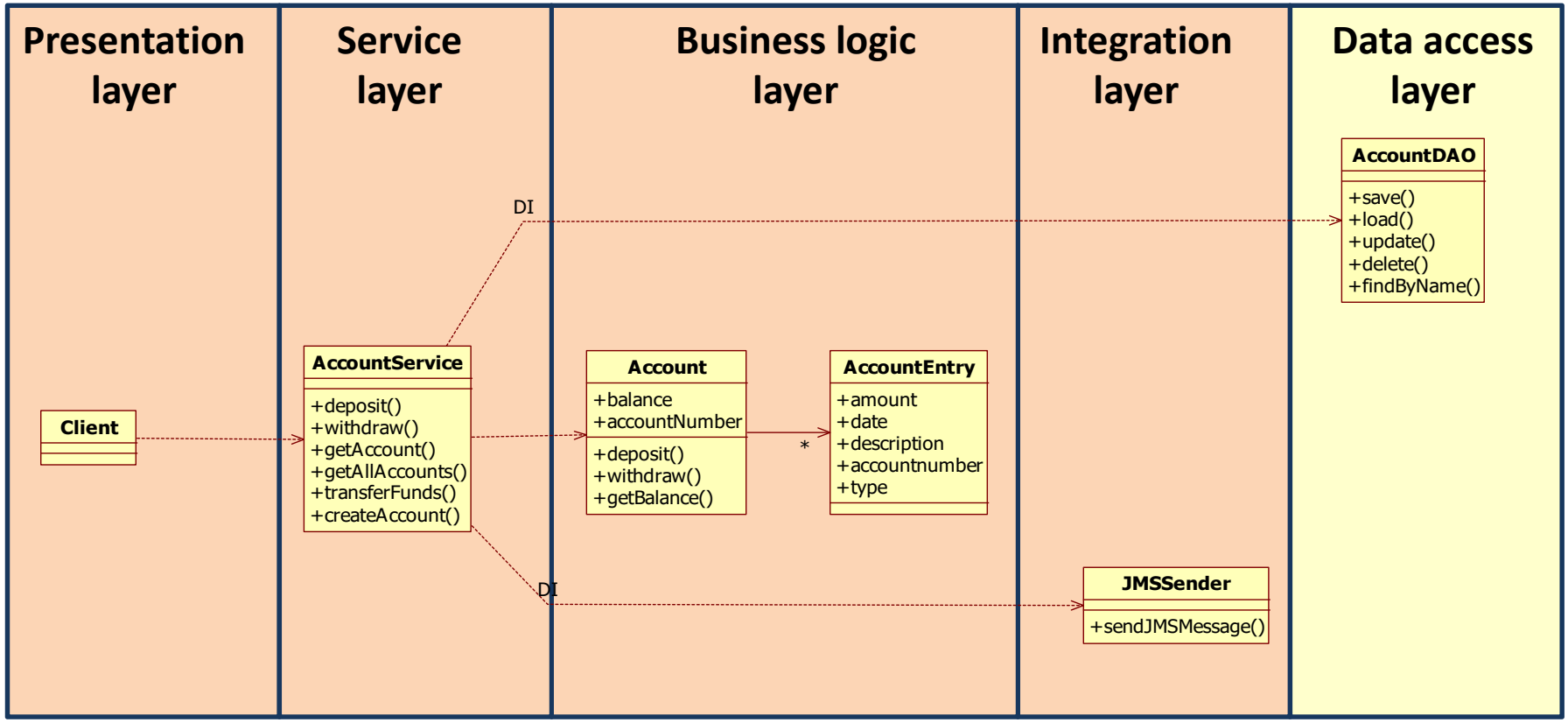


SERVICE LAYER

Service class

- Like a reception (façade pattern)

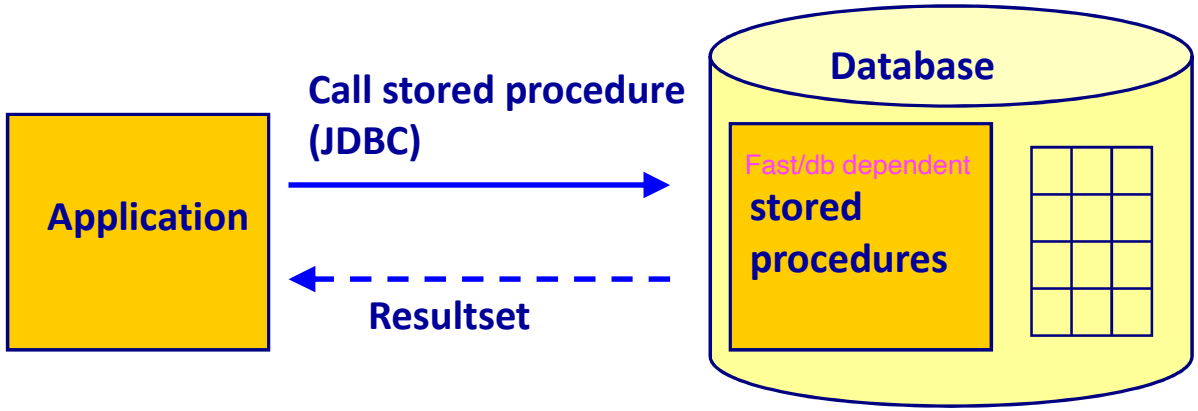




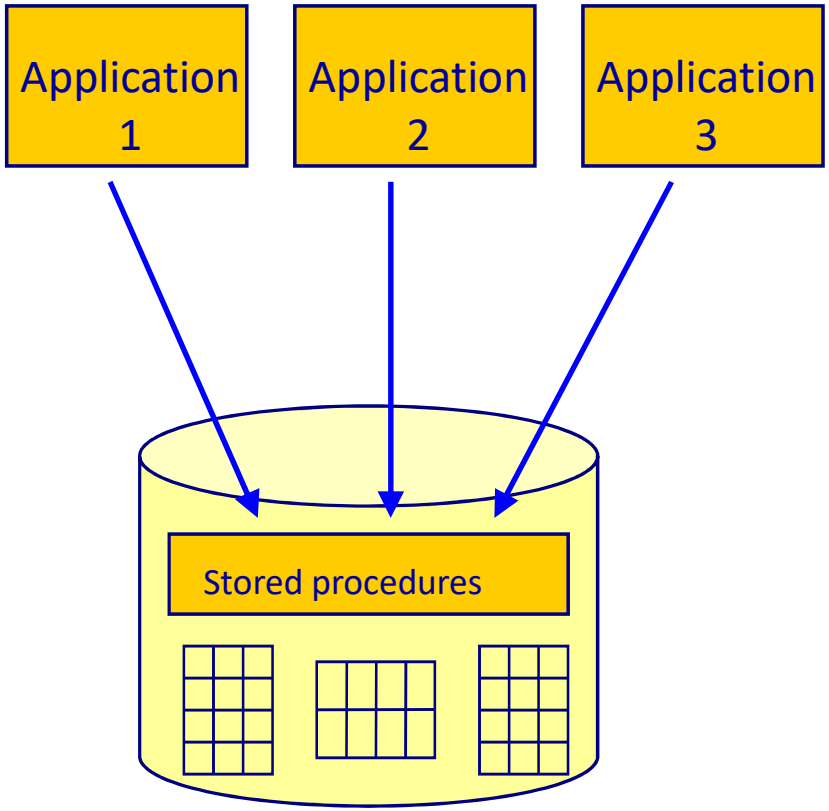
DATA ACCESS LAYER

Stored procedures

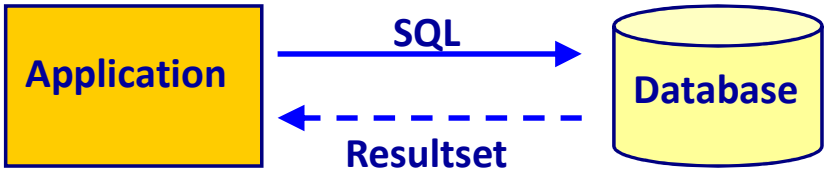
- Logic that runs in the database
- Fast
- Difficult to maintain when number of stored procedures grows
  - Every schema change leads to changes to the stored procedures
  - Lot of duplications, not much reuse
- PL/SQL
- Java Stored Procedures



Layer of indirection

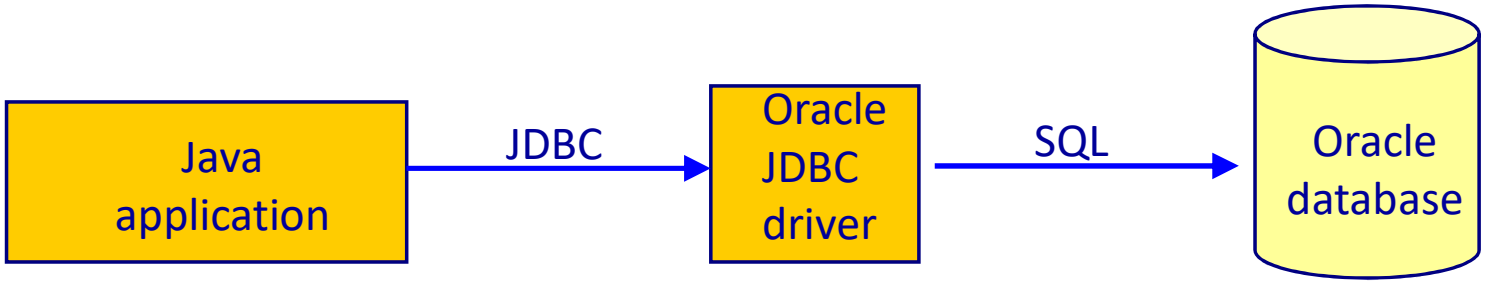


SQL based approach: JDBC



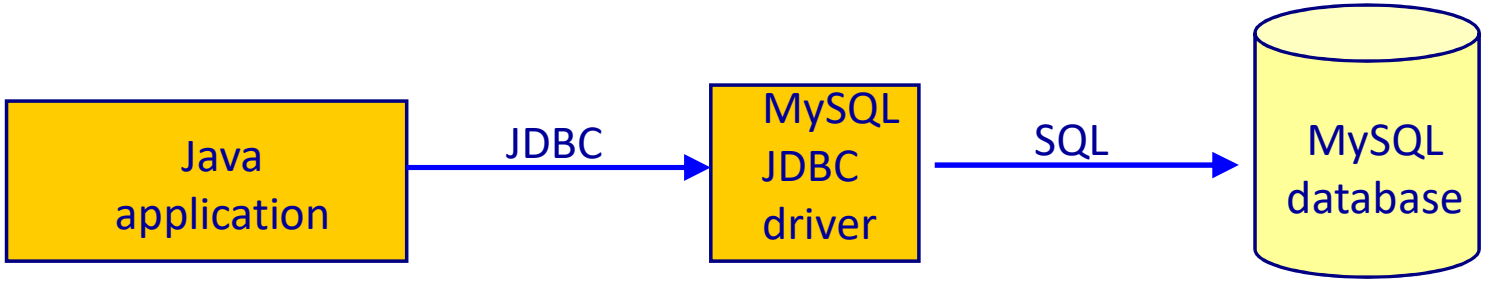
JDBC

You write the same code & JDBC translate it for different dbs



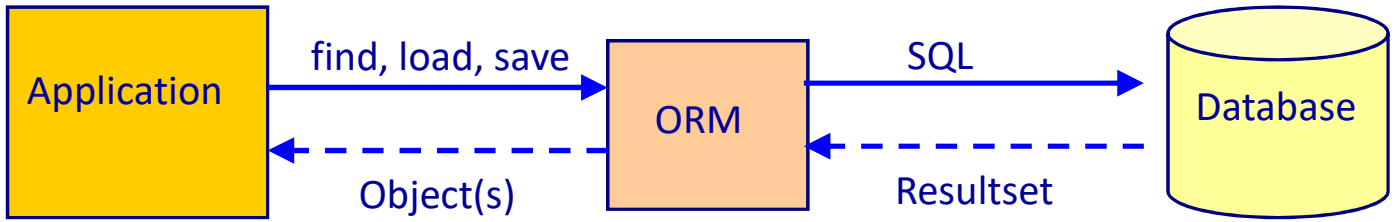
Tight coupling between db and app

hard to maintain, hard to optimize, difficult to debug because it is String



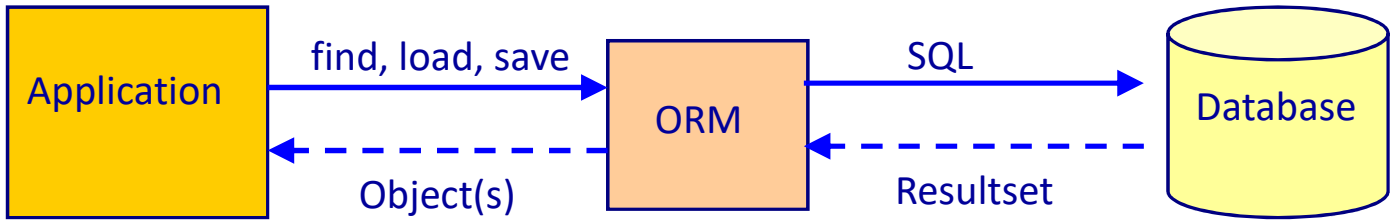
Object Relational Mapping (ORM)

- Object Relational Mapping lets the programmer focus on the Object Model
  - Supports Domain Driven Development (DDD)
  - Programmer can just work with objects
  - Once an object has been retrieved any related objects are automatically loaded as needed
  - Changes to objects can automatically be stored in the database



Advantages of ORM

Advantage	Details
Productivity	•Fewer lines of persistency code
Maintainability	•Fewer lines of persistency code •Mapping is defined in one place
Performance	•Caching •Higher productivity gives more time for optimization ✓Projects under time pressure often don't have time for optimization •The developers of the ORM put a lot of effort in optimizing the ORM



Transactions

- A Transaction is a unit of work that is:
  - **ATOMIC:** The transaction is considered a single unit, either the entire transaction completes, or the entire transaction fails.
  - **CONSISTENT:** A transaction transforms the database from one consistent state to another consistent state Nothing never loses, even temprary
  - **ISOLATED:** Data inside a transaction can not be changed by another concurrent processes until the transaction has been committed
  - **DURABLE:** Once committed, the changes made by a transaction are persistent

BIG DATA

3 V's of Big Data

- Volume
  - We need to handle large volumes of data
  - Still growing
- Velocity
  - Data needs to be used quickly to maximize business benefit before the value of the information is lost.
- Variability
  - Data can be structured, unstructured, semi-structured or a mix of all three. It comes in many forms including text, audio, video, click streams and log files.

Relational databases are great

- SQL provides a rich, declarative query language
- Database enforce referential integrity
- ACID semantics
- Well understood by developers, database administrators
- Well supported by different languages, frameworks and tools
  - Hibernate, JPA, JDBC, iBATIS
- Well understood and accepted by operations people (DBAs)
  - Configuration
  - Monitoring
  - Backup and Recovery
  - Tuning
  - Design

Relational databases are great … but

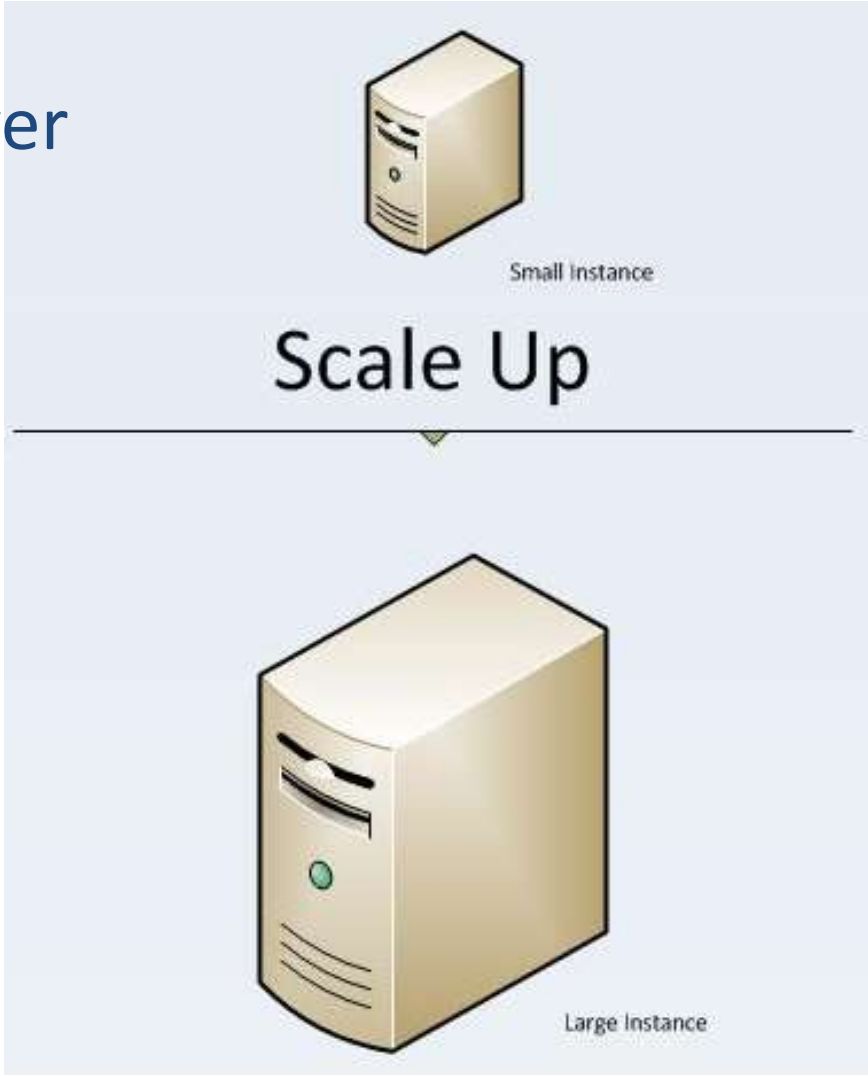
- Object/Relational impedance mismatch
  - Complicated to map rich domain model to relational schema
  - Performance issues
- Schema evolution
  - Adding attributes to an object => have to add columns to table
  - Expensive, if lots of data in that table
  - Holding locks on the tables for long time
  - Application downtime ...

Relational databases are great … but

- Relational schema doesn‘t easily handle semi-structured data
  Such as pic or movie => no need to query on them => no need to be in relational structure
  - Common solutions
    - Name/Value table
      - Poor performance
      - Lack of constraint
    - Serialize as Blob
      - Fewer joins, but no query capabilities
- Scaling writes are difficult/expensive/impossible => BigData
  - Vertical scaling is limited and is expensive
  - Horizontal scaling is limited and is expensive

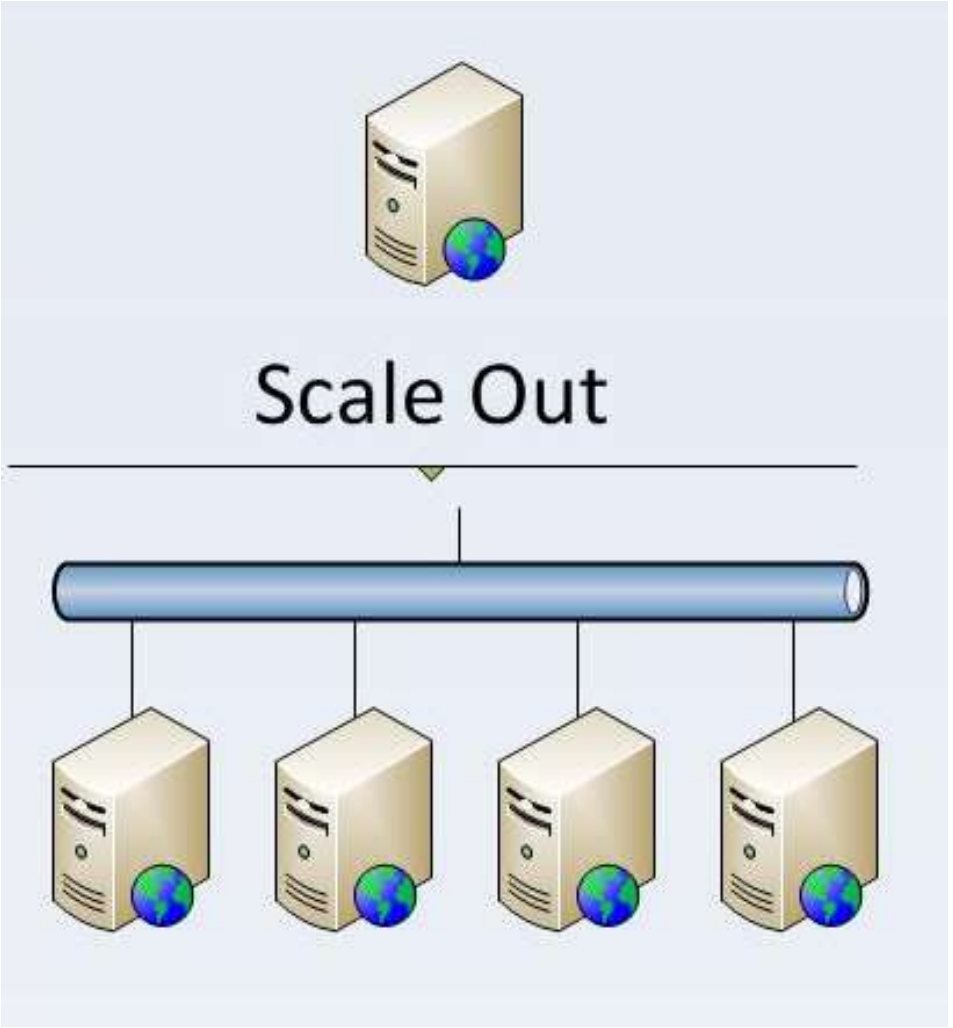
Vertical Scaling

- Scale up
- Use a more powerful server
- Single point of failure
- Upgrading results in downtime
- Limitations
  - Cost
  - Software does not use all resources
  - Hardware
- Vendor lock-in



Horizontal scaling

- Scale out
- Divide the data over multiple servers
- Easy to add more servers
  - Without downtime



Horizontal scaling

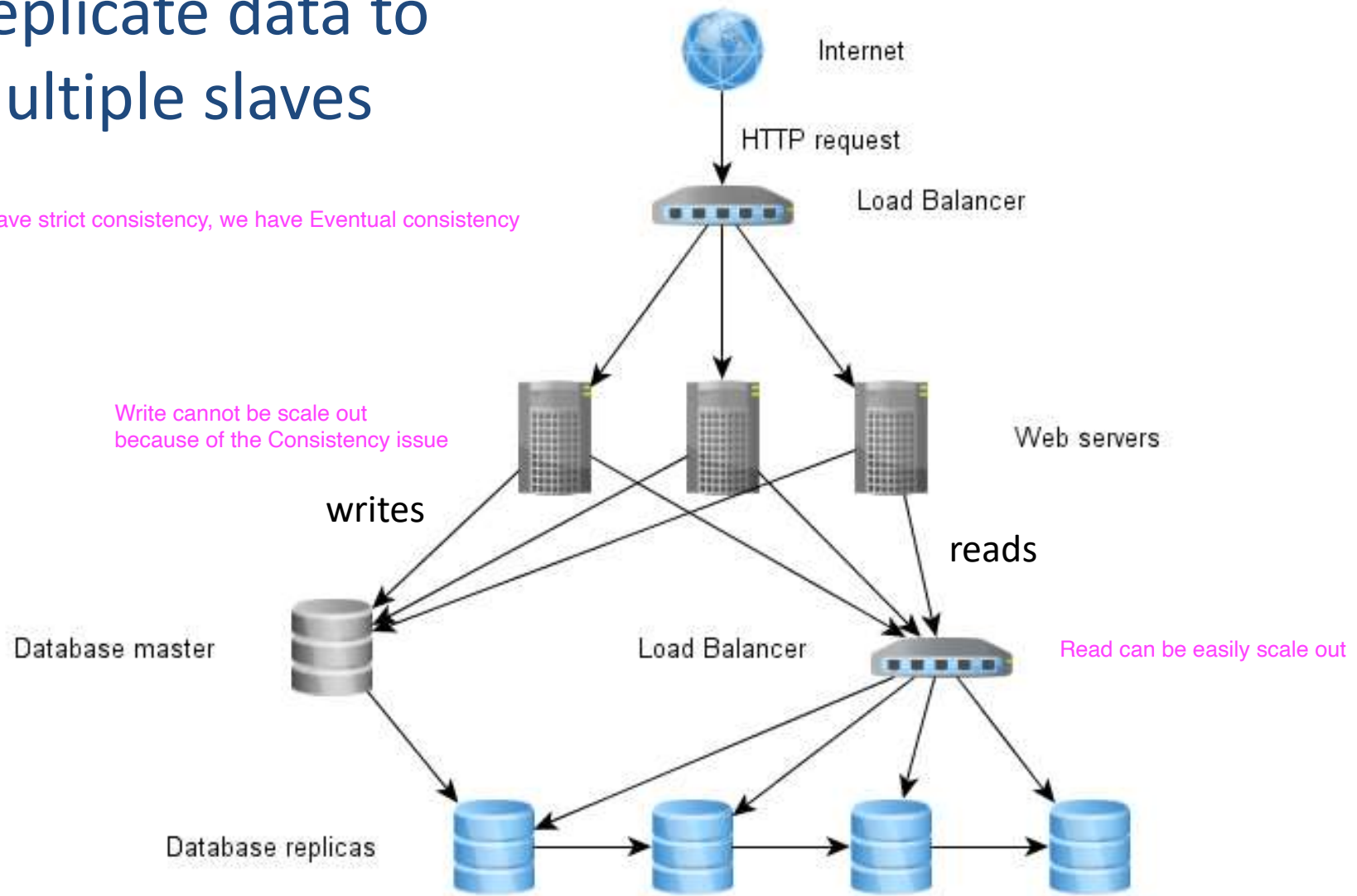
- Replication
- Partitioning
- Sharding

Different ways to do H scaling

Replication

- Replicate data to multiple slaves

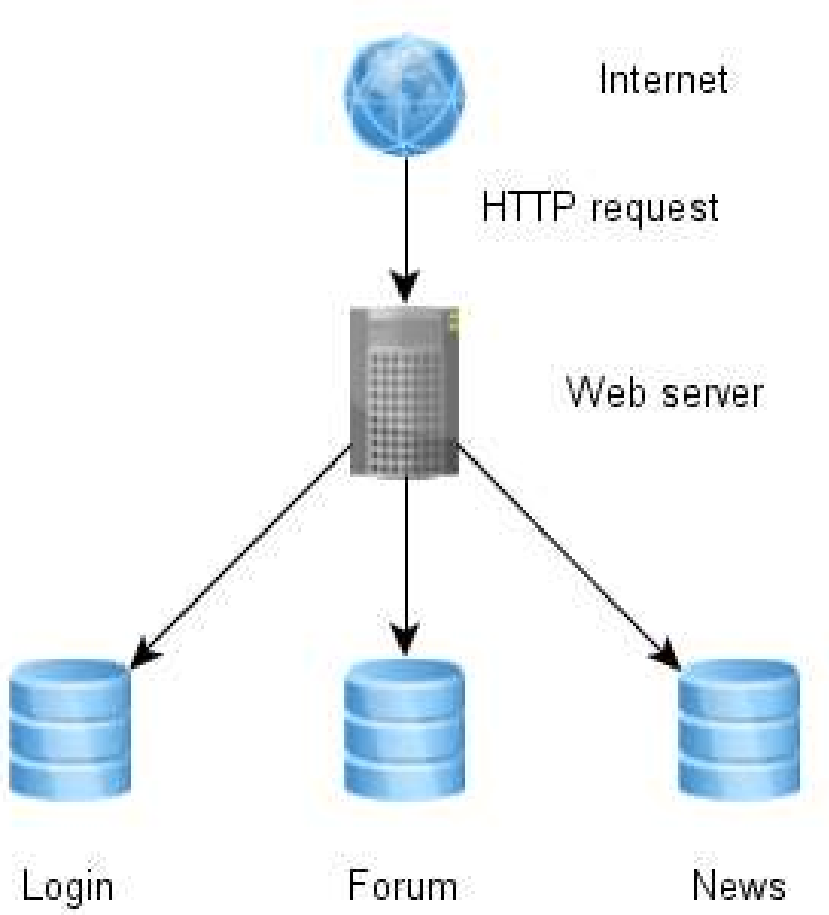
For write we don't have strict consistency, we have Eventual consistency



Functional partitioning

- Split up data in functional areas

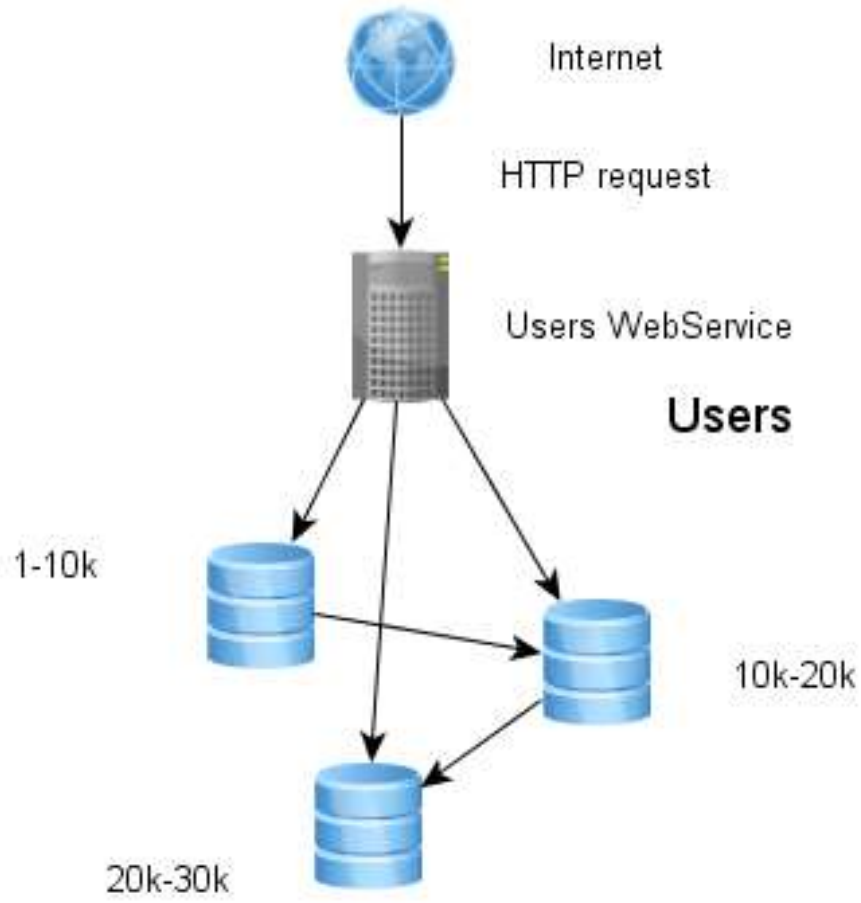
Problem is for the time that we wanna do the Join



Sharding

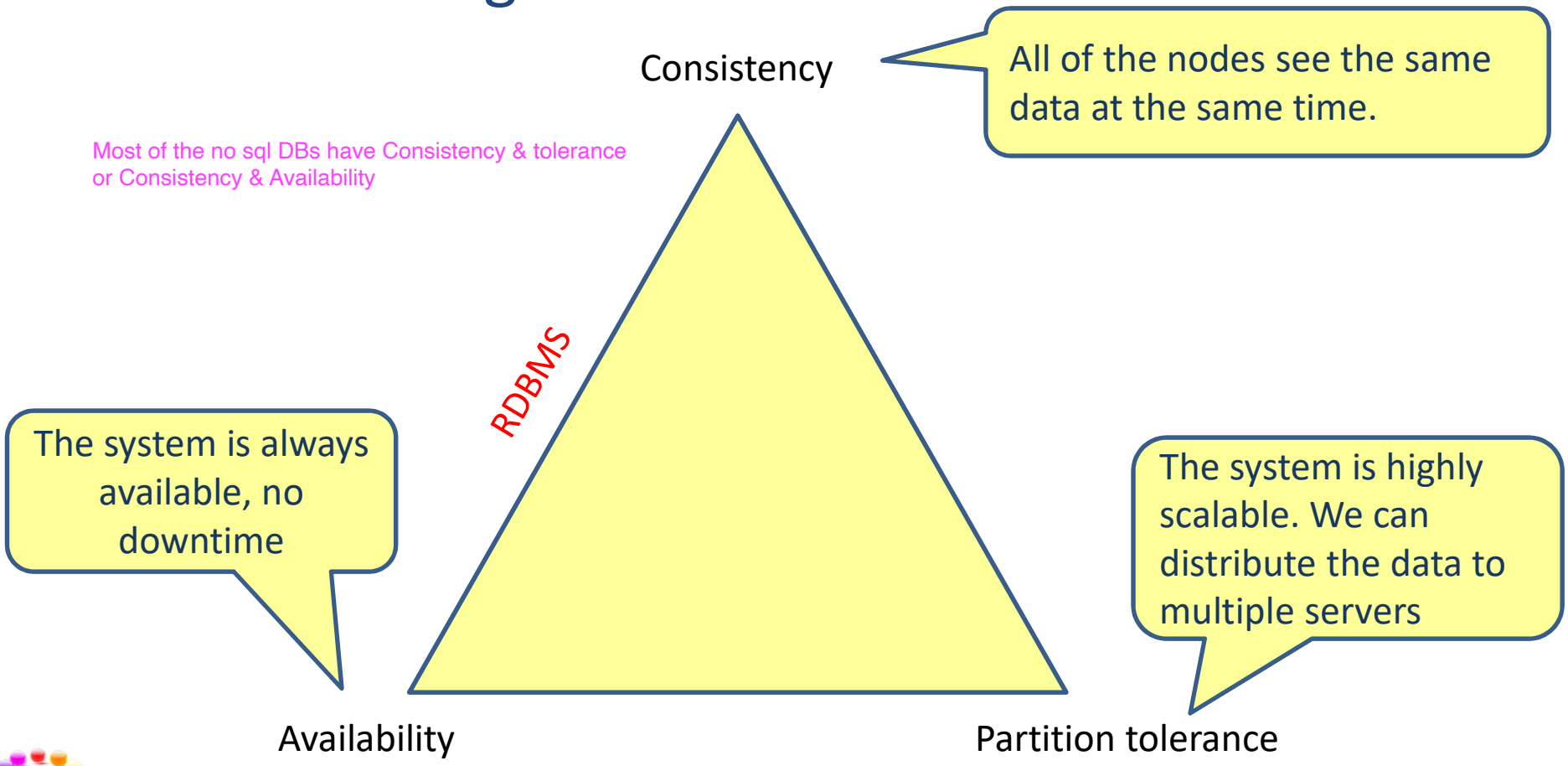
- Split the data into pieces(shards) and store them on different nodes

The problem is Join and queries for getting all the data



Brewer‘s CAP Theorem

- A distributed system can support only two of the following characteristics



NoSQL or RDBMS

NoSQL

- Schema-free
- Scalable writes/reads
- Auto high-availability
- Eventual consistency

RDBMS

- Relational schema
- Scalable reads
- Custom high-availability
- Strict consistency

MongoDB features

- Document model
  - No fixed schema
- Queries
- Indexes
- Scaling
  - Auto-sharding
- Replication

Document data model (JSON)

For giving structure to data

Relational - Tables

Customer ID	First Name	Last Name	City
0	John	Doe	New York
1	Mark	Smith	San Francisco
2	Jay	Black	Newark
3	Meagan	White	London
4	Edward	Daniels	Boston

Account Number	Branch ID	Account Type	Customer ID
10	100	Checking	0
11	101	Savings	0
12	101	IRA	0
13	200	Checking	1
14	200	Savings	1
15	201	IRA	2

Document - Collections

```
{   customer_id : 1,
    first_name : "Mark",
    last_name : "Smith",
    city : "San Francisco",
    accounts : [   {
        account_number : 13,
        branch_ID : 200,
        account_type : "Checking"
    },
    {   account_number : 14,
        branch_ID : 200,
        account_type : ”IRA”,
        beneficiaries: […]
    } ]
}
```

Disadvantage = Redundency

Documents are rich structures

```
{
        category: “glove”,
        model: “PRO112PT”,
        name: “Air Elite”,
        brand: “Rawlings”,
        price: 229.99,
        available: Date(“2013-03-31”),
        position: [“infield”, “outfield”, “pitcher”]
}
```

Fields can contain arrays

Documents are rich structures

```
{
        category: “glove”,
        model: “PRO112PT”,
        name: “Air Elite”,
        brand: “Rawlings”,
        price: 229.99,
        available: Date(“2013-03-31”),
        position: [“infield”, “outfield”, “pitcher”],
        endorsed: {name: “Ryan Howard”,
                   team: “Phillies”,
                   position: “first base”},
}
```

Fields can contain sub-documents

Documents are rich structures

```
{
        category: “glove”,
        model: “PRO112PT”,
        name: “Air Elite”,
        brand: “Rawlings”,
        price: 229.99,
        available: Date(“2013-03-31”),
        position: [“infield”, “outfield”, “pitcher”],
        endorsed: {name: “Ryan Howard”,
                   team: “Phillies”,
                   position: “first base”},
            history: [{date: Date(“2013-03-31”),  price: 279.99},
                      {date: Date(“2013-06-01”), price: 259.79},
                      {date: Date(“2013-08-15”), price: 229.99}]
}
```

Fields can contain an array of sub-documents

Documents are flexible

schema less

```
{
        category: bat,
        model: B1403E,
        name: Air Elite,
        brand: “Rip-IT”,
        price: 399.99

        diameter: “2 5/8”,
        barrel: R2 Alloy,
        handle: R2
}
```

```
{
        category: glove,
        model: PRO112PT,
        name: Air Elite,
        brand: “Rawlings”,
        price: “229.99”

        size: 11.25,
        position: outfield,
        pattern: “Pro taper”,
        material: leather,
        color: black
}
```

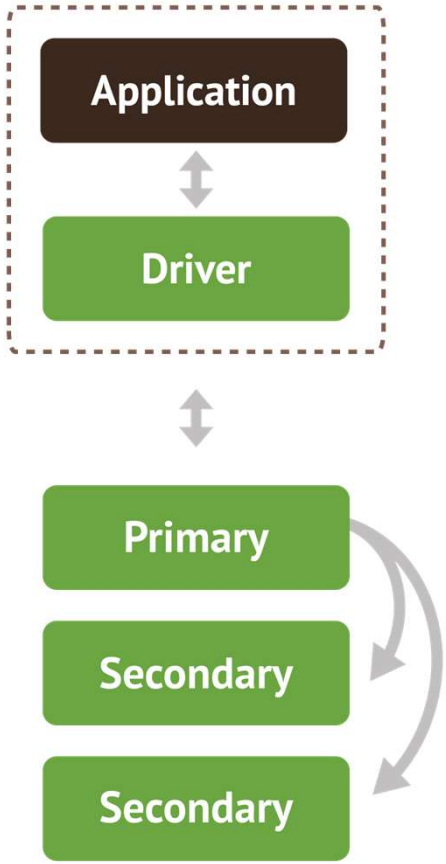
Automatic Sharding

Automatically done by DB
You can configure it

- Increase or decrease capacity as you go
- Automatic balancing

Replica Sets

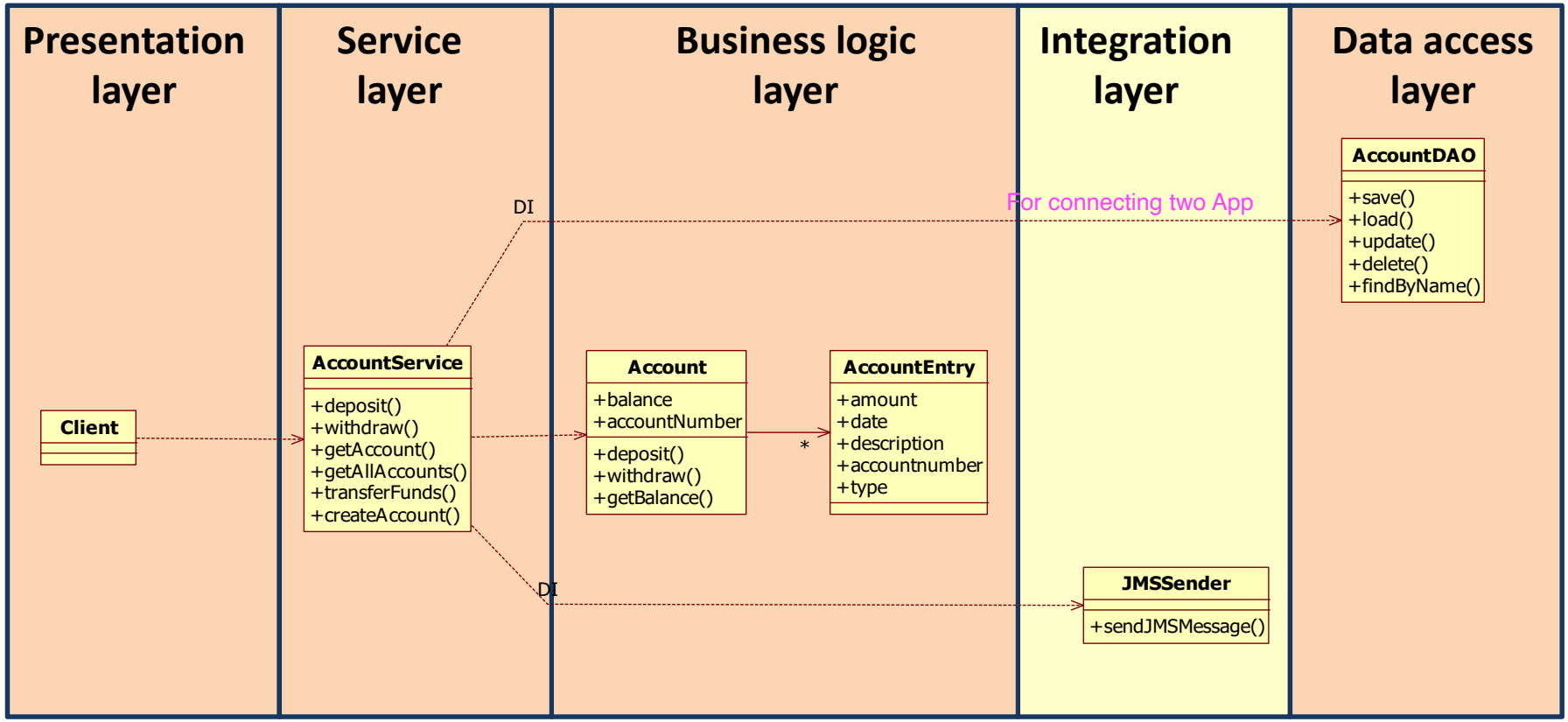
Increase Reliablility



- Replica Set – two or more copies
- “Self-healing” shard
- Addresses many concerns:
  - High Availability
  - Disaster Recovery
  - Maintenance

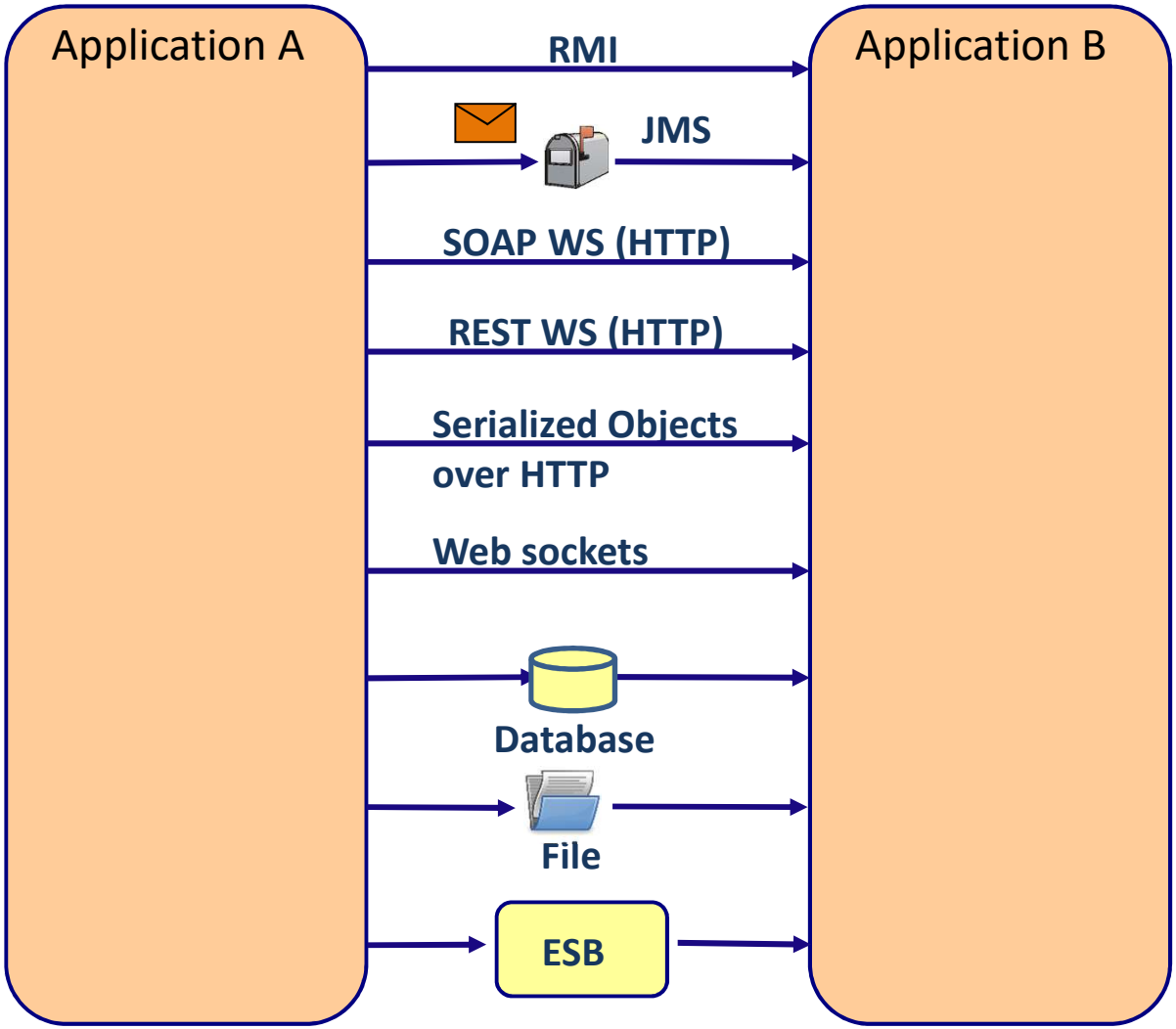
Main point

- In distributed systems one can have only 2 of the following aspects:
  - Availability
  - Strict consistent
  - Partition tolerance
- The unified field is the field of all possibilities.



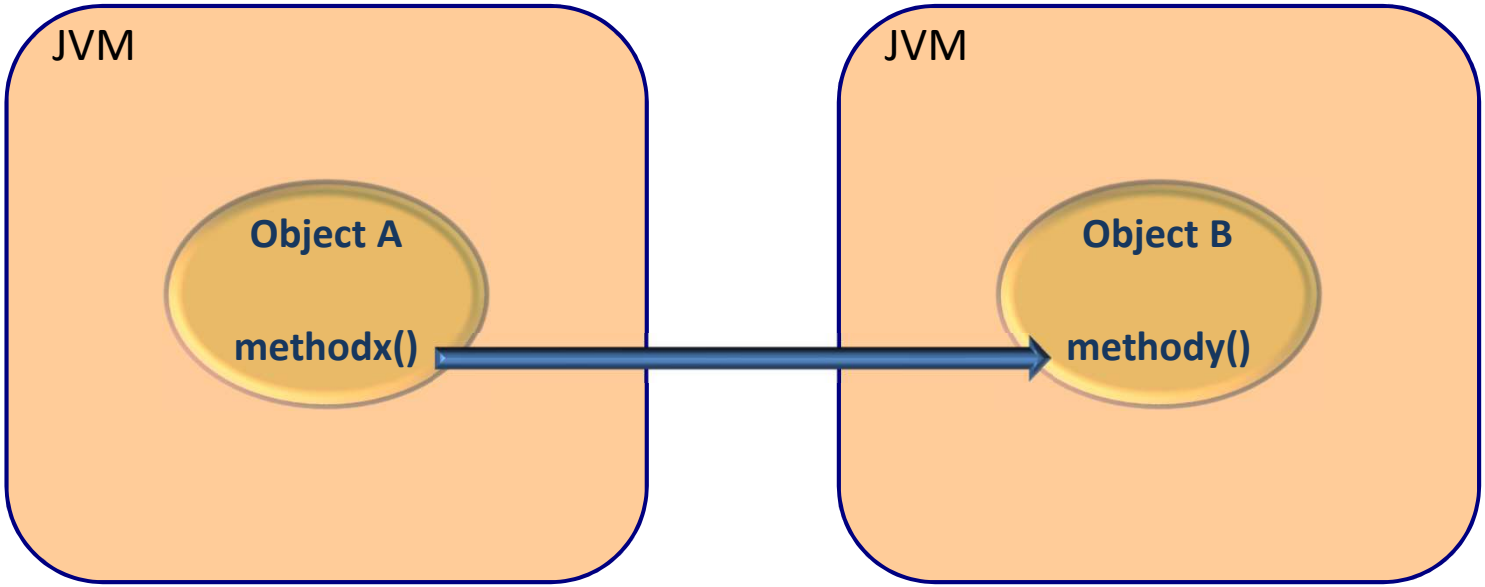
INTEGRATION LAYER

Integration possibilities



RMI

- An object calls a method of another object that lives in a different virtual machine.



Characteristics of RMI

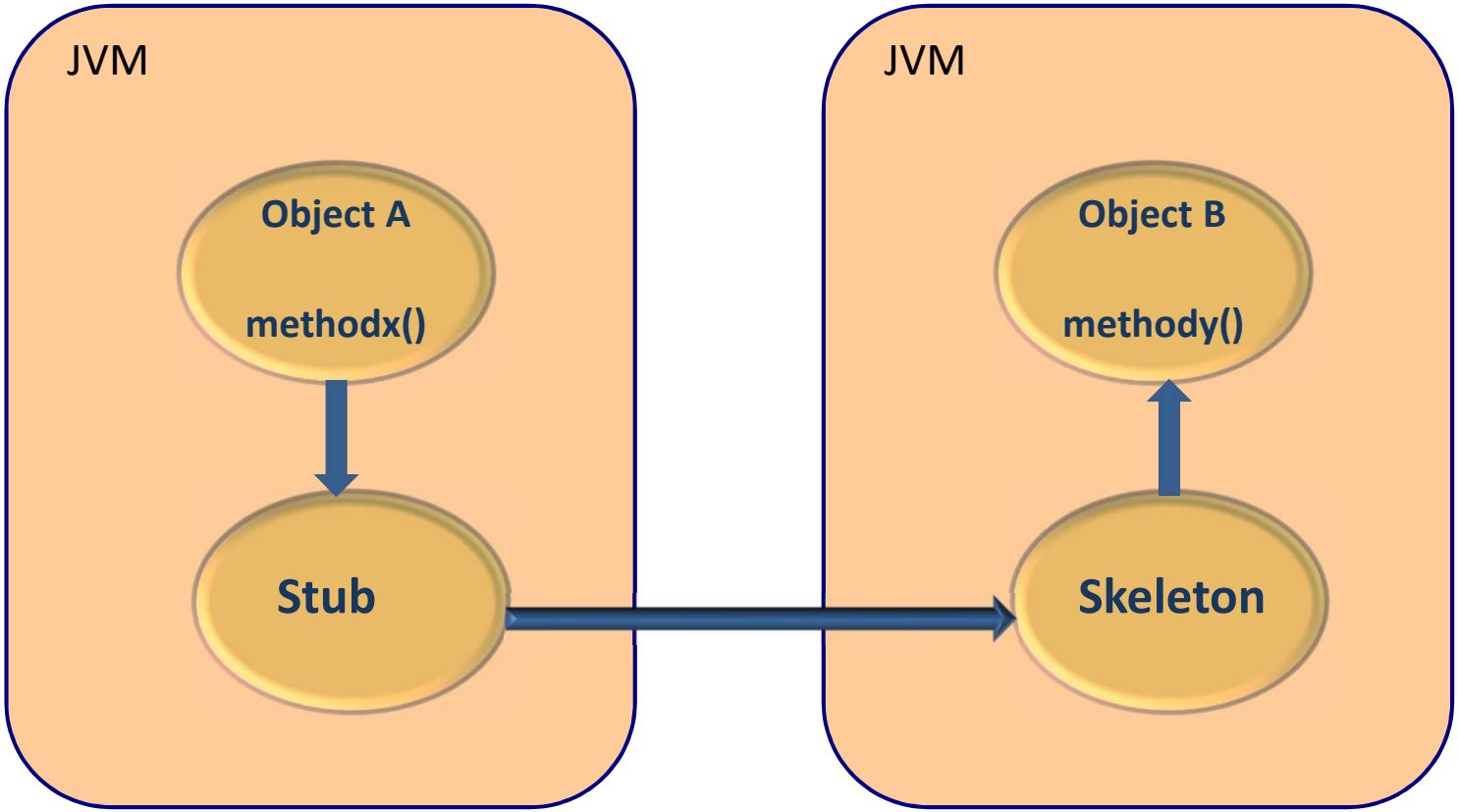
- Synchronous
  - The calling object has to wait until the remote method call returns
- Call by value

In virtual machine like Java, every thing is called by reference

  - If the remote method needs other objects as parameters, these parameter objects will be serialized and will be sent to the remote object.
  - All associated object will also be serialized.

Stub and skeleton

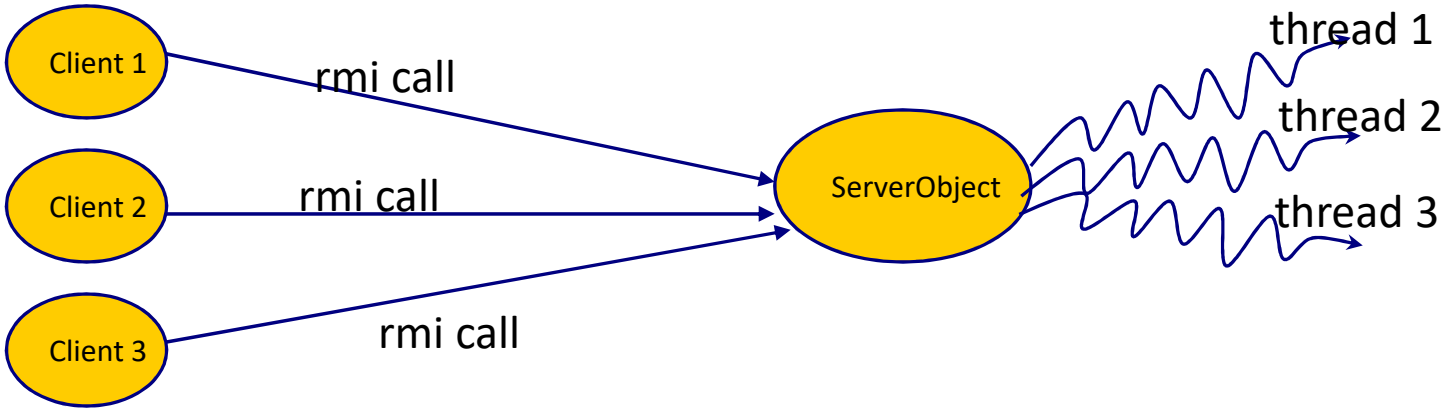
For serialize & deserialize



RMI and concurrency

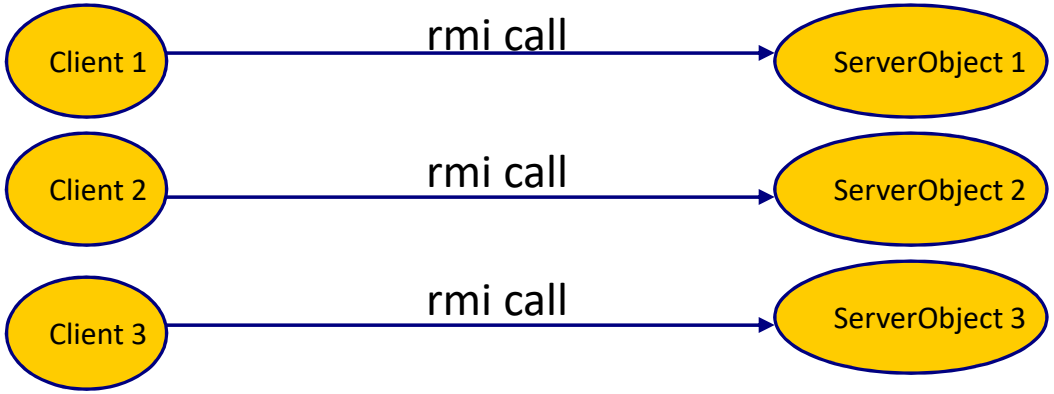
On server side we need either multi threading or pooling

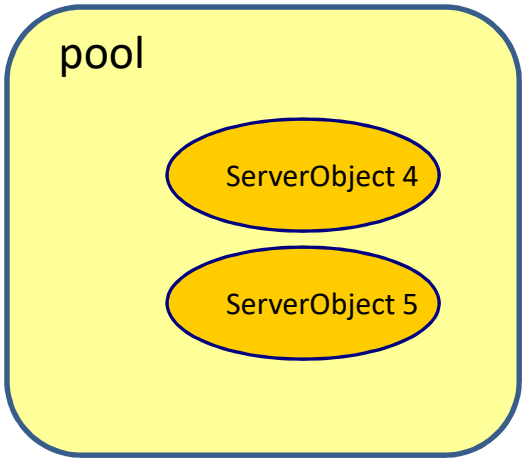
- Every remote method call executes in its own thread



Should be thread safe => should be stateless

- Another option: pooling





Thread safety

- A method is not thread-safe if it writes to instance variables (or calls other non thread-safe methods).
- Example:

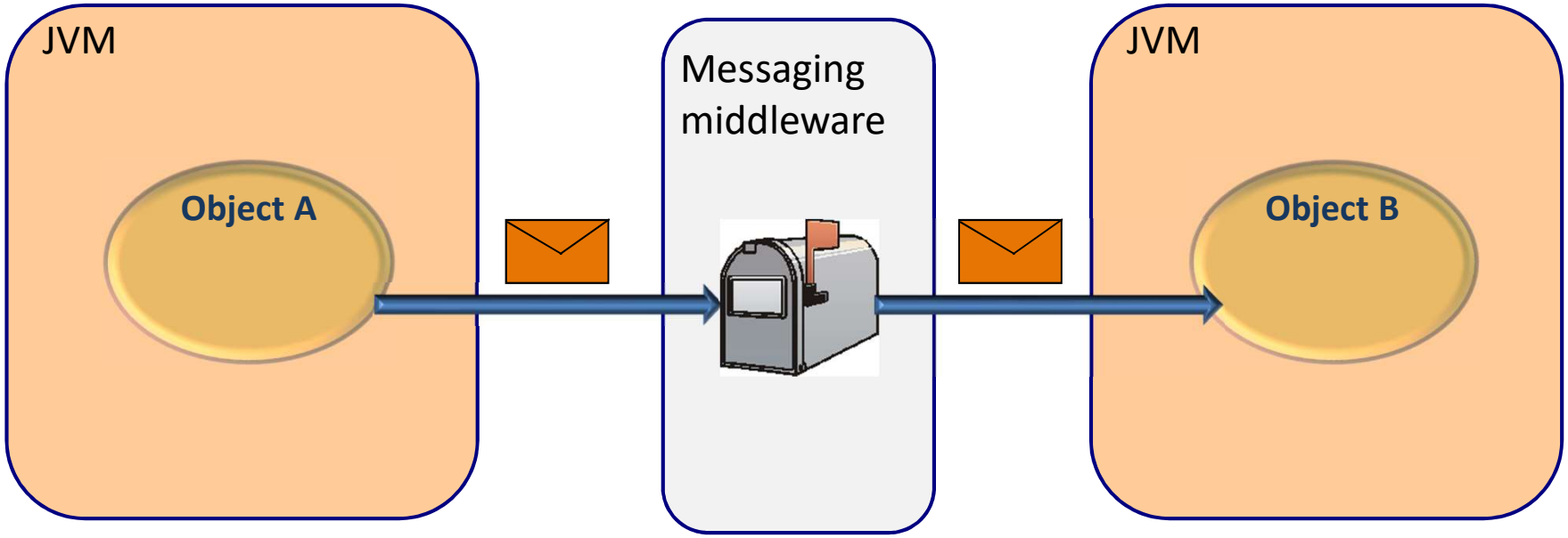
```
public class Calculator {
  private int currentValue=0;

  public int add (int value){
    currentValue=currentValue+value;
    return currentValue;
  }
  public int subtract (int value){
    currentValue=currentValue-value;
    return currentValue;
  }
}
```

State

The instance variable currentValue is changed

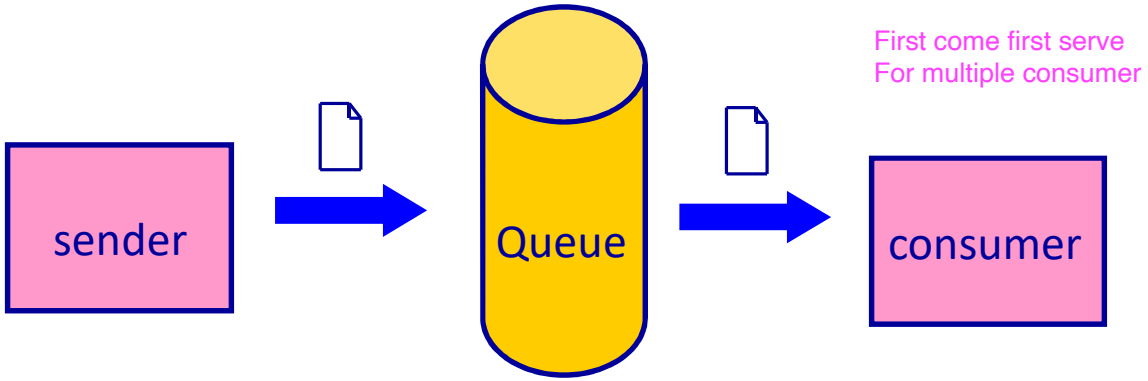
The instance variable currentValue is changed

Java Message Service (JMS)



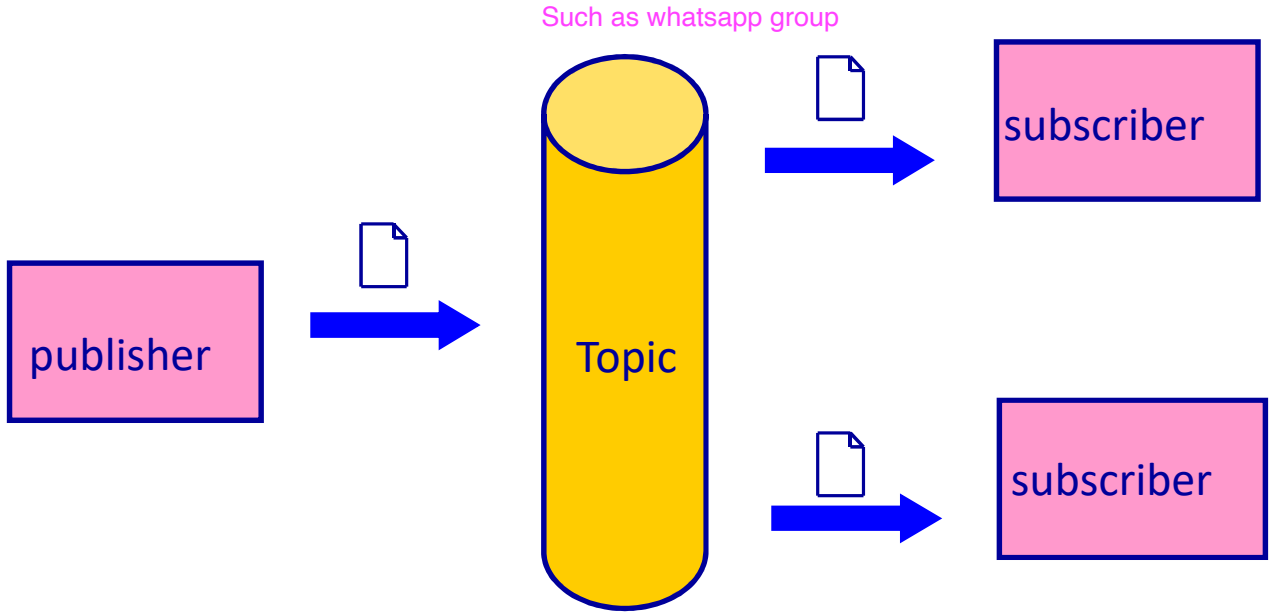
Point-To-Point (PTP)

- A dedicated consumer per Queue message



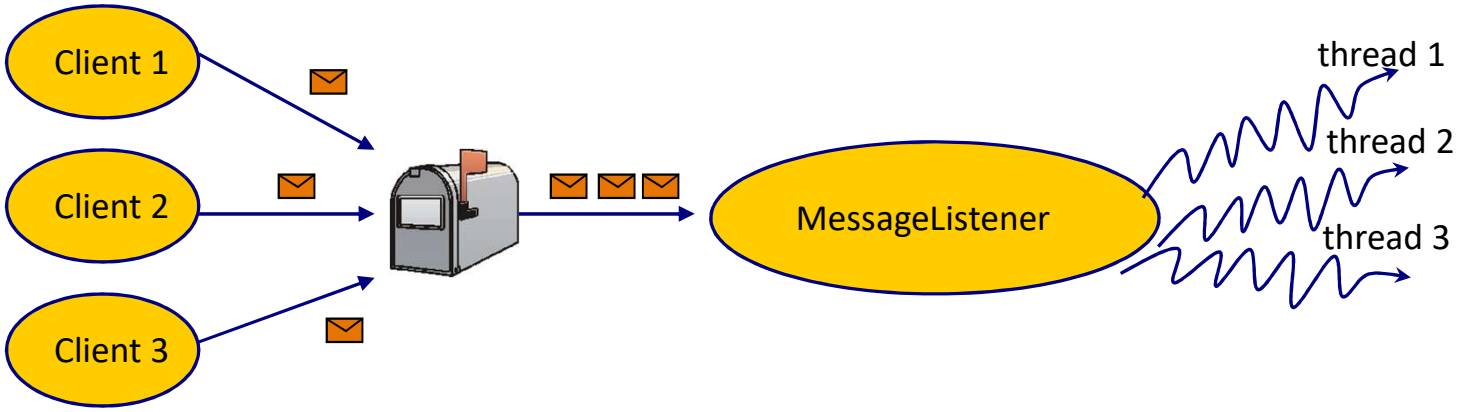
Publish-Subscribe (Pub-Sub)

- A message channel can have more than one '*consumer*'
  - Ideal for broadcasting

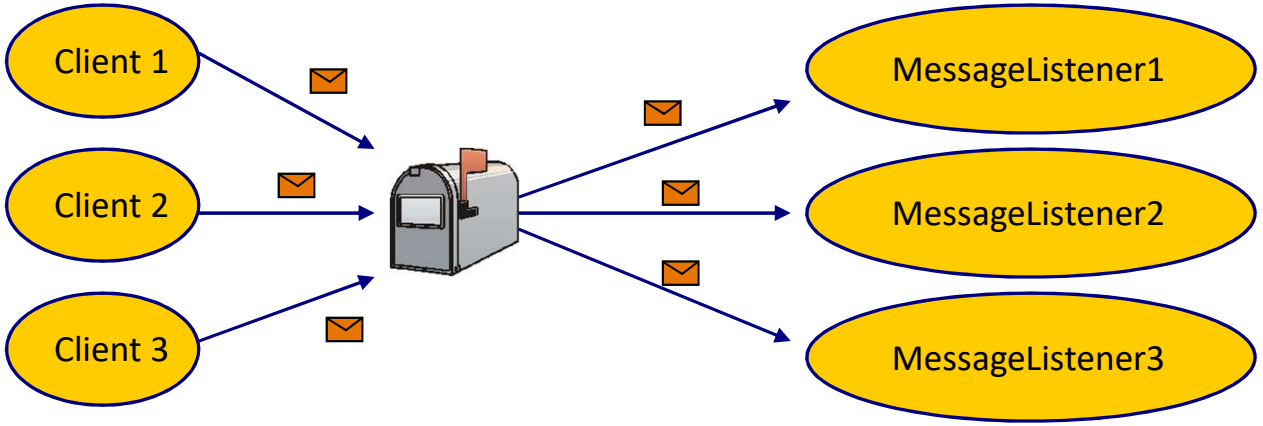


JMS and concurrency

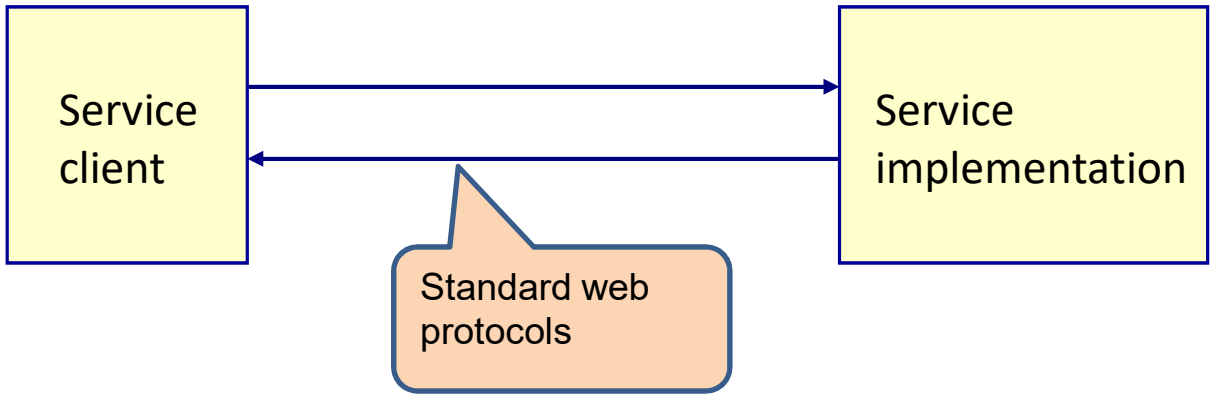
- Every MessageListener method executes in its own thread



- Another option: pooling



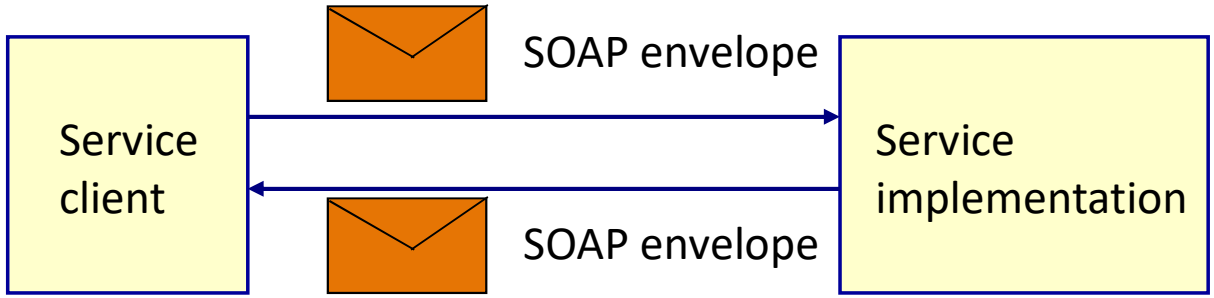
What is a Web Service?



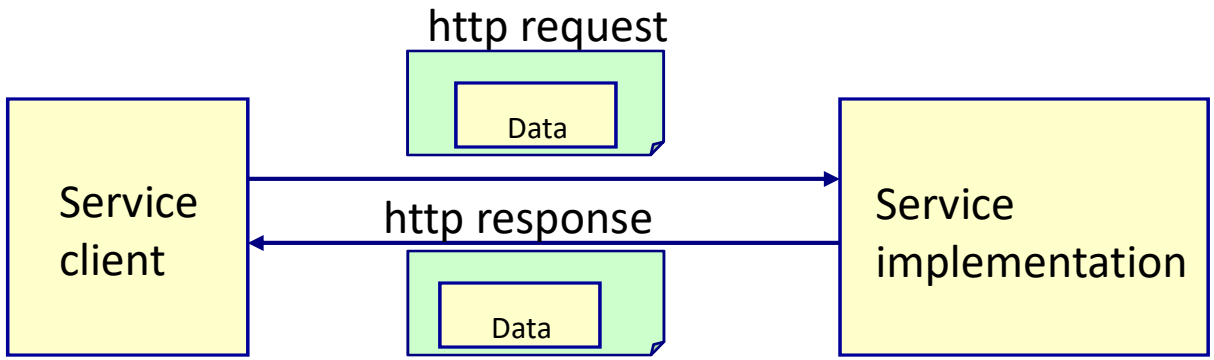
- A web service offers functionality that can be called by other clients using standard web protocols (SOAP, XML, HTTP)

Types of Web Services

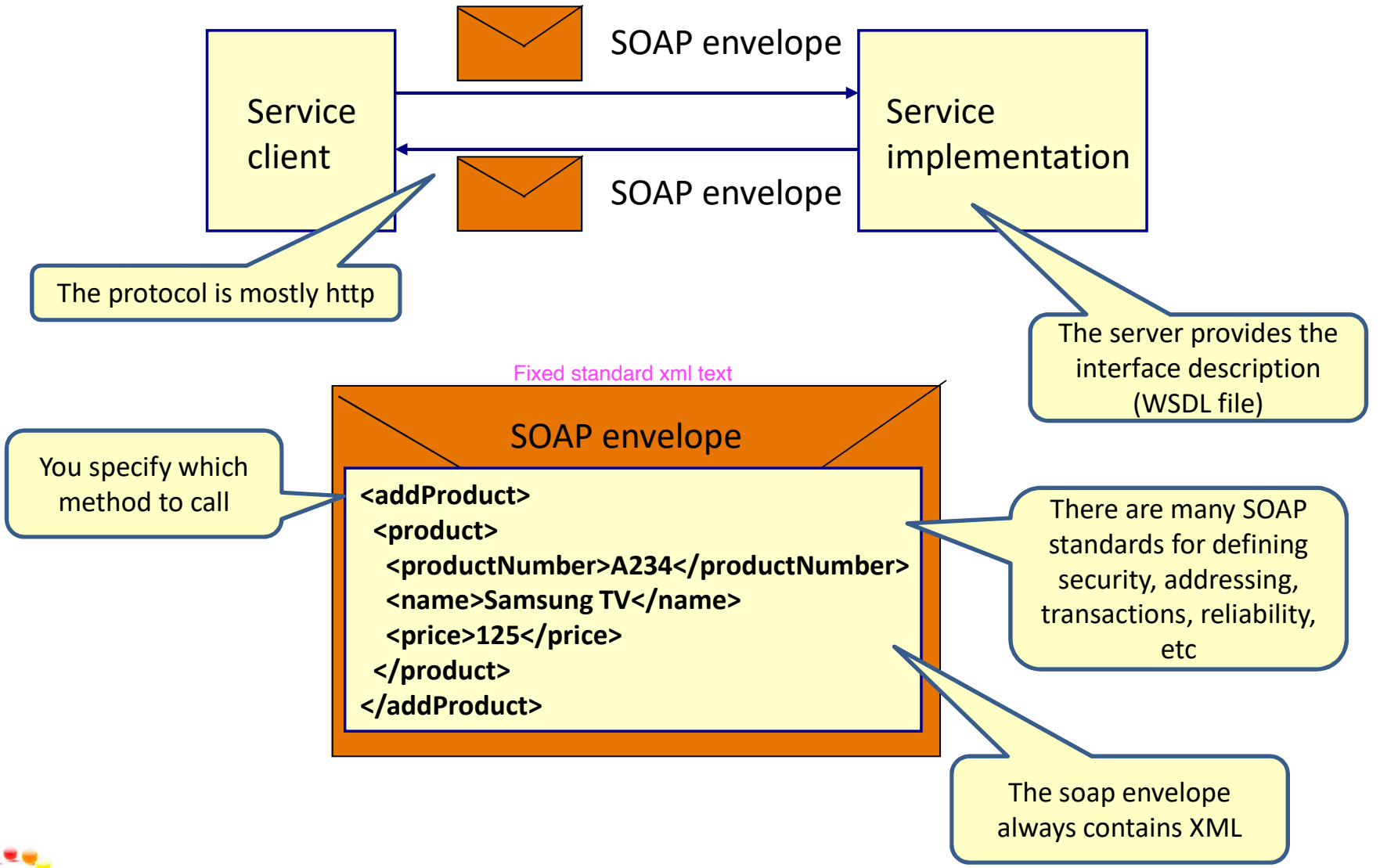
- SOAP



- REST

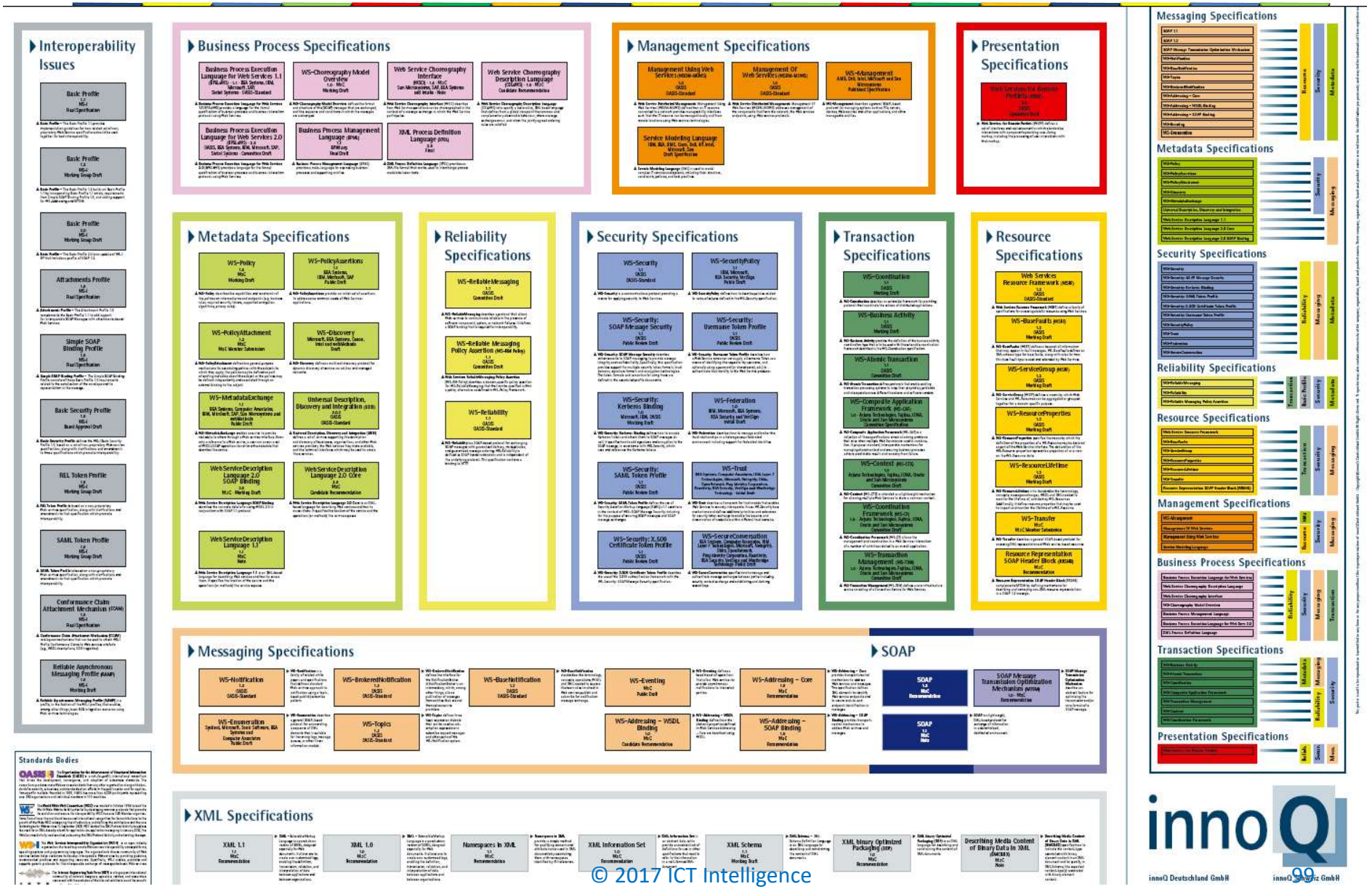


Simple Object Access Protocol (SOAP)

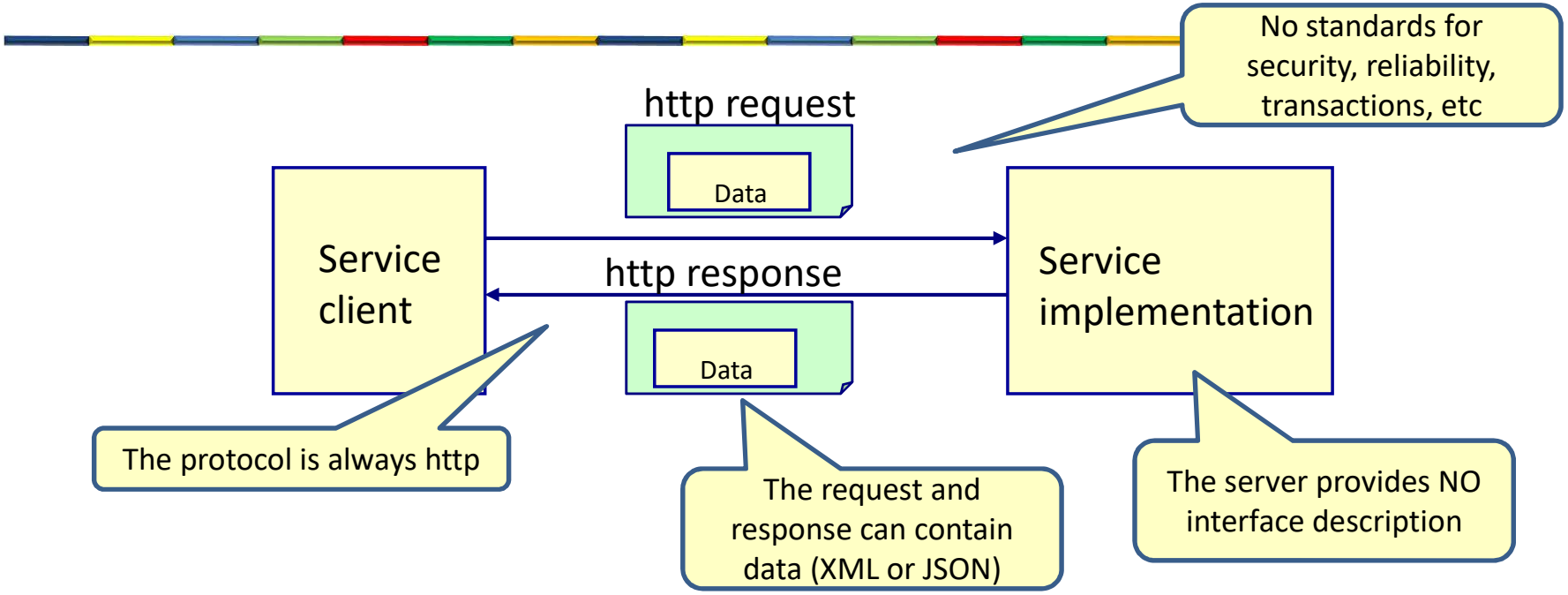


SOAP example

SOAP standards

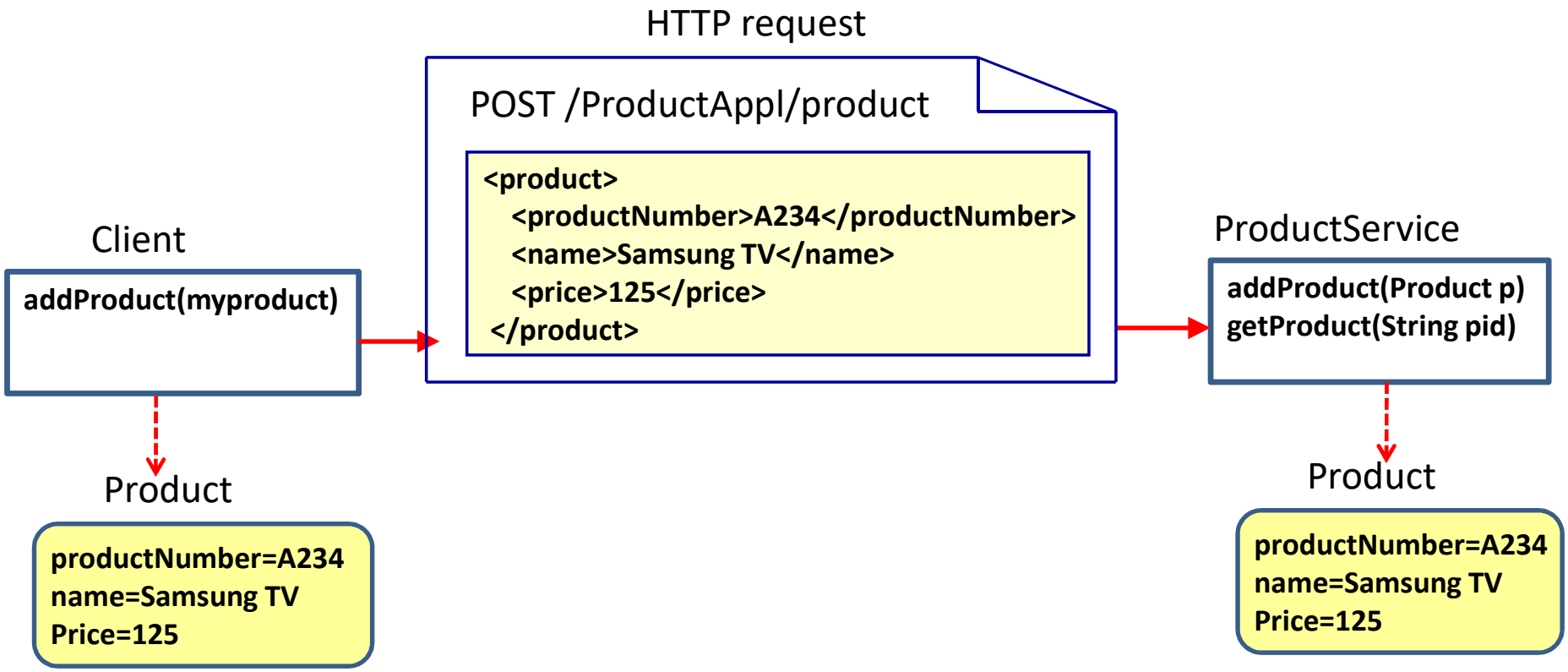


RESTful Web Services



- Data in HTTP messages
  - GET message for retrieving data
  - POST message for creating data
  - PUT message for updating data
  - DELETE message for deleting data

REST example

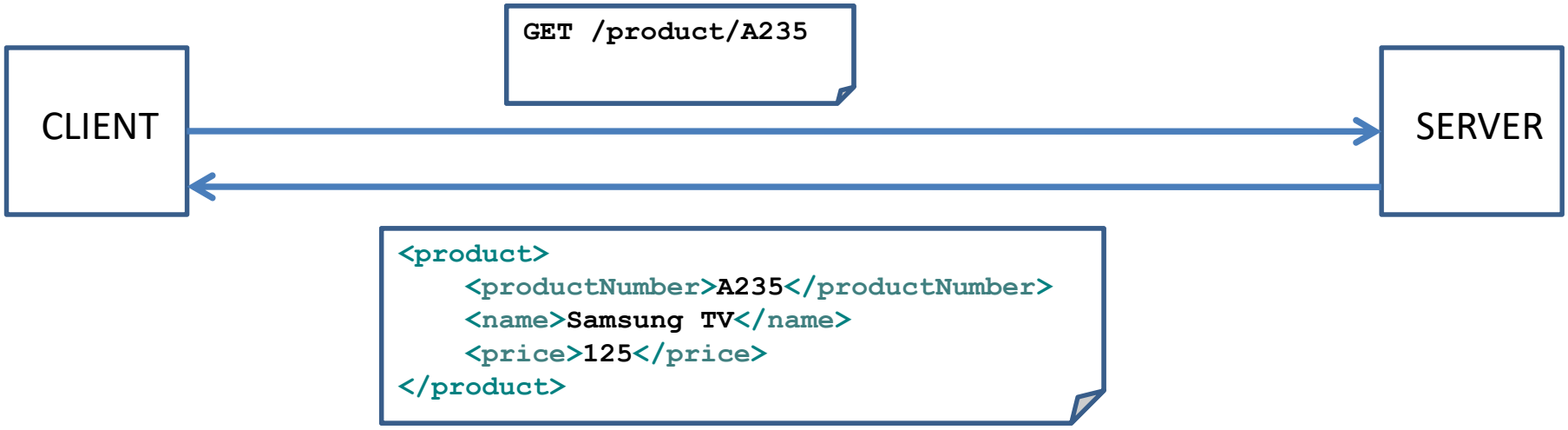


POST method using XML

We don't specify the method here

```
POST /product

<product>
    <productNumber>A235</productNumber>
    <name>Samsung TV</name>
    <price>125</price>
</product>
```

GET method using XML

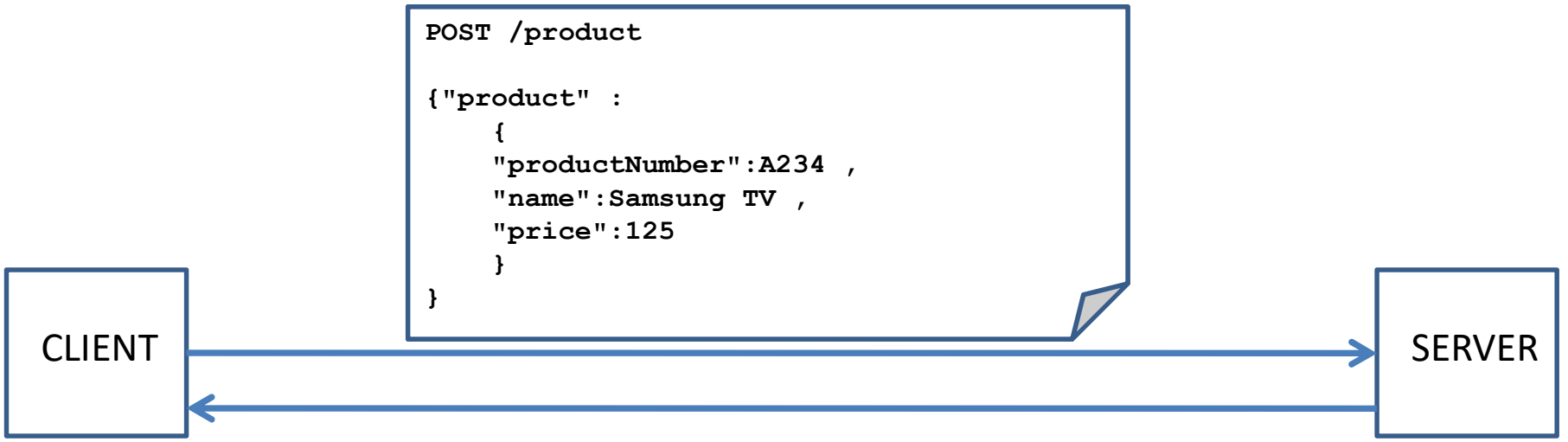


XML vs. JSON

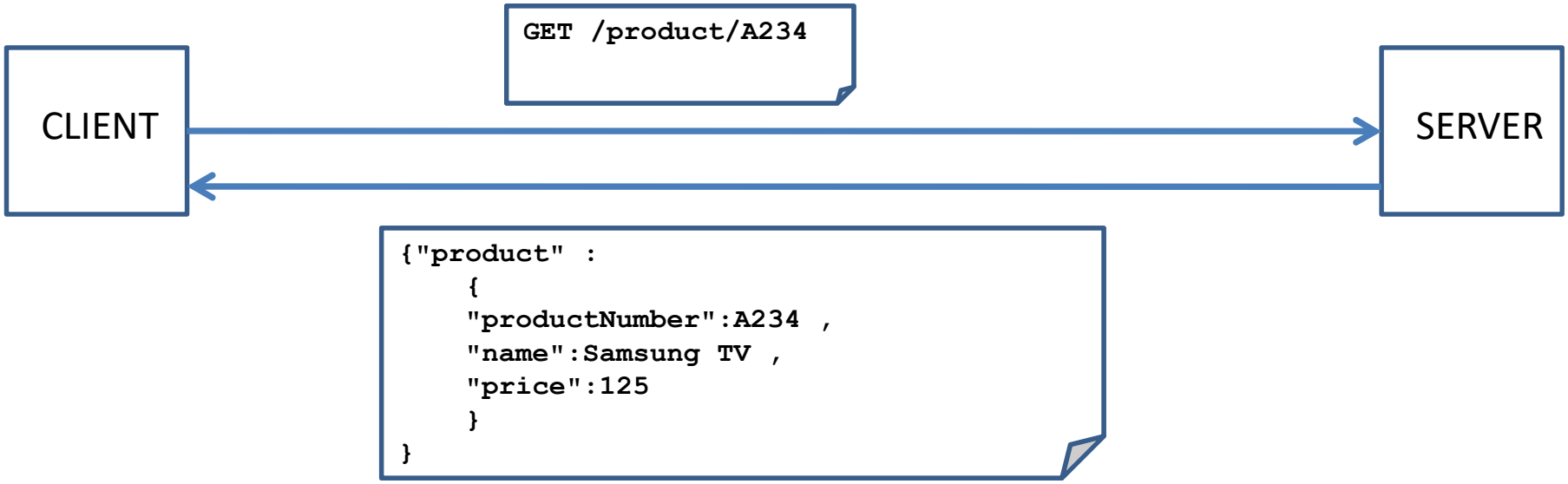
```xml
<empinfo>
  <employees>
    <employee>
      <name>Scott Philip</name>
      <salary>£44k</salary>
      <age>27</age>
    </employee>
    <employee>
      <name>Tim Henn</name>
      <salary>£40k</salary>
      <age>27</age>
    </employee>
    <employee>
      <name>Long yong</name>
      <salary>£40k</salary>
      <age>28</age>
    </employee>
  </employees>
</empinfo>
```

```json
{ "empinfo" :
    {
        "employees" : [
        {
          "name" : "Scott Philip",
          "salary" : £44k,
          "age" : 27,

        },

        {
          "name" : "Tim Henn",
          "salary" : £40k,
          "age" : 27,
        },

        {
         “name” : “Long Yong”,
         “salary” : £40k,
         “age” : 28,

        }
                  ]
    }
}
```

POST method using JSON

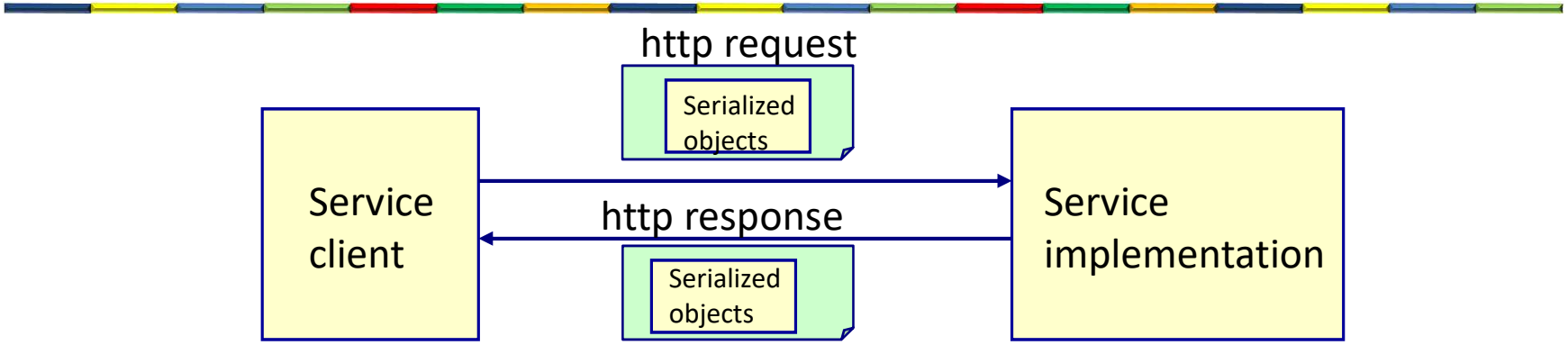


GET method using JSON



For both soap and rest, we should translate request to xml/Json and then translate to Java => slow
the advantage is than client and server can be in different languages

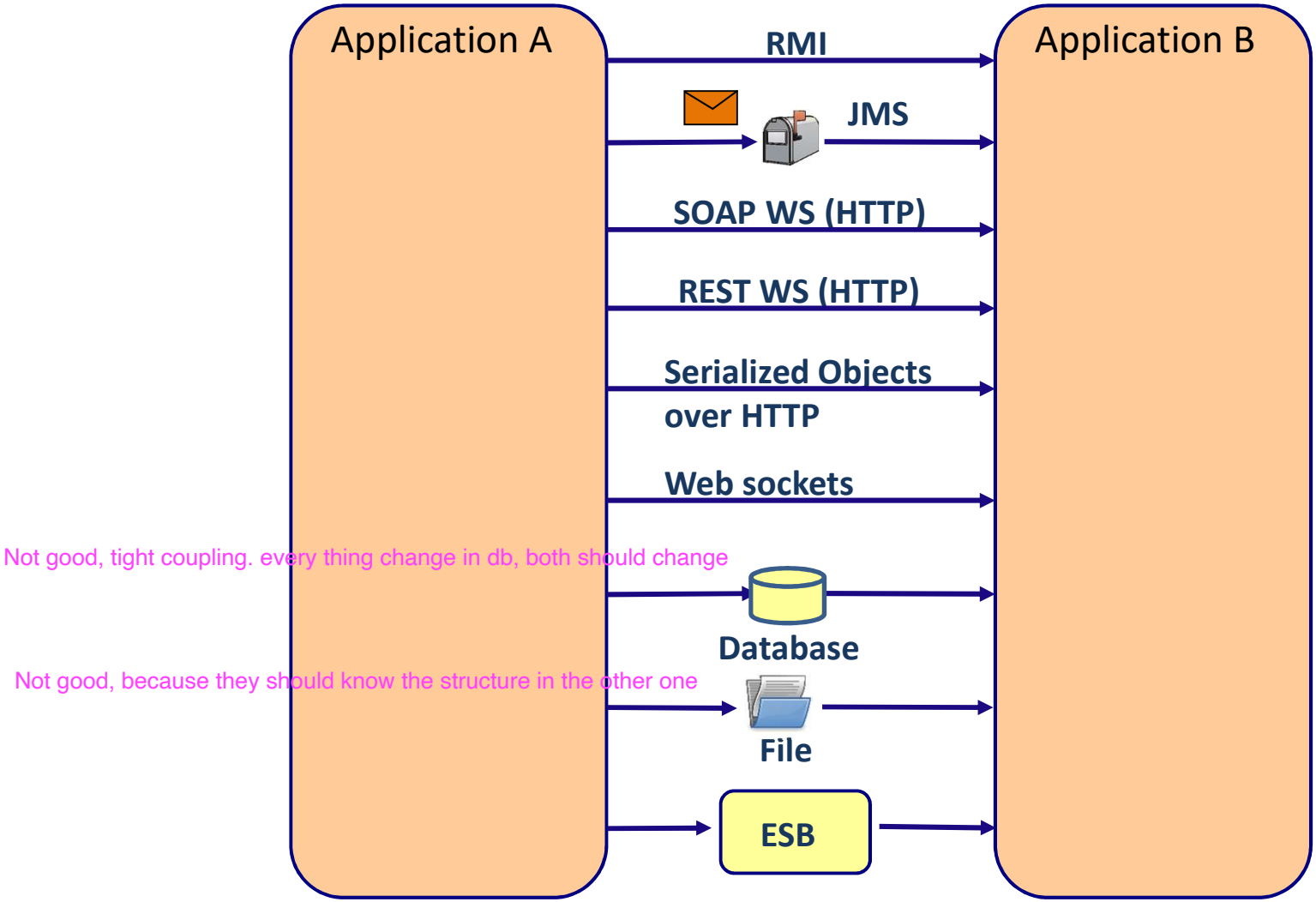
Serialized objects



If both are the same language serilize is faster =>
better to use RMI over Http

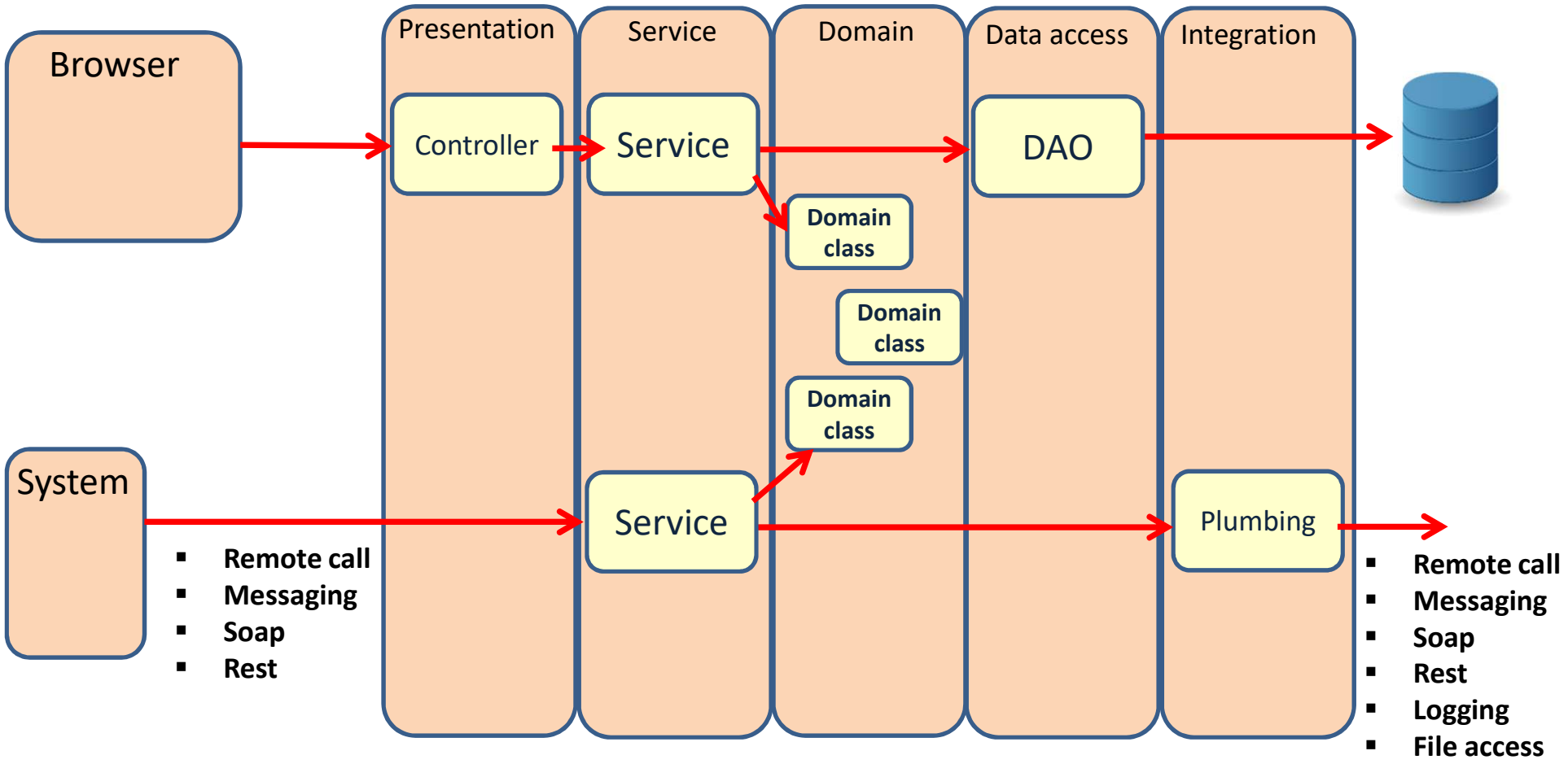
- If the client and server are both Java
- Sending serialized object is faster than sending XML
- Like RMI over HTTP

Integration possibilities



TYPICAL APPLICATION ARCHITECTURE

Application architecture

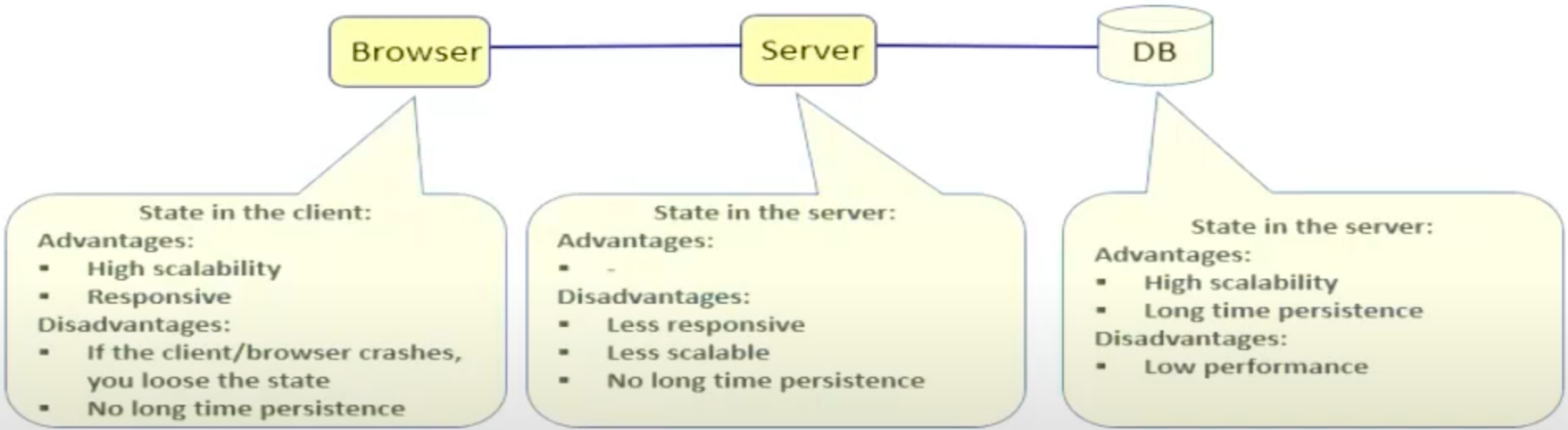


Connecting the parts of knowledge with the wholeness of knowledge

1. Layering is a powerful technique to separate different aspects of a system
2. The service class is the connection point between the different layers

3. **Transcendental consciousness** is the direct experience of pure consciousness, the unified field of all the laws of nature.
4. **Wholeness moving within itself:** In unity consciousness, one appreciates the inherent underlying unity that underlies all the diversity of creation.

Where to store state?



Integration techniques

All the security and multi-threading should do yourself

technique	advantage	disadvantage	When to use?
RMI It's a protocol	Sync	Only Java to Java High coupling	Never
Messaging ✓	Buffer Low coupling Asynchronous	You need messaging middleware	Between applications within the same organization When you need asynchronous requests
SOAP	Standards for security, transactions, etc. Standard for interface description Sync	Complex	When you need standards
REST ✓	Simple Sync	No standards for security, transactions, etc No standard for interface description	Everywhere you need synchronous requests
Serialized objects over HTTP	Simpler as RMI Sync Webcontainer functionality Security & Multi threading is built in container	Only Java to Java High coupling	For fast Java to Java integration between applications managed by the same project
Database integration	Simple Always win	High coupling	Never
File based integration		High coupling	If you have to communicate large files

Distributed systems

- Advantages
  - More resources available (CPU's, memory, ports, etc)

Martin Fowler law First low of distribution: do not distribute

- Disadvantages
  - Very complex
  - Lots of thinks that can go wrong
  - Remote calls are extremely slow
  - Security is more complex
  - Transactions are more complex

need serialize / deserialize