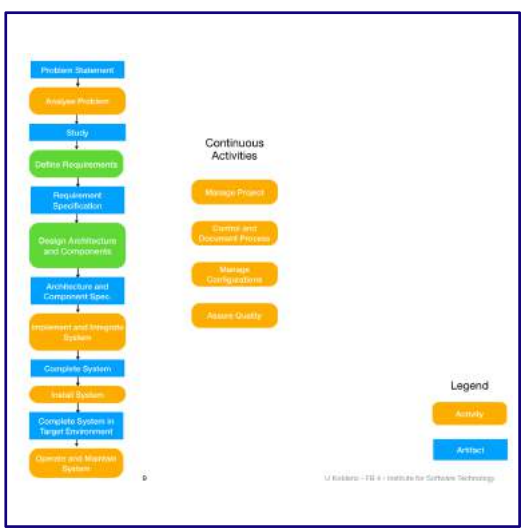

Topic Order
Intro
Architecture
Modeling
Persistence
Communication
Security
Summary
Not in exam
Security
Attacks / Problems
OWASP
CVE
Countermeasures
Access Control
Communication
Networks
Protocols
IP, TCP, UDP
HTTP(S), TLS
Data Representation
HTML, CSS
XML, DOM
JSON
Server Side Components
Data Storage
Web Server
Load Balancer
Web Services
Client Side Components
Web Browser
JS
WS Clients
Persistence
Data Models
Relational
MariaDB
Graphs
Neo4J
Document Stores
CouchDB
Distributed Data
CAP Theorem
Engineering Web and Data-intensive Systems
Web Applications
Characteristics
Types of WAs
Quality Goals
Performance
Availability
Usability / HCI
Maintainability
Scalability
Correctness
Robustness
Architecture
Styles
Client-Server
Layered
SOA
REST
Patterns
MVC
DAO
DTO
Facade
Modeling
Structural
Domain Model
Content Model
Behavioral
Use Cases
Process Model
Navigation Model
Context
User Model
Languages
UML
UWE
Model Transformation

01 - Introduction

- 5 steps in engineering — a) gain background knowl.
  b) understand & formulate the problem
  c) design solution
  d) Realize & implement the sol.
  e) verify the sol.

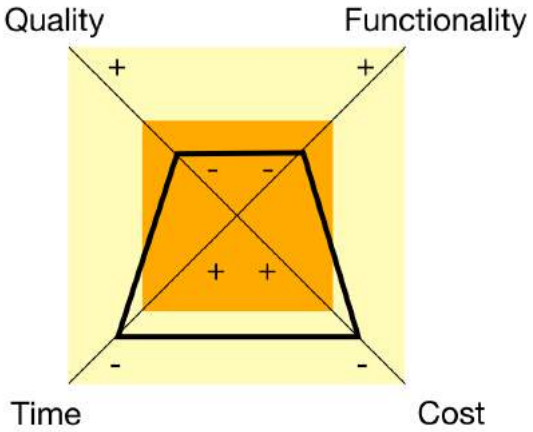
- Soft. engeneering → construction, usage & enhc.
- Soft. System → set of progeramm with artifacts source code + artifacts to use & operate system.

- Soft. lifecycle. →



- Soft. Quality → correct
  reliable
  robust
  scalable
  maintainable
  extensible
  reusable etc. . . .

- Depict the dependency in Devil Square by Harry Sneed.



Orange – productivity
→ low cost & time reduces quality + functionality.
(Black trapezoid)

- Security → challenges :- 24/7 online
distributed
massive attack
high security
strict regulation (GDPR)

- **Web Systems**

A web application is soft. system based on standards of world wide web → resources, contents through user interface & web browser.

Categorizing Web Systems

1) • **Documents** which were static, HTML,
updated manually

- Interactive Web Apps → dynamic info
  CGI, gateways, generated by scripts of external data sources.

2) • Transactional → update content (database read & write)
  eg. bank
  • Workflow based → business process, structed collaborations
  eg. e-commerce

3) Collaborative → grp of ppl can work together on artifacts, database in real time.
  synchronization
  eg. wiki

Social Web → collaboration non-anonymous personal (socialy connected ppl)
  eg. facebook.

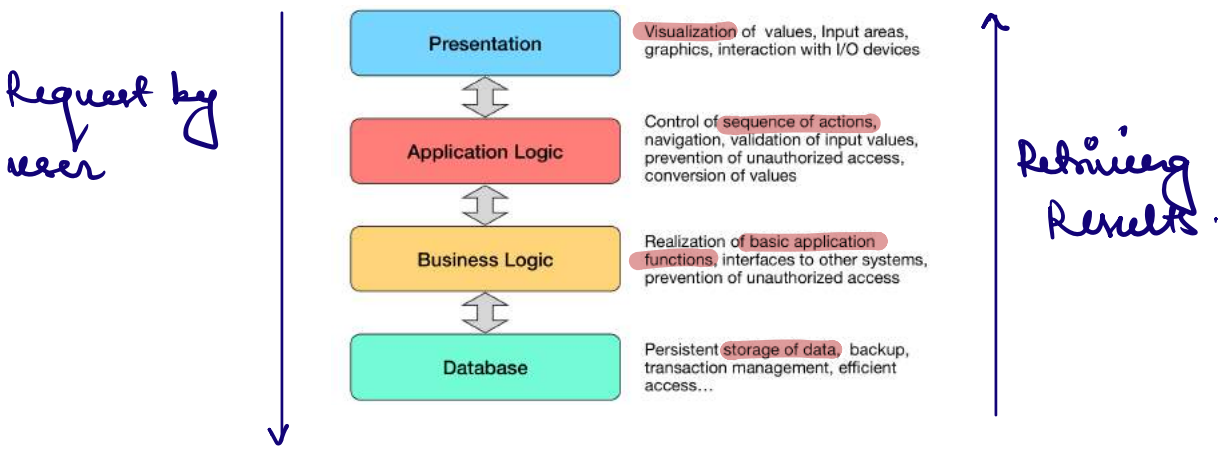
4) Portal Oriented → info. of heterogeneous sources.
  eg. search engines

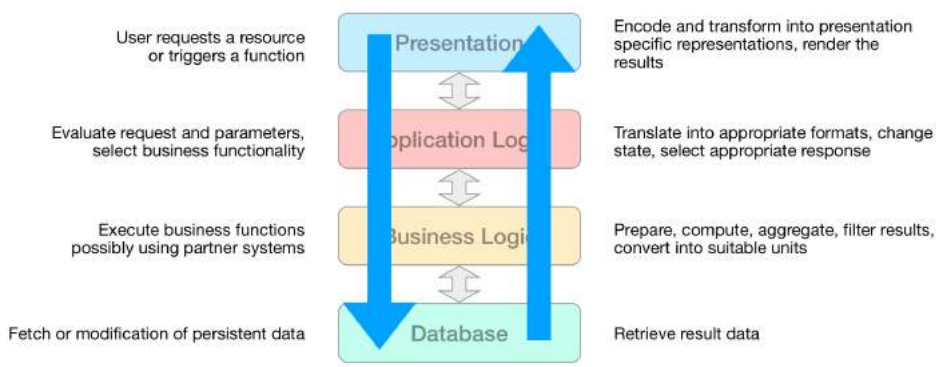
Ubiquitous → customized services on devices
  eg. device-idependent apps.

* Age, location controlled content of app.

5) Semantic Web → desire to make more of data. info is to gain or store knowledge. lags. Ontologies based knowledge (hierarchy) m/c readable.

High level Architecture. (4 layer Architecture)





Data Intensive Systems

- high complexicity of DS
- big amount of data.

EWADIS deals with :— (logical & implementation models)

- diff. data models : relational, document/key value & graph
- foundation of data distribution & reliability issue.

Watch lecture again

Lecture 2 Software Architecture

- s/w lifecycle

Req. specification → Design or Architect Component → Component spec.

s/w Architect → component & connectors.

- decomp system into component on relations. & relation to other environment.

Rules & policies of designing component.

s/w Architecture are documented in diagram.

- layer Architecture. (which we studied earlier)
- Java → Tiers, machines
- Business entities. → server and m/c communicate

web application architecture → check on google. or educative.io

- Archi. diagram is described by Viewpoint which is kind of a legend. so tells components and relations and rules. → of certain class of digram. (architecture diagram)
- Views → are the diagrams which corresponds on viewpoint

eg. In arch. of building you can have diff. views like how building look outside → walls, windows, roof — 1 viewpoint
- electric system → another viewpoint
- usage of room → viewpoint (types of room etc)

- Styles → patterns which can be reused or adapted. they have reason of how comp. communicate.

Scenario → what are expected changes in the system! (System & environment)

Examples for Components and Relations

- **Components**
  - (complete) System, Subsystem
  - Module, Package, Class, Component
  - Layer, Tier
  - Procedure, Function, Method
  - Segment, Paragraph, Refinement,
  - File, Relation, (Database-)Table, Directory
  - Global Data Structure, Object, Variable
  - Partner System
  - Physical Machine
  - Network Device
  -
- **Relations**
  - A contains B
  - A realizes B
  - A uses B
  - A calls B
  - A follows B
  - A instructs B
  - A sends message to B
  - A delivers data to B
  - ...

- logical, development, process, physical (View)

- logical view → designs buisness objects. (object model)
  we use E-R diagram

- process view → is abt concurrency & synchronization.
  data consistency, parallel structure etc.

- Physical view → how system is deployed. distributed or centralized.

- Development view → organization, structure subsystem.

- production environment
- Use case or scenario.

examples of Viewpoints →

- Structural Perspective
  - Components: subsystem, package, class, method
  - Relations: contains
- Usage Perspective
  - Components: component, class, method
  - Relations: uses, calls
- Data Flow Perspective
  - Components: activity, process, data object
  - Relations: reads, writes

Widerange on details or granularity →

More detailed level → what system is meant for ?

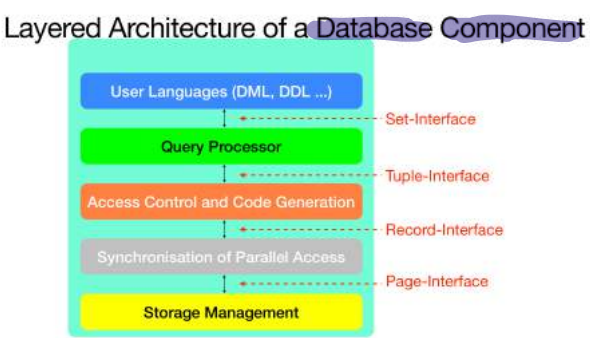
↳ requires more detail.

when we have many detail → we can zoom in for more detail.

* This is recursive decomposition

Coarse → so in 4 layer we can do more detailed.

further zoomed in for more fine layering



Software Quality → external quality perceived by user. (Usability)
internal quality ability to evolve or maintaining system. (Controllability)

Usability → Reliability (how to depend on system)
Correctness
Robustness → tolerant
Resilience → reacts on powerfail
- ease of use
- efficiency

Controllability →
- Maintainability → managed, evolved
  - Verifiability → system is correct
  - flexibility → new demands
  - understandability → clear is the design.
- Portability → how easy to switch.
- Scalability → how to extend. on demand.

Relation of SW Archi & S/w Quality

- s/w Arch is <u>not</u> free
- Best practices, so we have standard frameworks etc. (s/w Arch).
- Decision does not depend on functionality. It is driven by non-functional requirement.

- Suitable sol to emerge & evolve.
  # styles & patterns which are transferable are reusability.

Principles for good Architecture.

1) Simplicity → if you have options choose the simple one.
   (KISS → keep it small & simple)
   It supports maintainability & understandibity.

2) Analogy → similar things to be handled similarly. (uniformity)
   → more understandibity
   → easier conventions.

3) Completeness → Consider all possible cases even you need few.

- supports robustness.

or evidence

4) Verifiability → proof if goals can be achieved.
i.e. testing.
- Reliability & correctness.

5) Least surprise → should react in meaningful way.

6) Decomposition → distribute the work.

7) Encapsulation → flexible, design as black-box.
clear interface.

8) Locality → things kept together for better maintainance.

9) Separation of Concerns → kept separately.

10) Abstraction → common properties are to be indentified

Architecture Styles

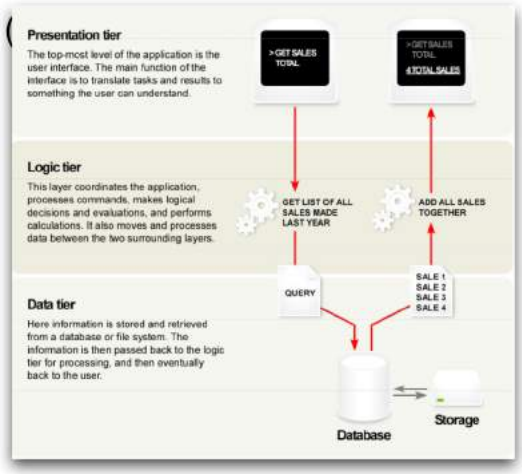
- Styles basic structure of architecture diagram.
- nodes & edges (components & connectors).

1) Pipe - Filter Style → Data flow perceptive & linear structure.

o linear set of actions

2) Layered Style → Usage
Hierarchical or tree-shaped.

Client layer
↓
Presentation
↓
logic
↓
Data
↓
Service Provide



3) Repository style → Active/inactive repository connected with components.

* Star shaped.

Architectural Patterns → are system specific blueprints.

- transactional
- portal-oriented
- collaborative/social

For layered Architecture (what can be pattern)
defining patterns for boundaries b/w the layers.

1) Decomposition → assigning applications to different layers.
more easy to understand and maintain.

2) Encapsulation → functional interfaces for services each layer provide to client. what services do? you don't know how?
- flexibility
- SOA (Service Oriented Arch)
- Reuse
- Verifiability → tested independently
- distributing the work to diff. team.

3) locality & separation of concerns → meaningful grouping of functions at different levels.

Services are provided by lower layer to upper layer.

eg. application logic is service to presentation logic.

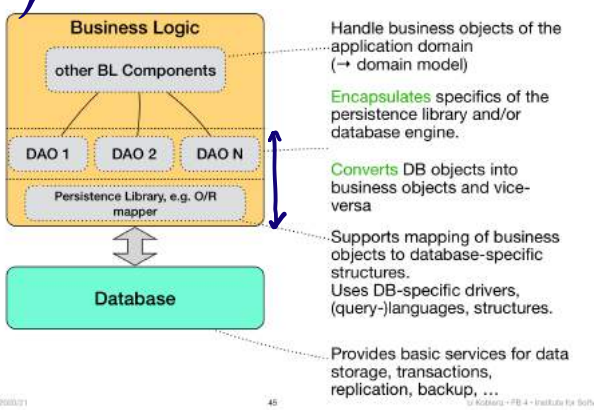
Interface definition → b/w layers of boundaries

Patterns b/w the boundaries of layers

1) Data Access Object (DOA) → slide 31 of Persistance

Data Mapper

- convert Database Objects into Business Objects (vice-versa)

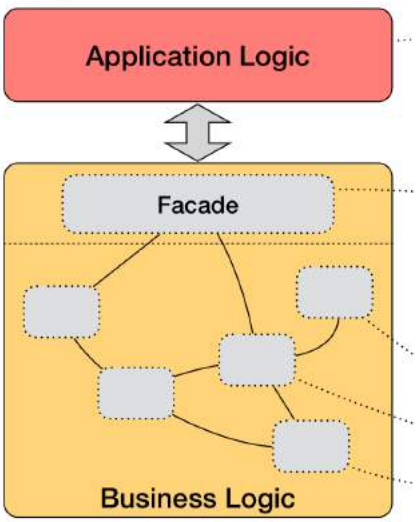


Read more abt DOA.

2) (Remote) Facade pattern

Boundary b/w Business logic & Application logic.

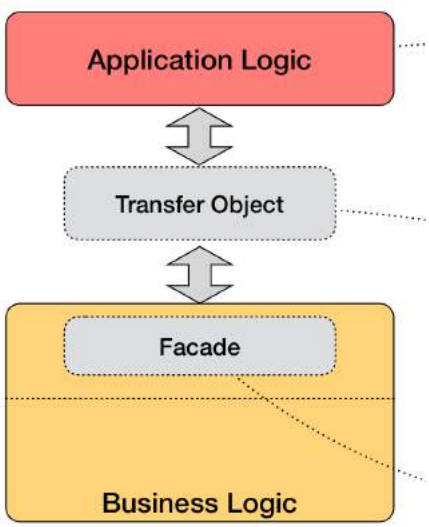
- it hides complexicity.
- It defines exported services or functionalities
- BL will have various components (subsystems)
- tasks → delegate to specific BL component.
  also group data & calls to minimize communication.



It is possible to have multiple facade to provide different protocol

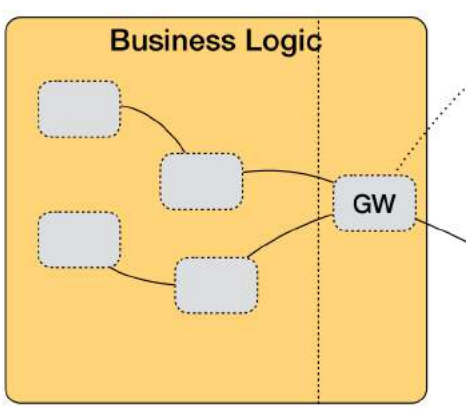
3) Data Transfer Object (DTO or TO)
B/w business logic & application logic

- Facade was responsible to aggregate data.
- Only purpose is to transfer data.
- we don't need transfer object for every business object. we can aggregate & use it



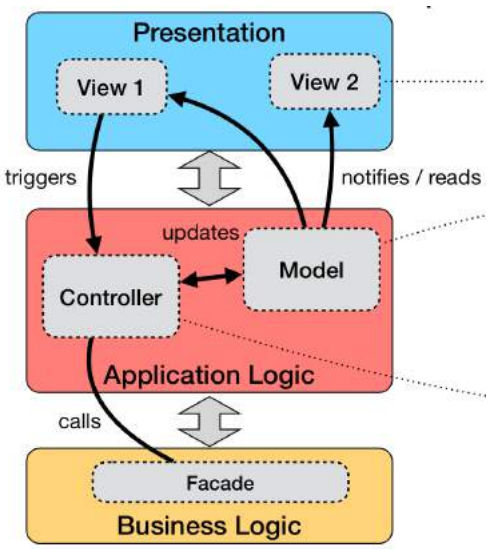
4) Gateways

access partnering systems.
uniform acess various service provider.



5) Model - View - Controller (MVC)

Study again later important.



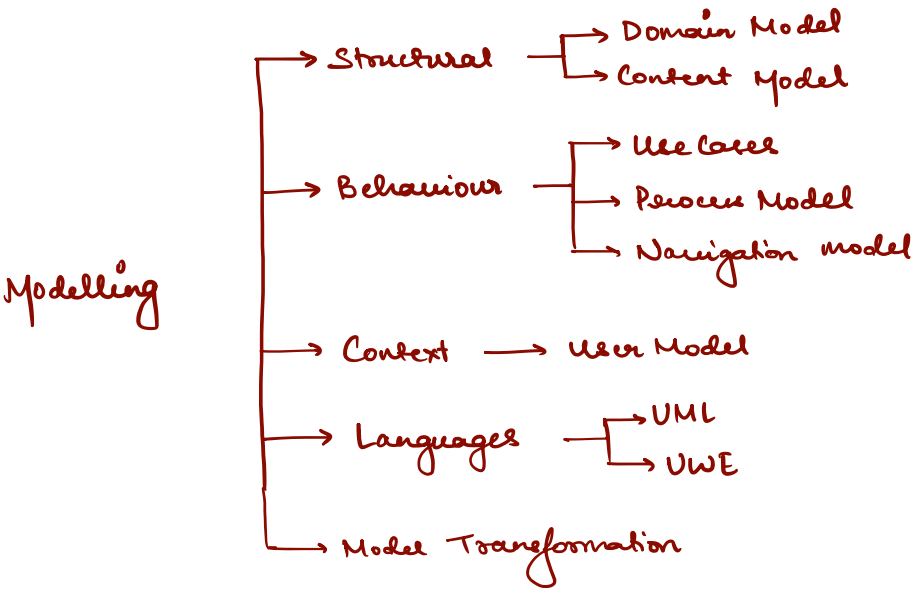
- Boundary of Presentation & Application logic.
- User interface components they trigger Controller.
- Controller modifies/updates model.
- Model is changed by interaction.
- Model notifies all views affected by that change.

Eg. Shopping Cart

- All views only communicate with controller.

* Can be same machine

Chapter 3 - Modelling (Part 1)



A model serves as an abstract of subject area. Mapping of real world entities into IT.

- leave out details of reality (omitting). simplify by non-relevant info.
- you decide your domain & relevance.

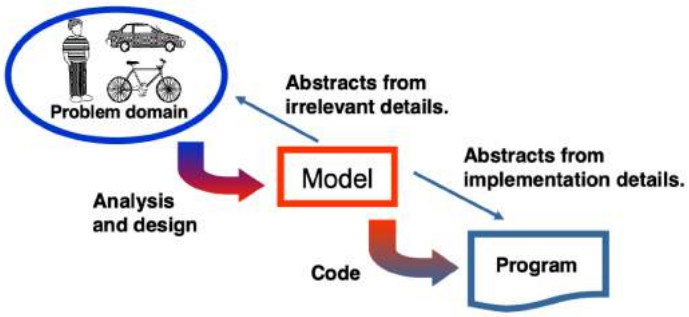
Purpose of models

- Reduce complexicity of real world
- communication → graphical notation to depict to non IT people (other stakeholders) eg use case diagram.
- Documentation → enhance your system with easier understand, descriptive.
- Specification → prescriptive, more detailed & formal ways.
- transformations → models to source codes

* Models have different granularity (level)
  → degree of formalization.
  → good semantic.

Model Based S/W Engineering :-

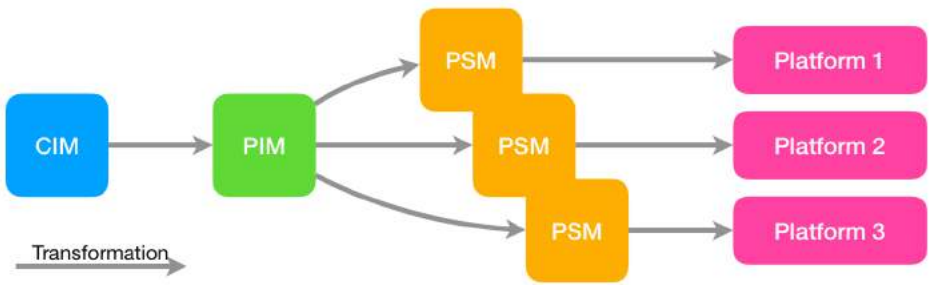
•



(MDE / MDA)
Model Driven Engineering or Model Driven Architecture doesn't have model for documentation or communicative role but plays a central role.

- Models are at the core and we have transformation the construction of system to derive in more detailed way completly or automatically.

Hierarchies of model



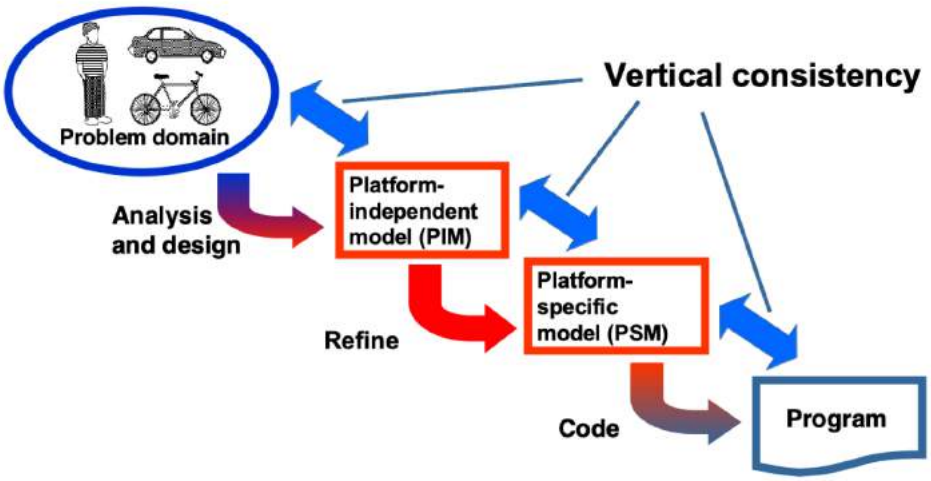
Computation Independet Model :- User perspective.
all process can be done without IT.
not dependent on any technology.

Platform Independent Model :- Design phase starts.
specific to computation.
You don't decide on programming language, technology or application.
more clos to IT solution.

Platform Specific Model :- Specific PL, databases, applications. more detailed.

- MDE approach is rarely used for complete system but parts of the system.

In MDA, we start with a problem domain, we generate platform independent model. PSM adds more detail with program.



It looks like waterfall but it has feedbacks on various layers.

Modeling Languages

- Various aspects of system requires various modeling language.

- Modeling language →
  1) Syntax → building blocks, which elements to use. which rules or constraints

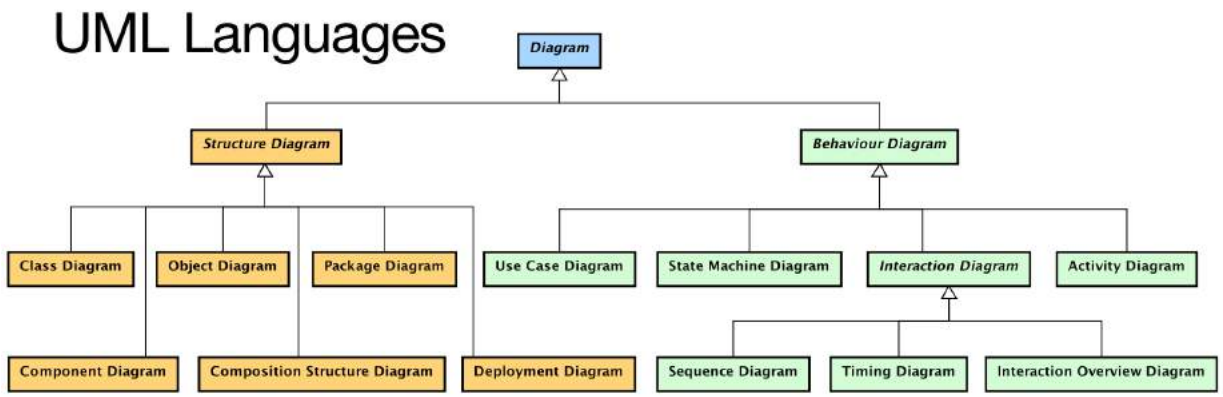
  2) Semantics → denotes set of objects of a class. Map semantic of modeling language to logic calculus for applying reasoning etc. to correctness.

3) Pragmatics → how can be language used in real world.

UML (Unified Modeling Language)

- describe s/w system on level of granularity close to system development part.

UML Language is divided into structural or Behaviour diagram.



Structure of Models

1) Coarse Grained → high level and user perspective
   - how business objects are present & linked.

2) Medium Granularity → architectural level & development perspective.
- UML component diagram.
- deployment of nodes or related connection b/w nodes.

3) Fine Grained (low-level) → classes or entities and internal components.

Behaviour of Models

- Coarse grained behaviour (high level)
  - different usecase for each role
  - sequence of actions
  - process.
- Fine Grained Behaviour (specification)
  - activities in process tasks
  - sequence of actions
  - timings → when actions take place.

Specific Models for Web Applications

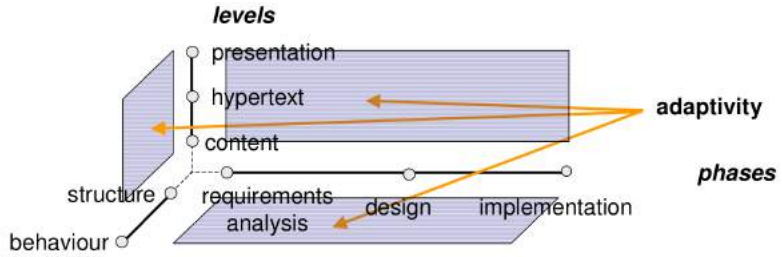
- Because they client - server systems.
- they relay on web pages & interaction schemes
- they use stateless protocol
- Hence we need specific models.

Some aspects which has to be addressed are :-

- what is the content ?
- what is navigation structure ?
- what is presentation ?
- how is the processing going ?

We can have models for cross cutting aspects like security, permissions, roles etc.

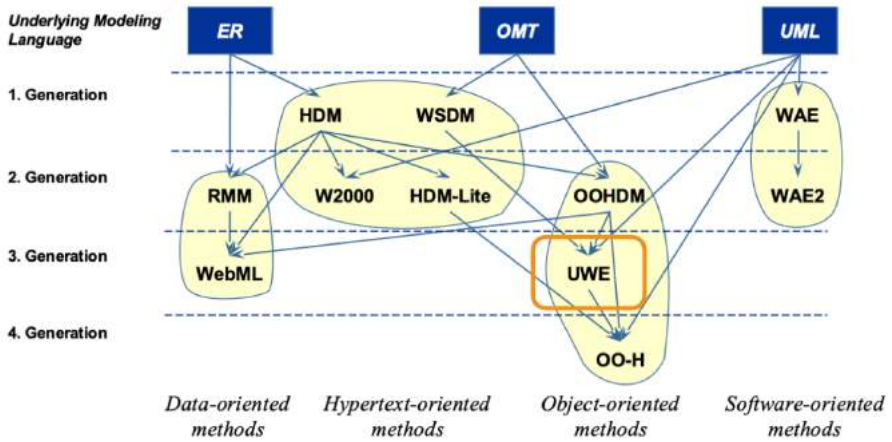
Dimensions of web Modeling :-



Adaptivity → up to date modern applications.

- Customize & adapt some aspects on based of user.
- providing services based on current content to specific users.

History of Modeling Methods



UWE – UML Based Web Engineering

- Goal of UWE → use MDE approach, generate 90% of model automatically
  - use UML and extend it using built-in extension mechanism.
- UWE adds domain level extension
- UWE provides tooling & automatic transformations. (as research prototype)
- Technical Aspects of UWE
  - → stereotypes
  - → meta model

→ model-driven development process

Refer document
↳ check on slide.

3.2 Modeling Part 2

UWE Navigation Model example
(Modele Manual System is the example) MoMa

Q. Draw a navigation model for an application.

- functionalities
- links
- texts → static & dynamic (database)

* In this lecture we connect this with Archite.
- this website uses 4 layer & MVC

Presentation layer → everything you see on the browser (visualization)

App. layer → source code (Javascript & html)

* Navigation through web-page from client (your m/c) to web server (ist.uni-koblenz.de)
→ web server is responsible to deal with all pages & communicate with business layer.

ist.uni-koblenz.de / MoMa

ist.uni-koblenz.de — web server
MoMa — URL

* web server decides which application to be handled for request.

Appl. logic → Decides what do to for a specific request.

↓

It calls a function in Business logic

↓

B.L. asks to retrieve data from DB i.e. queried from DB

* So the source package is the active controller. and in that source package we have models & this model calls a function in B.L

* Result is returned to the presentation layer

This tutorial explains 4 layer & MVC with example of MoMa

Data Transfer Objects → links to further more information.

Now we relate UWE navigation model to this application :-

purpose of Nav. model is to model all the pages & links.
Nav model is located at presentation logic layer.
pages, links & options.

Nav. model in UWE are Class Diagrams.

- This is a class
- which is called home page

(stereotypes)

- << >> → adds semantic to UML class diagram
- This class represents the page

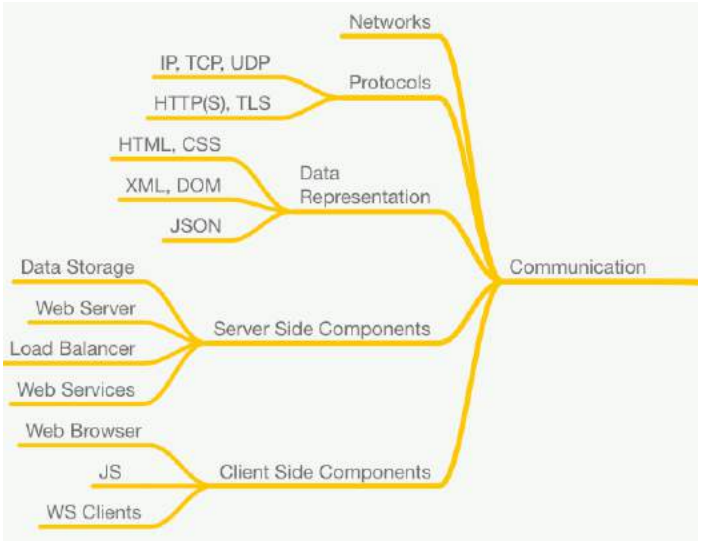
- □ → you can add icons.
- ● this is the index page

- Use composition or association.

Check Astah for diagram

Check VWE website for tutorials

4. COMMUNICATION



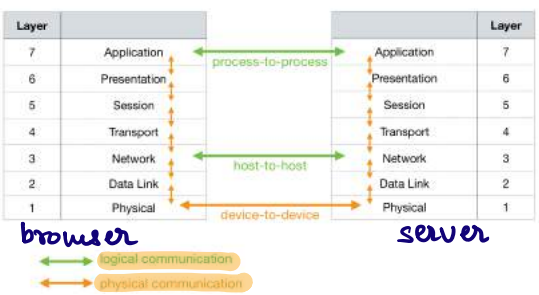
Network Communication

- communication requires → n/w nodes
  models
  architecture
  logical & physical properties
  } etc

- Various Protocols TCP/IP , IPX/SPX , SNA , UMTS etc.
- By ISO → open systems Interconnections (OSI)

- it has 7 layer.
- eg. web browsers wants to access web server using HTTP (logical)
- wire or radio (physical)

OSI Reference Model



* OSI Layers and its features.

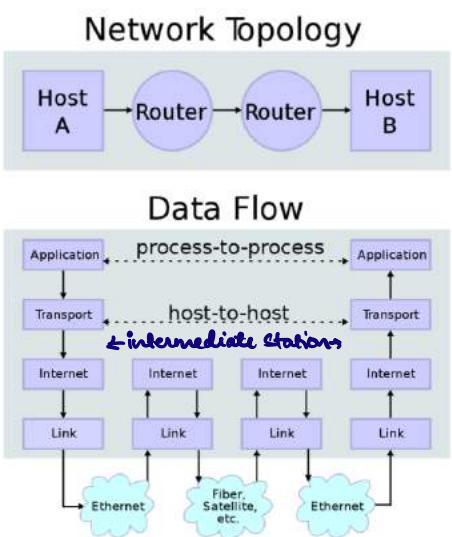
OSI model				
Layer			Protocol data unit (PDU)	Function[6]
Host layers	7	Application	Data	High-level APIs, including resource sharing, remote file access
	6	Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption
	5	Session		Managing communication sessions, i.e., continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes
	4	Transport	Segment, Datagram	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing
Media layers	3	Network	Packet	Structuring and managing a multi-node network, including addressing, routing and traffic control
	2	Data link	Frame	Reliable transmission of data frames between two nodes connected by a physical layer
	1	Physical	Symbol	Transmission and reception of raw bit streams over a physical medium

from [https://en.wikipedia.org/wiki/OSI_model]

TCP / IP (Transmission Control Protocol / Internet Protocol)

- it forms the base of world wide web.
- It comes in different versions (4 & 6 are used)
- built on abstraction of OSI stack.
- TCP has 4 layers. (application, transport internet, link)

- Hosts are communicating parties.
- If they are not on same physical local n/w, they need routers to distribute the packets over internet to route the data from source to destination.
- On each station, connection is to be built.



- 2 protocols → UDP (User Datagram Protocol)
  - like sending letter, you send & forget.
  - No control on outcome or ack.
  - Used for streaming

  ⇒ TCP
  - like phonecall, you can communicate bidirectional.

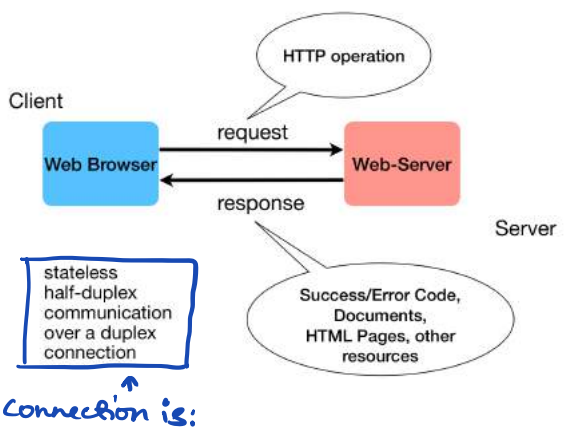
IP → packet transfer & routing scheme.

* Other application protocols are mounted on TCP. (eg. http, SMTP etc)
* Ports → endpoint to decide which service to use.

HTTP Protocol (HyperText Transfer Protocol)

- connects **web browser** to **web server** which delivers web pages, offers end point for **HTTP protocol**.
- **browser** acts as a client & it gets web pages in HTML format via protocols to render them & allow interaction.
- **Protocol** → formal description of msg send over n/w & giving them semantic.
- Client server relations ⟹
- sends HTTP request
- server answers with response (which can be webpage, error etc)

* exchange of msg.
* Communication is stateless



Connection is:

- text based protocol on application layer of OSI.
- exchange of document b/w browser & client.
- transfer static & dynamic content.
- provides web services.

HTTP Version → didn't studied

check if needs to be studied for exam.

HTTP Operations / Methods (specified in header)

Safe methods / read-only

GET → loads resource from web server.

HEAD → if doc. is present or its info (meta data)

OPTIONS → allowed methods for a resource

TRACE → echoes the command to check server is alive.

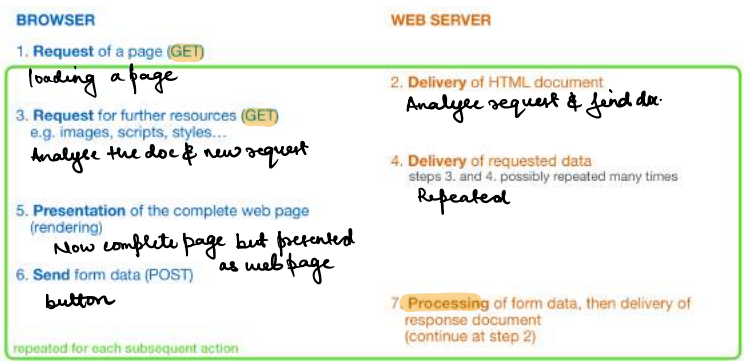
POST →

PUT →

DELETE →

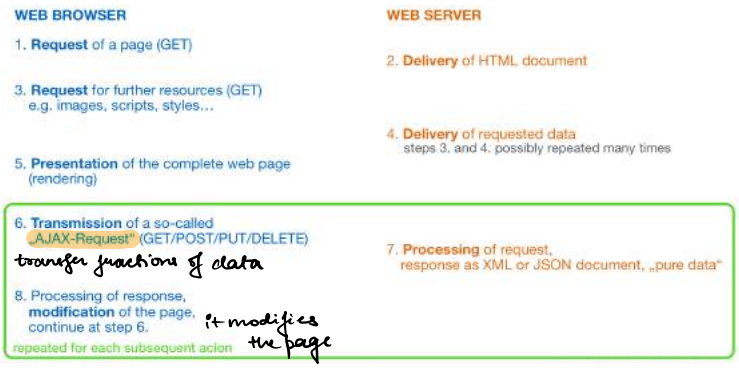
etc.

Example of Form Based Web Application :-



- curl command to communicate with web servers.

Example of Single Page Web Application



first 5 steps are same.

* uses AJAX Request.

* eg. Sodoku game, when you click update the boxes are filled. So there is only 1 web page which gets updated every time.

HTTP Request & Response Headers

Request

```
> GET / HTTP/1.1
> Host: www.uni-koblenz.de
> User-Agent: curl/7.64.1
> Accept: */*
>
```

- methods, URL path, hostname
- if required more info

Response

```
< HTTP/1.1 301 Moved Permanently
< Date: Thu, 16 Jan 2020 12:06:13 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Location: https://www.uni-koblenz-landau.de/de/koblenz/
< Content-Length: 338
< Content-Type: text/html; charset=iso-8859-1
<
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
<p>The document has moved <a href="https://www.uni-koblenz-
landau.de/de/koblenz/">here</a>.</p>
<hr>
<address>Apache/2.4.18 (Ubuntu) Server at www.uni-koblenz.de
Port 443</address>
</body></html>
```

- Protocol header, status code
- content info
- other content may be transferred.

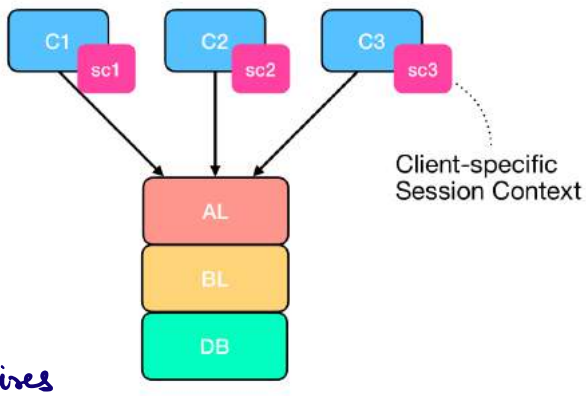
* when internet grows, protocols where added on "top of HTTP."
* You use firewalls (packet filters), privileged port-numbers.
* Extension of HTTP protocol with same port no :-
  - WebDAV
  - WebSockets
  - Messaging (Skype, iMessage, WhatsApp etc)

SESSIONS

- We need certain state that is maintained over certain period of time.
- HTTP doesn't define any means to establish such a session. (stateless)
- In web application we need session during which content is to be stored somewhere. Eg. shopping cart of online shopping during login & logout.
- where to store ? → client side
  server side.

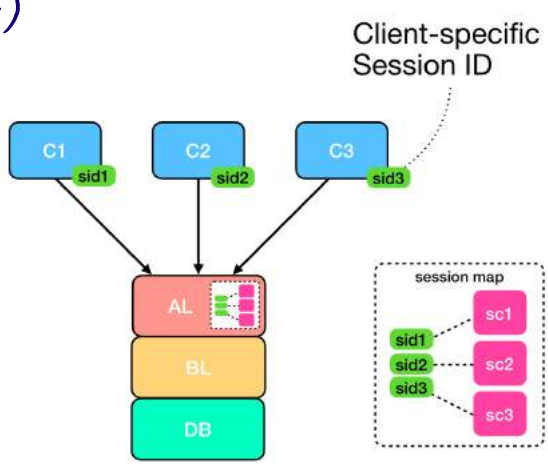
Client side session context

- each client has own session context.
- no memory consumption at server side.
- local storage → hijacking is difficult.
- but since its at client, transfer requires high data volume.
- your m/c crashes or your browser close → You loose the session.



Server side session Context (mostly used)

- clients don't store info but use session id, which is transfered to server.
- Server side has session map has ID with context.
- Adv → less data transfer
- Dis → more memory usage.
  switching server looses data.



⇒ Realizing session :- No memory of any session.

- So using session IDs.
- use session ID in GET / POST
- using session cookie.

URL rewriting

passing session as URL

```
http://host/application/page.ext?SessionID=XYZ
or
http://host/application/XYZ/page.ext
```

→ this is messy

→ can't bookmark

Using Session Cookies :-

- cookies are small bit of text stored on browser side. They have name-value pair
- used to store session ID

How? 1) Web server transfers cookie in Response header

2) Browser retransfer cookies to web server. (request header)

Session Cookies → kept in browser memory

deleted as browser terminate

Permanent Cookies → stored permanent, retransmitted

* Cookies have sensitive info.

Session Cookie Header →

```
---> GET ...

<--- HTTP/1.1 200 OK
   Content-type: text/html
   Set-Cookie: name=value
   (content of page)

---> GET ... HTTP/1.1
   Host: www.uni-koblenz.de
   Cookie: name=value
   Accept: */*
```

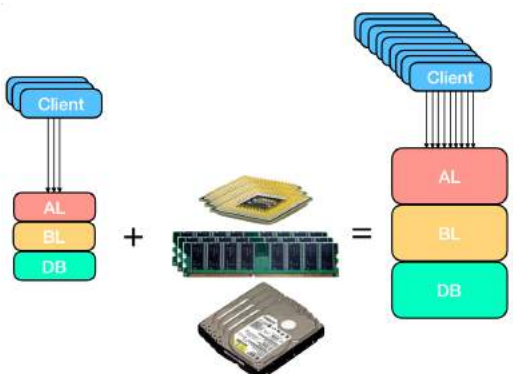
SCALABILITY

- Quality measure for how easy is to adapt a system on varying load of clients.
- If users increases → scale up with more resources.
- Ideally, scalability should be without user knowing
- It is based Architecture design.
- Web App has distributed systems. Scalability requires we scale individual component.
- Layered Architecture are more flexible to scaling.

- Resource Congestions →
  - shared clients
  - N/w bandwidth limitation
  - limitation of memory consumption, storage, CPU etc.
- Results in slow processing or denial of service.

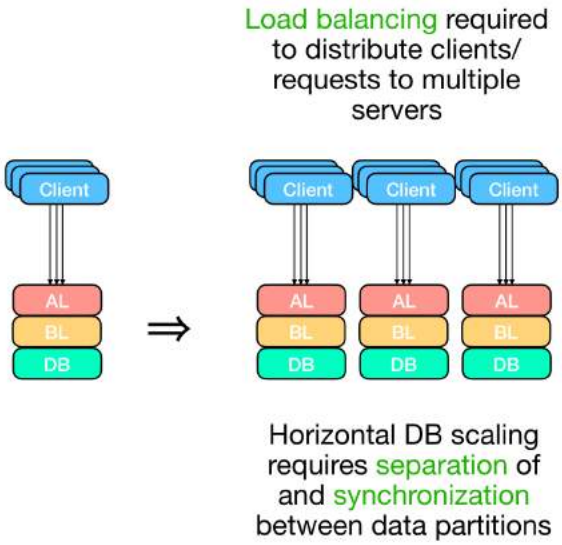
Vertical Scaling → more CPU / main memory / storage / power / bandwidth to server.

- But if the problem is not at server but at n/w this not applicable.



Horizontal Scaling → replicating servers i.e. increasing no. of m/cs

- Used in :-
  - → n/w congestion
  - → concurrent connection

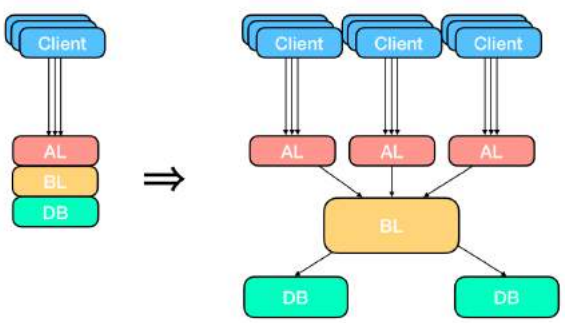


* usually done in cloud system.

But the client doesn't know of such replicated server, they want to use single address for a server (no matter how many m/cs are), therefore we need load balancing to distribute client requests to multiple server.

- Replicated DB requires separation & synchronization.

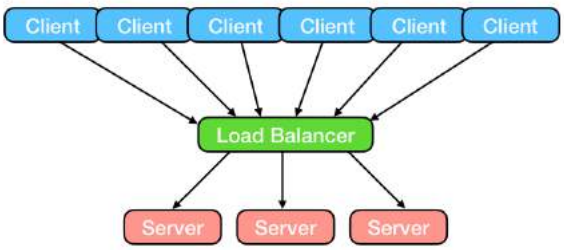
Combination Of both Scaling



- No. of clients increases then we can put more no of web servers in place
- increase no. of AL but have singe BL.
- DB can be scaled separately

LOAD BALANCING

- usually client don't care abt replication.
- Scaling should be transparent to client.



- In general clients don't connect to server directly, they use gateway (load balancer) which further distributes.
- Can be used on various layers.
- Can be static & dynamic variant.
- they have combind s/w & H/w devices.
- Load balancer are not involved deeply into protocol but work on N/w layer.
- DNS Round Robin
  - → multiple server with same name but different server ID.
  - → easy to implement.
  - → client can see multiple server.
- NAT N/w Addres translation
  - → load balancer translates & secured packets to reach internal server.
  - → Server Affinity → for sessions.
  - → User see sing node.

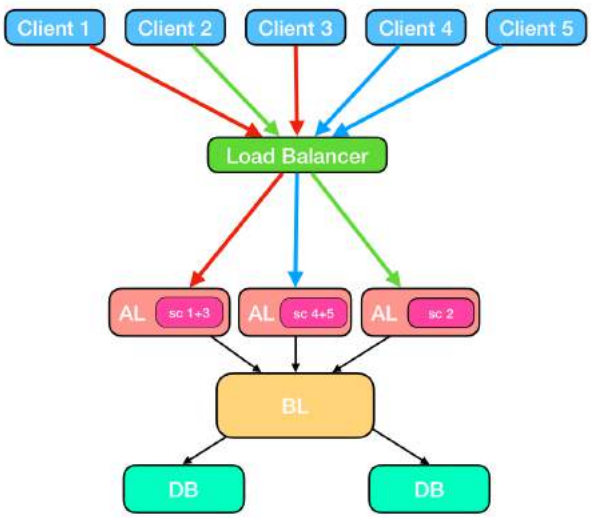
- Flat based balancing → incoming through load balancer
  outgoing through server to user.

- beneficial for data vol. as bulk traffic is send directly.

- Anycast load balancing → Global Routing
  - multiple server with same IP.
  - Border Gateway Protocol uses shortest no. of hops.
  - closest server is returned.

Load Balancing with Session

* Server Affinity → Using same server for subsequent requests of clients.
- Client 1 & 3 goes to Server 1 etc.
- Depending on scalability, sometimes sessions have to be moved to BL or DB.
  * traffic will increase.
  * synchronize DBs
- Makes it more tedious.



WEB SERVICES

- Offer functionality to client by use of connectivity & protocols of internet.
- connected by HTTP or HTTPs
- 2 mainstream types for calling these web services are :-

1) SOAP or Big Web Services

- use XML document for communication
- can be used with HTTP & other protocols
- using in email transfer.

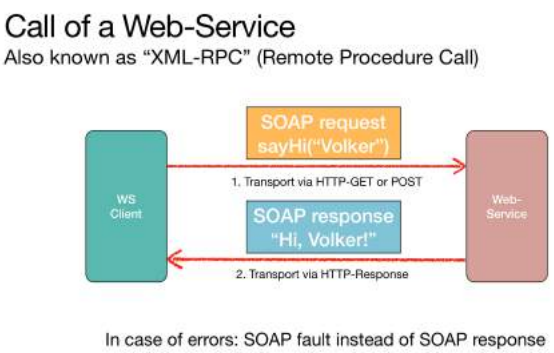
2) RESTful web services

- light weight
- interactive web-pages
- eg. SODUKO game (AJAX Javascripts calls)
- data representation - JSON documents.

SOAP (Simple Object Access Protocol)

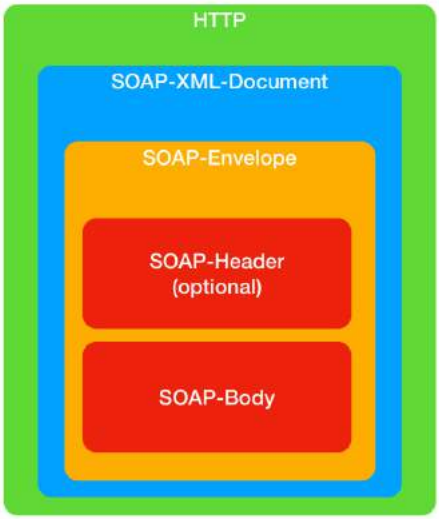
- heavily based on XML schema
- Remote methods (parameterized)
- synch & asynch. communication.
  (HTTP) (email)
- Data exchange is using XML schema language.
- allows attachment.

- Synchronizes HTTP get & Post with parameters.
- Big Messages → because they contain Meta Data.



- SOAP message ⟶
  - ↳ contains a lot of data.
  - ↳ exchanges xml documents
  - ↳ with envelop
  - ↳ header
  - ↳ soap body

* Hence high payload.



SOAP REQUEST

```
<SOAP-ENV:Body>
  <m:GetLastTradePriceResponse xmlns:m="Some-URI">
    <Price>34.5</Price>
  </m:GetLastTradePriceResponse>
</SOAP-ENV:Body>
```

SOAP Response

```
<SOAP-ENV:Body>
  <m:GetLastTradePriceResponse xmlns:m="Some-URI">
    <Price>34.5</Price>
  </m:GetLastTradePriceResponse>
</SOAP-ENV:Body>
```

SOAP Response with header has additional data in envelop

```
<SOAP-ENV:Header>
    <t:Transaction
      xmlns:t="some-URI"
      xsi:type="xsd:int" mustUnderstand="1">
        5
    </t:Transaction>
</SOAP-ENV:Header>
```

SOAP fault if something goes wrong.

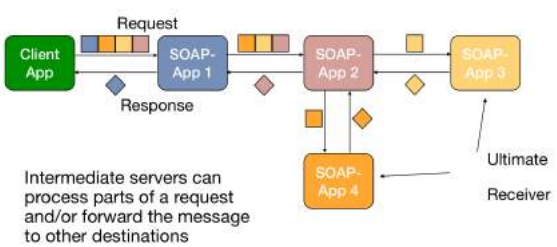
```
<SOAP-ENV:Fault>
    <faultcode>SOAP-ENV:Server</faultcode>
    <faultstring>Server Error</faultstring>
    <detail>
        <e:myfaultdetails xmlns:e="Some-URI">
          <message>
            My application didn't work
          </message>
          <errorcode>
            1001
          </errorcode>
        </e:myfaultdetails>
    </detail>
</SOAP-ENV:Fault>
```

* SOAP can deal with encryption & authentication.

* distribute single web server call to distributed servers.

- Identify the message
- Verify the message
- Fwd to next server.

Multiple SOAP App Server



WSDL (Web Service Description Language)

- one of the key service of web server is that it can describe itself.
- each SOAP comes with a file which describes the message. It has :-
  - Types
  - Message
  - Post
  - Binding
  - Service

Eg. Water level Services

* Purpose of WSDL file is to provide machine readable service description.
* But this doesn't have a semantic.
- WSDL has information Schema.
  - → payload data types
  - → Application specific

REST Services

* Representation state Transfer
* Not a protocol but architectural style
- every resource has unique identifier.
- CRUD operation using HTTP.
- Majority of XML using JSON.

Example is SODUKO game.

- when you start game a resource is created.
- this resource gives state of the game.

* GET → get the game using resource.
* DELETE → deleting the resource of game

* Idea of Rest is we have certain URL which are related to resources.
* execute HTTP methods on these resource
- We can use frameworks to built client & servers for such services.
  → Java, PHP, Node.js

- Using JSON → JavaScript Object Notation
  → JSON is more compact than XML.

Data Representation at Layer Boundaries

- Data is exchanged b/w client & server has to represented somehow.
- Many types of data exist.

MIME → Multipurpose Internet Mail Extension

Serialization :- whatever doc you want to transfer has to be transformed into msg that can be transferred using HTTP.

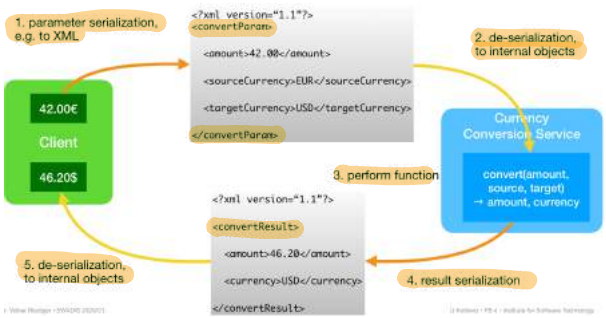
* transforms objects into sequential format.

character streams ⟶ Byte Streams

* At receiver ⟶ deserialized.
  * so should know format.

Serialization for Small Time ⟶ Marshalling

* parameters while transfer is serialized & deserialized.

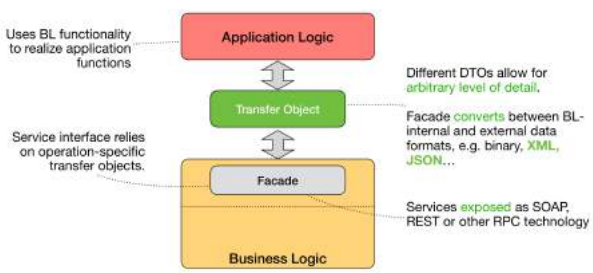


Marshalling

Relation of XML/JSON to Application Architecture

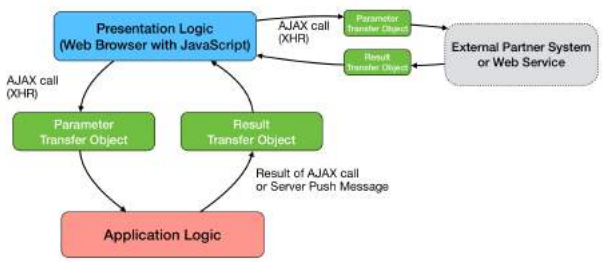
- Transfer Objects are responsible for serialization
- eg. web services offers function using REST.
  - AL ⟶ client
  - BL ⟶ server

  } serialized to JSON object using transfer object



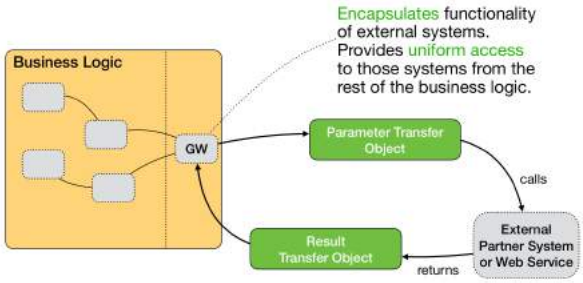
* For single page web services.

→ using JSON objects



Another example of using external web service with Gateways.

* serialize / deserialize happens at Transfer Object.



XML

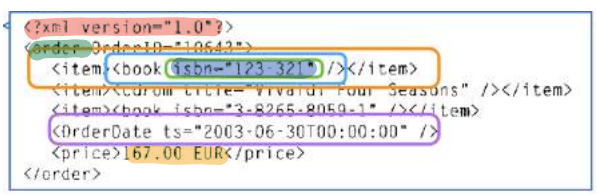
- serialized
- platform independent
- loadable
- interoperability

→ They have markups and Contents.

→ Elements & Attributes

→ they can be nested

* Attributes are name-value pairs



- Rules of well-formed XML Doc → in slide

* XML dialects → DTD, XML Schema } for domain schema

Using namespace → to distinguish vocabulary.

ns : set of namespace

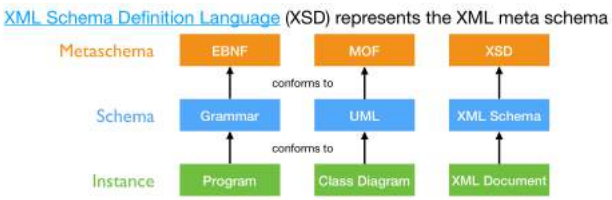
* avoid collision

xlmns : prefix

XML Schema

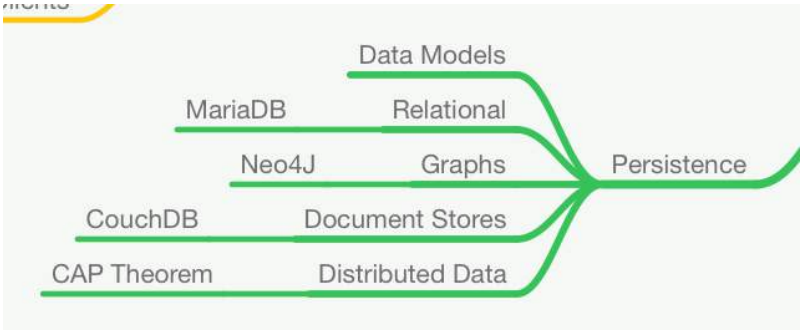
- DTD defines elements & attributes.
- XSD are more expressive but complicated.

- Example →




```
<?xml version="1.0"?>
<!DOCTYPE order [
<!ELEMENT order (item+,OrderDate,price)>
<!ATTLIST order OrderID ID #REQUIRED>
<!ELEMENT item (book,cdrom)+>
<!ELEMENT book EMPTY>
<!ATTLIST book isbn CDATA #REQUIRED>
<!ELEMENT cdrom EMPTY>
<!ATTLIST cdrom title CDATA #REQUIRED>
<!ELEMENT OrderDate EMPTY>
<!ATTLIST OrderDate ts CDATA '2003-06-30T00:00:00'>
<!ELEMENT price (#PCDATA)>
]>
```

5. PERSISTENCE



- Overview
- Data properties
- Persistence tasks
- O/R Mapping

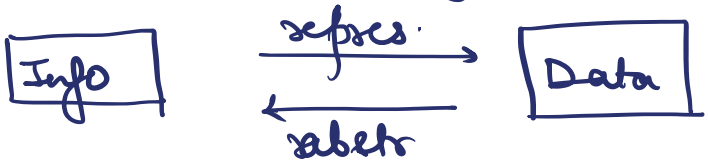
* This is b/w business logic & DB layers.

5.1 DATA PROPERTIES

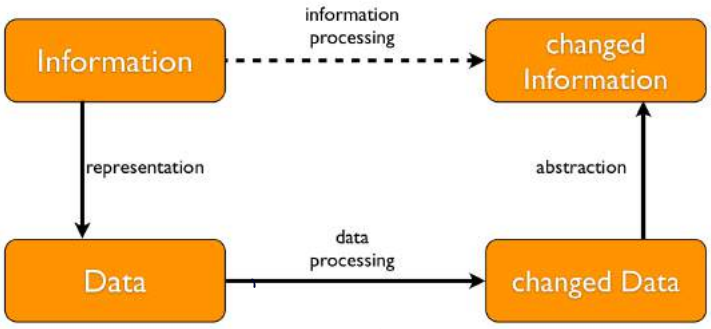
Persistence →

Data →

Information can be represented as Data.
Info can be abstracted from Data



* Computer programs process Data.



create/ update/ remove

eg. representation of no.

→ binary nos

$2^0 \cdot 2^0$ ——

→ then hexadecimal abstracted for no.

Data intensive → distributed storage & parallel processing

- data properties, sizes

Storage →

5.2 PERSISTENCE TASKS

1) Think abt concepts & relations in your application domain

→ Information Engineering

- This is more abt S/W lifecycle.

Not Relevant to this course.

2) Select Appropriate persistence technologies to serve and then map it to the schema (logical & physical schema)

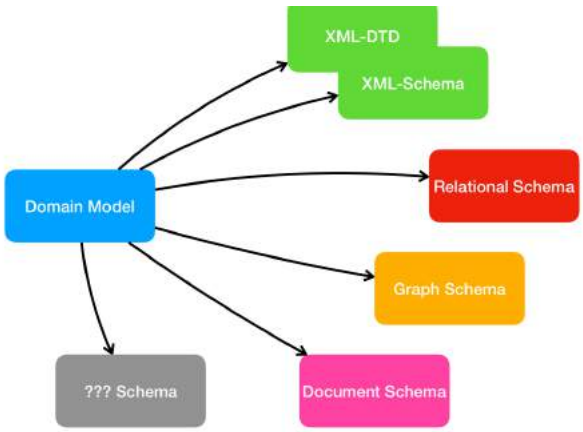
→ Derive logical Schema.

Derive logical Schema

- Various things for deciding persistence technologies :-
  - → type of data
  - → size of data
  - → data governance

  } etc. etc.

- After deciding technology, we see how our conceptual / domain model would be represented in this technology & this is called MAPPING SCHEMAS.
- This is a transformations.



- Relational Schema :-
  - tables, attributes
  - schema before data is stored.

- NoSQL Database :- schema is not fixed. it can emerge with adding data.
  - * but you cannot enforce data.

→ Read slides

* 2 variance of NoSQL DB :-

1. Graph DB → nodes & edges for data & relations.
   - directed & undirected
2. Document DB → structured content.
   - eg. XML or JSON doc.
   - documents have unique ID.
   - difficult to enforce integrity constraints.

- In many cases, a database structures already exist
- New applications have to use that database
- Application data mappings have to be defined in a way that the mapping framework uses the existing structures

3) Managing DB Content → Business logic layer to query DB, update data etc.

- DAO (Data Access Object) is Data Mapper b/w DB layer & BL.
- Resp. of Data Mapper is to connect BL components to persistence libraries
- DB is raw data.
- tasks of Data Mapper → in slide.

5.3 DATA MAPPING → 1. RELATIONAL

- OO programming V/s Relational DB

* Impedance Mismatch → use Object Relational (O/R) mapping to solve it.

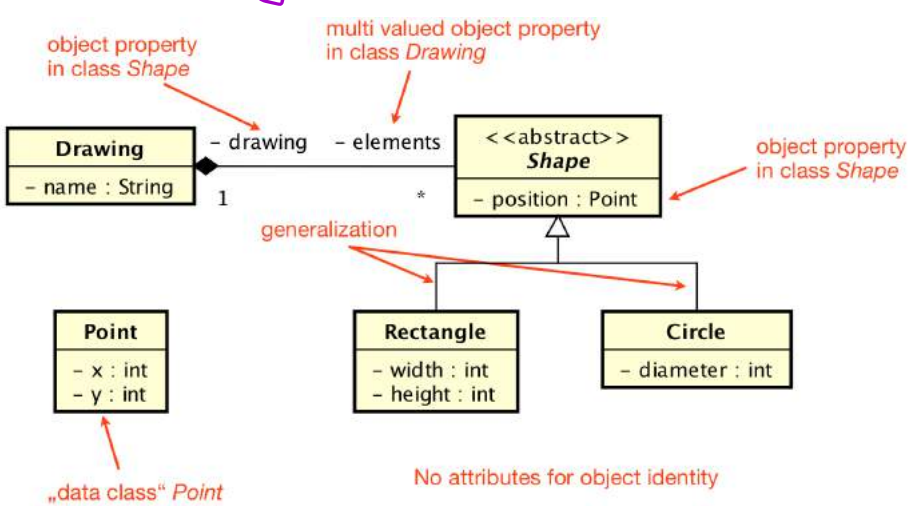
- Transform :-

Schema — class diagram ↔ Relational DB schema

Data — Object n/w ↔ table contents

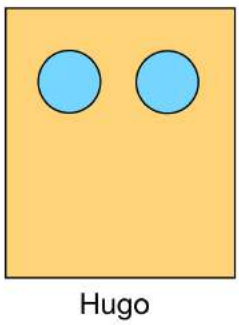
- class → set of similar objects
- Generalizations → class can have sub-classes
- Objects → instance of a class

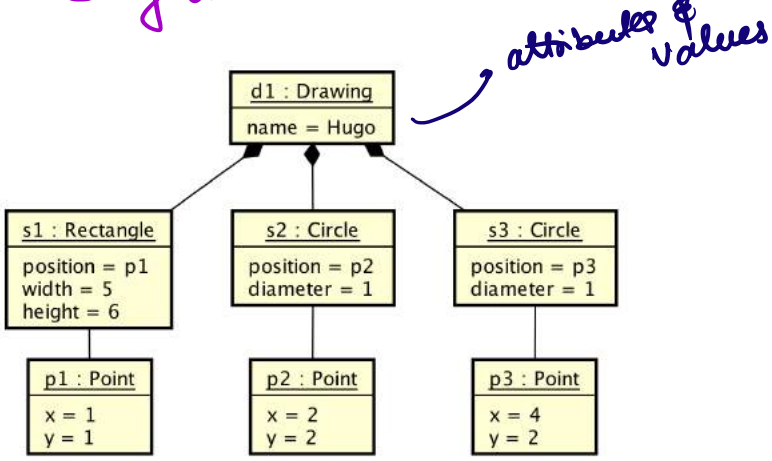
Object Model



* Drawing is associated with shape

Object Diagram





* equality → 2 objects with same properties.
* Identity → same object

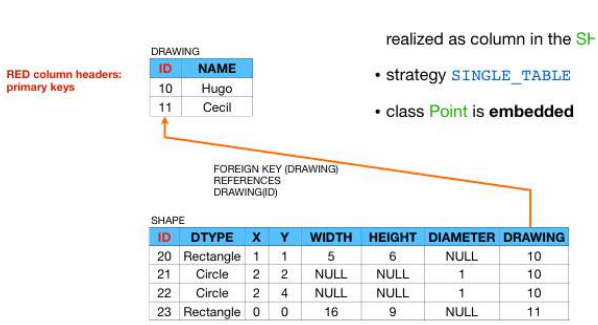
* Problems → read slides.

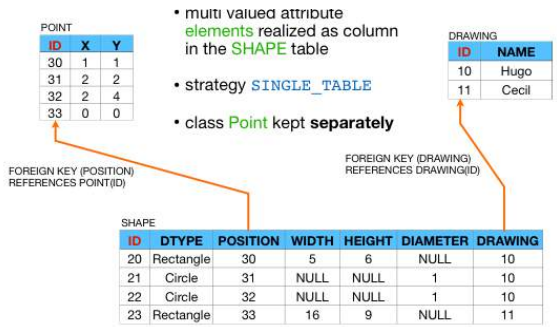
Complex Attributes :- in DB atomic values are allowed. so for more values.
- we use join tables.

* for many to many → using join table
  1 to many → column attribute

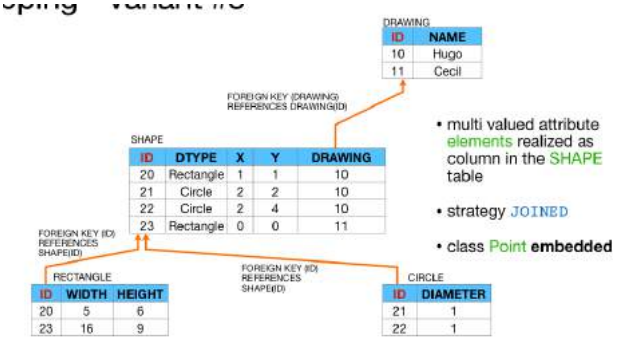
* Representing generalization
  3 strategies :- 1) using single table for all class
  2) Separate table for each class
  3) Separate table for each non-abstract class.

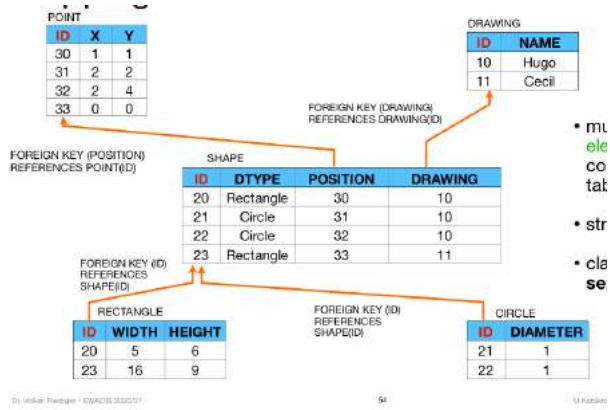
1. SINGLE_TABLE



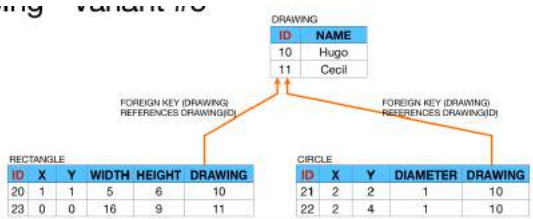


2. JOINED → creating table for each class.





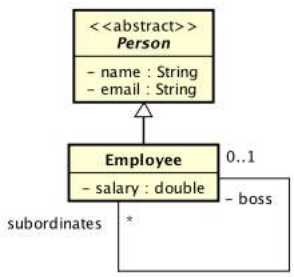
3. TABLE_PER_CLASS



	SINGLE_TABLE	JOINED	TABLE_PER_CLASS
Joins	none	many	few(er)
Memory and Runtime overhead	NULL values	indexes	indexes
Retrieve attributes of an object	single row	via joins	single row
SQL Access via superclass	use DTYPE	use DTYPE and joins	no simple solution
Multiple inheritance	ok	ok	ok

- Recursive Associations :-

1. Self Association.



PERSON

ID	DTYPE	NAME	EMAIL	SALARY	BOSS
10	Employee	Alice	...	...	13
11	Employee	Bob	...	...	13
12	Employee	Charlie	...	...	13
13	Employee	Debbie	...	...	NULL
14	Employee	Eleanor	...	...	10

FOREIGN KEY (BOSS)
REFERENCES PERSON(ID)

2. Using Join tables.

PERSON

ID	DTYPE	NAME	EMAIL	SALARY
10	Employee	Alice	...	...
11	Employee	Bob	...	...
12	Employee	Charlie	...	...
13	Employee	Debbie	...	...
14	Employee	Eleanor	...	...

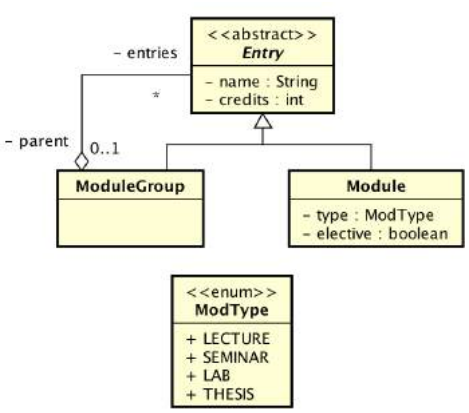
FOREIGN KEY (PERSON)
REFERENCES PERSON(ID)

PERSON_BOSS

SUBORD_ID	BOSS_ID
10	13
11	13
12	13
14	10

FOREIGN KEY (BOSS)
REFERENCES PERSON(ID)

3. Composite Pattern



5.4 DATA MAPPING – GRAPHS

1) Graph Databases

- storing in graph format.
  # check slide for theory.

- Graph DB are variant of NoSQL DB.
- Graphs are nodes & edges with objects & relations.
- persistence, consistency, APIs

- what is Graph ?

Graph is tuple of 2 sets.

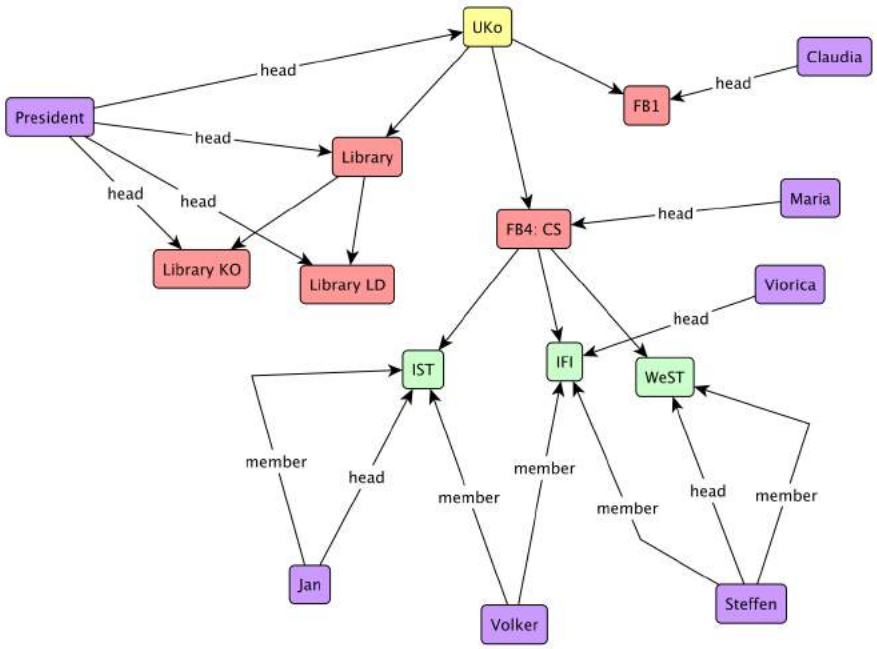
$$g = (V, E)$$

V → vertex or nodes

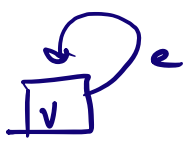
E → edges or relations.

E is a set of pair (v, w)

Simple Graph Example: University Structure



- Adjacent v_1 & v_2
- e is incident to v_1 & v_2



- $e = (v, v)$ is loop
- parallel:

University graph $= (V_1, E_1)$

$V_1 = \{ UKo, FB1, \ldots \}$

$E_1 = \{ (UKo, FB1), \ldots \}$

But this doesn't show :-

- order of vertices & edges
- order of incidence
- type/color of edges & vertices
- attributes
- directions

Graph can be represented using :-

1) Adjacency matrix
2) Incidence Lists

Adjacency Matrix

	UKo	FB1	FB4	Library	IST	IFI	...
UKo		e1	e2	e3			
FB1							
FB4					e4	e5	
Library							
IST							
IFI							
...							

Incidence Lists

	incoming	outgoing
UKo		e1, e2, e3
FB1	e1	
FB4	e2	e4, e5
Library	e3	
IST	e4	
IFI	e5	
...		

2) Graph Types

features of graph types :-

- directed/undirected edges
- vertices types
- labels
- orders
- system capabilities
- hierarchical graphs

Directed v/s undirected

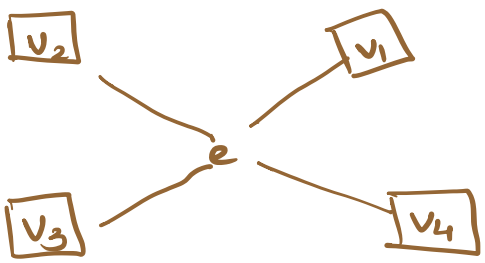
- edge has start & end

- edge connects vertices

Binary vs Hyper-edge

- edge has 2 ends

- edge has many nodes.



Ordered v/s unordered

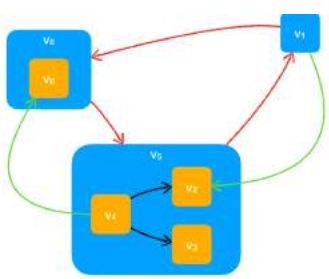
- order for edges or vertices

eg. from a vertex e_1 is 1st e_2 is 2nd etc.

- no order.

Hierarchical vs flat graphs

- graph having subgraphs



TGraphs → implemented in our university.

- each nodes & edge have type
- directed & binary (traversed both side)
- Properties (attributes) for both edges or nodes
- Strict schema.
- supports generalization & inheritance
- have deterministic order.
- No hyper edge
- No hierarchy.

Labeled Property Graph → for tutorial (in Neo4J)

- labeled nodes and can be more then 1 label.
- typed, binary, directed (both dir)
- Properties (attributes) as key-value pair

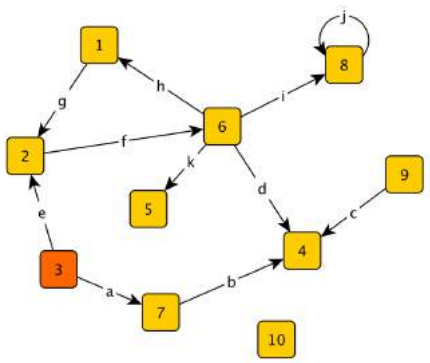
- No order
- No generalization & hierarchy
- No hyper edge.

3) Graph Traversal

- Navigation is traversal of the graph.

- Visiting all elements of data structure.

DFS → Depth First Search



if start from 3
(no edge order given)

* Not in exam

unreachable 9, 10

BFS → Breadth First Search

- Calculates shortest path

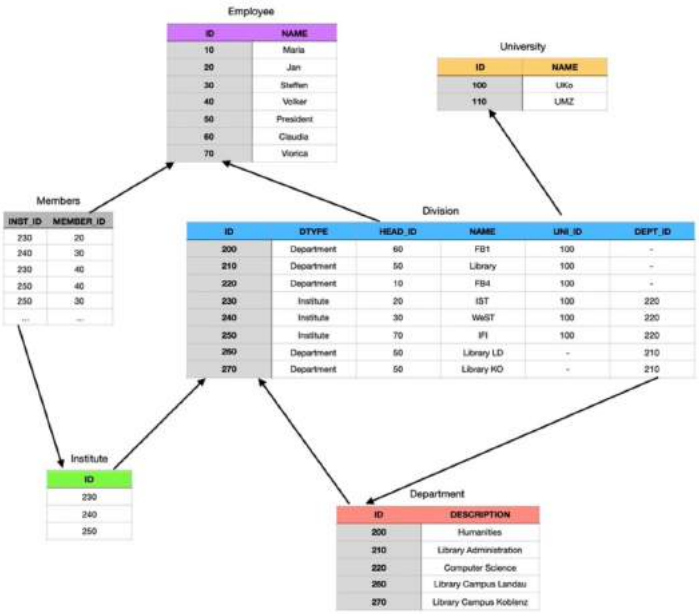
3 - a - 7 - e - 2 - b - 4 - f - 6 - (d) - h - 1 - i - 8 - k - 5 - (j) - (g)

Parameterized search

- putting constraints while searching
- specifying patterns
- they are base of graph queries
- parameters & conditions.

4. Graph Queries IMP

- Neo4J → graph DB engine



Employee

ID	NAME
10	Maria
20	Jan
30	Steffen
40	Volker
50	President
60	Claudia
70	Viorica

University

ID	NAME
100	UKo
110	UMZ

Members

INST_ID	MEMBER_ID
230	20
240	30
230	40
250	40
250	30
...	...

Division

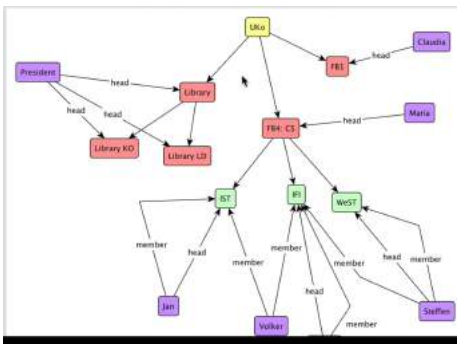
ID	DTYPE	HEAD_ID	NAME	UNI_ID	DEPT_ID
200	Department	60	FB1	100	-
210	Department	50	Library	100	-
220	Department	10	FB4	100	-
230	Institute	20	IST	100	220
240	Institute	30	WeST	100	220
250	Institute	70	IFI	100	220
260	Department	50	Library LD	-	210
270	Department	50	Library KO	-	210

Institute

ID
230
240
250

Department

ID	DESCRIPTION
200	Humanities
210	Library Administration
220	Computer Science
260	Library Campus Landau
270	Library Campus Koblenz

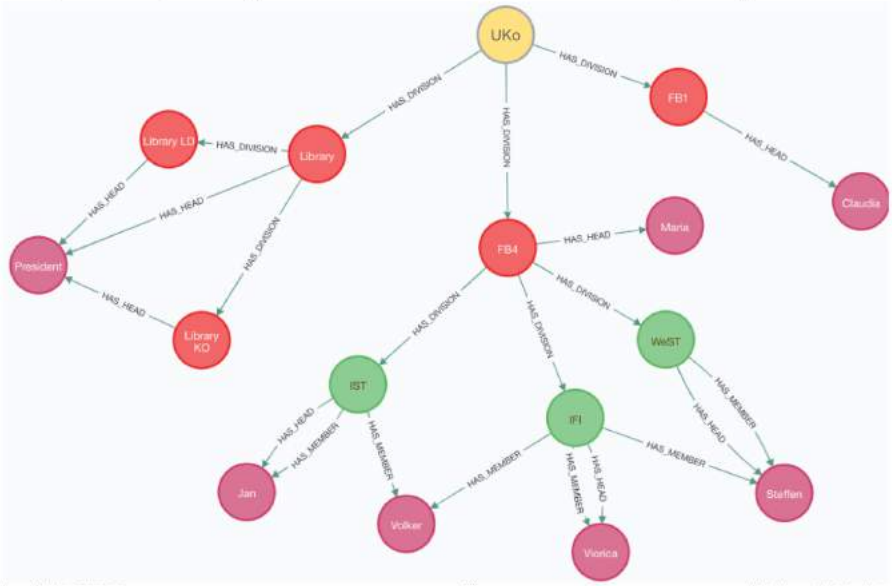


Graph Queries :- objects are nodes & relations are edges.

1) Find start nodes
2) Specify search pattern
3) Describe desired results

* Mapping to domain models.

Example Query: Determine all Employees of UKo



1. Finding Starting Nodes

Cypher Queries (CQL)

→ Declaritive graph query language.

→ General pattern :

```
MATCH <pattern>
WHERE <condition>
RETURN <result_values>
ORDER BY <sort_criteria>
```

- optional — where & orderby
- pattern can define variables to be used in where, return & orderby.

- for each match → where, return & order by is computed.
- return → only if where is TRUE
- use aggregations (count, sum, min, max, avg)

eg. find the starting node.

```
MATCH (u : University
       {name : 'UKo'}
      )
```

u → variable name

: University – node label

{...} → property values.

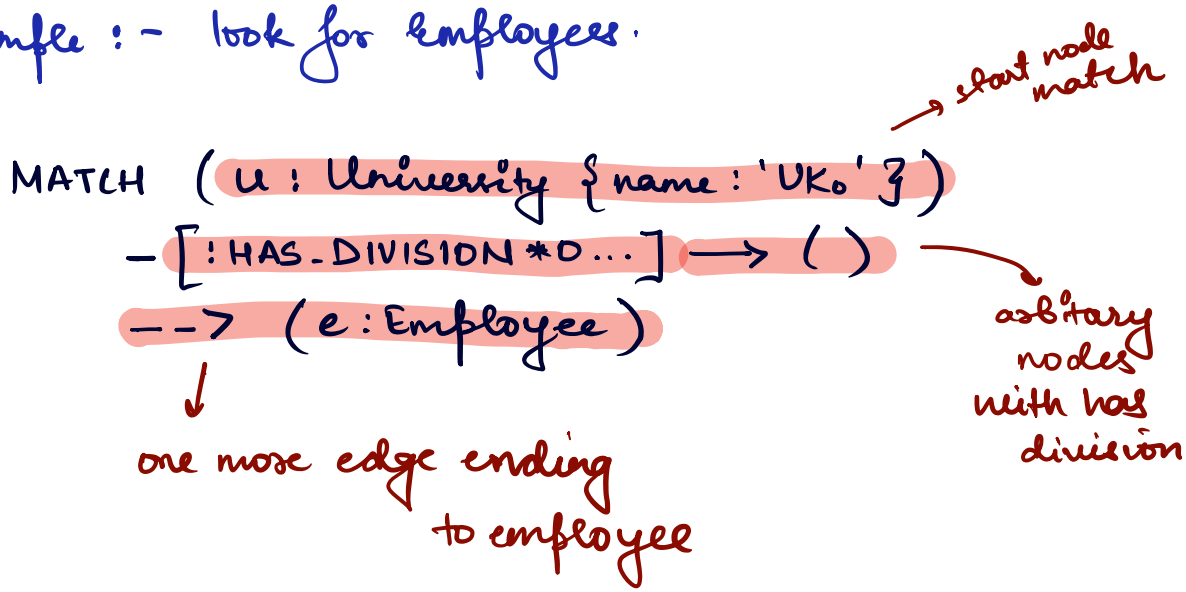
* specifying search patterns.

- In GQL, the pattern consists of a path description
- Alternate sequence of node and relation (edge) parts, ends with a node
- `-->` `<--` `---`
  `-[…]->` `<-[… ]-` `-[…]-`
  match a relation, with and without considering the direction
- `-[:Type]-` looks for an edge of the specified type
- At most one type can be matched
- Variables and properties can be specified as with node patterns
- Match repeated edges (optionally with type and direction)

  `-[*min..]-`
  at least `min` occurences

  `-[*min..max]-`
  at least `min`, at most `max` occurences
- `-[*1..5]->` matches 1 to 5 outgoing edges

Example :- look for employees.



2. Specifying search pattern.

```
MATCH (u :University {name: 'UKo'})
  -[:HAS_DIVISION*0..]-> ()
  --> (e :Employee)
WHERE e.name STARTS WITH 'Ma'
```

3. Return to get desired results
   - use distinct
   - use orderby

"e"
{"name":"Claudia"}
{"name":"Jan"}
{"name":"Jan"}
{"name":"Maria"}
{"name":"President"}
{"name":"President"}
{"name":"President"}
{"name":"Steffen"}
{"name":"Steffen"}
{"name":"Steffen"}
{"name":"Viorica"}
{"name":"Viorica"}
{"name":"Volker"}
{"name":"Volker"}

```
MATCH (u :University {name: 'UKo'})
  -[:HAS_DIVISION*0..]-> ()
  --> (e :Employee)
RETURN e
ORDER BY e.name
```

This returns nodes not employees with duplicate.

```
MATCH (u :University {name: 'UKo'})
  -[:HAS_DIVISION*0..]-> ()
  --> (e :Employee)
RETURN DISTINCT e
ORDER BY e.name
```

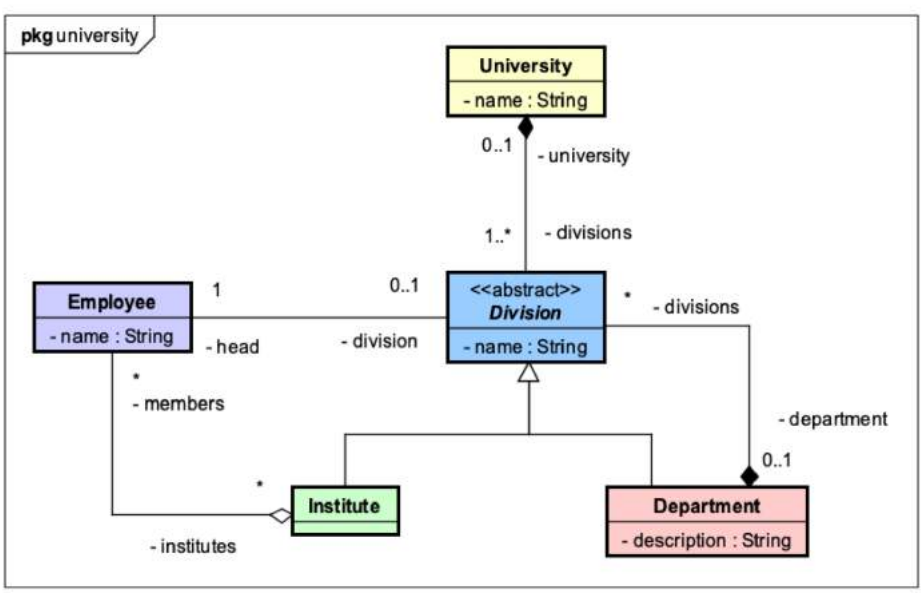
- Eliminating duplicates by `DISTINCT` finally gives the desired result

"e"
{"name":"Claudia"}
{"name":"Jan"}
{"name":"Maria"}
{"name":"President"}
{"name":"Steffen"}
{"name":"Viorica"}
{"name":"Volker"}

Return e.name → returns the names.

* Example of where
  - where e.name STARTS WITH 'S'
  - size (e.name) > 6

4. Graph Schema IMP



1. Map OO model to Graph (Schema)
2. Represent edges & nodes (Instance)

2. Store objects in graphs (Processing)

How to map OO schema to Graph Schema?

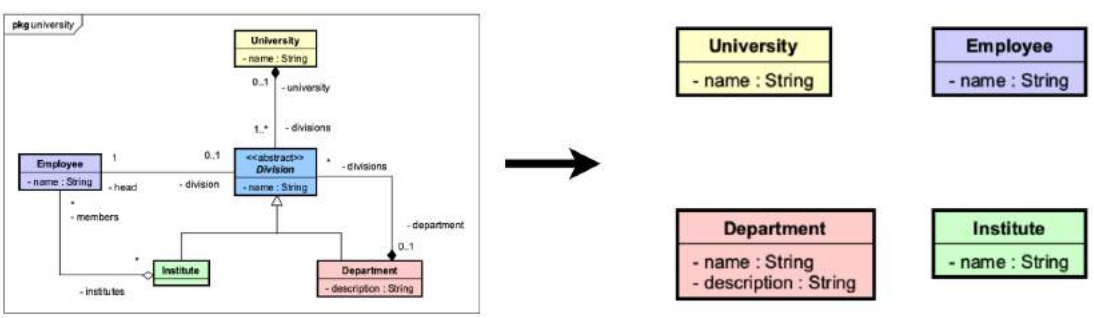
Graph Schema → • which graphs are valid instance.

use link in slide

- UML v/s Labeled Property Graphs

Mapping :-

- Class to nodes
- Attributes to properties
- leave abstract classes.
- inherit attribute of parent class

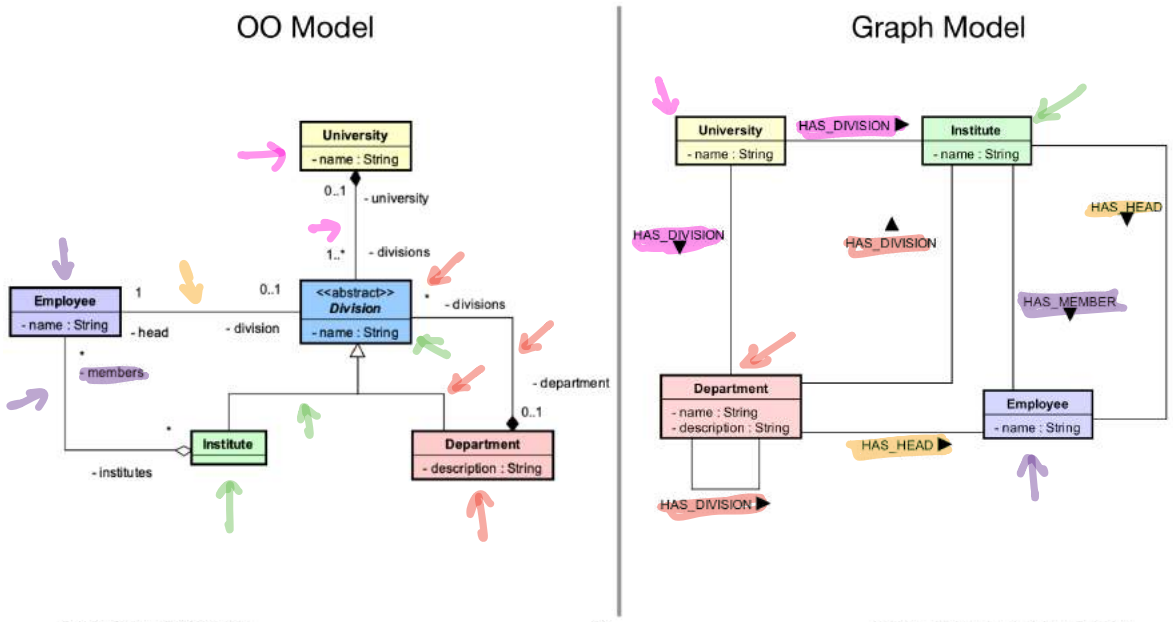


Mapping Associations → add edges as association.

- for directions :-
  - * Agg & comp
    - → parent to child

* Association

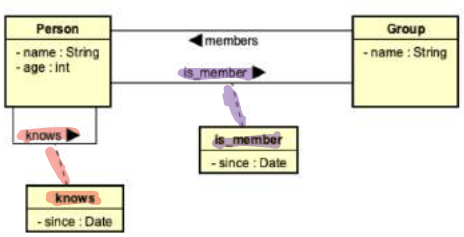
→ decide meaningful

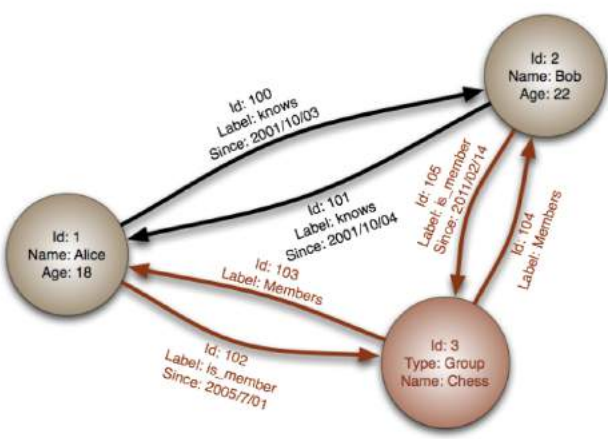


Adding properties to edges

Edge properties

- If required and appropriate, add properties to edges
- Depicted as association classes





5.5 DATA MAPPING – DOCUMENTS

* Persistence Objects are located at B.L

- The document data larger info with DB
- self-contained unit of interest.
- structured → XML or JSON
- various formats
- no schema
- no standardized query language.
- Object Document Mapping → objects/relations belonging together should form a document.
  - Domain knowledge requires.
- CouchDB → Document store
  * relies on REST API, index structure.

5.6 DISTRIBUTED DATA STORAGE

- we need distributed data if it becomes too large for single location
  → or processing performance is low.
  → if you can live with one storage → stick to that (don't complicate)

* suppose one of the problem is performance (queries) → this

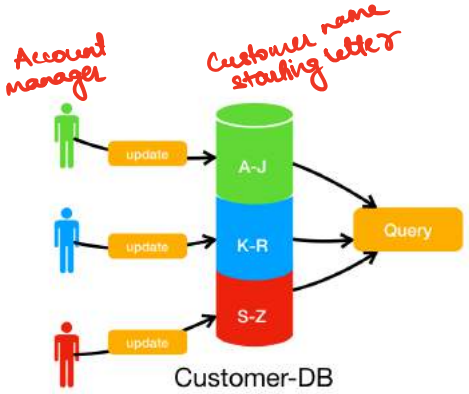
can be handled in local measures also. So you don't need distributed system. You can use parallelize queries in memory.

Vertical Scaling → put more power / memory / storage to a server.
Partitioning →
Horizontal Scaling →
Clustering →
These are all local measures.

Non-local measures : -

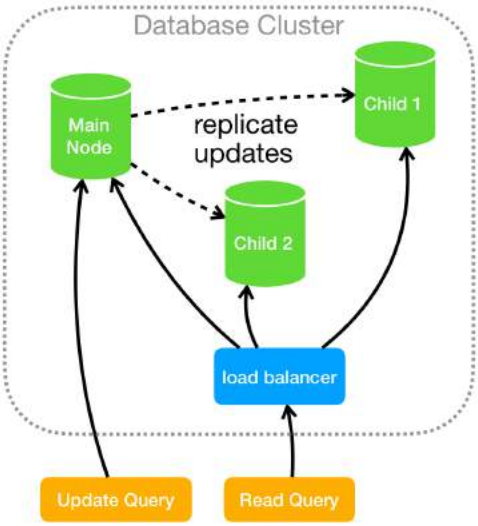
- distributed storage / processing
- Replication
- Sharding

PARTITIONING



- separate data at logical or organizational boundaries.
- eg. dividing few manager taking care of few customers.
- Concurrent changes → hardly occurs as their are different partitions.
- So we don't need synchronization while parallel updates (queries)
- minimize conflicts & blocking due to write operation or locked pages. Also less wait-time.

CLUSTERING

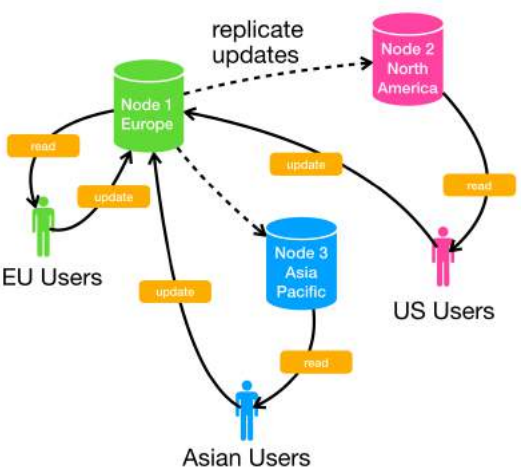


- type of horizontal scaling where you add more DB servers & create local replication.
- Read queries can be handled by replicated clusters through load balancer.
- Write/update are done through main node & then distributes to child node.
- Each node contains same data so child node can be considered as backup.

Data Intensive :- Systems

- huge amts of data
- users are distributed. (global access)
- Balancing these processing loads requires distributed parallel processing.
- Minimize moving of data. by keeping it near to user.

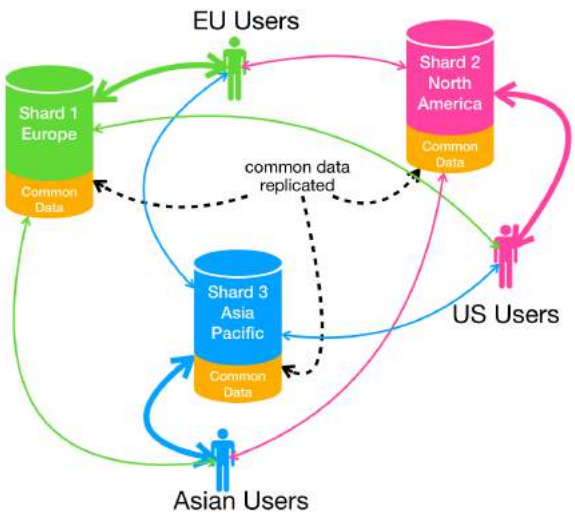
REPLICATION



- Nodes are more distant as compared to clustering.
- More data centers which are geographically close. (avoids high Bandwidth)

- Replication is that we provide all nodes with complete dataset.
  (in clustering we used Hierarchical Approach)
- Read queries from closest nodes.
- Used as backup as all nodes are same.
- Updates are replicated to each node.

SHARDING → used by Facebook



- Not all data is stored at every node
- there are local responsibilities.
- Replication is done for only small amount of common data. (as it receives few updates)
- If US user asks for EU info, he is directed to EU node, which is slow but assumption is less frequent.

ACID TRANSACTION

Atomicity, Consistency, Isolation, Durability

* Challenge to do concurrent Updates.
  - → used by multiple users
  - → integrity constraints.

- Dirty Read → time sharing multiple users.
  Reading modified value which are changed later by other user.

- Non-Repeatable Read → Reading same variable twice but getting different as in b/w it is updated by another user.

- Lost Update → Value rewritten by another user.

What is the Solution?

- locking data temprorary → pessimist Approach
- Checking for conflicts every time writing a data in concurrent modification → optimistic strategy

Transaction is seq. of logically associated read & write operation.
→ R/W should be seq. & grouped.

- All or none operations of such txn are executed.
- If conflict, during txn the view is hidden.
  - begin
  - commit → save all changes, succeed or fail
  - rollback → in case of failed commit

- Atomicity → All R/W are executed completly or not all all.
  No data loss. (lost update)

- Isolation → Concurrent txns are executed independently
  - Versions & Revisions
  - eliminates non-repeatable
  - eliminate Lost Update

- Durability → Changes of successful commit are saved permanently.

- Consistency → At low level. All constraints should hold. eg. primary key, foreign key. before & after execution of txn. Its application dependent.

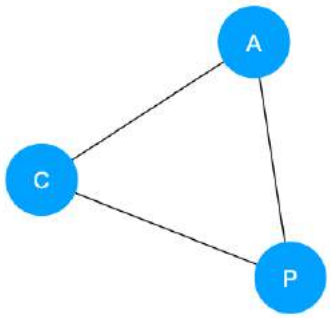
CAP THEOREM

Consistency, Availability & Partition Tolerance

- impossible for distributed system to provide more than 2 out of 3 properties at a time (simultaneously)
- so you can have C-A, C-P, A-P

→ Consistency :- If you read a data you receive most recent version or error.

→ Availability :- You get the data w/o guarantee that it is most recent write.



→ Partition Tolerance :- If one of nodes / links fails, system continues to provide service with remaining nodes.

C - P System (No Availability)

→ We want consistent system rather than having everything available.

- some operations might be unavailable.
- Detect partitions.
- Sometimes write is prohibited

Eg. Banking → everything is consistent, so you can have a look at actual balances etc. but may be some features like transfer money is not available right now since system is facing some issue.

C - A System (No Partition Tolerance)

→ It is unrealistic.

→ for non-distributed systems.

→ Single node may fail and becomes unavailable.

A-P System (No Consistency)

→ Generally used for Social N/W.

→ Availability is high for users.

→ while partioning consistency is violated.

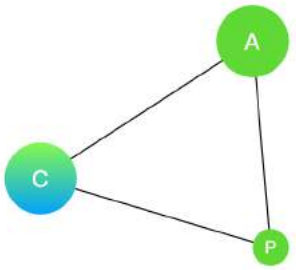
→ You can write but they can be inconsistent.

★ CAP is old (20 years) so used less, now BASE is used.

BASE

(Basically Available, Soft state, Eventually Consistent)

* we want all but we compromise bit of consistency.



- In global n/w, partitioning is low.
- Basically Available → Virtually available at all time.
- Soft State → Each node might not know synchronized complete state of system all the time.
  - So incomplete info must be present.
- Responses might not be latest changes.
- R/W are available at all time. So writes are delayed.
- Conflicts are present and should be handled.

* in BASE we are lowering our expectations from some properties.