

## Java 8

Classical : Imperative

# How

# Object mutability

Java 8 : Declarative Style

# What we want

# Object immutability

Interface:

# default method (definition)

# static method (definition)

collection api

interface ( 10 functionalities + 2)  
stream

Object Oriented approach : interface

Functional Interface :

Contain only one abstract method

might have static , default method in any count

Lambda:

anonymous function

no method param type, return type

not be encapsulated in any class

can be assigned to a variable of functional interface

```
() -> <single line>
```

```
()->{  
    // multi line  
}
```

```
(msg)->{}
```

```
msg -> {}
```

```
(a,b) -> {}
```

```
(a,b)->{  
    int sum = a+b;  
    return sum;  
}
```

```
(a,b)-> a+b; // by default associated with return statement
```

the method signature of the only abstract method of Functional interface must match with method signature of lambda expression

java.util.function

functional interface containing some very common prototype method

4 categories

Consumer

Predicate

Function

Supplier

Consumer :

void accept(<T>)

Predicate

boolean test(<T>)

Function

<R> apply(<T>);

Supplier

<T> get()

### Variants

Consumer : BiConsumer (Generic)

void accept(<T>,<M>)

### Primitive type implementation

IntConsumer()

### Predicate:

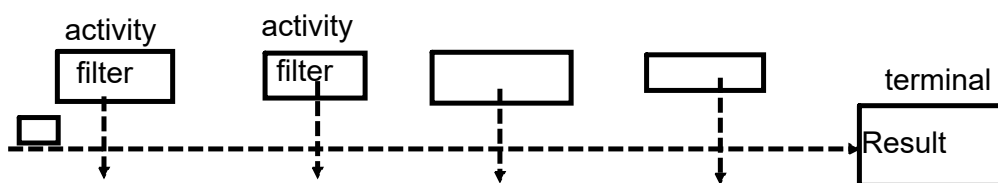
BiPredicate, Primitive type implementation

### Function : BiFunction

Functional programming remove overhead of creating objects and loading class files

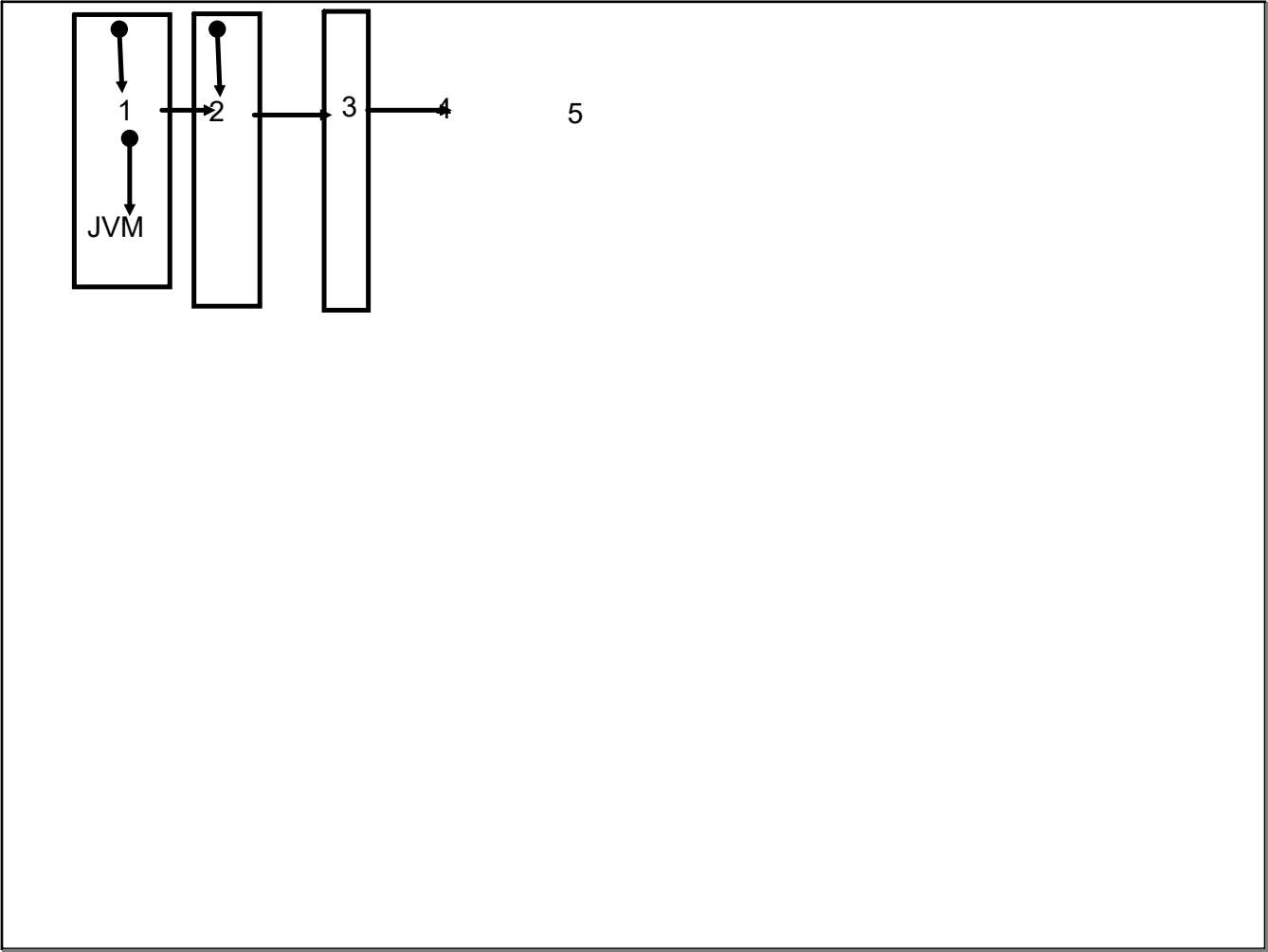


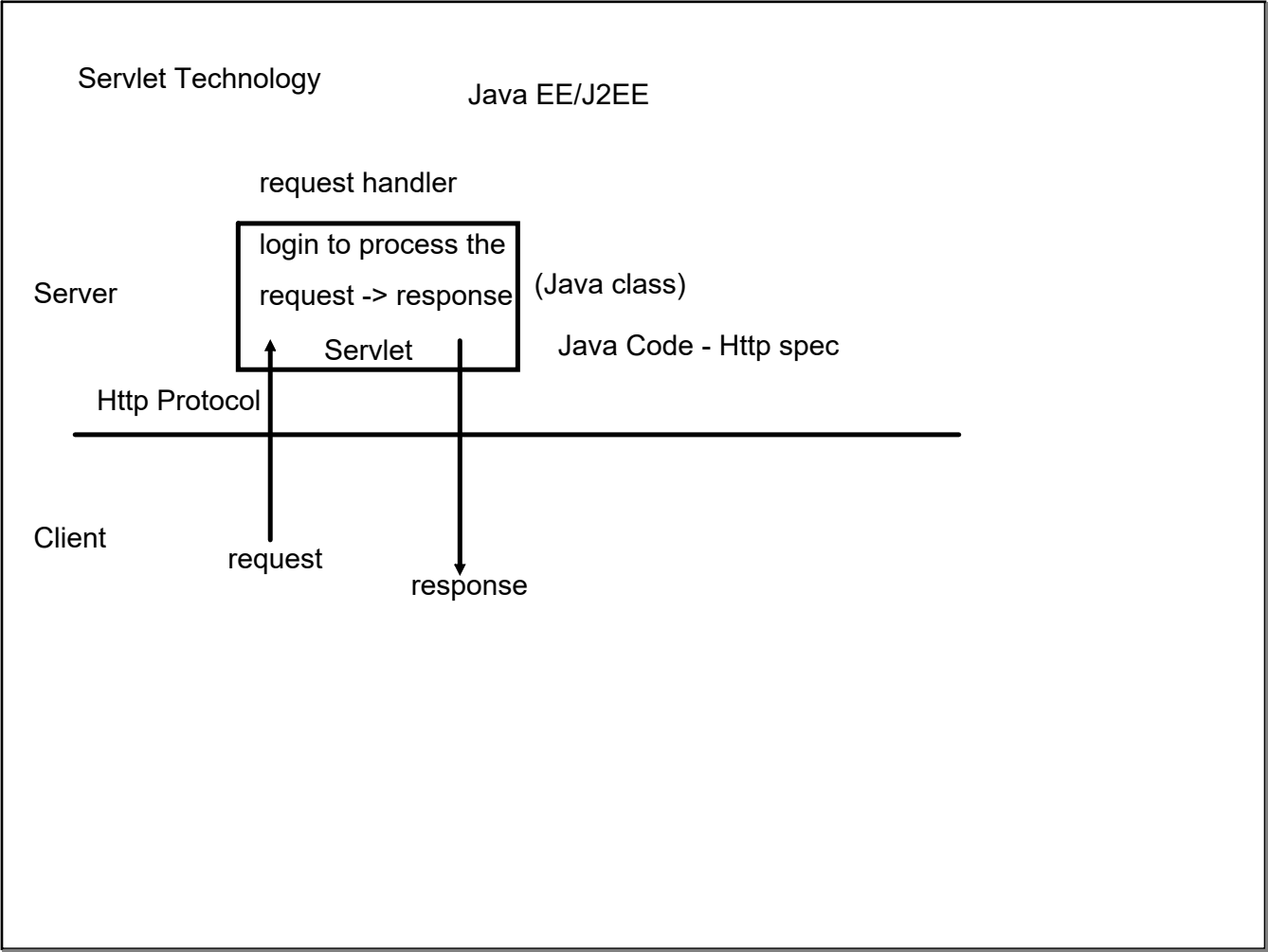
Conveyer belt



Parallel Processing not to be preferred

1. when an external mutable object is involved
2. when the stream activities involve some inherent complexity







Java EE

Servlet

Java Class

```
class MyServ extends GenericServlet/HttpServlet{
```

```
}
```

GenericServlet : Support only generic Http Verb (Form verbs) get/post

HttpServlet : identifies HTTP Verbs (get,post,put,delete)

identifies intention of http verb

```
class MyServ extends GenericServlet/HttpServlet{
    <---- lifecycle ----->
    init (load)
    service(GenericServlet) (doPost,doGet,doPut,doDelete: HttpServlet) : request/response
    destroy (unloaded)
}

service/(doPost,doGet,doPut,doDelete) (HttpServletRequest request,
                                         HttpServletResponse response){

}
```

### Named Core Datatypes of TS

number     -1/5.3/200 5~5.0

string     'Hello',"Hello",`Hello`

boolean    true/false

## Spring

Spring Core

Spring MVC (maven)

Spring Boot

Spring Framework : Servlet technology

### CORE

Bean Factory

IoC

DI

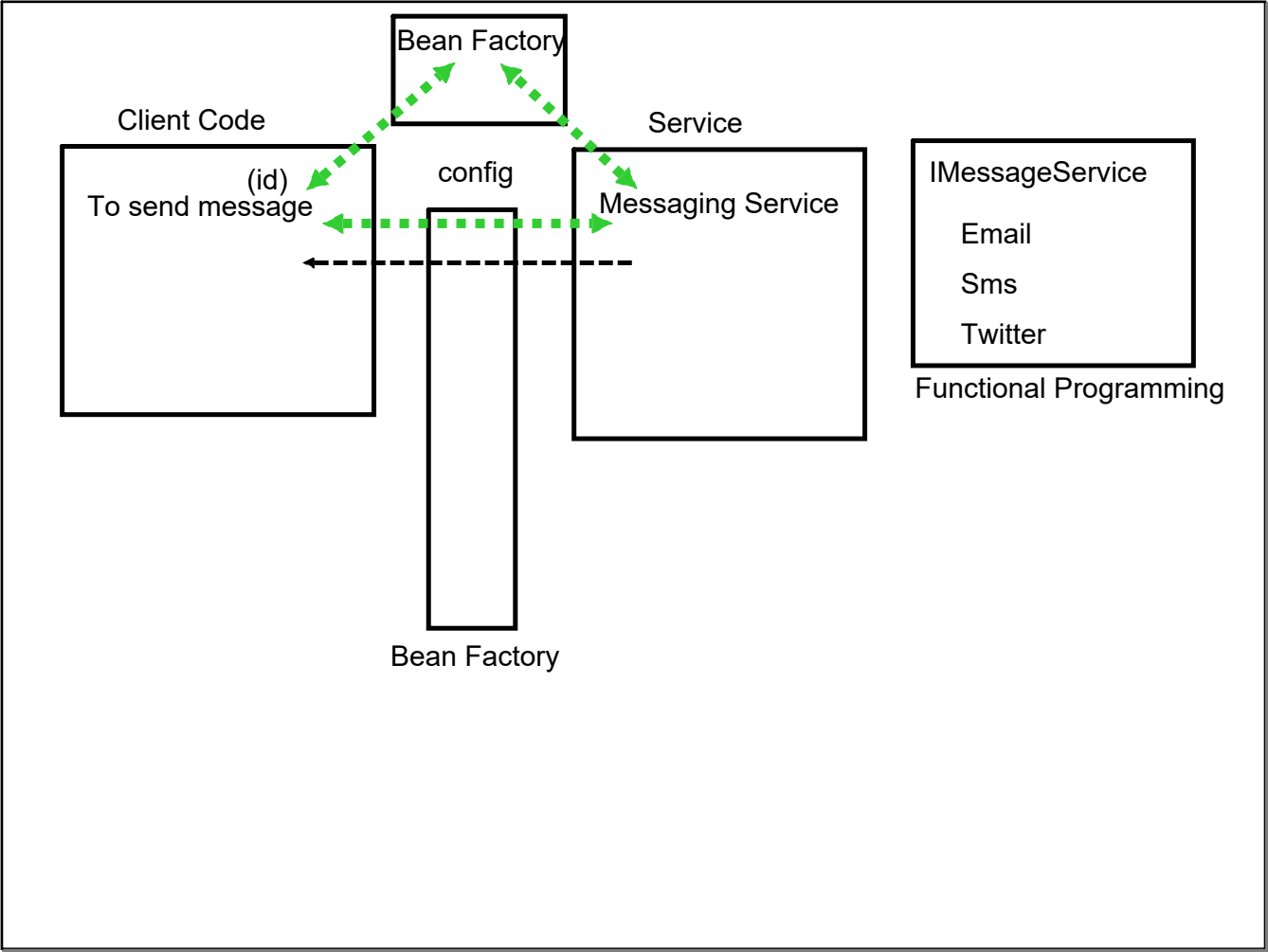
AOP

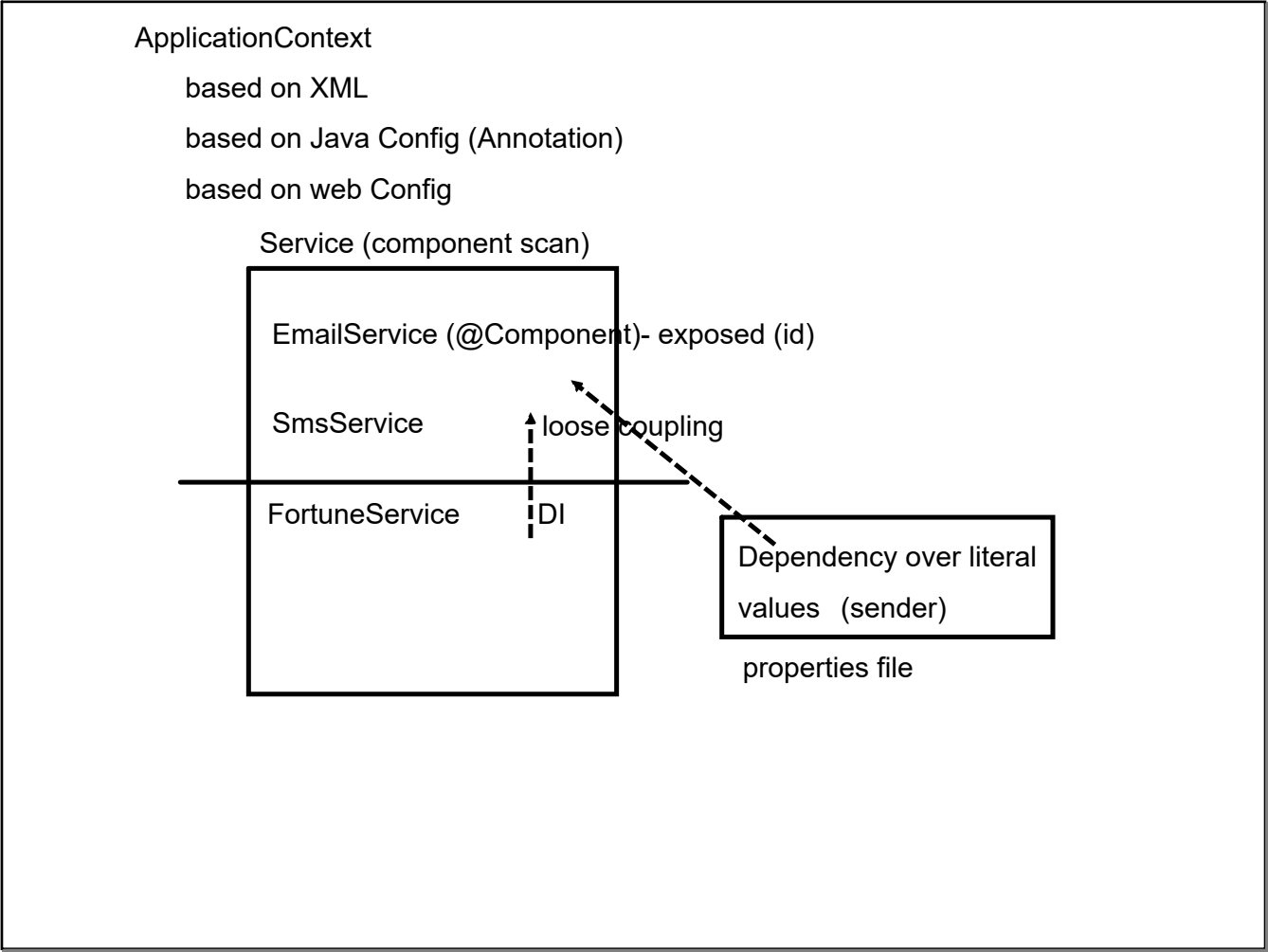
IoC : Outsourcing the creation and management of object

Bean : Java Object managed by container

AOP : Aspect Oriented Programming (Proxy)

Clean, Loosly Coupled, reusable JAva Code



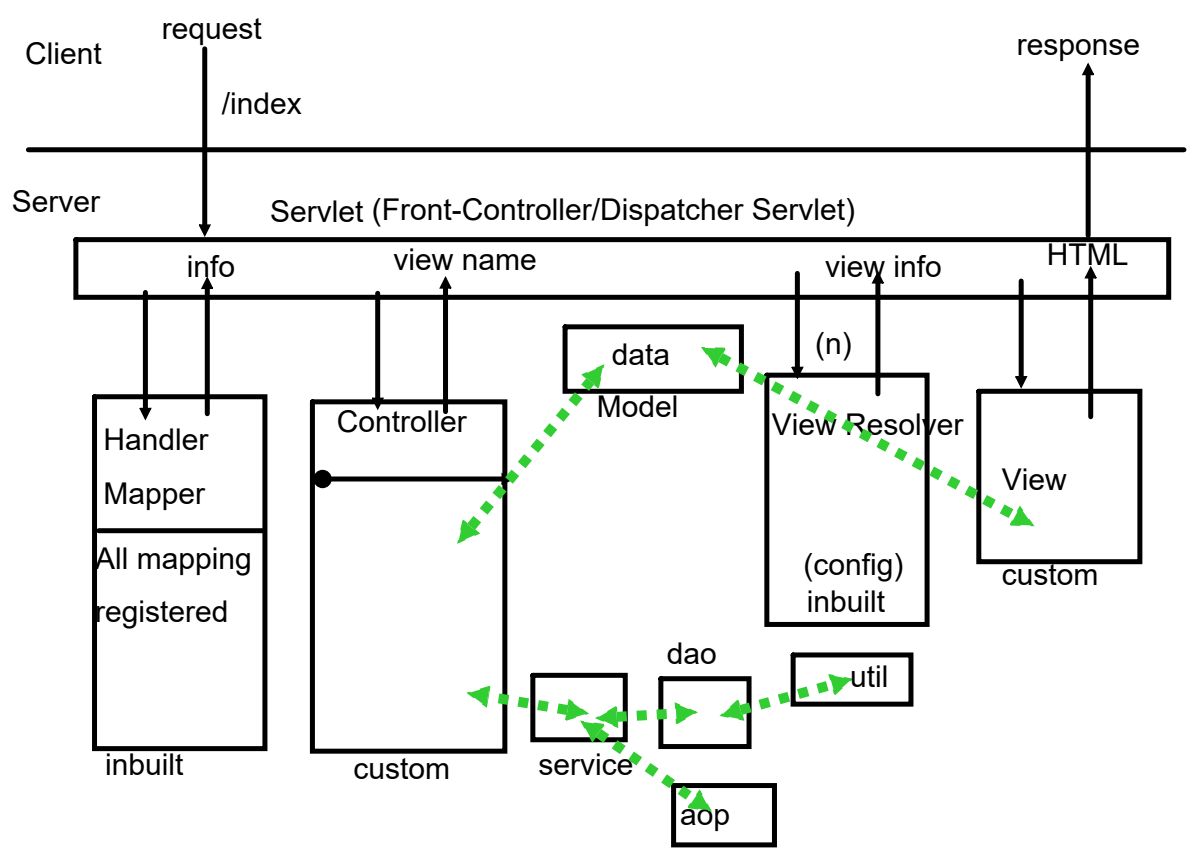


Scope : Singleton (Default)  
Prototype

request : single request-response cycle  
session : all request-response cycle for a particular user  
global : all request-response cycle for all user(web context)

Spring Context does not maintains complete lifecycle of Prototype bean

Servlet Technology being used in Spring MVC framework





Maven:

Dependency Management

Standard folder/file system

build

test

documentation

pom : project object model

all config related to maven activity

web.xml : a must file for servlet config

web.xml : Servlet config

Custom spring servlet config (java):

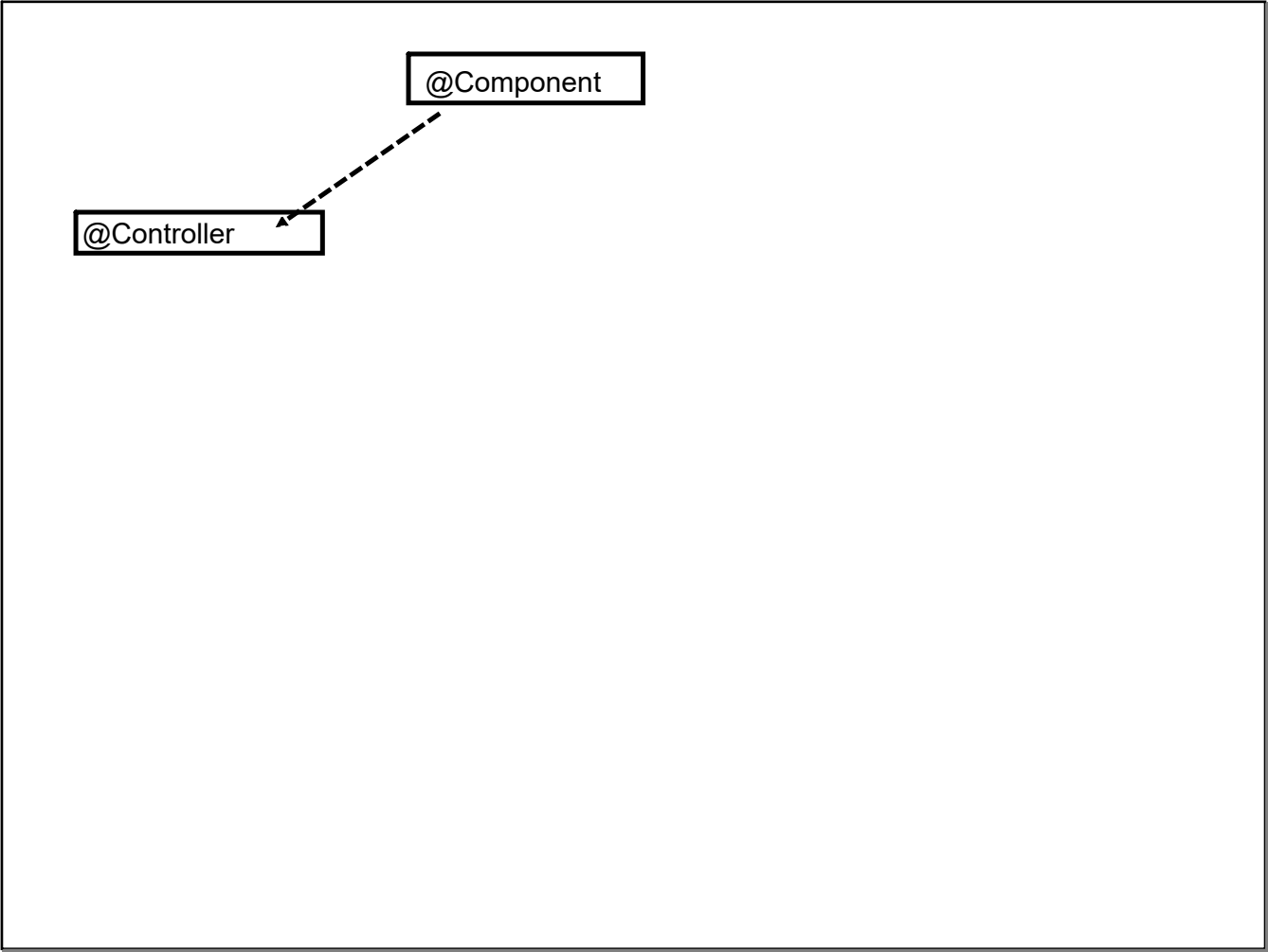
inbuilt servlet : register that servlet (DispatcherServlet : config)

config code: as per more requirement

controller code ( multiple support layers)

model : data structure

views : presentation purpose



**View Resolver:**

Single View File (jsp)

Modular View File (Tiles)

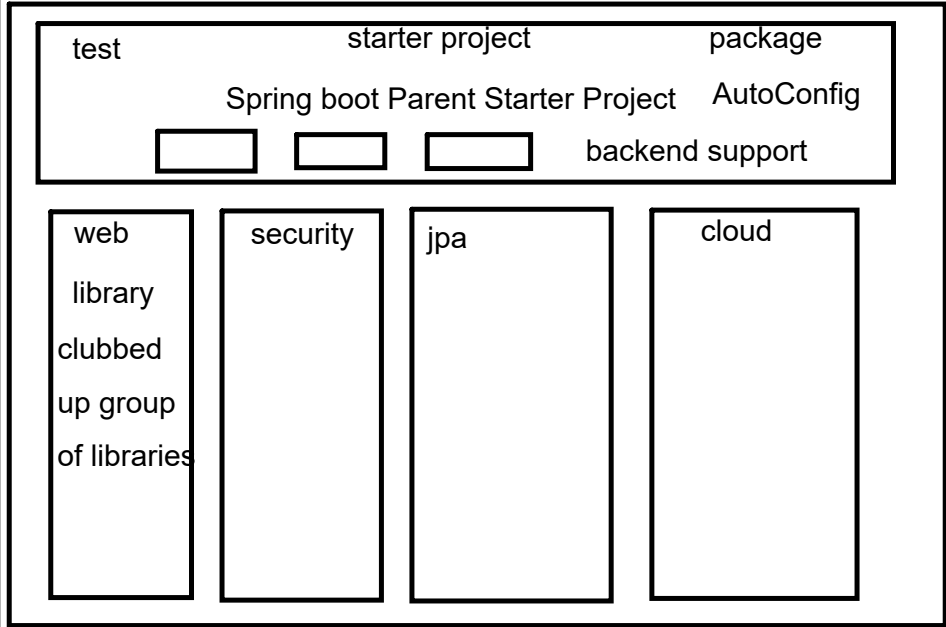
Multipart response (as downloadable file)

=>What type of responses you want

=>What type of responses your view templates

Spring Boot:

1. Dependency management 2.2



Configuration:

Auto/Easy

=> Curated clubbed up Annotation

=> Added new annotation for custom config

=> property files : add correct key-values pair

=> adding dependency : will activate that feature and auto configure

some default behavior

spring-security (

spring-actuator

spring-devtool

web application

- spring boot web application packaged as jar

- standalone : executed like a simple java

- Tomcat is embedded

Spring boot is self-sufficient for maven tool

- eg : mvn package/test/clean/install

spring boot tool:

- eg: mvnw package/test/clean/install

Spring boot application are by default

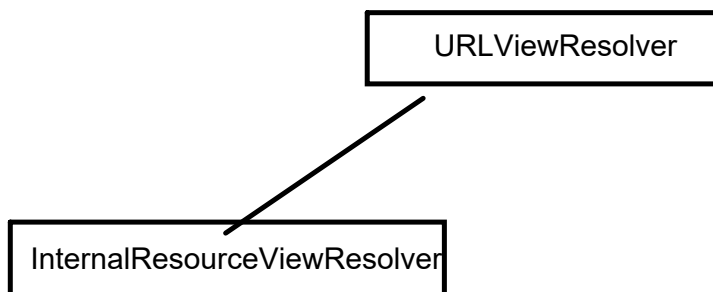
not configured to use jsp-jstl view templates

Spring boot is by default configured to use :

thymeleaf

```
> java -jar bootapp.jar server.port=9091
```

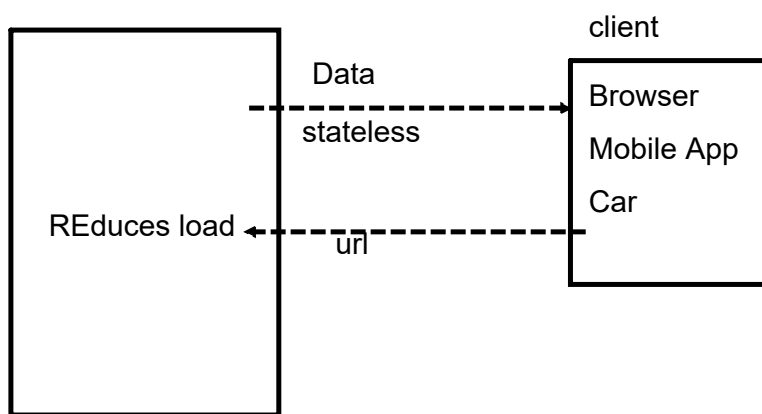
InternalResourceViewResolver (





## REST-WS

REpresentational State Transfer



1. how to generate a request ?
2. what is format in which data will arrive?

Request, REST :

=>purely the URL

=>Conventions for URL

=>ALL HTTP VERBS (intention)

eg :POST : add some

PUT : edit

SOAP / WSDL : programmatic request

REceiving data:

standard, simple as possible

JSON,XML,HTML,TEXT

Allowed to explore the concept of micro-service architecture

JAX-RS (specification)

Jersey

Restlet

RESTEasy

Apache CXF

Spring :

not a JAX-RS implementation

### @RestController

1. does auto : interconversion of JSON<->JAVA (jackson-databind project)
2. DEALS with Request /Response

Request Object



Response Object

header
content
status code

Jackson - databind project :

uses the getter/setter method for interconversion

lombok project

Convention:

Employee

/api/employees	GET	: asking for all employee records (/api/get-all-records)
/api/employees/{id}	GET	: asking for a single emp record with id : {id}
/api/employees	POST	: a record is submitted (add)
/api/employees	PUT	: a record is submitted (update)
/api/employees/{id}	DELETE	: delete a record with id : {id}
/api/employess	DELETE	: a array of id is submitted
/api/employees/{id1}/{id2}	DELETE	

spring-data-rest

## Actuators

### Microservice architecture

monolith :

Interdependency

Fragile in nature

deployment :

usage of resources

bound to specific technology

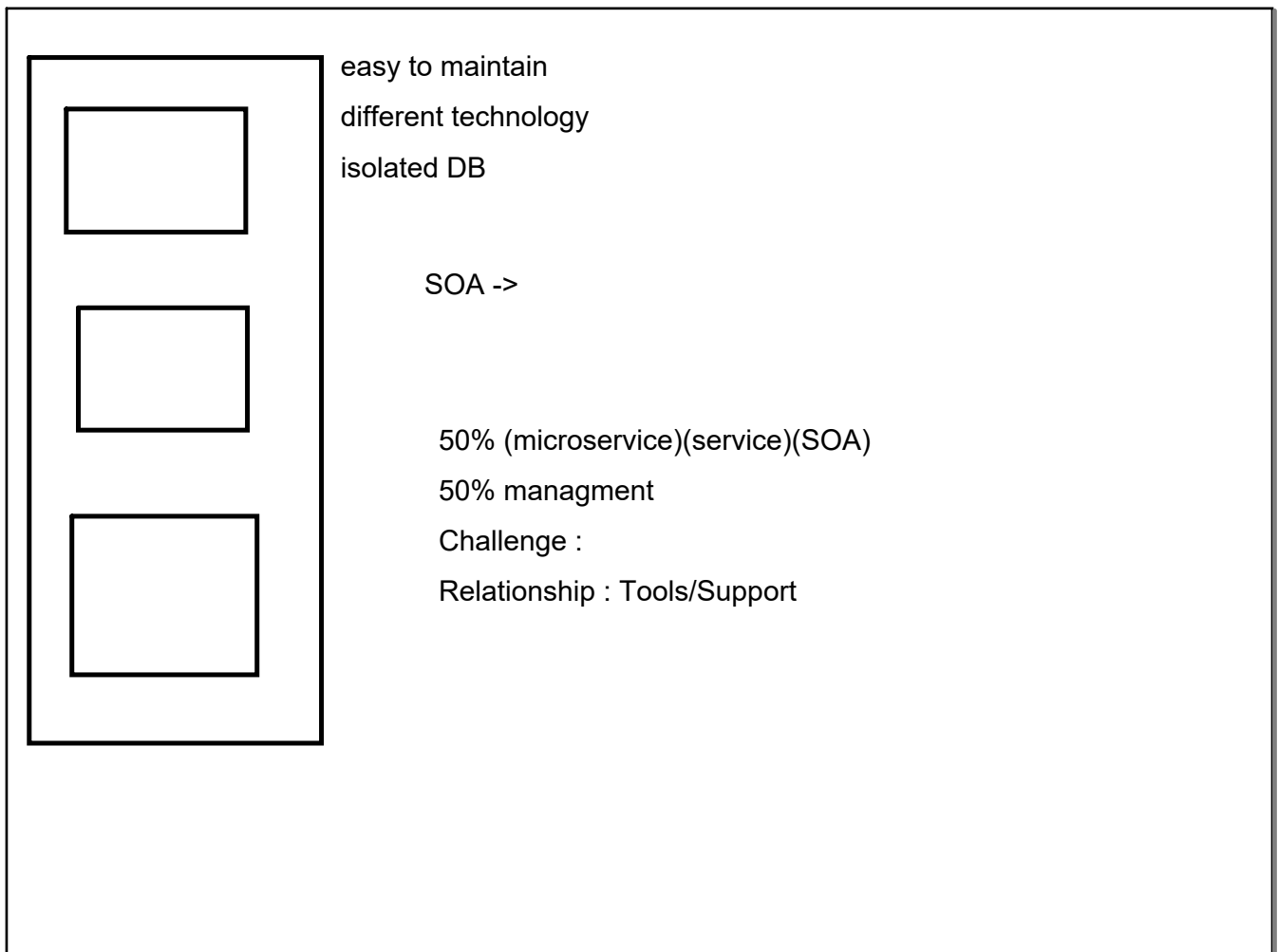
team division / management

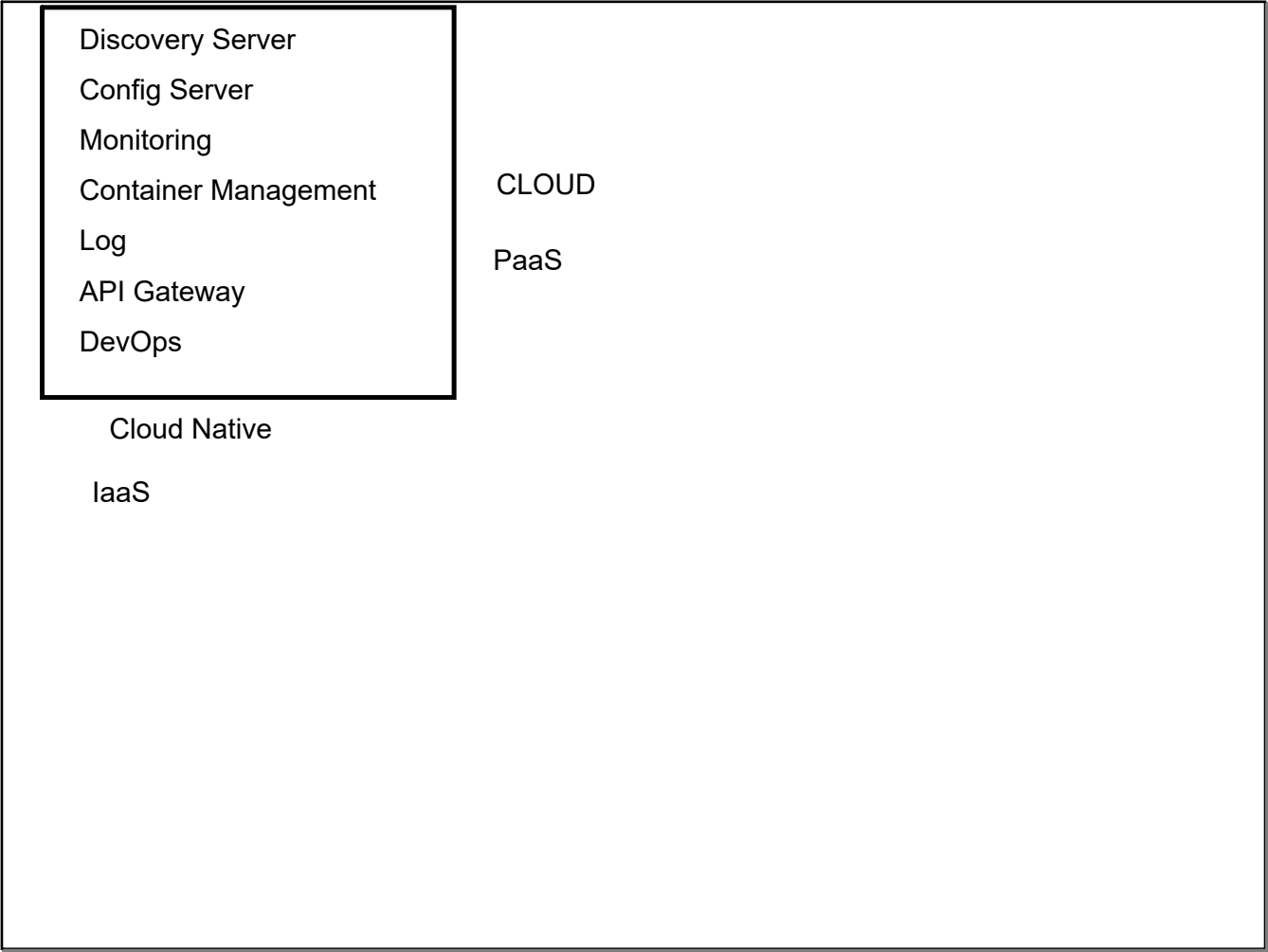
new team member inclusion

---

1. does not easily integrate/comply agile

2. CI/CD implementation is a challenge





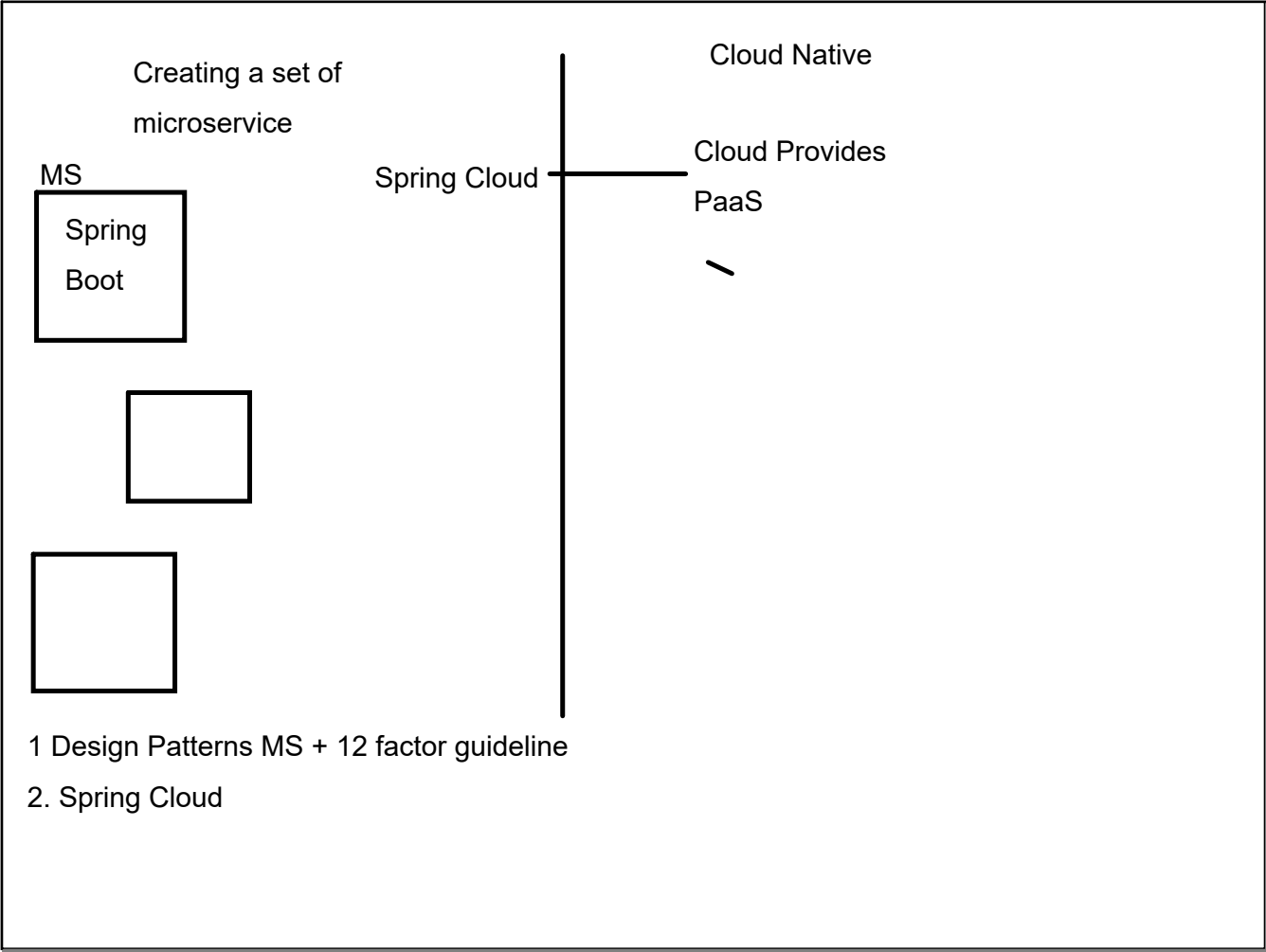


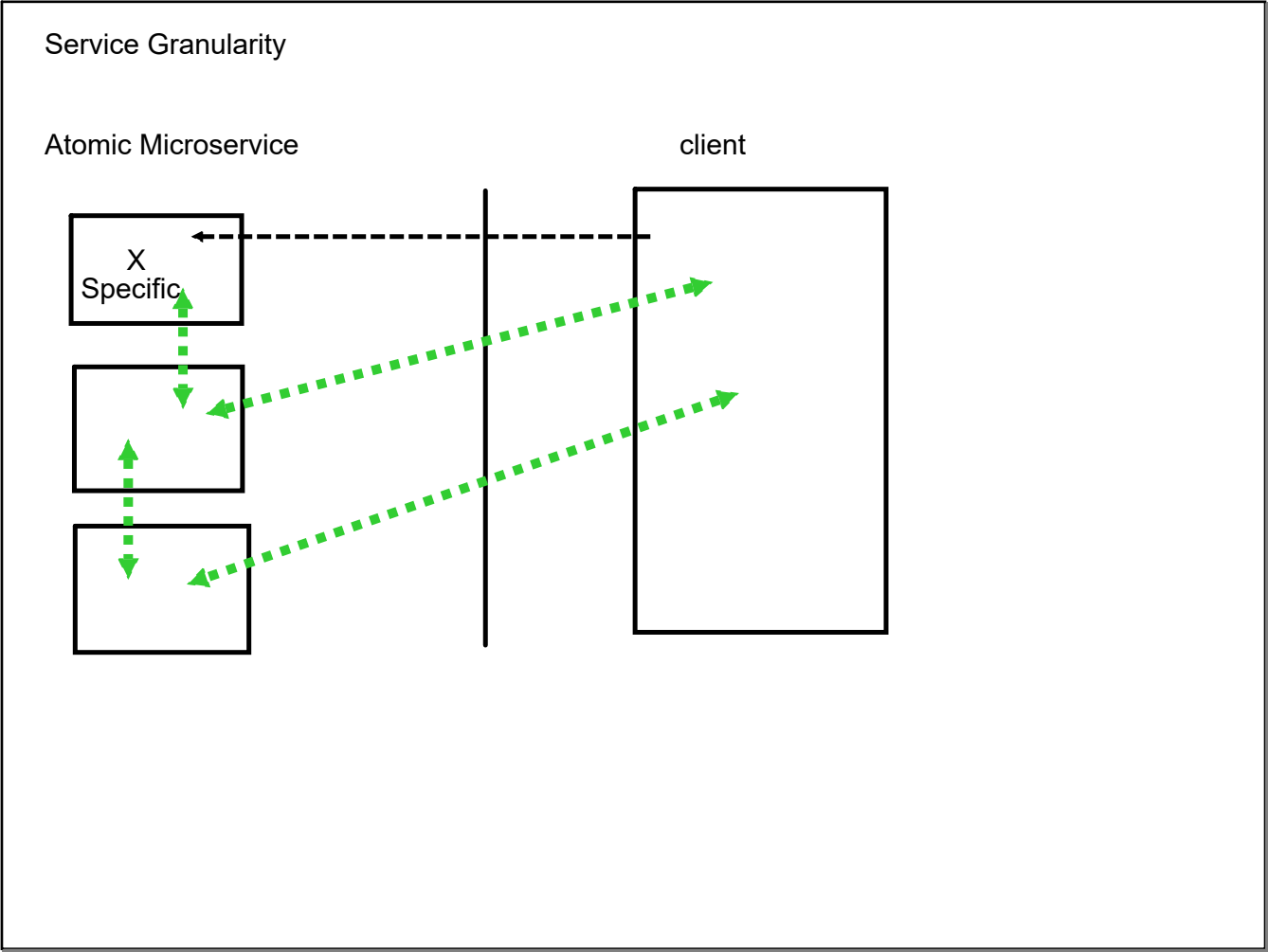
Spring boot :

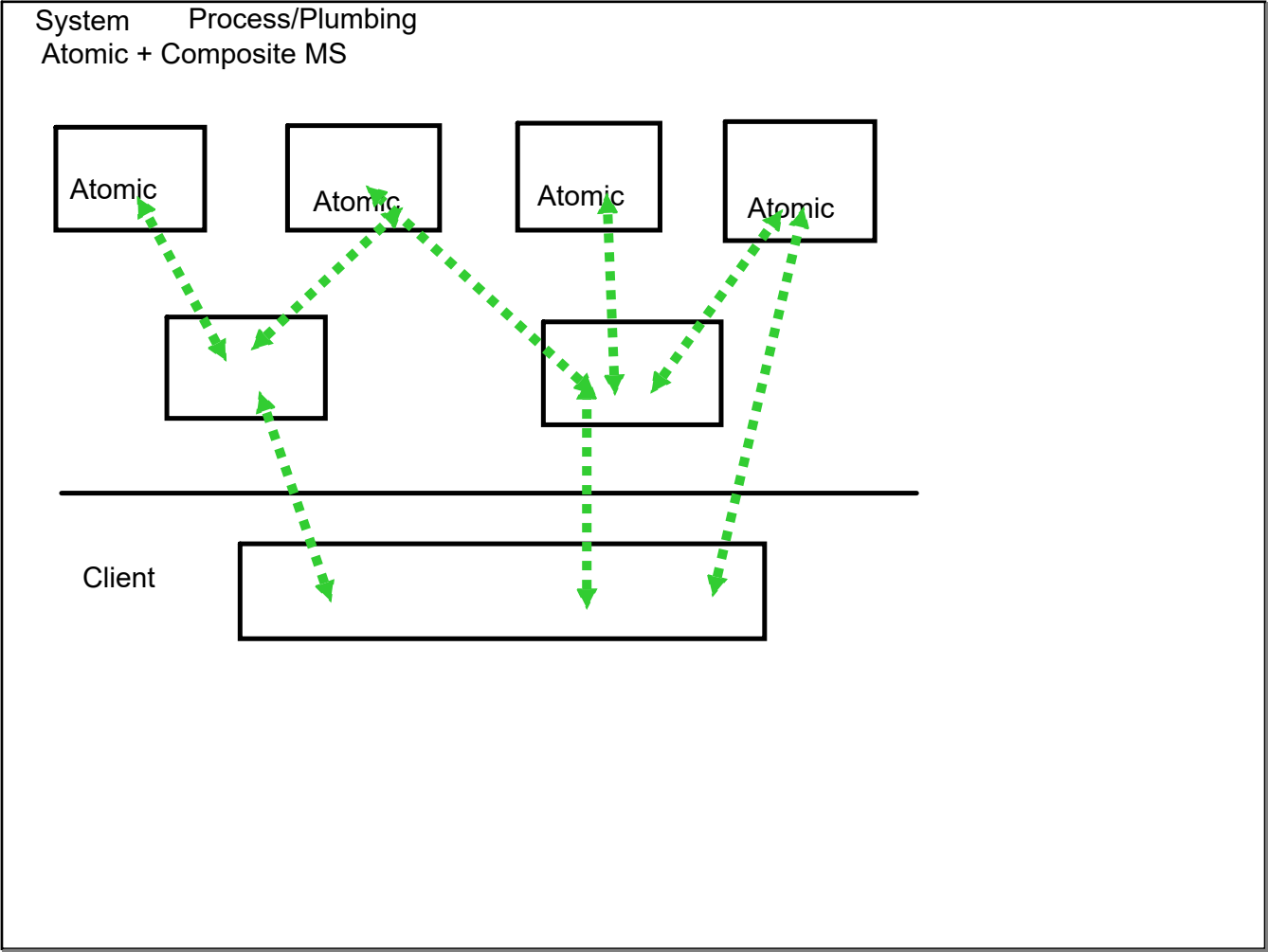
Spring / Spring MVC

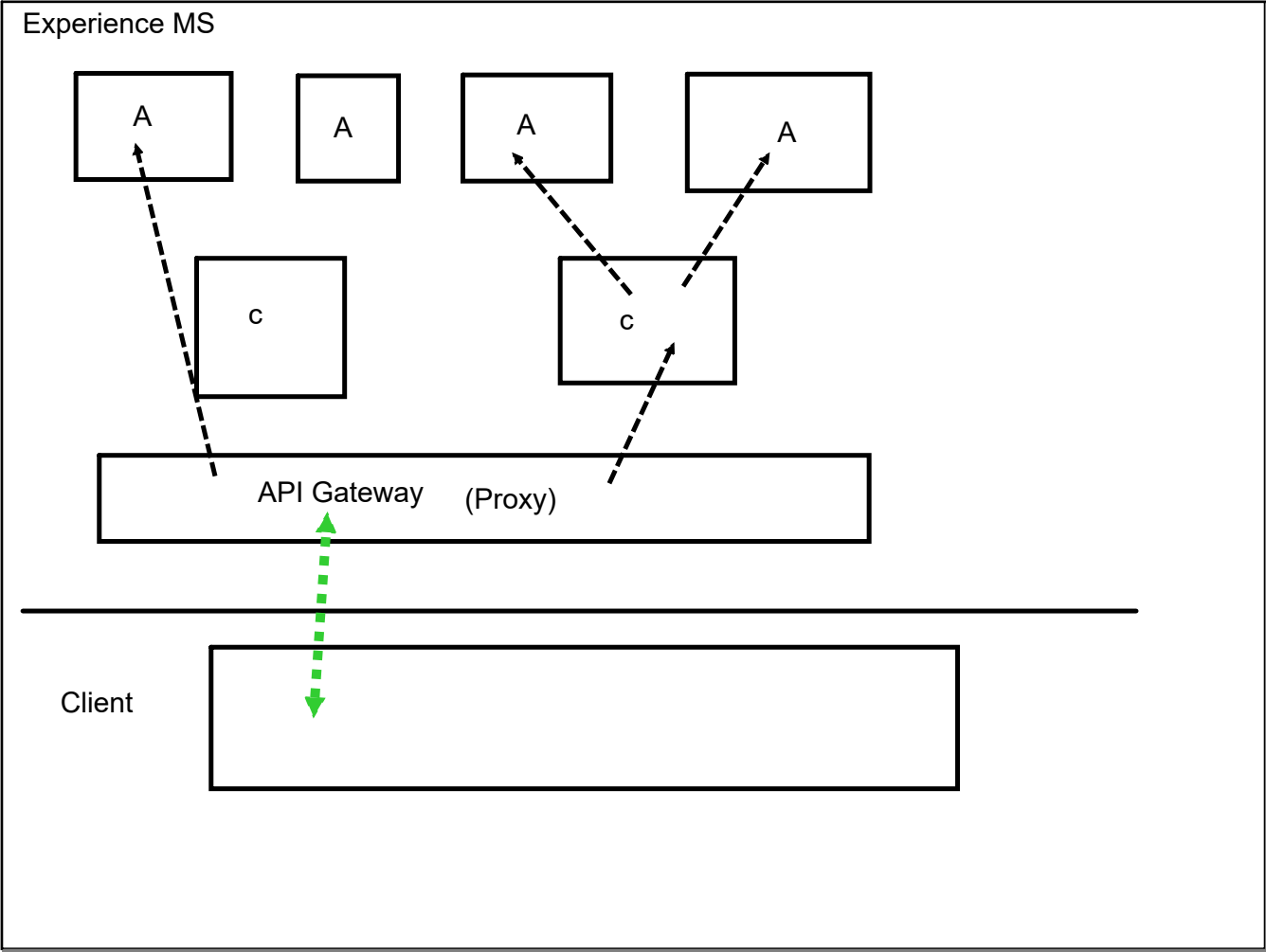
Most resonable default

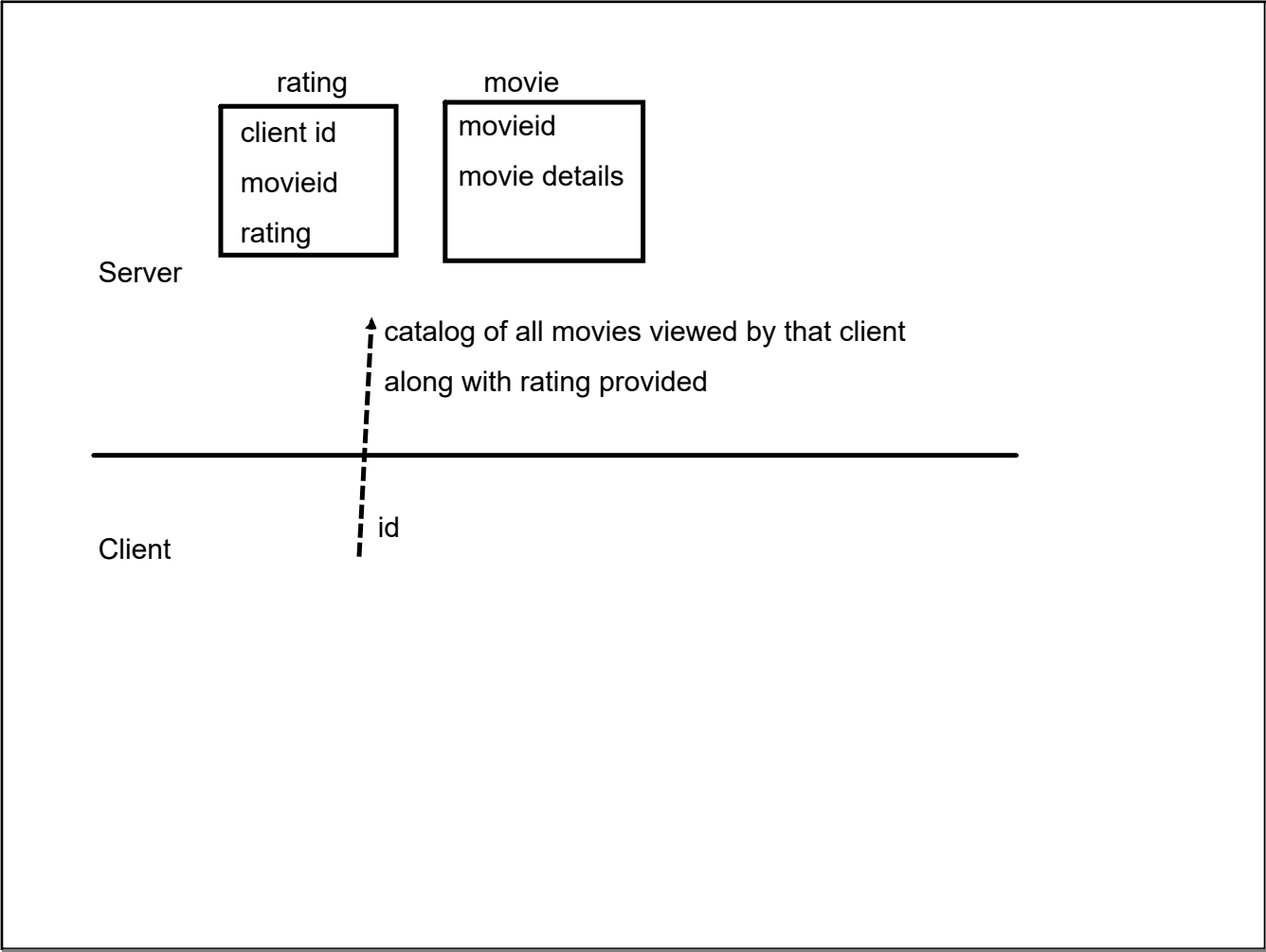
Integrates Spring cloud out of the box

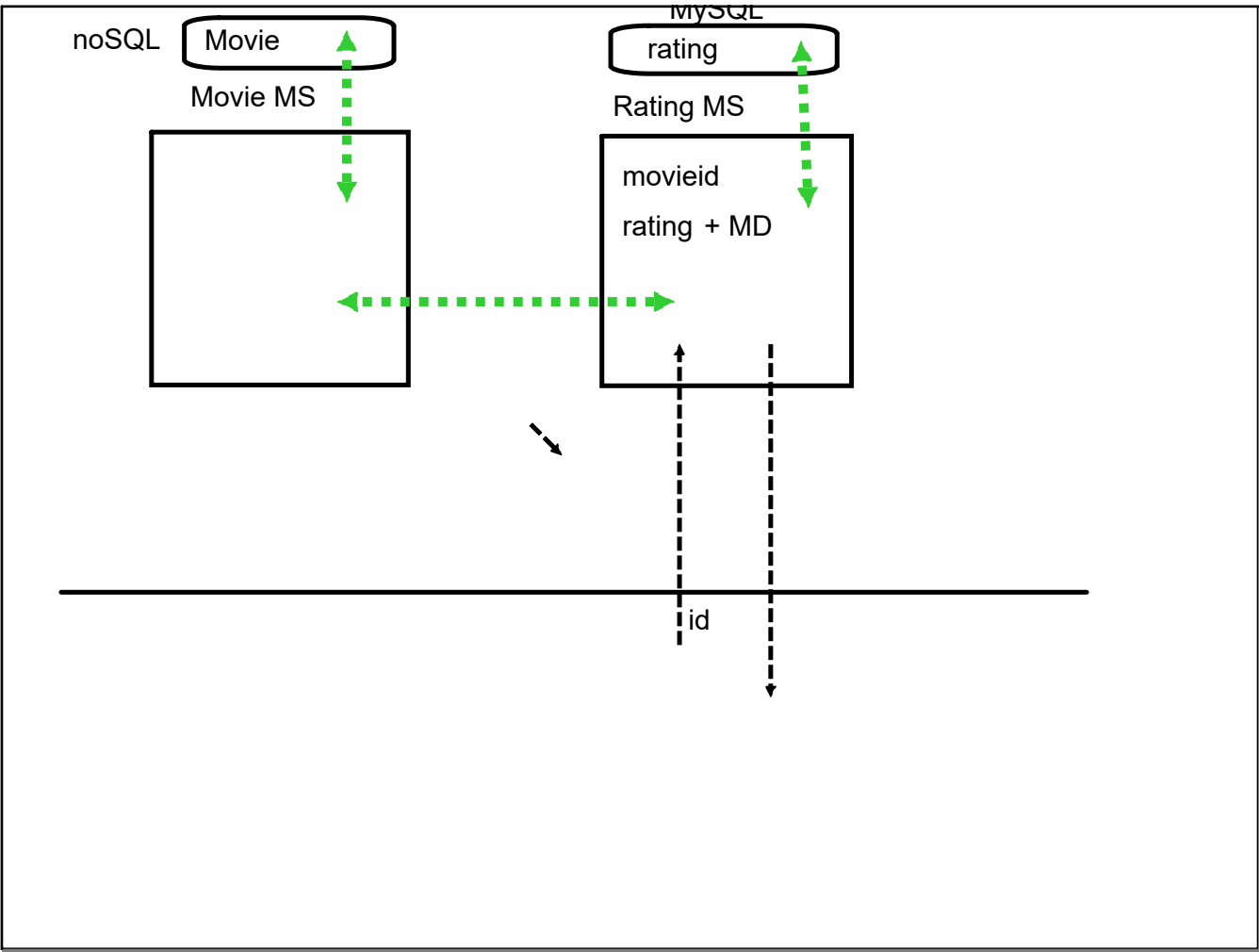


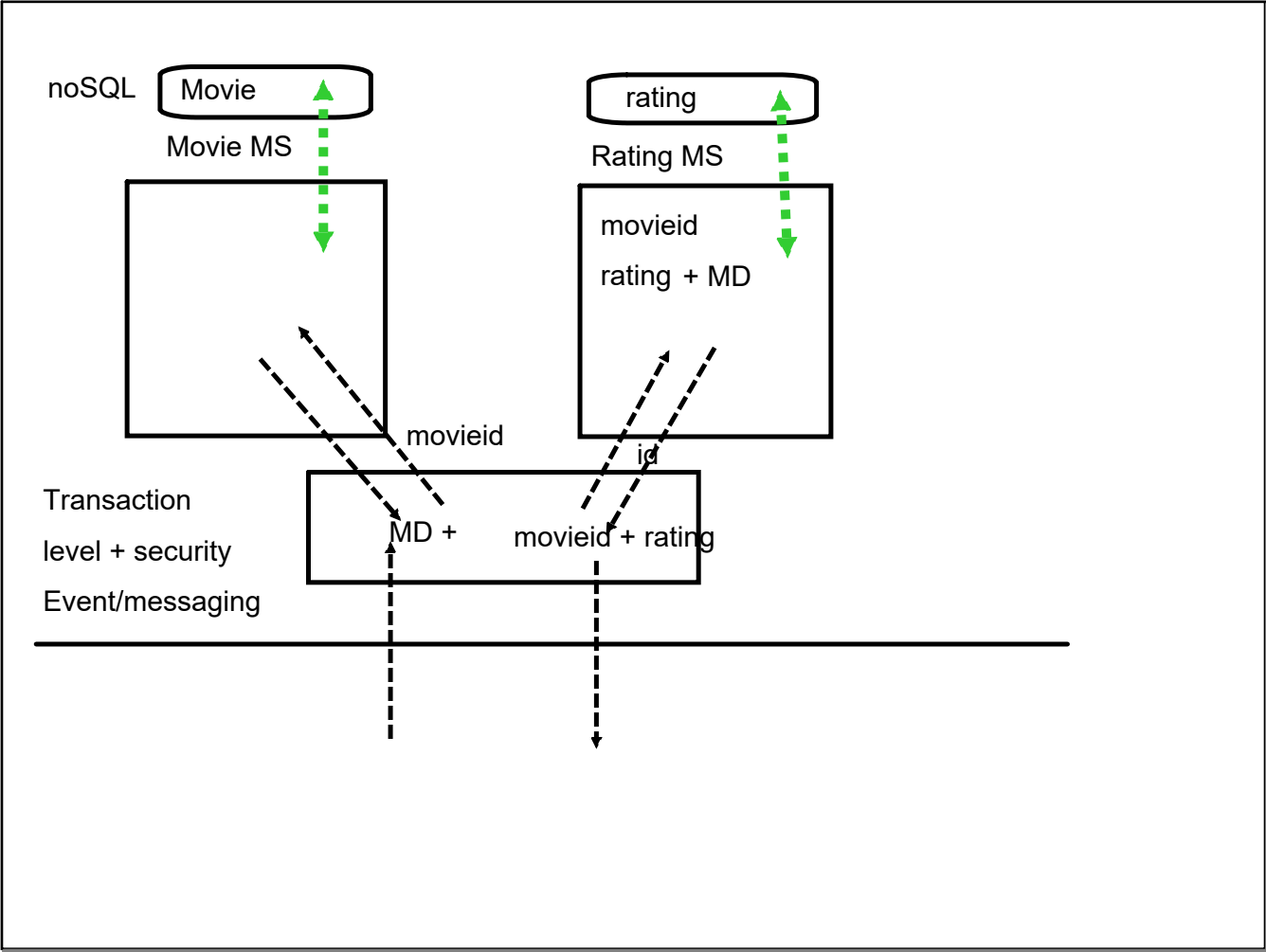














12-factor guideline

Lightweight

Reactive

Stateless

Atomic

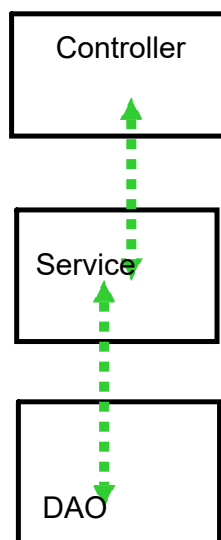
Externalized

Consistent

Resilient

Good Citizens

Versioned     x.x.x



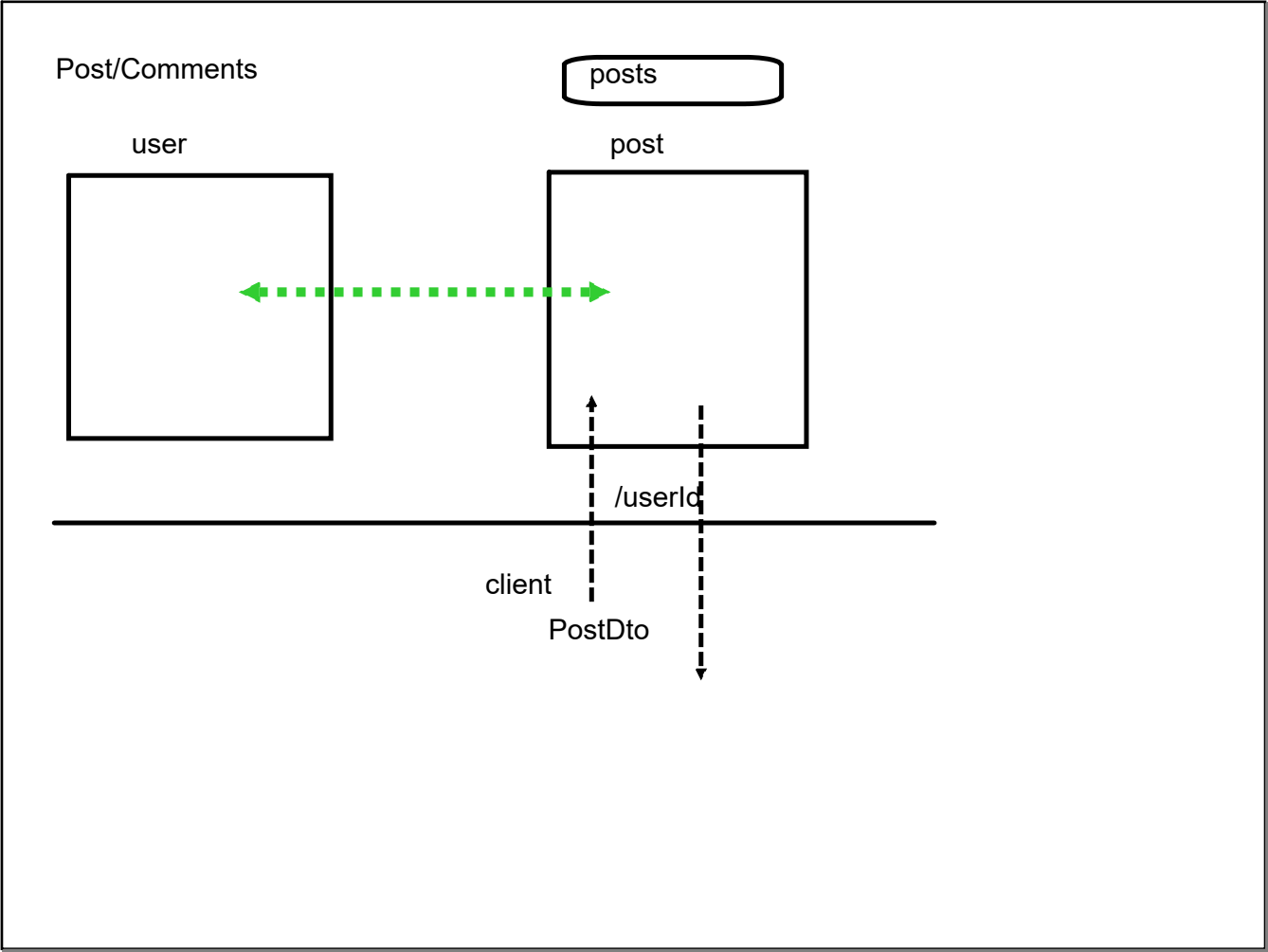
Spring-Data (persistent API)

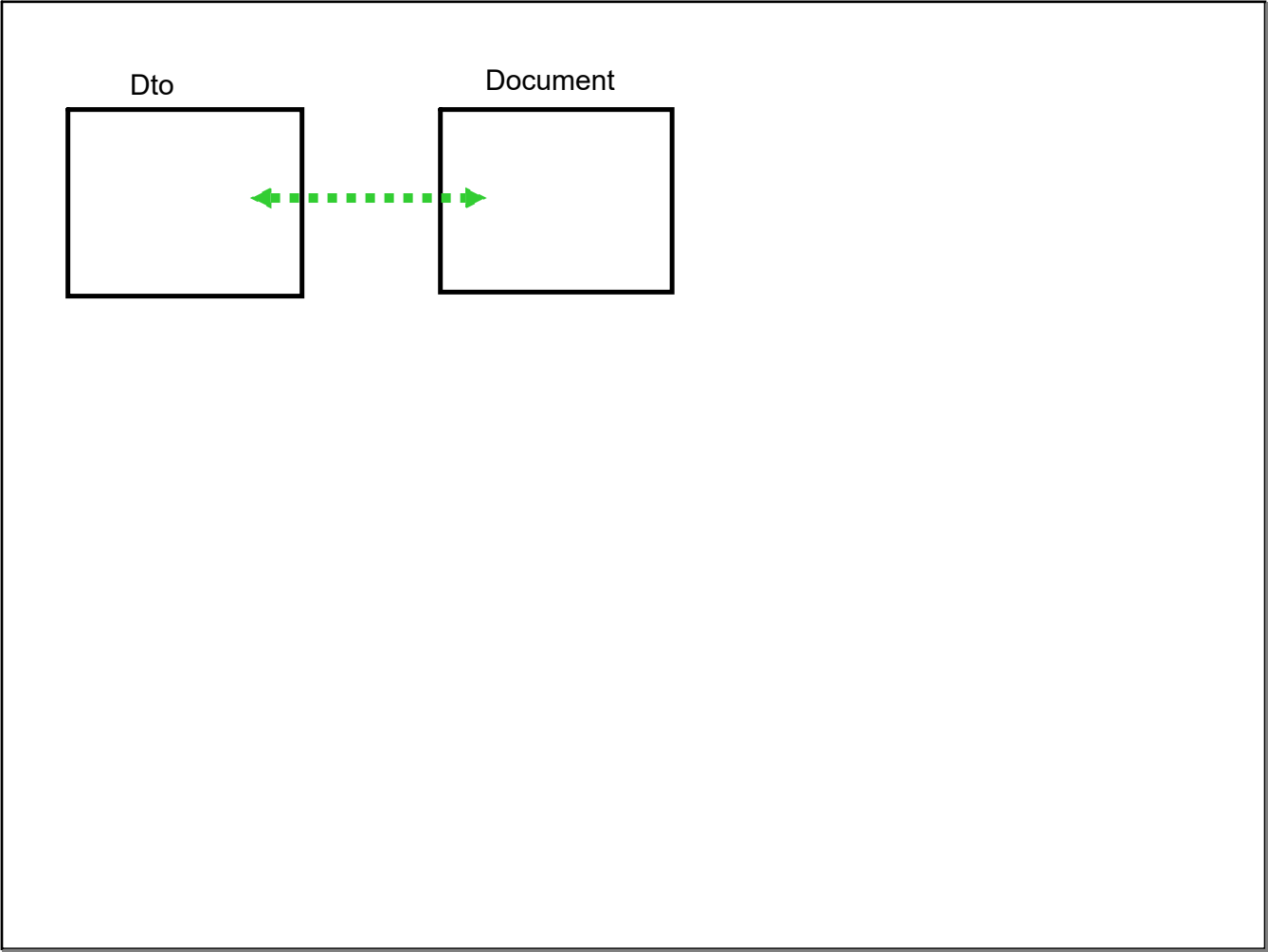
: Mysql : JPA

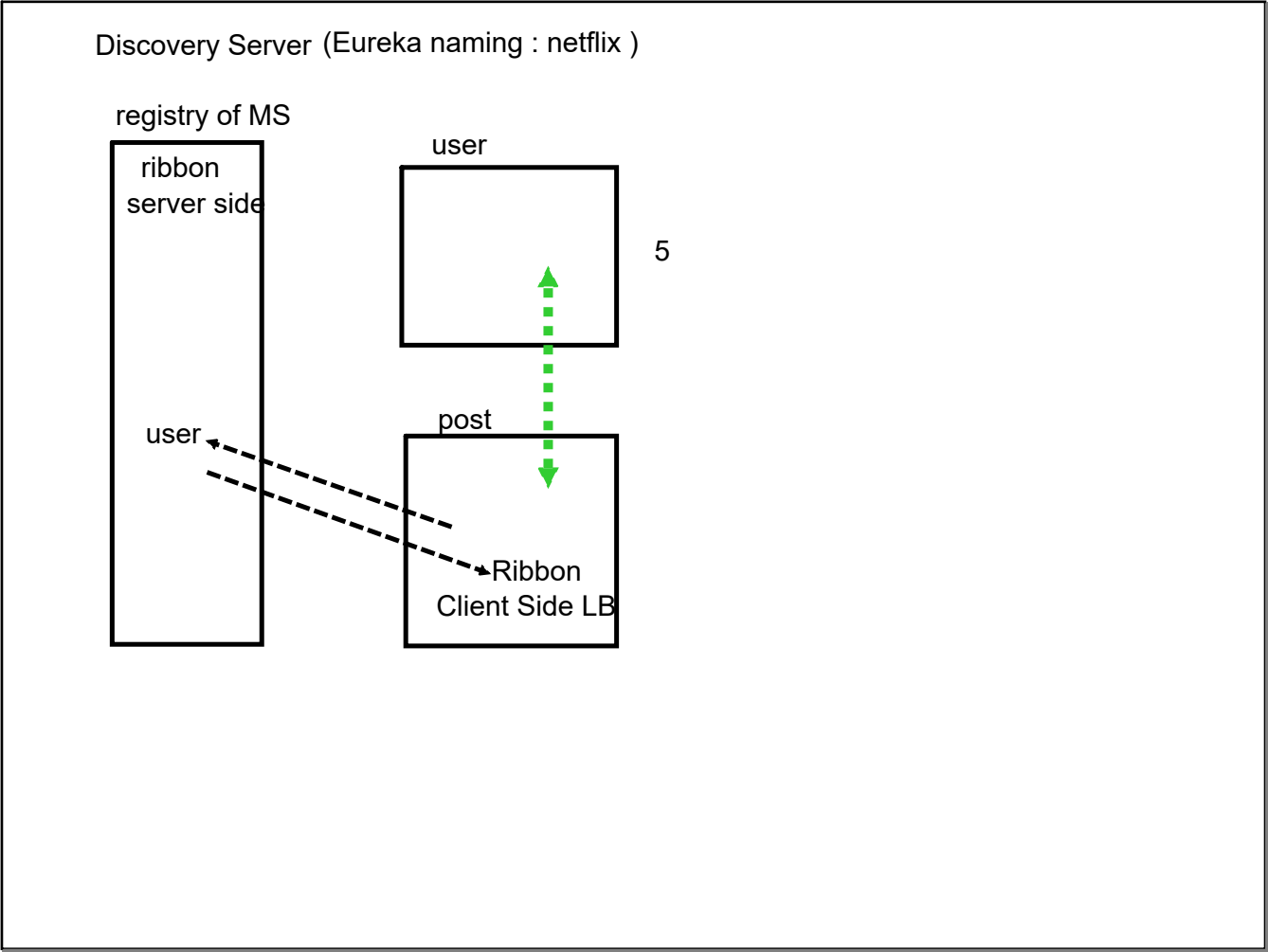
: Mongo-DB impl

=> Lots of pre-built DB interaction

=> Add custom method : implementation provided on the fly  
proper naming convention







HTML : STRUCTURE

CSS : PRESENTATION

Javascript : BEHAVIOR

HTML-5

- # backward compatibility
- # Standardized the error handling
- # New Semantics Elements
- # Built-in API
- # audio/video

Semantically correct Structural elements

traditional : <p>,<span>,<div>

article,

section

aside,

header,

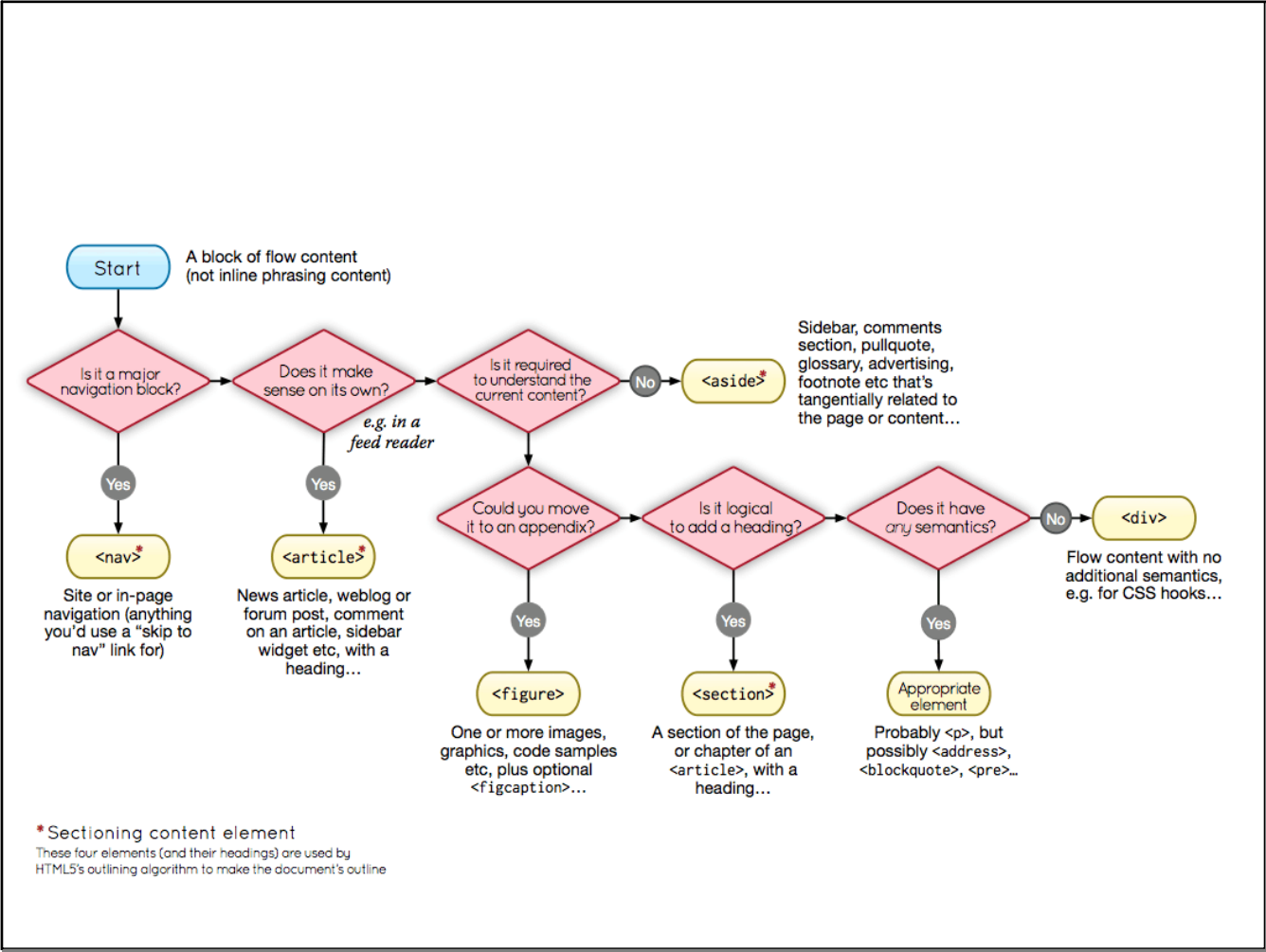
footer

OUTLINE ALGORITHM :

Smooth rendering

Assisstive technology

Search EO





**Categorization**

Metadata

Metadata

Interactive

HEading

Phrasing

Sectioning

**API**

DOM spec is part of HTML5

Built in APIs (internal activities)

A/V Api

Offline Application API

History API

WEb Protocol API

Drag n Drop API

Geolocation

2D Canvas

Local Storage

Session Storage

Messaging API

Local DB API

## CSS

### Cascade Style Sheet

SS : presentation rul

Cascading : rule for resolving conflicts with multiple SS applied on same elements

Browser SS

external

internal

inline

Location of style

Heirarchy of HTML

Assembly of properties

selector { property : value}  
p{color: #FFFFFF}

Cascading rule  
Specificity

Bootstrap + JS

element  
# (ID)  
. Class

BootStrap : mobile first CSS library ( device independent)

Grid System : use 5 grid breakpoint

extra small : <567 px : col (auto layout)

small 567-768 col-sm

medium 768-992 col-md

large 992-1200 col-lg

xl > 1200 col-xl

Vanilla JS ( Ecma Script )

ES - 5

ES-5 > (external javascript file)

transpilers (babel, traceur)

ES-5

```
graph TD; A[ES-5 > (external javascript file)] -.->|transpilers (babel, traceur)| B[ES-5]
```

Javascript (rich set of library allowed to integrate)

Objects

=> HTML/CSS code ( DOM Tree)

=> Browser window

=> history

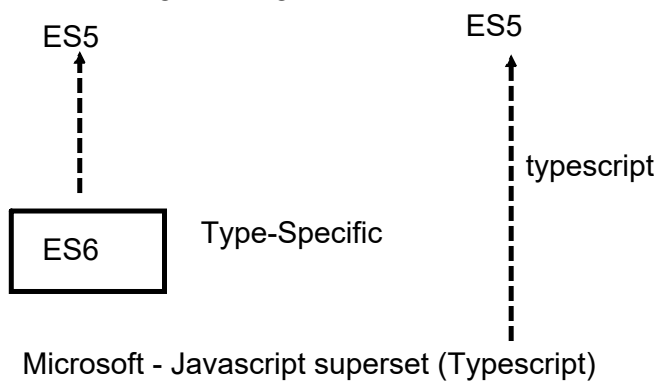
=> navigation

ES6

JS : making HTML page more dynamic/responsive/interactive

Server based logic --> Client (Outsource)

New Programming constructs



```
function add(num1, num2){  
    return num1 + num2;  
}
```

named Type  
interface/generic

```
add(20,30); 50
```

```
add('20','30'); 2030
```



```
npm
```

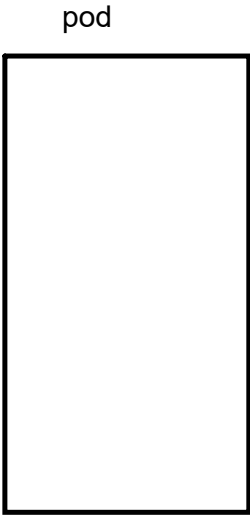
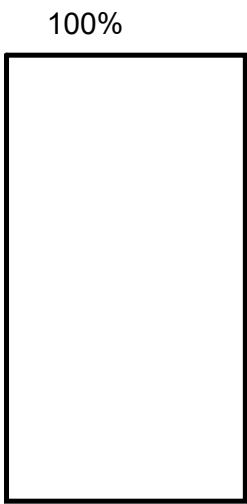
```
npm install -g typescript
```

```
var button = document.querySelector("button");  
var input1 = document.getElementById("num1");  
var input2 = document.getElementById("num2");
```

```
function add(num1, num2){  
    return num1 + num2;  
}
```

```
button.addEventListener("click", function(){  
    console.log(add(+input1.value, +input2.value));  
});
```

=>Proper division  
=>Management tools



Inter-microservice :

Discovery Server/Client

feign-client

load balancing

externalization :

config

security

single point access

CQRS : Transaction

Event Driven - Messaging API

-----

Fault Tolerance

Tracing

Documentation Swagger

-----

Kubernetes

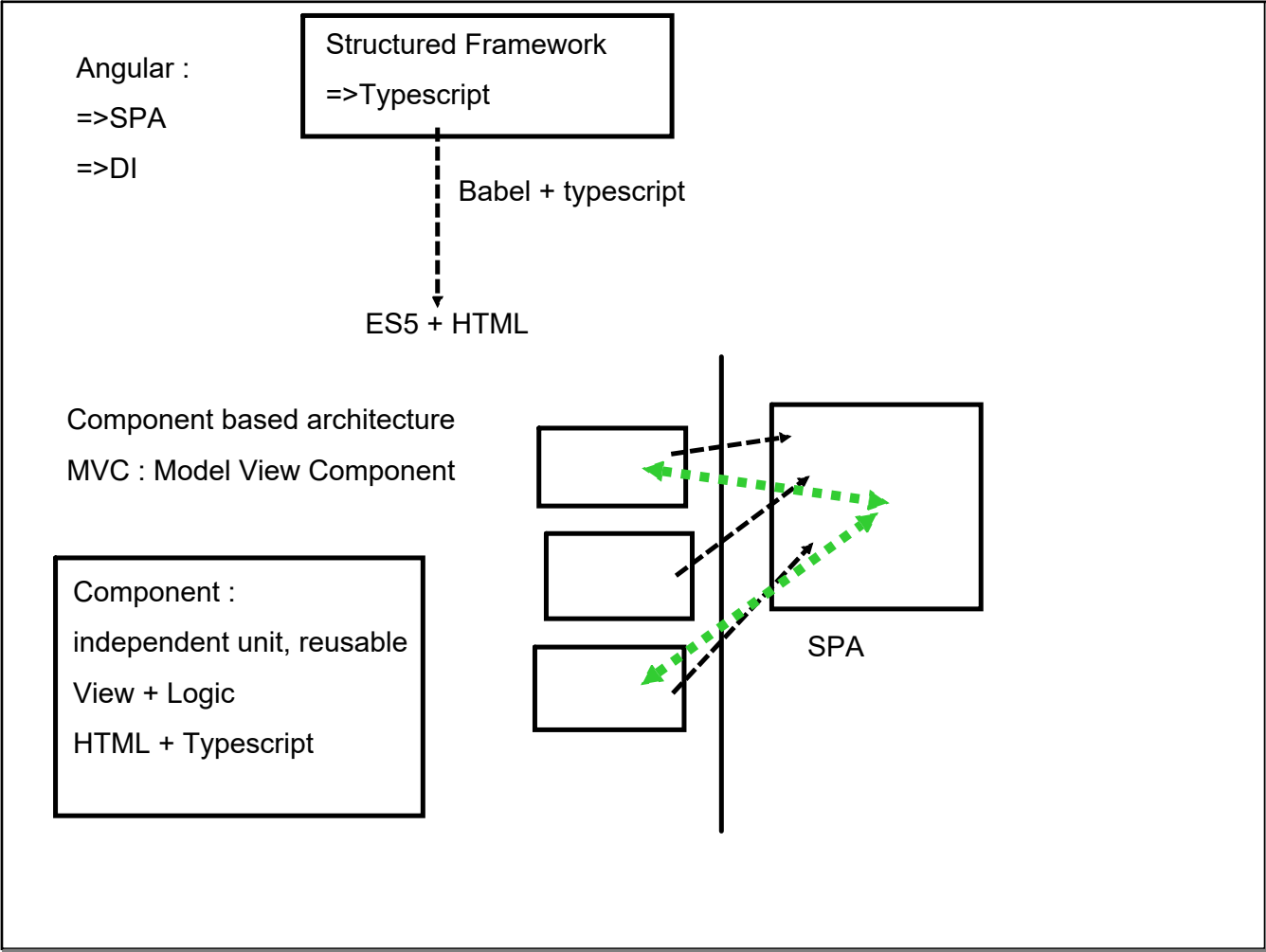
object : JavaScript objects, types of object

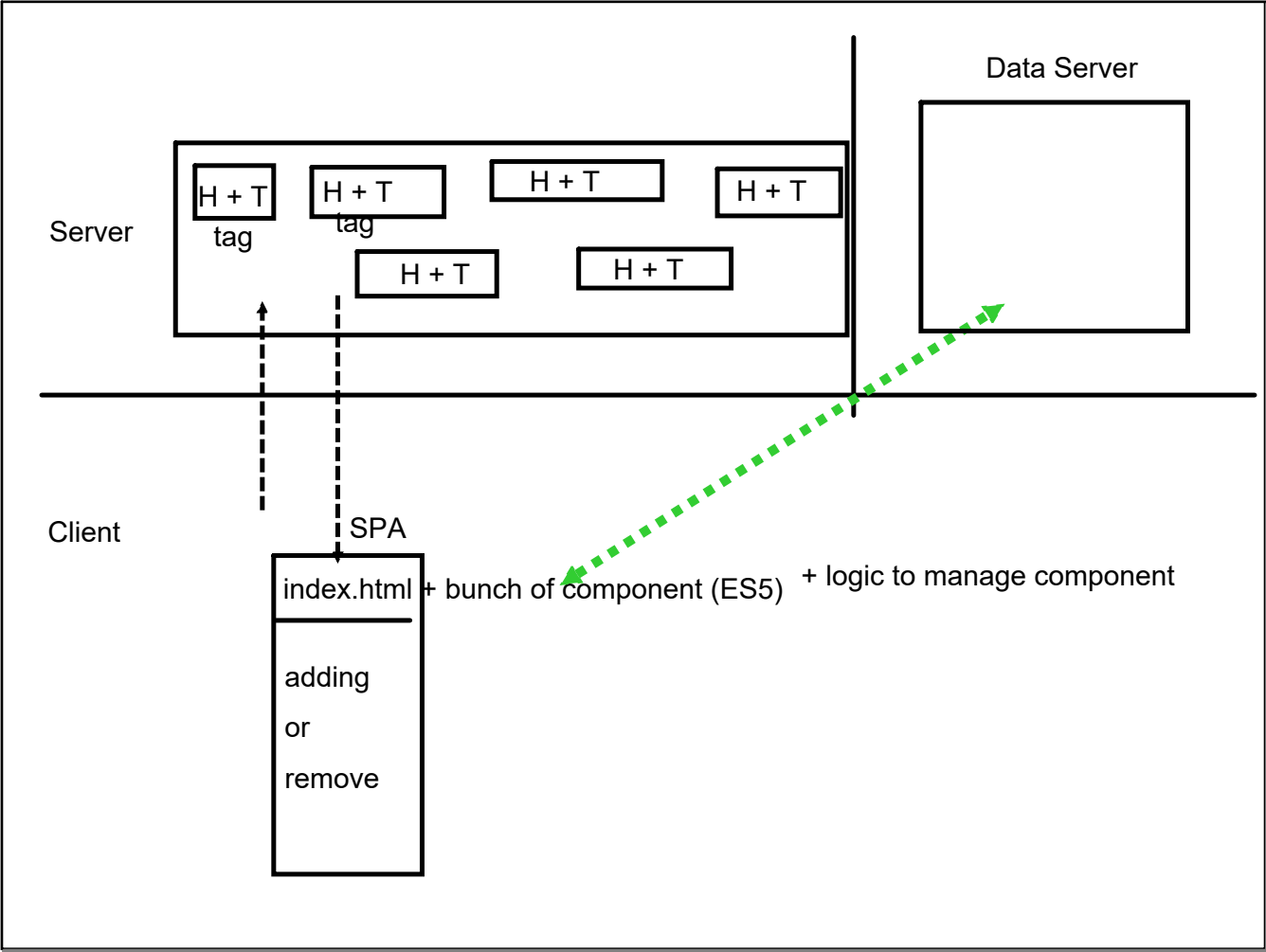
Tuple : fixed size, ordered array

```
class Base{                                enums
    constructor(name : string){
    }
}
```

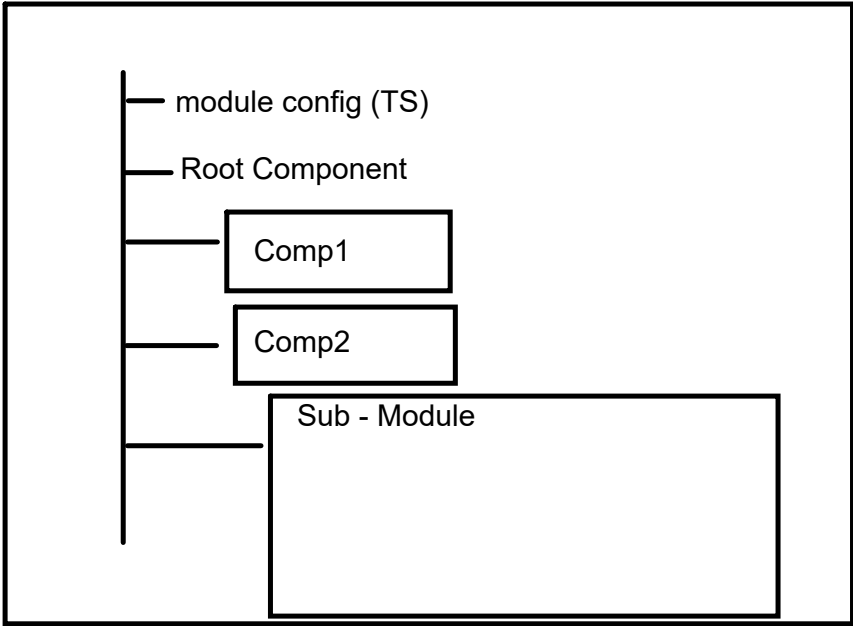
```
class Derived extends Base{
}
```

```
const der = new Derived("");
```





Modules (Packages)



NodeJS (npm)

# to download and install angular CLI

=>npm install -g @angular/cli

# Create a new Angular Application

=>ng new <app-name>

All npm :

package.json

npm install



Component :

=> TS class : config

=> presentation : HTML

=> CSS

=> TS class : unit test code

# Creating new component

ng generate component <component>

Entry Form : ( Add New Product)

<Collection>

List of Product

- 1.. Single Component (New Data Entry, holding the collection, show the list)
2. Two Component (New Data Entry, holding the collection | show the list)
3. Three Component (New Data Entry | holding the collection | show the list)

show the list

show the detail of one product

-> show only image

-> show reviews

-> show purchase (cost + add to cart)

Directives : ngFor

Services : shared resources ( functionality/algo/data)

Pipes | Customize/Transform the output

Angular directive :

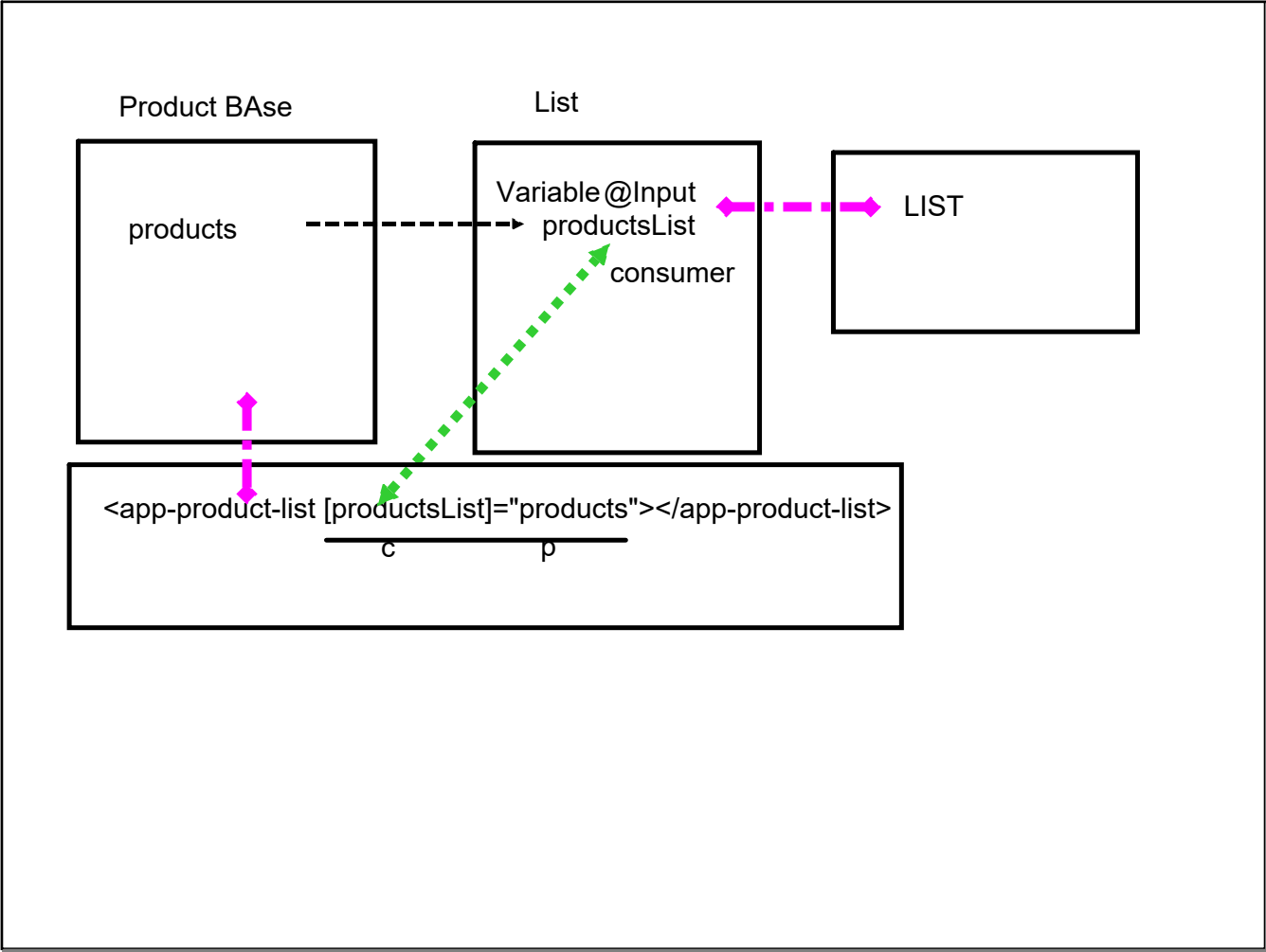
helps to add simple programmatic constructs directly into HTML  
majorly as attributes of existing HTML Components

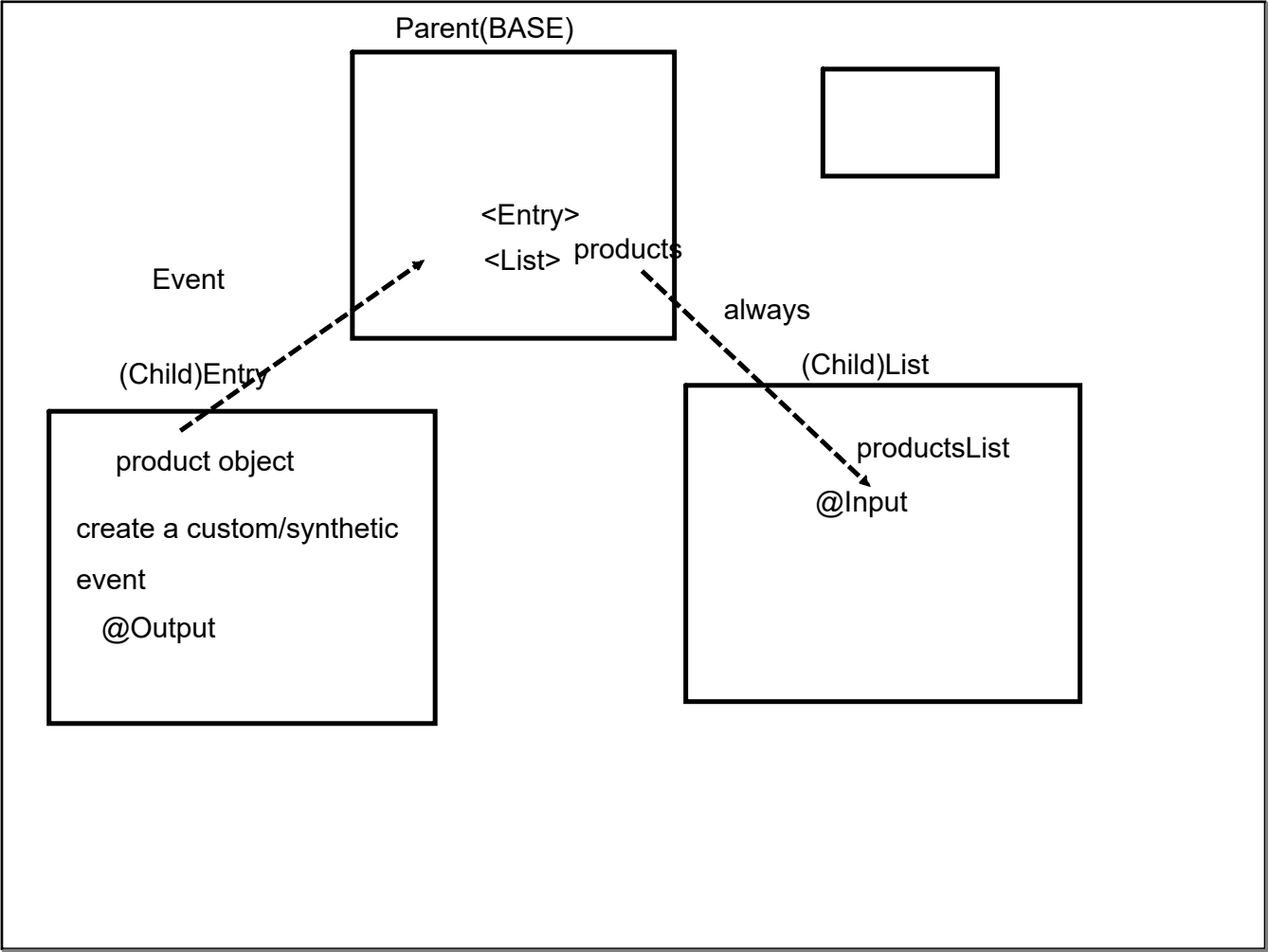
eg : FOR LOOP Directive

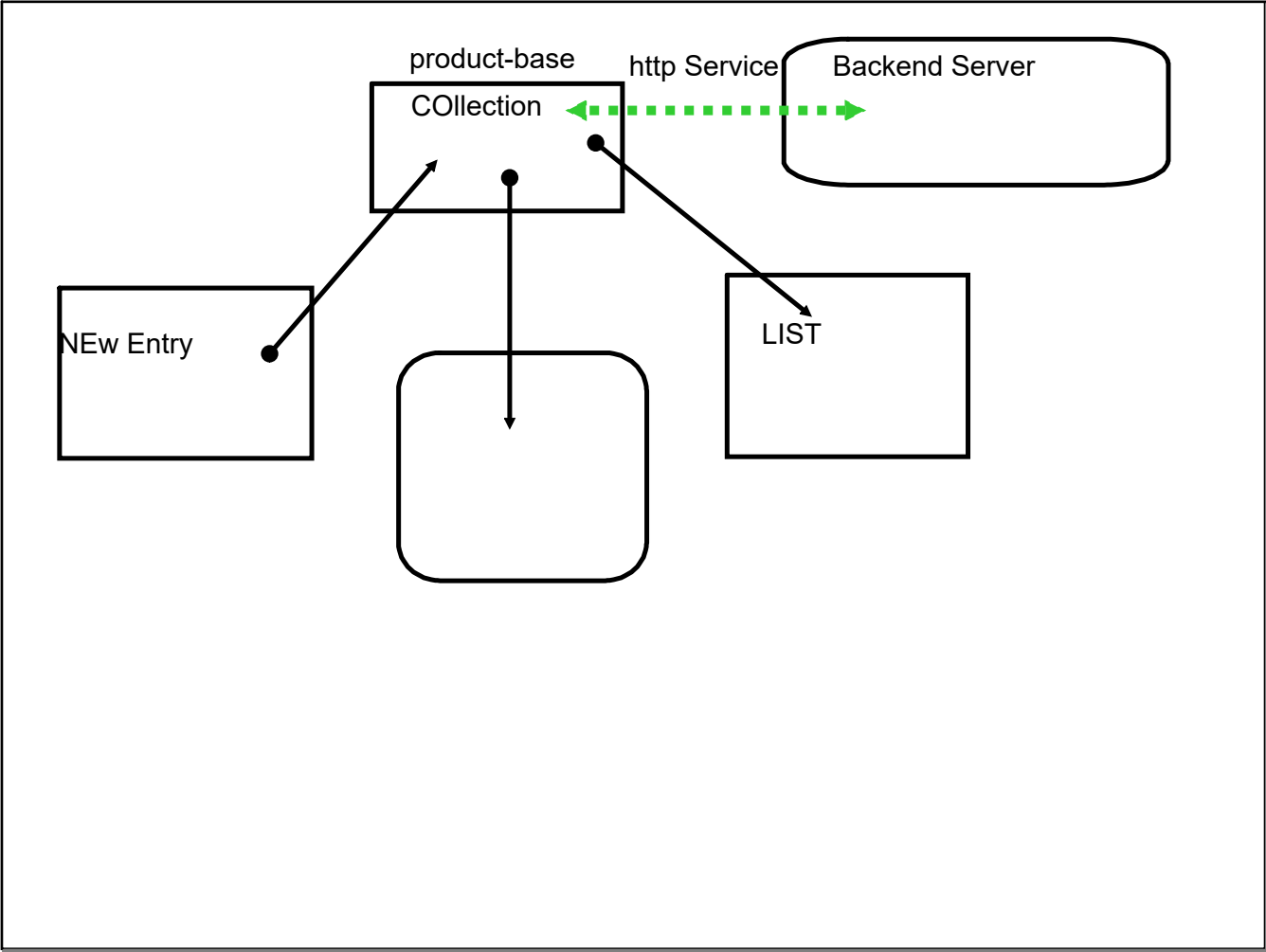
```
{{<var>}}
```

Form

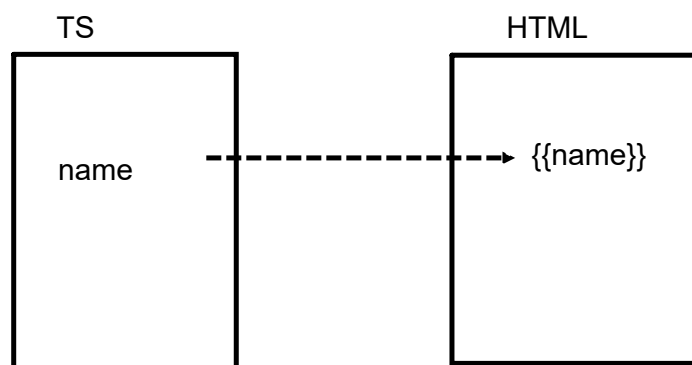
```
<form></form>
```











events : NO HTML EVENTS

events : synthetic events ( custom events provided by ANGULAR)

event handling compatible to platform + TS

ngIf

```
<div *ngIf="<condition>">  
  Something....  
</div>
```

condition :

true : div will be visible

false : div will not be visible

```
<div *ngIf="false">
```

Something....

```
</div>
```

not be displayed

```
<div *ngIf="a > b">
```

Something....

```
</div>
```

```
<div *ngIf="fun()">
```

Something....

```
</div>
```

ngSwitch : controls the visibility

\*ngFor

ngStyle

Single CSS property at a time

[style.<css property>] = "value"

Multiple CSS

[ngStyle] = "{font : 'Arial', background-color : bgColor}"

ngClass

Manage the forms :

Specialized Modules

FormsModules

ReactiveFormsModule

Two fundamental Object

FormControl : represents single input field  
value, state, error

FormGroups : represents complete form  
(collection of form controls)

FormsModules (template driven directives)

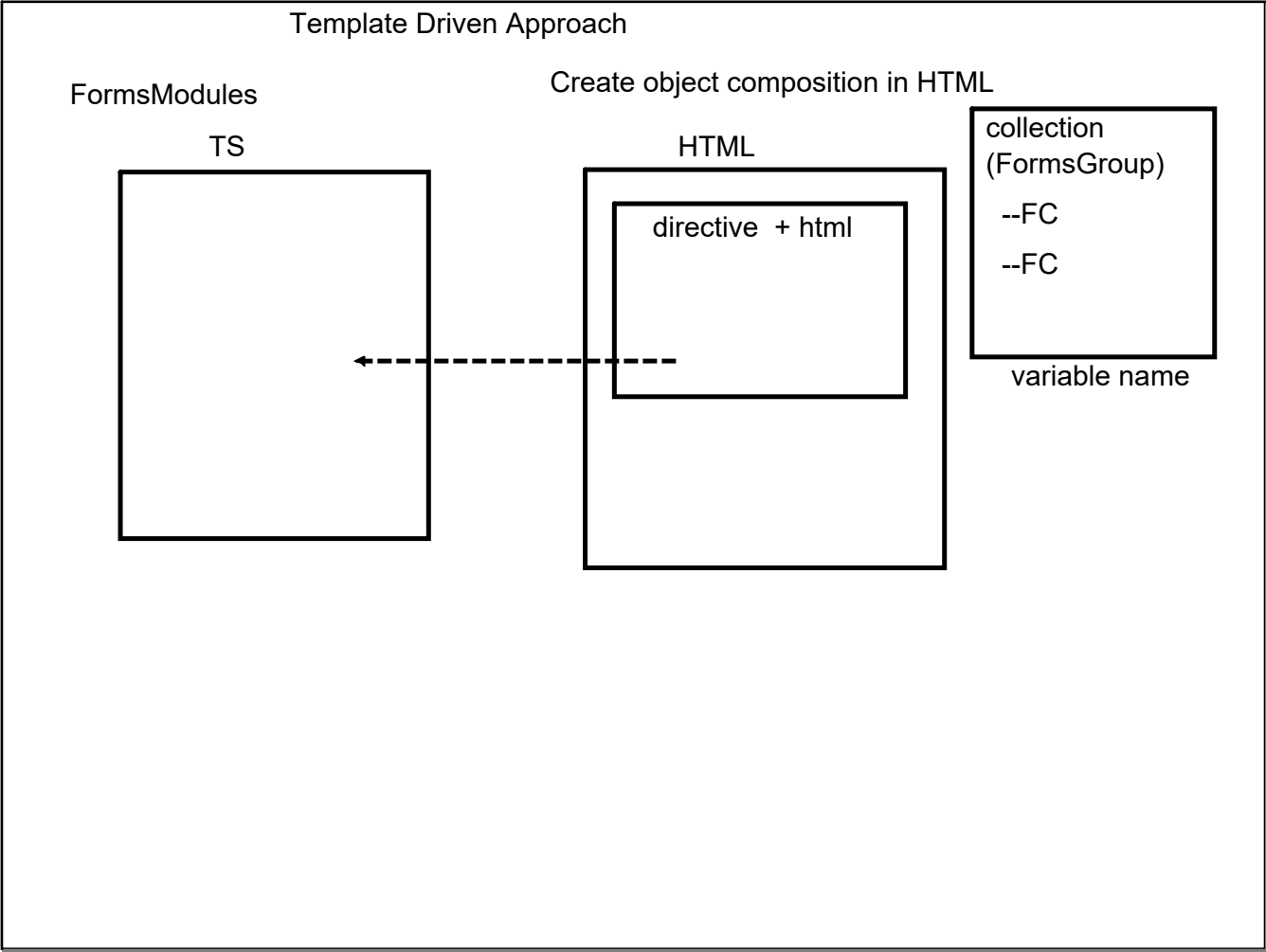
ngModel ~ FormControl

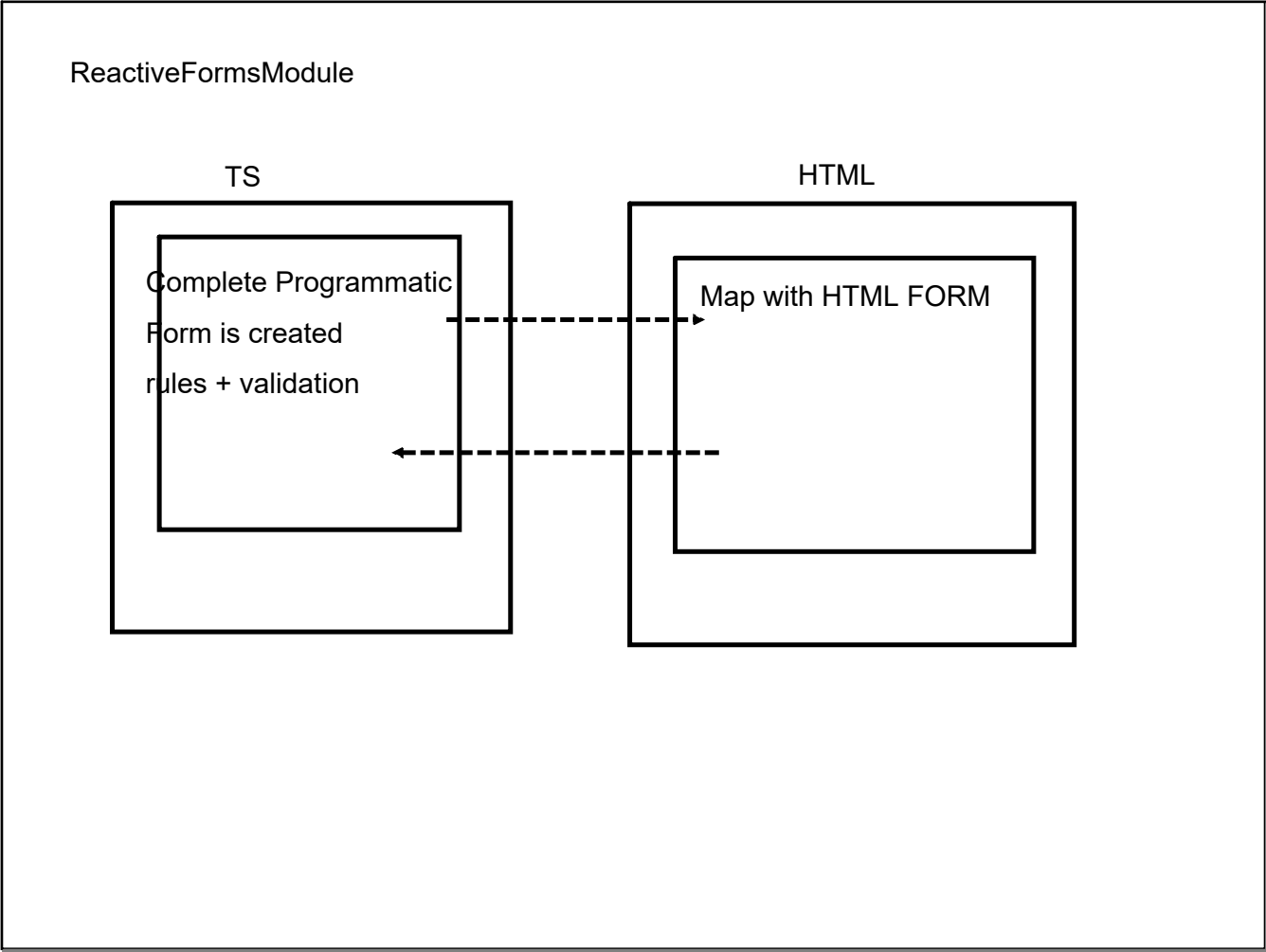
ngForm ~ FormGroups

ReactiveFormsModule (model driven directive)

formControl ~ FormControl

ngFormGroup ~ FormGroups





As Soon as Form dependency

1. auto exposed to directive
2. auto exposed to services

Object of these service are already exposed :

Object are AW/injected in constructors

Routing

Traditional

HOME   CONTACT   ABOUT

a new page gets loaded

Angular

HOME   CONTACT   ABOUT

Search



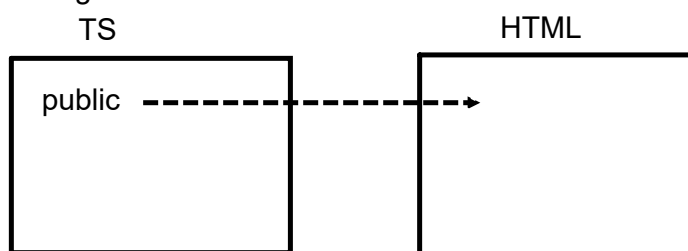
Routing programmatically

Angular components have life cycle : multiple phases  
each phase

=> Interface : method (lifecycle hook method)

OnInit() : as soon as component is loaded

Securing the route: Auth-Guard Service



json-server : dummy backend server

json file

install : npm install -g json-server

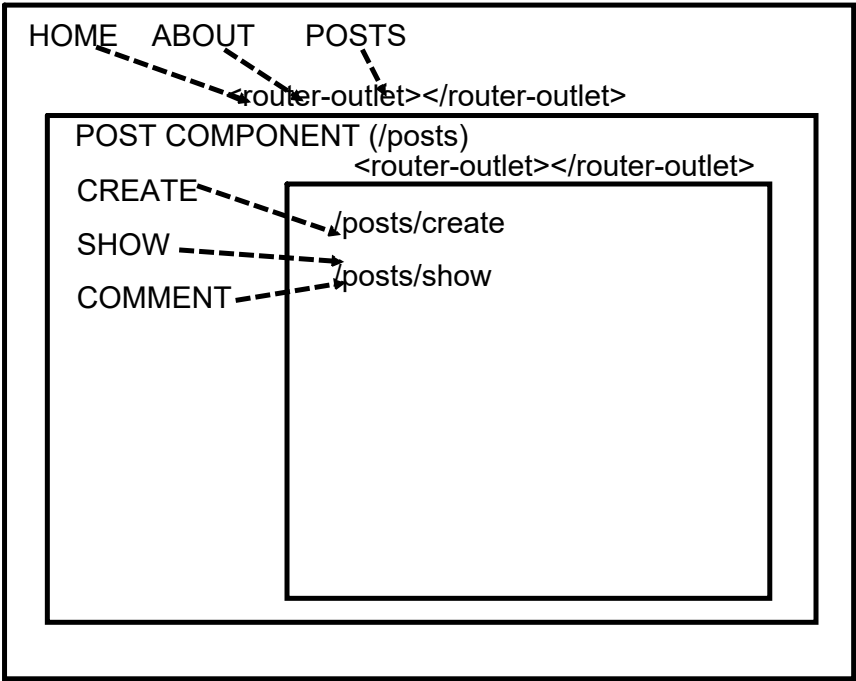
URL : Base url

<http://localhost:3000/products>

1. GET : <http://localhost:3000/products> (fetch all records)
2. GET : <http://localhost:3000/products/1> (fetch by id)
3. POST : <http://localhost:3000/products> (add new records)
4. PUT : <http://localhost:3000/products/1> (id of record to update)
5. DELETE : <http://localhost:3000/products/1> (id of record to delete)

<http://localhost:3000/post>

<http://localhost:3000/comment>



ReactJS : Javascript Lib :

View part : frequently changing data : Modern ES ---> ES5

Component based architecture : syntax approach : HTML + JS (JSX)

: slow rendering

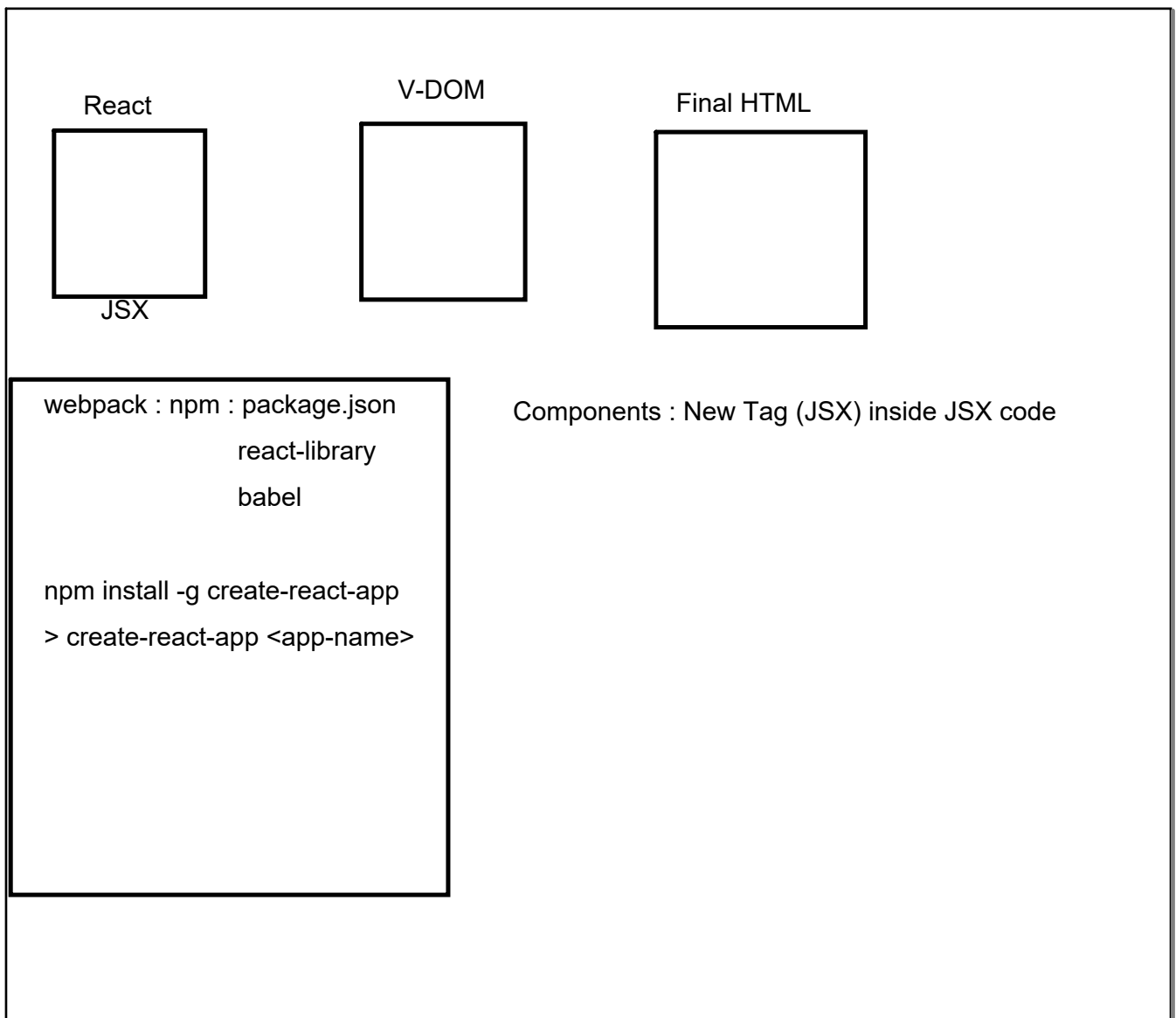
Problem : whenever a new info : DOM heirarchy changes :

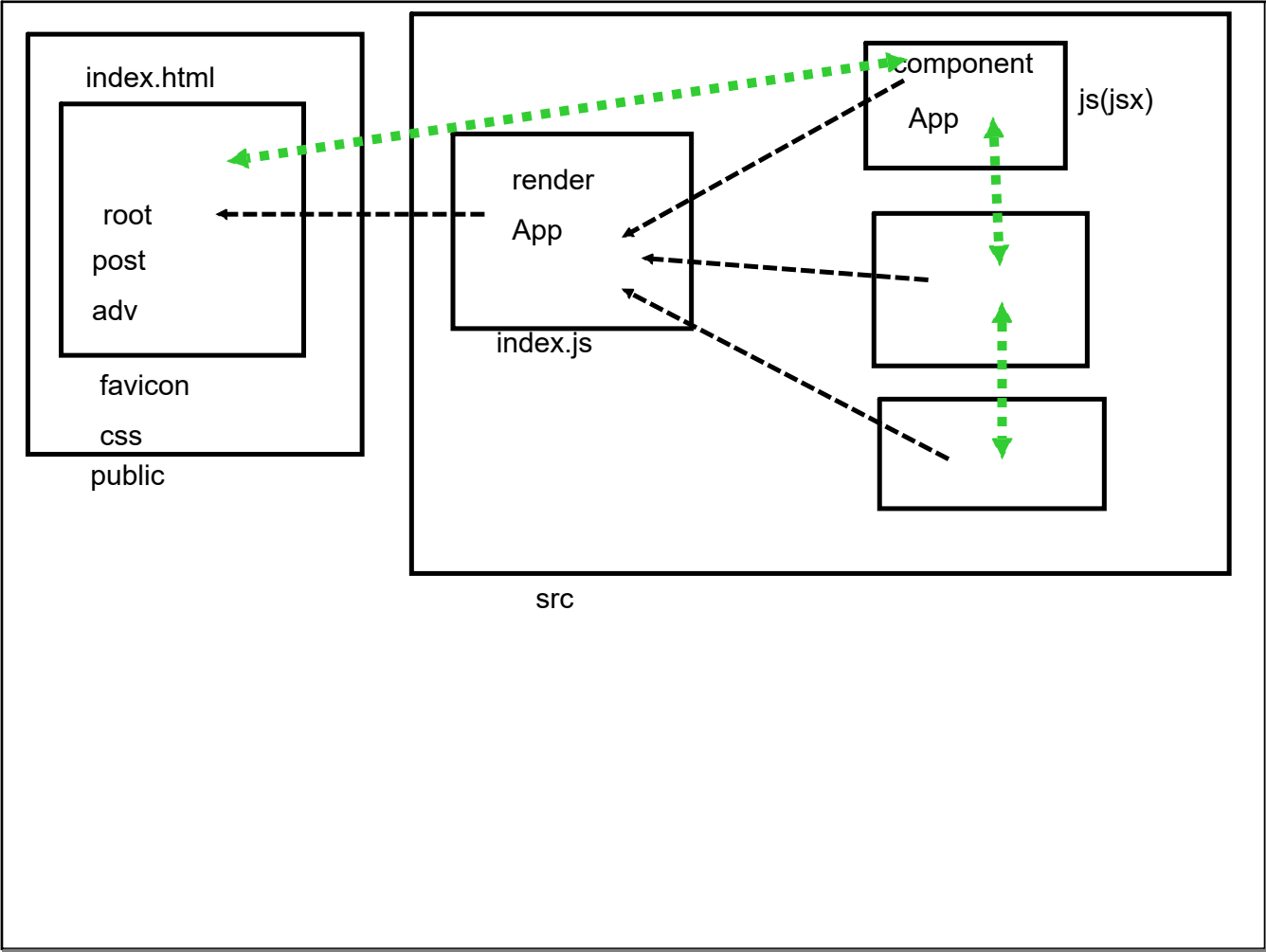
Browser has to re-render the DOM

Solution : Virtual DOM:

js create a in memory DOM Tree,

any changes is done in Virtual DOM : diffing engine

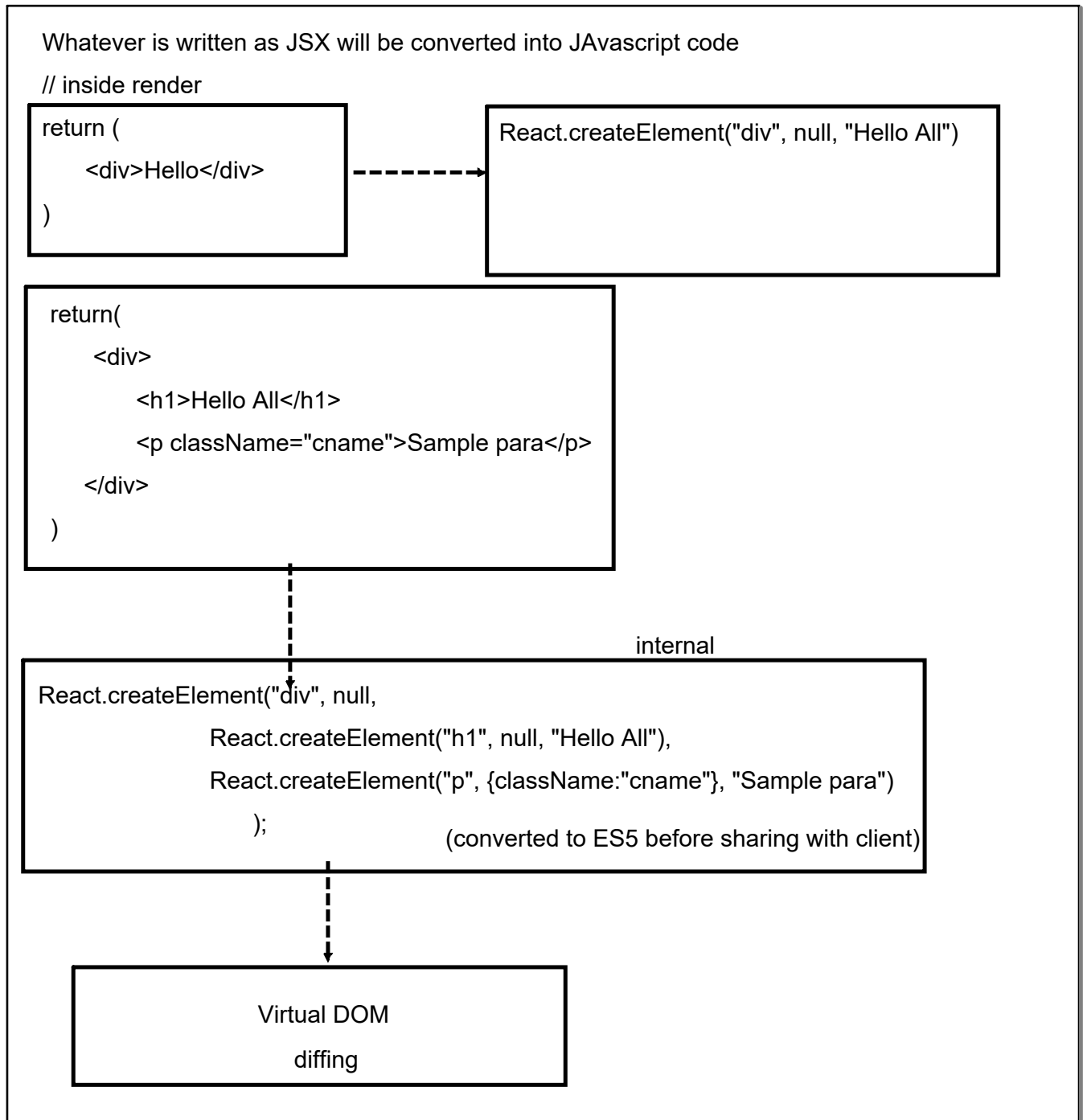




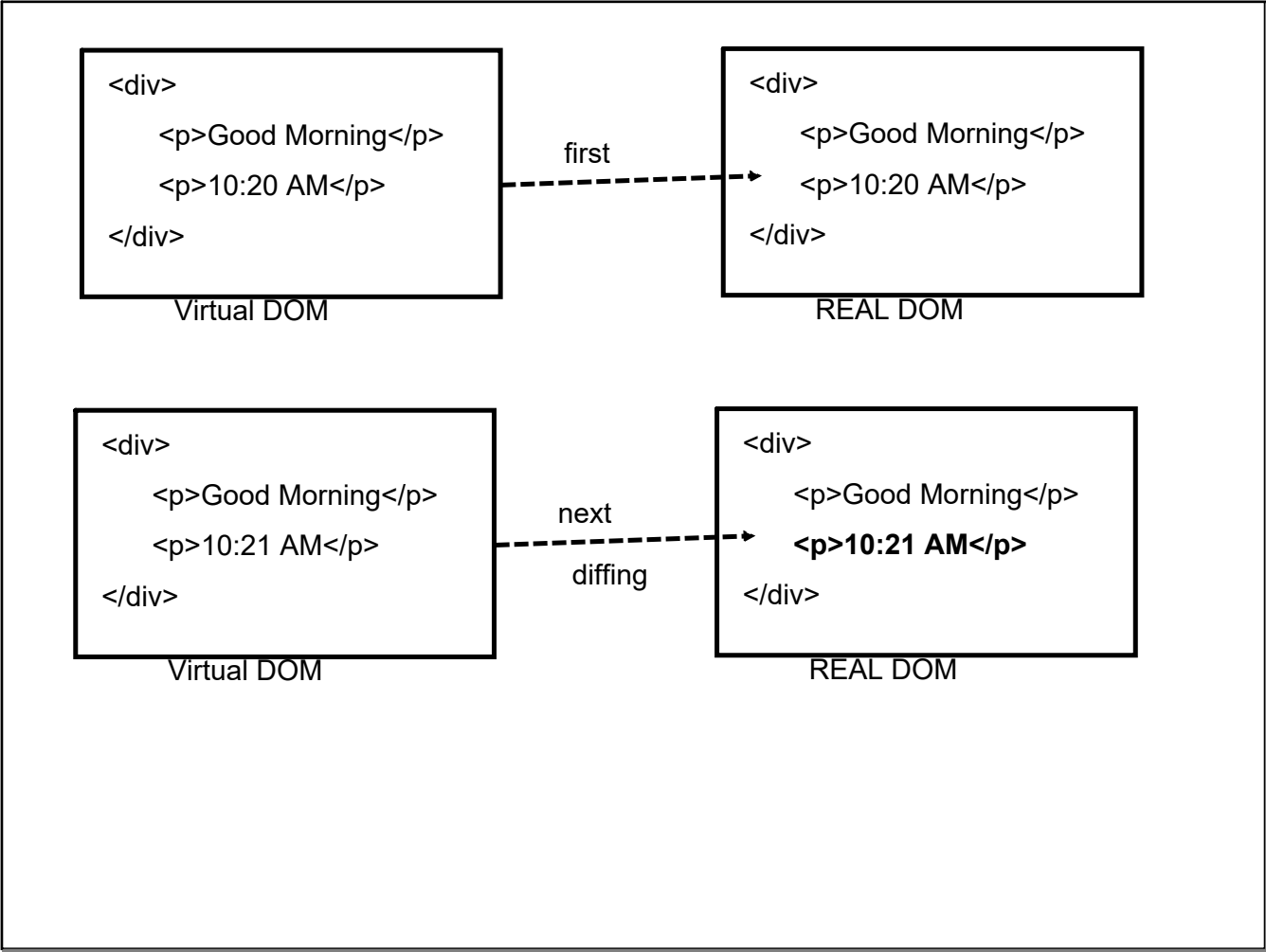
react : Main ReactJS lib

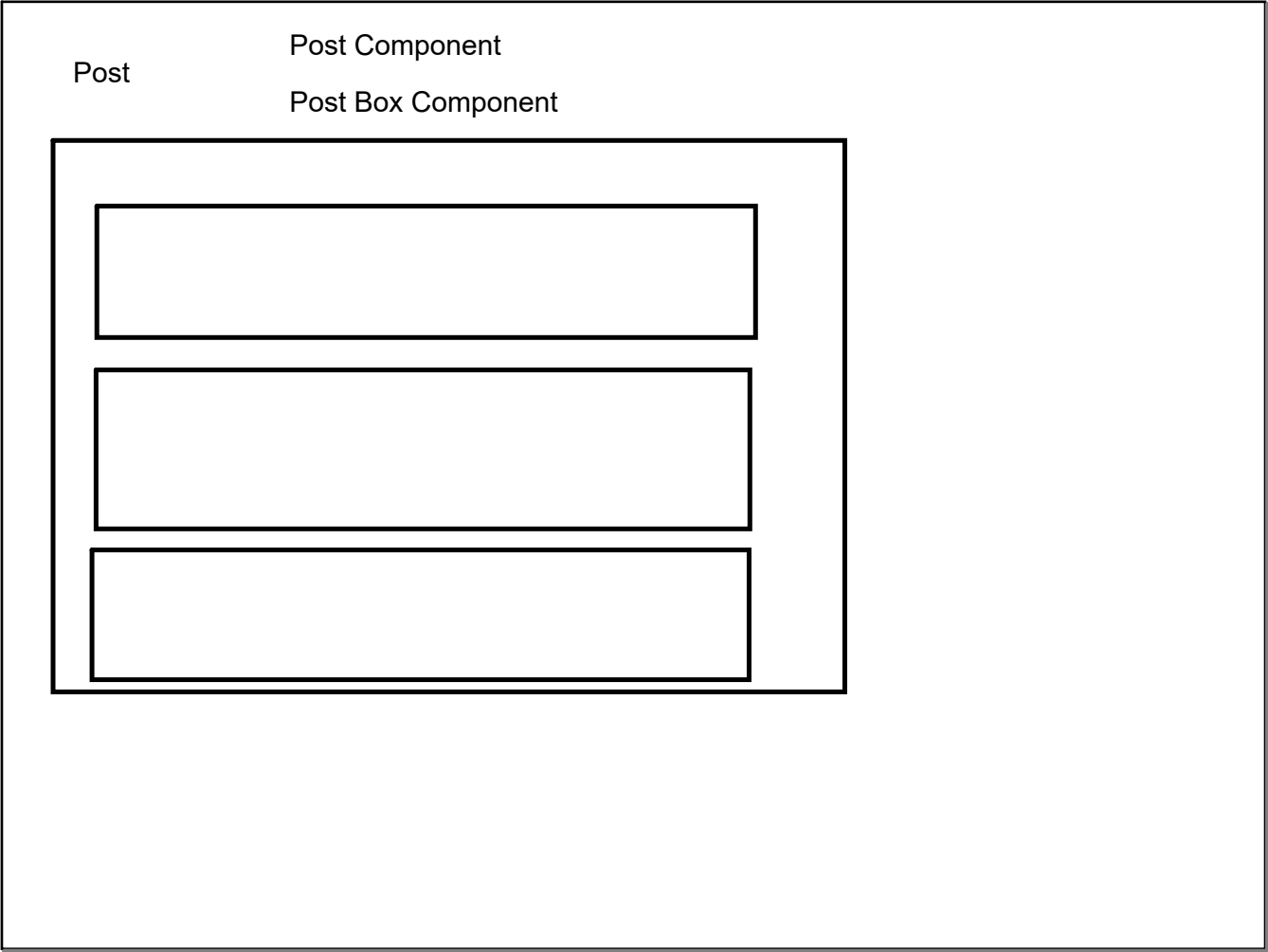
react-dom : Virtual DOM

1. Component (class) : must inherit Component (react)
2. must define (override) render method
3. must return a HTML template (JSX) from render method







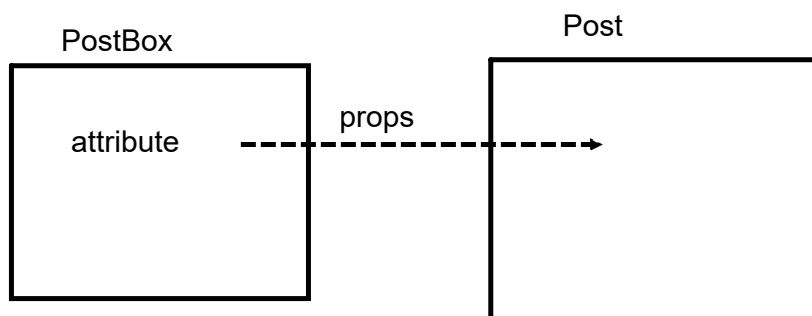


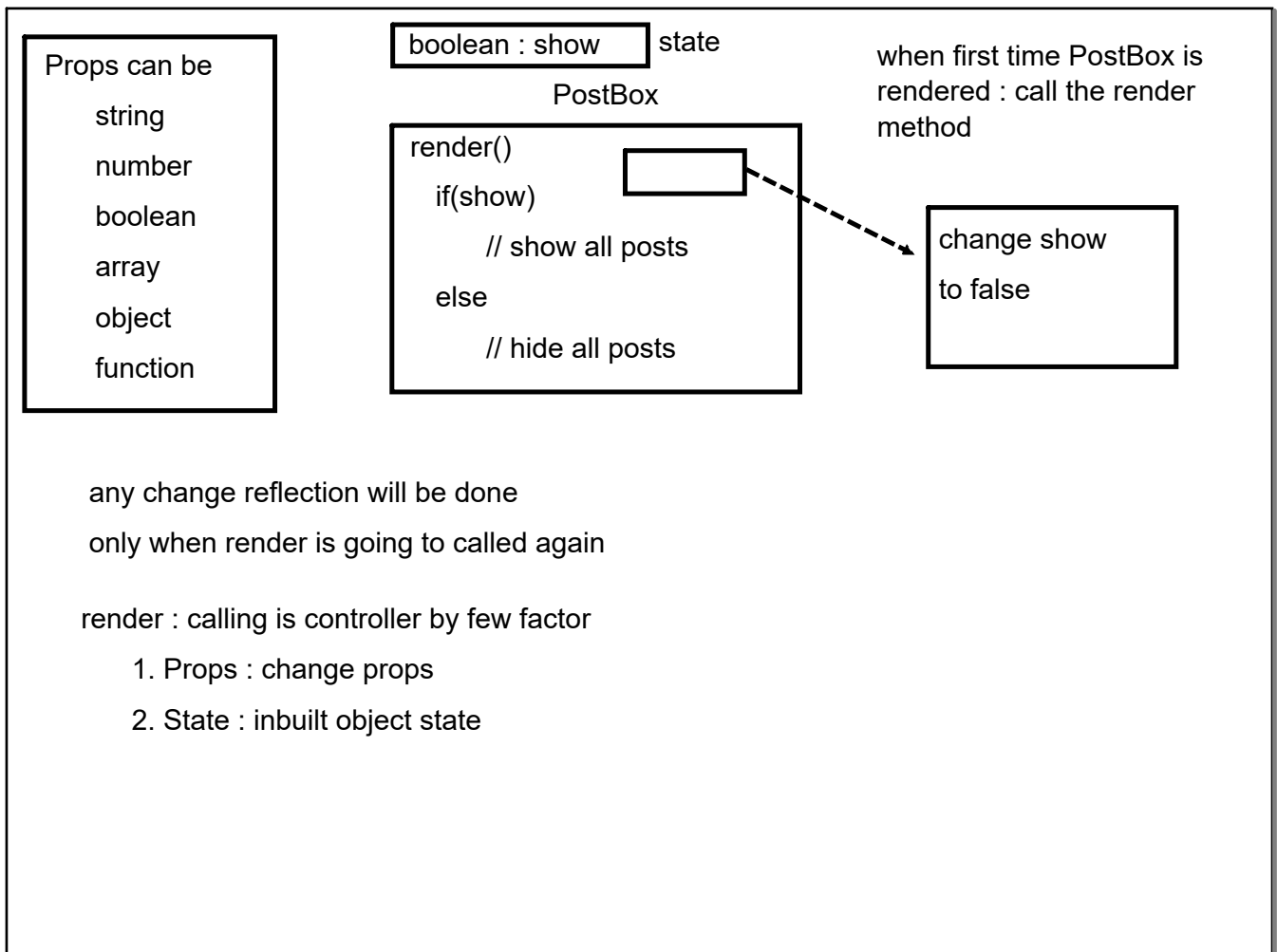
Props : Transfer data one component to another

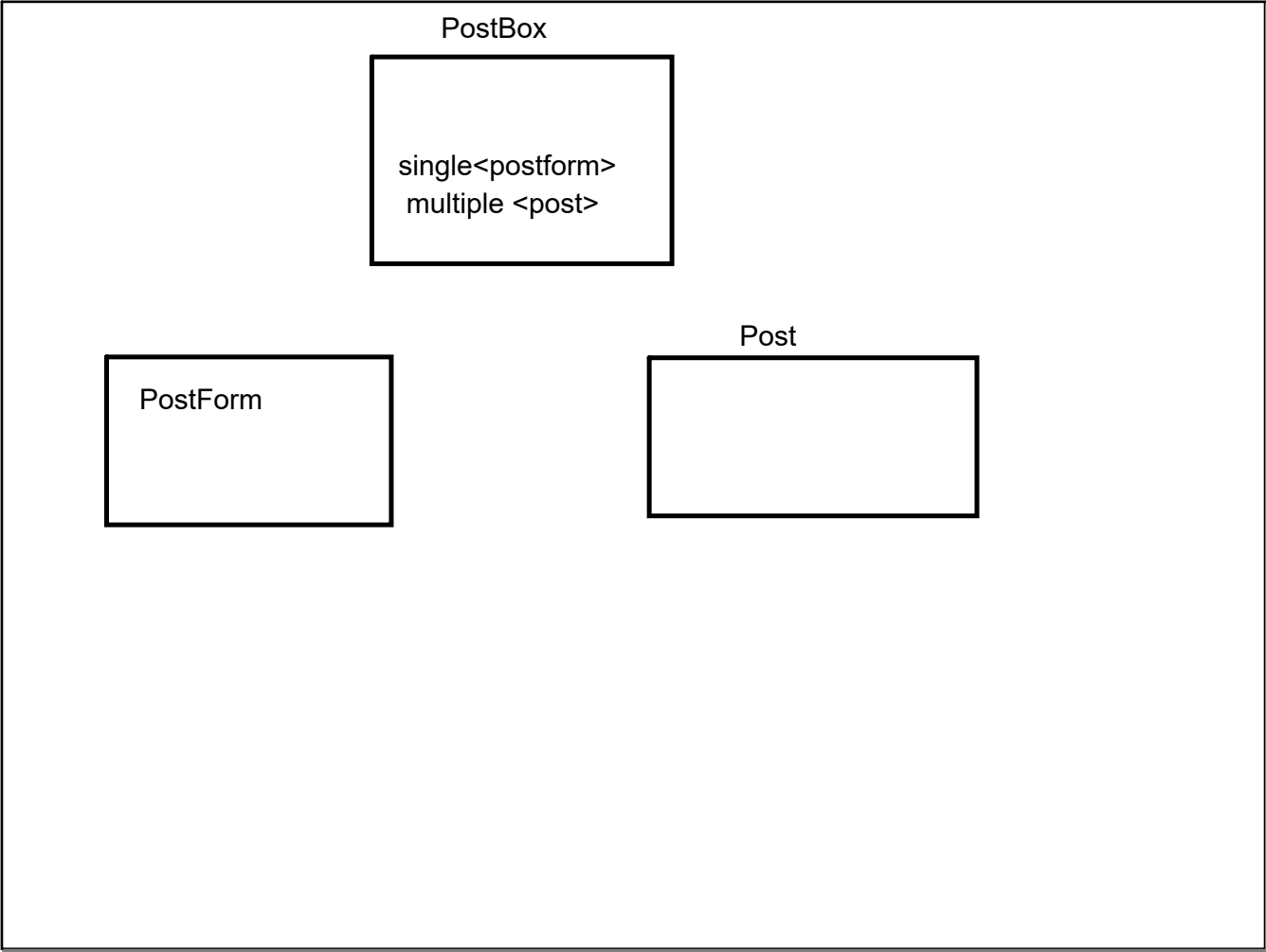
Created on the fly

all props passed to a component

automatically gets stored into an inbuilt object of component : props

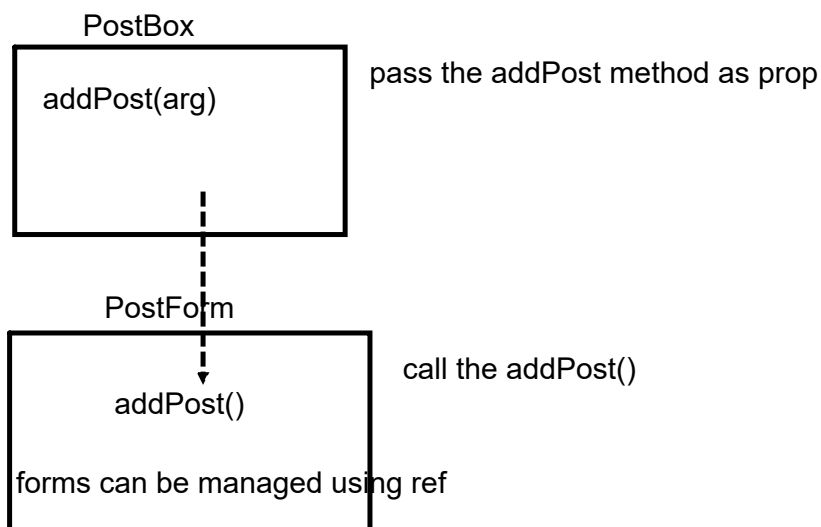






1. The new post info will req to be added into collection
2. Rerendering to take place

# need to transfer collection into state object



props : expected to controlled from parent

state : component specific controlling of data + effect the re-rendering

class variable : data needs to shared across multiple methods of that class

```
return this.state.postList.map(
  (post<x>) => {
    return (<Post author={post.author} tags={post.tags} body={post.body} key={post.id} />);
  });
```

transform logic goes inside function

<collect>=map(function())  
loops through all elements

x: Post Object	3 Post
y: Post Tag	[3 Post tag]

Backend Server : json-server

No service support

plain JS/jquery to be used

SPA : need to talk with server (async)



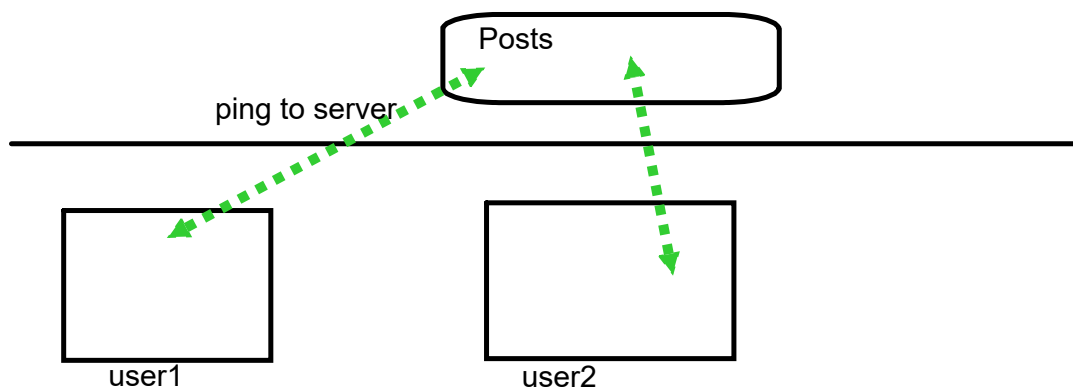
React lifecycle hook method :  
when component is loaded

1. Constructor
2. `componentWillMount()` : before rendering : if every thing is fine  
(only once : first time when component is loaded : not with every re-rendering)
3. `render()` method is called
4. `componentDidMount()` : just after rendering (once : first rendering)
5. `componentWillReceiveProps()` :  
invoked just before next rendering takes place:  
whenever prop/state changes  
props : immutable  
state : mutable
6. `shouldComponentUpdate()` :  
this can customized to control the flow  
: returns a boolean  
  
:true : re-rendering  
:false: no re-rendering
7. `componentWillUpdate()` : only if true is returned
- 8: `render()` : re-rendering
9. `componentDidUpdate()` : just after re-rendering
10. `componentWillUnmount()` : component is removed from Virtual DOM

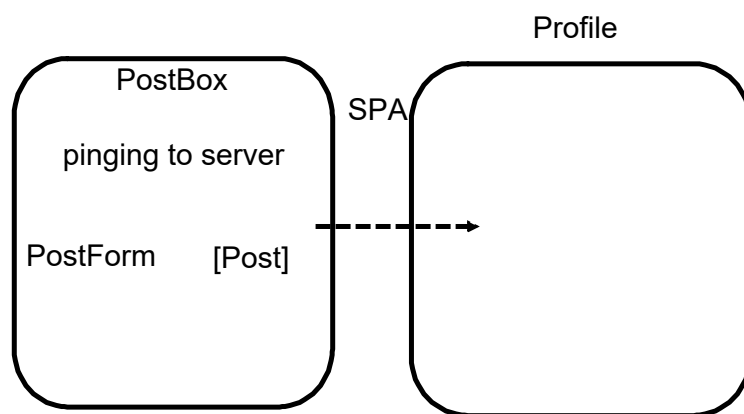
install jquery

# need to install jquery and make it as a part of package.json

> npm install --save jquery



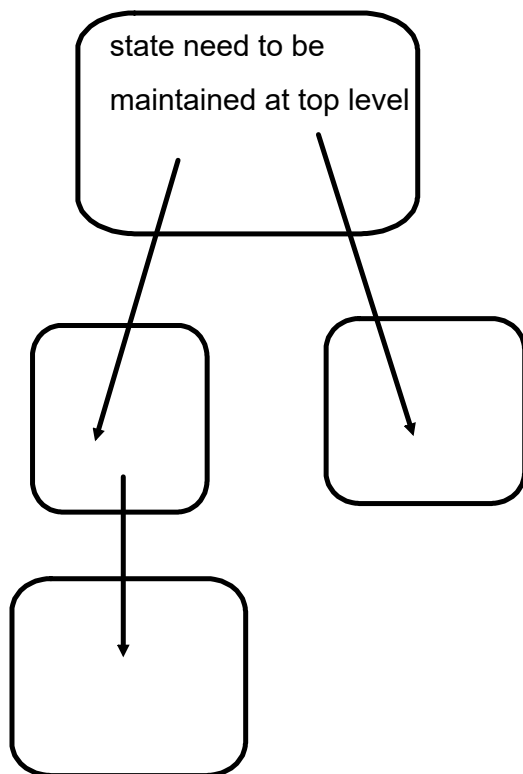
## Memory leakage



when component is  
unmounted :  
need to stop pingin

Redux API : FLUX design pattern

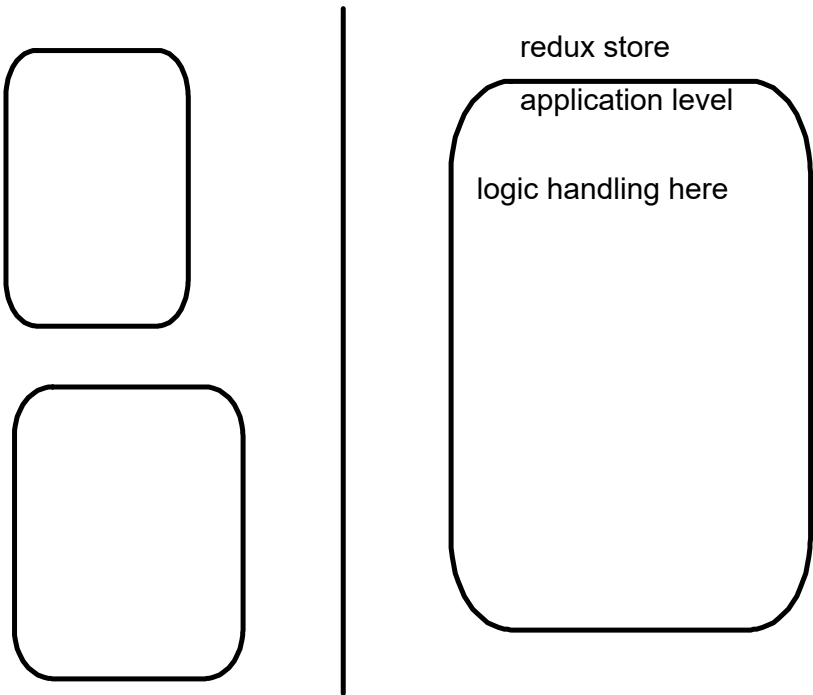
new Ecma Script feature



list of emails:

- click on email(event)
- list --> email view
- fetch data from server
- mark it read
- unread mail counter need to be reduced
- change the url
- all update need to be send to server

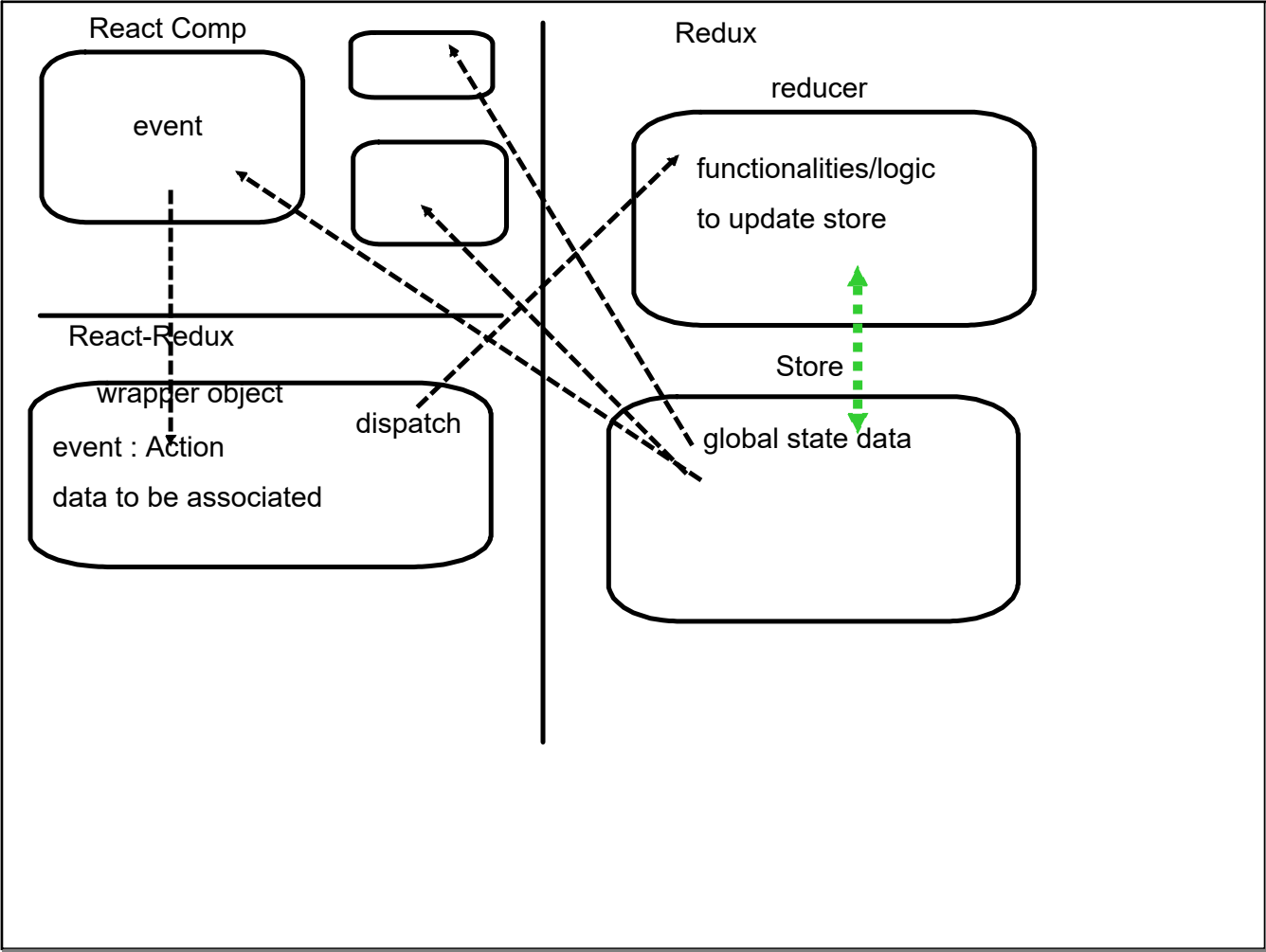
Redux : store (maintaining the global state)

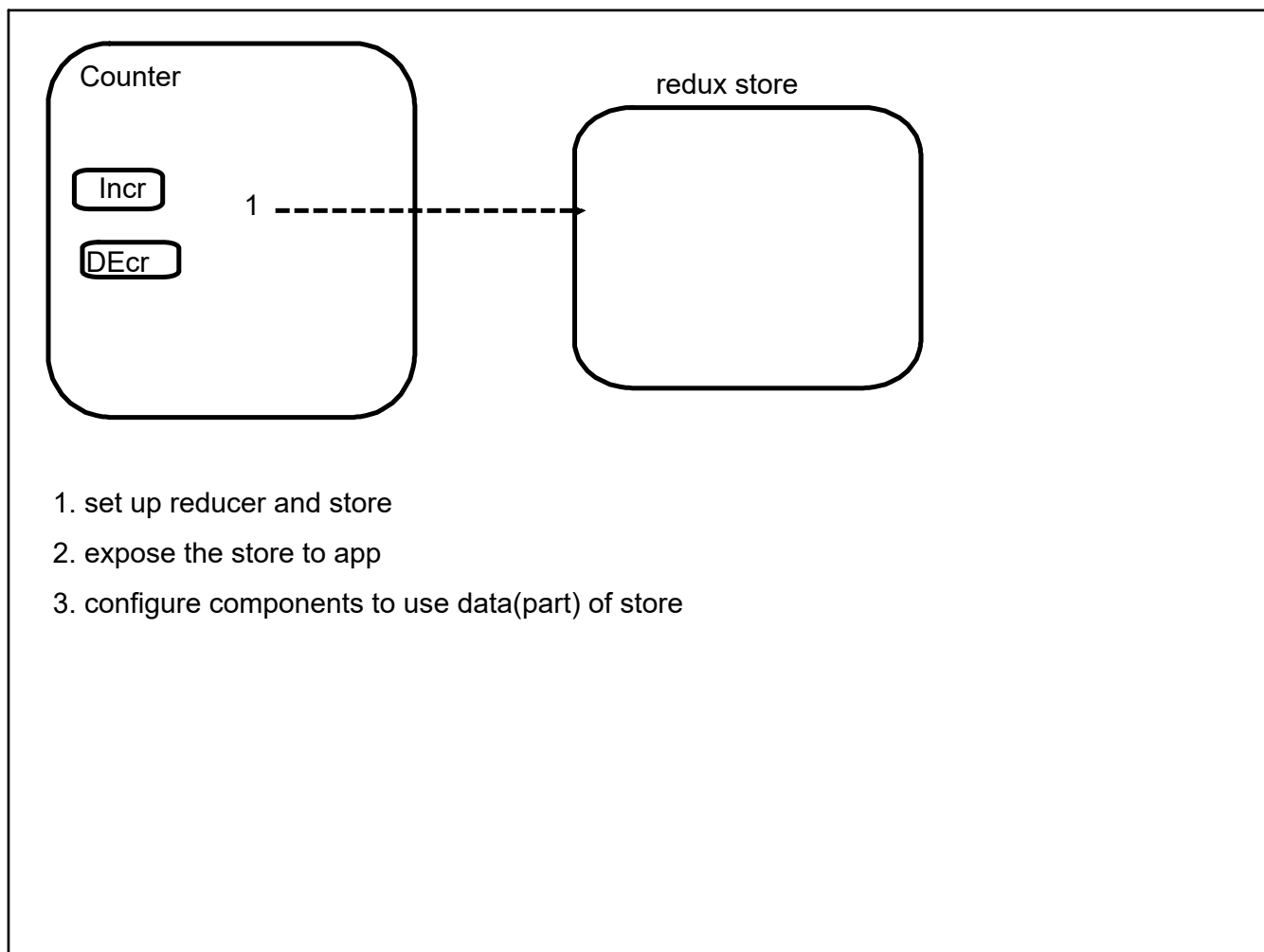


```
install redux lib
```

```
install plumbing (react-redux)
```

```
npm install --save redux react-redux
```







Parent1

```
<child1 name="First">  
<child2 name="First">
```

Parent2

```
<child1 name="Second">  
<child3>
```

Special Router lib : predefined JSX component

npm install --save react-router-dom

HOME

CONTACT

ABOUT

MongoDB :

=> cross-platform

=> document oriented

---

specific instance :

embedded

local DB

Cloud Mongo DB

### Mongo DB

Database

Collection (collection of multiple record)(Tables)

Document (record) : JSON

# schema less

different document in same collection can have  
different structure : scalable

### Mongo DB : Community Server

1. Installed Community Server
2. Define the location : (preferred) c:/data

mongod : connect with repo location : >mongod --dbpath "<db location>" | mongod --dbpath "C:\data"

mongo : launch the mongo db terminal

URI : spring.data.mongodb.uri=mongodb://[username]:[password]@[ip]:[port]/<dbname>

java class to configure MongoTemplate

Spring Data :

- ALL DB will have common approach

- # change : Repository

- # Query syntax

Inherit a repository

- # all standard CRUD functionality (by default)

- # method declared with proper naming convention (implementation provided internally)

- # method declared , associate a query @Query

- # Custom : if need to add detailed business logic

Custom :

1. Create custom interface

2. declare custom method

3. Create a implementation class (inherting the interface)

4. inherit the custom interface into repository

```
db.<collection>.find({  
  "author" : "First",  
  "author" : {$in : ["First","Second"]}  
  "likes" : {$gte : 50}  
})
```

creating a security credentials for DB

Create a user for DB

Hadoop : DB

=> query based as well

=> programmatic control

=> document | RDBMS

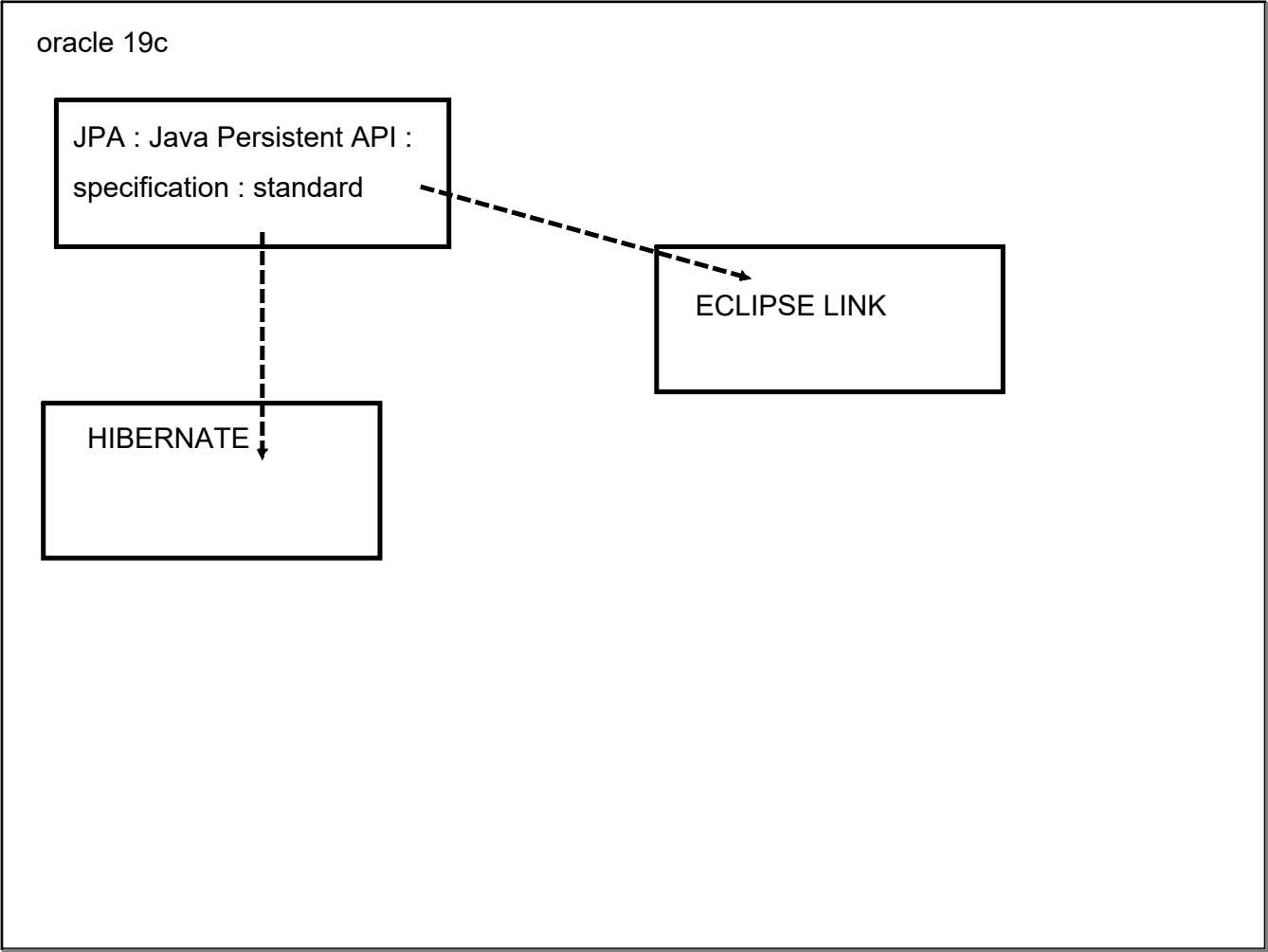
=> index : normalization

=> complex relationship among tables

=> algorithm (hashing equation)

=> distributed DB





JPQL : ---> ORACLE Dialect

PL/SQL : PAscal based

Reusable programmatic unit (Procedure/Function)

Programmatic construct

Integration with SQL

structural

**Structure**

Declaration (optional)

Executable Commands (mandatory)

Exception Handling(optional)

Procedure/Function

Procedure : cannot return value explicitly

Function : have to return a value explicitly

Maintained as Objects in Oracle

## Procedure

```
CREATE OR REPLACE PROCEDURE <procedure name> ([arguments])  
<IS|AS>  
    [declaration]  
BEGIN  
    <execution code/commands>  
EXCEPTION  
    <exception handling code>  
END;
```

Three different types:

IN : read-only variable

OUT : write value in it (data out of procedure)

INOUT : read-write

## Procedure

```
CREATE OR REPLACE FUNCTION <function name> ([arguments]) RETURN <return type>
<IS|AS>
    [declaration]
BEGIN
    <execution code/commands>
EXCEPTION
    <exception handling code>
END;
```

Procedure : process, cant use queries, out

Function : some calculation, queries (no DML stmt), out|RETURN

PL/SQL : inbuilt

Conversion : one type into another type

String

Date

TDD : Test driven development

Agile : Testing : Manual  
Automation

Unit-Tests : testing a specific logic in isolation

Integration : Component class(unit) + HTML Template + service(unit)

End-to-End :

Backend :

JUnit : java test cases  
Spring test api  
Mockito

Frontend

Jasmine (JS)  
Karma



JUnit :

Javascript :

Jasmine (independent) : does not require DOM

Karma : test-runner : run test cases over angular app

#### Jasmine

1. Karma : spec.ts : test files

wrap test case in test suite : describe() function

every test will be defined under : it()

matcher : expect() ~ assert()

beforeAll(<function>) once

beforeEach(<function>) : before each it

afterAll(<function>) : once

afterEach(<function>) : after each it

Fragile

```
spyOn(<service>, <method>).and.callFake(<fake function>);  
spyOn(<service>, <method>).and.returnValue(<value>);
```



AWS :  
Account

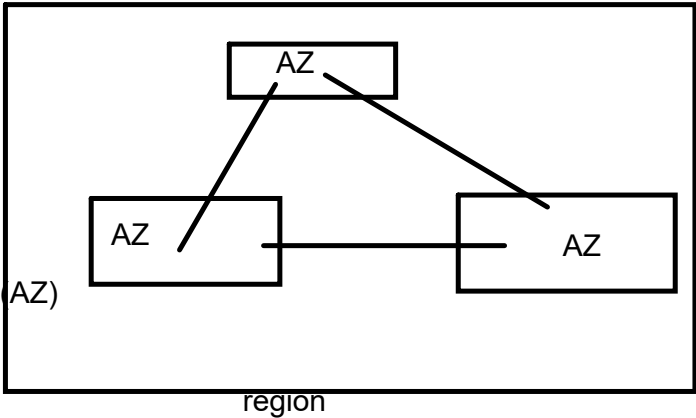
Regions : AWS services are region scoped  
clusters of data centers : Availability Zones (AZ)

3 6(max) (2 min)

us-east-1a

us-east-1b

us-east-1c



EC2 Service : Virtual Server  
(Face of AWS)

Serverless Lambda Service (Modern)

EC2:

Renting a Virtual Machine

# store data virtual drive (EBS)

# Load distribution

# Auto-scaling

SSHing into your EC2 Instance

ssh : Windows 10

putty

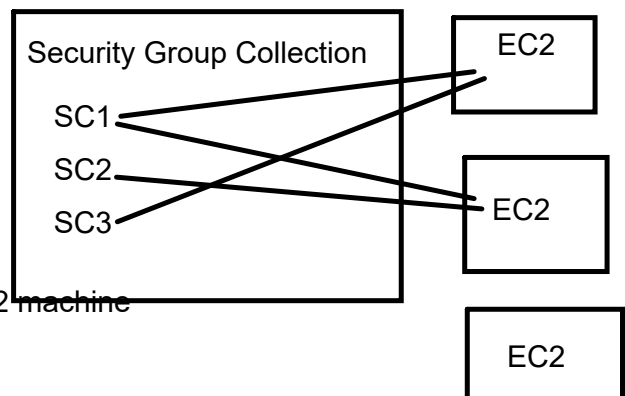
Security Groups :

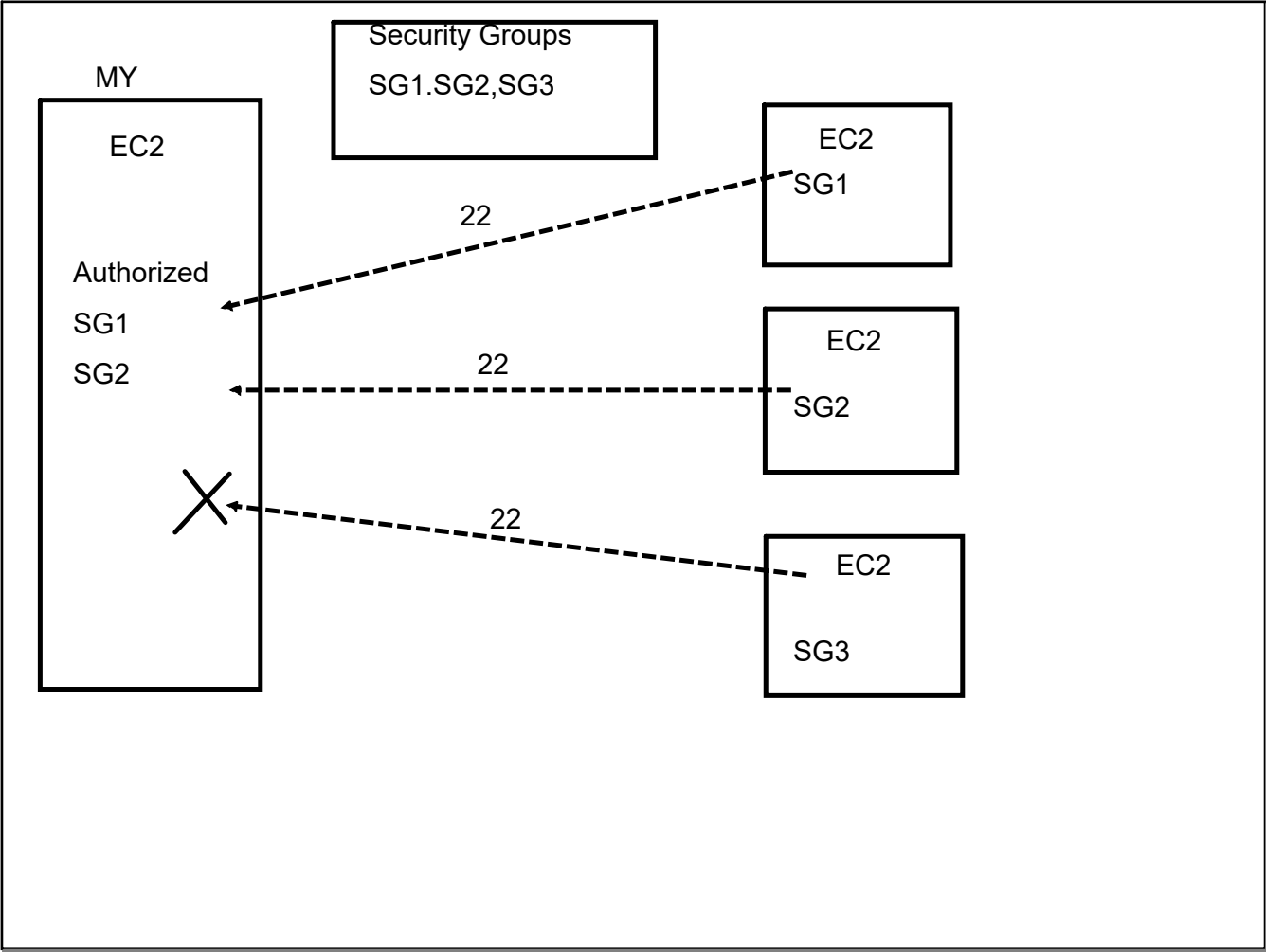
controls how traffic is allowed in/out of EC2 machine

1: port

2. protocol

3. IP ranges (IPv4/IPv6)





### Multiple SG

Each SG multiple rules (inbound/outbound)

SG bound to regions

live outside EC2 instance

Private IP : remains same on restart of EC2 instance

Public IP : changes with every re-start

Elastic IP : want it to remain same on every restart

On demand Instances :

Reserved ( min 1 yr)

Spot Instance : : not reliable

Dedicated Instance : h/w will not be shared

Dedicated Host : entire physical server



Storage:

EBS : S3

EBS : Elastic Block Store

EC2 instance :

root volume (main drive) : lost if EC2 instance is terminated

EBS Volume : network drive attached with EC2 instance

It uses network to communicate

detached and connected to other EC2 instance

Its locked to an AZ

provisioned capacity

SIZE | THROUGHPUT | IOPS (I/O ops per second)

EBS 4 types :

GP2 (SSD) : General Purpose SSD balance price and performance

1 - 16TB,

IOPS

IO I(SSD) : High Performance

ST I (HDD) : Low Cost frequently accessed data

SC I (HDD) : Lowest Cost

S3 : infinitely scaling storage

Buckets(dir);globally unique ( region specific)

no uppercase

no underscore

3-63 char

first char can be lowercase alph/ number

: store object(files) : key

key : (path)

s3://<region based instance>/<bucket name>/some.txt (some.txt : key)

s3://<bucket name>/<sub-folder>/some.txt

1 object ( 1 file) : 5TB (multi-part upload)

AWS Lambda :

Concept of using Cloud service :

Serverless : no more need to manage the server

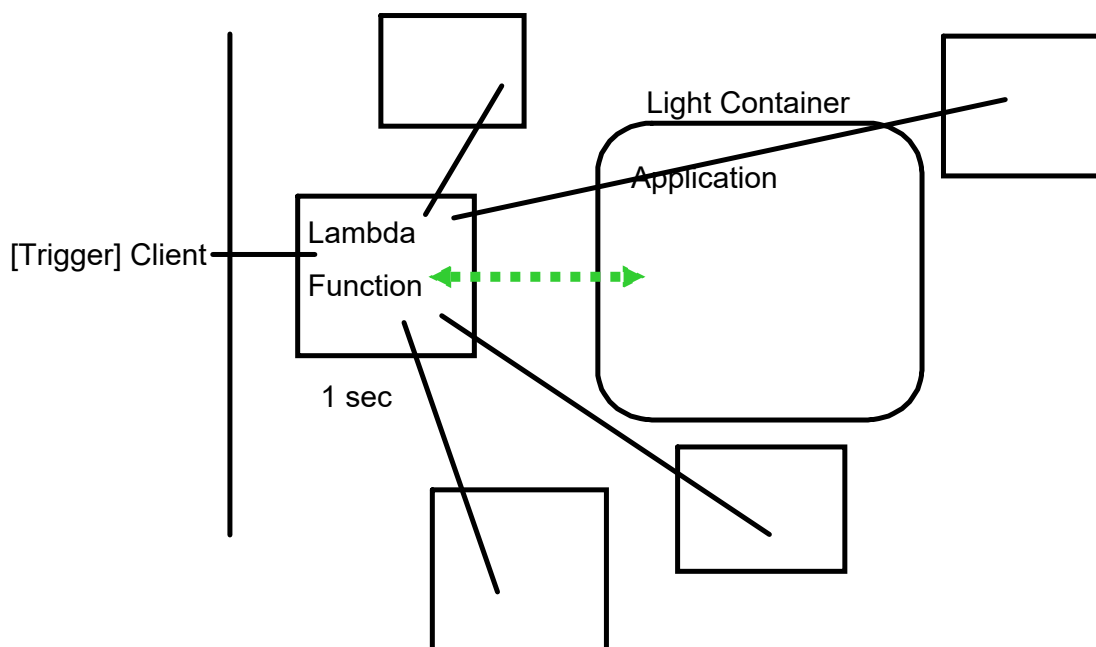
#deploy your code : Lambda function :

Ec2 : setup a server

Serverless : create a function

FaaS : (Function as a Service)

1. Write a code and upload it to Lambda Service
2. Spawn the server(s) on receiving the request - trigger (Automatic scaling)
3. You only pay for the amount of time your code is running



Compute :

EC2

EBS

ELB

Lambda

Node.JS

Python

JAva

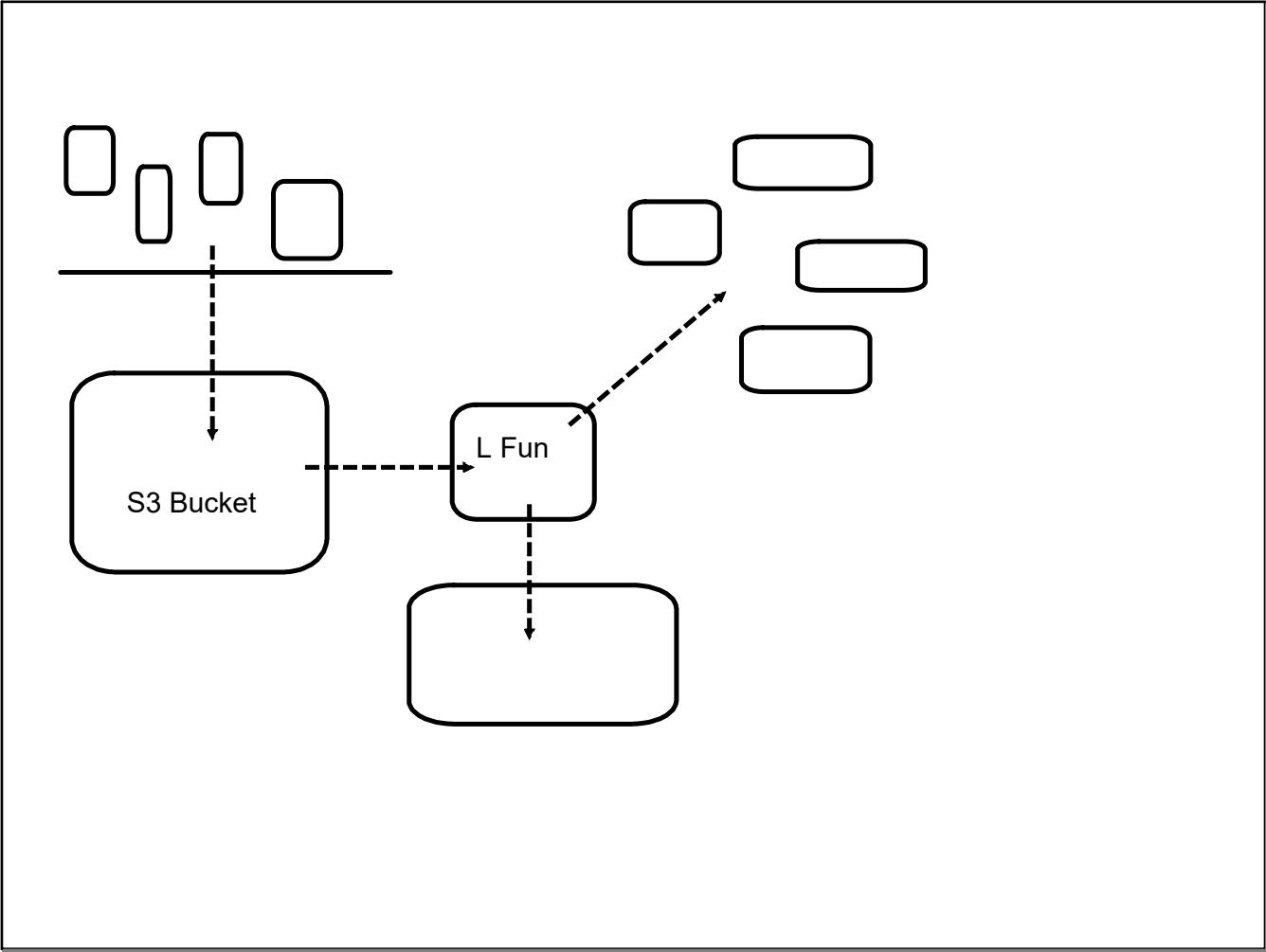
C#

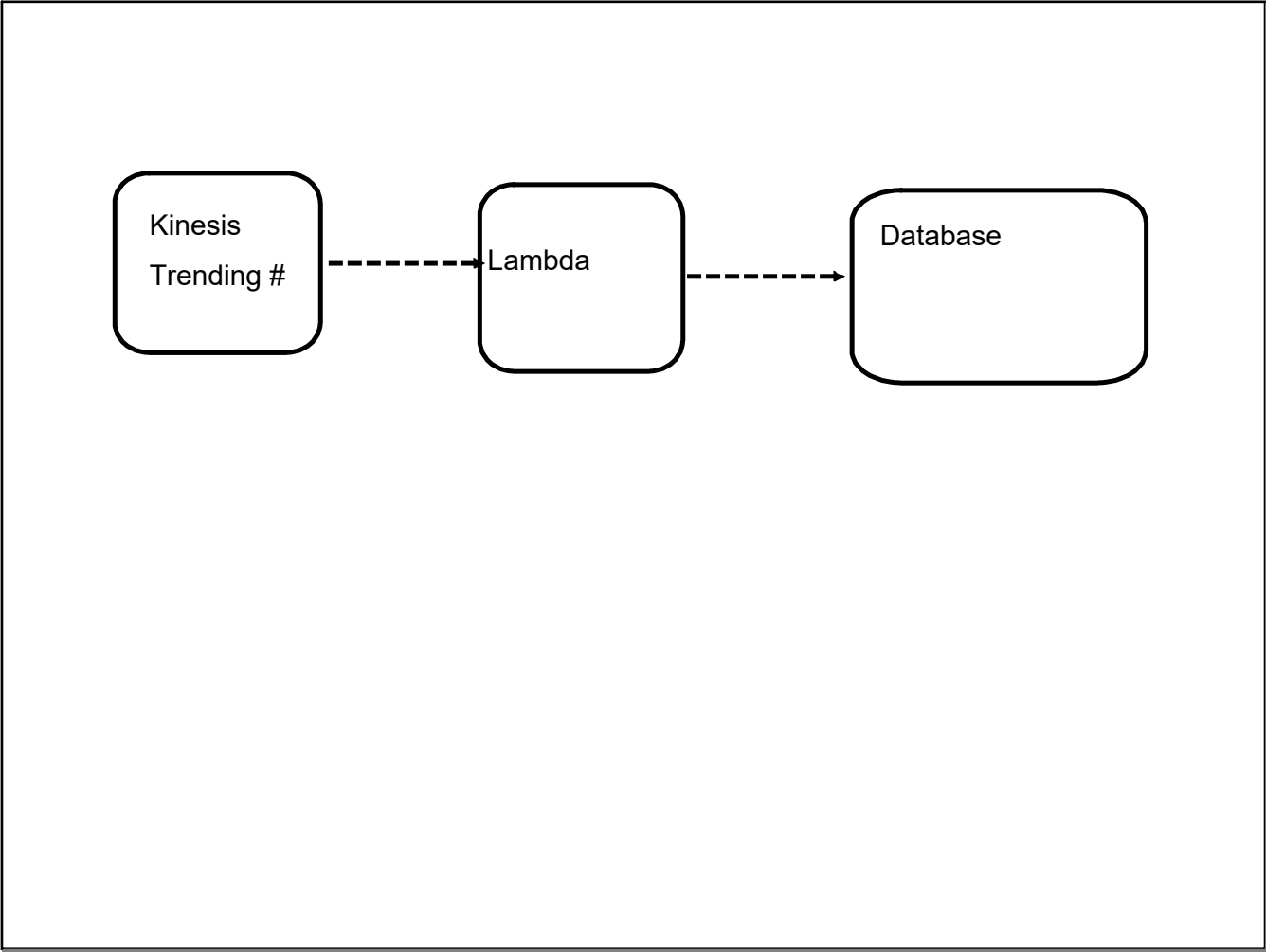
Ruby

10 lakh requests  
40,000 GB of compute time

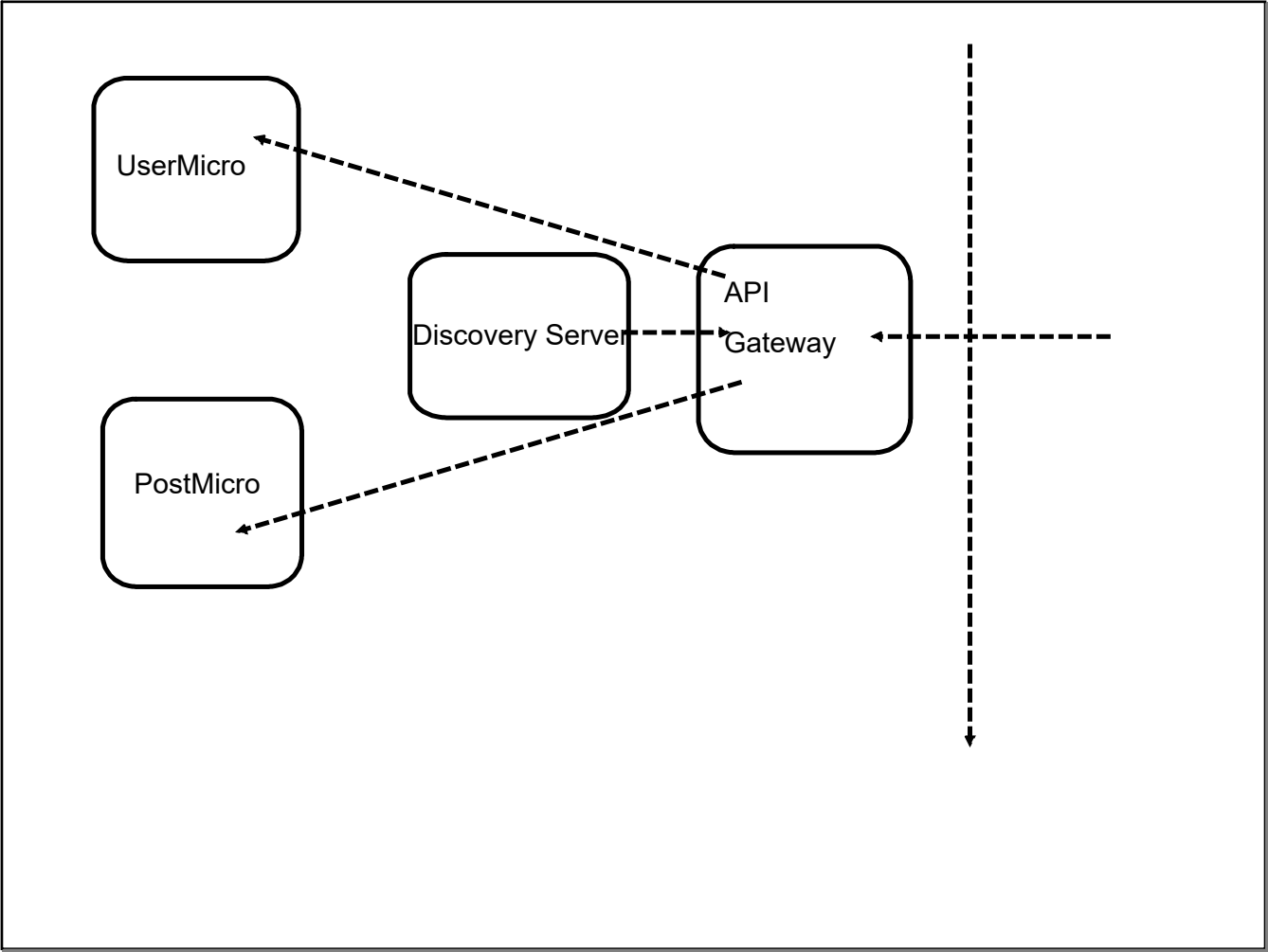
Trigger :  
multiple AWS service  
: API GATEWAY (http) :  
: S3  
: DynamoDB  
: Kinesis

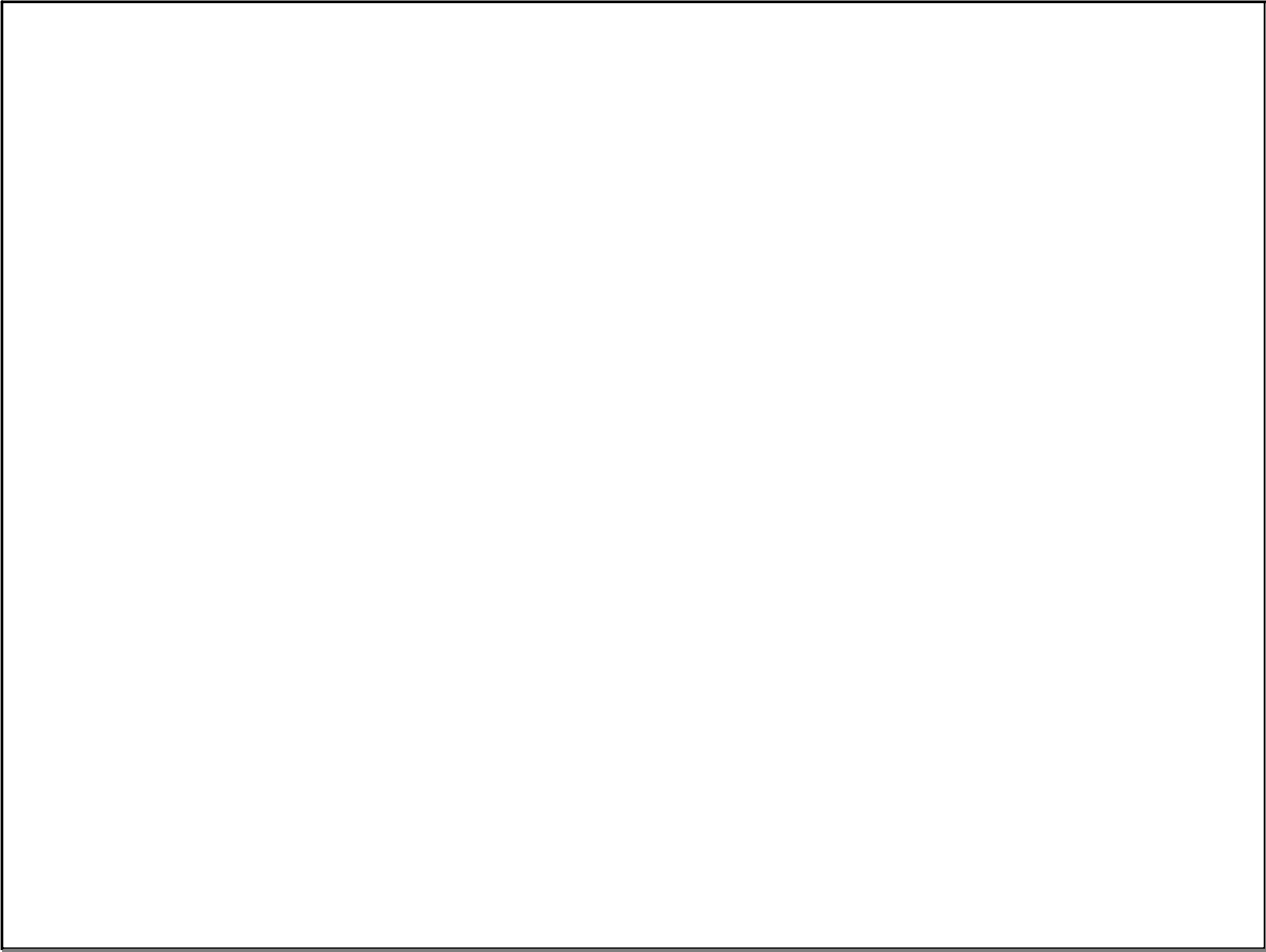






- 1.. Lambda function defined over AWS Service Console
2. Code designed on dev m/c , have the lambda function created and embedded with code and upload the packages content
3. aws console installed over local machine :  
upload it using CLI





```
{{filesize | size : "MB" }}
```

```
arr | slice : 1 : 4
```

```
{{ <value> | <pipe-name : [options]>}}
```