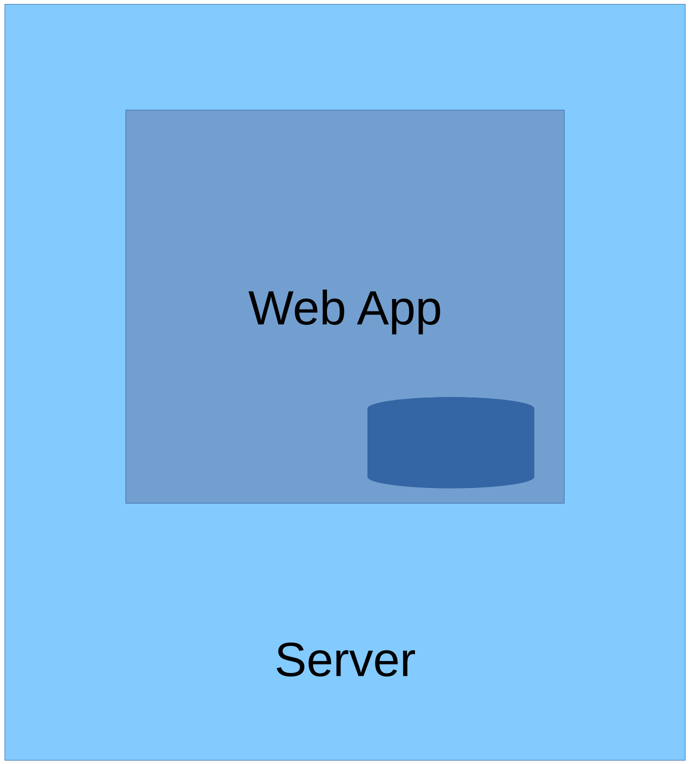
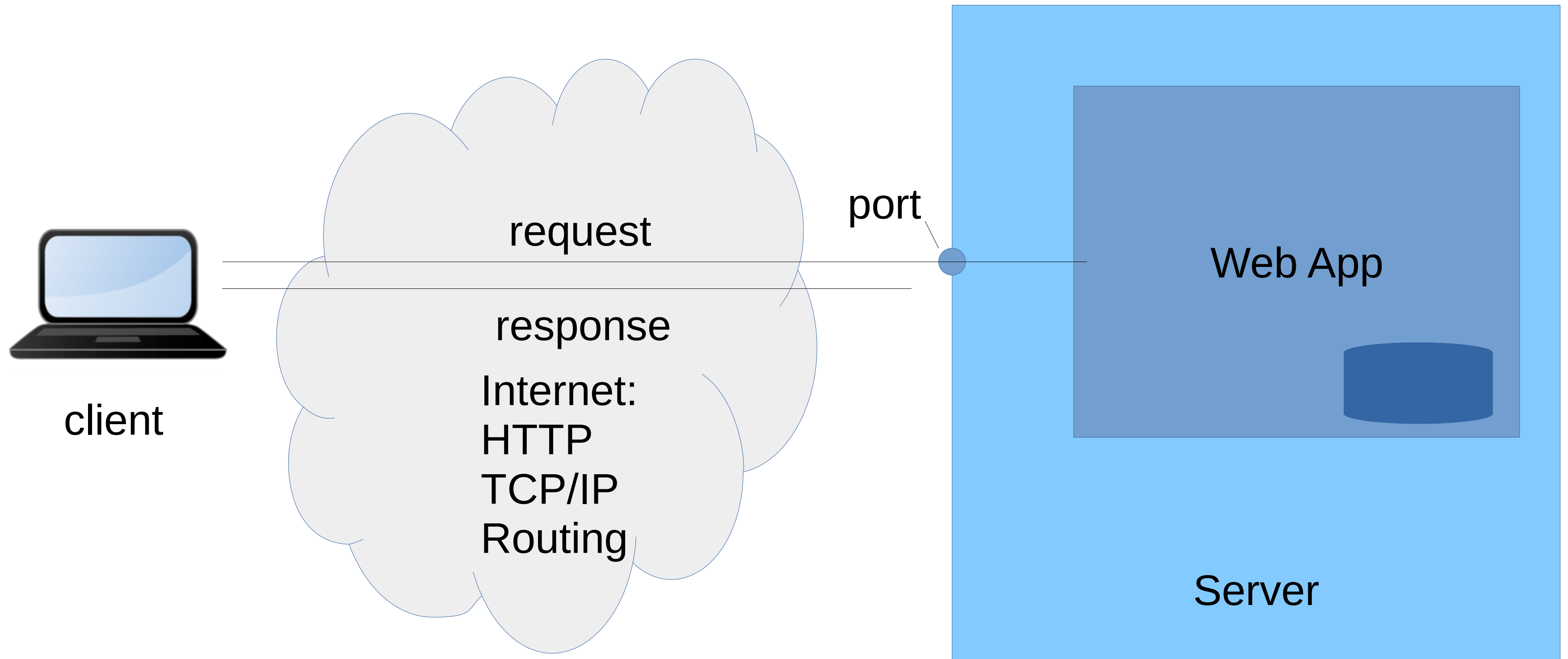
Anatomy of a Web App

Anatomy of a Web App

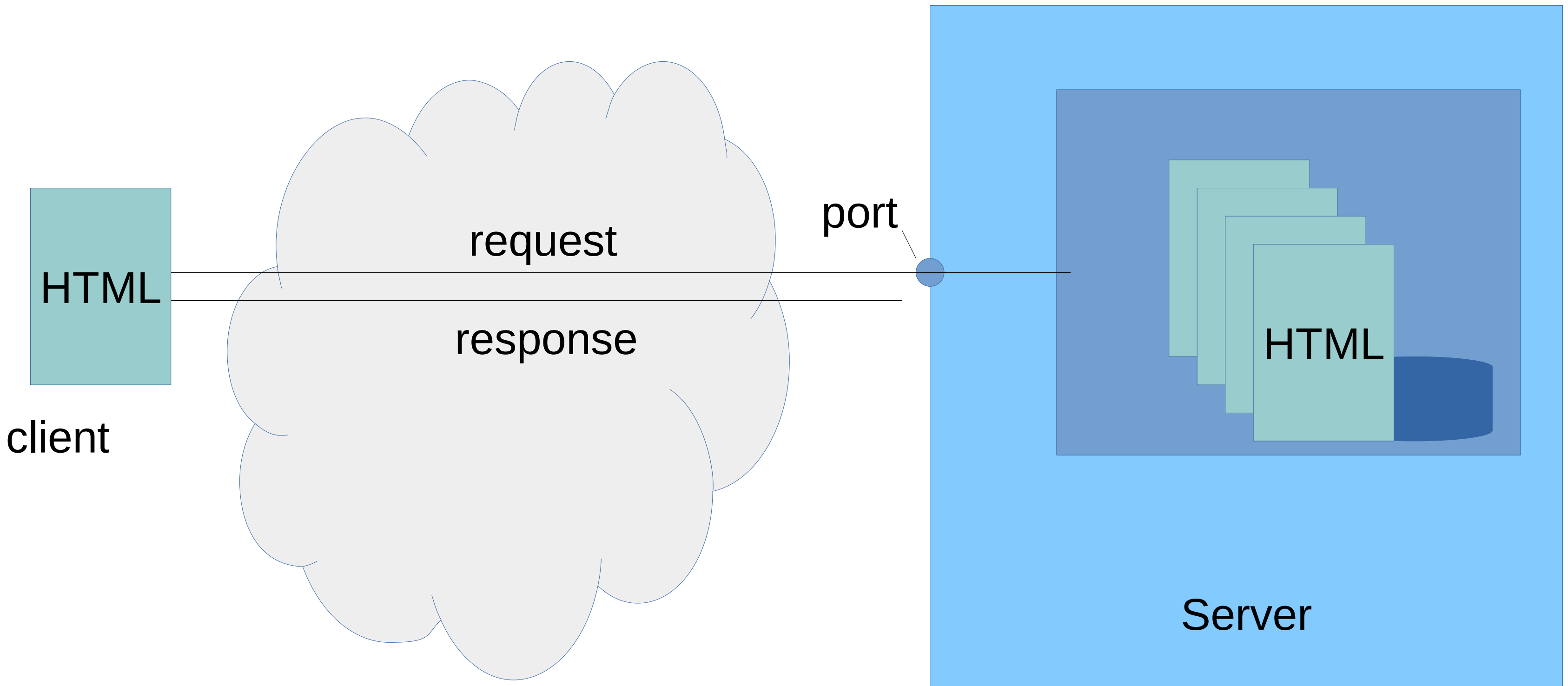


Anatomy of a Web App



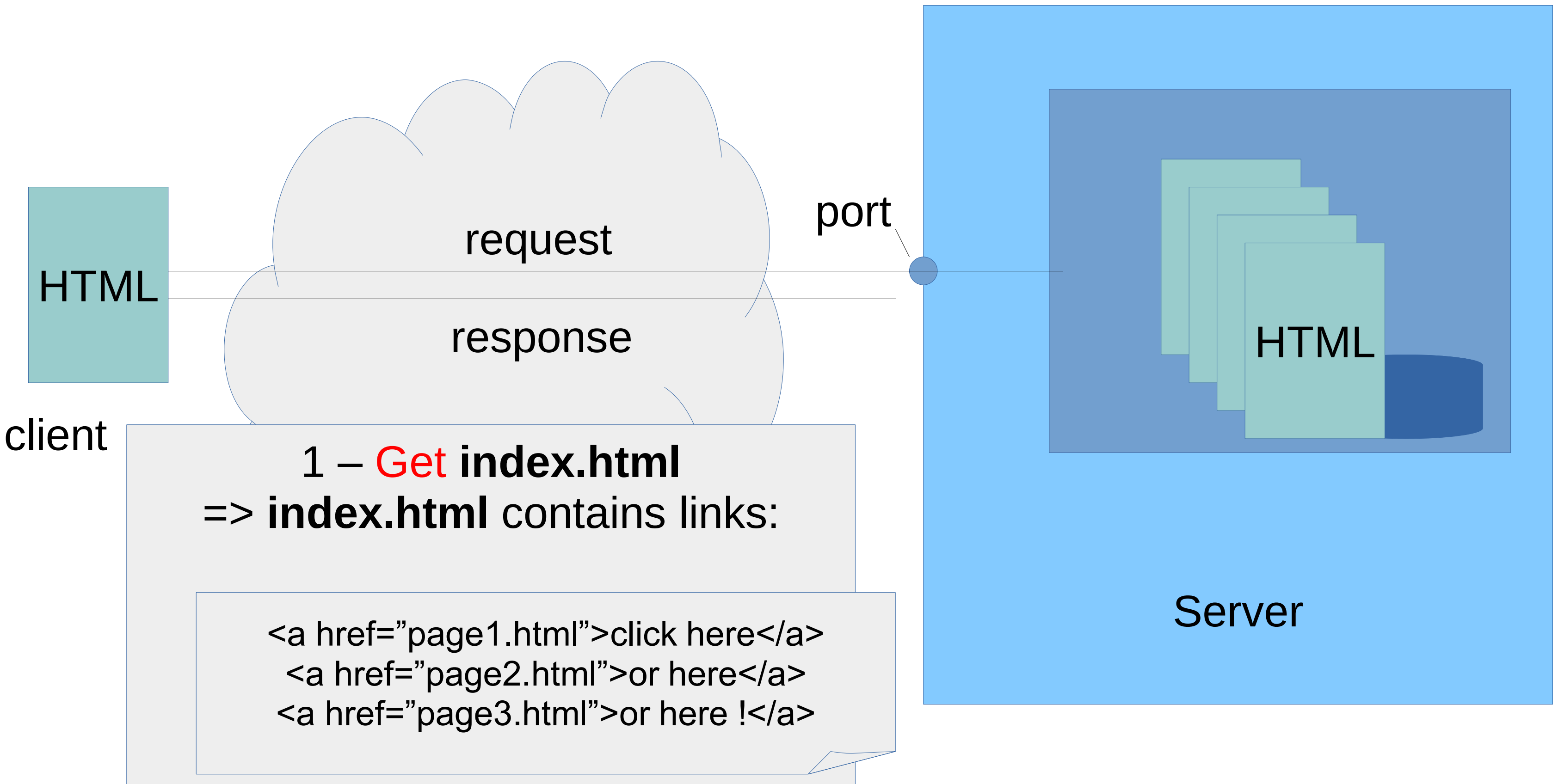
Anatomy of a Web ~~App~~ site

(1990's)



Anatomy of a Web ~~App~~ site

(1990's)

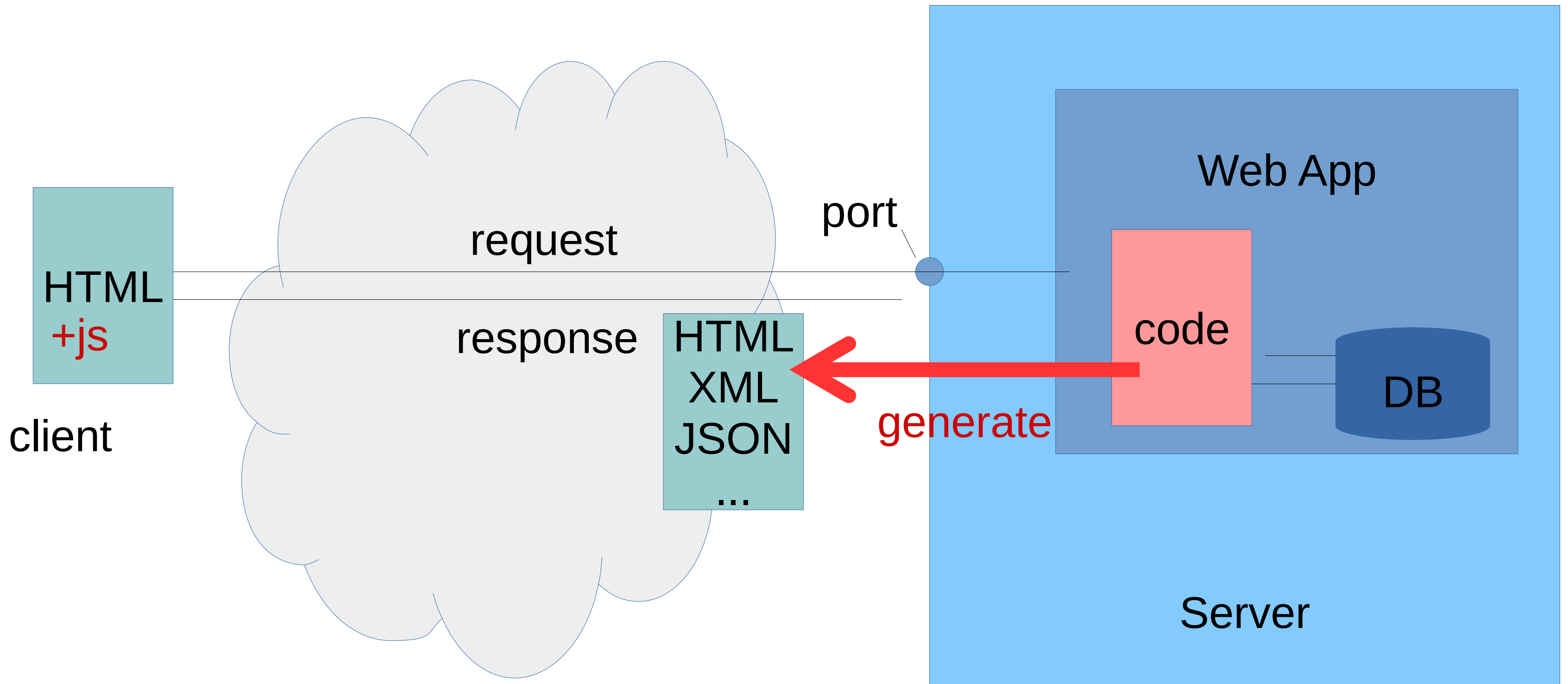


1 – Get **index.html**
=> **index.html** contains links:

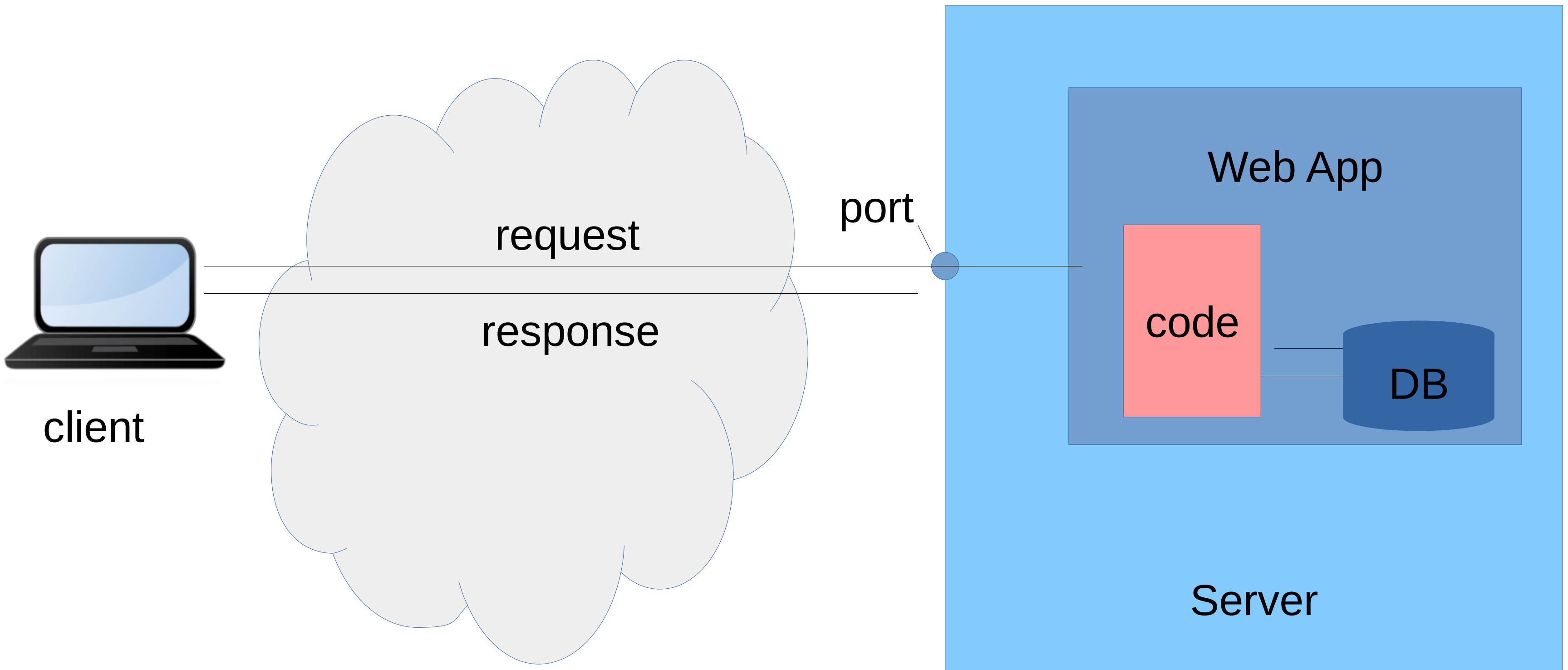
```
<a href=”page1.html”>click here</a>
<a href=”page2.html”>or here</a>
<a href=”page3.html”>or here !</a>
```

2 – user sends: [get page2.html]
=> page2 has new links...

Anatomy of a Web App



Anatomy of a Web App



The Problem: how to organize the code?

- Need to handle complex logic in a maintainable way
- Patterns to the rescue again!
- Martin Fowler's (remember him?) Patterns of Enterprise Application Architecture has become the reference (though some of it is now dated)

The Addison-Wesley Signature Series
Patterns of Enterprise Application Architecture
Martin Fowler

Three-tiered architecture

- Fowler relies on the traditional *three-tiered architecture:*
  - separation between presentation layer, domain layer, data source layer

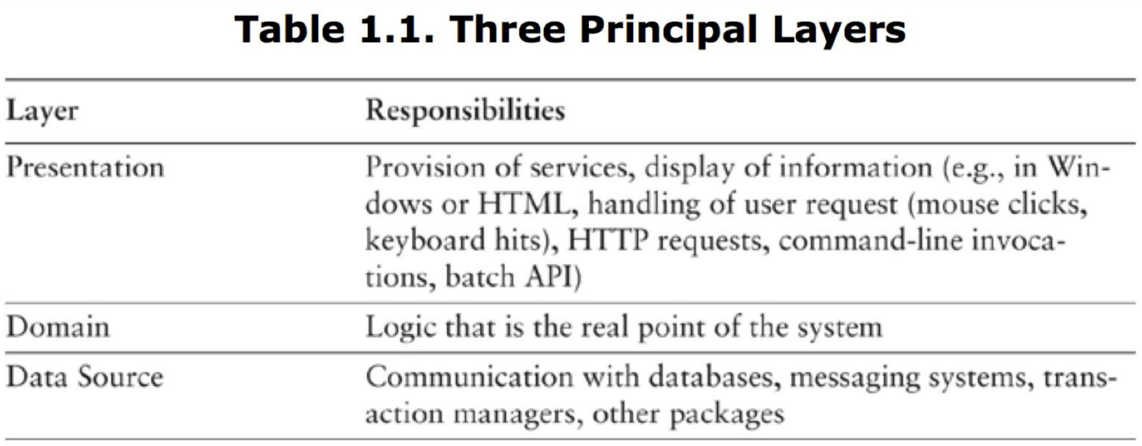


Table 1.1. Three Principal Layers

Layer	Responsibilities
Presentation	Provision of services, display of information (e.g., in Windows or HTML, handling of user request (mouse clicks, keyboard hits), HTTP requests, command-line invocations, batch API)
Domain	Logic that is the real point of the system
Data Source	Communication with databases, messaging systems, transaction managers, other packages

  - where does each live?

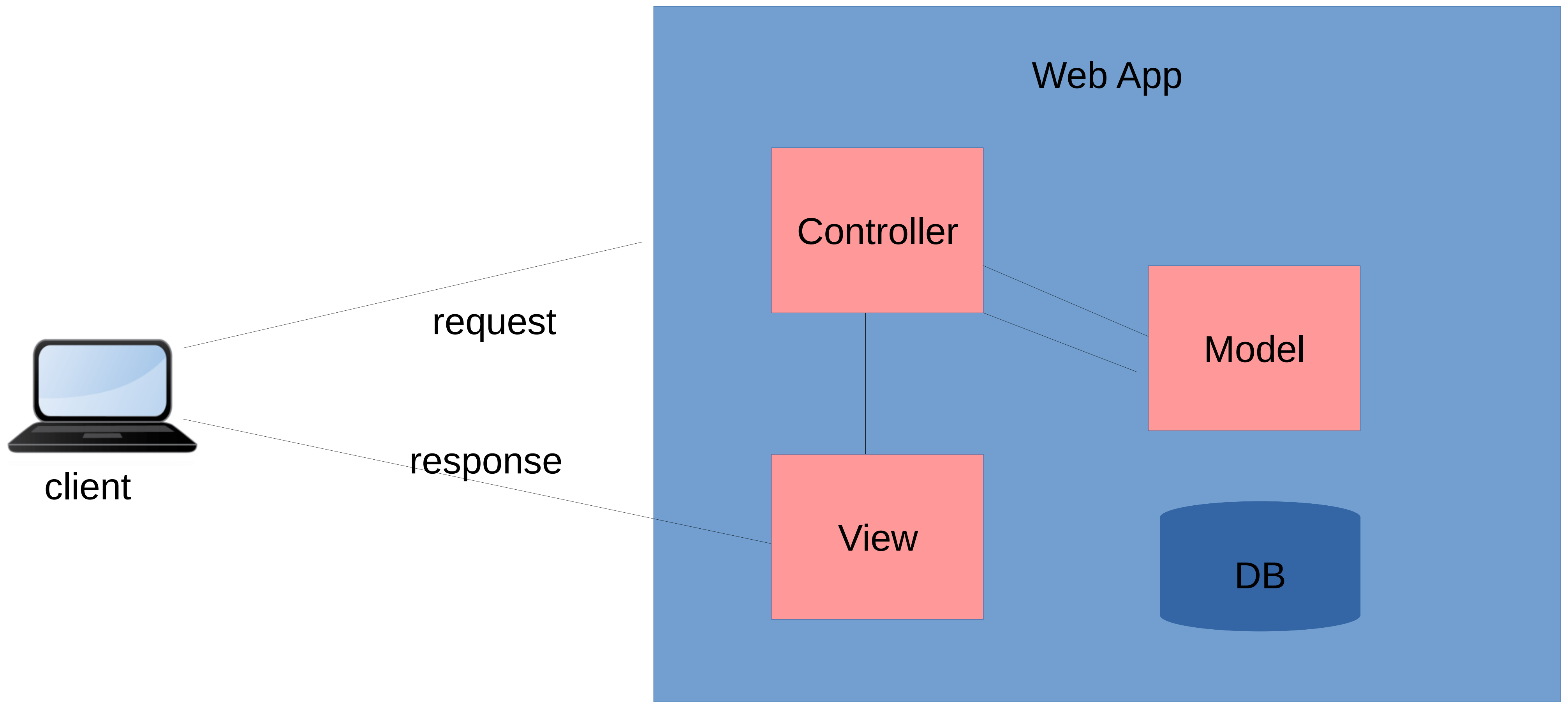
Organizing Domain Logic: “Transaction Script” vs “Domain Model”

- in “Transaction Script” pattern, a single procedure processes the input, validates it, makes calculations, queries/updates the database, and formats the response
    - early CGI perl scripts, Java servlets, were written this way
    - that’s great until:
        - the calculations become more and more complex (coherence pb!)
        - there are increasing overlaps between various scripts (not DRY!)
- the object-oriented alternative is to come up with a “Domain Model”
    - domain logic is organized by classes and relationships

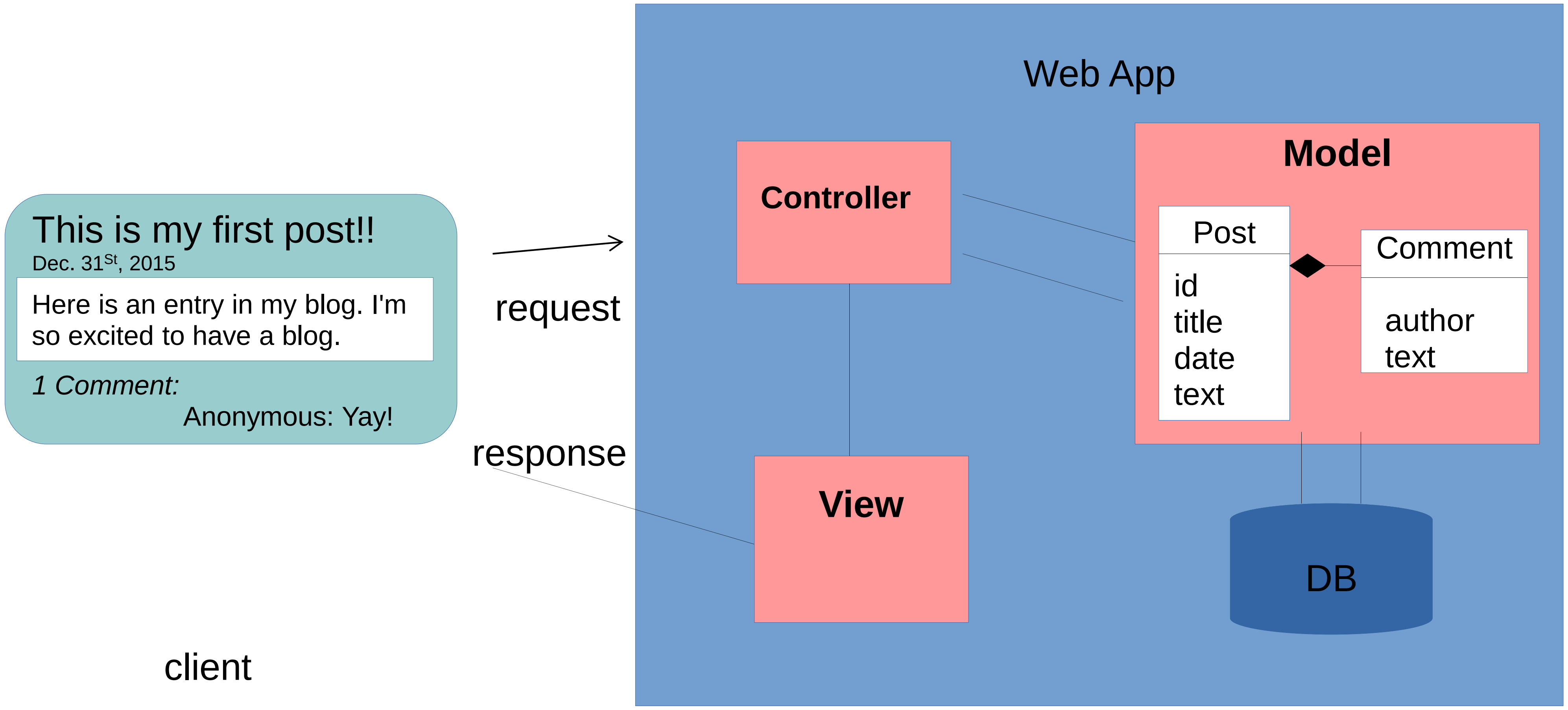
“Service Layer”

- A “Service layer” can be seen as an intermediary between the presentation layer and Domain Model
  - it provides a Façade with a clear API
  - this is where *cross-cutting concerns* such as authentication, transaction management, security can be applied
  - in modern web frameworks, this is where the Controller is
- Domain Model logic can sometimes creep into the Service Layer, but you should resist it
  - keep your controller “thin”

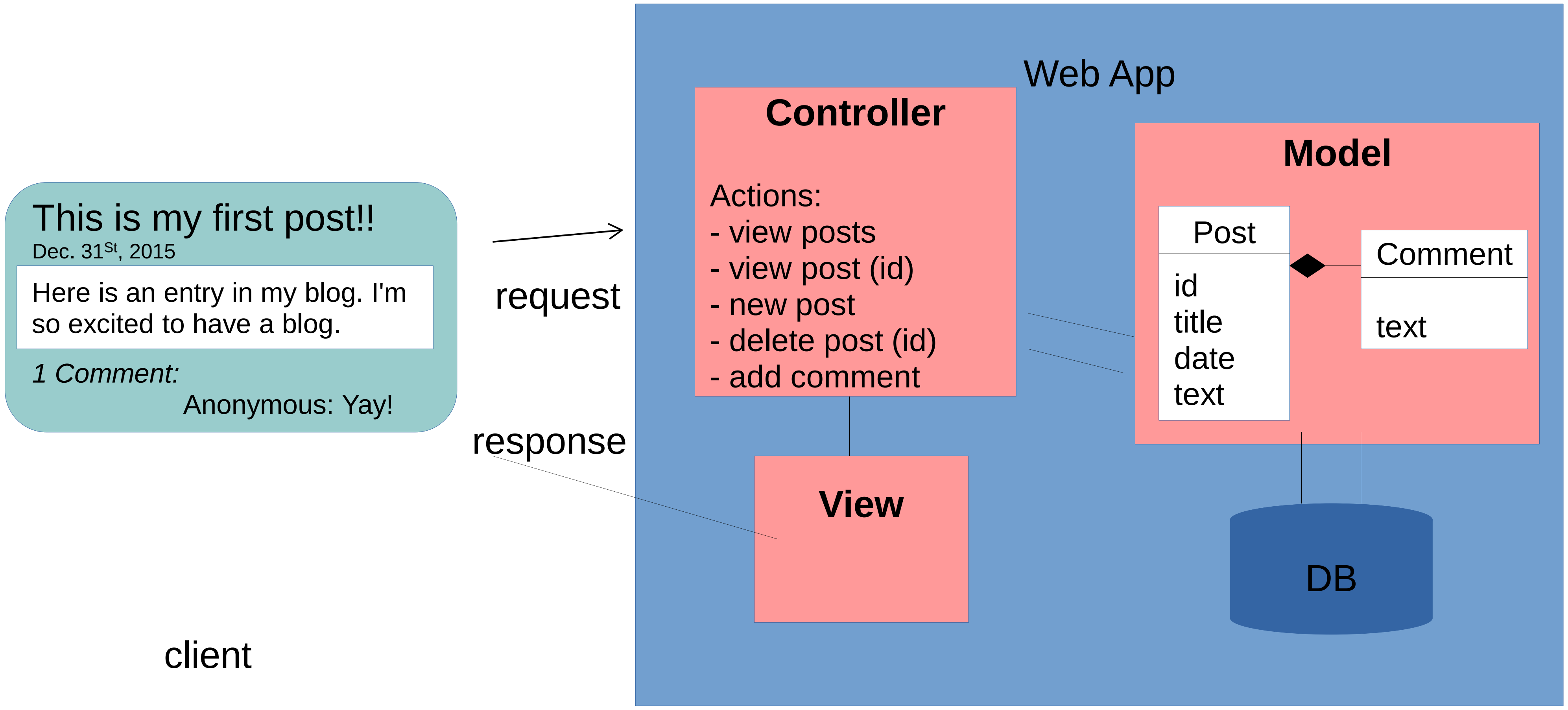
“MVC”



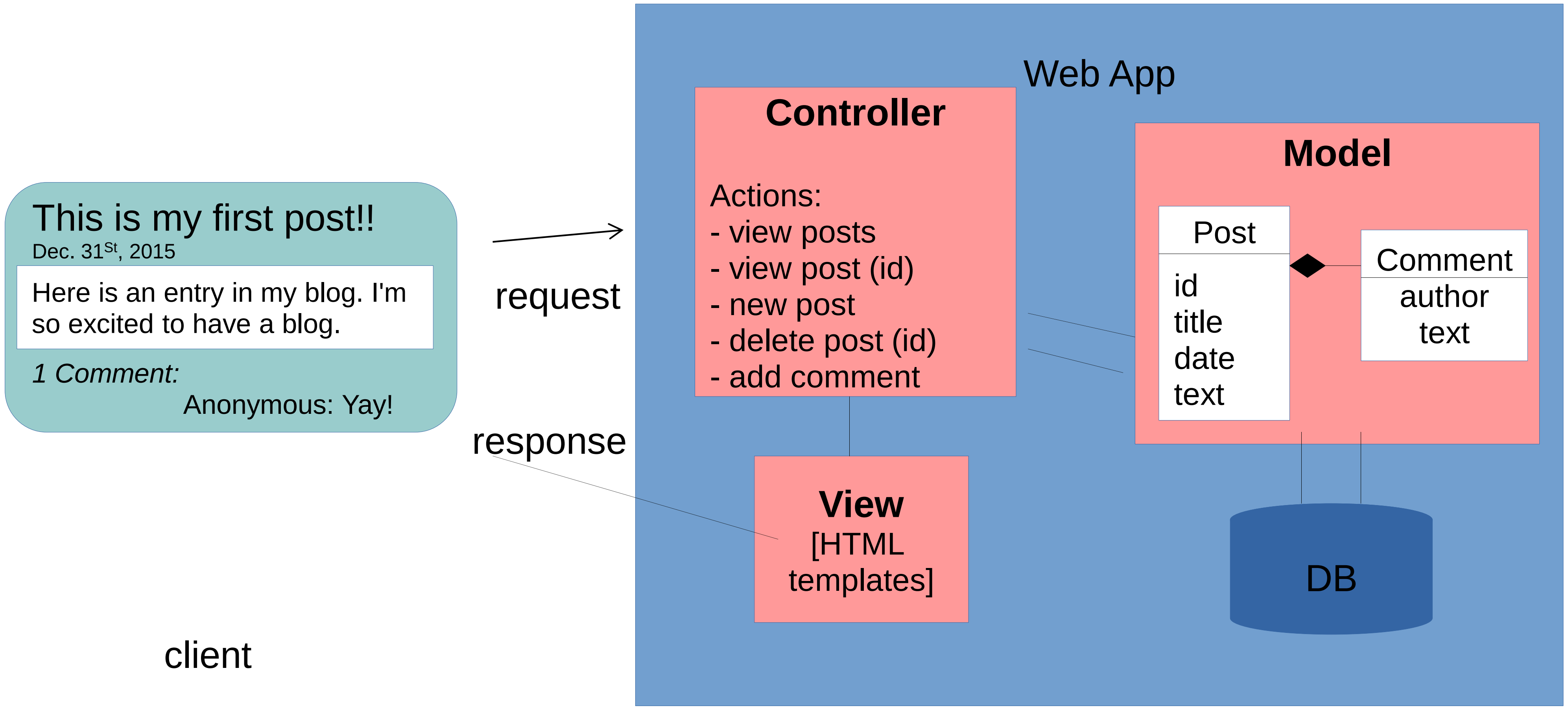
MVC example: My Blog



MVC example: My Blog



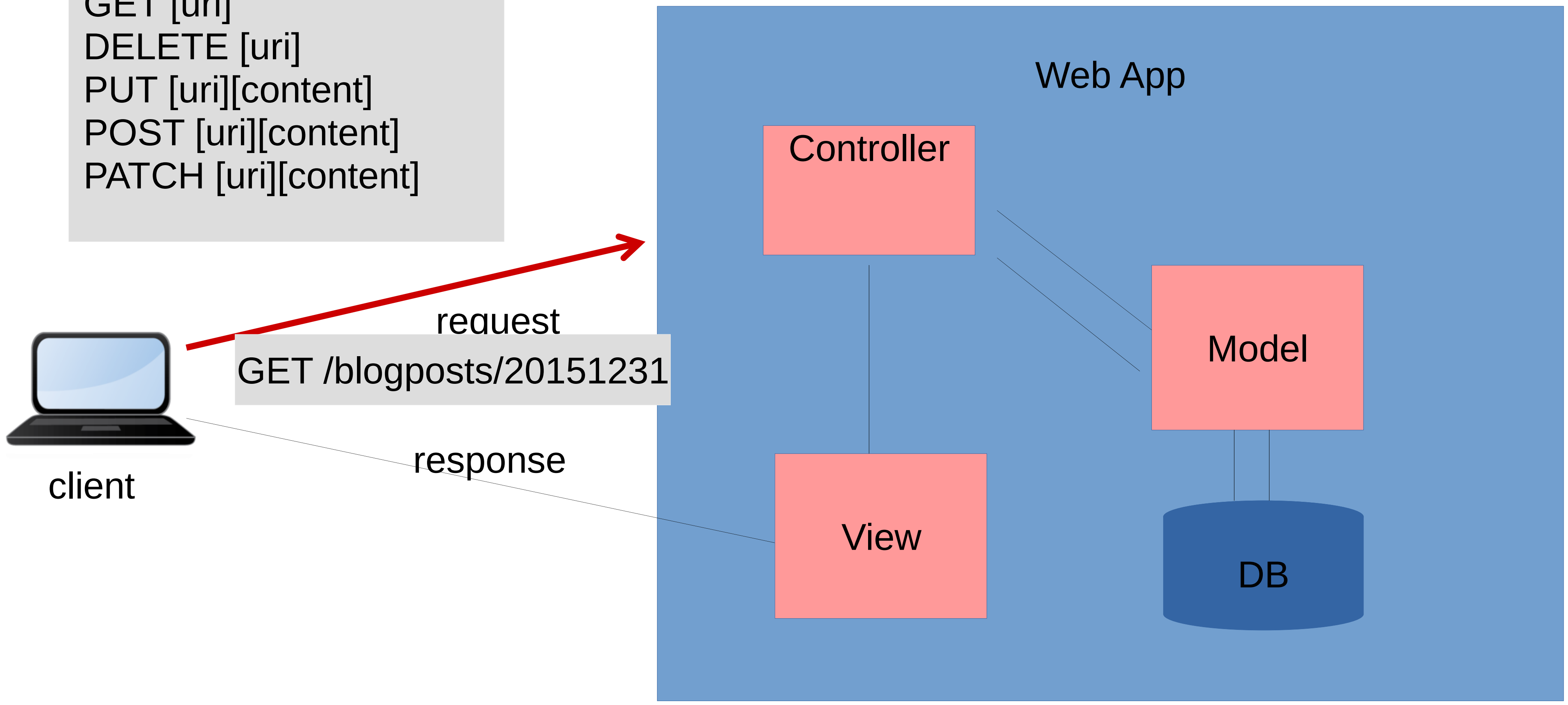
MVC example: My Blog



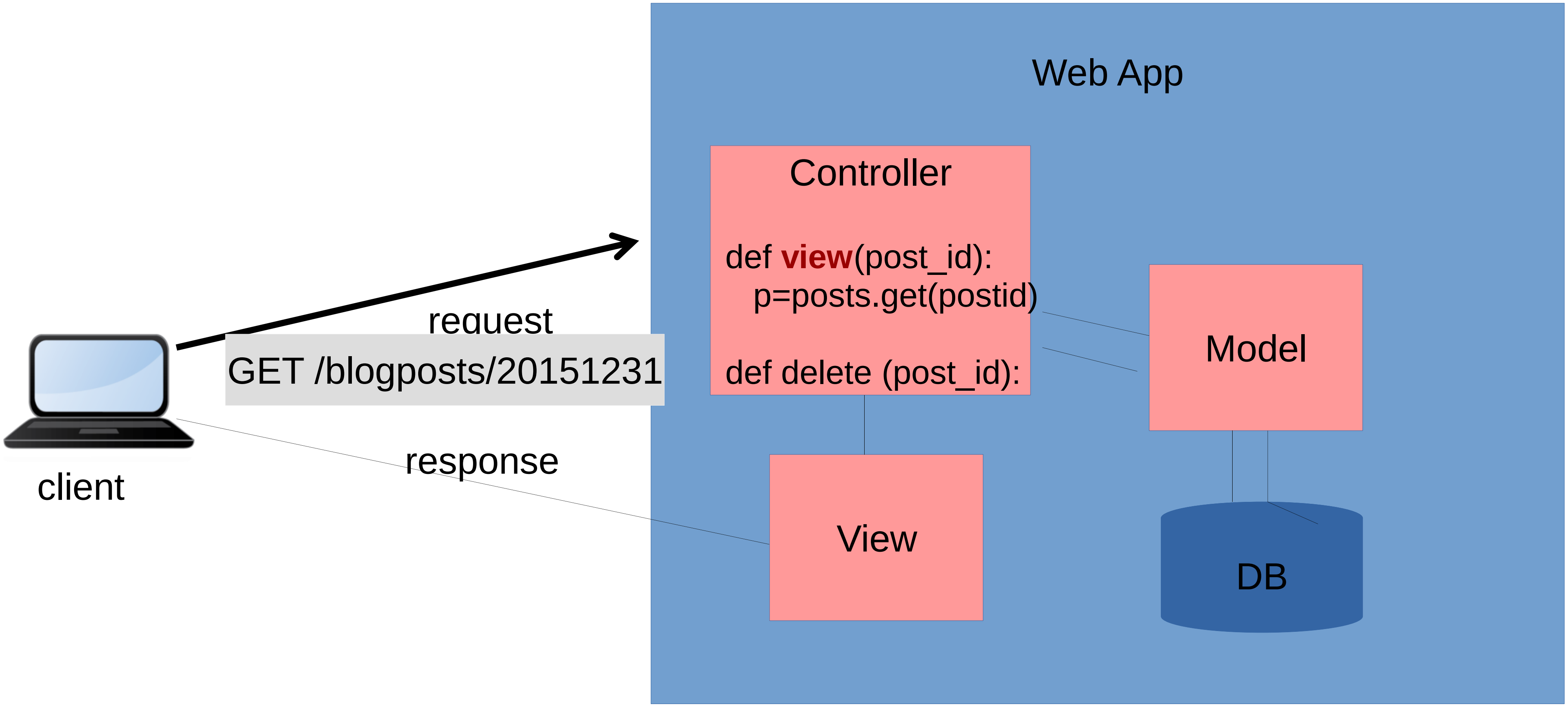
client

MVC

GET [uri]
DELETE [uri]
PUT [uri][content]
POST [uri][content]
PATCH [uri][content]



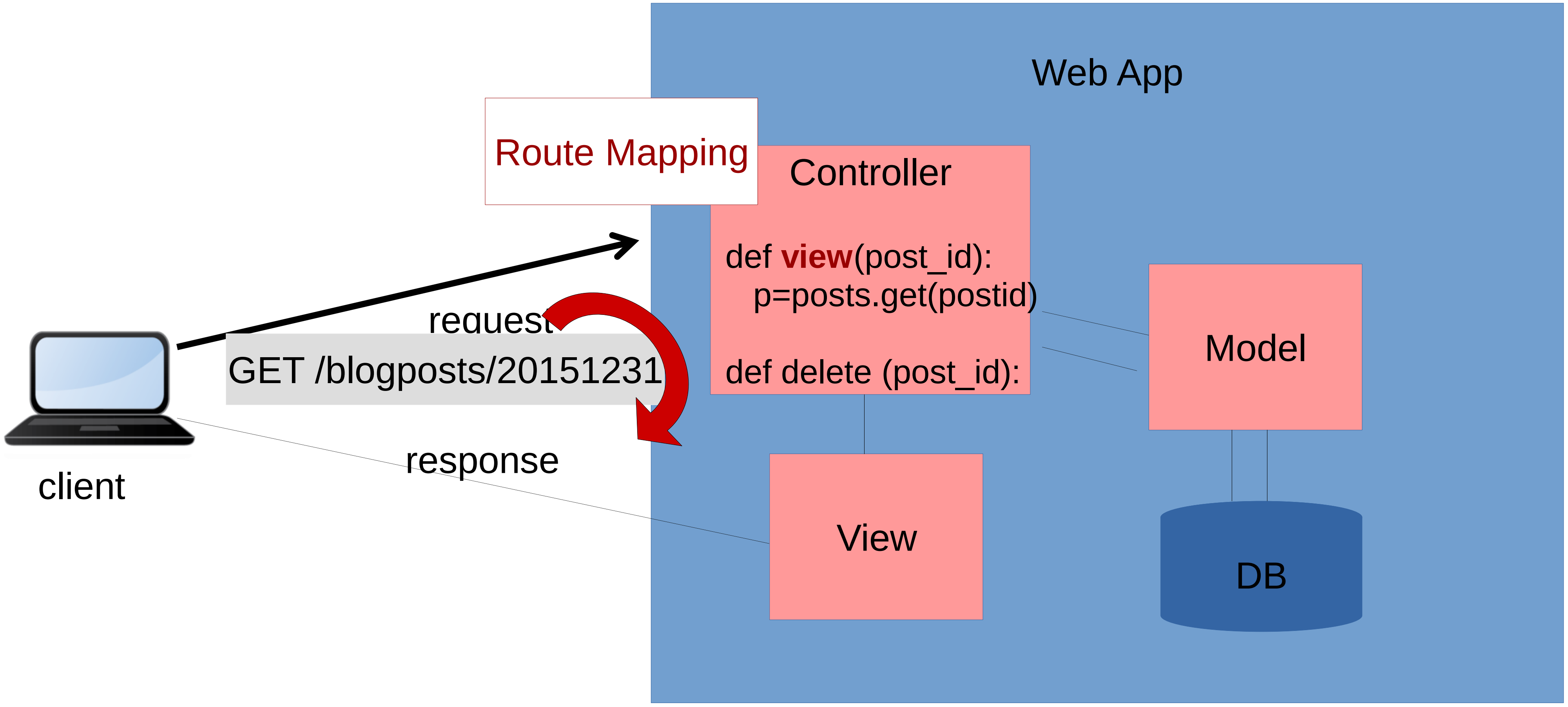
MVC



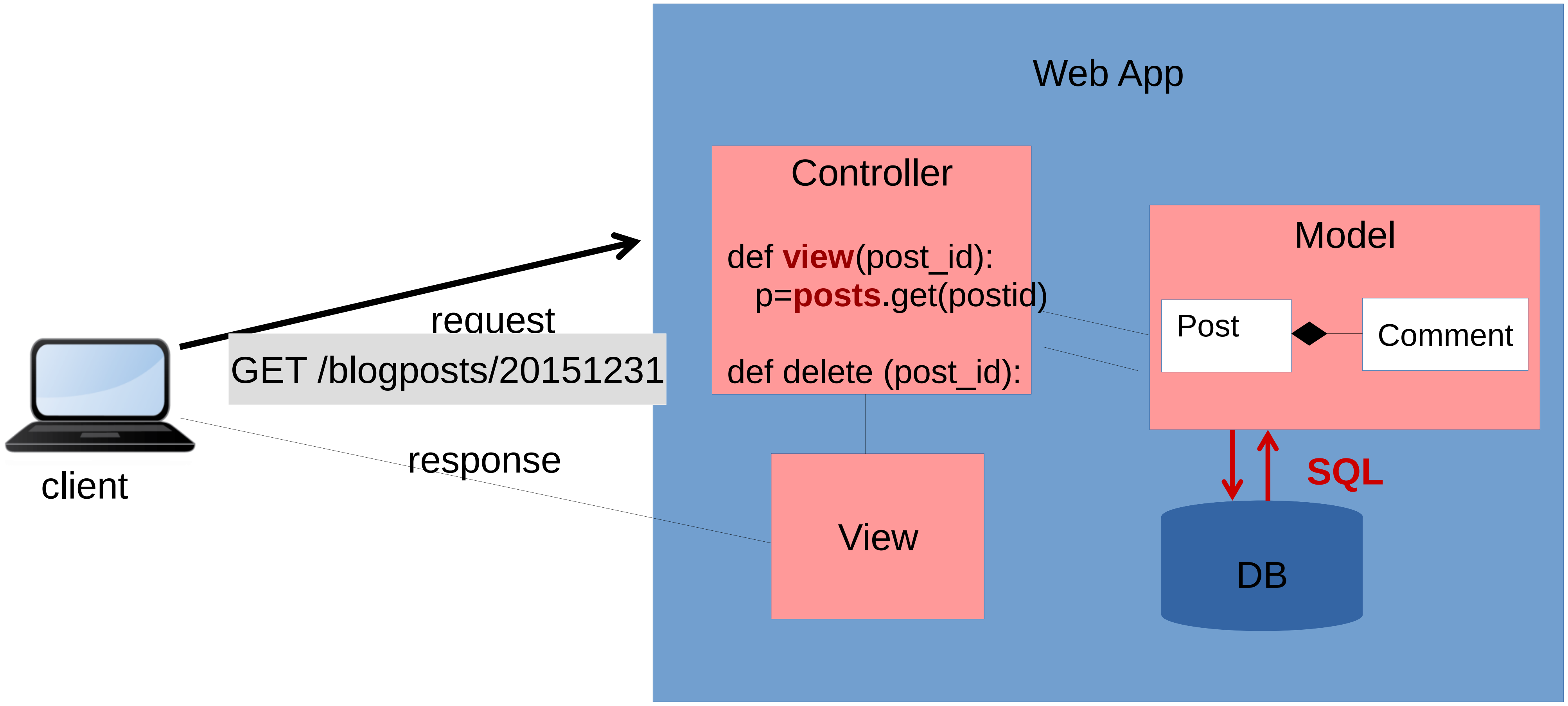
```
def view(post_id):
   p=posts.get(postid)

def delete (post_id):
```

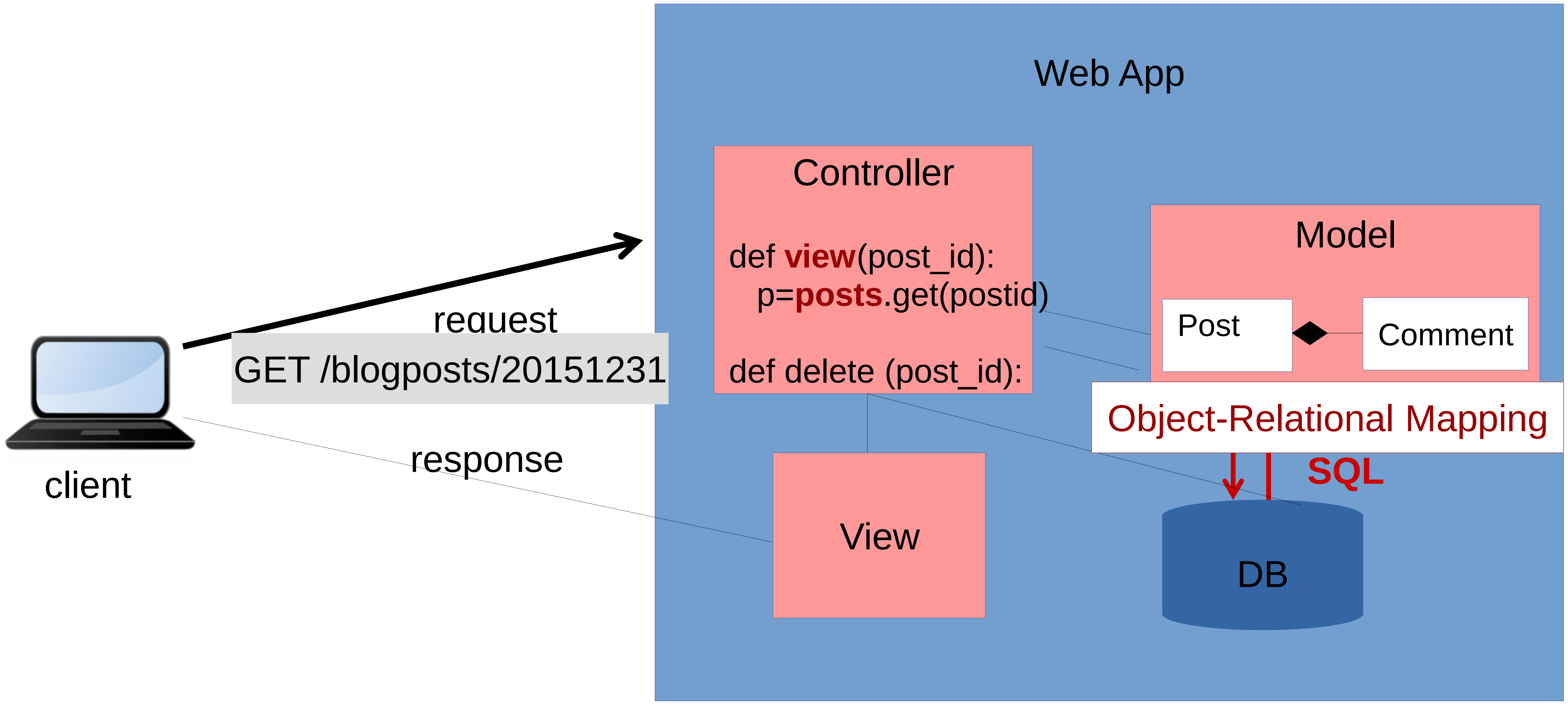
MVC


Web App
Route Mapping
Controller
def view(post_id):
p=posts.get(postid)
def delete (post_id):
Model
View
DB
request
GET /blogposts/20151231
response
client

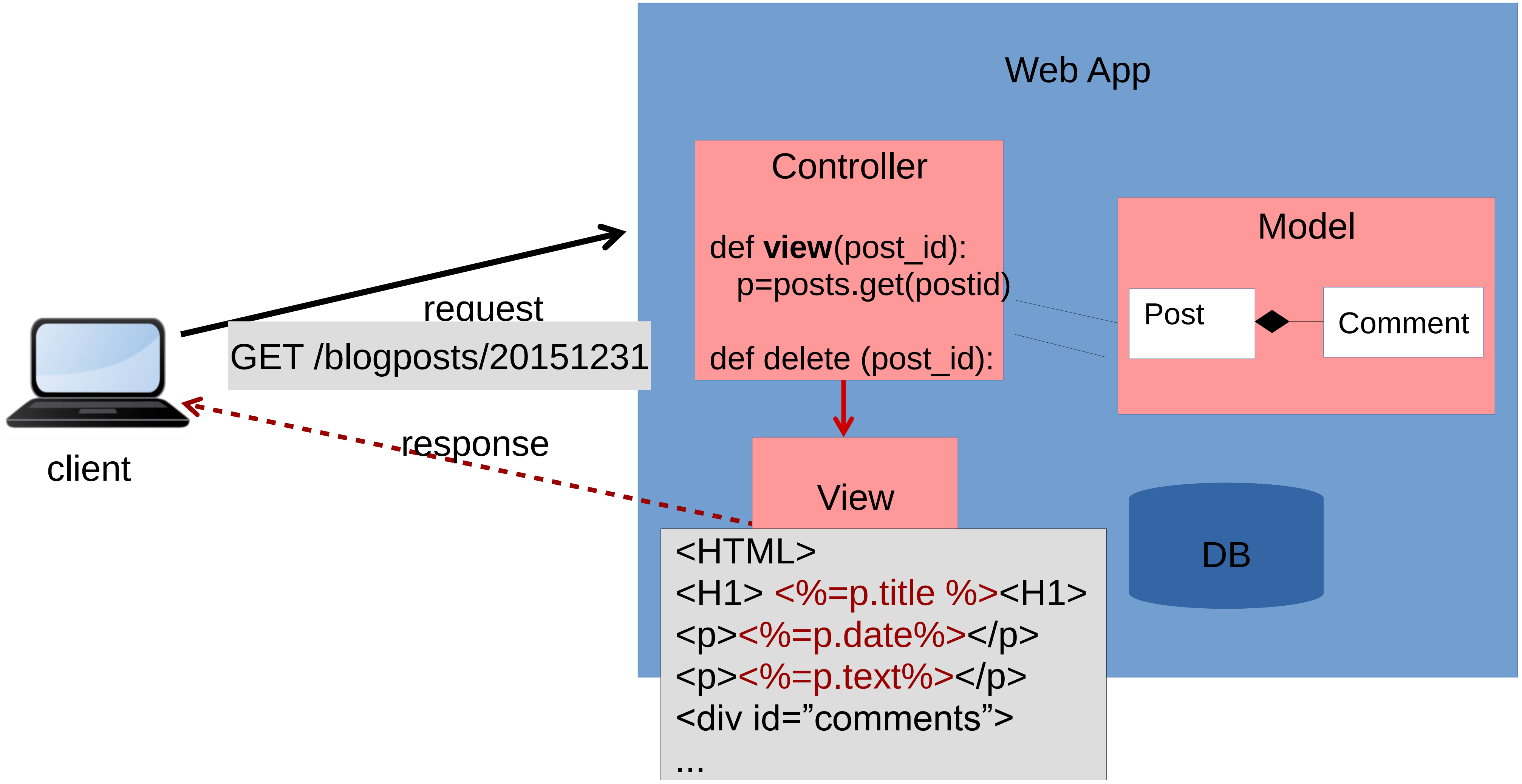
MVC


Web App
Controller
def view(post_id):
p=posts.get(postid)
def delete (post_id):
Model
Post
Comment
SQL
DB
View
request
GET /blogposts/20151231
response
client

MVC


request
GET /blogposts/20151231
response
client
Web App
Controller
def view(post_id):
p=posts.get(postid)
def delete (post_id):
Model
Post
Comment
Object-Relational Mapping
SQL
View
DB

MVC



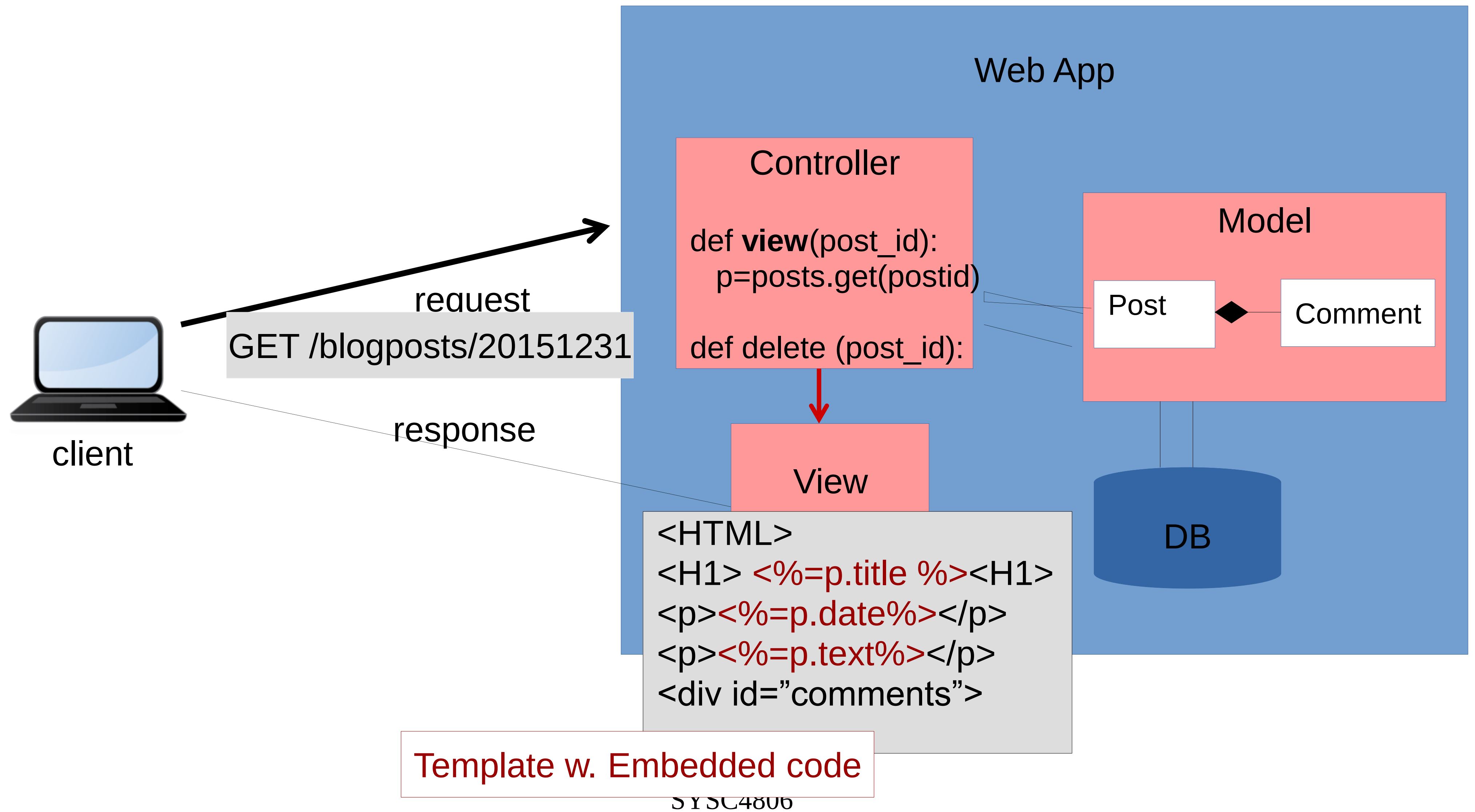
```
GET /blogposts/20151231
```

```
def view(post_id):
   p=posts.get(postid)

def delete (post_id):
```

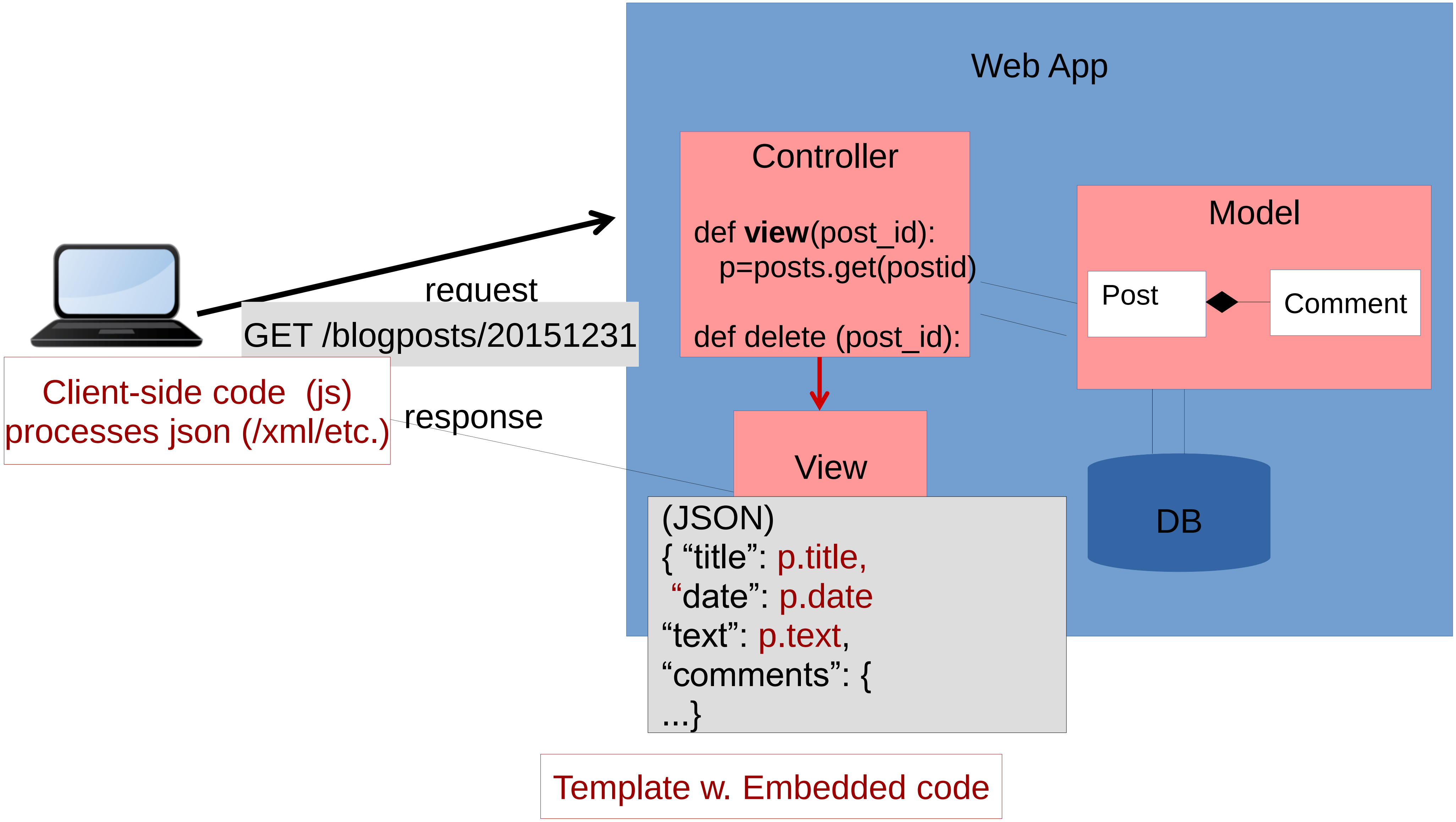
```
<HTML>
<H1> <%=p.title %><H1>
<p><%=p.date%></p>
<p><%=p.text%></p>
<div id=”comments”>
...
```

MVC



```
<HTML>
<H1> <%=p.title %><H1>
<p><%=p.date%></p>
<p><%=p.text%></p>
<div id="comments">
```

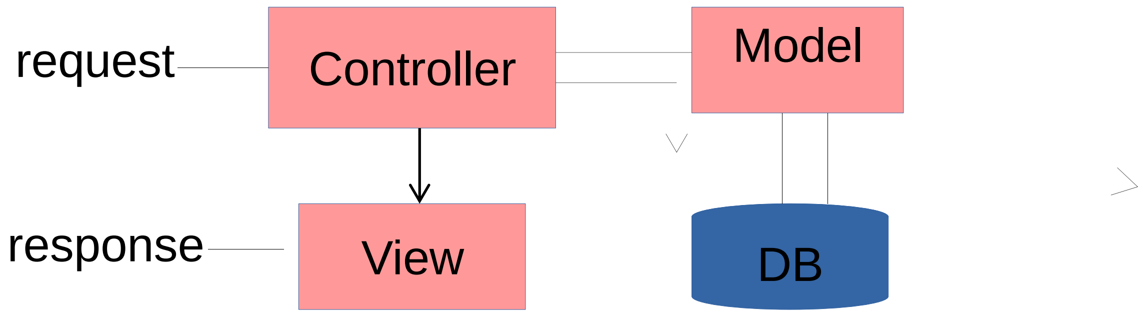
MVC



```
(JSON)
{ “title”: p.title,
 “date”: p.date
“text”: p.text,
“comments”: {
...}
```

Anatomy of a Web App: Summary

- Architecture:



- Languages:

1)Request: HTTP: verb + URI [+content]
2)Controller: [Some programming language]
3)Model to DB: SQL
4)Response: HTML (or JSON, XML, etc.)

- Patterns:
  - Route Mapping, Service Layer
  - Object-relational Mapping (ORM): Data Mapper, ActiveRecord
  - Presentation Layer: Templating