SOEN 387

Web-based Enterprise Application Design

Credit to Dr. Chalin

- These notes are based on his originals

Reminders

- Read the material!
- I need your teams today
  - Sit with you teammates in class

Concurrency

- On the people side this means?
- On the computer side this means?

When Does it Go Wrong?

When Does it Go Wrong?

- Resources are shared?
- Shared resources are changed?

What can be done?

- Transaction mechanisms
- Better yet, avoid sharing where you don’t have to
  - (or assign responsibility of dealing with sharing to someone else)

Contexts

- More than a means of limiting sharing in a concurrent environment
- Still, that’s a good first reason to consider

What Contexts

- We’ll only concern ourselves with a few, but thing really big and really small.

Contexts

- *Server*
- Application
- Session
- Request
- *Page*

What should the interface be?

- What is it like in other languages?
- How does this relate to parameters?
- Are there any sensible short-hands?

Moving on to MVC

- Model View Controller
  - It gets lots of air-time on the web
  - It’s more and less

MVC, the pieces

- Recall GRASP Controller
  - Takes care of “Who handles system events”

Two flavors of Controller

- Represents Overall System
- Represents Use-Case Scenarios

Which one?

- Consider the question:
  - Who handles system events
- What are those system events?

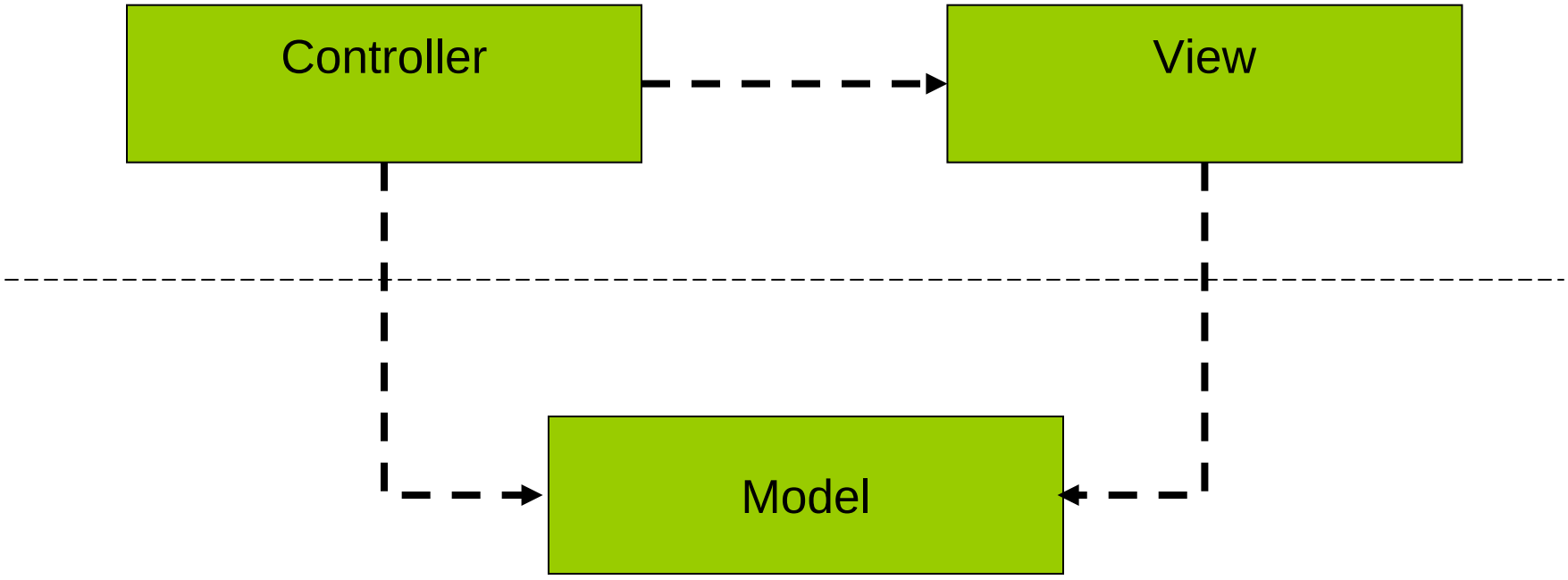
Controller choice

- Really, as we'll see, it can be either type of controller.
- As we learn more patterns, we see that arguments for both can be right… vague, eh?

What will our controller do?

- MVC:
  - Delegate to Model for domain logic
  - Pass part of model to the view

Note the Dependencies



Model relates to View how?

Model independent of View

- What advantage does this give us?
  - Windowed
  - Webapp
  - Console

MVC for Hello World?

- What are the elements?
- What are they in terms of servlets?

Nitty Gritty of the MVC

- Controller doGet() body
  -3 parts (what do they do?):

```
String name = request.getParameter("name");

Greeter greeter = new Greeter(name);
request.setAttribute("Greeter", greeter);

RequestDispatcher view = request.
  getRequestDispatcher("Greeting.jsp");
view.forward(request,response);
```

What about the View

- Greeting.jsp

```
<%@ page language="java" contentType="text/html…"
    import="Greeter" %>
<% Greeter greeter =
    (Greeter)request.getAttribute("Greeter"); %>
…
<html>
…
<body>
<p><%= greeter.getGreeting() %> from JSP page.</p>
</body>
</html>
```

Selling you on EL

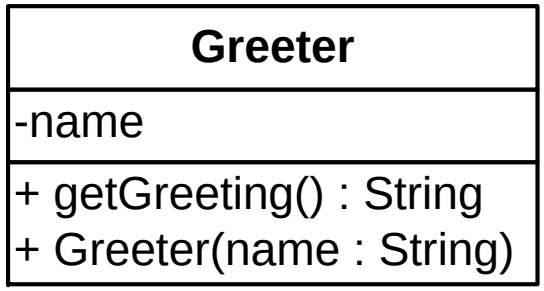
- Greeting.jsp

```
...
<html>
...
<body>
<p>${greeter.greeting} from JSP page.</p>
</body>
</html>
```

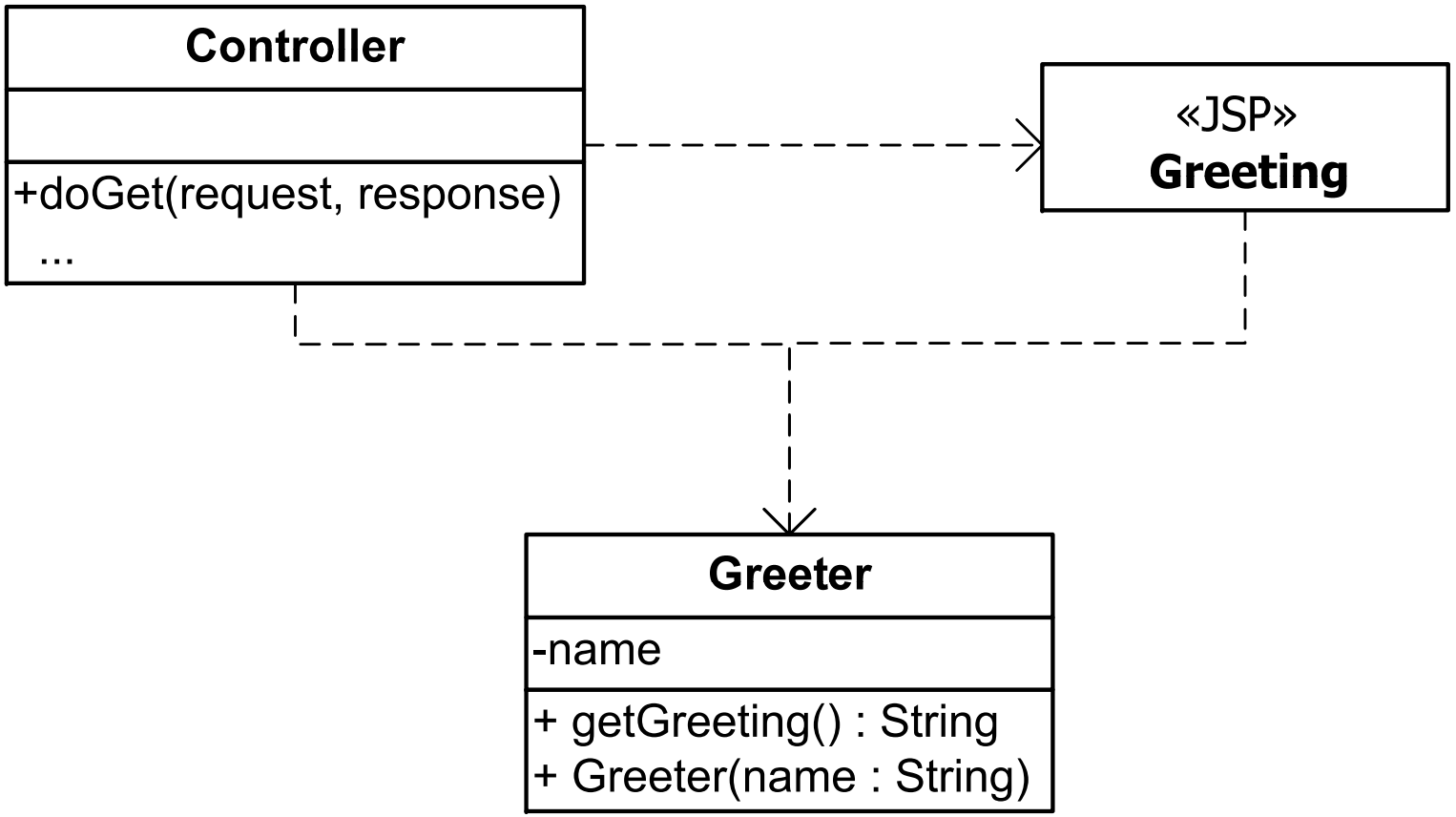
Greeter is a POJO

- Uninteresting, but useful

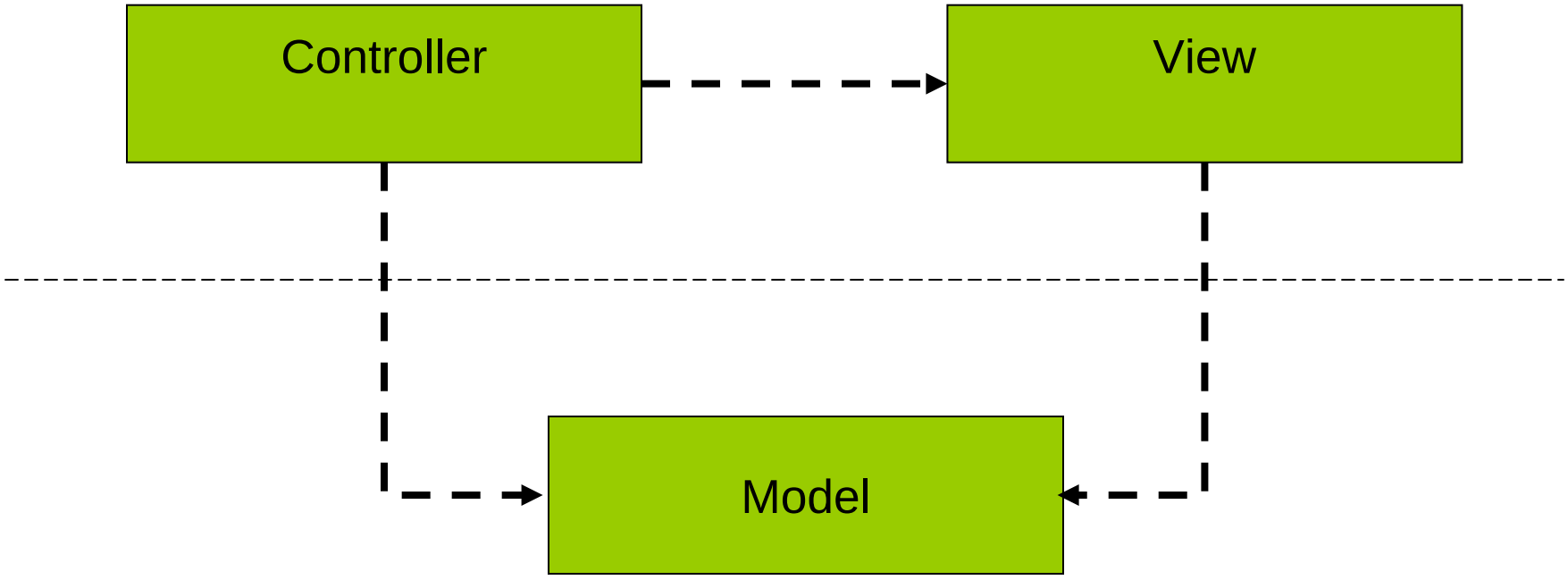
What are the Dependencies?



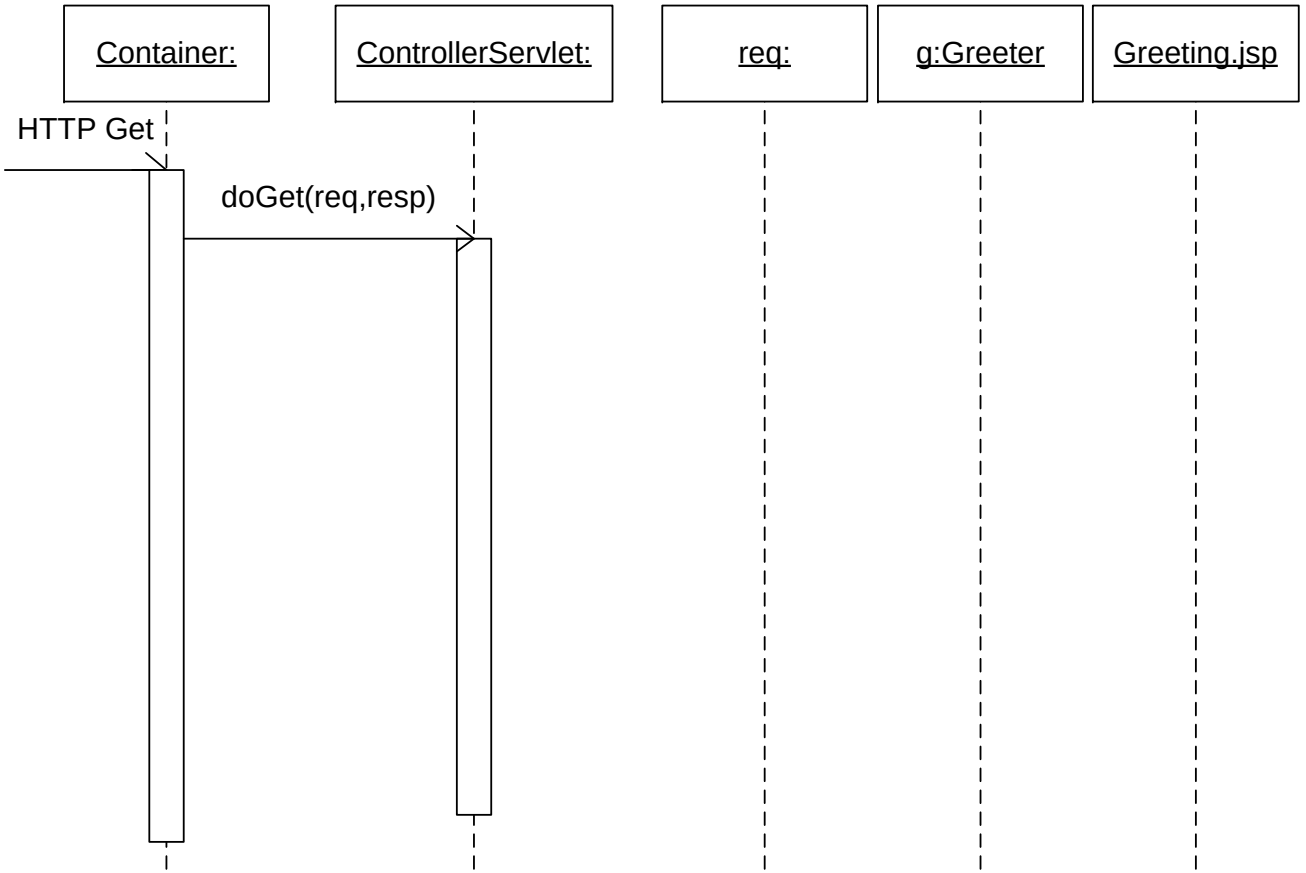
Look familiar?



Note the Dependencies



What Happens? Fill out and hand in at end of class. Take 5 now.



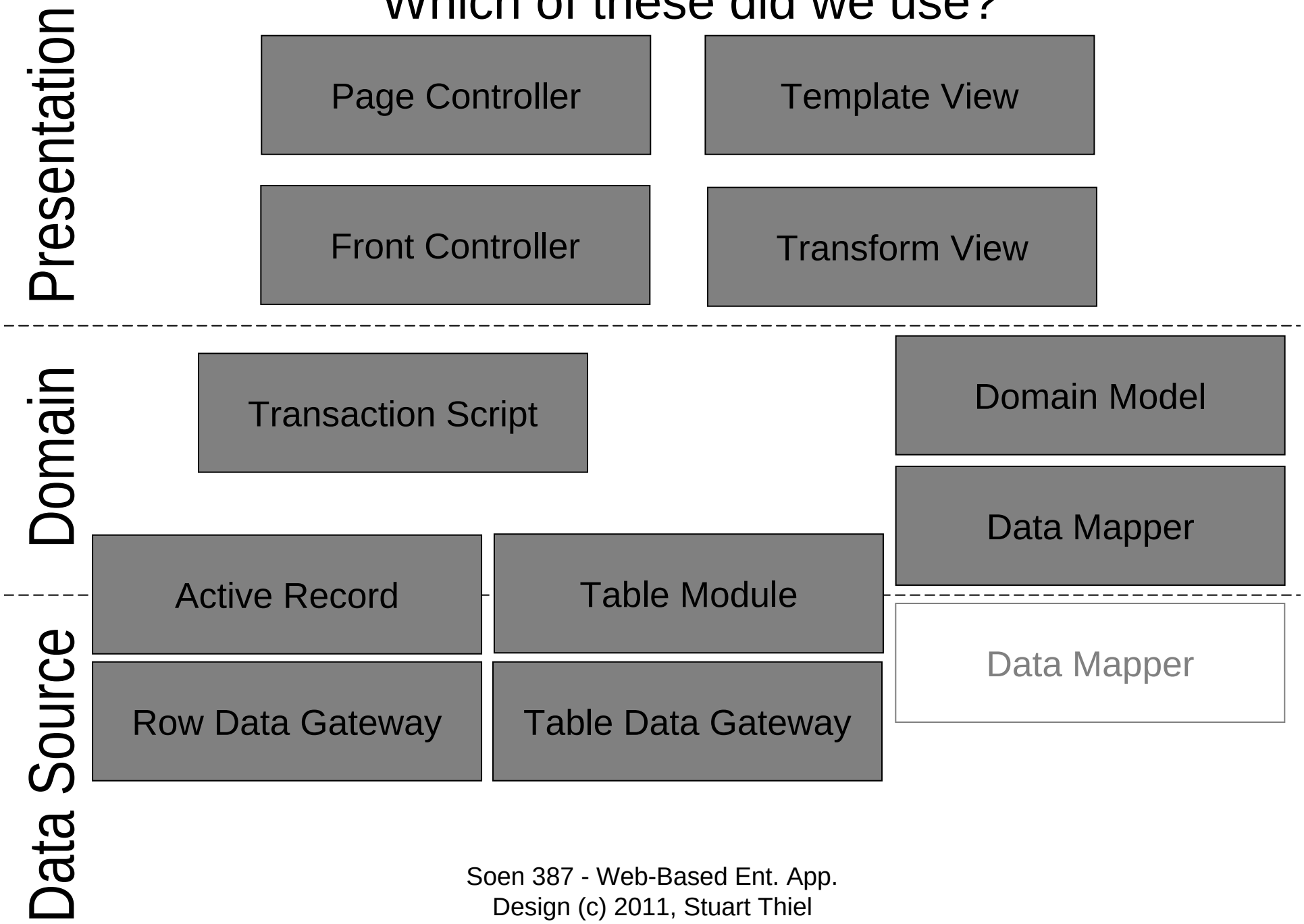
A quick question while we Segue

- What are we missing to make our example plausibly EA-like?

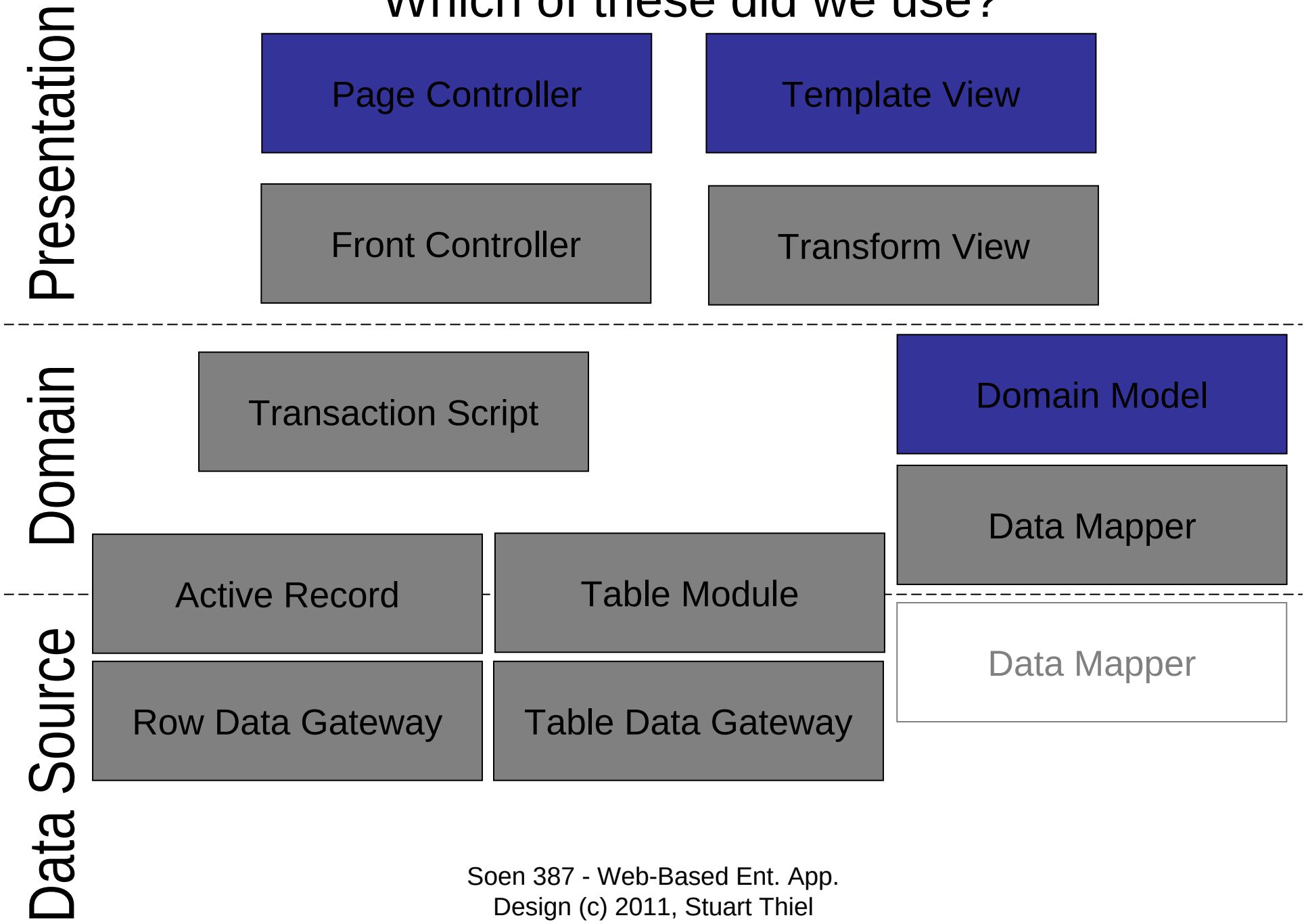
A Reminder

- Fowler is a pattern reference
  - Narrative offers some progression
- My thesis offers much more progression
  - Refinements later
  - Additions later

Which of these did we use?

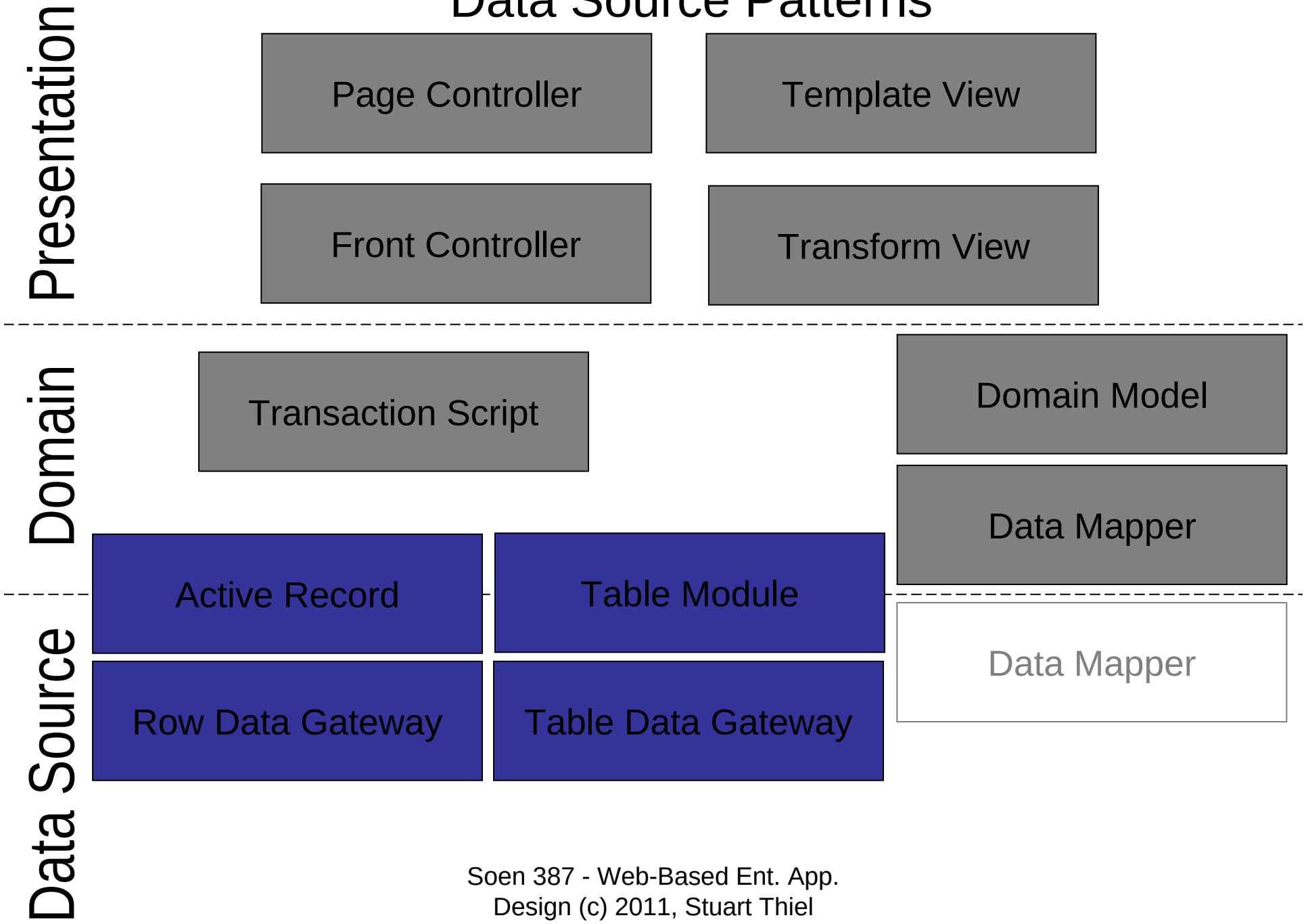


Which of these did we use?



So, what are we missing?

Data Source Patterns

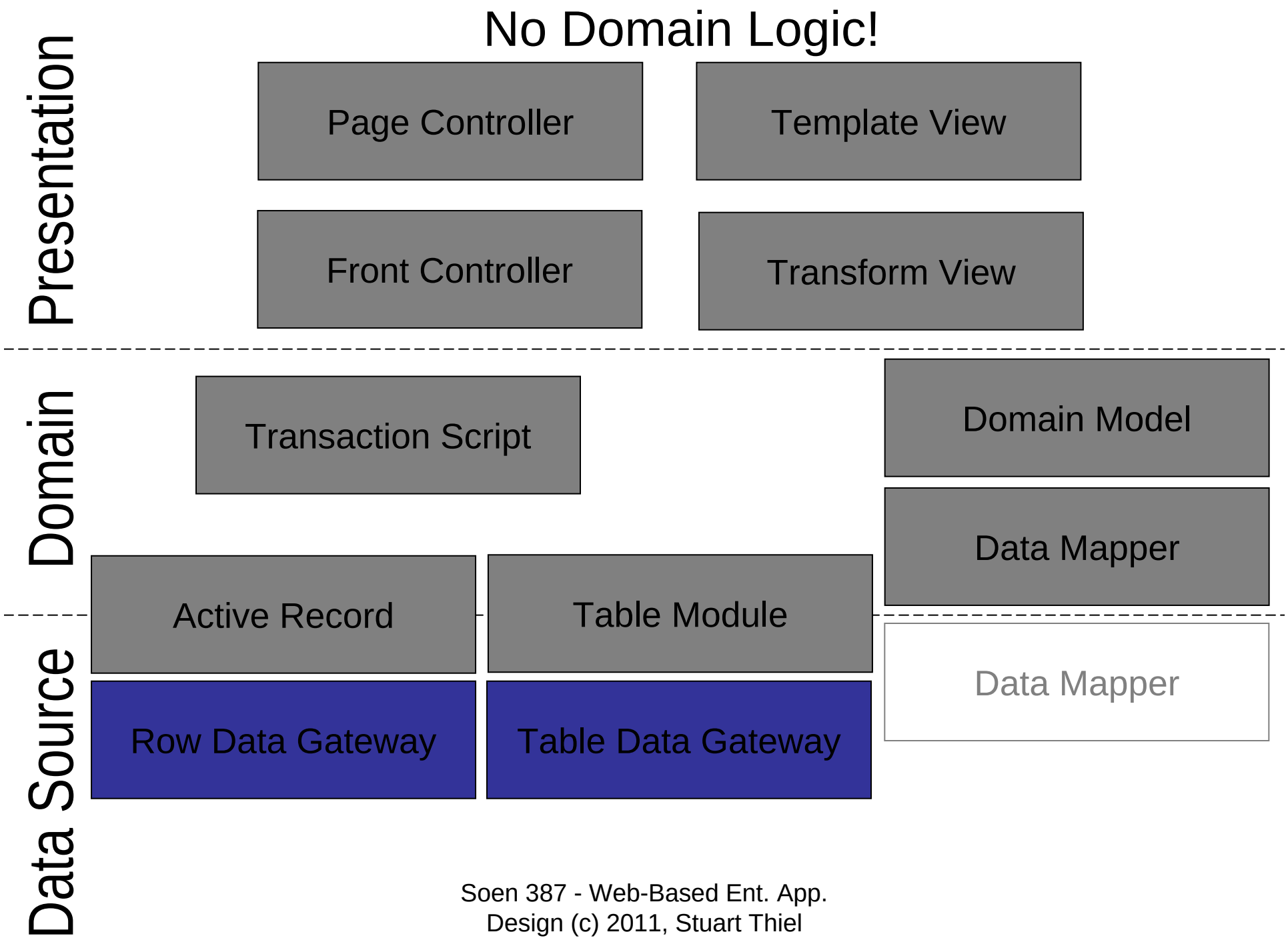


Datasource Responsibilities

- Hide SQL
- Provide abstraction
  - One row
  - Rows from a table
  - Rows from a view (is that different?)

Common Datasource

- RDG
  - instance as row
- TDG
  - no instances, just resultsets


No Domain Logic!
Presentation
Page Controller
Template View
Front Controller
Transform View
Domain
Transaction Script
Domain Model
Data Mapper
Active Record
Table Module
Data Source
Data Mapper
Row Data Gateway
Table Data Gateway

Active Records

- RDGs with Domain Logic
  - I avoid them, but sometimes expedient
  - You will see these in the wild

Example: Person-Grade Table

Table attributes:

- name : String
- grade : int

name	grade

Row Data Gateway

PersGradeRDG
- name : String - grade : int

Row Data Gateway

PersGradeRDG
- name : String -grade : int
+ PersGradeRDG(name, grade) + find(name) : PersGradeRDG + findInRange(fromGr,toGr): List<PersGradeRDG> + insert() : void + update() : void + delete() : void + … getters and setters …

No setter for primary key

Row Data Gateway: Find Code

```
public static RowDataGateway find(String name)
{
    PreparedStatement findStatement = null;
    ResultSet rs = null;
    try {
        findStatement = db.prepareStatement(findPersSQL);
        findStatement.setString(1, name);
        rs = findStatement.executeQuery();
        if(!rs.next()) {
            return null;
        }
        int grade = rs.getInt("grade");
        RowDataGateway result =
            new RowDataGateway(name, grade);
        return result;
    } catch (SQLException e) {
        return null;
    }
```

Table Data Gateway

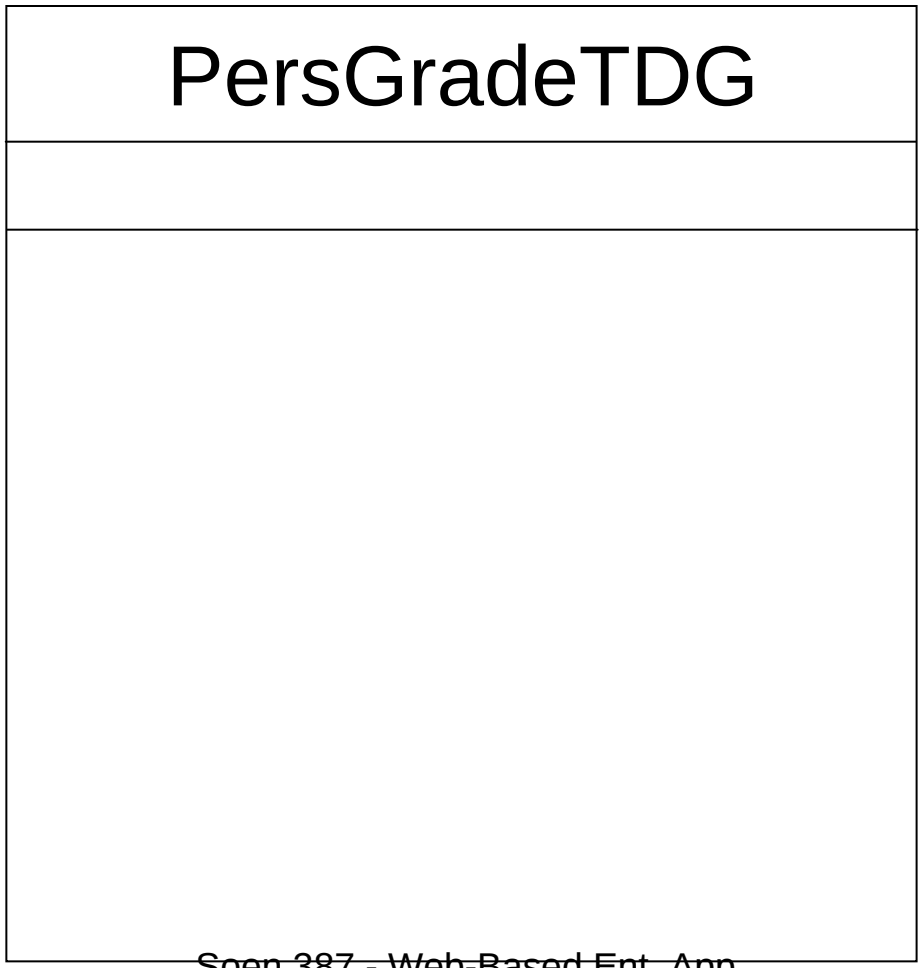


Table Data Gateway

PersGradeTDG
- PersGradeTDG() + find(name) : ResultSet + findInRange(fg,tg) : ResultSet + insert(name,grade) : void + update(name,grade) : void + delete(name) : void

Table Data Gateway: Find Code

```
public static ResultSet find(String name)
{
    PreparedStatement findStatement = null;
    ResultSet rs = null;
    try {
        findStatement = db.prepareStatement(findPersSQL);
        findStatement.setString(1, name);
        rs = findStatement.executeQuery();
        return rs;
    } catch (SQLException e) {
        return null;
    }
}
```

Table Data Gateway: FindInRange

```
public static ResultSet findInRange(int fromGrade, int toGrade)
{
    PreparedStatement findStatement = null;
    ResultSet rs = null;
    try {
        findStatement = db.prepareStatement(findGradesSQL);
        findStatement.setInt(1, fromGrade);
        findStatement.setInt(2, toGrade);
        rs = findStatement.executeQuery();
        return rs;
    } catch (SQLException e) {
        return null;
    }
```

Active Record (Row Data Gateway)

PersGradeAR
name : String grade : int

PersGradeAR(name, g)
find(name)
… // like RDG
// Can also have domain logic
getRank()