



NUS | Computing

National University
of Singapore

IT 5007: Software Engineering on Application Architecture Lecture-5

Prasanna Karthik Vairam
Lecturer
Department of Computer Science
NUS School of Computing



National University
of Singapore

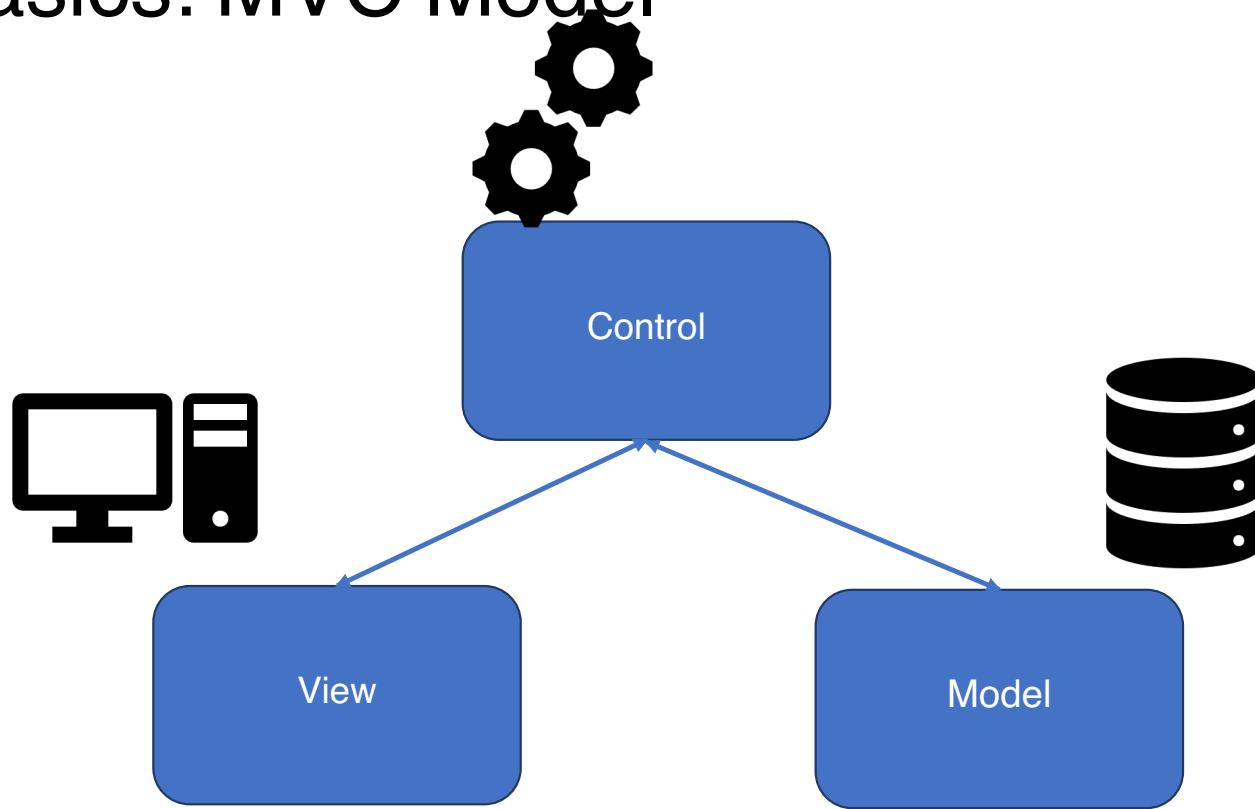
NUS | Computing

React: Motivation

What exactly is React?

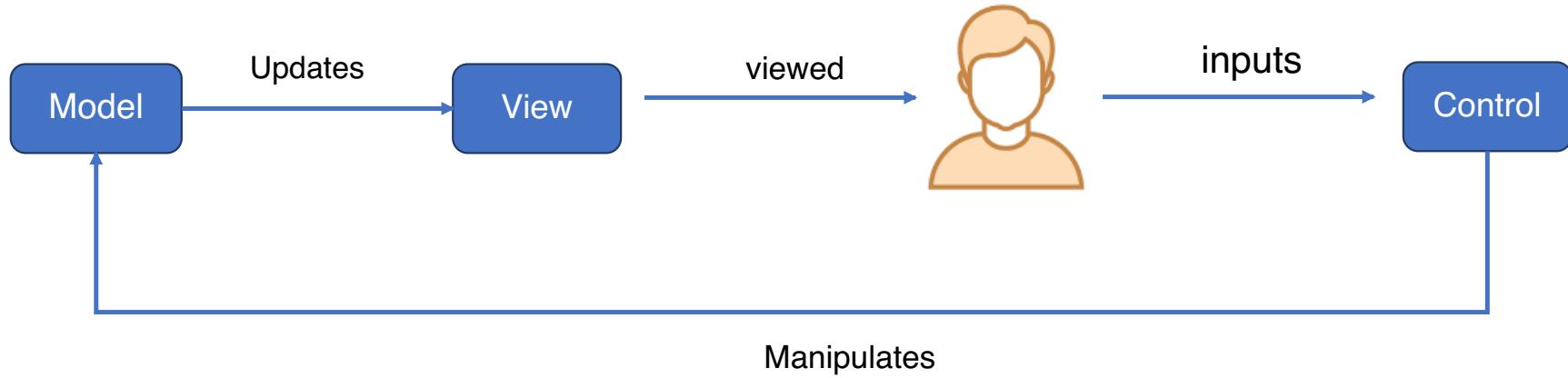
- React may sound like a lot of things.
- React is a high level language that generates JS code.
- It is just a JS library that you include to your HTML file
- **Digression:** What is a Library?
- **Why is it popular?:** React Provides numerous off-the-shelf libraries (components) for you to use. (DOM query, charting, math, etc.)
- **Importantly:** React allows you to create your own custom components from these off-the-shelf components.
- **More importantly:** React defines some nice rules that make coding and debugging easier.
- **Most importantly:** It binds the View and Model to boost performance.
- What is the intuition? Primevideo example.

Basics: MVC Model



Isn't this what JS does already?

Javascript: The FB Ad story



Curious case of cascading updates!

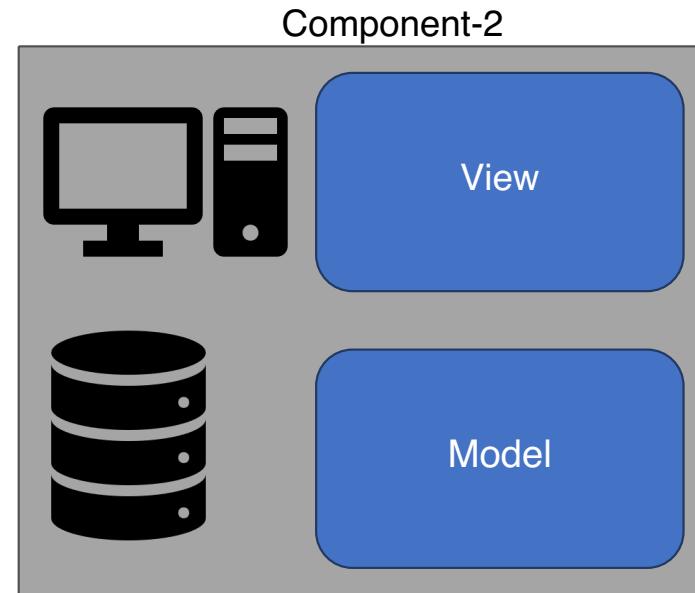
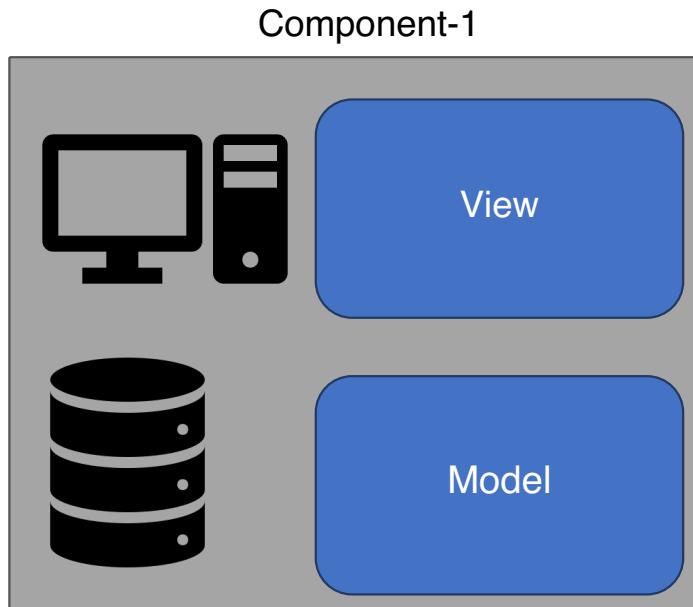


NUS | Computing

National University
of Singapore

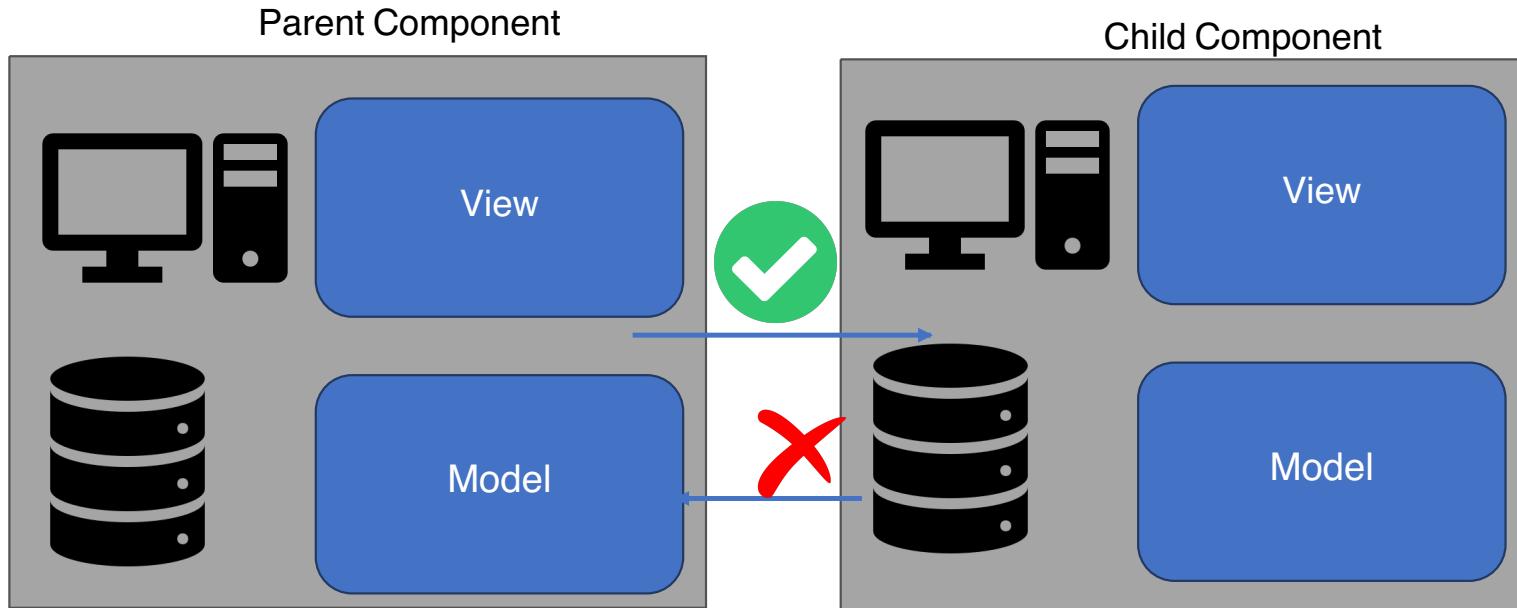
React: Features

Motivation for React: Component based



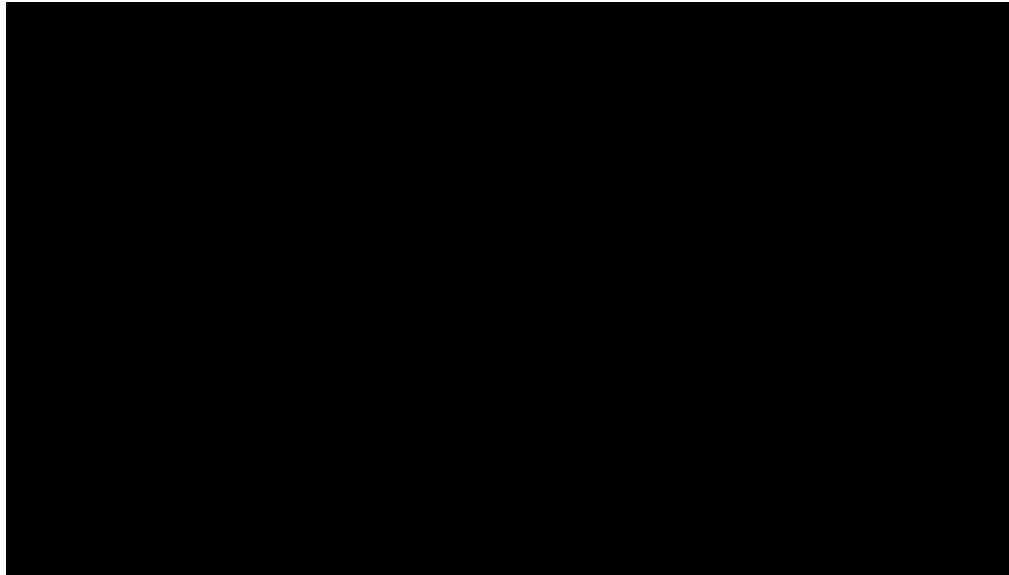
Components are self-contained and take care of themselves.

Motivation for React: Data flow Rules



Unlike JS, we do not allow data to flow randomly!

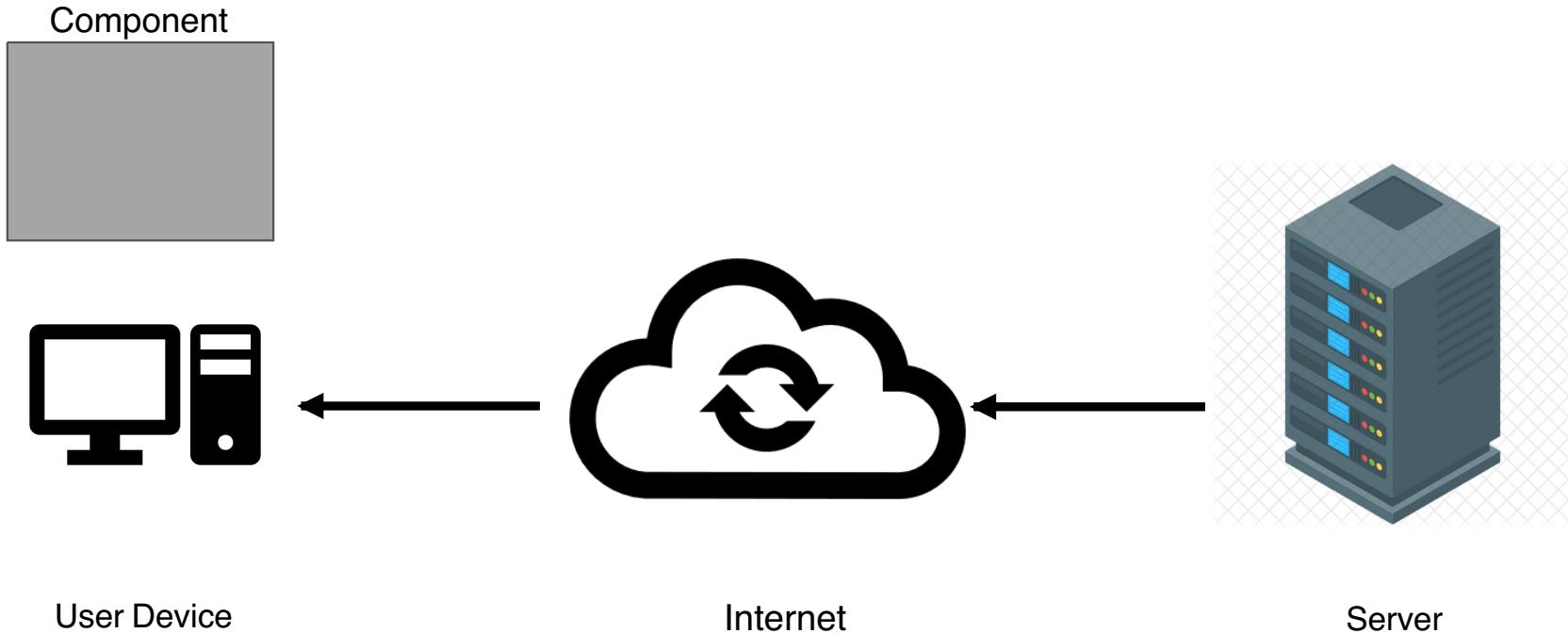
Motivation for React: Data flow Rules



That's like how most developers handle data in JS!

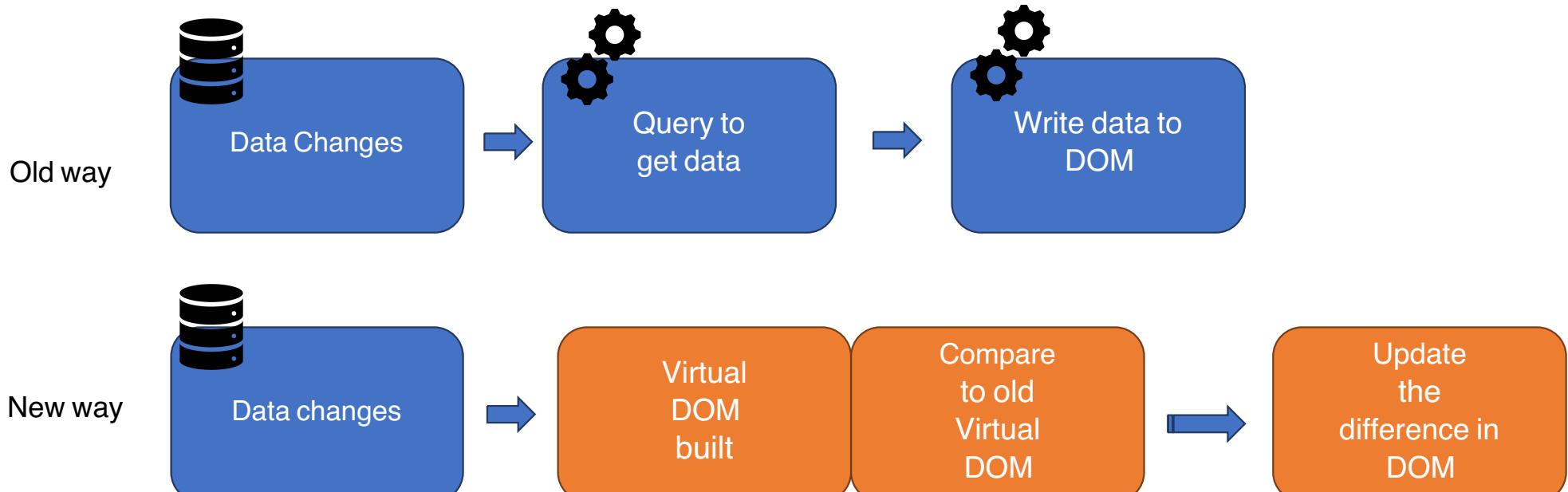
Motivation for React: Isomorphic

Write once – decide where to host later!



Motivation for React: Efficiency

Assume you are creating a <table> from the data.



Can you guess what happens in Facebook?



National University
of Singapore

NUS | Computing

React/JS: JS Class Basics and Class Components

Class Definition (Js)

- Class is a **generic template** for creating many objects/instances.
- E.g., a Student class can be used to create 100 real student objects/instances.
- Class is Student and s1/s2 are **instances**.
- **Constructor** is called when an instance is created and its objective is to initialize the object.
- Class can have **variables** and **functions**.
- **this.name** refers to the current object's name.
 - E.g., when s1 is created, this.name refers to s1's name.

app1

```
class Student {  
    constructor(x_name, x_matricno)  
    {  
        this.name = x_name;  
        this.matricno = x_matricno;  
    }  
    getname()  
    {  
        return this.name;  
    }  
    getmatricnumber()  
    {  
        return this.matricno;  
    }  
}  
var, let  
const s1 = new Student("Prasanna", 1);  
const s2 = new Student("Karthik", 2);  
s1, name
```

CodePen – Example 1

- Example1: <https://codepen.io/pkarthik88/pen/oNdjoNm>
- Exercise:
 - Add one more field (NUSNETID) to student class.
 - Add get function for new field.
 - Update the instance creation code (s1 and s2).

Inheritance

- Canvas requires some details of students
 - Student class
- IT5007 may require “additional” information from students.
 - IT5007Student class.
- The idea is to **reuse** the variables/functions in Student class and add more variables/functions.
- We use *extends* to inherit from a base/parent class.
- *Super* is used to call the base class constructor. Why?

The image shows a code editor with two classes: `Student` and `IT5007Student`. The `Student` class has a constructor that initializes `this.studentname` and a `getname()` method. The `IT5007Student` class extends `Student` and adds a constructor for projects and a `getprojectname()` method. Handwritten annotations include:

- A blue circle highlights the `super(name)` call in the `IT5007Student` constructor, with arrows pointing to it from the text "add. for" and "super".
- A bracket labeled "add. for" is placed under the `super(name)` call.
- An arrow points from the word "super" to the `super` keyword in the call.
- A bracket labeled "super" is placed under the `super` keyword.
- A bracket labeled "base class parent class" is placed around the `Student` class name.
- A bracket labeled "2 var" is placed under the `name` variable.
- A bracket labeled "name project" is placed under the `projectname` variable.

```
1
2  class Student {
3      constructor(name)
4      {
5          this.studentname = name;
6      }
7      getname()
8      {
9          return this.studentname;
10     }
11  }
12
13  class IT5007Student extends Student {
14      constructor(name, project)
15      {
16          super(name);
17          this.projectname = project;
18      }
19      getprojectname()
20      {
21          return this.projectname;
22      }
23  }
```

CodePen – Example 2

- Example2: <https://codepen.io/pkarthik88/pen/LYmpKQo>
- Write code to extend IT5007Student class to create a new class called IT5007StudentUnofficial, where you have an additional field called WeChat_ID.



NUS | Computing

National University
of Singapore

MVC: View (Display)

Realizing views in React

- We don't write HTML code for view.
- React code is written in JS or JSX (JavascriptXML) files.
- React components should *render?* JS code which in-turn *renders?* HTML code.
- **React package** responsible for generating HTML code is the ReactDOM.
- The **react function** responsible is ReactDOM.render().

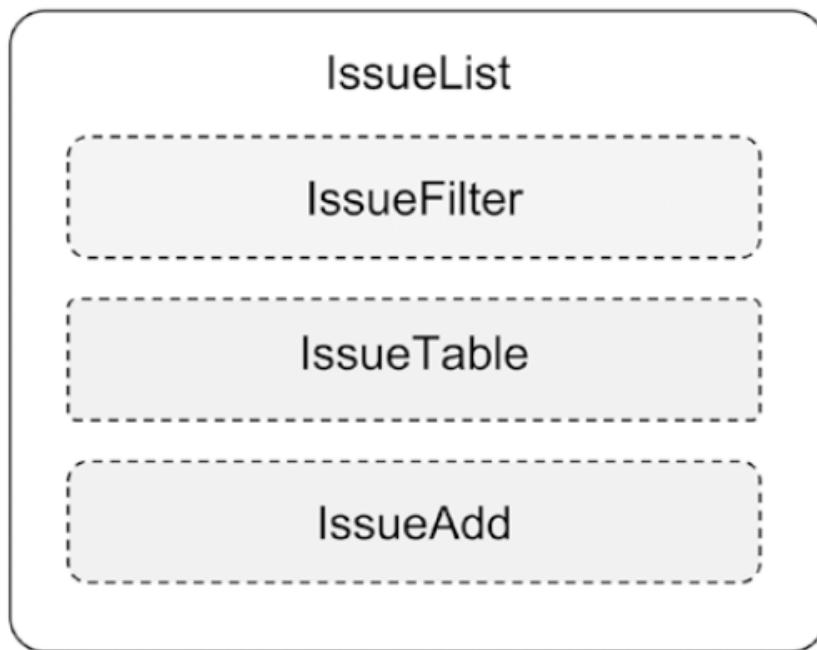
The screenshot shows a code editor interface with two tabs: 'HTML' and 'JS (Babel)'. The 'HTML' tab contains the following code:

```
<div id="content"></div>
```

The 'JS (Babel)' tab contains the following code:

```
const element = (<h1>hello</h1>);  
ReactDOM.render(element,  
document.getElementById('content'));
```

Realizing views in React



React Class Components

- **Built-in** class components. (e.g., `<h1>`, `<div>`)
- **Custom class** components built from many built-in classes. (e.g., `<IssueList>`, `<HelloWorld>`)
- Custom react classes always extend `React.Component`. ✓ ↗
- React.Component gives essential variables and functions that we will use later (Props, Lifecycle Methods)!
- How to componentize?
 - Each component has a single idea.
 - How to know if your component has a single idea?
 - If you are passing too many data items to a class.

```
class IssueFilter extends React.Component {  
  render() {  
    return (  
      <div>This is a placeholder for the issue filter.</div>  
    );  
  }  
}  
  
class IssueTable extends React.Component {  
  render() {  
    return (  
      <div>This is a placeholder for a table of issues.</div>  
    );  
  }  
}
```

Codepen Example 3

- <https://codepen.io/pkarthik88/pen/oNdjrEr>
- Change <React.Fragment> to <div> and see if there is any change to the view.
- Name the components that only use build-in react components. Which built-in component does it use?



National University
of Singapore

NUS | Computing

MVC: Model (Data)

Model-View Integration

- Class Component has
 - render() function that handles view.
 - Constructor() that handles data.^(Model)
 - **Integration:** We can use JS/React data variables in the view code.

JS (Babel)

```
24
25 <div>
26   <IssueList systemname="Bug Tracker" />
27 </div>
28
29 <script>
30   class IssueList extends React.Component {
31     constructor() {
32       super();
33       this.state = { systemname: "Bug Tracker" };
34     }
35     render() {
36       return (
37         <React.Fragment>
38           <h1>{this.state.systemname}</h1>
39           <IssueFilter />
40           <hr />
41           <IssueTable />
42           <hr />
43           <IssueAdd />
44           <hr />
45         </React.Fragment>
46     );
47   }
48 }
49 </script>
```

JS script 3

JS code

IssueList

Codepen Example 4

- <https://codepen.io/pkarthik88/pen/OJZyeem>
- Create one more class variable called ‘footer’ and use it to add a copyright statement to bottom of the View.

Building Data models in React

- **Data model** is different from database.
- DB will be introduced in Lecture 7. For now, let us use simple data-structures that resemble a database (e.g., an array of items).
- Data model refers to the **structure/shape of data**, **how they relate to a real-world system**, and **how they relate to one other**.
 - IssueList is an array of issues.
 - IssueList has a real-world application called bug tracker.
 - IssueList is related to EmployeeList the same way as real-world issues/bugs are related to employees.

3 Types of Data in React (Any web/mobile platform)

- **Domain data:** The dataset or static data that your application works with.
 - For example, Netflix works with a list of movies.
- **(optional) App State:** The current state of the App at this instant of time. As time goes on, the app state will change.
 - For example, Netflix maintains a list of seen movies, movies that are added as favorites for each user profile.
- **(optional) UI State:** The current state of the Netflix UI. As time goes on, the UI state will change.
 - For example, Has the user scrolled to the next page?

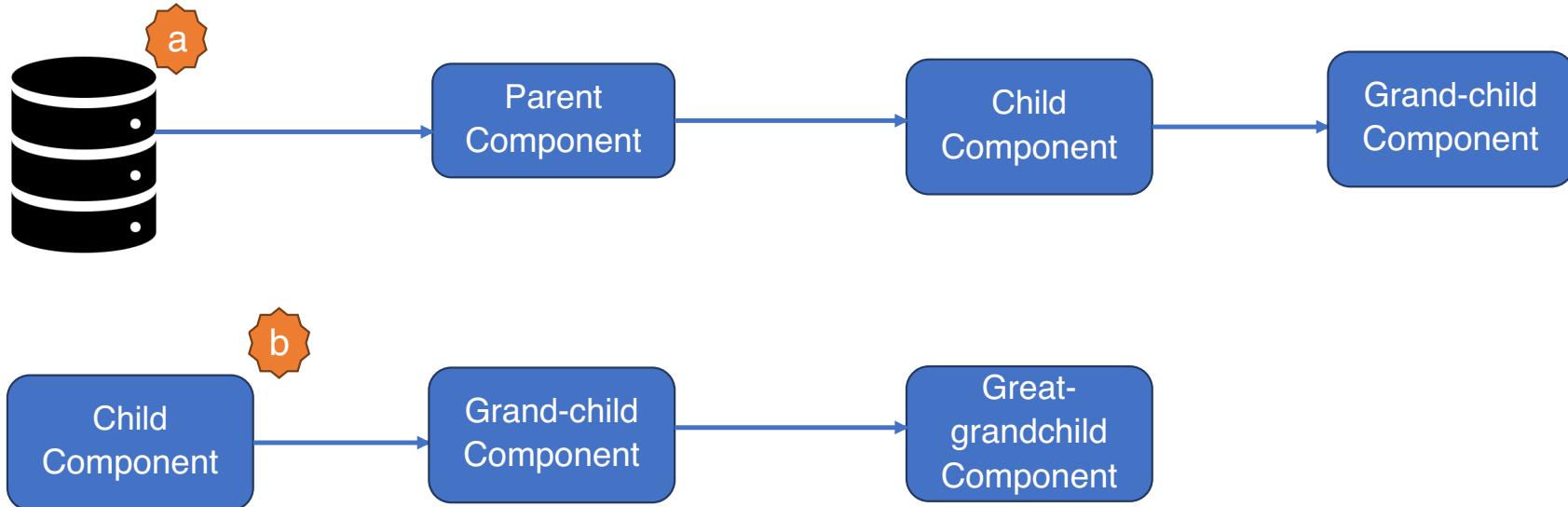


NUS | Computing

National University
of Singapore

MVC: Model (Data) for Domain Data

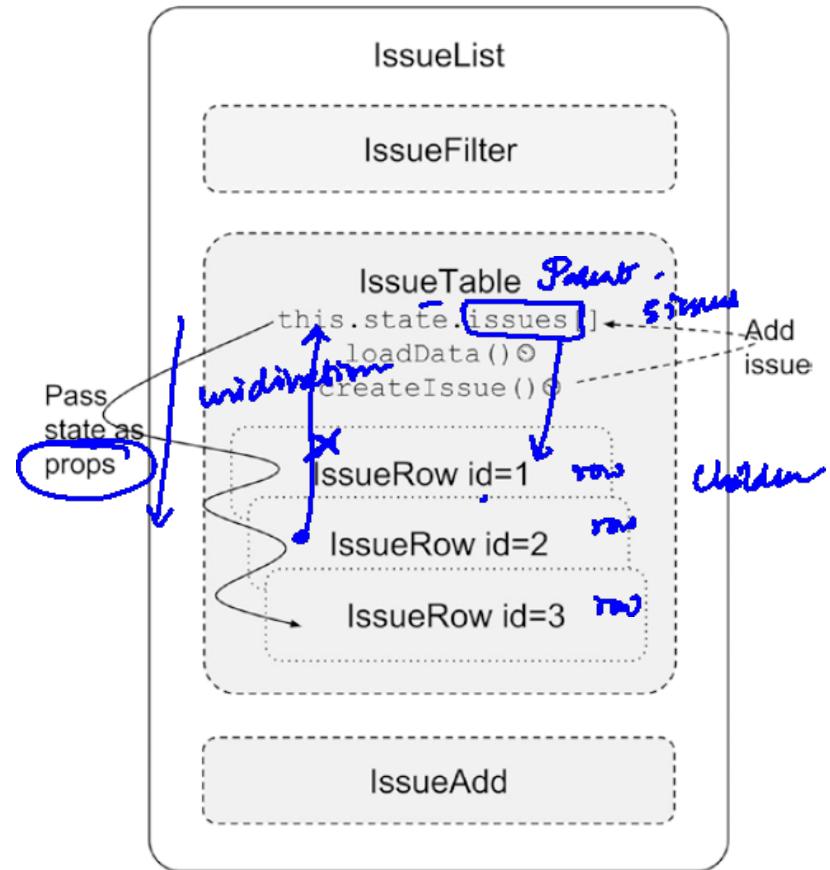
Data Flow rules



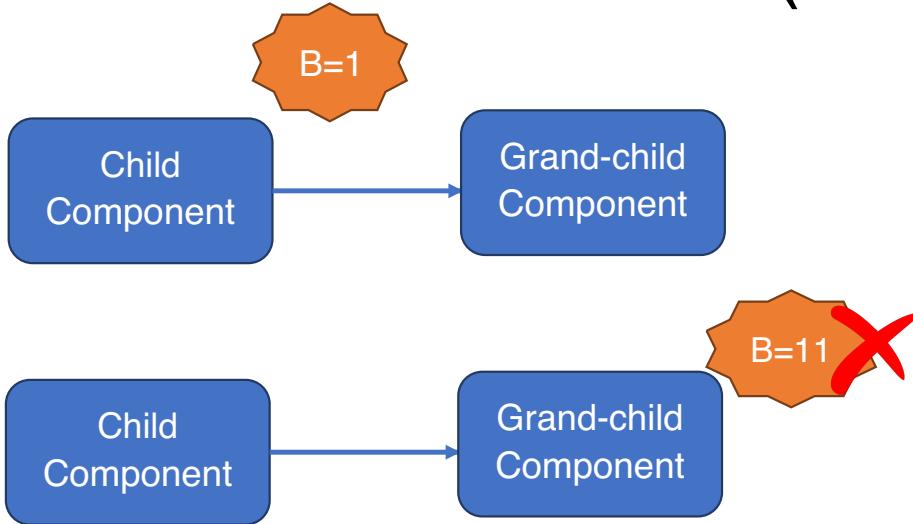
- Data flow is unidirectional
- Data cannot flow from child to parent.

Example: IssueTracker Application

- Data is passed down from IssueTable to IssueRow.
- Idea is to draw/render one row at a time.
- **New term:** props

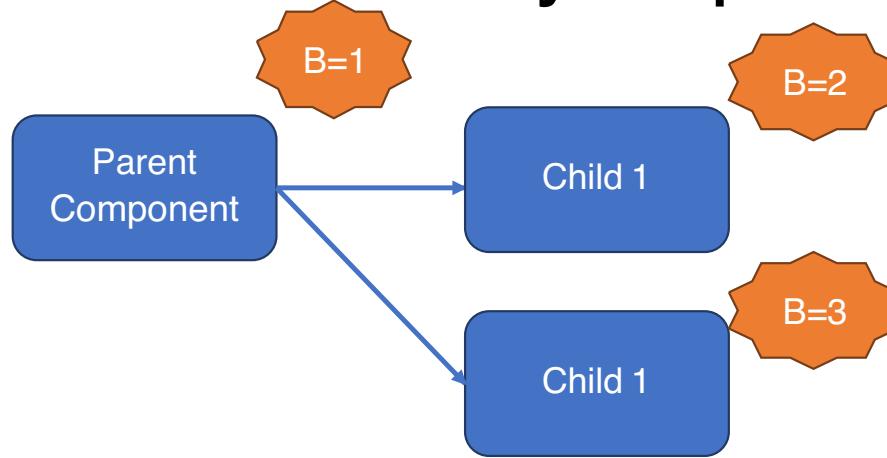


Data Modification Rules (Mutation)



- Data cannot be modified in child component.
- Why? Isn't this a huge restriction?

Why is immutability required?



Bottomline: This will lead to inconsistencies in data values.

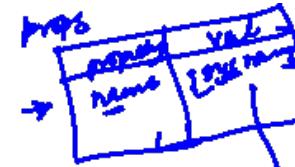
Passing Domain data: props

- The idea is to pass some data from a parent to a child.
- Props:** property attribute.
Parent → *children* → *value* → *Property*
- Parent class** passes the variable and its value to the child.
- Example: **IssueList** is the parent trying to pass `name={systemname}` to its children.

```
JS (Babel)
28
29 <div>
30 <h1>{systemname}</h1>
31 <hr />
32 <IssueFilter name={systemname}>
33 <hr />
34 <IssueTable name={systemname}>
35 <hr />
36 <IssueAdd name={systemname}>
37 <hr />
38 </React.Fragment>
39
40
41
42 );
43 }
44 }
45
46 const element = <IssueList />;
47
48 ReactDOM.render(element,
  document.getElementById('content'));
```

Passing Domain data: props

- Child class uses **props** to access it.
- At the child, e.g., IssueFilter
- Variable is accessed as: *this.props.name*



```
JS (Babel)
1 class IssueFilter extends React.Component {
2   render() {
3     const systemnameinfilter = this.props.name;
4     return (
5       <div>This is a placeholder for the issue filter in
6       {systemnameinfilter}.</div>
7     );
8   }
}
```

Codepen Example 5

- <https://codepen.io/pkarthik88/pen/PoePMZj>



NUS | Computing

National University
of Singapore

React: Function Components (Efficiency)

Function Components for Efficiency

- Industry standard for implementing components.

The diagram illustrates the conversion of a class-based component into a function-based component. On the left, the original code is shown in a Babel interface:

```
JS (Babel)
1 class IssueFilter extends React.Component {
2   render() {
3     const systemnameinfilter =
4       this.props.name;
5     return (
6       <div>This is a placeholder for the issue
7         filter in {systemnameinfilter}.</div>
8     );
9   }
}
```

Handwritten annotations on the left side highlight several parts of the code:

- A red star is drawn above the class definition.
- A blue box encloses the string "This is a placeholder for the issue".
- A blue circle highlights the word "filter".
- A blue bracket at the bottom is labeled "WTF?".

An arrow points from the right side of the first code block to the second code block. The second code block shows the converted function-based component:

```
JS (Babel)
1 function IssueFilter(props) {
2   {
3     const systemnameinfilter = props.name;
4     return (This is a placeholder for the issue
5       filter in <div>{systemnameinfilter}</div>);
6   }
7
8
9 }
```

Handwritten annotations on the right side highlight the conversion:

- A red star is drawn above the function definition.
- A blue arrow points from the word "props" in the first argument of the function to the word "props" in the third line of the returned JSX.
- A blue arrow points from the word "filter" in the third line of the returned JSX to the word "filter" in the original code.
- A blue arrow points from the word "original" to the first line of the converted code.
- A blue arrow points from the word "explicating" to the second line of the converted code.
- A blue arrow points from the word "code" to the bottom of the converted code block.

CodePen Example 6

- <https://codepen.io/pkarthik88/pen/YzjMajX>
- Convert the footer class component into a function component.



NUS | Computing
National University
of Singapore

The Setup

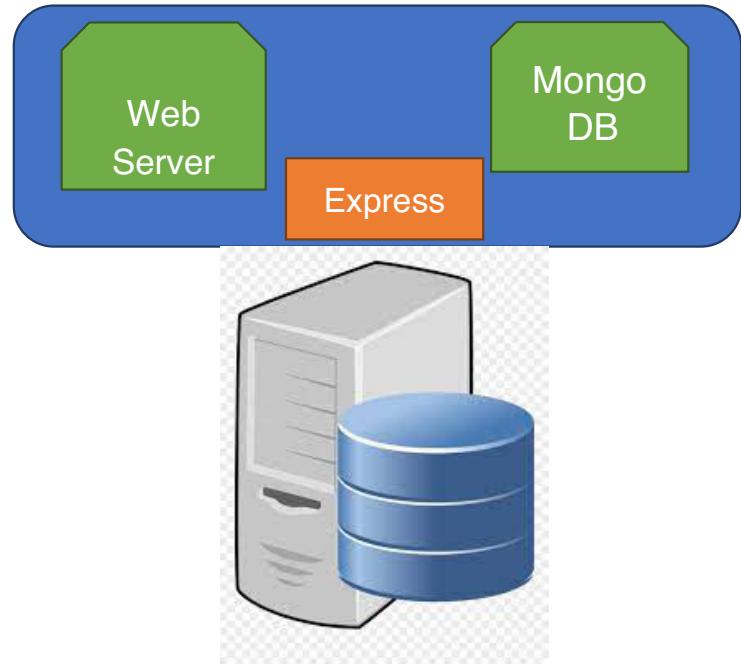
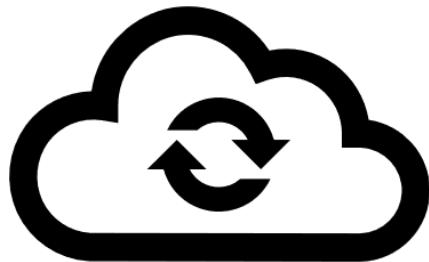
Developing without Servers

- Internet, Server, Client: Are we rich enough to have a server for ourselves?
- Instead, Try to **emulate** the server environment within your laptop.
- Our laptop serves as both server and client
- Virtualization – VM vs. Containers. We will be using **containers**.
- **Server:** Docker Container (runs as an application)
- **Client:** Web browser (runs as an application)
- Why not just code it like Tutorial 2? Have all files in one folder and double click on HTML file!

Real World Applications

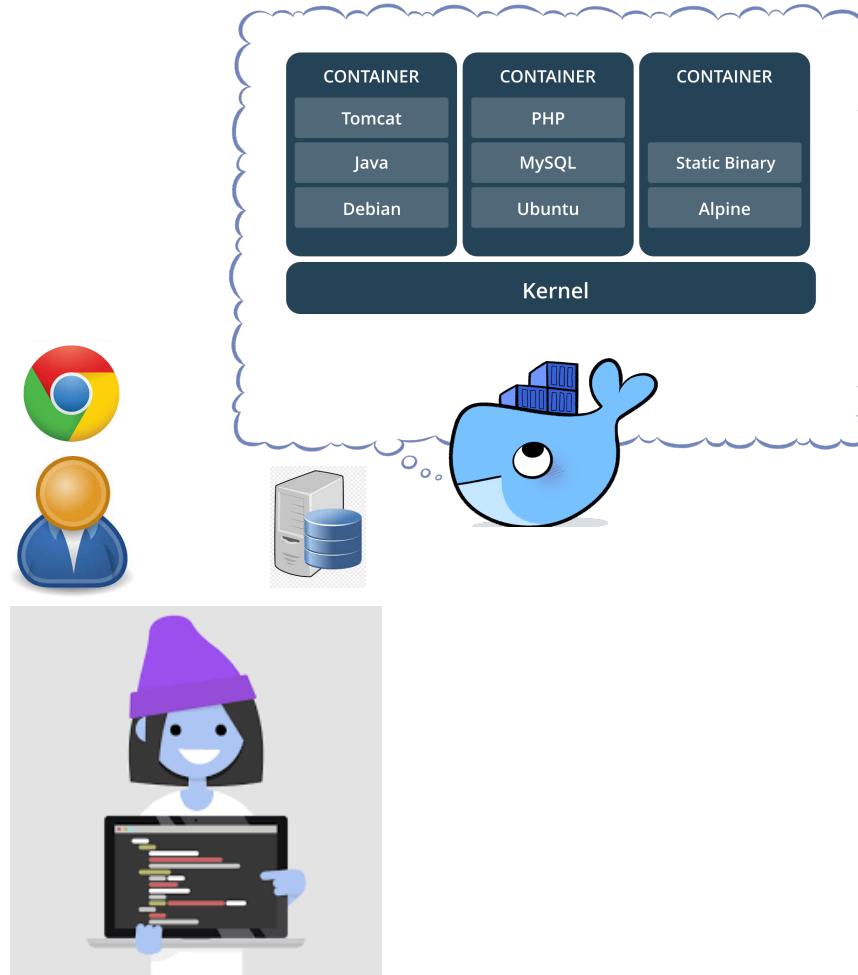


Client

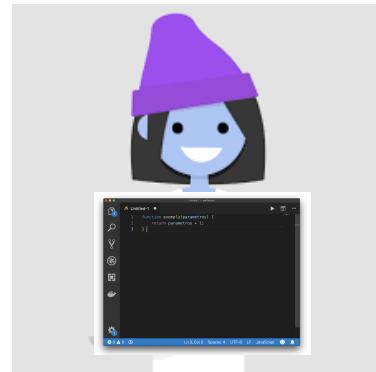
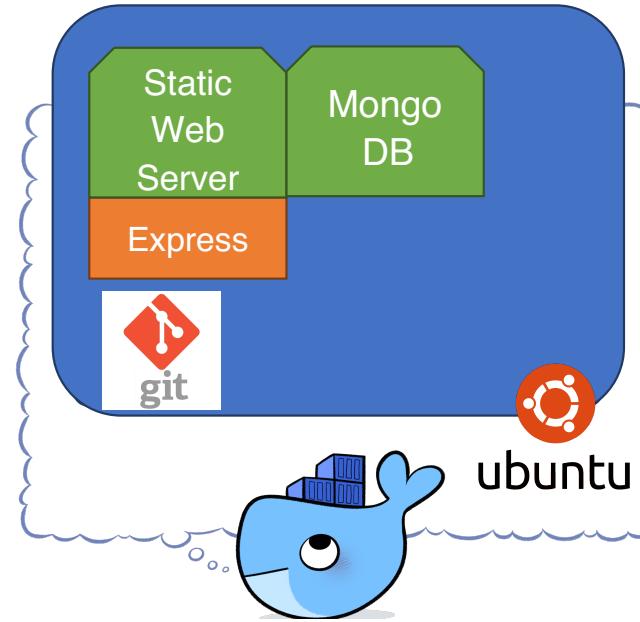


Server

Developer's Setup



Developer's Setup





NUS | Computing

National University
of Singapore

Setup Walkthrough Session

Follow instructions in

<https://github.com/pkarthik88/IT5007-Lecture5.git>