LWC

Created By Dharmesh Kumar
9/2/2022

Visualforce VS Aura

VisualForce	Aura Component
Server-Side: - User requests a page . - The server executes the page's underlying code and sends the resulting HTML to the browser . - The browser displays the HTML	Client side: - The user requests an application or a component . - The application or component bundle is returned to the client - The browser loads the bundle - The JavaScript application generates the UI when the user interacts with the page, the JavaScript application modifies the user interface as needed

Evolution of the Web Stack

TRAILHEAD ACADEMY

2014

Lightning Component Framework and Aura programming model launched

Web Standards

Renaissance

2019

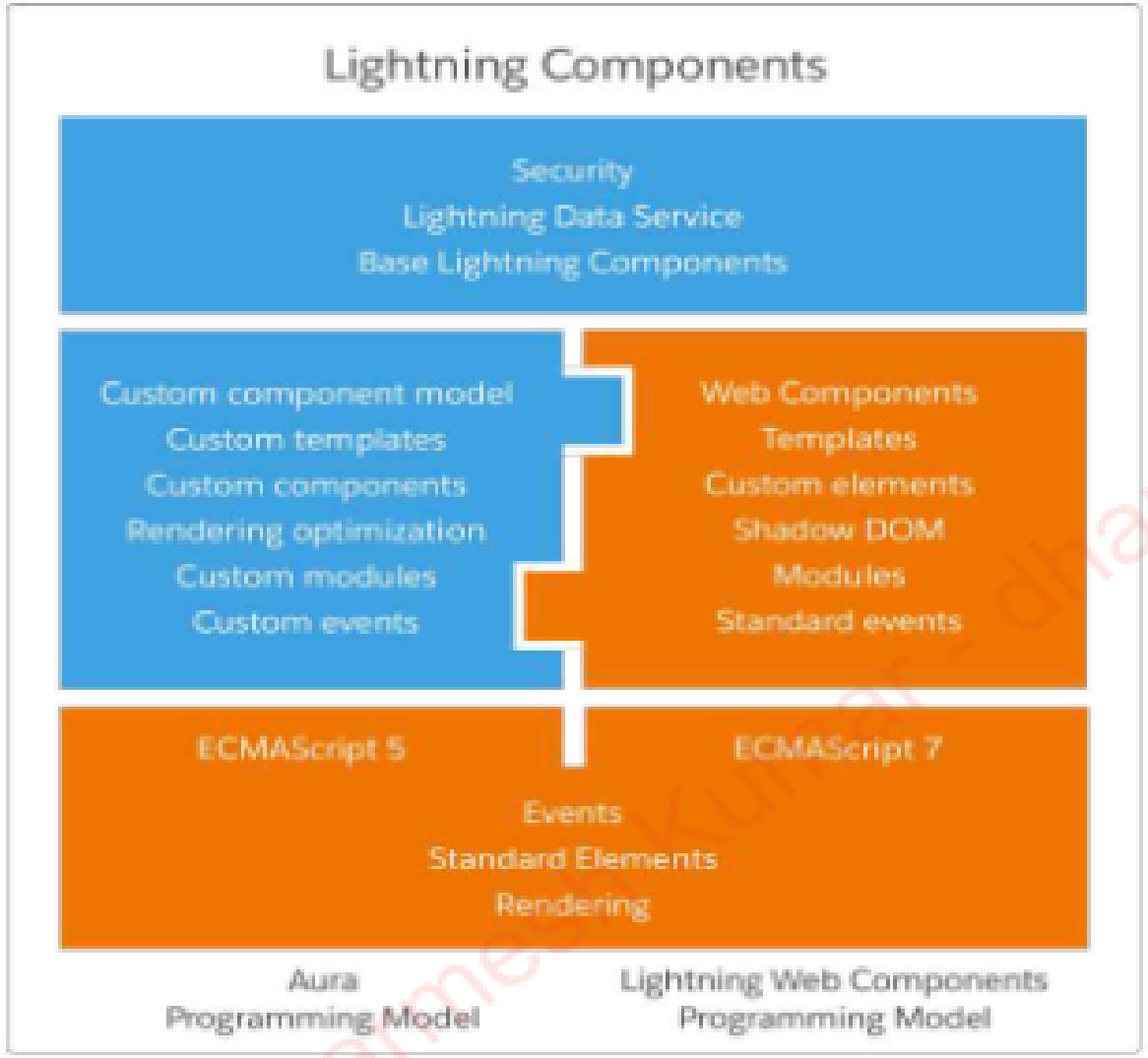
Lightning Web Components launched

What are Lightning Web Components?

TRAILHEAD ACADEMY

- A new programming model for building Lightning components that leverages the **web standards** breakthroughs of the last five years
- Provides a layer of specialized Salesforce services on top of the core stack
- Can coexist and **interoperate with the Aura** model
- Unparalleled **performance**

Aura and Lightning Web Components Interoperability



Aura and LWC:

- Can **coexist** and **interoperate**
- Share the same **high level services**
- Can **coexist** on the same page
- Share the same **Base Lightning components**
- Share the same **underlying services**
- Aura can **include** LWC but LWC cannot include Aura

Why would you choose LWC?

While the Aura system is expected to execute an exclusive part model, restrictive language expansions, and restrictive modules, LWC utilises web stack highlights carried out locally by programs, implying that LWC applications are considerably more performant.

Easy to learn

LWC is essentially takes the structure through local web guidelines that is in the program. It implies that no additional deliberation layer like Aura Framework.

Better performance

In view of the no additional reflection layer, LWC is probably going to deliver quicker than aura components since execution is imperative to deliverability.

Faster loading sites

Like lightning, LWC is quicker in stacking the created parts than Aura Components and is lightweight system which is based on web norms.

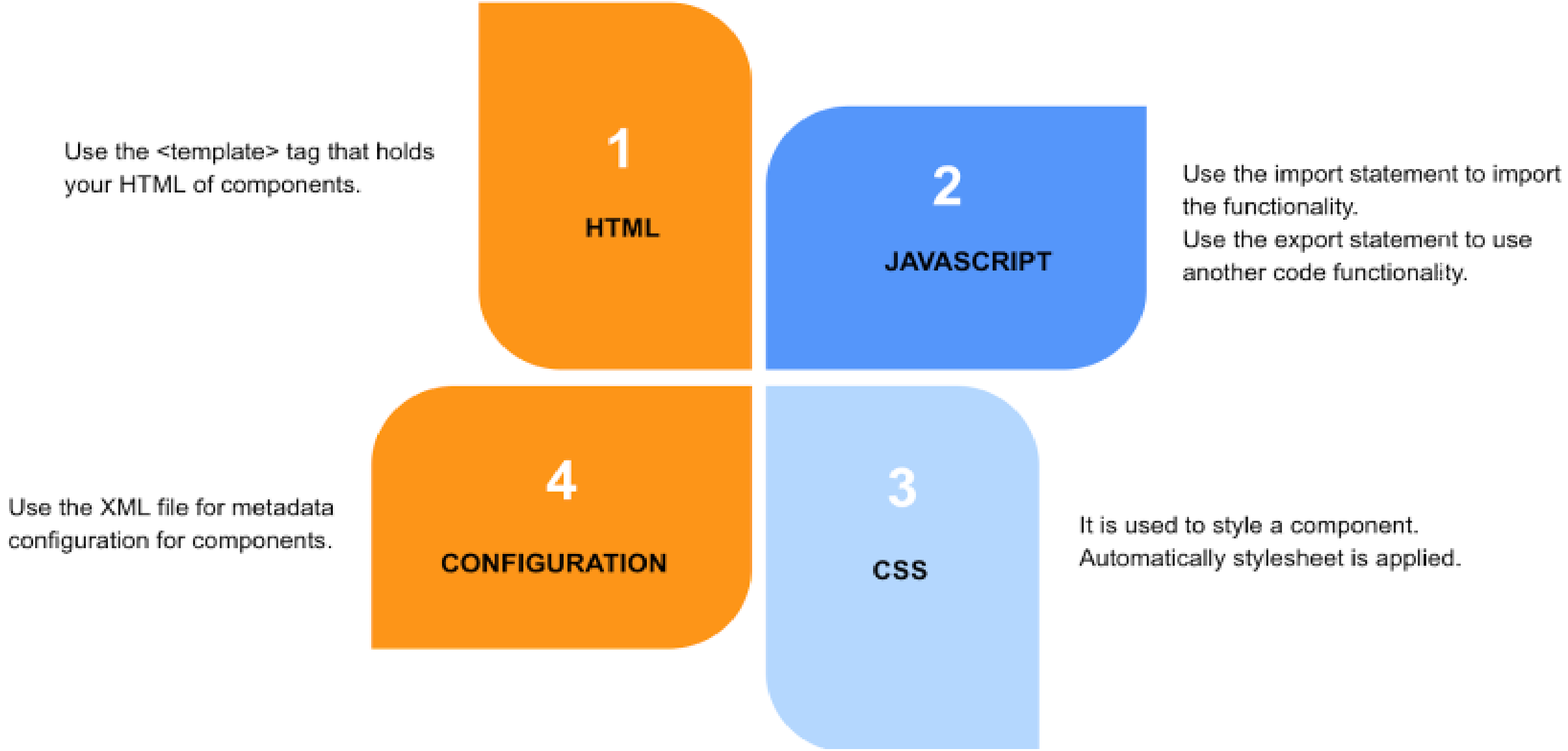
More standards, less proprietary

Understanding that there's a need for change and LWC has implicit program security highlights from Web Components Standards, so the use of out-of-the-container is more.

Better security

With LWC, CSS, Script and DOM Isolation are better and has more restricted occasion scope. With every one of these we have more consistency in planning Components.

The main contents in the LWC component are formed by HTML, javascript, CSS as similar to an aura component.The CSS is optional content in LWC and the XML configuration is included, which describes the metadata values. So, an LWC component will look like this:



Two ways to Create Component

1. **Using Terminal**

   *sfdx force:lightning:**component**:**create** --type **lwc** -n helloWorld*

2. **Using Command Palette**

   VsCode => View => Command Palette =>Type Create Lightning Web Component => Hit Enter =>Enter desired filename => hit enter => Again hit enter to choose default path

Naming Conventions for LWC

1. **camelCase** : Each word in the middle of the respective phrase begins with a capital letter.
2. **PascalCase**: It is same like Camel Case where first letter always is capitalized.
3. **kebab-case**: Respective phrase will be transferred to all lowercase with hyphen(-) separating words.

Case Name	camelCase	PascalCase	kebab-case
Example	helloWorld	HelloWorld	hello-world
Usage	component Name	Component class Name	Component reference and HTML attribute Name

Data Binding in LWC

Read 15mins

Synchronization of data between the business logic and the view of the application is known as data binding.In LWC, there are two ways of data binding - One-way and Two-way data binding.

In this task, you will learn

- One-Way Data Binding
- Two-Way Data Binding

One-way Data Binding

In one-way data binding, the information is flown only from the controller to the HTML template in one direction. Now, let's see an example of one-way data binding. These are the steps that we are performing for one-way binding:

Component Composition

Composition is Adding Component Within the body of another component

- Composition enables you to build complex components from simpler building-block components.

How to refer child components name in parent components

1. childComponent `<c-child-component></c-child-component>`
2. childComponentDemo `<c-child-component-demo></c-child-component-demo>`
3. sampleDemoLWC `<c-sample-demo-l-w-c></c-sample-demo-l-w-c>`

Replace capital letter with small letter and prefixed with hyphen

Try to avoid continuous capital letters in your component name.

Accessing Elements in LWC

To access elements rendered by a component, use the `template` property.

```
this.template.querySelector(selector);
this.template.querySelectorAll(selector);
element.template.querySelectorAll(selector);
```

Note** - Don't use ID selector with querySelector

lwc:dom="manual"

Add this directive to a native HTML element to attach an HTML element as a child.

CSS in LWC

1. **Inline Styling**
2. **External CSS**
3. Using Lightning Design System
4. SLDS Design Token
5. Shared CSS in LWC
6. Dynamic Styling
7. Third-party css library
8. Applying CSS across shadow DOM

Created By Dharmesh Kumar
9/2/2022

```css
/* element, classes, pseudo */
p{
    color:green;
    font-size: 30px;
    border: 1px solid red;;
}
.user{
    color:pink;
    font-size: 20px;
}
.user:hover{
    font-size: 50px;
    color:red;
}
```

Lightning Design System

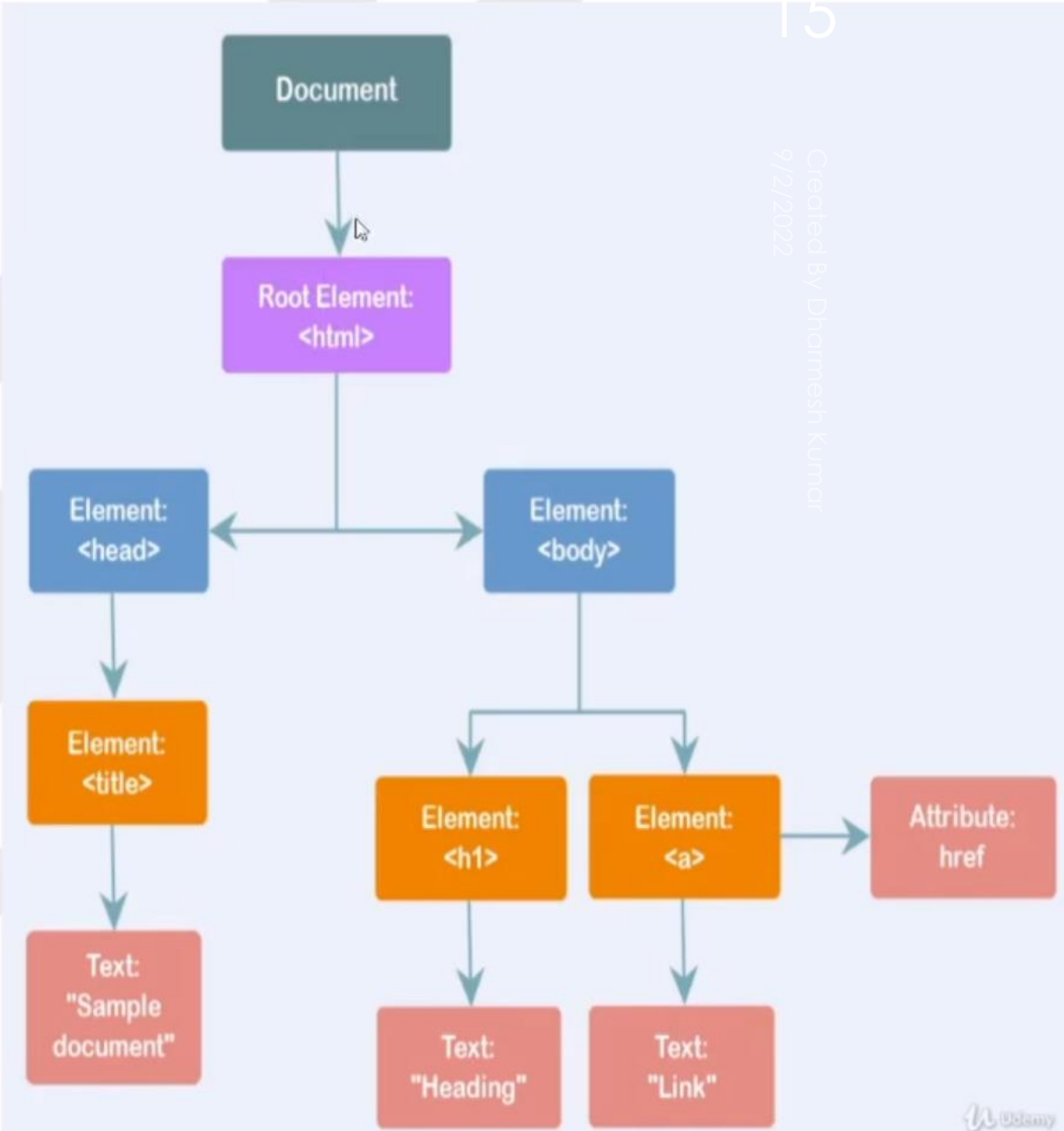
The Salesforce Lightning Design System includes the resources to create user interfaces consistent with the Salesforce Lightning principles, design language, and best practices.
https://www.lightningdesignsystem.com/

- Alignment
- Borders
- Box
- Description List
- Floats
- Grid
- Horizontal List
- Hyphenation
- Interactions
- Layout
- Line Clamp
- Margin
- Media Objects
- Name Value List
- Padding
- Position
- Print
- Scrollable
- Sizing
- Position
- Print
- Scrollable
- Sizing
- Text
- Themes
- Truncation
- Vertical List
- Visibility

What is DOM

Document Object Model **(DOM)** is a programming API for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.
Basically, The DOM is a tree structure representation of web page.

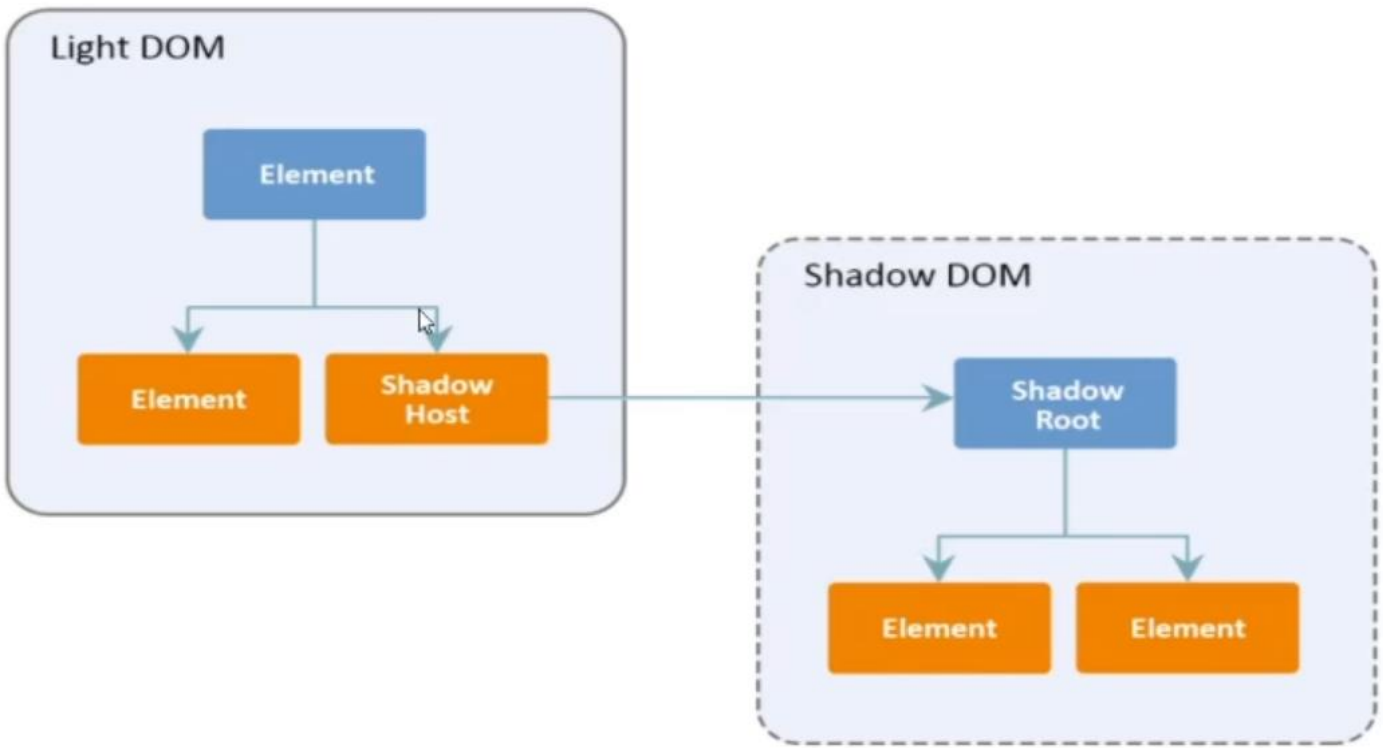
```
<html>
  <head>
    <title>Zero to Hero</title>
  </head>
  <body>
    <h1>Heading</h1>
    <a href="https://salesforcetroop.com">Link</a>
  </body>
</html>
```



What is Shadow DOM

Shadow DOM brings encapsulation concept to HTML which enables you to link a hidden separated DOM to an element.

Benefits of Shadow DOM- DOM queries, event propagation and CSS rules cannot go to the other side of the shadow boundary, thus creating encapsulation.



LWC lifecycle Hooks

A lifecycle hook is a callback method triggered at a specific phase of a component instance lifecycle.

Mounting Phase

- constructor()
- connectedCallback()
- render()
- renderedCallback()

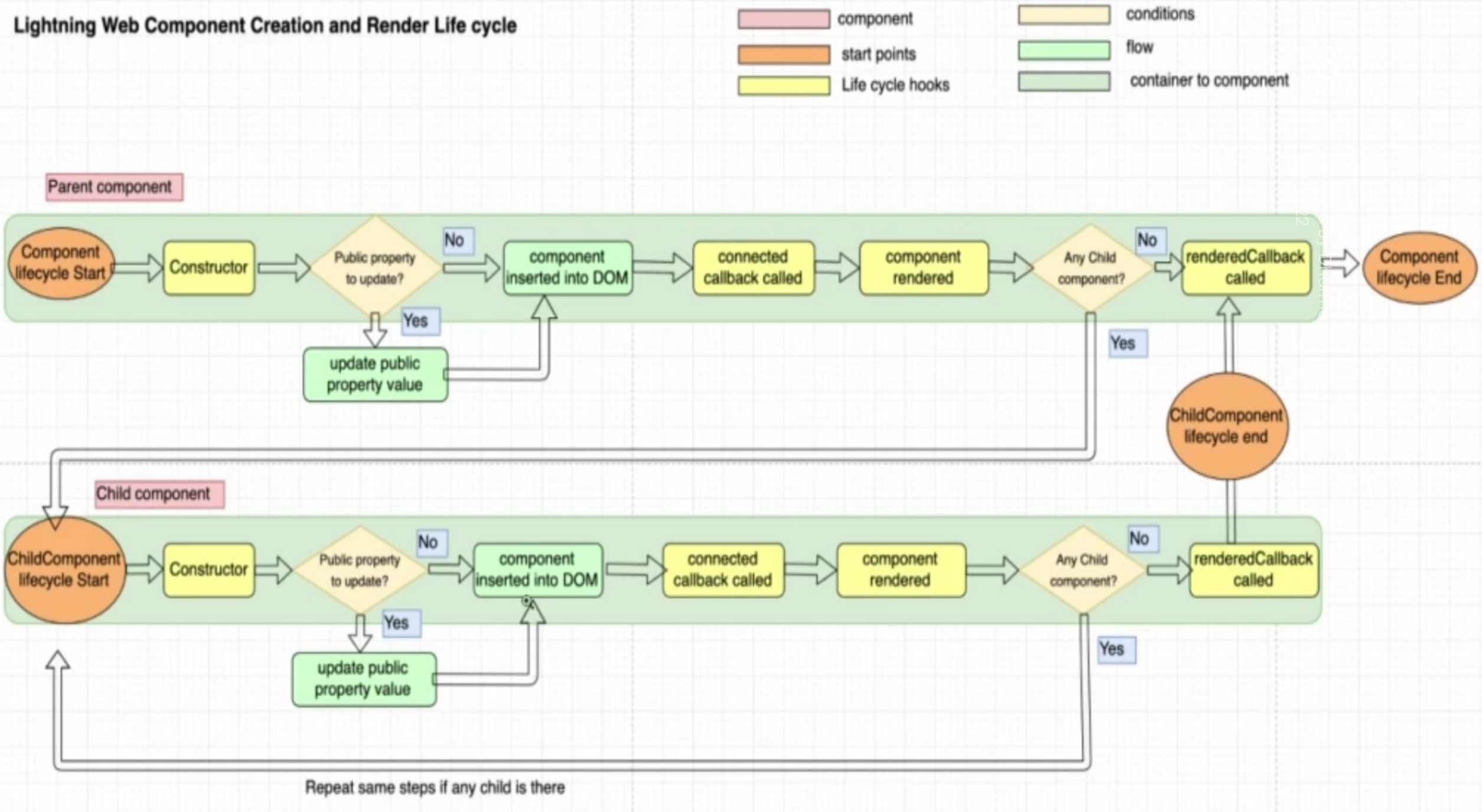
Unmounting Phase

- disconnectedCallback()

Error Phase

- errorCallback()

Lightning Web Component Creation and Render Life cycle



Constructor Method

Points to remember

- This hook is invoked, when a component instance is created.
- You have to call super() first inside this
- At this point, the component properties won't be ready yet.
- To access the host element, use this.template
- This method lifecycle flows from Parent to Child component.
- we can't access child elements in the component body because they don't exist yet.
- Don't add attributes to the host element in the constructor

renderedCallback Method

Points to remember

- Fires when a component rendering is done.
- It can fire more than once.
- This hook flows from child to parent.
- When a component re-renders, all the expressions used in the template are reevaluated.

Do not use renderedCallback()

1. to change the state or update property of a component
2. Don’t update a wire adapter configuration object property in renderedCallback(), as it can result in an infinite loop.

disconnectedCallback Method

Points to remember

- Fires when a component is removed from the DOM.
- It flows from parent to child
- This callback method is specific to Lightning Web Components, it isn't from the HTML custom elements specification.

errorCallback(err,stack) Method

Points to remember

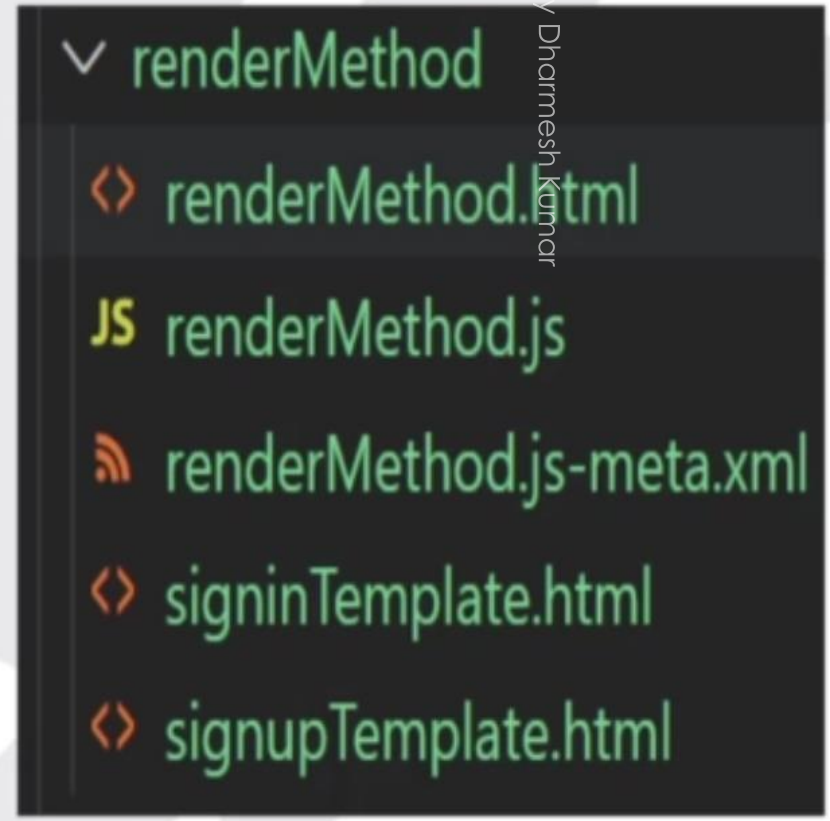
- This method called when a descendant component throws an error in one of its callback.
- The error argument is a JavaScript native error object, and the stack argument is a string.
- This callback method is specific to Lightning Web Components, it isn't from the HTML custom elements specification

render Method

Render is a method that tells the component which template to load based on some conditions. It always return the template reference

Points to remember

- The **render()** method is not technically a lifecycle hook. It is a protected method on the ***LightningElement*** class.
- Call this method to update the UI. It may be called before or after connectedCallback()
- It's rare to call render() in a component. The main use case is to conditionally render a template.

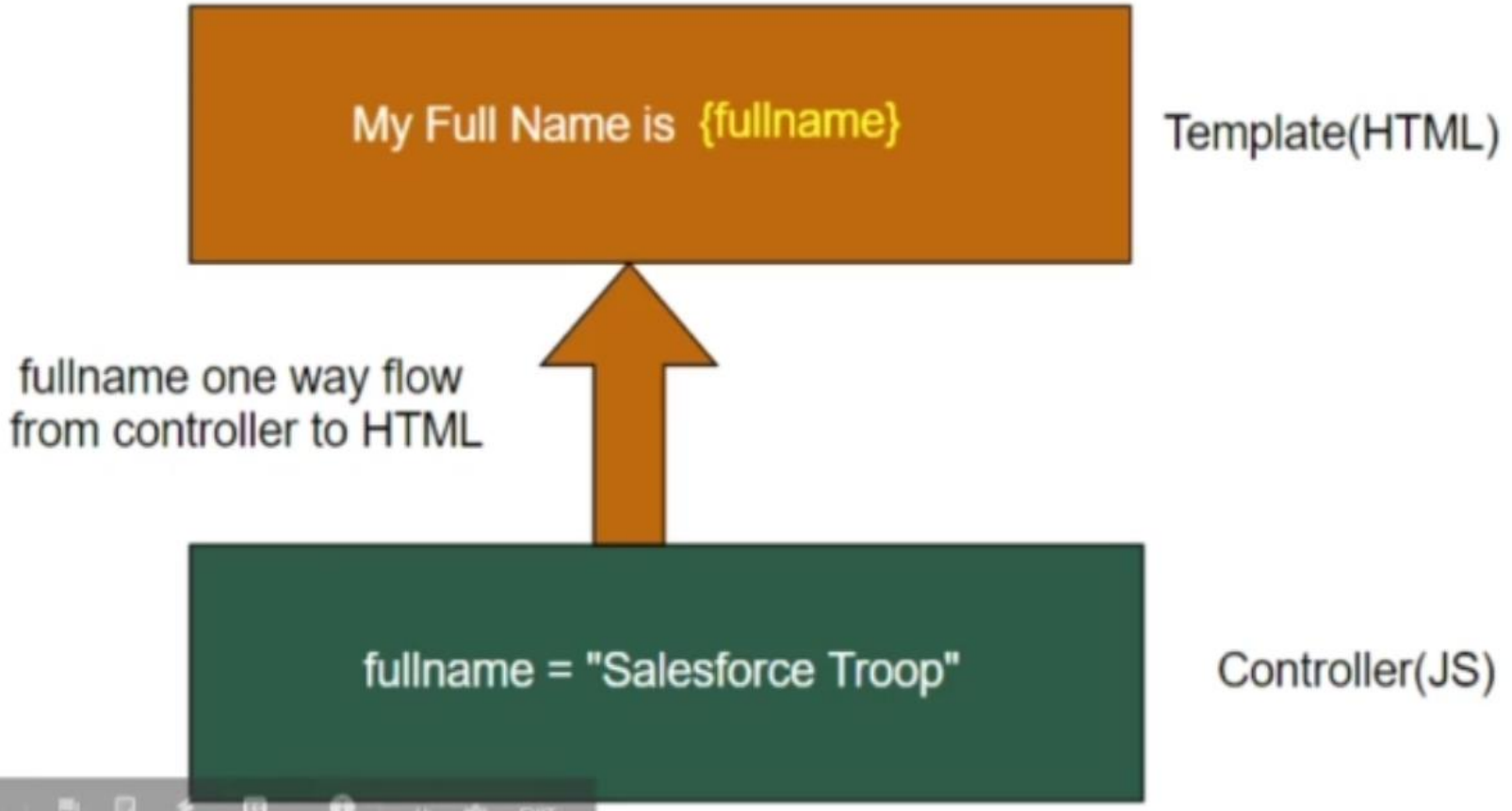


When to prefer multiple template over if:true/if:false

- **if:true/if:false** is recommend whenever there is small template to hide and show
- Ideally it's always recommend to break down your component into smallest unit.
- Whenever we have a scenario in which we have same business logic but we want to render a component with more than one look and feel.
- Whenever we have two design in same component but not want to mix the HTML in one file.

Data Binding in a Template

Data binding in the Lightning web component is the synchronization between the controller and the template(HTML).

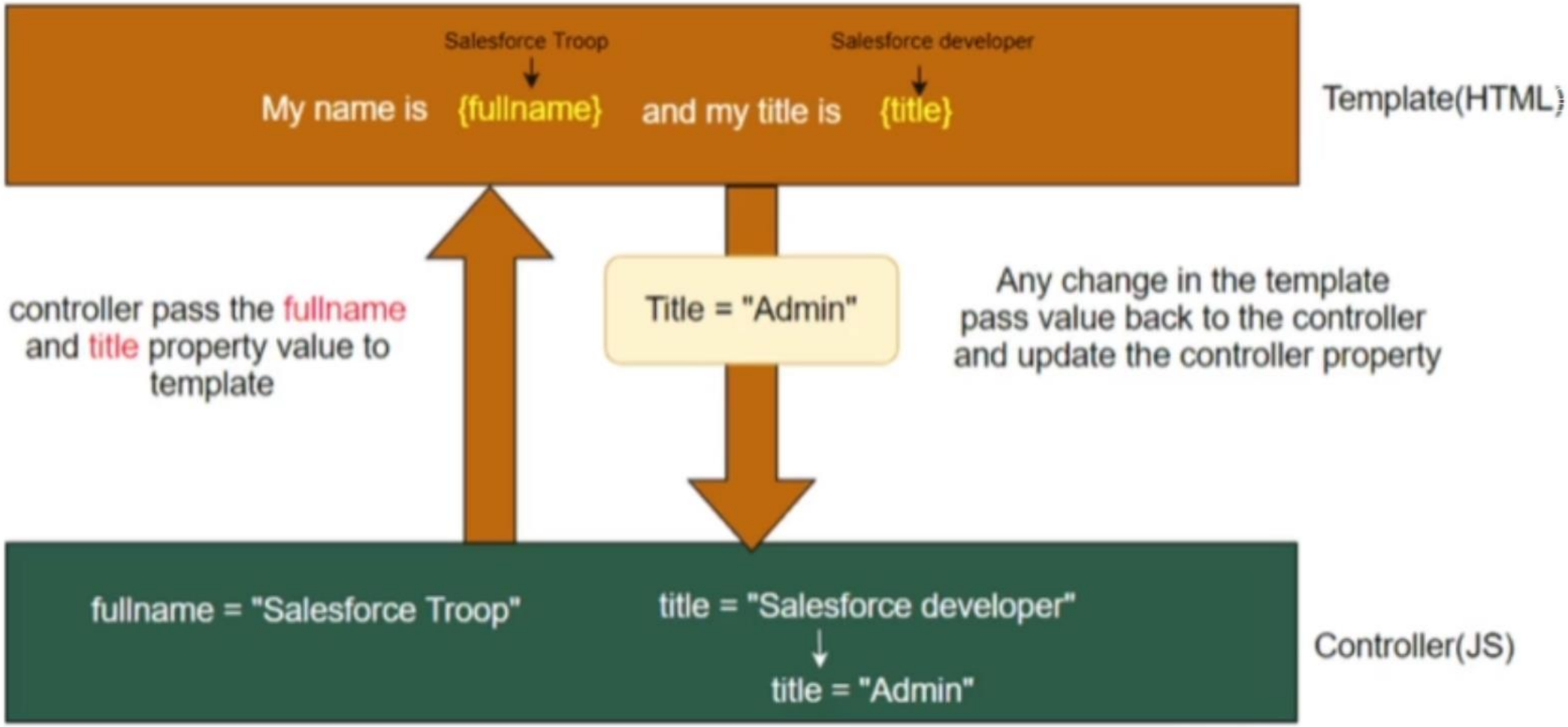


Two-way Data Binding

In two-way data binding, the information or the data only flew in both directions from the controller to the HTML template and vice-versa. Thus, it will allow users to establish communication bi-directionally. Now, let's see an example of one-way data binding.

These are the steps that we are performing for two-way binding:

Two way Data Binding in a Template



Decorators in LWC

Read 20mins

The Lightning web component model has three decorators that add usefulness to property or capacity. Decorators powerfully modify the use of a property or capacity & the capacity to make decorators is essential for ECMAScript. In lightning web components, we have three decorators: @api, @track and @wire.

In this task, you will learn

- @api decorator
- @track decorator
- @wire decorator

Decorators

TRAILHEAD ACADEMY

@api	Use to expose a public reactive property
@track	Observe object properties or array elements
@wire	Use to read Salesforce data or invoke Apex

@track Property

When a field contains an object or an array, there's a limit to the depth of changes that are tracked. To tell the framework to observe changes to the properties of an object or to the elements of an array, decorate the field with `@track`.

Normal Property vs @track property

Without using `@track`, the framework observes changes that assign a new value to a field/property. If the new value is not `===` to the previous value, the component re-renders.

@api decorators

We use @api decorator when we want to expose a public property. The syntax of decorating a property is as follows
@decoratorName propertyName='propertyValue'

The syntax of decorating a method is as follows

```
@decoratorName
getMethodName(){
return somevalue;
}
```

1. Public Property is reactive and if the worth of a responsive property is changed, the part is rerendered, so when a component is rerendered, every one of the articulations utilized in the format are reconsidered once again.
2. When we utilize the **@api decorator**, we should import it expressly from lwc as displayed below.
   import { LightningElement,api } from 'lwc';
3. Parent component can utilize the **Public Property**.
4. A component that defines a public property can set just its default esteem.
5. A parent component that utilizes the child part in its markup can set the child part's public property estimation.

@track decorator

There are some key points to remember for implementing @track decorator:

1. Private reactive property.

2. Reactive means if there is a change in javascript property the component will rerender and all the expressions utilized in the template are reevaluated one more time.

3. Private Property can be utilized only in the component where it is declared.

4. When we use the @track decorator so, we must import it explicitly from lwc as shown below.

@wire decorators

There are some key points to remember for implementing @wire decorator:

1. @wire is used to read Salesforce data.
2. It is reactive like when it provisions the data the components rerender.
3. It is used to call an apex method.

Component Composition

Composition is Adding Component Within the body of another component

- Composition enables you to build complex components from simpler building-block components.

How to refer child components name in parent components

1. childComponent `<c-child-component></c-child-component>`
2. childComponentDemo `<c-child-component-demo></c-child-component-demo>`
3. sampleDemoLWC `<c-sample-demo-l-w-c></c-sample-demo-l-w-c>`

Replace capital letter with small letter and prefixed with hyphen

Try to avoid continuous capital letters in your component name.

Event Communication in LWC

Read 1h 40mins

The events in lightning web components are used to communicate between the components. It is built on the "Data Object Model", a collection of API and the available objects in every browser.

1. Communication using methods(Parent to Child)

We use the @api decorator to make the children methods public so that the parent can call that method directly using the javascript API.

Child component

SYNTAX

```
@api
changeMessage(strString) {
        this.Message = strString.toUpperCase();
}
```

For accessing this child method in our parent component, we can do this by:

2. Custom event communication(Child to Parent)

In LWC, for the communication from the child to the parent component, we use the custom event. Thus, we can create as well as dispatch the custom event.

Components Communication Approaches

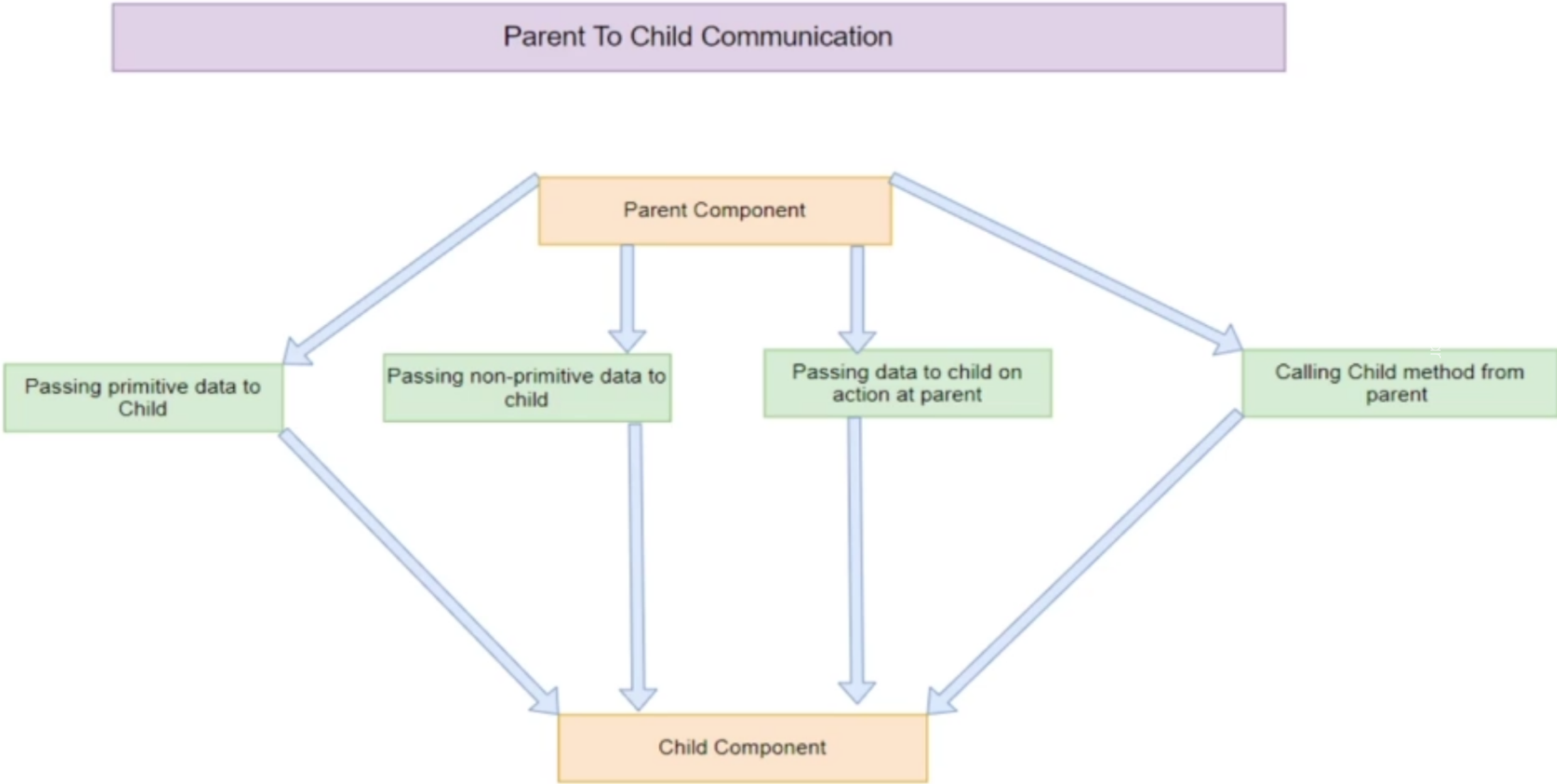
Parent To Child Communication

Child To Parent Communication

Sibling Component Communication Using PubSub

Communication Across VF pages, Aura and LWC Using Lightning Messaging Service

Parent to Child Communication



@api decorator

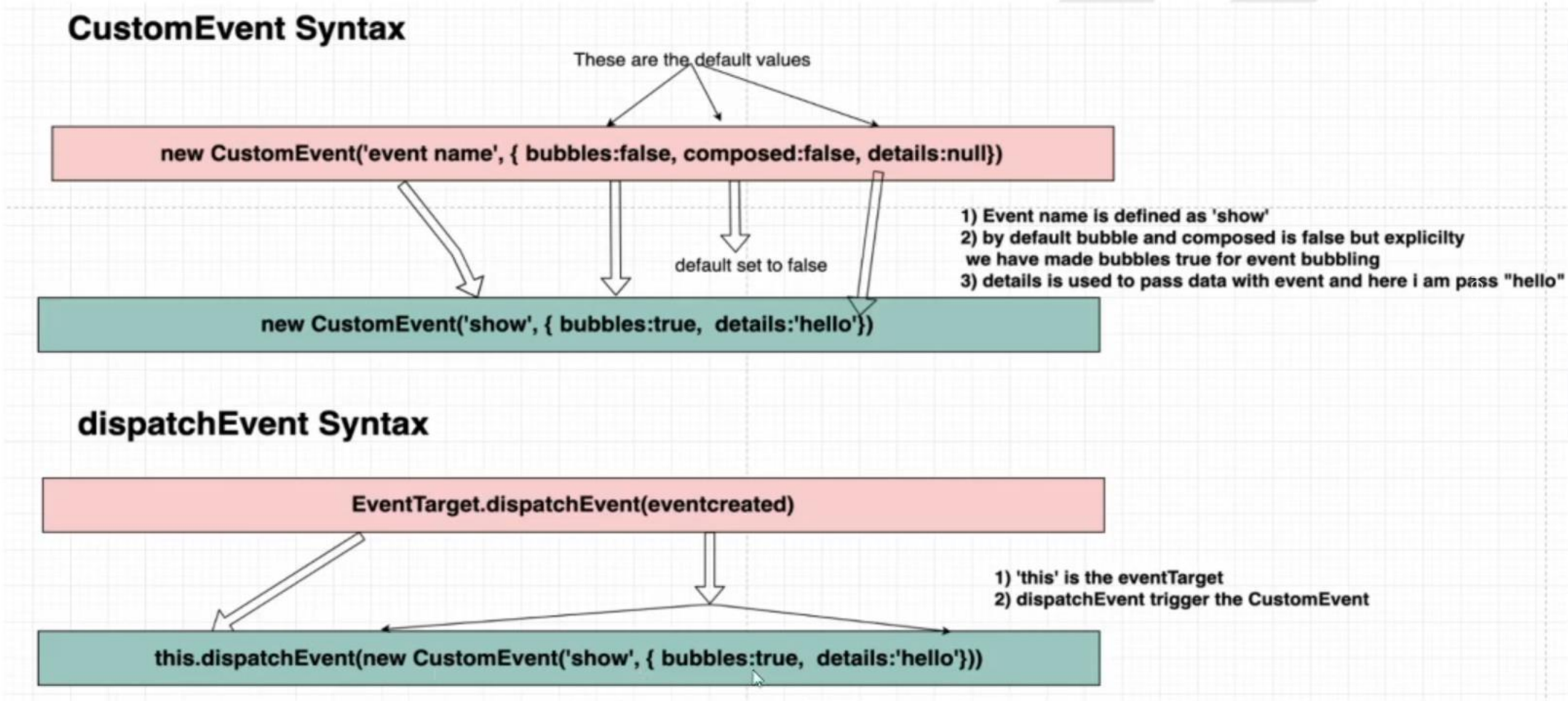
1. To make a field/property or method public, decorate it with @api decorator
2. When we want to expose the property we decorate the field with @api.
3. An owner component that uses the component in its HTML markup can access the component's public properties via HTML attributes.
4. Public properties are reactive in nature and if the value of the property changes the component's template re-renders.

```
import { LightningElement, api } from 'lwc';
export default class Child extends LightningElement {
    fullname

    @api username

}
```

Create and Dispatch Events

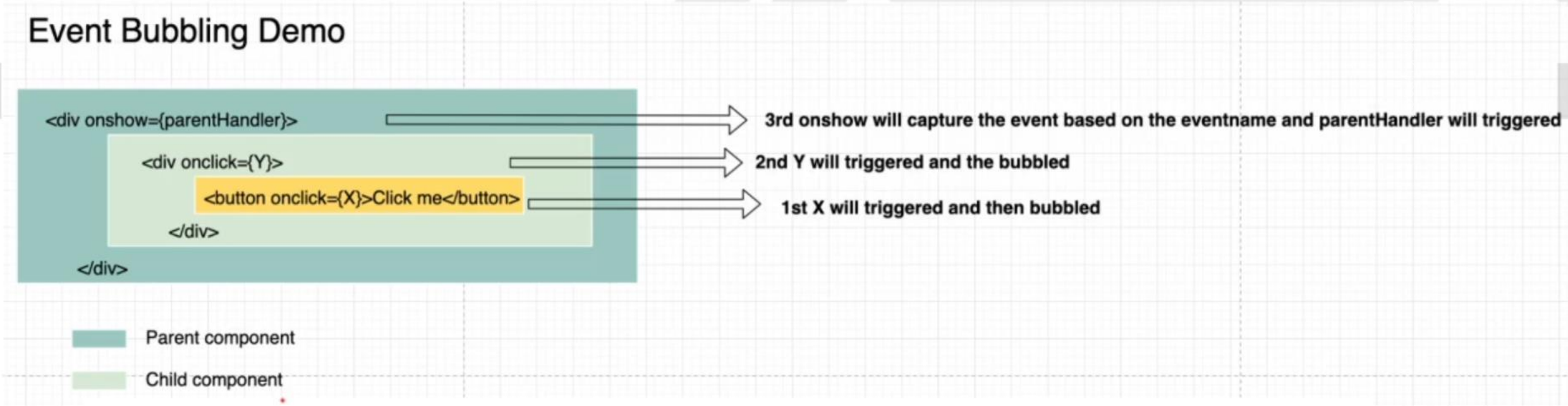


Event Name Naming Conventions- Only String, No uppercase letters,No spaces,Use underscores to separate words

Event Bubbling

Lightning web component has only two event propagation

- ***bubbles*** - A Boolean value indicating whether the event bubbles up through the DOM or not. Defaults to ***false***.
- **composed** - A Boolean value indicating whether the event can pass through the shadow boundary. Defaults to ***false***.

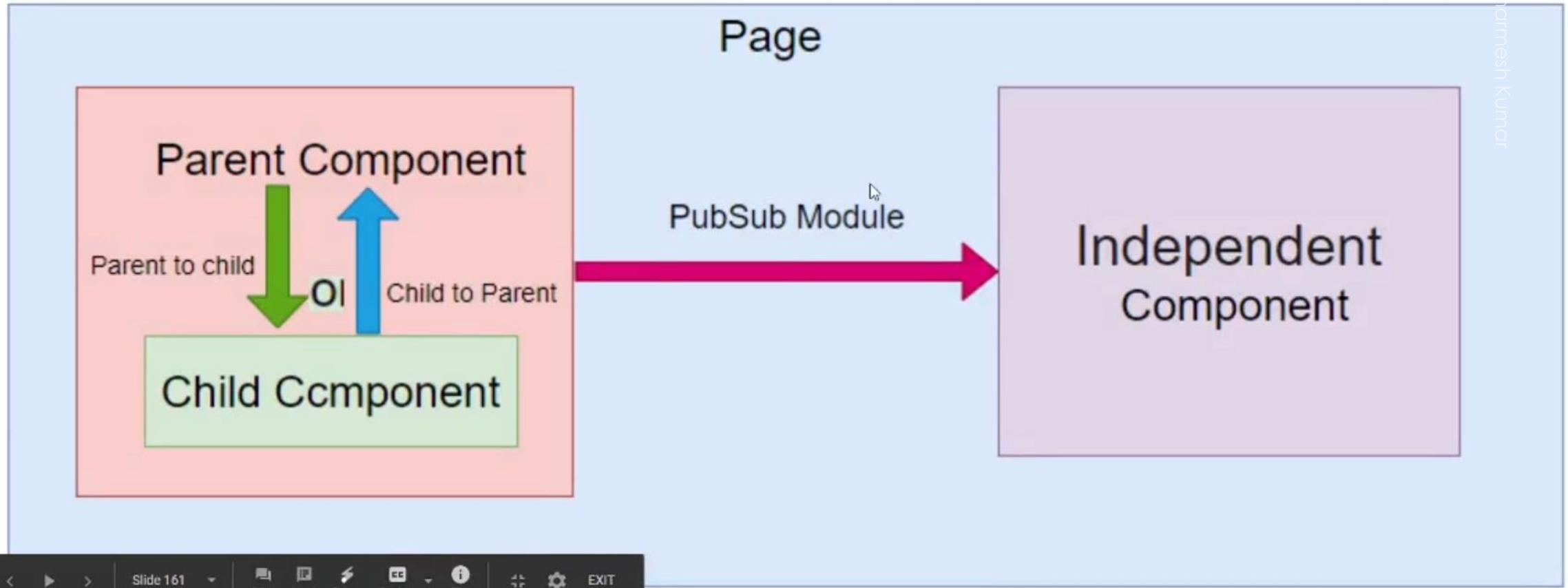



```
this.dispatchEvent(new CustomEvent('show', { bubbles:true,  details:'hello'}))
```

Component Communication Using PubSub

There are two ways to communicate between Independent Components
1) pubsub
2) Lightning Messaging Service

NOTE ** Use this approach, if Lightning Messaging Service not serve your purpose. It's an Old technique to communicate with the independent components in LWC



Setter Method

This method is use to modified the data coming from parent component.
If Object is passed as data to setter, to mutate the object we have to create a shallow copy.

Passing Markup into Slots

1. Slot is useful to pass HTML markup into the another component.
2. <slot></slot> markup is used to catch the HTML passed by parent component
3. You can't pass an Aura component into a slot

There are two types of Slots

Named Slots	When name attribute is defined in slot element ***<slot name="user"></slot>***
Unnamed Slots	When a slot without a name attribute ***<slot></slot>***

CSS Behaviour in Parent Child Component

1. Parent CSS can't reach into a child component.
2. A parent component CSS can style a host element(<c-css-child>).
3. A Child Component CSS can reach up and style its own host element
4. You can style to a element pass into the slot from the parent component only

What is Getter and When to use it

Not Allowed in Template

```
<div>users[0]</div>
<div>{num1 * num2}</div>
```

Javascript

```
users = ["John", "Mike", "Smith"]

num1 = 10
num2 = 20
```


Template

```
<div>{firstUser}</div>
<div>{sumOfNumbers}</div>
```

Javascript

```
//Example 1
users = ["John", "Mike", "Smith"]
get firstUser(){
    return this.users[0]
}

//Example 2
num1 = 10
num2 = 20
get sumOfNumbers(){
    return this.num1 * this.num2
}
```

Aura Coexistence

1. Lightning Web component and Aura component can work together.
2. Aura components can contain Lightning web components. However, the opposite doesn't apply. *Lightning web components can't contain Aura components.*

Lightning Data Service

1. Lightning Data Service is a centralized data caching framework and it is built on top of User Interface API

2. UI API gives you data and metadata in a single response and also respect CRUD access, field-level security settings, and sharing settings.

3. LDS displays only records and fields for which users have CRUD access and FLS visibility

4. LDS caches results on the Client

5. LDS, Invalidates cache entry when salesforce data and metadata changes

6. Optimizes server calls

Base Lightning Components

Base Lightning Components are built on Lightning Data Service. So, Lightning Data Service is used behind the scenes by base components and inherits its caching and synchronisation capabilities

There are three types of base lightning components built on LDS are

1) lightning-record-form
2) lightning-record-edit-form
3) lightning-record-view-form

When to Use these form?

- Create a metadata-driven UI or form-based UI similar to the record detail page in Salesforce.
- Display record values based on the field metadata.
- Hide or show localized field labels.
- Display the help text on a custom field.
- Perform client-side validation and enforce validation rules.

Base Lightning Components

- **lightning-record-edit-form**—Displays an editable form.
- **lightning-record-view-form**—Displays a form in view mode.
- **lightning-record-form**—Supports edit, view, and read-only modes.

FEATURE	LIGHTNING-RECORD-FORM	LIGHTNING-RECORD-VIEW-FORM	LIGHTNING-RECORD-EDIT-FORM
Create Records	✅		✅
Edit Records	✅		✅
View Records	✅	✅	
Read-Only Mode	✅	✅	
Layout Types	✅		
Multi Column Layout	✅		
Custom Layout for Fields		✅	✅
Custom Rendering of Record Data		✅	✅

With lightning-record-form, you can specify a layout and allow admins to configure form fields declaratively. You can also specify an ordered list of fields to programmatically define what's displayed. lightning-record-form allows you to view and edit records.

lightning-record-form

Use the lightning-record-form component to quickly create forms to add, view, or update a record.

The lightning-record-form component provides these helpful features:

- Switches between view and edit modes automatically when the user begins editing a field in a view form
- Provides Cancel and Save buttons automatically in edit forms
- Uses the object's default record layout with support for multiple columns
- Loads all fields in the object's compact or full layout, or only the fields you specify

lightning-record-form is less customizable. To customize the form layout or provide custom rendering of record data, use lightning-record-edit-form (add or update a record) and lightning-record-view-form (view a record).

Note** *Whenever possible, to boost performance, define fields instead of a layout.Specify a layout only when the administrator, not the component, needs to manage the provisioned fields.*

Lightning-record-form key attributes

object-api-name - This attribute is always required. The *lightning-record-form* component requires you to specify the object-api-name attribute to establish the relationship between a record and an object.

Note- Event and Task objects are not supported.

record-id - This attribute is required only when you're editing or viewing a record.

fields - pass record fields as an array of strings. The fields display in the order you list them.

layout-type - Use this attribute to specify a Full or Compact layout. Layouts are typically defined (created and modified) by administrators. .

modes - This form support three mode

- ***edit*** - Creates an editable form to add a record or update an existing one. . Edit mode is the default when record-id is not provided, and displays a form to create new records.
- ***view*** - Creates a form to display a record that the user can also edit. The record fields each have an edit button. View mode is the default when record-id is provided.
- ***readonly*** - Creates a form to display a record that the user can also edit

columns - Use this attribute to show multiple columns in the form

Create a Record Using lightning-record-form

Import references to Salesforce objects and fields from **@salesforce/schema**.

```
// Syntax
import objectName from '@salesforce/schema/objectReference';

import fieldName from '@salesforce/schema/object.fieldReference';
```

Syntax

```
<lightning-record-form
    object-api-name={objectName}
    fields={fieldList}
    onsuccess={successHandler}>
</lightning-record-form>
```

Display a record using lightning-record-form

We can display a record in two modes

1) **view** - In this mode, form using output fields with inline editing enabled.
2) **read-only** - In this mode, form loads with output fields only and you will not see edit icons or submit and cancel buttons.

Syntax

```
<lightning-record-form
    record-id="0010o00002niCSzAAM"
    object-api-name={objectName}
    fields={fieldList}></lightning-record-form>
```

Edit a record using lightning-record-form

Component using view mode by default. We can allow users to update fields directly in edit mode

Syntax

```
<lightning-record-form
      record-id="Your Record Id"
      object-api-name="Your Object API Name"
      columns="2"
      mode="edit"
      layout-type="Compact">
 </lightning-record-form>
```

setTimeout

The setTimeout() is a method of the window object. The setTimeout() sets a timer and executes a callback function after the timer expires.

setInterval

The setInterval() is a method of the window object. The setInterval() repeatedly calls a function with a fixed delay between each call.

lightning-record-view-form

- Use the lightning-record-view-form component to create a form that displays Salesforce record data for specified fields associated with that record. The fields are rendered with their labels and current values as read-only.
- You can customize the form layout or provide custom rendering of record data. If you don't require customizations, use lightning-record-form instead.
- To specify read-only fields, use lightning-output-field components inside lightning-record-view-form.

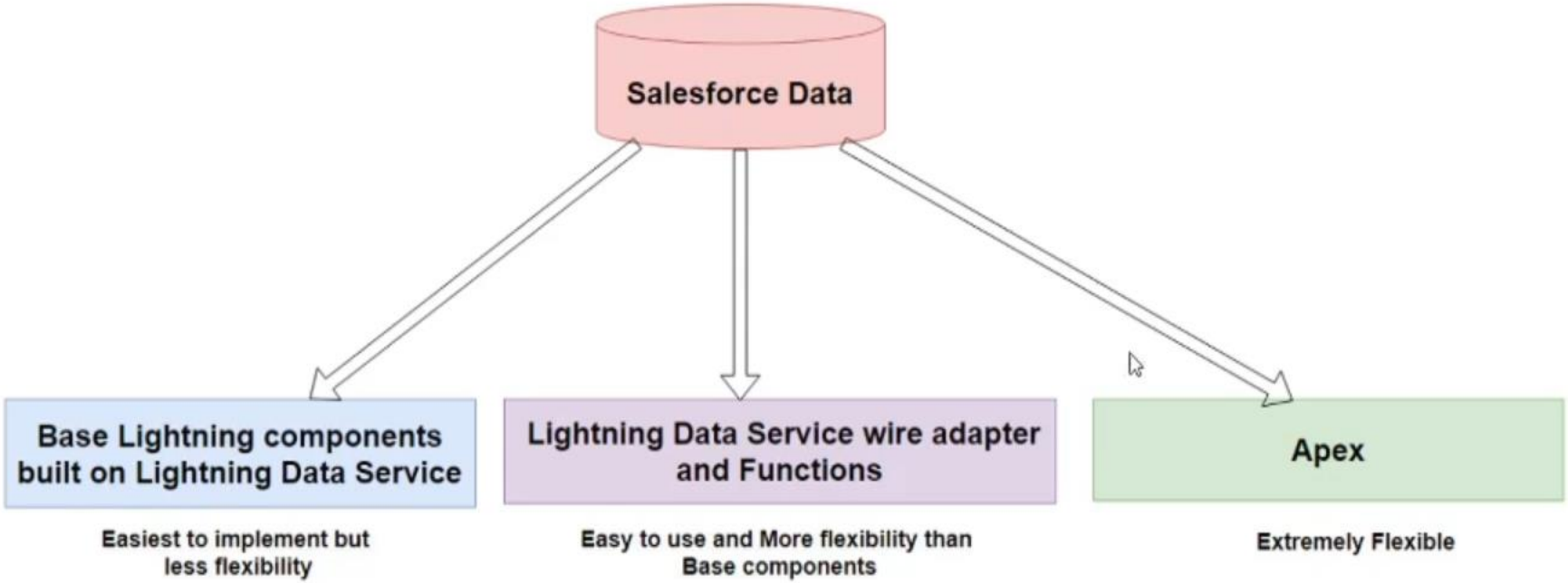
FEATURE	LIGHTNING-RECORD-FORM	LIGHTNING-RECORD-VIEW-FORM	LIGHTNING-RECORD-EDIT-FORM
Create Records	✅		✅
Edit Records	✅		✅
View Records	✅	✅	
Read-Only Mode	✅	✅	
Layout Types	✅		
Multi Column Layout	✅		
Custom Layout for Fields		✅	✅
Custom Rendering of Record Data		✅	✅

lightning-record-edit-form

- This component is used to create and edit the records.
- It provides custom layout of fields and custom rendering of record data

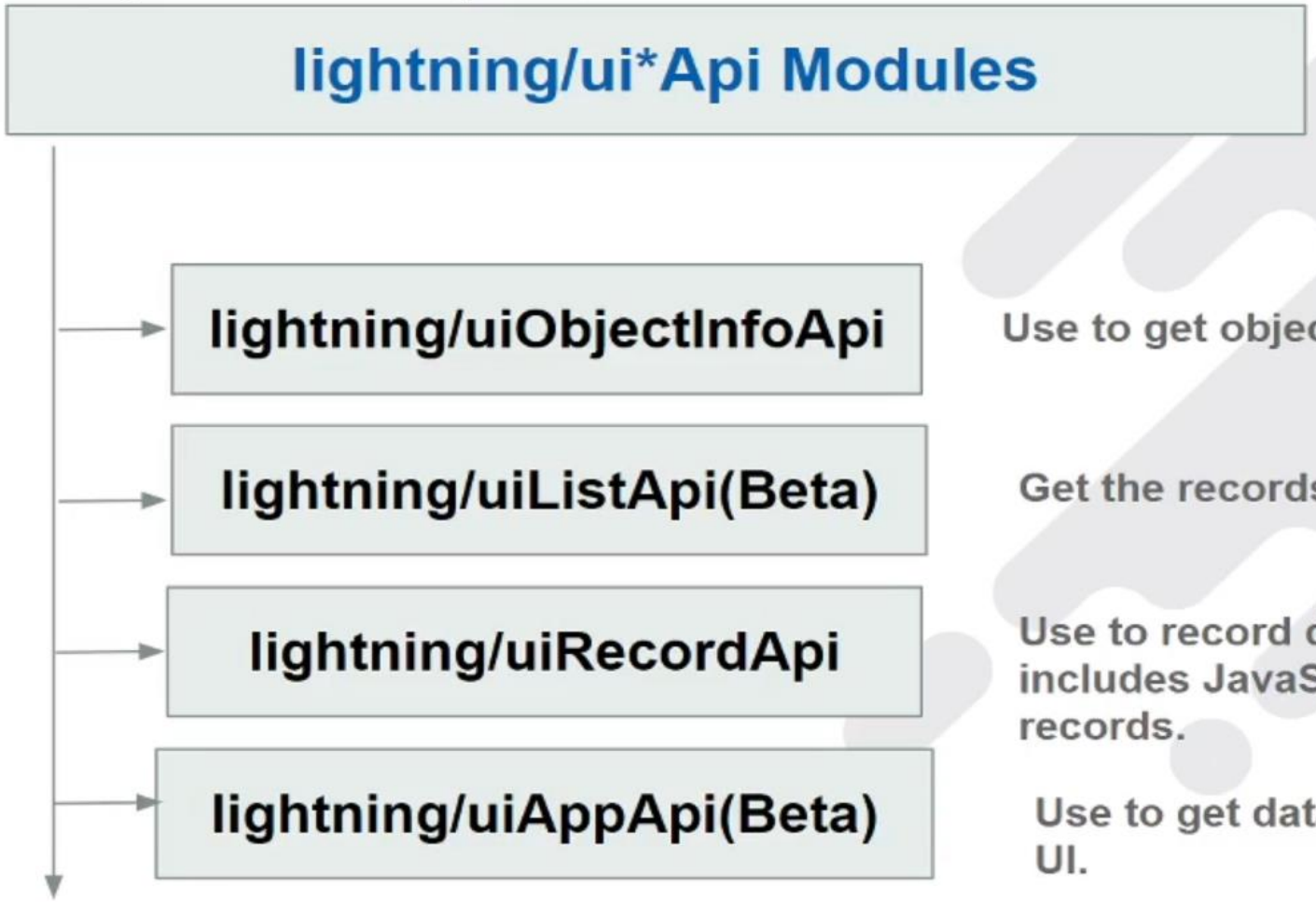
FEATURE	LIGHTNING-RECORD-FORM	LIGHTNING-RECORD-VIEW-FORM	LIGHTNING-RECORD-EDIT-FORM
Create Records	✅		✅
Edit Records	✅		✅
View Records	✅	✅	
Read-Only Mode	✅	✅	
Layout Types	✅		
Multi Column Layout	✅		
Custom Layout for Fields		✅	✅
Custom Rendering of Record Data		✅	✅

Lightning Data Service Wire Adapters and Functions



1. Use this to get the raw record data like picklist, recordInfo, objectInfo etc.
2. Want more customization to create a forms
3. You want to perform business logics

The wire adapters and JavaScript functions are available in lightning/ui*Api modules. which are built on User Interface API. User Interface API supports all custom objects and many standard objects

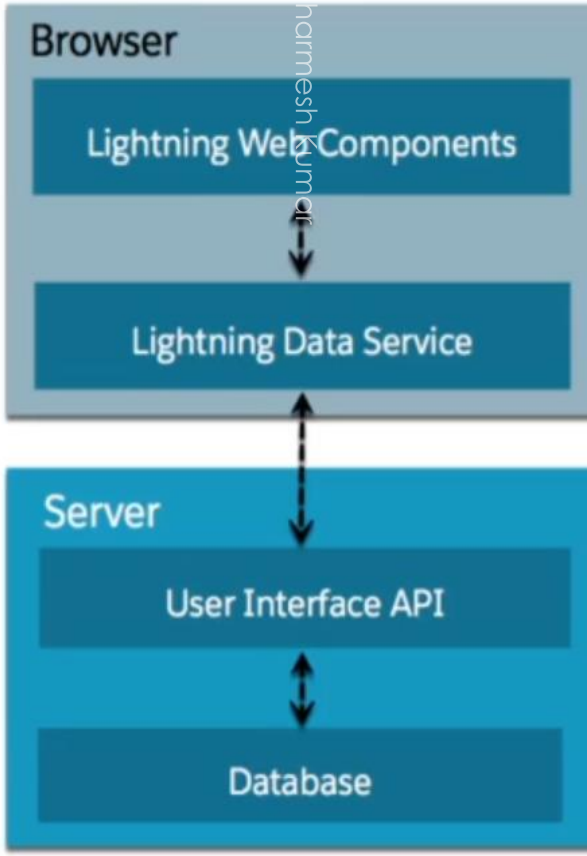


What is @Wire Service?

1. Wire service is built on Lightning Data Service
2. LWC component use @wire in their JavaScript class to read data from one of the wire adapters in the lightning/ui*Api namespace.
3. @wire is a reactive service
4. The wire adapter defines the data shape that the wire service provisions in an immutable stream

Syntax

```
import { adapterId } from 'adapterModule';
@wire(adapterId, adapterConfig)
propertyOrFunction;
```



Import property of Salesforce Standard Object

Syntax

```
import objectName from '@salesforce/schema/object';
```

Example

```
import ACCOUNT_OBJECT from '@salesforce/schema/Account';
```

How to Import Reference to Salesforce custom Object

Syntax

```
import objectName from '@salesforce/schema/object';
```

Example

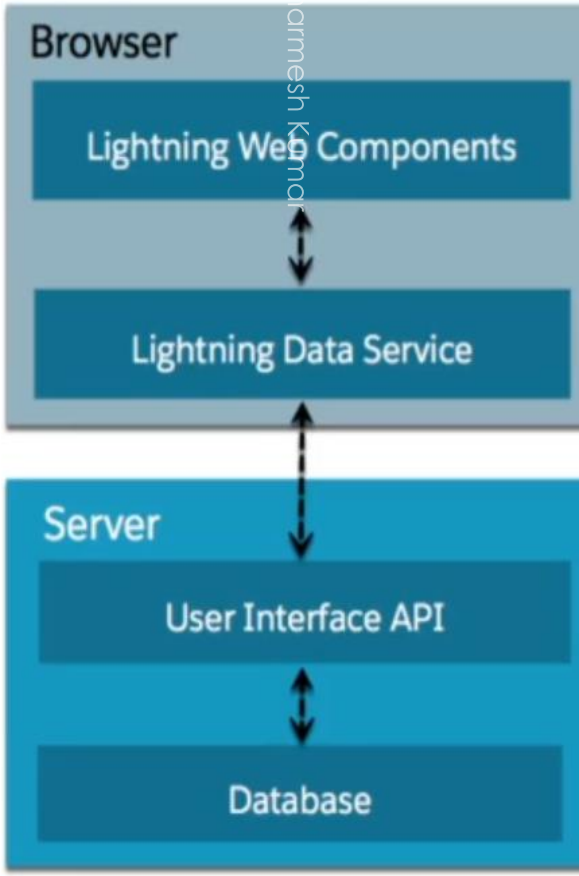
```
import PROPERTY_OBJECT from '@salesforce/schema/Property__c';
```

What is @Wire Service?

1. Wire service is built on Lightning Data Service
2. LWC component use @wire in their JavaScript class to read data from one of the wire adapters in the lightning/ui*Api namespace.
3. @wire is a reactive service
4. The wire adapter defines the data shape that the wire service provisions in an immutable stream

Syntax

```
import { adapterId } from 'adapterModule';
@wire(adapterId, adapterConfig)
propertyOrFunction;
```



How to Import References to Salesforce Fields

Syntax

```
import FIELD_NAME from '@salesforce/schema/object.field';
```

Example

```
import ACCOUNT_NAME from '@salesforce/schema/Account.Name';
import PROPERTY_NAME from '@salesforce/schema/Property__c.Name';
```

How to Import Reference to a field via a relationship

Syntax

```
import REF_FIELD_NAME from '@salesforce/schema/object.relationship.field;
```

Example

```
import ACCOUNT_OWNER from '@salesforce/schema/Account.Owner.Name';
```

getObjectInfo

Use this wire adapter to get metadata about a specific object. The response includes metadata describing the object's fields, child relationships, record type, and theme.

Syntax

```
import { LightningElement, wire } from 'lwc';
import { getObjectInfo } from 'lightning/uiObjectInfoApi';
import ACCOUNT_OBJECT from '@salesforce/schema/Account';

export default class Example extends LightningElement {
    @wire(getObjectInfo, { objectApiName: ACCOUNT_OBJECT })
    propertyOrFunction;
}
```

1. First import wire
2. Import adapter
3. Import dependencies of adapter i.e. fields or object reference
4. Call adapter using @wire
5. Pass adapterconfig

getObjectInfos

Use this wire adapter to get metadata for multiple objects. The response includes metadata describing the fields, child relationships, record type, and theme for each object.

Syntax

```
import { LightningElement, wire } from 'lwc';
import { getObjectInfos } from 'lightning/uiObjectInfoApi';
import ACCOUNT_OBJECT from '@salesforce/schema/Account';
import OPPORTUNITY_OBJECT from '@salesforce/schema/Opportunity';

export default class GetObjectInfosExample extends LightningElement {
    @wire(getObjectInfos, { objectApiNames: [ ACCOUNT_OBJECT, OPPORTUNITY_OBJECT ] })
    propertyOrFunction;
}
```

getPicklistValues

Use this wire adapter to get the picklist values for a specified field.

Syntax

```
import { LightningElement, wire } from 'lwc';
import { getPicklistValues } from 'lightning/uiObjectInfoApi';
import INDUSTRY_FIELD from '@salesforce/schema/Account.Industry';

export default class Example extends LightningElement {
    @wire(getPicklistValues, { recordTypeId: '012000000000000AAA', fieldApiName: INDUSTRY_FIELD })
    propertyOrFunction;
}
```

recordTypeId - The ID of the record type. Use the Object Info *defaultRecordTypeId* property, which is returned from *getObjectInfo*

fieldApiName - The API name of the picklist field

getPicklistValuesByRecordType

Use this wire adapter to get the values for every picklist of a specified record type

Syntax

```
import { LightningElement, wire } from 'lwc';
import { getPicklistValuesByRecordType } from 'lightning/uiObjectInfoApi';
import ACCOUNT_OBJECT from '@salesforce/schema/Account';

export default class Example extends LightningElement {
    @wire(getPicklistValuesByRecordType, { objectApiName: ACCOUNT_OBJECT, recordTypeId: '012000000000000AAA' })
    propertyOrFunction
}
```

recordTypeId - The ID of the record type. Use the Object Info *defaultRecordTypeId* property, which is returned from *getObjectInfo*

objectApiName - The API name of the Object

getRecord

Use this wire adapter to get the record's data

Syntax

```
import { LightningElement, wire } from 'lwc';
import { getRecord } from 'lightning/uiRecordApi';

@wire(getRecord, { recordId: string, fields: string|string[], optionalFields?: string|string[])
propertyOrFunction

@wire(getRecord, { recordId: string, layoutTypes: string|string[],
                   modes?: string|string[], optionalFields?: string|string[])
propertyOrFunction
```

recordId- The ID of the record type.
fields- A field or an array of fields to return
or
layoutType - it support two values Compact or Full(default)
Modes - used with layout. Values supported are Create, Edit and View(default)
optionalFields - a field name or an array of field names. If a field is accessible to the user, it includes in response otherwise it will not throw an error.

getFieldValue

Use this to gets a field's value from a record

Syntax

```
import { getFieldValue } from 'lightning/uiRecordApi';
getFieldValue(record: Record, field: string)
```

getFieldDisplayValue

Use this to gets a field's value in formatted and localized format from a record

Syntax

```
import { getFieldDisplayValue } from 'lightning/uiRecordApi';
getFieldDisplayValue(record, field)
```

getListUi(Beta)

Use this wire adapter to get the records and metadata for a list view

`lightning/uiListApi` *is for evaluation purposes only, not for production use.*

Syntax

```
import { LightningElement, wire } from 'lwc';
import { getListUi } from 'lightning/uiListApi';
import CONTACT_OBJECT from '@salesforce/schema/Contact';

export default class Example extends LightningElement {
    @wire(getListUi, { objectApiName: CONTACT_OBJECT, listViewApiName: 'AllContacts' })
    propertyOrFunction;
}
```

objectApiName- The API name of a supported object
listViewApiName - The API name of a list view such as AllAccounts, AllContacts etc.

sortBy, pageSize, pageToken

getNavItems(Beta)

Use this wire adapter to retrieve the items in the navigation menu

`lightning/uiAppsApi` *is for evaluation purposes only, not for production use.*

Syntax

```
import { LightningElement, wire } from 'lwc';
import { getNavItems } from 'lightning/uiAppsApi';


export default class Example extends LightningElement {
    @wire(getNavItems, {
        navItemNames: ['standard-Account'],
        pageSize: 30,
    })
    propertyOrFunction;
```

getRecordUi

Use this wire adapter to get layout information, metadata, and data to build UI for one or more records.

Syntax

Created 9/2/2022

```
import { LightningElement, wire } from 'lwc';
import { getRecordUi } from 'lightning/uiRecordApi';

export default class Example extends LightningElement {
  @wire(getRecordUi, { recordIds: string|string[], layoutTypes: string|string[],
                       modes: string|string[], optionalFields?: string|string[] })
  propertyOrFunction;
}
```

recordIds- IDs of records to load.
layoutType - it support two values Compact or Full
Modes - used with layout. Values supported are Create, Edit and View
optionalFields - a field name or an array of field names. If a field is accessible to the user, it includes in response otherwise it will not throw an error.

getRecordUi

Use this wire adapter to get layout information, metadata, and data to build UI for one or more records.

Syntax

```
import { LightningElement, wire } from 'lwc';
import { getRecordUi } from 'lightning/uiRecordApi';

export default class Example extends LightningElement {
  @wire(getRecordUi, { recordIds: string|string[], layoutTypes: string|string[],
                       modes: string|string[], optionalFields?: string|string[] })
  propertyOrFunction;
}
```

recordIds- IDs of records to load.
layoutType - it support two values Compact or Full
Modes - used with layout. Values supported are Create, Edit and View
optionalFields - a field name or an array of field names. If a field is accessible to the user, it includes in response otherwise it will not throw an error.

createRecord(recordInput)

CreateRecord is use to creates a record

Syntax

```
import { createRecord } from 'lightning/uiRecordApi';
createRecord(recordInput: Record): Promise<Record>
```

recordInput-A Record Input object is used to create the record. This object take object apiName and fields details as input

updateRecord(recordInput, clientOptions)

Use this wire adapter to updates a record. Provide the recordId of the record to update in recordInput.

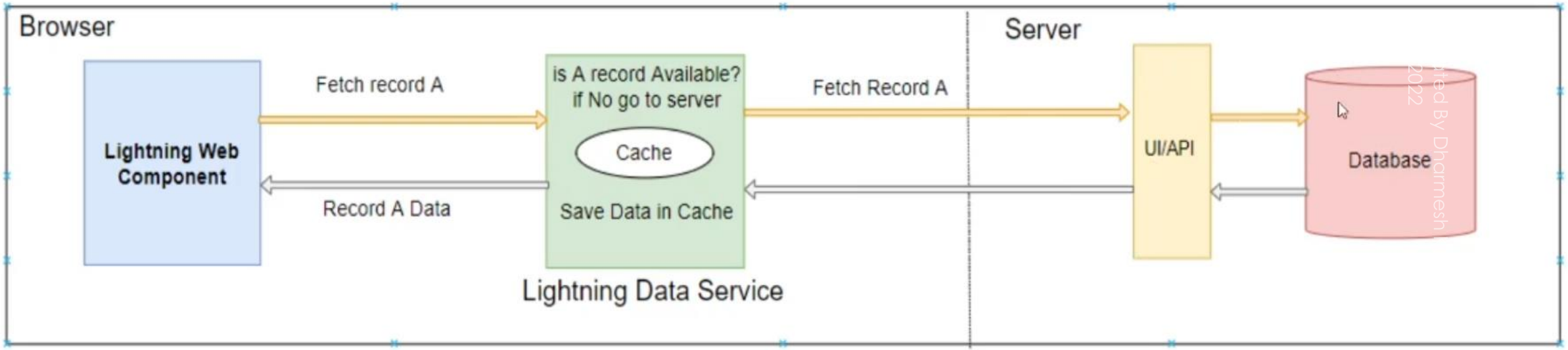
Syntax

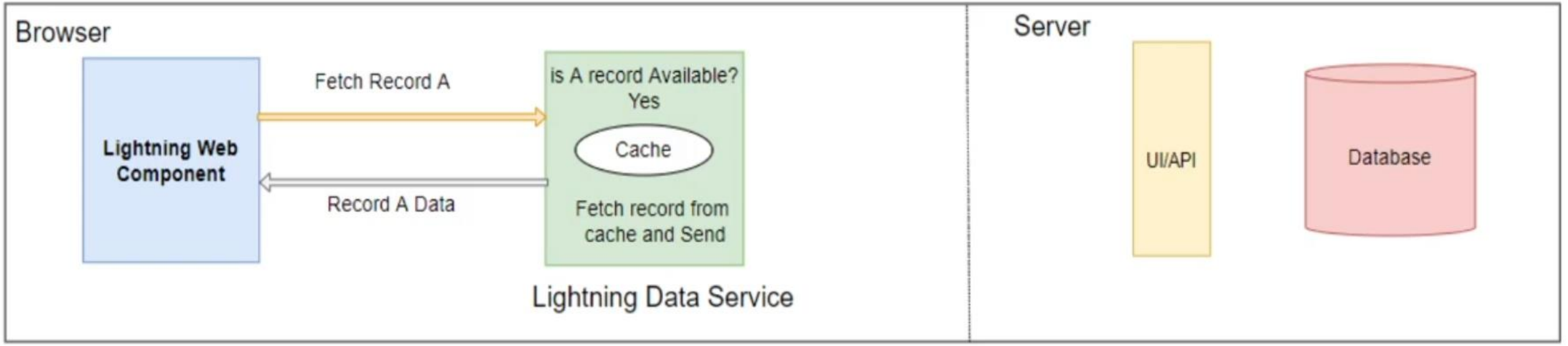
```
import { updateRecord } from 'lightning/uiRecordApi';
updateRecord(recordInput: Record, clientOptions?: Object): Promise<Record>
```

recordInput-A Record Input object is used to update the object.
clientOptions- (optional) - To check for conflicts before you update a record

Lightning Data Service

Lightning Data Service is a centralized data caching framework and it is built on top of User Interface API

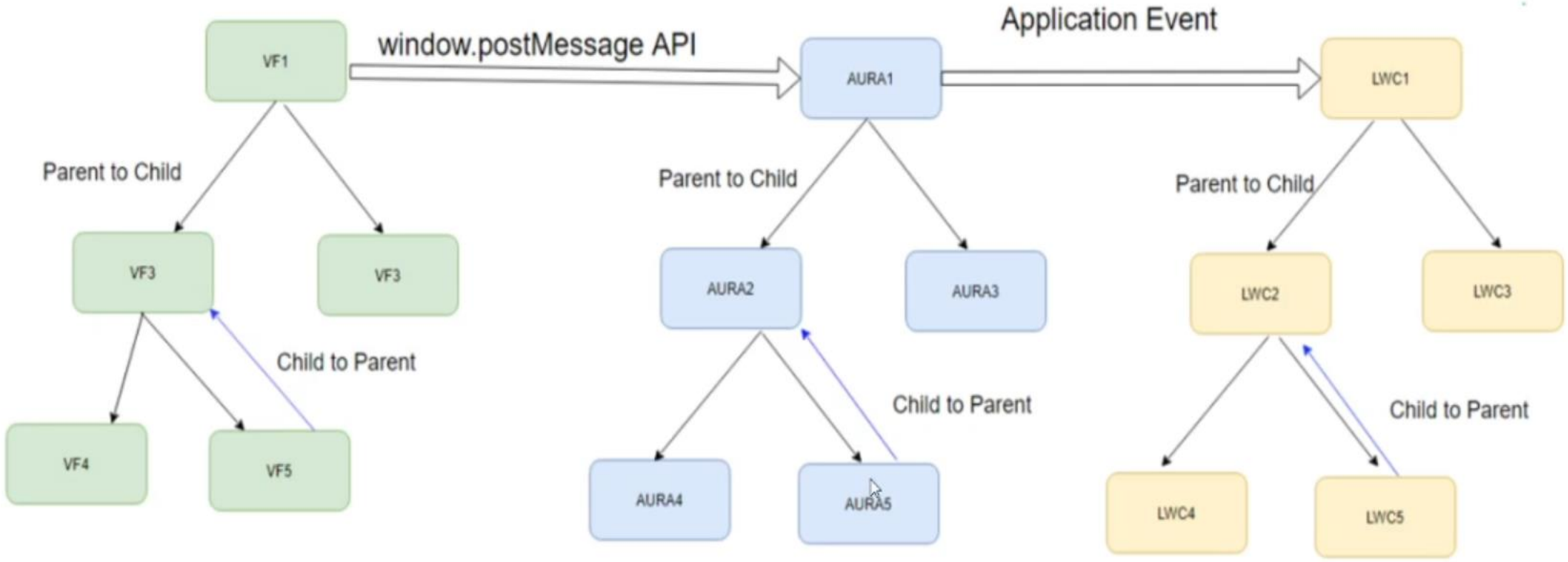




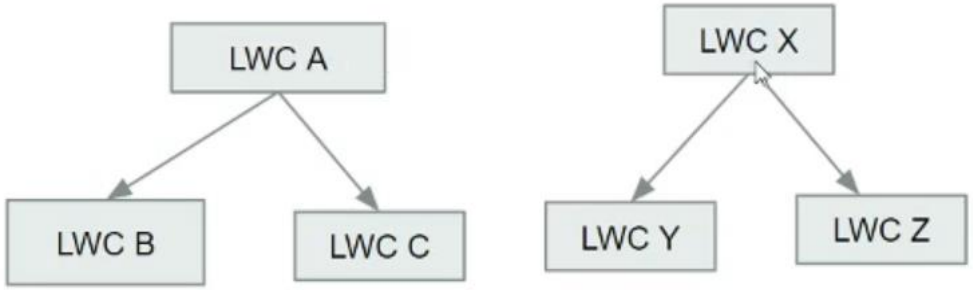
Lightning Messaging Service

LMS is the first Salesforce technology which enables you to communicate between Visualforce, Aura component and Lightning web component anywhere in lightning pages including utility components.

In Winter 20, Salesforce is released "Lightning Message Service" (LMS)



LWC TO LWC Communication using LMS



1. Reference a message channel in LWC

```
import SAMPLEMC from "@salesforce/messageChannel/SampleMessageChannel__c"
```

2. Import LMS API

```
import { APPLICATION_SCOPE, publish, subscribe, unsubscribe, MessageContext } from 'lightning/messageService';
```

3. MessageContext Wire Adapter is use to get information of all LWC using LMS

```
@wire(MessageContext)
context;
```

deleteRecord(recordId)

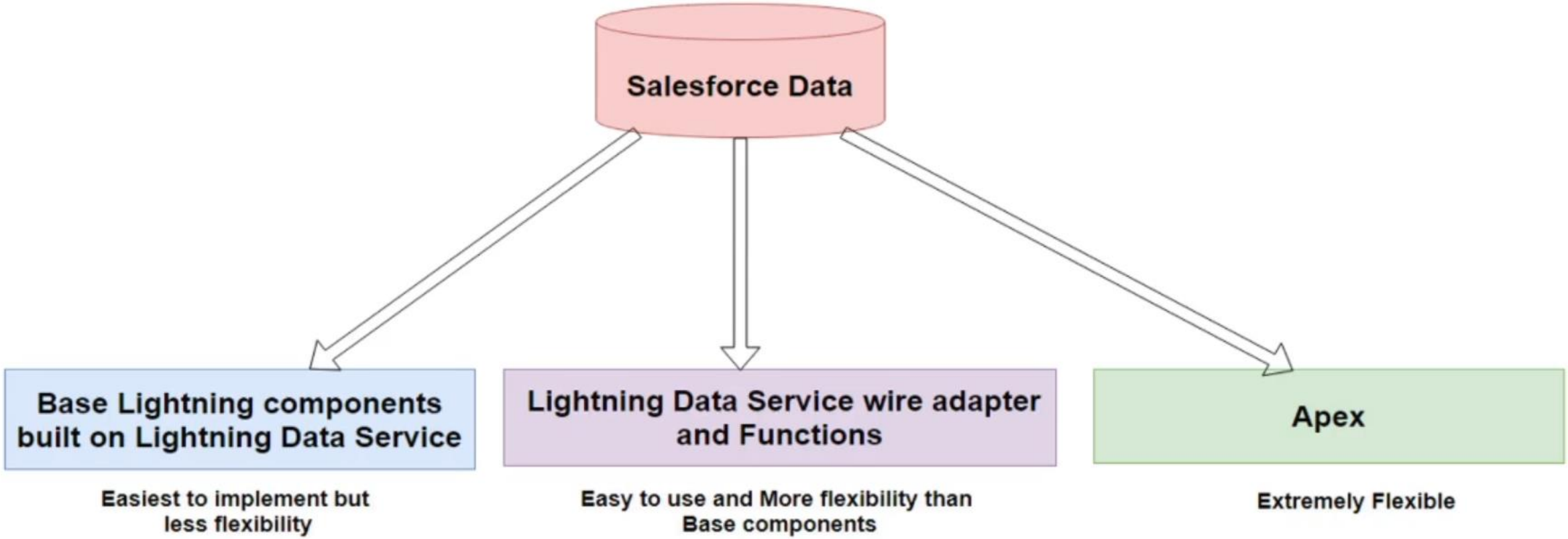
Use this function to delete a record

Syntax

```
import { deleteRecord } from 'lightning/uiRecordApi';
deleteRecord(recordId: string): Promise<void>
```

`recordId`—(Required) The ID of the record to delete.

Apex in LWC

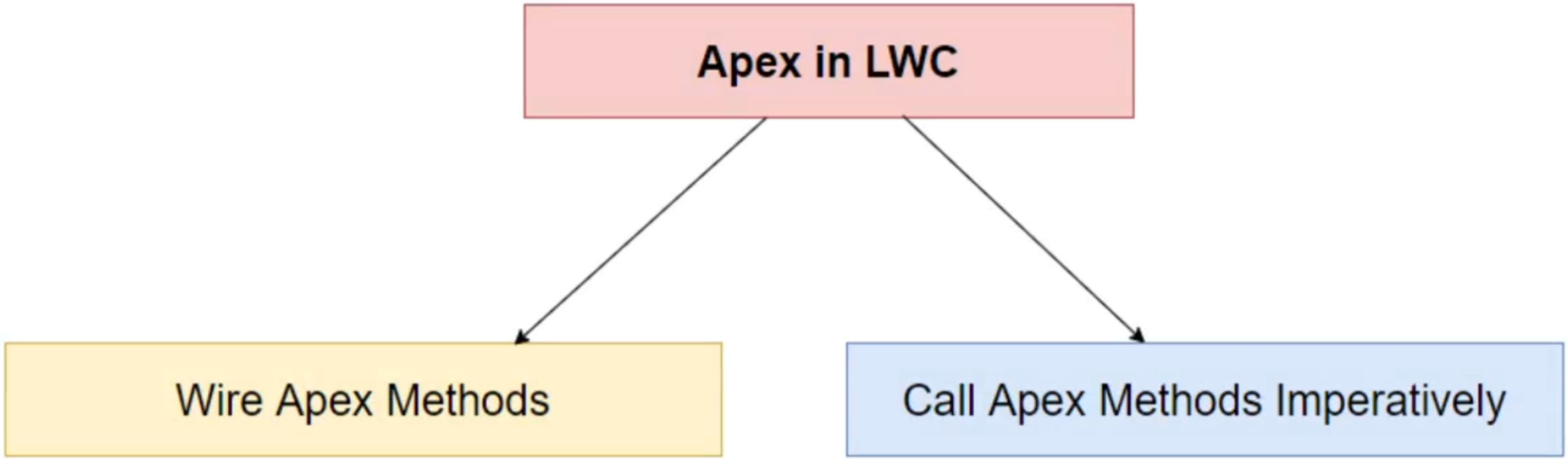


If you can't use a base component, and you can't use the Lightning Data Service wire adapters or functions, use Apex.

When to Use Apex approach

- To work with objects that aren't supported by User Interface API, like Task and Event.
- To work with operations that User Interface API doesn't support, like loading a list of records by criteria (for example, to load the first 200 Accounts with Amount > $1M).
- To perform a transactional operation. For example, to create an Account and create an Opportunity associated with the new Account. If either create fails, the entire transaction is rolled back.

Two Ways to Call Apex Methods in LWC



Expose Apex Methods to LWC

1. Apex Method must be ***static*** and either ***global*** or ***public***
2. Method should be annotated with ***@AuraEnabled***

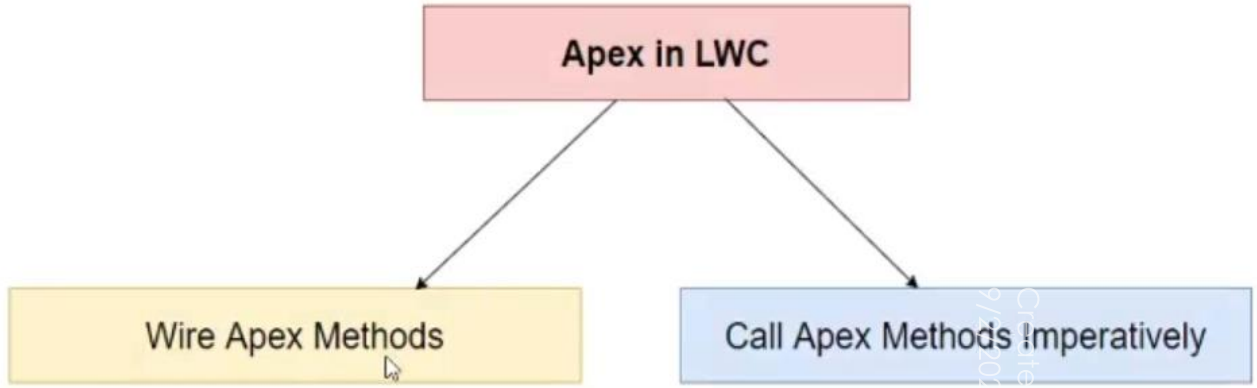
Syntax

```
public with sharing class AccountController {
    @AuraEnabled(cacheable=true)
    public static List<Account> getAccountList() {
        return [SELECT Id, Name, Type, Industry from Account];
    }
}
```

wire Apex Methods

We can wire an apex methods in two ways

1) Wire an Apex Method to a Property
2) Wire an Apex Method to a Function



Syntax

```
import apexMethodName from '@salesforce/apex/Namespace.Classname.apexMethodReference';
@wire(apexMethodName, { apexMethodParams })
propertyOrFunction;
```

- `apexMethodName`—A symbol that identifies the Apex method.
- `apexMethodParams` -- An object with properties that match the parameters of the apexMethod

Note* If a parameter value is null, the method is called but if the value is undefined, the method isn't called.**

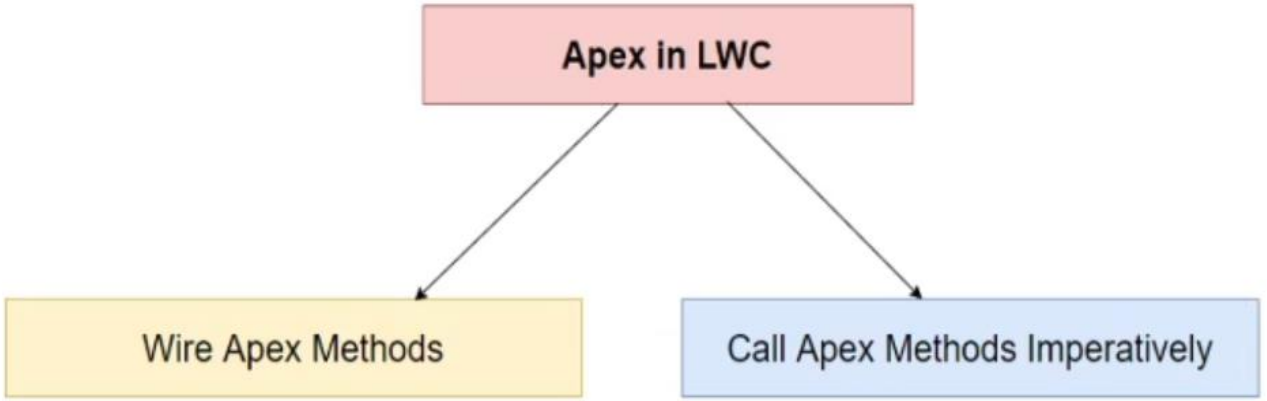
Call Apex Methods Imperatively

Use this approach over @wire in the following situations

1) To call a method that isn't annotated with `cacheable=true`, which includes any method that inserts, updates, or deletes data.
2) To control when the invocation occurs.
3) To work with objects that aren't supported by User Interface API, like Task and Event.
4) To call a method from an ES6 module that doesn't extend `LightningElement`

Syntax

```
fetchData() {
    getAccountList()
        .then((result) => {
            this.accounts = result;
            this.error = undefined;
        })
        .catch((error) => {
            this.error = error;
            this.accounts = undefined;
        });
}
```



Created By Dharmesh Kumar
9/2/2022