Lightning Component State Management

Unidirectional data flow, smart components, dumb components, and Lightning. Oh My.

Mike Tetlow
CTO Bracket Labs
mike@bracketlabs.com
@Mikename

Common Lightning Development Questions:

How do I tell the parent something happened on the child?

How can a component communicate with a sibling component?

General answer is events!

How do I add/delete a bunch of components via javascript?

How do I pass a value by reference or by value?

Can we make this more obvious/less of a problem via a design pattern?

The Problem:

State and where it is held

What is state?

- Data you load from the server
- Pending changes on the client side
- Newly modified data from the server

How do you change state?

- Some kind of user interaction =>
  - A non-persisted client side change
  - A persisted server side change
  - Some random background action

Why do you change state?

- Some kind of user interaction necessitates change of data or how the data is displayed

Can every component have state?

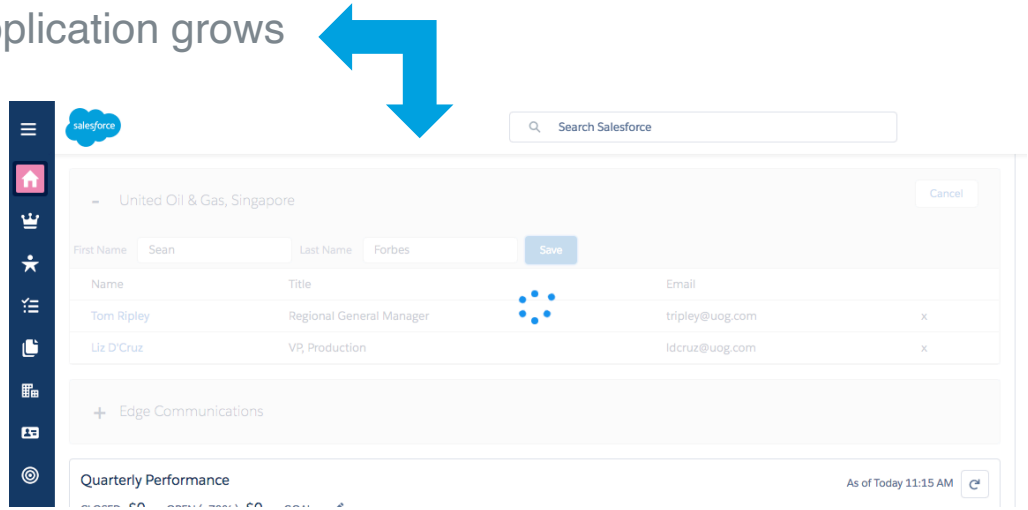
- Sure… should it?

salesforce

The Solution:

Keep all state in one spot

- Benefits
  - One source of truth
  - Lots of times state winds up shared between components as an application grows
  - Debugging
  - Large teams working on the same codebase
- Cons
  - One source of truth means that you have to change your thinking
  - More work
    - You have to go up the component tree to modify
    - You have to pass more stuff down the component tree



How do?

One component holds the state and is “Smart”

- Smart components do
  - Fetch data from the server
  - Apply changes to the application state
  - Persist data on the server
- Smart components do not
  - Deal with complex UI layout

Other components are “Dumb”

- Dumb components do
  - Deal with complex UI layout and all presentation of data
  - Say something has happened from the user
- Dumb components DO NOT
  - Mutate State
  - Temporarily change the presentation of data

salesforce

Smart Component Shape

What does a Smart Component look like?

- Needs
  - Handle data fetching
  - Handle what dumb components say
  - Store and handle all application state
    - Keep track of how dumb components should present data

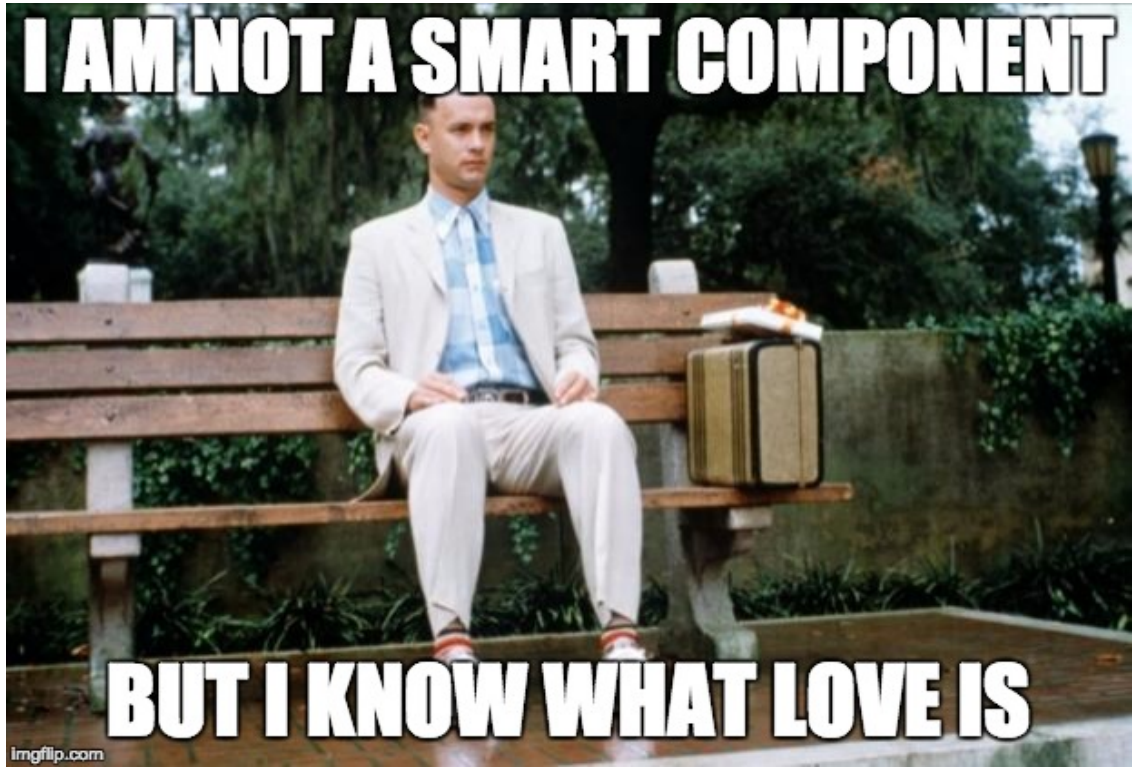
```
<aura:component implements="flexipage:availableForAllPageTypes" controller="flAccountEditorCtrl">
  <aura:attribute name="accounts" type="Array" />
  <aura:attribute name="expandedAccountId" type="String" />
  <aura:attribute name="newContactExpandedAccountId" type="String" />
  <aura:attribute name="operationPending" type="Boolean" default="false" />

  <aura:handler name="init" value="{!this}" action="{!c.doInit}" />
  <aura:handler name="genericEvent" event="c:flGenericEvent" action="{!c.handleGenericEvent}"/>
  <div class="account-editor--container">
    <aura:if isTrue="{!v.operationPending == true}">
      <c:flLoadingMask />
    </aura:if>
    <aura:iteration items="{!v.accounts}" var="account">
      <c:flAccountEditor_accountDetails_view
        account="{!account}"
        isExpanded="{!(account.Id == v.expandedAccountId) ? true : false}"
        showNewContactForm="{!(account.Id == v.newContactExpandedAccountId) ? true : false}" />
    </aura:iteration>
  </div>
</aura:component>
```

Dumb Component Shape

What does a Dumb Component look like?

- Needs
  - Do a whole lot of layout
  - Say the user has done something
  - All UX work



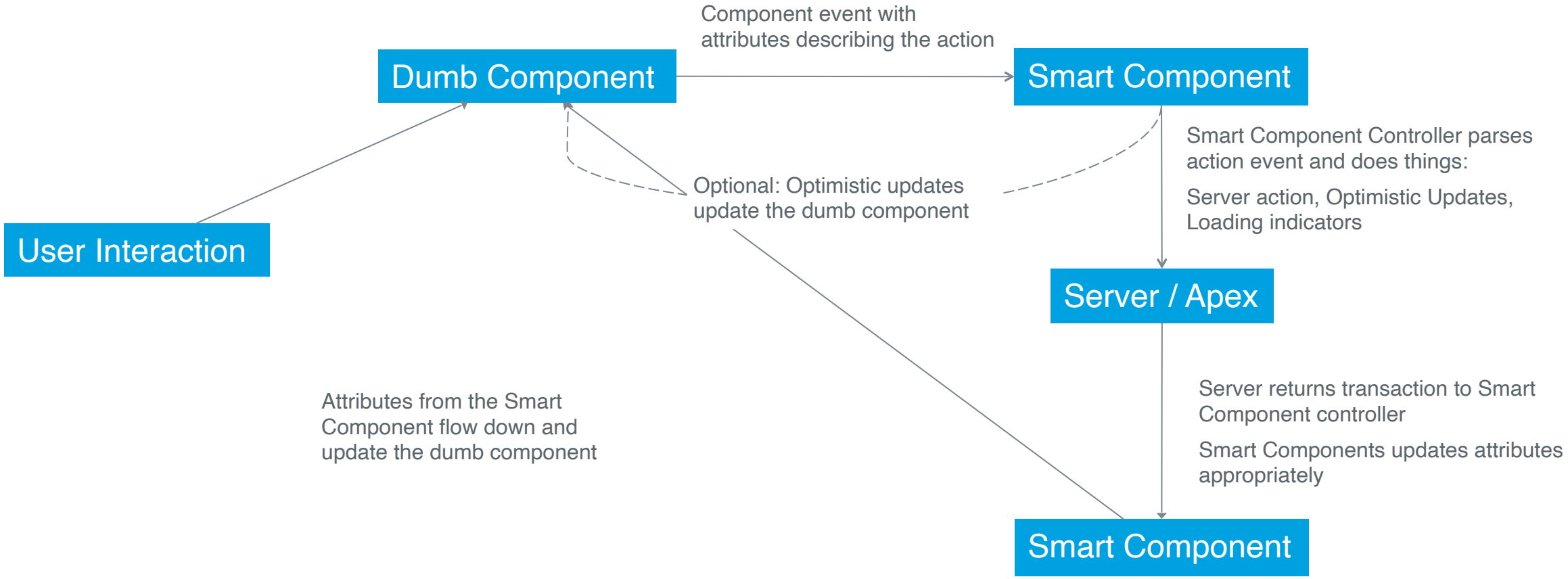
Dumb Component -> Smart Component Communication

When something happens on a dumb component, the smart component needs to be informed to change application state

- Needs
  - Execute something on the parent component
    - You can pass a attribute as an AURA.Action that maps to a function on the parent component
    - You can fire a component event from the dumb component and handle it in the smart component
- Events approach
  - We can make a single generic action event and identify the actions via a key, and provide needed attributes within the event
  - We can make a generic handler component that looks a lot like a Flux store

salesforce

Communication Flow



Let’s make a component!

- United Oil & Gas, Singapore

Cancel

First Name | Last Name | Save

Name	Title	Email	
Tom Ripley	Regional General Manager	tripley@uog.com	x
Liz D'Cruz	VP, Production	ldcruz@uog.com	x
Sean Forbes			x

+ Edge Communications

Smart: Account Editor (Container)
Dumb: Account Detail
Dumb: Contact Detail

salesforce

Sample App Demo

https://github.com/mtetlow/Fluxy-Lightning/

salesforce

What do we think about this approach?

- Cons
  - Lots of typing
  - Component Event bubbling is a little weird
    - Events are great, they allow us to do a lot of cross component communication
    - But, they don't bubble like traditional DOM events. WTF is a facet provider? and why does the average lightning developer need to know that?
- Pros
  - Consistency is key
    - If you are developing a large set of components, keep them consistent, working between components that have different philosophies on where state lives will be miserable.
    - Large teams all understand
- Possible Gotchyas
  - Render speed? Solved in React world via immutable data and lifecycle hooks. How do in Lightning world?

salesforce

Lessons learned

- We all know state is evil
  - State is bad because if it mutates and you're not expecting it, strange things happen in your app
  - Keeping the locations where state is mutated contained dramatically increase the speed at which issues can be resolved
- Apply proven concepts to Salesforce development
  - While the APIs and shape of Salesforce front end dev are new and different, lessons learned from all component based front end dev apply to Lightning Components

salesforce

Q&A

https://github.com/mtetlow/Fluxy-Lightning/
@Mikename