Building Better Lightning Components: Reusable and Generic

Christian Menzinger, Product Developer
cmenzinger@appero.com, @chris_menzinger, @appero_com

“Please allow me to introduce myself”

Sympathy for the Devil - Rolling Stones

Christian Menzinger

@chris_menzinger

Working for ISV partner appero

3.5 years Salesforce Developer

Co-Leader Munich Developer Group

appero
smart use of the force

salesforce registered ISV partner

http://www.appero.com

@appero_com

appero-com

SALESFORCE CERTIFIED Administrator

SALESFORCE CERTIFIED Force.com Developer

salesforce developers

Developer Group Leader - Munich, Germany

salesforce

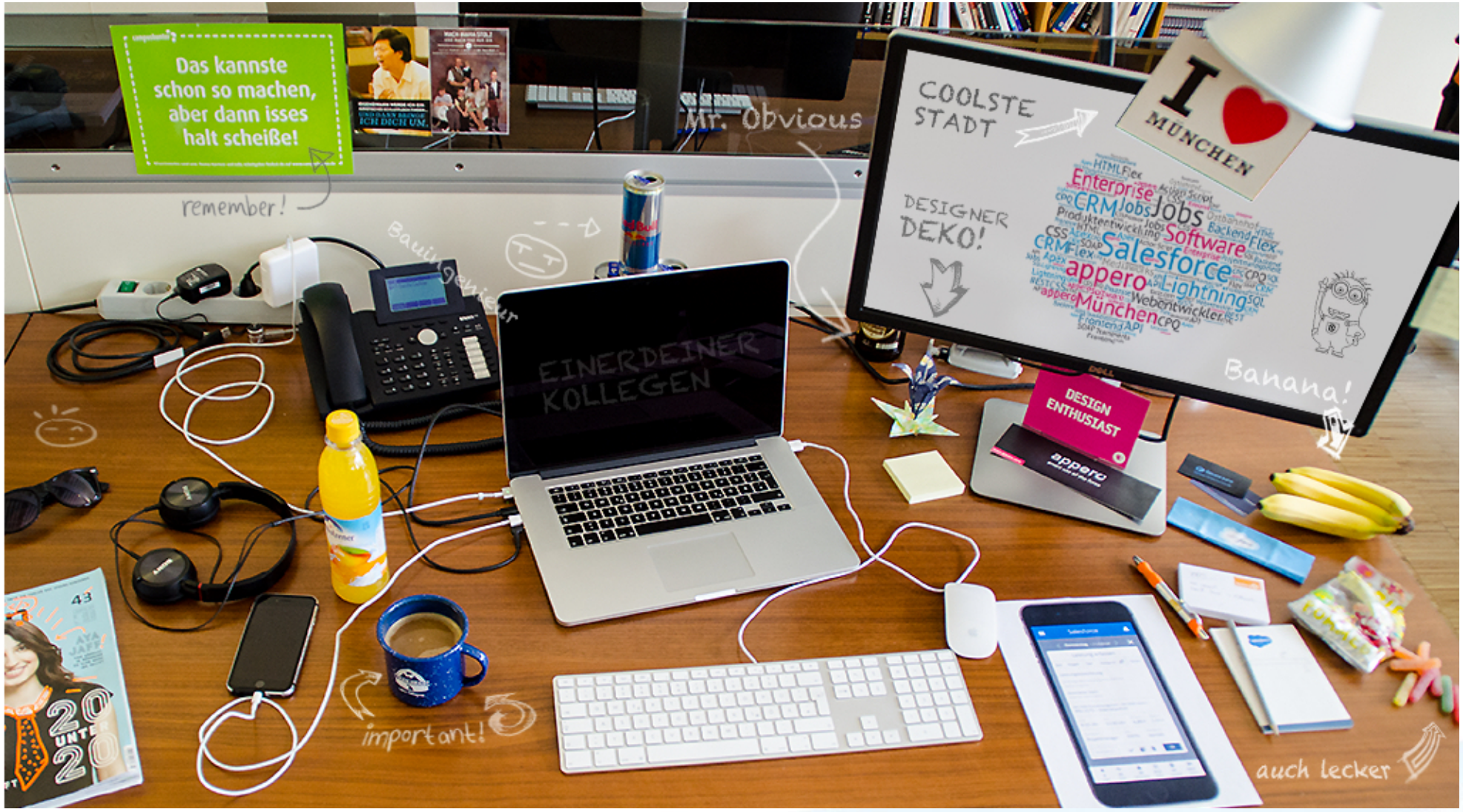
Objectives

Climbing up the hill

Leveraging the framework

My Life as a Product Developer

Example: Calling a server side action using Apex



My Life as a Product Developer

Google Search Result

Lightning Components Developer Documentation ->

Ever copy & paste this code?

Calling a Server-Side Action

Call a server-side controller action from a client-side controller. In the client-side controller, you set a callback, which is called after the server-side action is completed. A server-side action can return any object containing serializable JSON data.

A client-side controller is a JavaScript object in object-literal notation containing a map of name-value pairs.

Let's say that you want to trigger a server-call from a component. The following component contains a button that's wired to a client-side controller `echo` action. `SimpleServerSideController` contains a method that returns a string passed in from the client-side controller.

```
<aura:component controller="SimpleServerSideController">
    <aura:attribute name="firstName" type="String" default="world"/>
    <lightning:button label="Call server" onclick="{!c.echo}"/>
</aura:component>
```

This client-side controller includes an `echo` action that executes a `serverEcho` method on a server-side controller.

Use unique names for client-side and server-side actions in a component. A JavaScript function (client-side action) with the same name as an Apex method (server-side action) can lead to hard-to-debug issues. In debug mode, the framework logs a browser console warning about the clashing client-side and server-side action names.

```
({
    "echo" : function(cmp) {
        // create a one-time use instance of the serverEcho action
        // in the server-side controller
        var action = cmp.get("c.serverEcho");
        action.setParams({ firstName : cmp.get("v.firstName") });

        // Create a callback that is executed after
        // the server-side action returns
        action.setCallback(this, function(response) {
            var state = response.getState();
            if (state === "SUCCESS") {
                // Alert the user with the value returned
                // from the server
                alert("From server: " + response.getReturnValue());

                // You would typically fire a event here to trigger
                // client-side notification that the server-side
                // action is complete
            }
            else if (state === "INCOMPLETE") {
                // do something
            }
            else if (state === "ERROR") {
                var errors = response.getError();
                if (errors) {
                    if (errors[0] && errors[0].message) {
                        console.log("Error message: " +
                                 errors[0].message);
                    }
                } else {
                    console.log("Unknown error");
                }
            }
        });

        // optionally set storable, abortable, background flag here

        // A client-side action could cause multiple events,
        // which could trigger other events and
        // other server-side action calls.
        // $A.enqueueAction adds the server-side action to the queue.
        $A.enqueueAction(action);
    }
})
```

salesforce


Is this production code?

```
({
    "echo" : function(cmp) {
        // create a one-time use instance of the serverEcho action
        // in the server-side controller
        var action = cmp.get("c.serverEcho");
        action.setParams({ firstName : cmp.get("v.firstName") });

        // Create a callback that is executed after
        // the server-side action returns
        action.setCallback(this, function(response) {
            var state = response.getState();
            if (state === "SUCCESS") {
                // Alert the user with the value returned
                // from the server
                alert("From server: " + response.getReturnValue());

                // You would typically fire a event here to trigger
                // client-side notification that the server-side
                // action is complete
            }
            else if (state === "INCOMPLETE") {
                // do something
            }
            else if (state === "ERROR") {
                var errors = response.getError();
                if (errors) {
                    if (errors[0] && errors[0].message) {
                        console.log("Error message: " +
                                 errors[0].message);
                    }
                } else {
                    console.log("Unknown error");
                }
            }
        });

        // optionally set storable, abortable, background flag here

        // A client-side action could cause multiple events,
        // which could trigger other events and
        // other server-side action calls.
        // $A.enqueueAction adds the server-side action to the queue.
        $A.enqueueAction(action);
    }
})
```

Problem: “I did it my way”

My Way – Frank Sinatra

several Developers

multiple Apex calls

different code locations

nonuniform error handling

have to re-Google

Different implementations leads to inconsistent behavior!


Idea: “One ring to rule them all”
The Lord of the Rings – J.R.R. Tolkien
salesforce

Bridge the gap

Need a consistent behavior

Need one code location

Implementation must be context independent

Don’t want to fiddle around with the same code again and again!

Identify what changes

What are the essentials?

Focus on:

- Apex function name
- calling parameters
- return value

```
({
    "echo" : function(cmp) {
        // create a one-time use instance of the serverEcho action
        // in the server-side controller
        var action = cmp.get("c.serverEcho");
        action.setParams({ firstName : cmp.get("v.firstName") });

        // Create a callback that is executed after
        // the server-side action returns
        action.setCallback(this, function(response) {
            var state = response.getState();
            if (state === "SUCCESS") {
                // Alert the user with the value returned
                // from the server
                alert("From server: " + response.getReturnValue());

                // You would typically fire a event here to trigger
                // client-side notification that the server-side
                // action is complete
            }
            else if (state === "INCOMPLETE") {
                // do something
            }
            else if (state === "ERROR") {
                var errors = response.getError();
```

“One Size Fits All” – Really?

Frank Zappa and the Mothers of Invention

Different server calls lead to different responses

Processing response cannot be part of a generic approach

Apex calls are asynchronous

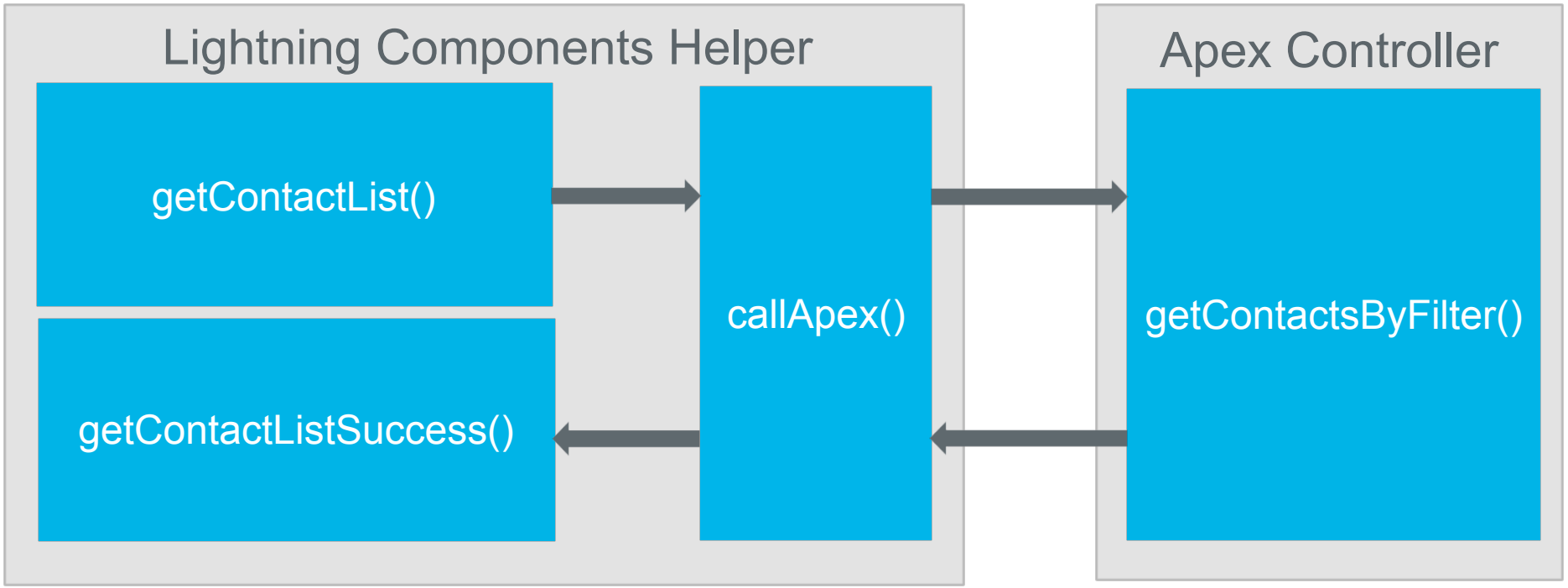
Need a callback

OOP: behavior = method

Prototyping: using a helper function

Callback flow

Use case: load list of Contacts



Prototyping: “v1UsingHelperMethodHelper.js”

Only little changes from original code from developer documentation

```
({
    callApex: function(component, controllerMethod, actionParameters, successCallback) {
        // create a one-time use instance of the serverEcho action
        // in the server-side controller
        var action = component.get(controllerMethod);
        action.setParams(actionParameters);

        // Create a callback that is executed after
        // the server-side action returns
        action.setCallback(this, function(response) {
            var state = response.getState();
            if (state === "SUCCESS") {
                //->HERE WE CALL THE CALLBACK RATHER PROCESS THE RESPONSE
                successCallback(component, response.getReturnValue())
            }
            else if (state === "INCOMPLETE") {
```

Prototyping: “v1UsingHelperMethodHelper.js”

Only little changes from original code

```
({
    getContactList: function(component, filterString) {
        this.callApex(component, "c.getContactsByFilter", {"filter": filterString}, this.getContactListSuccess);
    },

    callApex: function(component, controllerMethod, actionParameters, successCallback) {
        // create a one-time use instance of the serverEcho action
        // in the server-side controller
        var action = component.get(controllerMethod);
        action.setParams(actionParameters);

        // Create a callback that is executed after
        // the server-side action returns
        action.setCallback(this, function(response) {
            var state = response.getState();
            if (state === "SUCCESS") {
                //->HERE WE CALL THE CALLBACK RATHER PROCESS THE RESPONSE
                successCallback(component, response.getReturnValue())
            }
            else if (state === "INCOMPLETE") {
```

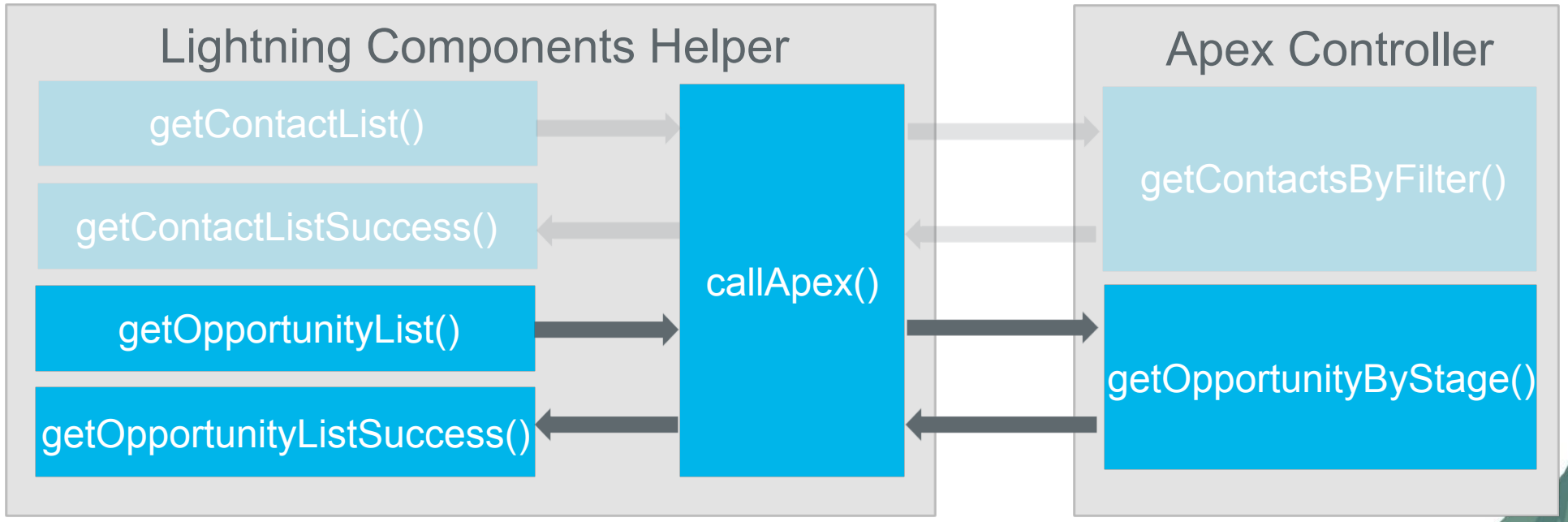
Prototyping: “v1UsingHelperMethodHelper.js”

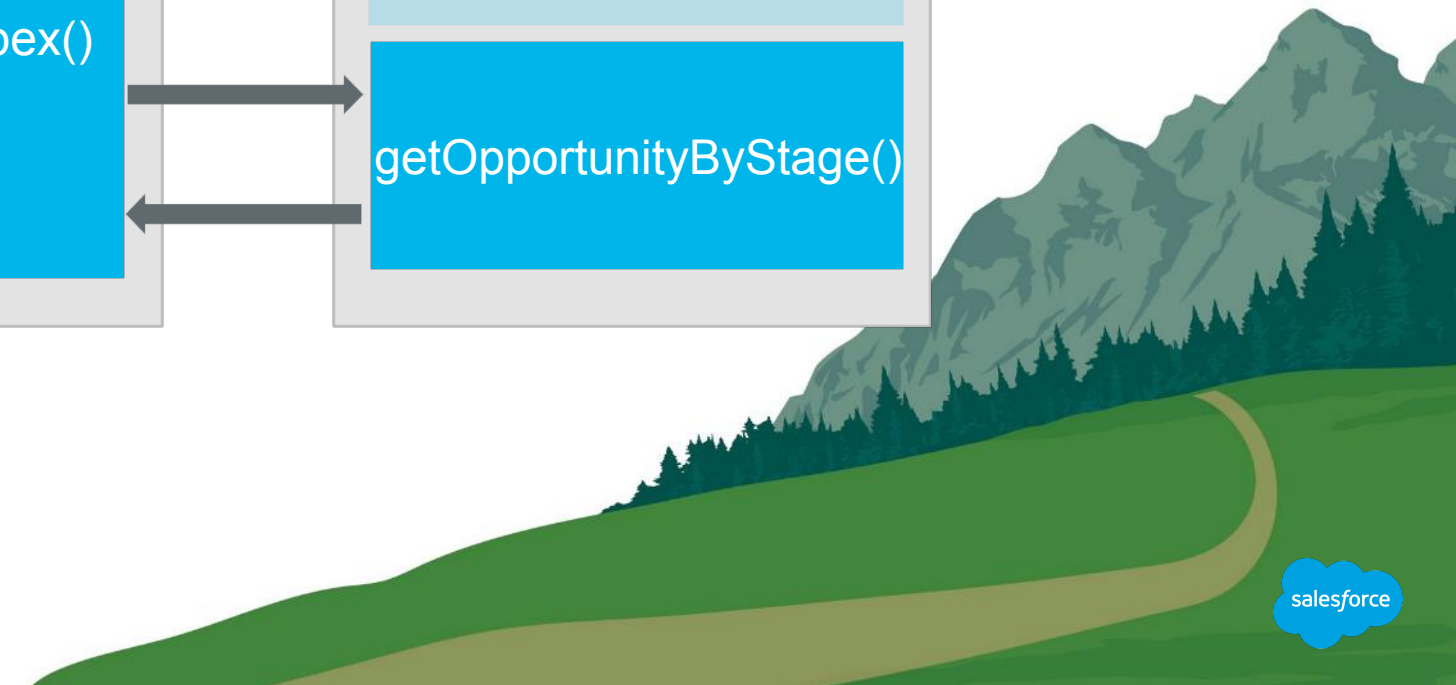
Only little changes from original code

```
        // which could trigger other events and
        // other server-side action calls.

        // $A.enqueueAction adds the server-side action to the queue.
        $A.enqueueAction(action);
    },

    getContactListSuccess: function(component, returnValue) {
        //process result in some way
        component.set("v.contactList", returnValue);
    }
})
```

How far do we get?

“callApex” can be reused for a second server call:





Problem:

Only possible within the **same helper**

Only **one Apex Controller** per component

Reusability

How can other components use this **abstraction** of an Apex service?

Encapsulate functionality!

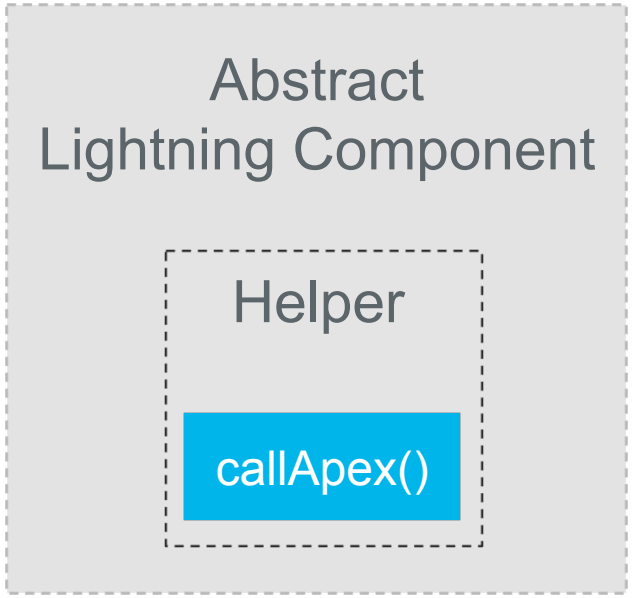
OOP: **class** = **extensible** program-code-template

Idea:

Create an **abstract** base **class** where other components can **extend** from!

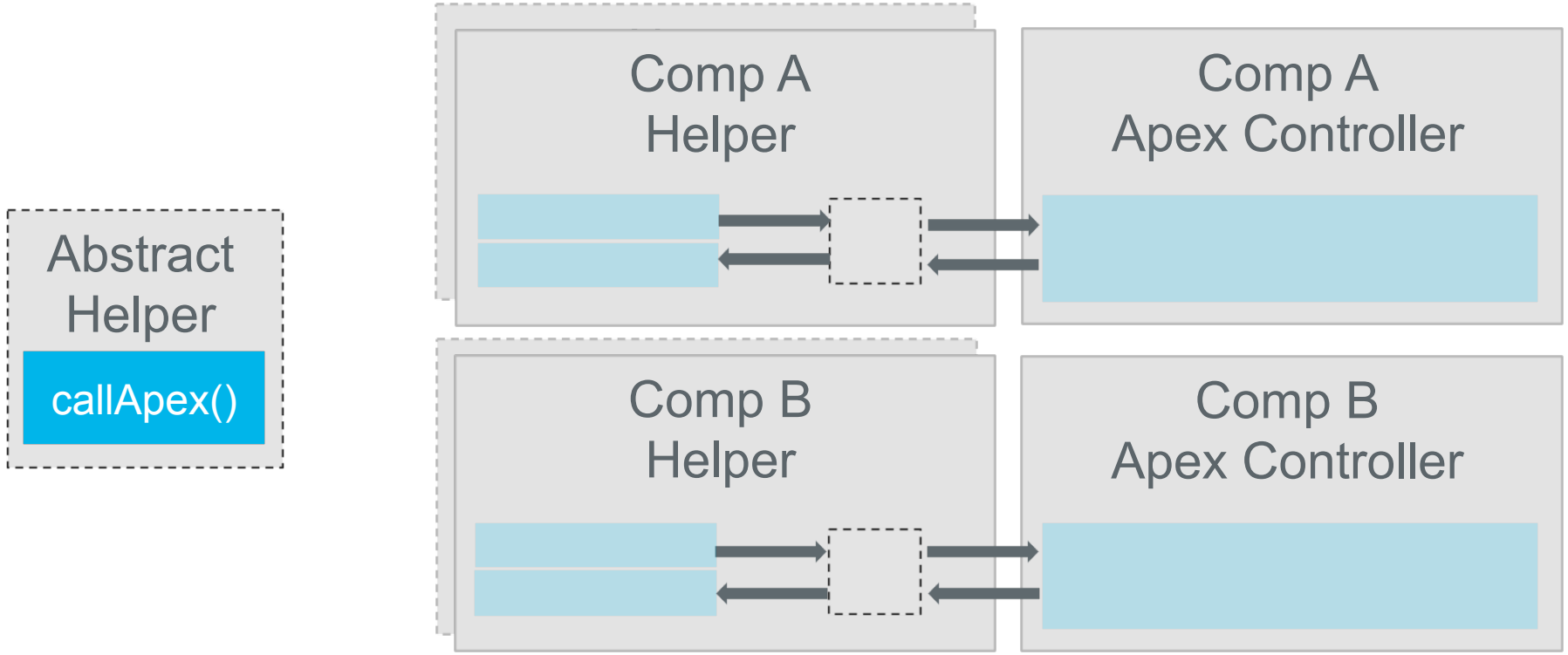
Abstract base class

Only need a helper



Abstract base class

Helper function is inherited by a concrete component



Prototyping: “v2AbstractComponent.cmp”

Abstract base class

```
1 <aura:component abstract="true" extensible="true">
2     {!v.body}
3 </aura:component>
```

Don‘t forget {!v.body}

No controller needed!

Prototyping: “v2AbstractComponentHelper.js”

Copy v1-Helper-JavaScript: callApex() **did not change**

```
({
    callApex: function(component, controllerMethod, actionParameters, successCallback) {
        // create a one-time use instance of the serverEcho action
        // in the server-side controller
        var action = component.get(controllerMethod);
        action.setParams(actionParameters);

        // Create a callback that is executed after
        // the server-side action returns
        action.setCallback(this, function(response) {
            var state = response.getState();
            if (state === "SUCCESS") {
                //->HERE WE CALL THE CALLBACK RATHER PROCESS THE RESPONSE
                successCallback(component, response.getReturnValue())
            }
            else if (state === "INCOMPLETE") {
```

Prototyping: “v2ConcreteComponent.cmp”

Extend the abstract component

```
1 <aura:component extends="c:v2AbstractComponent" controller=
2     [...]
3 </aura:component>
```

Prototyping: “v2ConcreteComponentHelper.js”

Copy v1-Helper-JavaScript getContactList() & getContactListSuccess(): **did not change**

```
({
    getContactList : function(component, filterString) {
        this.callApex(component, "c.getContactsByFilter", {"filter": filterS
    },

    getContactListSuccess : function(component, returnValue) {
        //process result in some way
        component.set("v.contactList", returnValue);
    }
})
```

How far do we get?

“v2AbstractComponent” is **generic** − no hardcoded dependencies

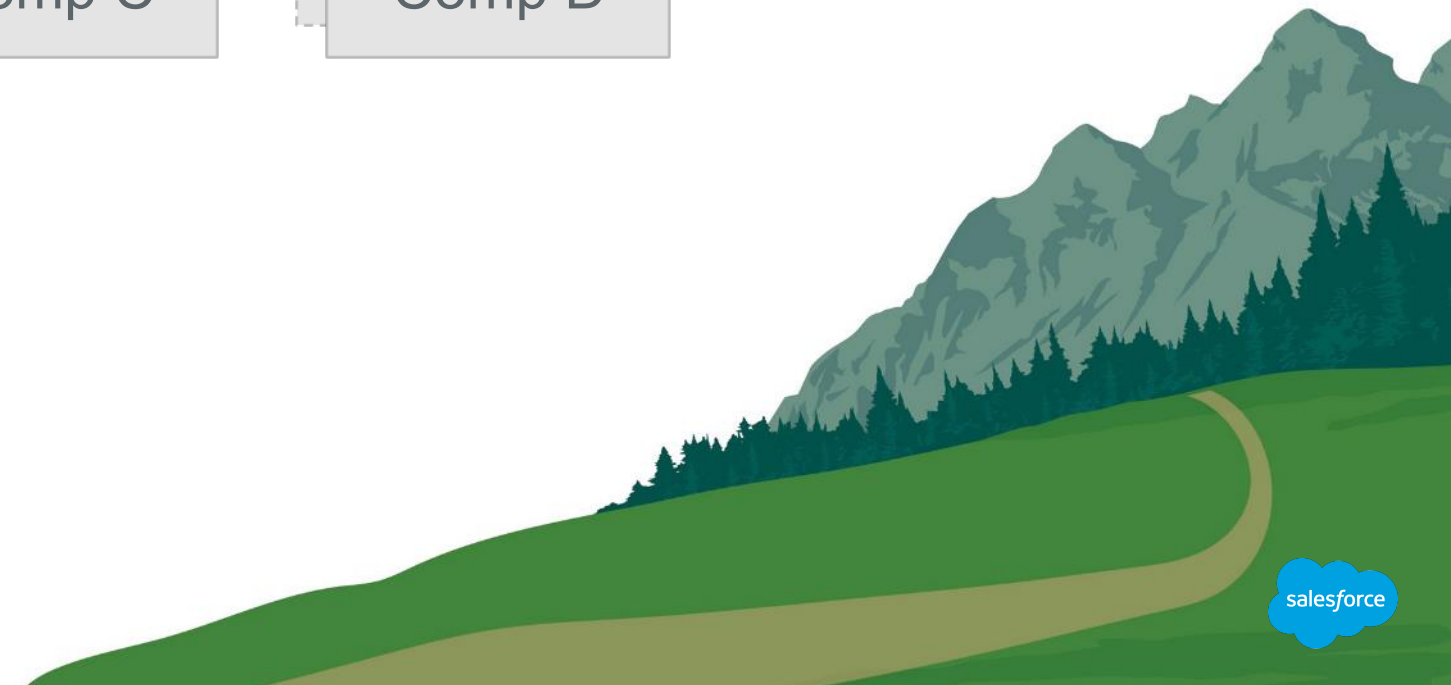
“v2AbstractComponent” is **reusable** − every component can extend if needed

Comp A

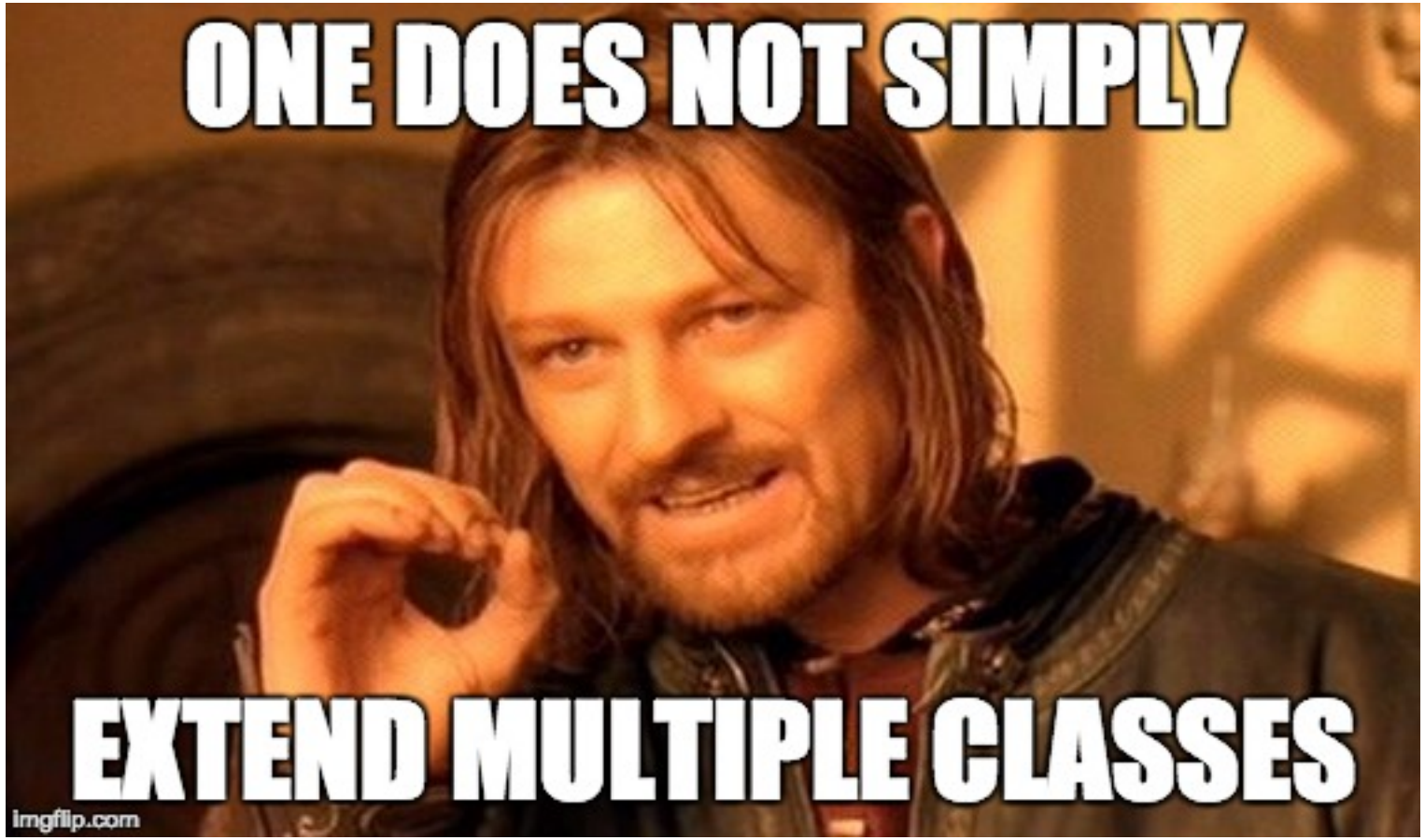
Comp B

Comp C

Comp D



Problem:



Problem: One does not simply extend multiple classes

Inheritance is one to one

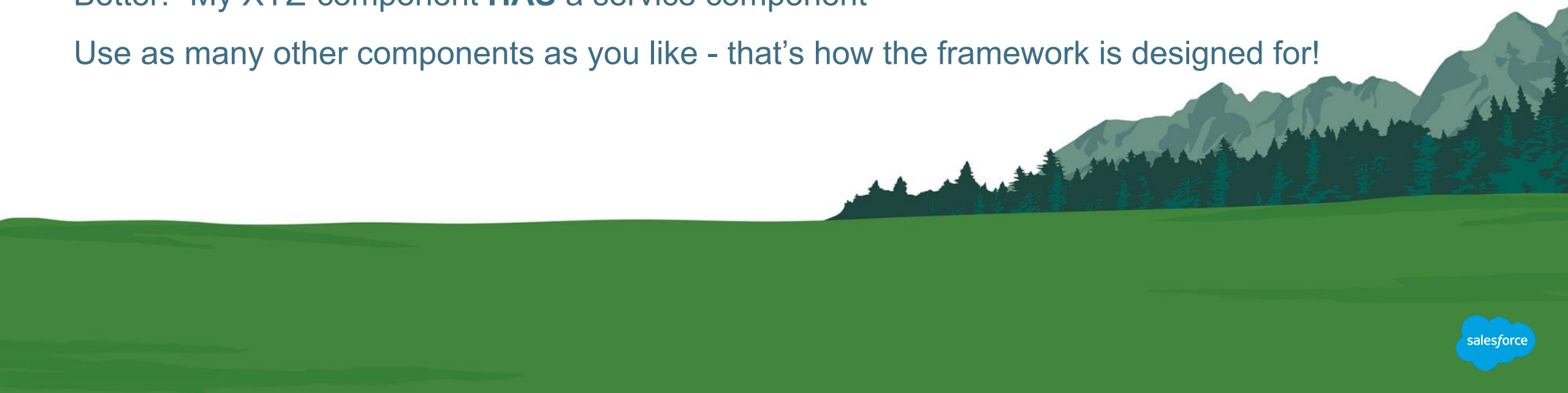
Sometimes already “used” by another use case

OOP: “**Favor composition over inheritance**”!

Now: “My XYZ-component **IS** a service component”

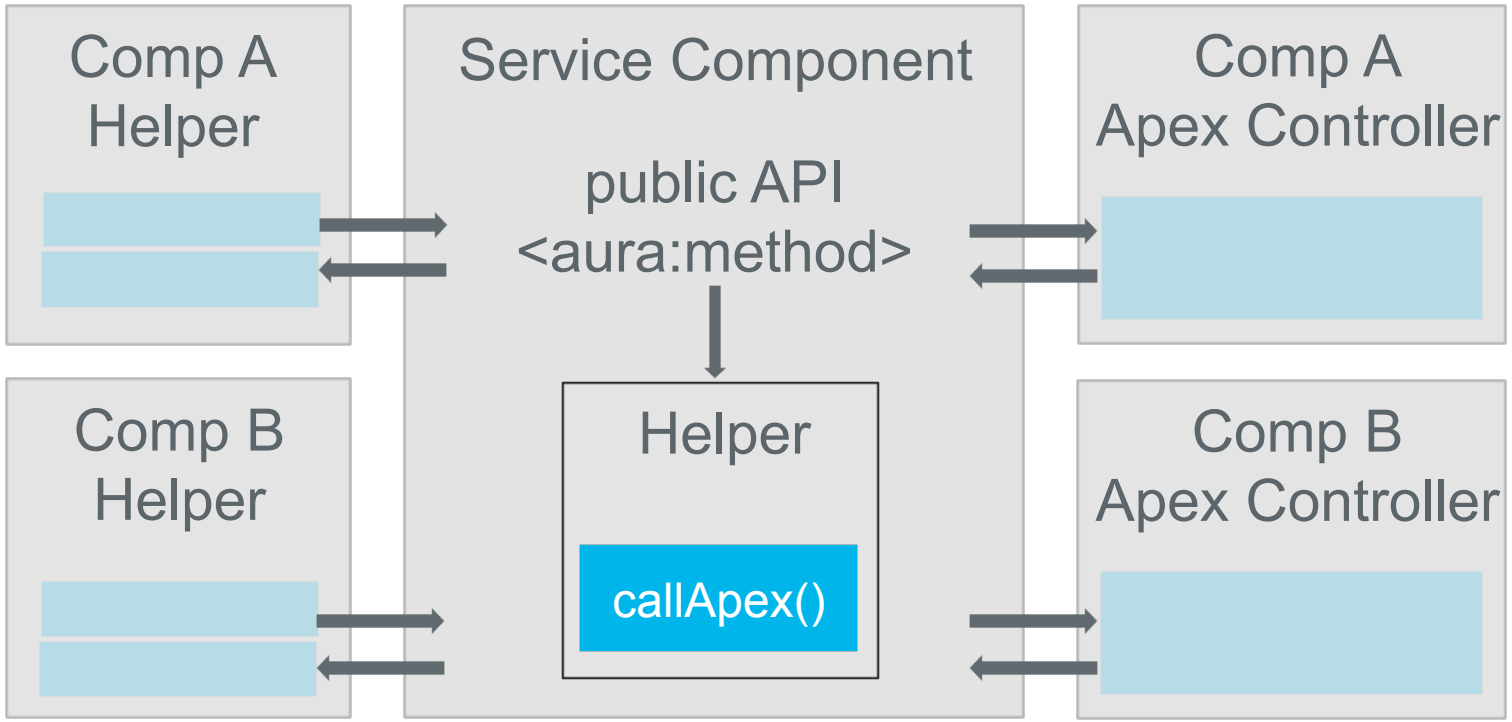
Better: “My XYZ-component **HAS** a service component”

Use as many other components as you like - that’s how the framework is designed for!



Service component

Calling a method in another component



Prototyping: “v3ServiceComponent.cmp”

Copy & modify v2-Component-markup

```
1  <aura:component extensible="true" abstract="true">
2
3
4
5
6
7
8
9
10      {!v.body}
11
12 </aura:component>
```

Prototyping: “v3ServiceComponent.cmp”

Add aura:method

```
1  <aura:component extensible="true" abstract="true">
2
3      <aura:method name="callApex" action="{!c.onCallApex}">
4          <aura:attribute name="component" type="Aura.Component" />
5          <aura:attribute name="controllerMethod" type="String" />
6          <aura:attribute name="actionParameters" type="Object" />
7          <aura:attribute name="successCallback" type="Object" />
8      </aura:method>
9
10     {!v.body}
11
12 </aura:component>
```

Prototyping: “v3ServiceComponentController.js”

Controller invocation from method

```
({
    onCallApex : function(component, event, helper) {
        //get the method parameters
        var params = event.getParams().arguments;
        var callerComponent = params.component;
        var controllerMethod = params.controllerMethod;
        var actionParameters = params.actionParameters;
        var successCallback = params.successCallback;
        helper.callApex(callerComponent, controllerMethod, actionParameters, successCallback);
    }
})
```

Prototyping: “v3ServiceComponentHelper.js”

Copy v2-Helper-JavaScript: callApex() **did not change**

```
({
    callApex: function(component, controllerMethod, actionParameters, successCallback) {
        // create a one-time use instance of the serverEcho action
        // in the server-side controller
        var action = component.get(controllerMethod);
        action.setParams(actionParameters);

        // Create a callback that is executed after
        // the server-side action returns
        action.setCallback(this, function(response) {
            var state = response.getState();
            if (state === "SUCCESS") {
                //->HERE WE CALL THE CALLBACK RATHER PROCESS THE RESPONSE
                successCallback(component, response.getReturnValue())
            }
            else if (state === "INCOMPLETE") {
```

Prototyping: “v3JustAnotherComponent.cmp”

Use the service component

```
1  <aura:component controller="AuraActionDemoController">
2
3      <c:v3ServiceComponent aura:id="service" />
4
5      [...]
6
7  </aura:component>
```

Prototyping: “v3JustAnotherComponentHelper.js”

Invoke the service component

```
({
    getContactList: function(component, filterString) {
        //call aura method from service component:
        component.find("service").callApex(component, "c.getContactsByFilter",
    },

    getContactListSuccess: function(component, returnValue) {
        //process result in some way
        component.set("v.contactList", returnValue);
    }
})
```

Arrive at the summit

Heavy lifting is done!

“Nobody's gonna slow me down”

Highway to Hell - AC/DC

Work on with the service component

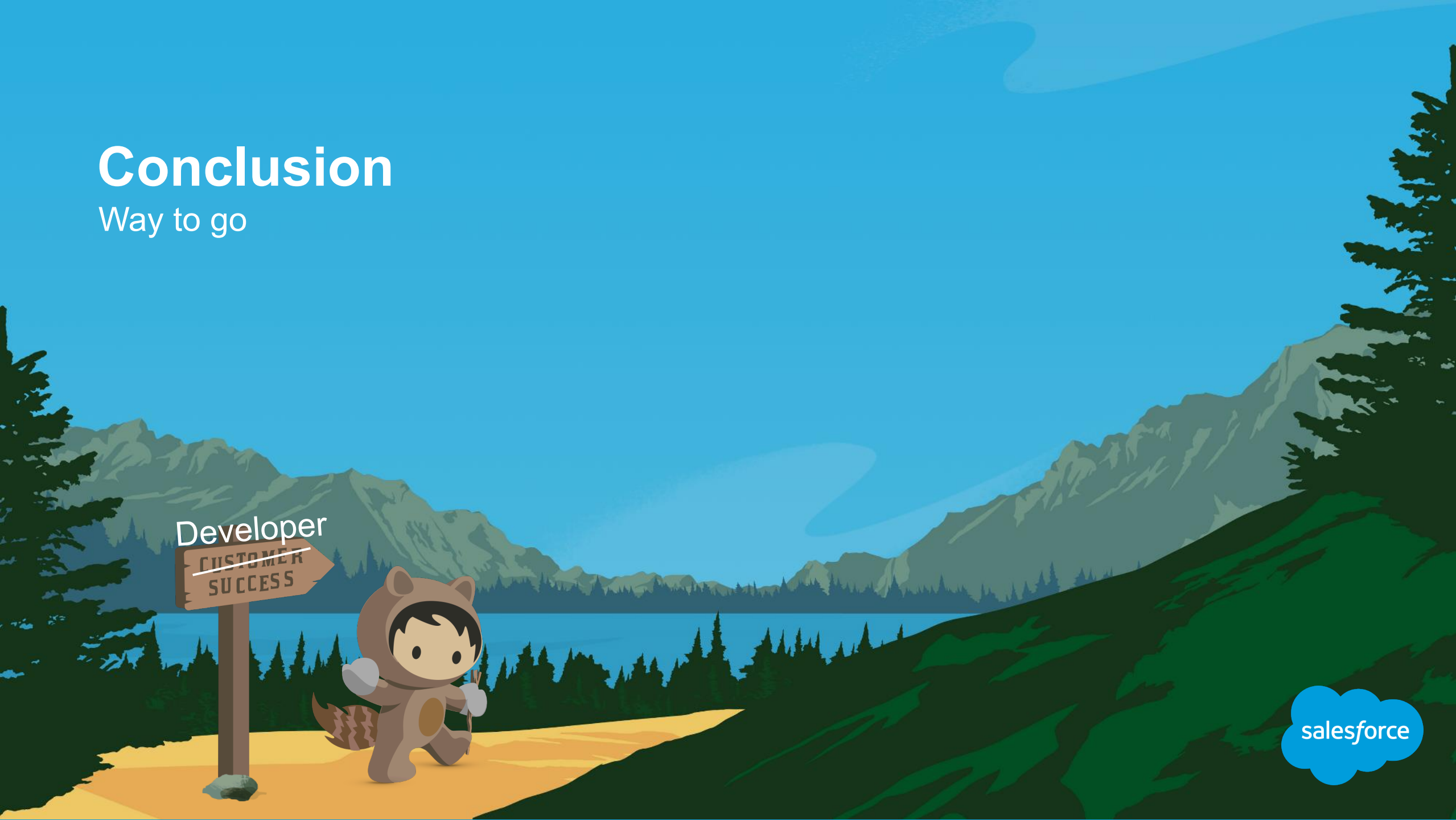
Implement proper error handling

Other improvements: show a spinner, lock GUI, use storable actions

Reusable also means: other developer have to know how to use your component − **Documentation**!

Conclusion

Way to go



Conclusion

What you have to do now

Build more components!

Identify recurring tasks / code!

Be **generic** to be **reusable**!

Use inheritance wisely!

Favor composition!


Thank You

Resource

Calling a Server-Side Action
https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/controllers_server_actions_call.htm

Using Object-Oriented Development
https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/oo_intro.htm

Calling Component Methods
https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/js_cmp_methods.htm

Providing Component Documentation
https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/components_documentation.htm

Code Examples
https://github.com/appero-com/DF17-Building-better-lightning-components

Join a local Developer Group
https://developer.salesforce.com/dugs