Evolving JavaServer™ Faces Technology: AJAX Done Right

Edward Burns, Sun Microsystems, Inc.

Jacob Hookom, McKesson

Adam Winer, Oracle

TS-1161

JavaServer™ Faces Technology Is Great for AJAX, Here's Why

The architecture of the JavaServer Faces platform is perfect for the next era of web applications. JavaServer Faces technology is evolving to better address the details.

What You Will Learn in This Session

Scalability of Application Requirements

How JavaServer Faces helps Manage Complexity

JavaServer Faces and AJAX:
Perfect Together

The Many Faces of Scalability

Leading by Example

Scalability of Application Requirements

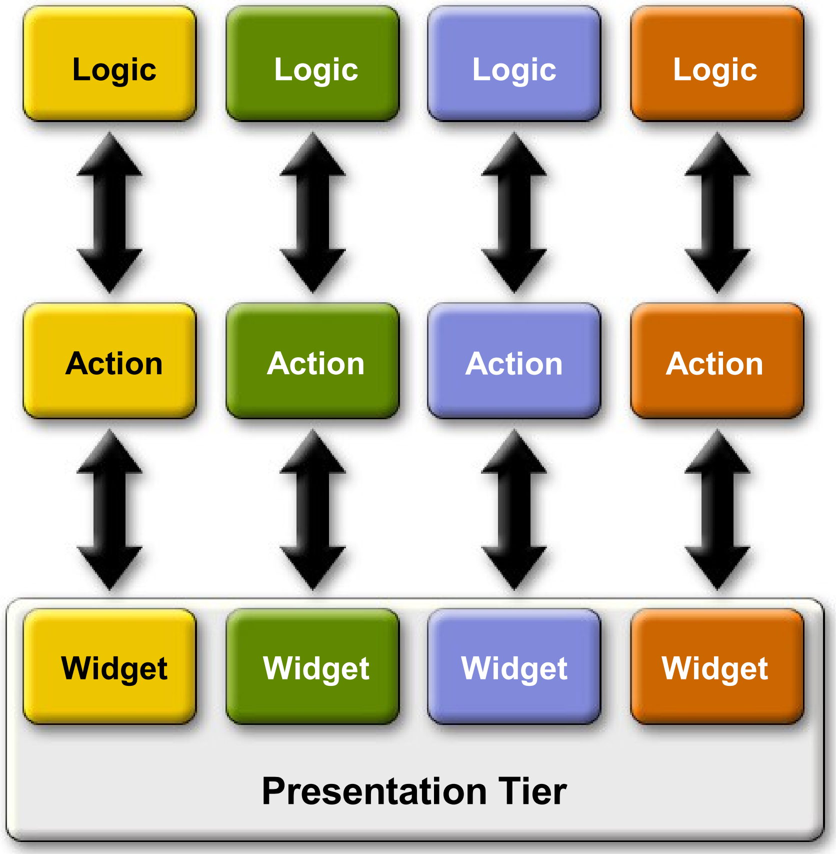
The Complexity of Expectations

- People expect more from your web applications and development today
- Are you going to be able to quickly adapt and scale to application requirements?
- We chose OO to manage application complexity, why throw it away for a procedural UI?
- Simply choosing AJAX is not going to make things any easier for managing complexity

How “Everyone” Does AJAX

Splitting up the presentation

- Essentially breaking up the page into separate ‘parts’
- No ability to collaborate (swim lanes)
- Logic on the server unable to affect other ‘parts’
- UI Coordination requires replicated logic in the presentation
- Updating multiple widgets often in separate requests



DEMO

Common Approaches to AJAX

So What's Wrong with this Picture?

Adding rhinestone-covered hurdles to your project

- Frameworks still unable to coordinate UI for the developer
- Buisness Logic leaks into the presentation tier
- Introduces even more specialization between actions/view– for each procedure/context
- JavaScript Remoting/RPC can be a Security Risk
- Can we learn anything from 5 years of SOA?

A Lesson From SOA

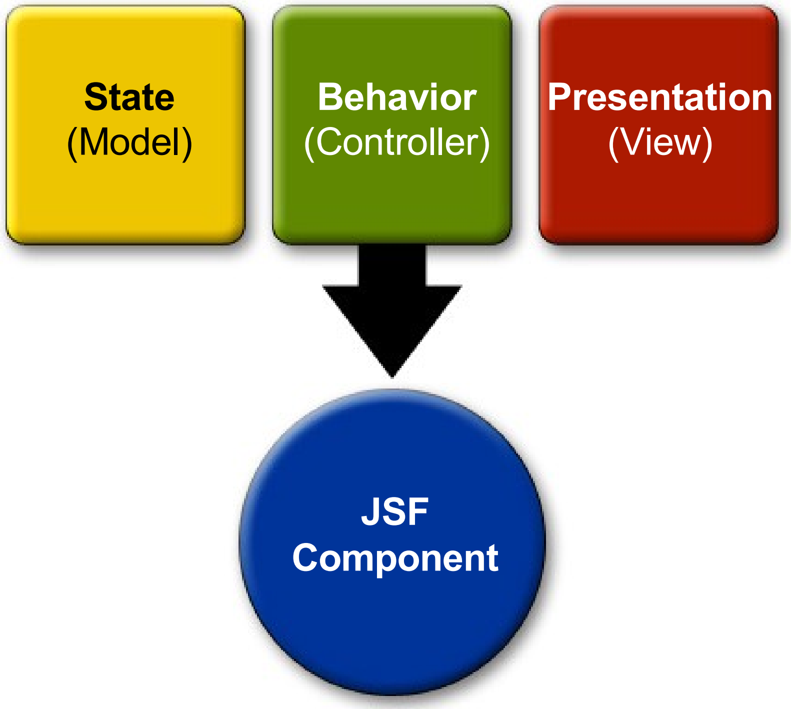
A Public Service Announcement

- XML/etc is horrible at transporting Object graphs
  - Relational Data vs. Business Behaviour
- Automating Object marshalling over AJAX is often wasteful
  - Where and what to send in the response?
- Breaks encapsulation
- Forces knowledge of API's and models into things like JavaScript technology over RPC
- Still have to update the UI
- Same issues as with coordinating in Model 2 MVC

Managing Application Complexity with JavaServer Faces Based Components

Components over the Web?

- Participant in the full MVC Lifecycle
- Vertical, self-sustaining participant
- Encapsulates Programmer Concerns
- Easily Re-Usable
- Stateful when necessary, stateless when not
- Accommodating 'me too' features



JavaServer Faces and AJAX: Natural Evolution

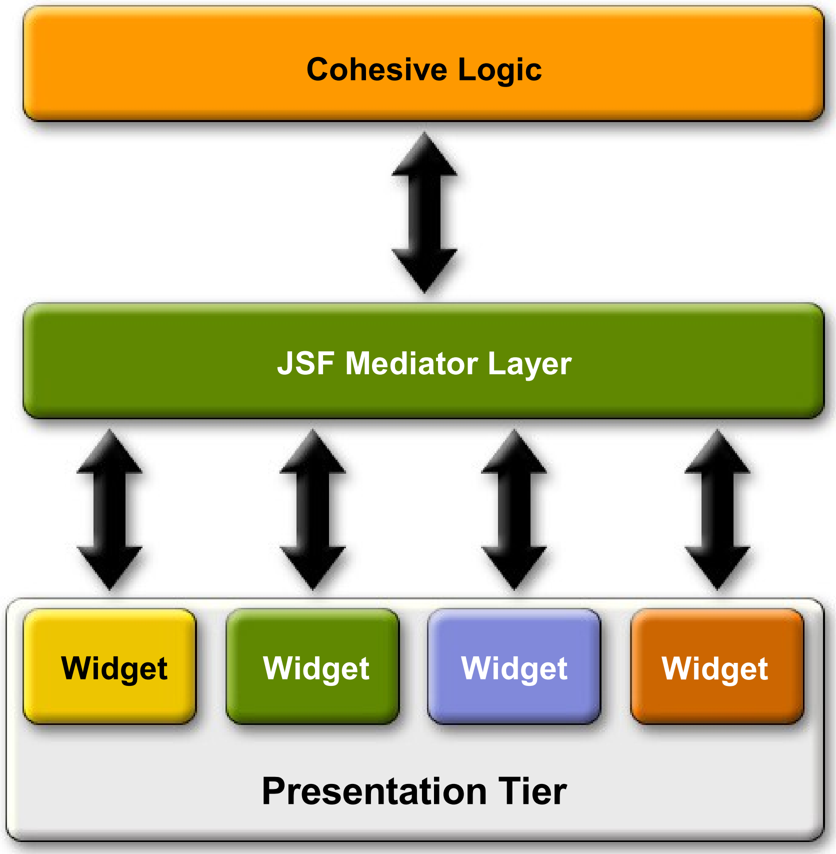
Looking at the Big Picture of UI Development

- JavaServer Faces technology was already able to handle interaction with multiple concerns/actions before AJAX*
- Components encapsulate both the client and server aspects of rich features
- Can render any part of a view—no intermediary logic on your part
- Iterative scope handling with proper context
- Mediates both Client and Java invocation

A Mediated Approach with JavaServer Faces

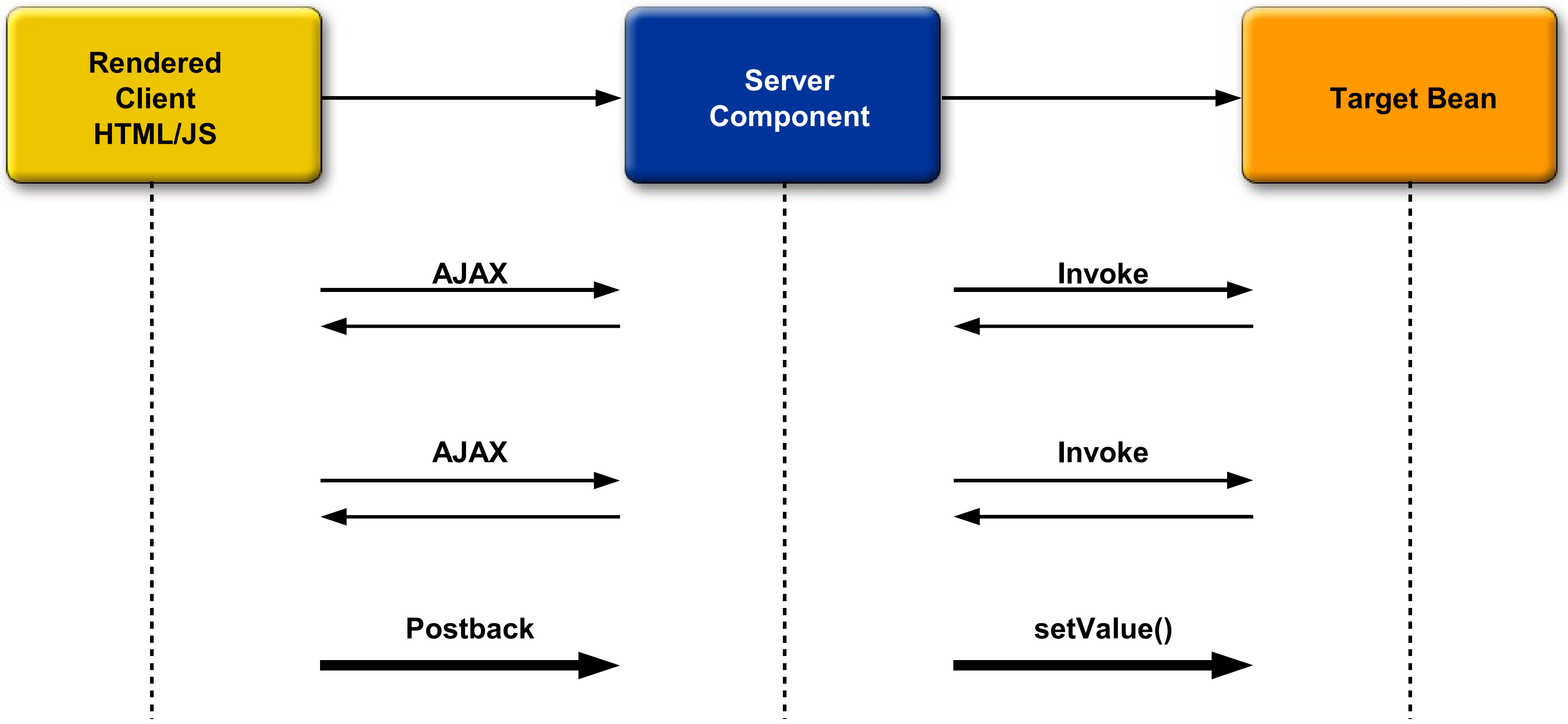
Actual collaboration over AJAX

- A single widget can affect multiple parts at once
- Business logic can update multiple parts at once
- Logic can stay in Java
- A single AJAX request over the network can produce multiple updates



Components as Mediators

<x:suggest value="#{foo.bar}" from="#{util.suggest}"/>



Who Should Be Responsible?

Mediating client and server communications

Server-side Components	Client-side Remoting
Knows/Prunes data it needs for the client representation	Pushes object data to the client via XML or JSON
Ability to do server-side templating of HTML	JS-DOM manipulation is awkward
Keeps Java and domain logic in one spot	Repetition of business logic in JavaScript
Secure, Java model never directly available to the client	Invoking methods on Java Objects from client
Burden of shipping/restoring client state	Lack of 'automatic' context with the client UI

DEMO

What we're shooting for

Example A: Separate Servlet/Service

Jumping out of the Framework

```
<h:dataTable id="tbl" value="#{products}" var="prd">
    <bp:ajax value="#{form.prop}" service="foo.jsp"/>
</h:dataTable>
```

- Deployment issues
- Zero collaboration between components
- Additional development as with Model 2
- Forces knowledge of transport details

Example B: RPC with JavaServer Faces

15 yard penalty for illegal use of EL

```
<h:dataTable id="tbl" value="#{products}" var="prd">
    <bp:ajax value="#{form.prop}" rpc="#{prd.method}"/>
</h:dataTable>


http://localhost/faces/remote/prd/method.faces
```

- Inability to collaborate at the component level
- Only works with global variables
- Mass confusion with developers

Example C: findComponent(...)

What’s a client identifier?

```
<h:dataTable id=“tbl” value=“#{products}” var=“prd”>
    <bp:ajax value=“#{form.prop}” method=“#{prd.method}”/>
</h:dataTable>


UIAjax c = viewRoot.findComponent(“tbl:8:ajax”);
```

- Used with PhaseListeners
- Can collaborate within the component model
- Encapsulates transport details
- An instance for what iteration?

JavaServer Faces 1.2 and invokeOnComponent(…)

The sixth dimension of JSF’s lifecycle

```
ContextCallback cc = new ContextCallback() {
    public void invoke(FacesContext f, UIComponent c) {
        ((UIAjax) c).doMethod(f);
    }
};
view.invokeOnComponent(facesContext, “tbl:8:ajax”, cc);
```

- Any time we deal with Collections of data
- Correcting contextual issues across the board

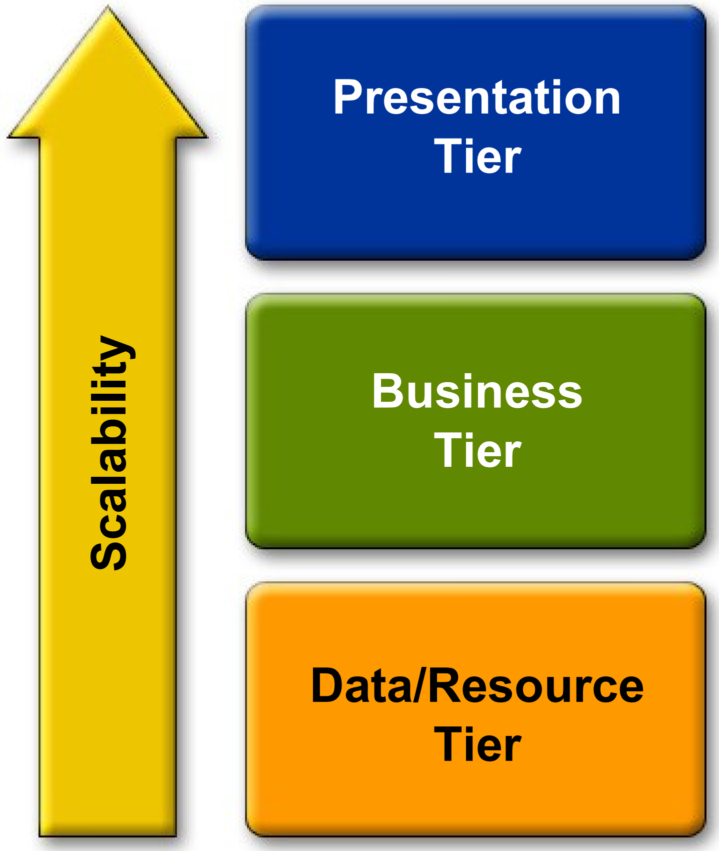
DEMO

Contextual AJAX

The Many Faces of Scalability

All things to all people?

- Usually associated with scalability of hardware
- But at what cost: usually increased development and maintenance time
- Most solvable in the UI Tier
- What did you choose for persistence on your last project?



What’s the Cost?

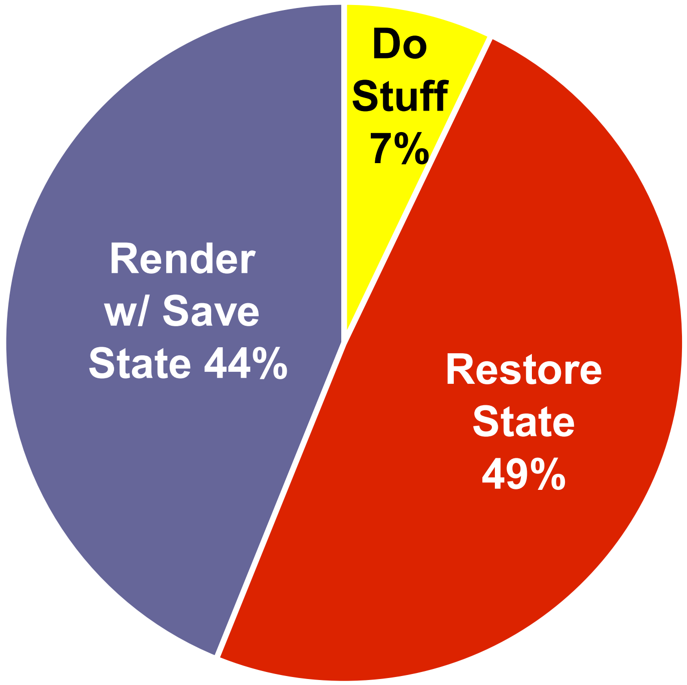
The complexity has to live somewhere

- The cost of JavaServer Faces technology's brand of abstraction: State management
- This cost can be mitigated to the point of irrelevance
- JavaServer Faces technology performance and AJAX
  - Component frameworks make fine-grained processing easier
  - But it has to be fast and scalable
  - ...and we all know that action frameworks are faster and more scalable, right?
  - Not necessarily, let's see how

Breaking Down a JavaServer Faces Platform Request

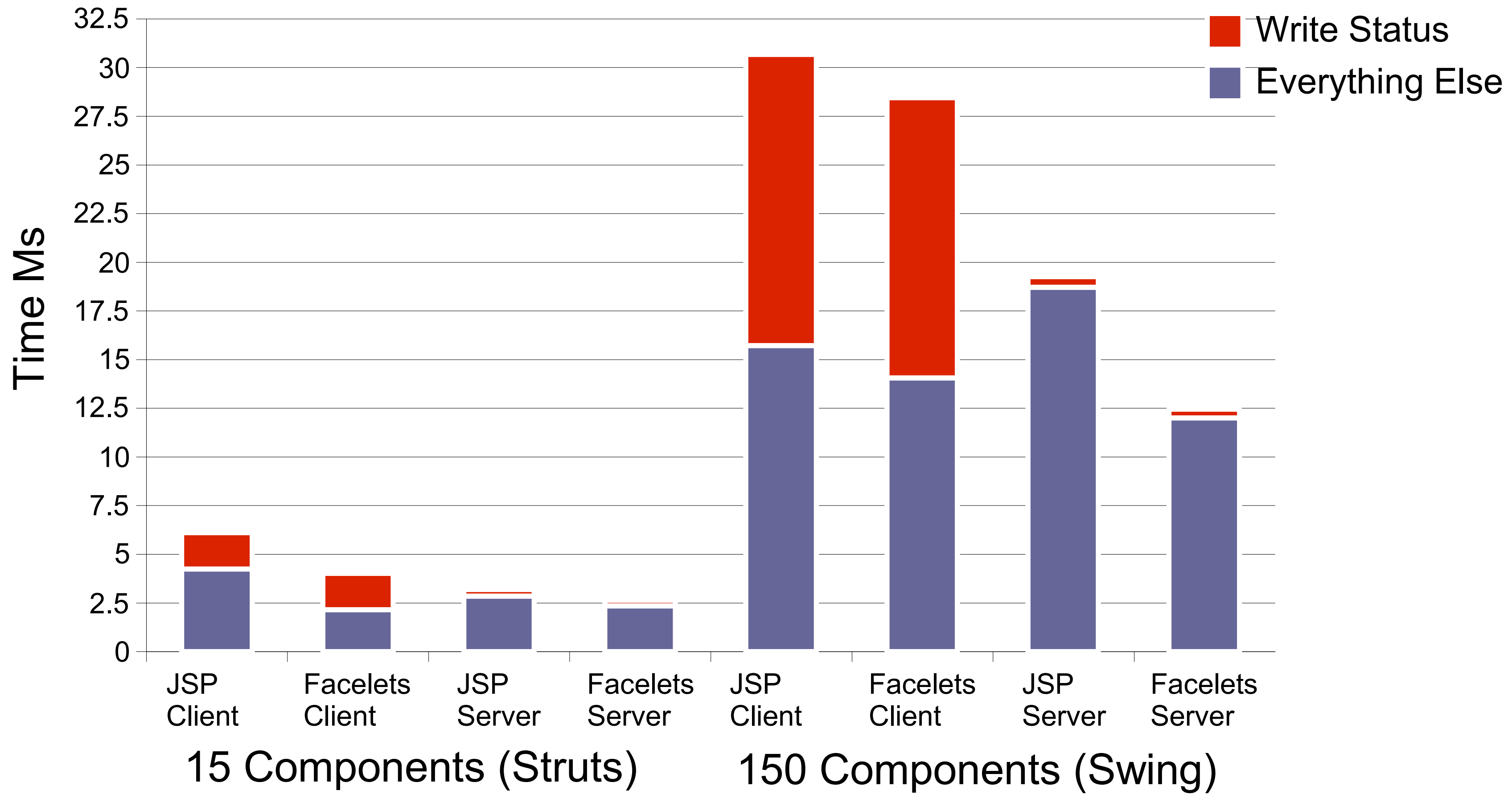
Where does the time go?

- Do “stuff” is very fast
- Rendering is improved
- State Saving is problematic



Types of State Saving in JavaServer Faces Platform

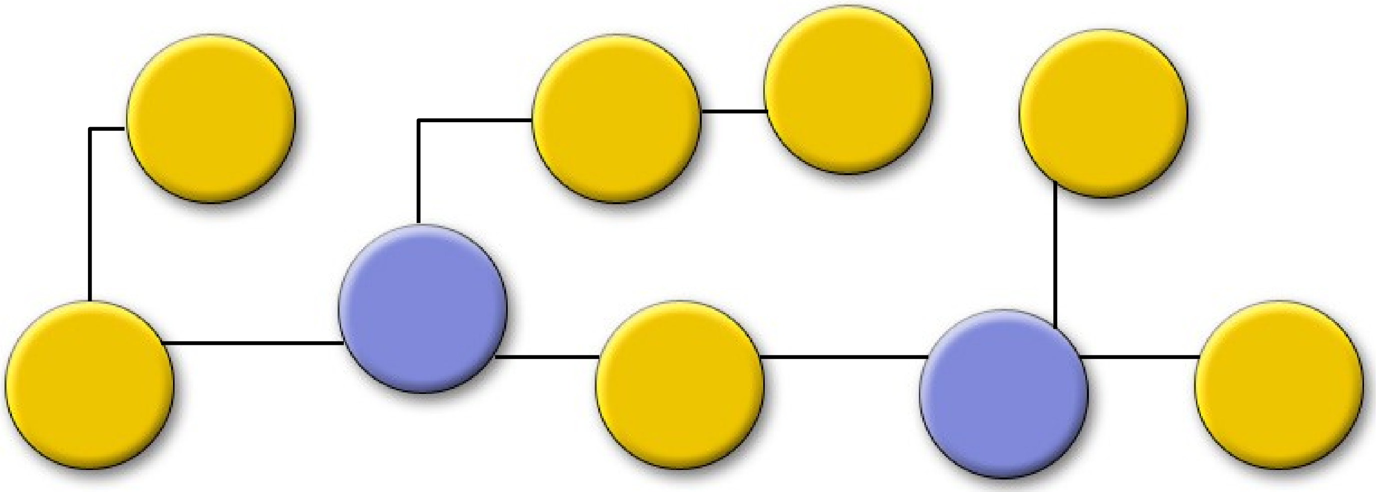
Where does the time go?



State Saving and AJAX

Lets make some ‘changes’

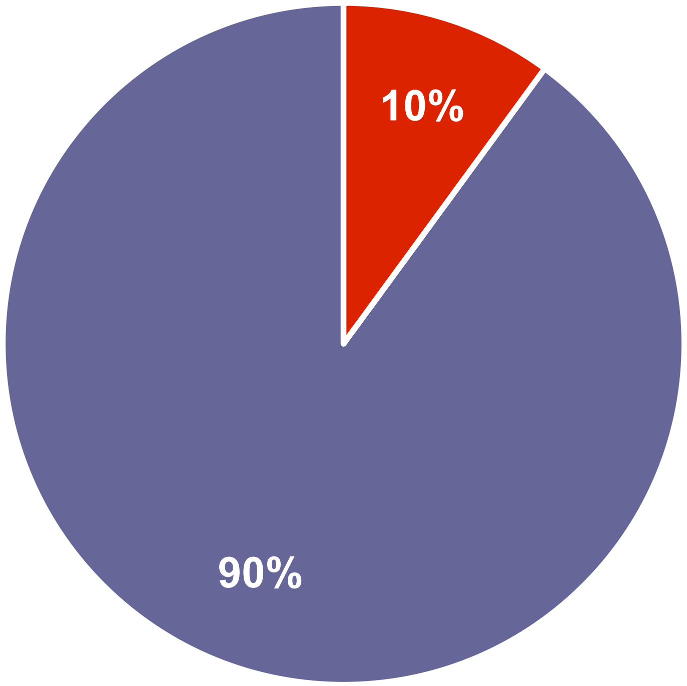
- Going declarative
- Managing shared/transient parts of the view
- Learning from ORM Frameworks



Using State Deltas

Impressive results with ADF, Facelets, and JavaServer Faces 1.2

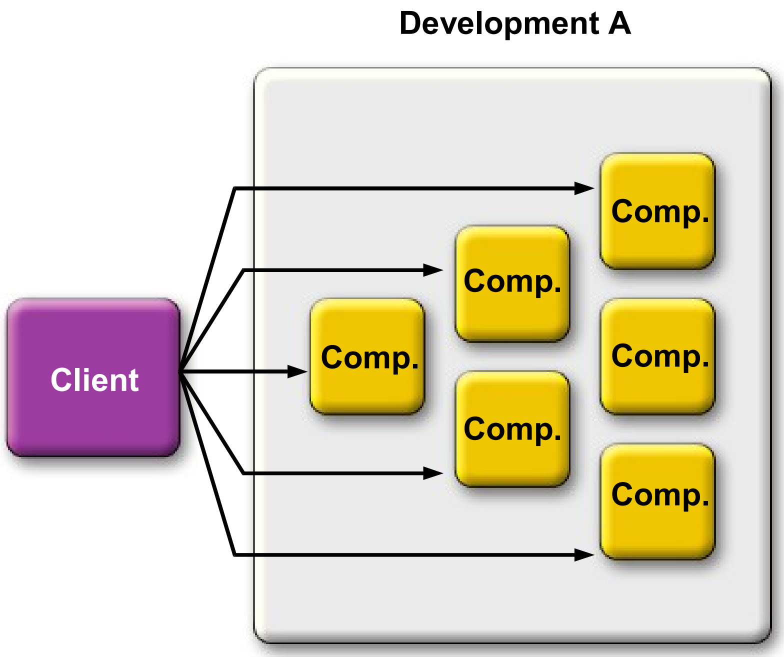
- Utilizing declarative views (Facelets)
- 10% of the size of state required
- Similar reduction in CPU time to store state
- Works with both client and server state saving algorithms



Speeding Up View Restoration

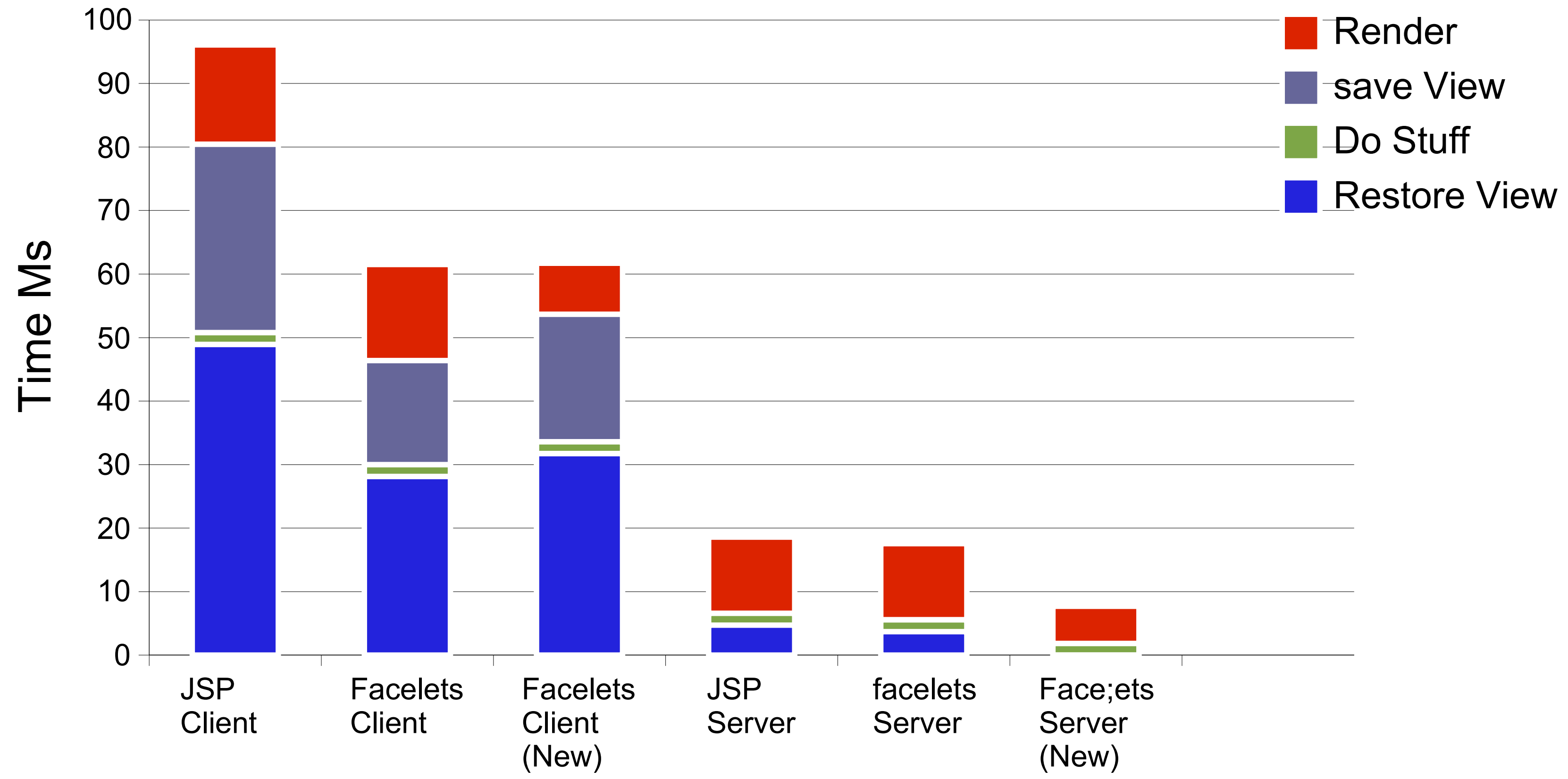
Keeping the channels open

- Just keep reference to the last view requested and its captured state
- Restore view becomes nearly free
- Covers 99.3% of practical AJAX use cases



AJAX with New State Saving

Timings for a 150(!) Component Page



Tips for Performance w/ AJAX

Things you can do now

- JavaServer Pages™ technology and Facelets already optimized for handling pure HTML content
- Avoid JavaServer Pages Standard Tag Library (JSTL) and opt for ‘rendered’ to hide and show components
- h:dataTable, af:iterator, and ui:repeat components are much faster with JavaServer Faces technology
- Don’t be afraid to mark components ‘transient’ to prevent them from being stored

Reducing the Complexities of AJAX

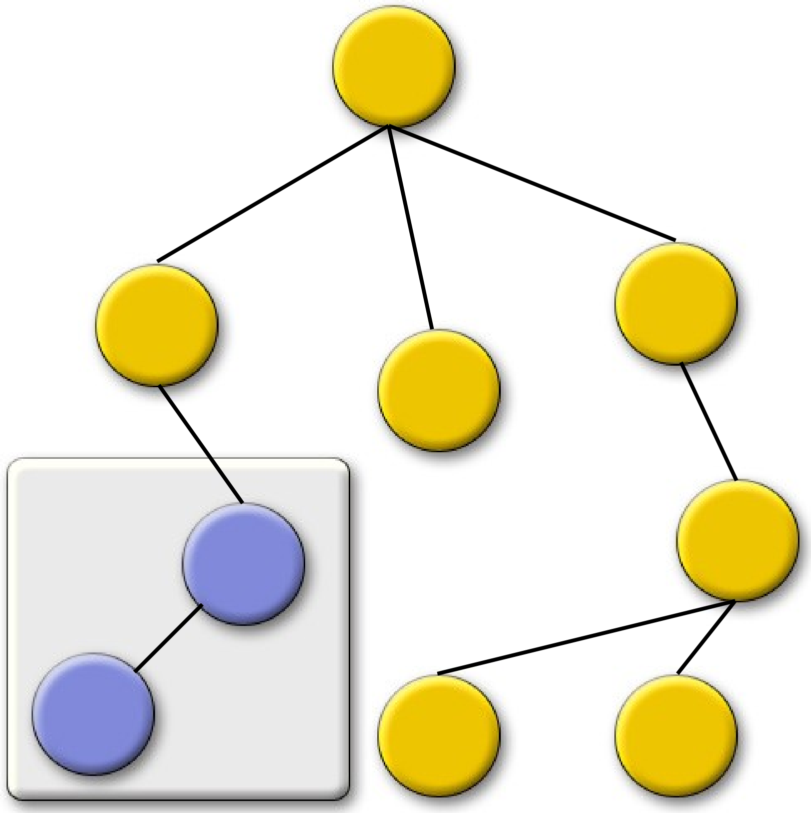
Enabling AJAX in new ways

- Many flavors of AJAX, even within JavaServer Faces technology
- Different types of widgets that need to work together in the UI, not separately
- Desire to make partial processing as natural as possible
- Prevent the need to specifically write your application ‘for’ different AJAX libraries
  - Natural Method Invocation
- Enable easily upgrading to AJAX when the need arises with existing development completed

Example Implementation

The ingredients of a full-blown AJAX solution for JSF

- A simple JSF wrapper for the AJAX transaction on the client
- A mediator for Lifecycle processing on the server
- A “simple” contract between the client and server



DEMO

Example Implementation

Long Term Prevalence of JavaServer Faces

How applicable is the technology three years from now?

- Components will always exist within architectures
- Encapsulation can handle the details of AJAX, etc
- Renderer/Markup/Deployment Agnostic
- Foundation for contextual communication, extending the concept of a remote URI
- Applicable within RPC/SOA for disconnected architectures such as thin client deployments
- AJAX was only one way that JavaServer Faces design can be utilized

For More Information

- http://javaserverfaces.dev.java.net/
- http://jsf-extensions.dev.java.net/
- http://facelets.dev.java.net/
- http://myfaces.apache.org/
- http://java.sun.com/javaee/

Evolving JavaServer™ Faces Technology: AJAX Done Right

Edward Burns, Sun Microsystems, Inc.

Jacob Hookom, McKesson

Adam Winer, Oracle

TS-1161