Java is a trademark of Sun Microsystems, Inc.

JavaOne℠

Building Next-Gen Web Applications with the Spring 3.0 Web Stack

spring source

Keith Donald and
Jeremy Grelle
SpringSource
Web Products Team

Intro

Battling the rising complexity of web application development with Spring

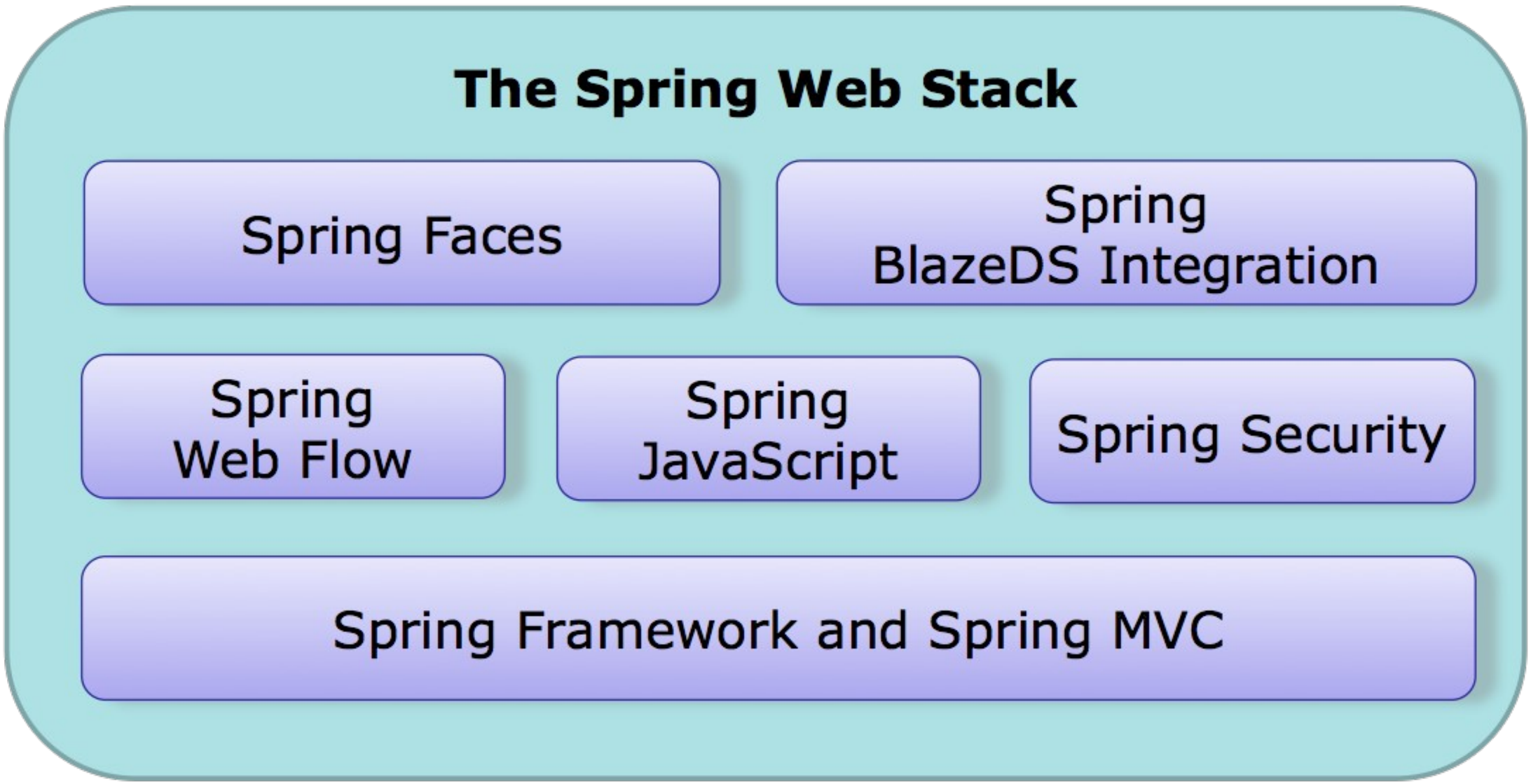
The Modern Web Dilemma

- Web application requirement complexity continues to rise
  - How do we manage this complexity?
  - How do we expose our services to the largest possible audience?
  - How do we give users the best possible experience?

The Spring Web Stack - Battling Complexity

- The Spring Web Stack gives you:
  - unified programming model
  - multiple client types served with the same foundation
  - adaptability - the right approach (i.e., stateless, stateful) for the given use case

What makes the Spring Web Stack?



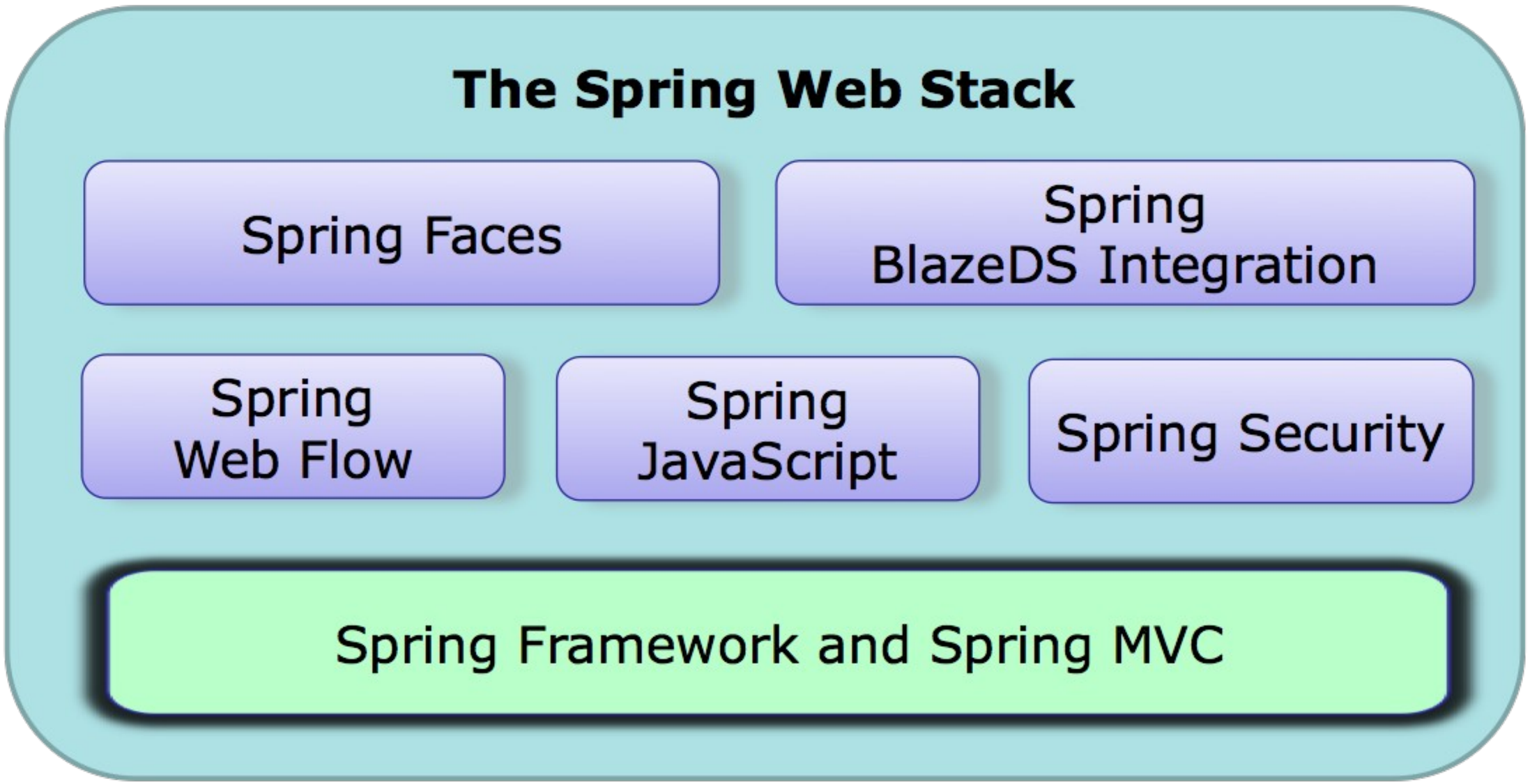
Web Stack Components

- Spring Framework and Spring MVC
  - The foundational web MVC platform
- Spring JavaScript
  - Ajax and JavaScript / JSON integration
- Spring Web Flow
  - Framework for stateful web conversations

Web Stack Components (cont.)

- Spring Security
  - Security framework
- Spring Faces
  - Integration with JavaServer Faces
- Spring BlazeDS Integration
  - Integration with Adobe Flex™ clients

Spring Framework / MVC



RESTful @MVC

- RESTful @MVC is the preferred controller model for Spring MVC 3.0
  - Promotes organization of services around logical resources
    - Java™@Controllers control access to resources
    - @Controllers delegate to views to render resource representations

Key Elements of REST in @MVC

- URI Templates
  - map unique URLs to resources
- Uniform Interface
  - honor the semantics of HTTP verbs
- Content Negotiation
  - provide multiple representations of a resource

URI Templates

- URI templates supported through use of @RequestMapping and @PathVariable

http://springsource.com/travel-app/hotels/1

```
@RequestMapping(“/hotels/{id}”, method = GET)
public Hotel show(@PathVariable Long id) {
    return bookingService.findHotelById(id);
}
```

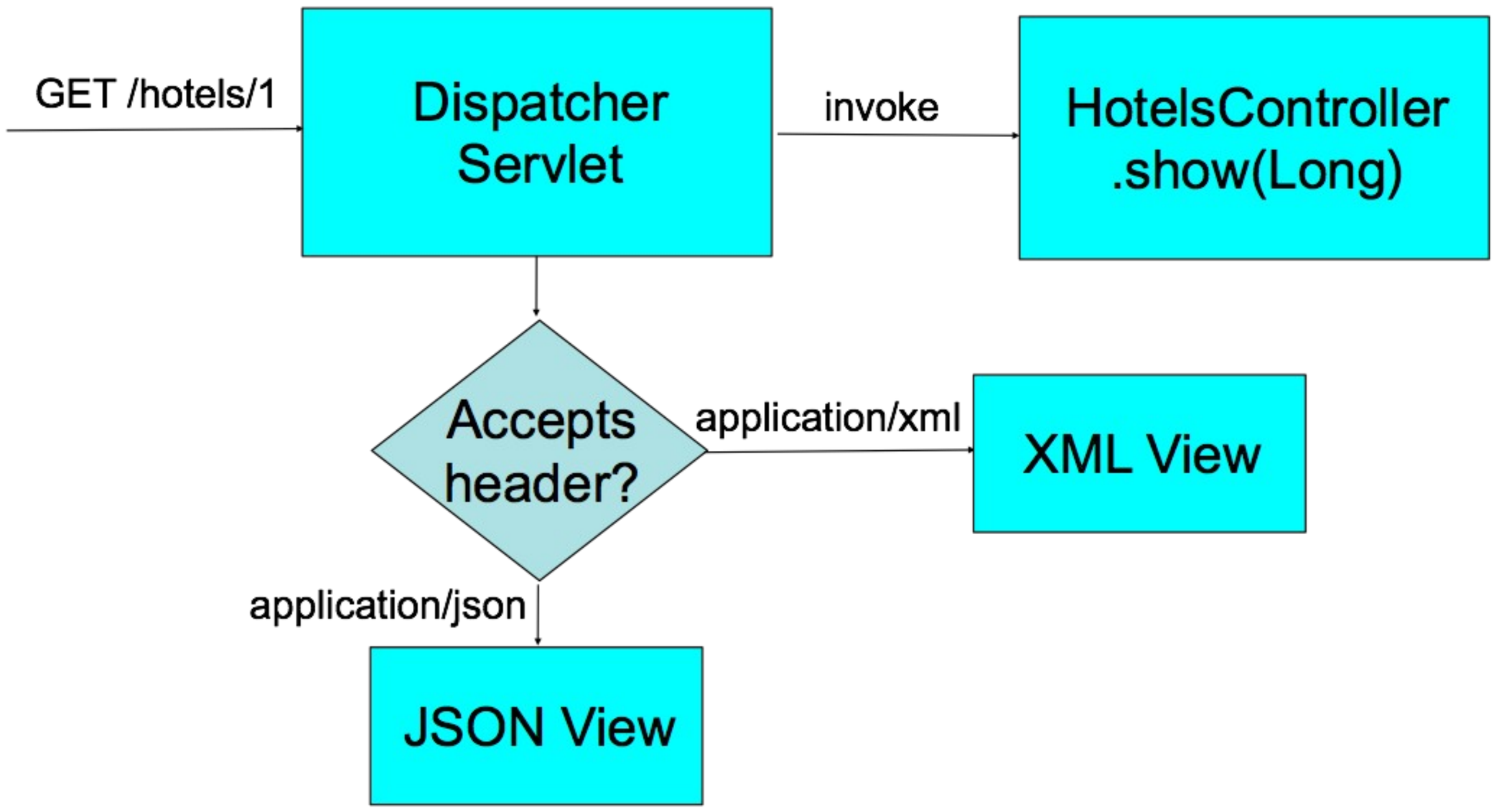
Uniform Interface

- HTTP verbs supported in @RequestMapping
  - GET - retrieves representation of a resource
  - POST - creates a new resource as a child of another
    - Response-Location header returns URI of newly created resource
  - PUT - updates a resource or creates a new resource when the target URI is known
  - DELETE - deletes a resource

Content Negotiation

- Access resource through representations
- More that one representation possible
  - HTML, XML, JSON, PDF, ATOM/RSS, etc
  - Content negotiation figures out the right one for a given request
- Desired representation specified in Accept header or via file extension
- Delivered representation shown in Content-Type

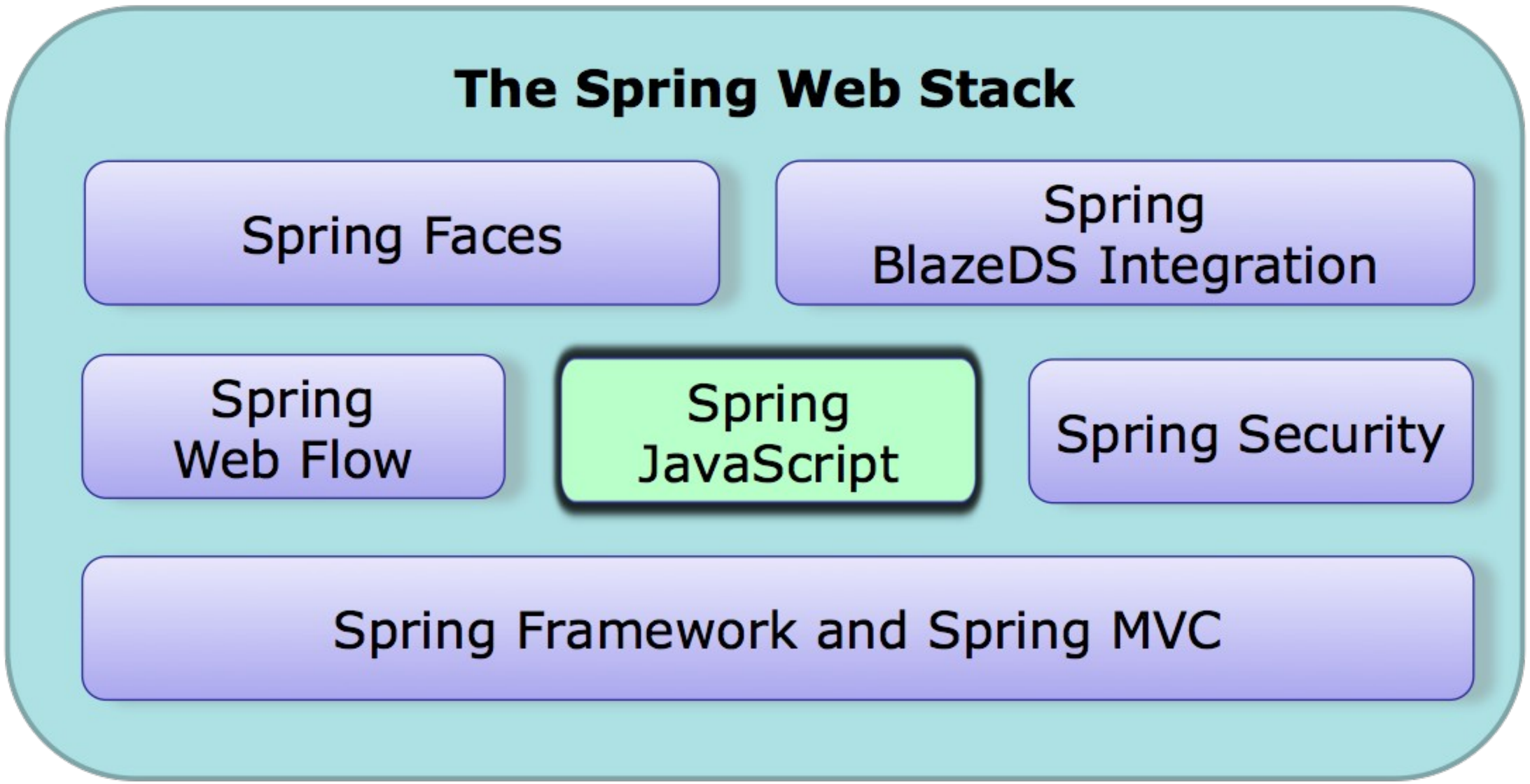
Content Negotiation



Demo

RESTful @MVC

Spring JavaScript



Spring JavaScript

- JavaScript abstraction framework
  - Use to enhance HTML elements with rich behavior
- Integrates the Dojo Toolkit
- Provides JSON support
  - Jackson-based JsonView implementation
- Promoted to top-level project for Spring 3.0

Goals of Spring JavaScript

- Encapsulate use of Dojo for common enterprise use cases
  - Ajax
  - Rich widgets such as a data grid
  - Client-side validation
- Promotes progressive enhancement
  - Robust in the face of JavaScript failure
  - Maximizes potential audience
  - Accessibility

Working with the Spring JavaScript API

- Use API to apply decorations to HTML elements
- Different types of decorations
  - WidgetDecoration
  - AjaxEventDecoration
  - ValidateAllDecoration

Ajax with Partial Rendering

```
<a id="moreResultsLink" href="search?q=$
   {criteria.q}&page=${criteria.page+1}">
  More Results
</a>
<script type="text/javascript">
  Spring.addDecoration(new Spring.AjaxEventDecoration({
    elementId: "moreResultsLink",
    event: "onclick",
    params: {
      fragments: "searchResults"
    }
  }));
</script>
```

Name of tile to re-render on server

No callback function necessary to link in response

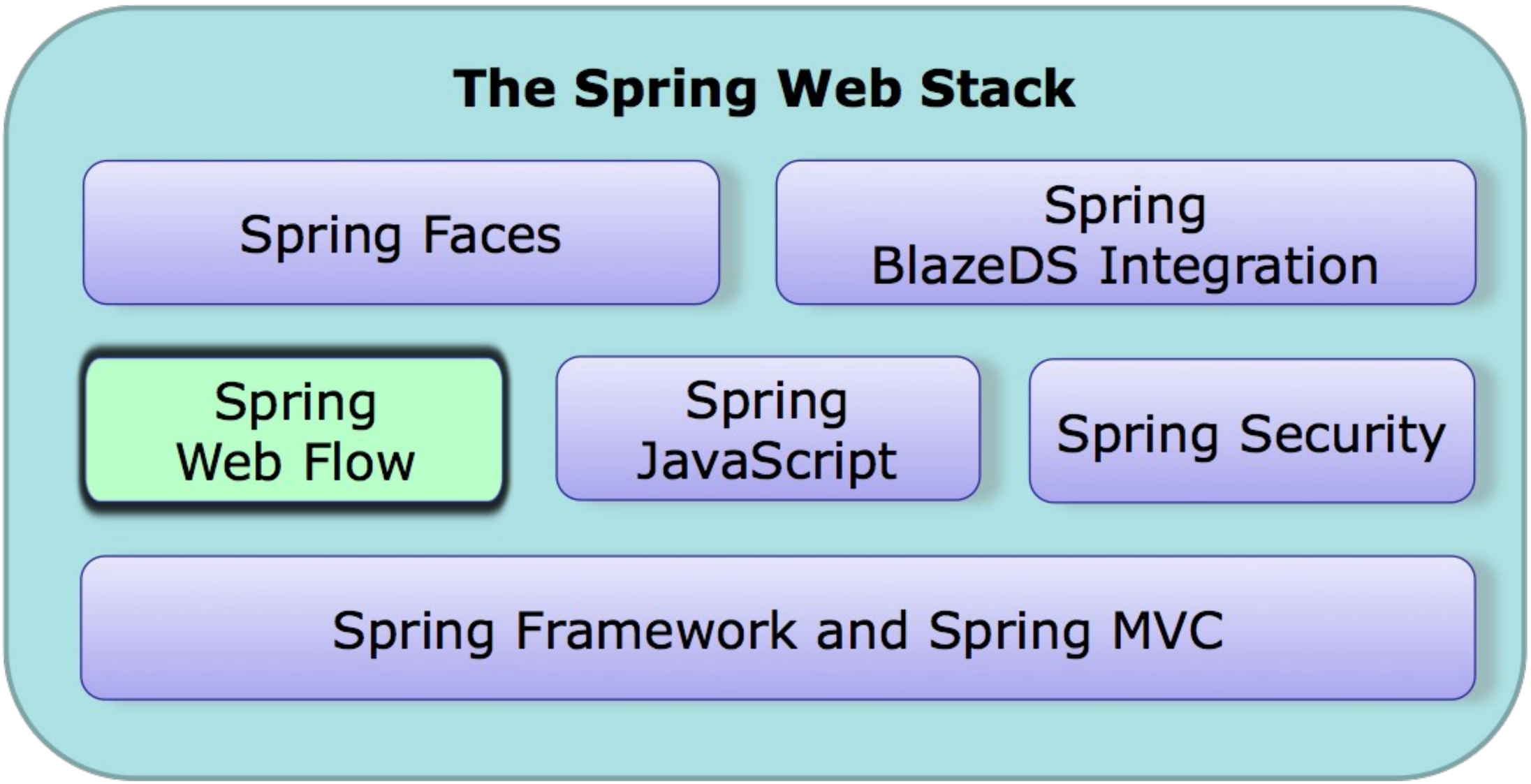
Form Validation

```
<form:input path="creditCard"/>
<script type="text/javascript">
  Spring.addDecoration(new Spring.ElementDecoration({
    elementId : "creditCard",
    widgetType : "dijit.form.ValidationTextBox",
    widgetAttrs : {
      required : true,
      invalidMessage : "A 16-digit number is required.",
      regExp : "[0-9]{16}”
    }
  }));
</script>
```

Demo

Spring JavaScript Client with JSON-based REST interface

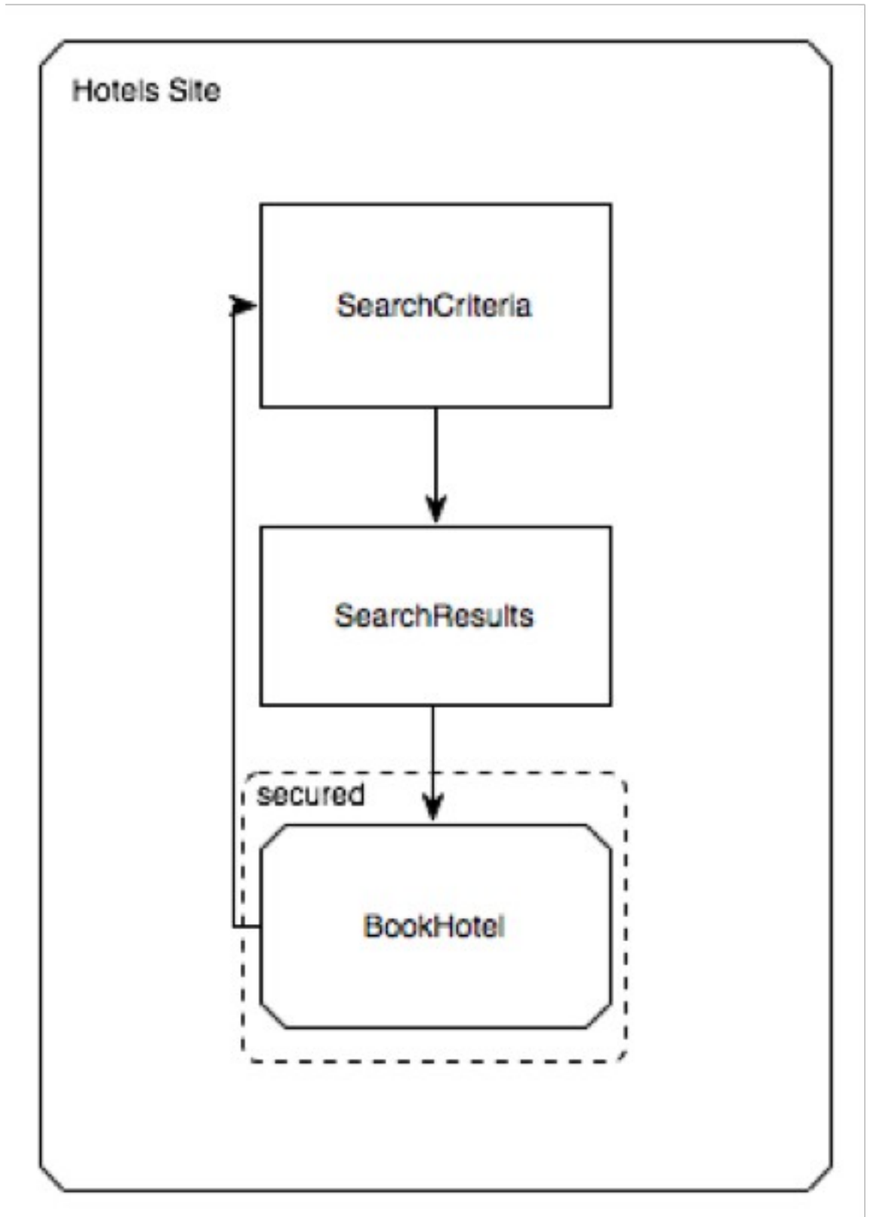
Spring Web Flow

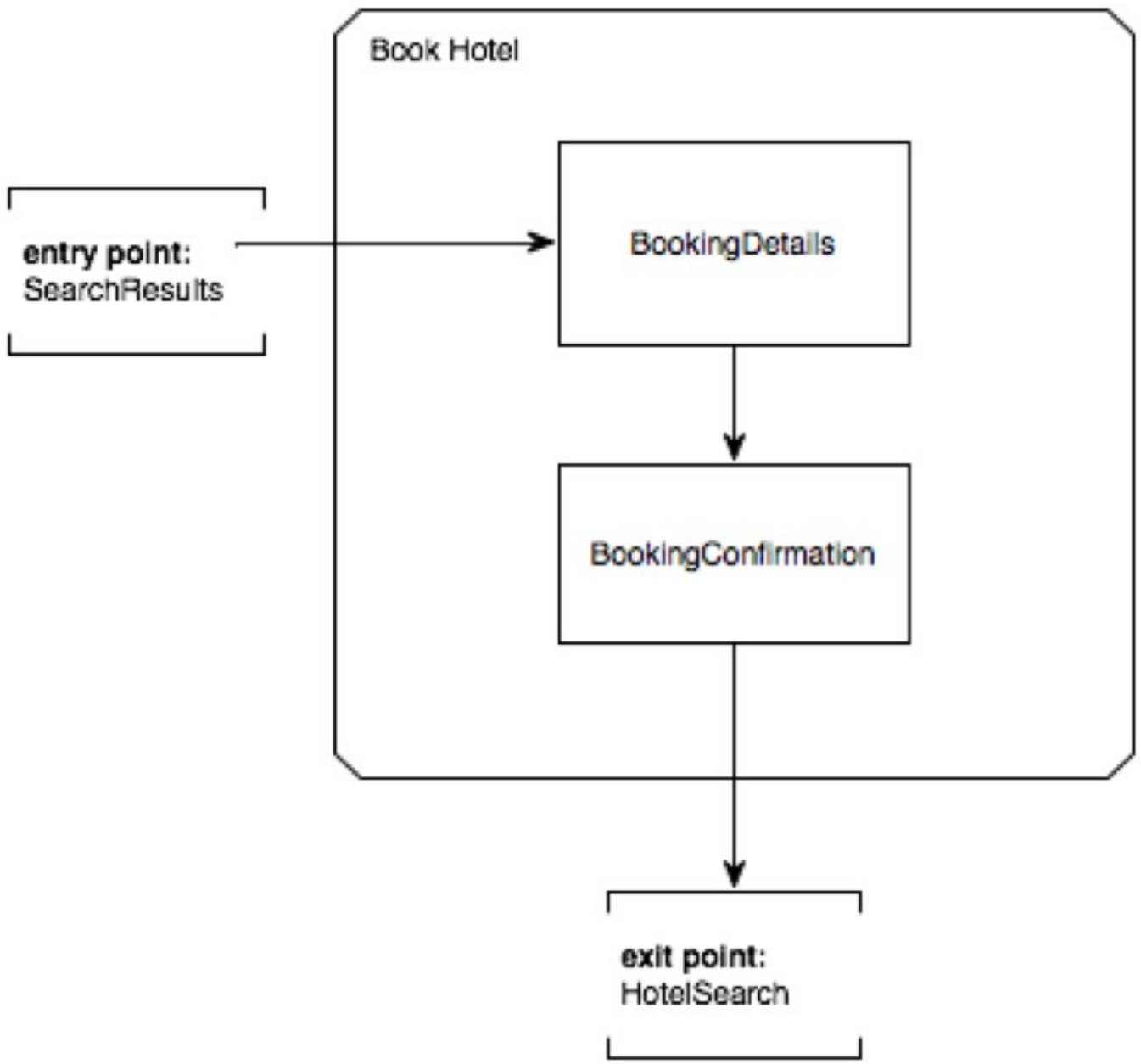


Spring Web Flow

- For implementing stateful flows
  - Reusable multi-step user dialogs
- Plugs into Spring MVC
- Spring Web Flow 2 current release
  - Incorporates lessons learned from 1.0
  - Offers many new features

Web Flow Sweet Spot





New Web Flow 2 Features

- Ajax support
  - Partial page re-rendering in flow DSL
- Spring security integration
- Flow-managed persistence
- Convention-over-configuration
  - View rendering
  - Data binding and validation

Spring Web Flow 3 - @Flow

- Extends @Controller model
- Define stateful UI flow control using plain Java™
- Builds on Spring Web Flow infrastructure
- Part of Web Flow project

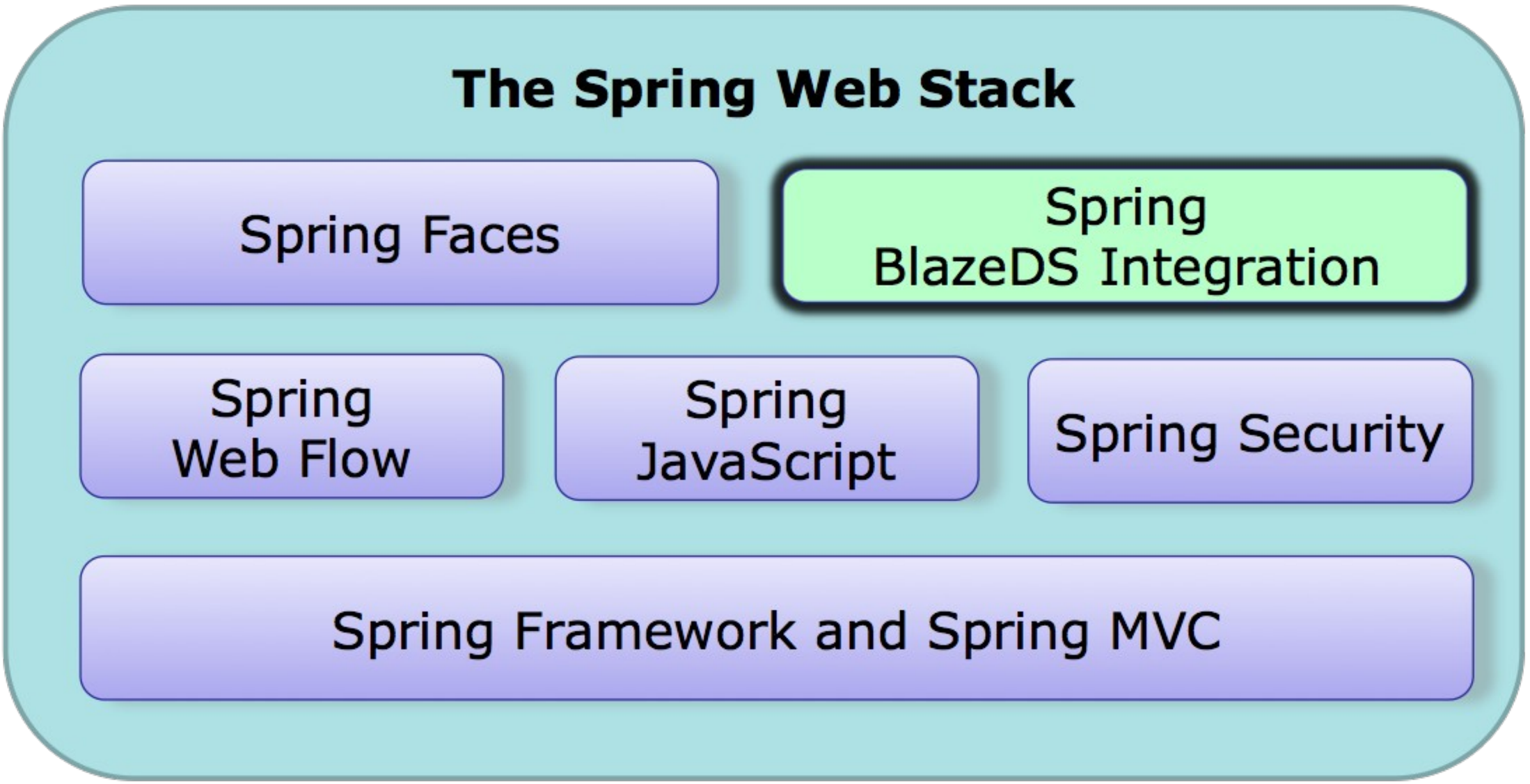
@Flow Example

```
@Flow
public class BookHotel {
   private Booking booking;
   @Autowired private transient BookingService booking;

   public State start(@Input Long hotelId, Principal user) {
      booking = bookingService.createBooking(hotelId, user);
      return new EnterBookingDetails();
   }

   private class EnterBookingDetails extends ViewState {
      @Transition
      State next() { return new ReviewBooking() };
   }

   private class ReviewBooking extends ViewState {}
```

Demo

Integrating Stateful Web Conversations

Spring BlazeDS Integration



Spring BlazeDS Integration

- Spring's Adobe Flex integration project
  - Connects Flex clients to Spring-managed services
    - Using BlazeDS MessageBroker boot-strapped by Spring
- Makes Flex natural fit in a Spring environment
  - Uses Spring XML namespace for simplified setup
  - Reduces the need for BlazeDS-specific config

Spring BlazeDS Integration

- Expose Spring beans as Remoting destinations (for RPC style interaction)
  - <flex:remoting-destination>
- Message-style communication through native BlazeDS messaging, JMS, and Spring Integration channels
  - enables server-side push
- Integrate Spring Security to secure Flex apps
  - <flex:secured>

Demo

Consuming Spring Services with RIA Clients

Summary

- Apply a REST-ful approach to your web application development
  - Think about your app as a set of resources
  - With potentially multiple representations to support different client types
- Spring provides robust web stack for implementing REST-ful web apps
  - A-la carte open-source components
  - Integration with an array of web technologies
- Have fun!

JavaOne℠

Thank You

Keith Donald and
Jeremy Grelle

http://www.springsource.org
http://
www.springsource.com