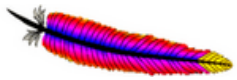
Apache Tuscany: Not the Same Old Architecture

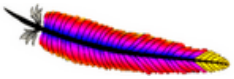
Jim Marino
BEA Systems, *Office of the CTO*

Jeremy Boynes
IBM Corporation, *Gluecode CTO*



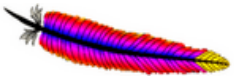
Agenda

- What is Apache Tuscany?
  - The Problem Domain
  - A New Architecture: The Service Network
- Tuscany Technical Overview
- Where We Are Going



Agenda

- What is Apache Tuscany?
  - The Problem Domain
  - A New Architecture: The Service Network
- Tuscany Technical Overview
- Where We Are Going



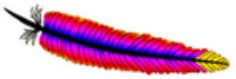
What is Apache Tuscany?

- An open source project in incubation at Apache that provides infrastructure for building service-oriented applications
  - http://incubator.apache.org/tuscany/index.html

- Independent technologies designed to work well together based on
  - Service Component Architecture (SCA) for assembling *service networks*
  - Service Data Objects (SDO) for representing and tracking data as it flows across a service network
  - A Data Access Service (DAS) for declarative data access

- Multi-language
  - SCA and SDO have Java and C++ implementations

- Today we will talk about Java SCA…
  - When we refer to "Tuscany" we really mean "The Apache Tuscany Java SCA project"

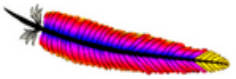


The Problem Domain

- Apache Tuscany is trying to solve the problem of how to *construct* and tie together, or *assemble,* services.

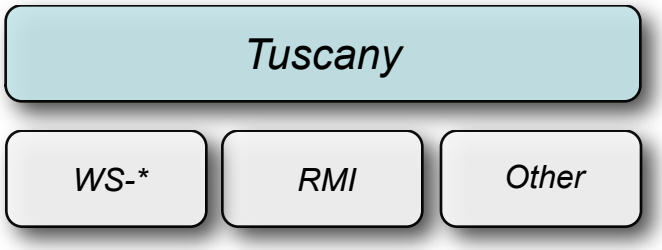
- A service is a unit of code that performs some function
  - May be addressed by a client locally or remotely
  - Offers a contract

- Services may be assembled in a heterogeneous environment
  - May be deployed across different runtimes
    - e.g. J2EE server, J2SE client, Servlet container, OSGi container, etc.
  - May be written in different languages

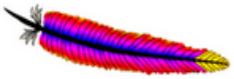


Isn't This Just Web Services?

- ***No…***
  - Web Services define wire-level formats and protocols for interoperability
  - Tuscany operates at a higher level by separating application code from these lower-level concerns
  - Tuscany may use web services technologies but is not bound to them



- Another way to think about this is that while WSDL describes a service contract, it does not do much to describe the relationships between services



For Example…

- The implementation of a simple Java calculator component

```
public class CalculatorImpl implements Calculator{

    private AddService addService;

    public void setAddService(AddService service){
                addService = service;
    }

    public float add(float operand1, float operand2){
                return addService.add(operand1, operand2);
    }
}
```

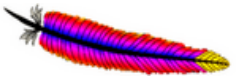
Services are POJOs, contracts can be defined with Java (or WSDL)

Components interact with services provided by other components through contracts and do not need to mess with transport protocols or location APIs. A component is said to have a ***reference*** to another service

- Configuring the service

```
<component name="Calculator">
    <implementation.java class="foo.CalculatorImpl"/>
    <reference name="addService">AddService</reference>
</component>
```

Service references are ***wired*** to targets, typically other services over a ***binding*** *(protocol and transport such as SOAP/HTTP)*. The runtime is responsible for handling the mechanics of this, even potentially selecting the binding.

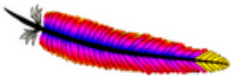


What We are Trying To Do

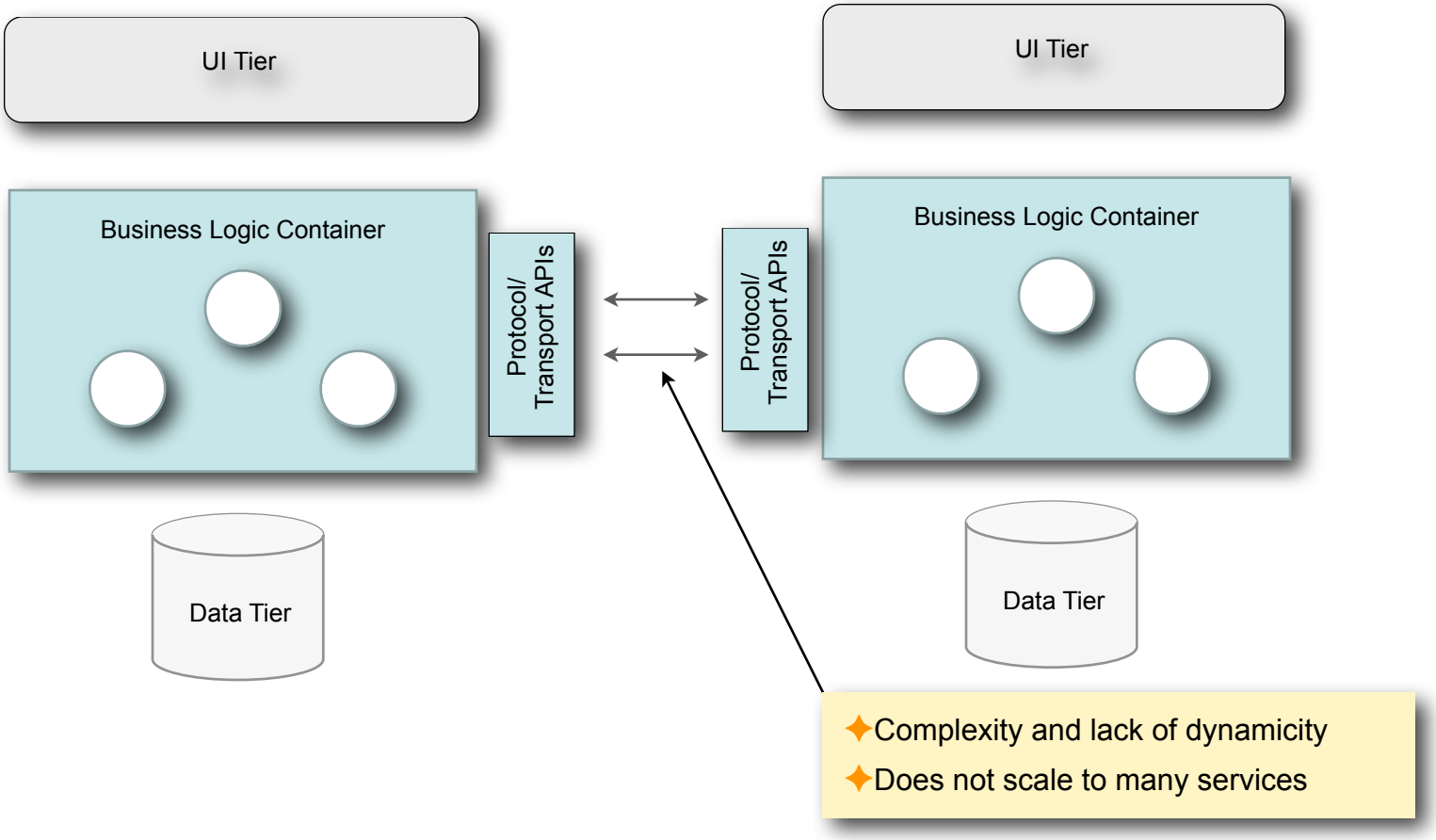
- Provide infrastructure that enables next-generation applications

- Infrastructure today suffers from
  - Complexity in development and configuration, particularly as this scales to SOAs
  - Lack of dynamicity,
    - e.g ability to re-wire target destinations
  - Focus on code reusability but not on *access* to services
  - Difficulty coping with heterogeneity
    - Services are deployed to multiple runtime types in multiple languages
  - Difficulty provisioning through gracefully scaling up *and* scaling down
    - Services may not need or be able to have an entire J2EE environment

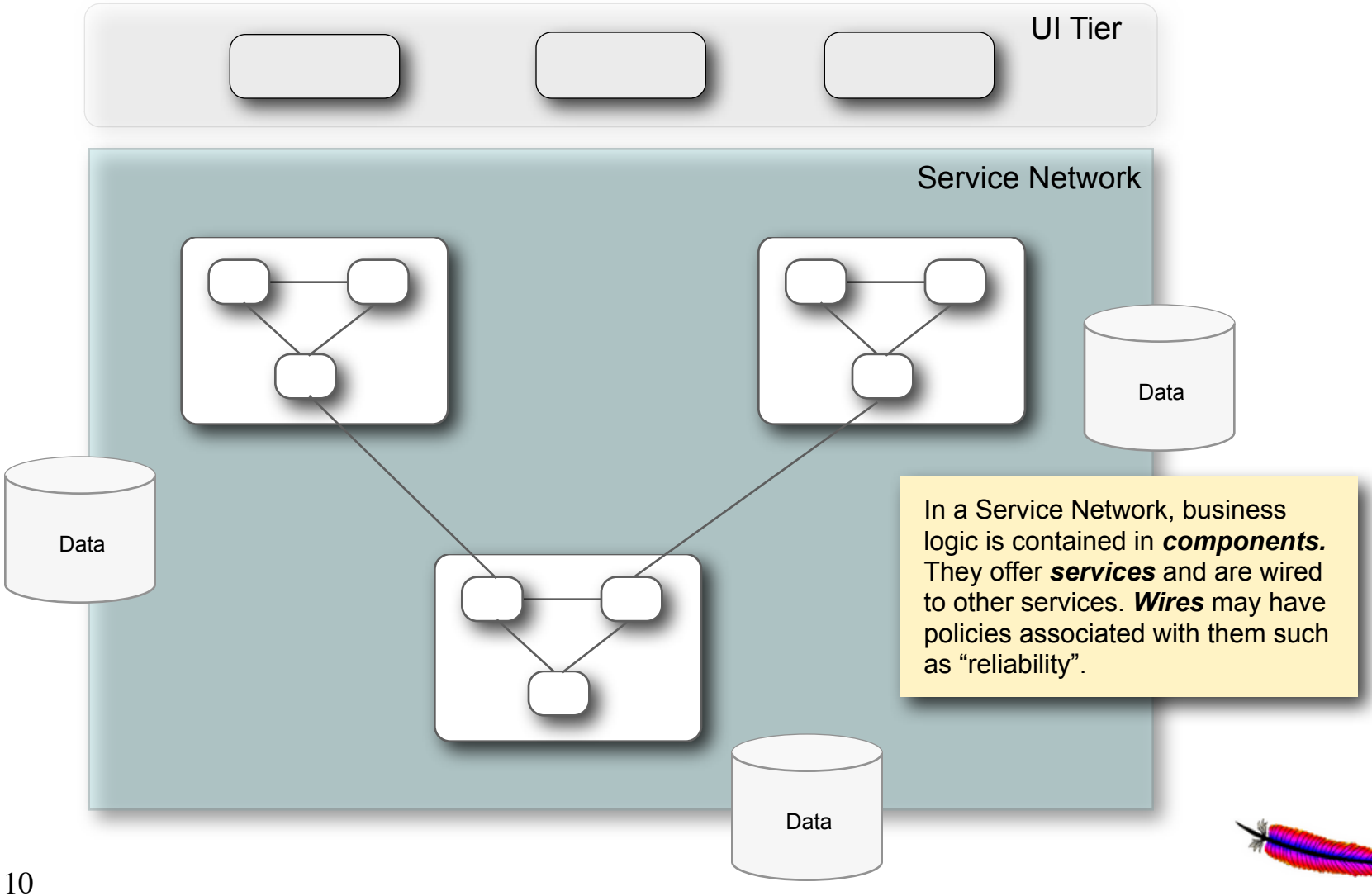
- This requires a new type of architecture…



An Existing (Strawman) Architecture

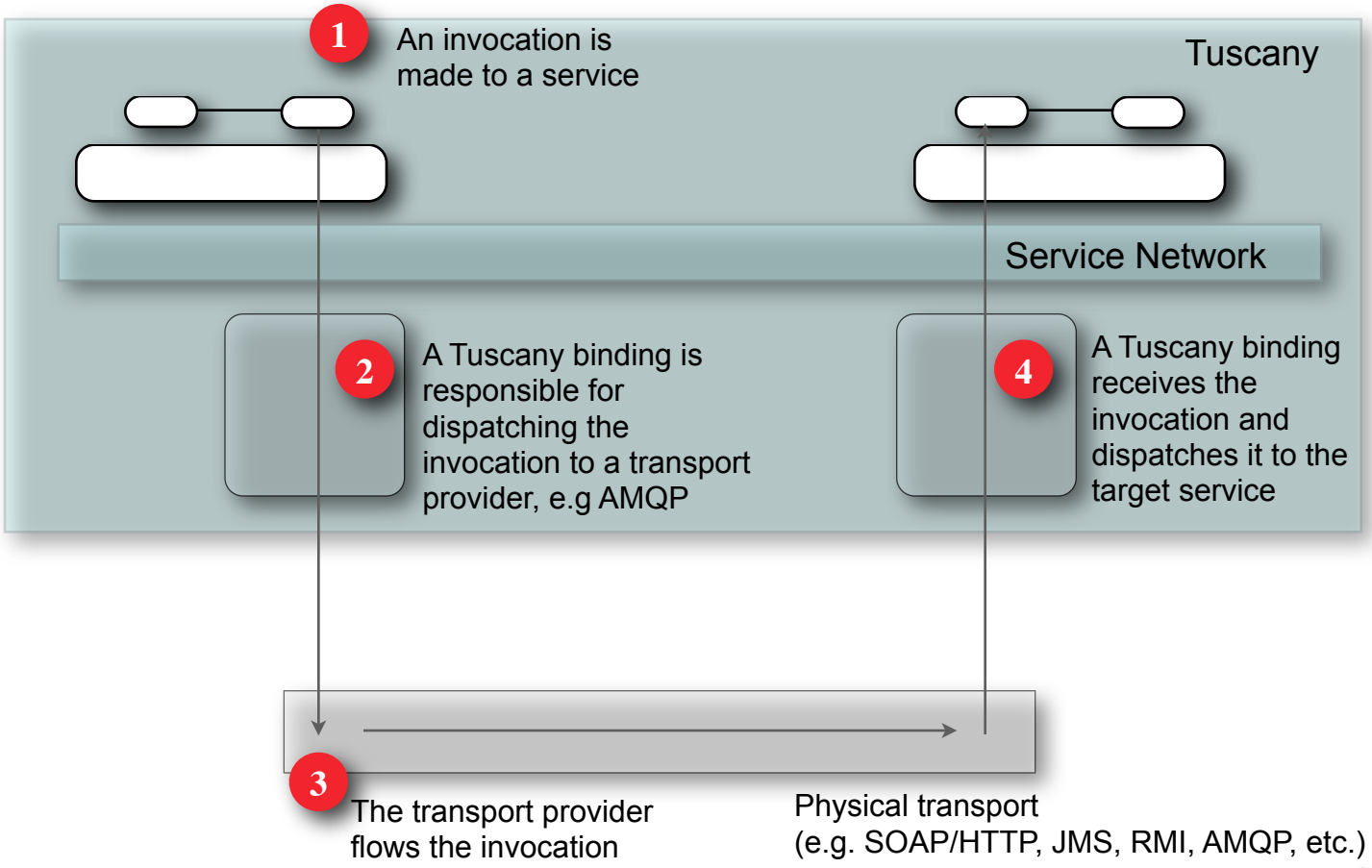


The Service Network



In a Service Network, business logic is contained in ***components.*** They offer ***services*** and are wired to other services. ***Wires*** may have policies associated with them such as “reliability”.

Tuscany and the Service Network



A Simple Example

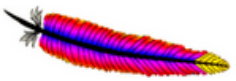
- The client component implementation

```
public class CalculatorImpl implements Calculator{

    private AddService addService;

    public void setAddService(AddService service){
                addService = service;
    }

    public float add(float operand1, float operand2){
                return addService.add(operand1, operand2);
    }
}
```

- Configuring the client component

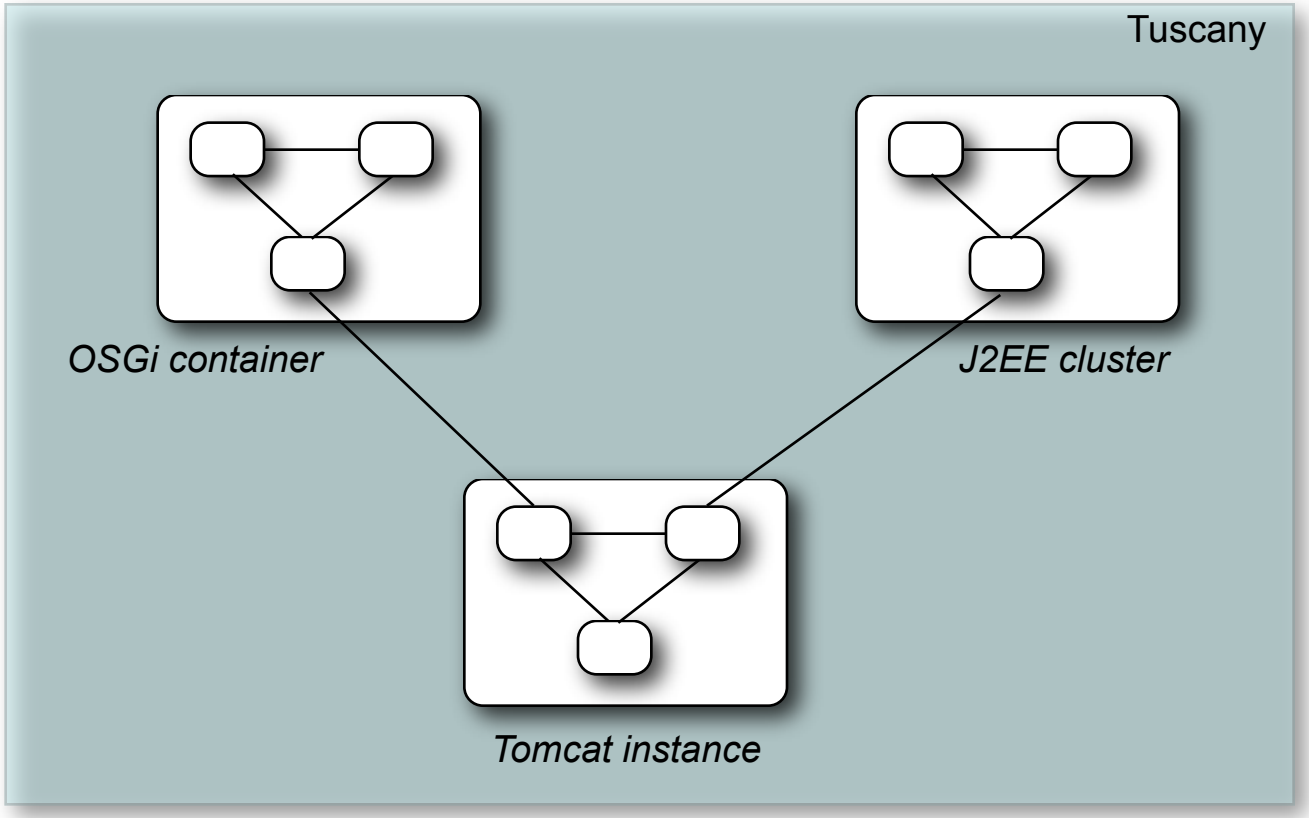
```
<component name="Calculator">
    <implementation.java class="foo.CalculatorImpl"/>
    <reference name="addService">AddService</reference>
</component>

<reference name="AddService>
    <interface.java interface="foo.AddService"/>
    <binding.sca/>
</reference>
```

...that's it!



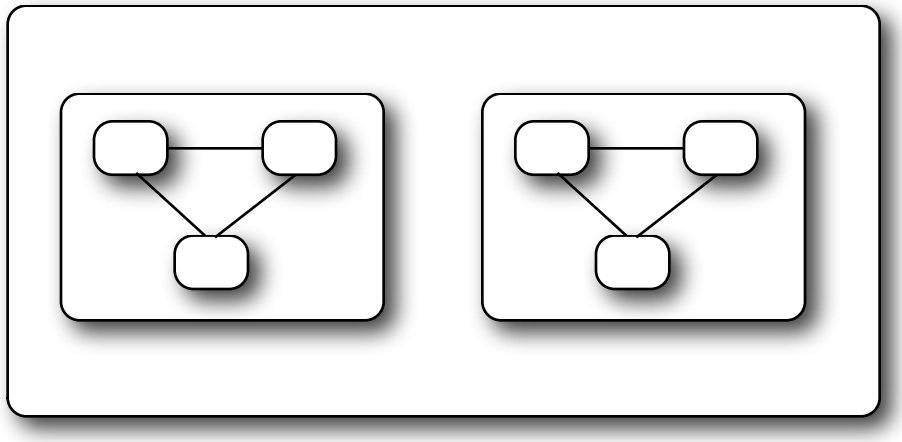
Heterogeneity

- *AddService* from the previous example could be written in a variety of languages and deployed in the network on a variety of runtimes

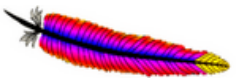


Service Networks are Small and Large

- Large enterprise servers to small-footprint devices
- Services are both local and remote
  - The code (implementation/interface) within a single-VM application is also a service network
- Service networks are *recursive:* components may contain other components

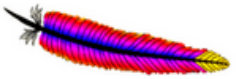


- The service network is itself a component



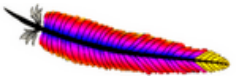
Service Component Architecture

- Tuscany is based on SCA
- SCA provides
  - A way to represent a service network, termed an *assembly*
  - A model for constructing components in a variety of languages including Java, C++ and BPEL
    - Also defines integration with existing programming models including Spring and EJB
  - A model for associating policies with services ....
- Tuscany provides innovation and real-world experience to SCA
- A bit of trivia: there is no such thing as a “Service Component” :-)



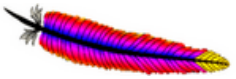
How Is This Different Than J2EE?

- Provides a representation of service networks
- Allows multiple implementation languages
  - e.g. BPEL, PHP, C++, proprietary ESB proxies
- Is better at abstracting communication technology
  - J2EE developer must decide whether creating
    - a web service (@WebService)
    - a stateless EJB (@Stateless)
    - a message driven bean (@MessageDriven)
  - SCA allows someone to decide later
- It can be very lightweight
- However, the two are complimentary in many areas
  - Tuscany can be deployed on J2EE servers
  - EJB and JAX-WS are defined as a component implementation types by SCA (not yet implemented in Tuscany)
  - Tuscany can use a variety of J2EE technologies
    - e.g., Servlets, JSP, JMS, JAXB



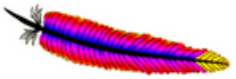
How Is This Different Than Spring?

- Spring focuses on (mostly) local, by-reference wiring
- The Tuscany (SCA) programming model is centered on SOA
  - Non-blocking/asynchronous invocations
  - Service contracts may be bi-directional, i.e. callbacks
  - Conversational state is managed by the runtime
- Tuscany also integrates with Spring
  - Spring can be used to wire local beans that are then wired by Tuscany to remote services
  - A Spring application context can be a component implementation type



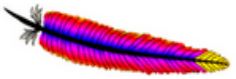
How Is This Different Than JBI?

- JBI standardizes an SPI for middleware suppliers
- Tuscany also has an extension model but is primarily for service developers and service assemblers
- JBI is based on the concept of a Normalized Message Router (NMR)
- Tuscany uses a point-to-point wiring model, similar to a “service switch,” with changeable bindings
- JBI is (currently) single-VM
- Tuscany is (intended to be) multi-VM



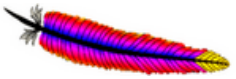
Agenda

- What is Apache Tuscany?
  - The Problem Domain
  - A New Architecture: The Service Network
- Tuscany Technical Overview
- Where We Are Going



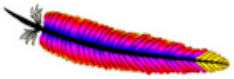
Tuscany is Based On

- A small footprint kernel
  - Less than 4MB including dependencies
  - Will hopefully be smaller
- That is highly extensible
  - Extensions are contributed as SCA components
  - The runtime bootstraps itself as a series of components
- And is deployable to a variety of host environments
  - J2SE
  - J2EE servers
  - Servlet containers
  - OSGi containers
  - Standalone server



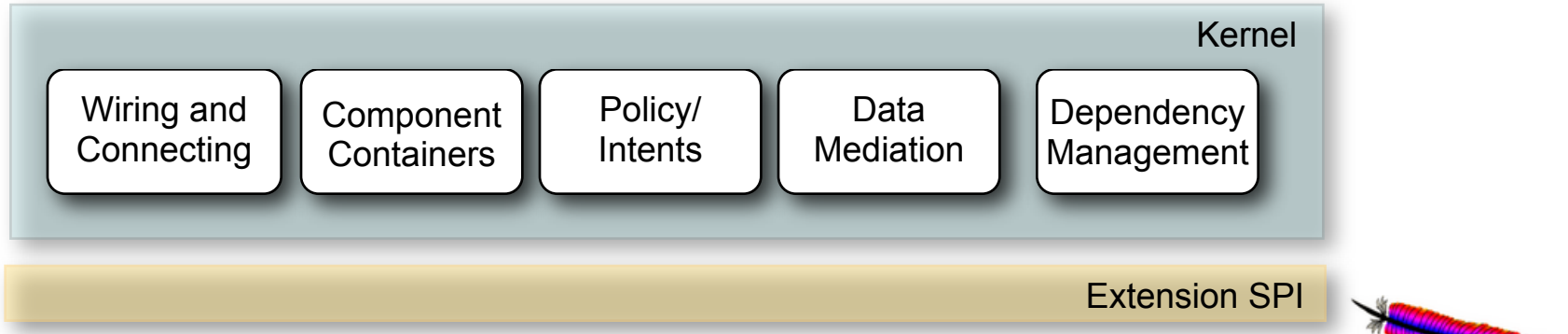
Design Goals

- Make things easy for users
  - Support a non-invasive programming model
  - Provide sensible defaults
  - Automate configuration
- Limit the Kernel to ~30K lines of code
  - Provide functionality through extensions
  - Similar to Eclipse
  - Allow capabilities to be “dynamically” added to the runtime
- Allow maximum choice
  - For example, not tied to a particular databinding or web service technology



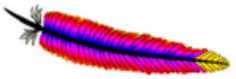
The Tuscany Kernel Consists Of

- A single-VM wiring engine
  - IoC-based (Inversion of Control)
  - Popularized by frameworks such as Spring, ATG Dynamo, and PicoContainer
- Infrastructure for managing components and component state
- A data mediation framework
- A framework for deploying policy and intents
- An extension framework
- Dependency management and provisioning



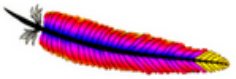
Tuscany Extensions

- Extensions are just components
  - They therefore can be virtually anything someone can think of
- There are some common types
  - Bindings
    - Axis, Celtix, JMS, RMI
  - Component Implementation Types
    - Spring, JavaScript, Ruby
  - DataBinding Frameworks
    - SDO, JAXB, XmlBeans, Castor
  - Miscellaneous
    - JPA, OSGi
- We are always looking for more extensions...



Agenda

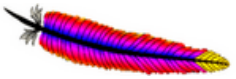
- What is Apache Tuscany?
  - The Problem Domain
  - A New Architecture: The Service Network
- Tuscany Technical Overview
- Where We Are Going



Where We Are Going

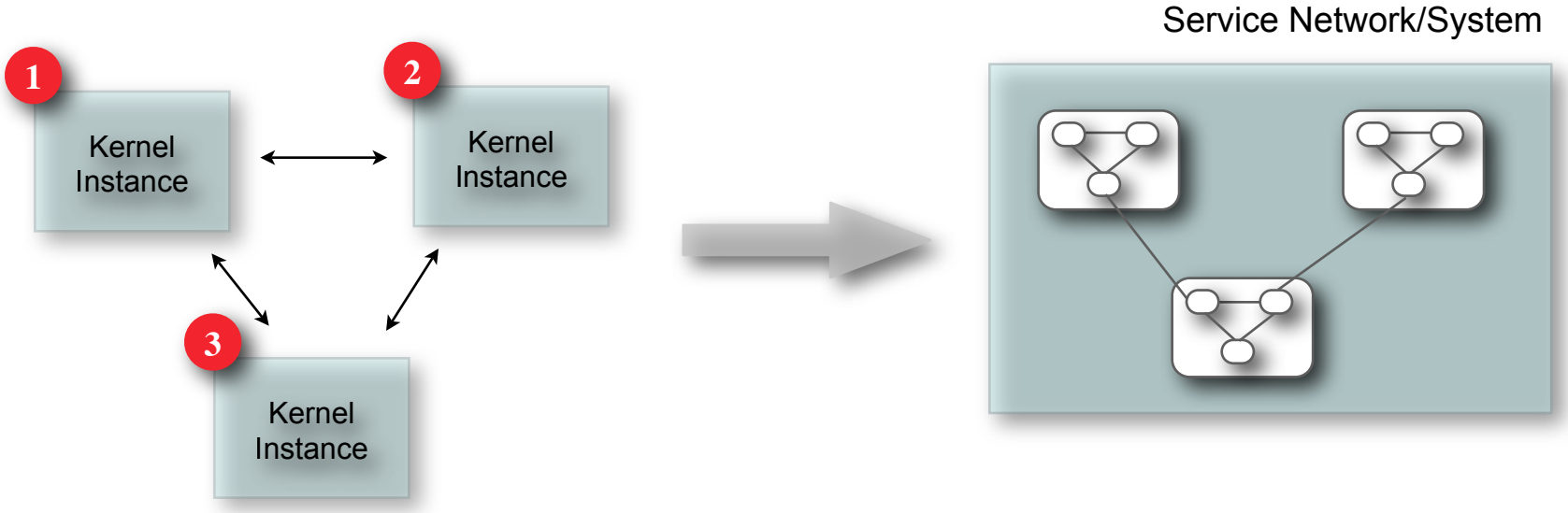
- It’s up to you!

A community of individuals working on things that interest them drives many areas of innovation



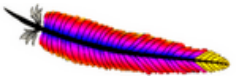
Constituting the Service Network

- Use the Kernel to constitute service networks
- A Service Network is a component referred to as the *SCA System*, or *System* for short
- Individual runtime instances boostrap a *System* through auto-discovery



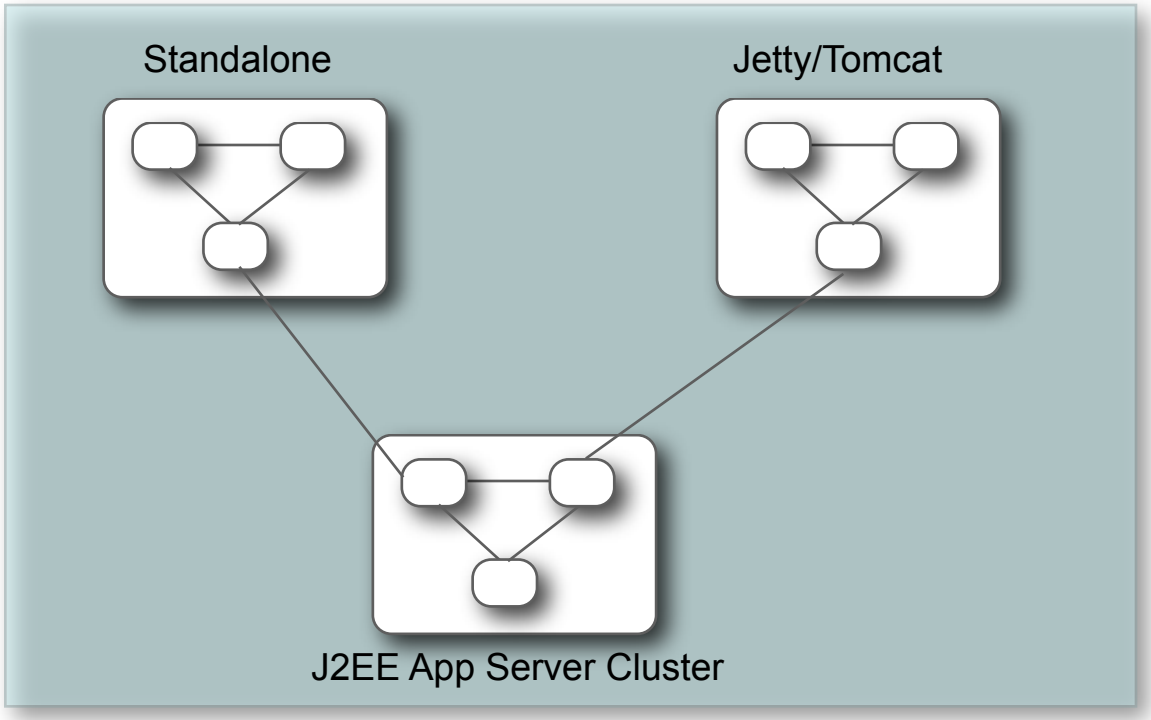
Candidate Technologies for Service Discovery

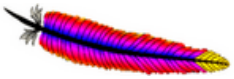
- In the spirit of choice, discovery should be pluggable
- Must scale to 1,000s of nodes
- Zeroconf
  - IETF standard developed by Apple
  - Based on multi-cast and uni-cast DNS
- UPnP
- Jini



Service Networks (Systems) are Federated

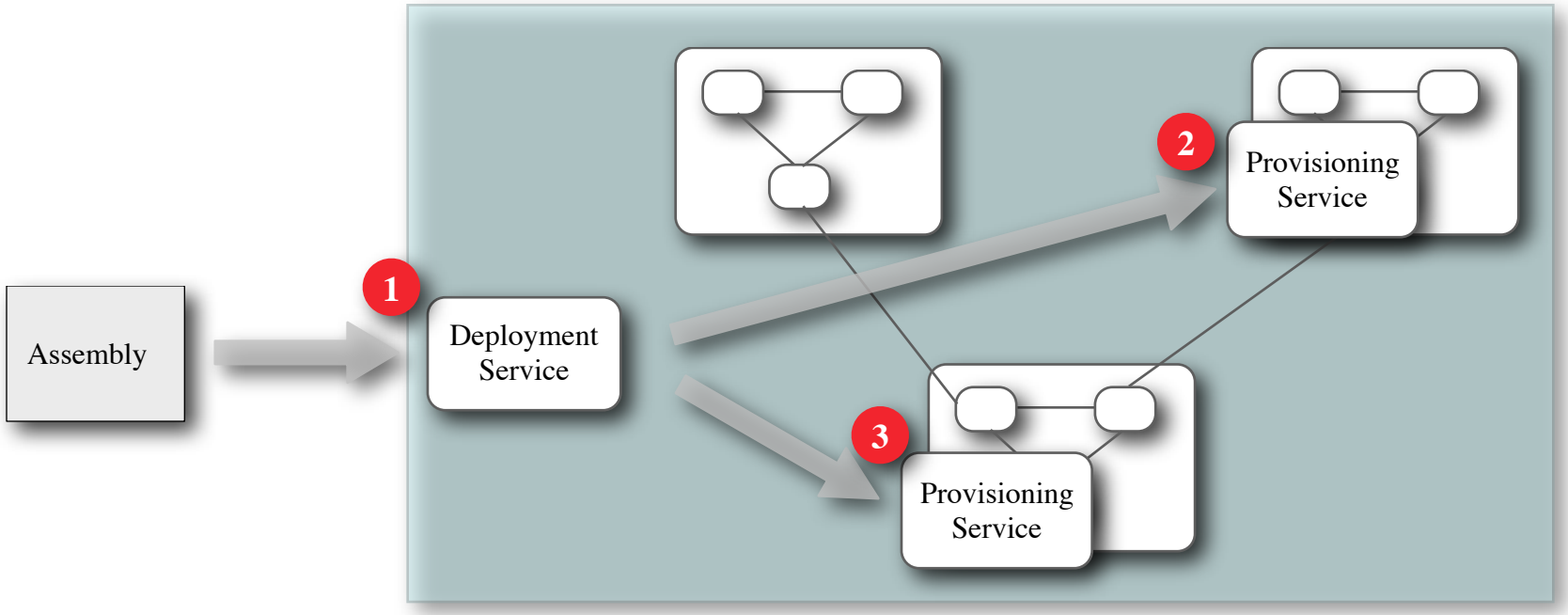
- Tuscany may be deployed to a variety of host environments
- On some nodes, components may be managed by another container and not directly by Tuscany, e.g. an EJB





Deployment and Provisioning in the System

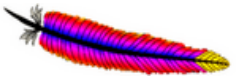
- Tuscany must be able to take an *assembly* (the SCA configuration) and its associated artifacts and deploy them to the System
- The artifacts must be processed and mapped to particular runtimes, where a provisioning agent handles the specific task of installing/ instantiating them



An Analogy

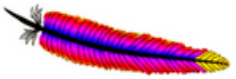
When I use my laptop to browse, I don't care if it is connected over Ethernet cable or wireless. I may want certain qualities of service or security such as HTTPS and I specify that "declaratively" to my browser by typing *https://*. Similarly, a target service (a web site) may require that it be accessed in a secure manner and hence prohibit access using *http://*.

- The physical networking infrastructure is abstracted by a combination of hardware and the OS
- Qualities of service (e.g. security) are specified independently of the physical transport (although the latter may restrict the former at times)
- Things generally just work without the need to know specifics of the physical networking topology



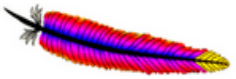
Intent-based Programming

- Tuscany is about abstracting services from the “physical” bindings they communicate over
- Intent-based Programming takes this further by allowing service developers and assemblers to declare requirements to the runtime such as “confidentiality” or “reliability”
  - These intents are mapped down to concrete policies
- Can be applied to (among other things)
  - Services
  - Bindings
  - Provisioning
  - Target/destination selection



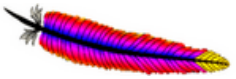
Binding Intents

- Declare a set of requirements on a reference such as “confidentiality” or “ordered delivery” and have Tuscany choose the appropriate binding (e.g. web services, AMQP, JMS) based on the requirements of the client and target
- Decouples services from underlying communications infrastructure
  - Easier to develop and configure
  - Provides runtime dynamicity based on changes in conditions. For example, selecting a different binding as it comes online into a *System*



Target Selection Intents, a.k.a. *Autowiring*

- A client declares certain characteristics about the target of a reference but does not wire directly to it by name
  - e.g. Service contract, SLAs
- The runtime selects the appropriate target and performs the wiring
- Helpful when service networks increase in complexity and number of services
- Provides a further level of decoupling



Thank You and Get Involved!

- We are always looking for contributions
- The best way to get involved is sign up to the the mailing list and start asking questions

http://incubator.apache.org/tuscany/get-involved.html