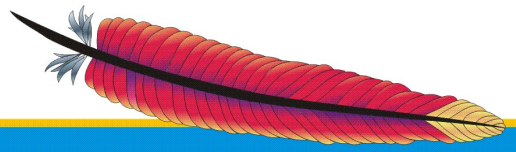
Embrace OSGi ~~Change~~

A Developer's Quickstart

Carsten Ziegeler
cziegeler@apache.org

ApacheCon



About

- Member of the ASF
  - Sling, Felix, Cocoon, Portals, Sanselan, Excalibur, Incubator
  - PMC: Felix, Portals, Cocoon, Incubator, Excalibur (Chair)
- RnD Team at Day Software
- Article/Book Author, Technical Reviewer
- JSR 286 Spec Group (Portlet API 2.0)

Agenda

1 Motivation

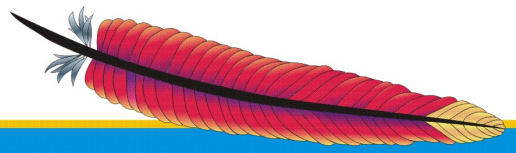
2 And Action...

3 Why OSGi?

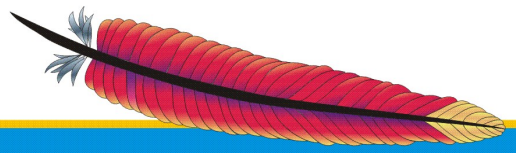
4 Apache Felix

5-7 Bundles, Services, Dynamics

8 Famous Final Words

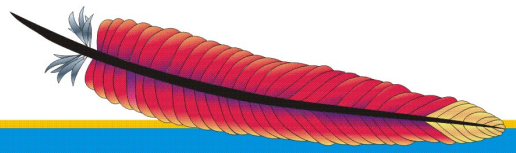


1 Motivation



Motivation

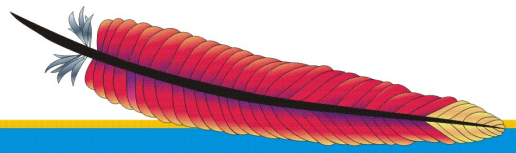
- Modularity is key
  - Manage growing complexity
  - Support dynamic extensibility
- No solution in standard Java
  - OSGi: tried and trusted
- Embrace change – Embrace OSGi
  - Only a few concepts – easy to get started



2 And Action...

Paint Program

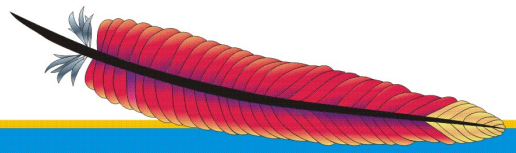
- Swing-based paint program
- Interface `SimpleShape` for drawing
  - Different implementations
  - Each shape has name and icon properties
  - Available shapes are displayed in tool bar
- Select shape and then select location
  - Shapes can be dragged, but not resized
- Support dynamic deployment of shapes



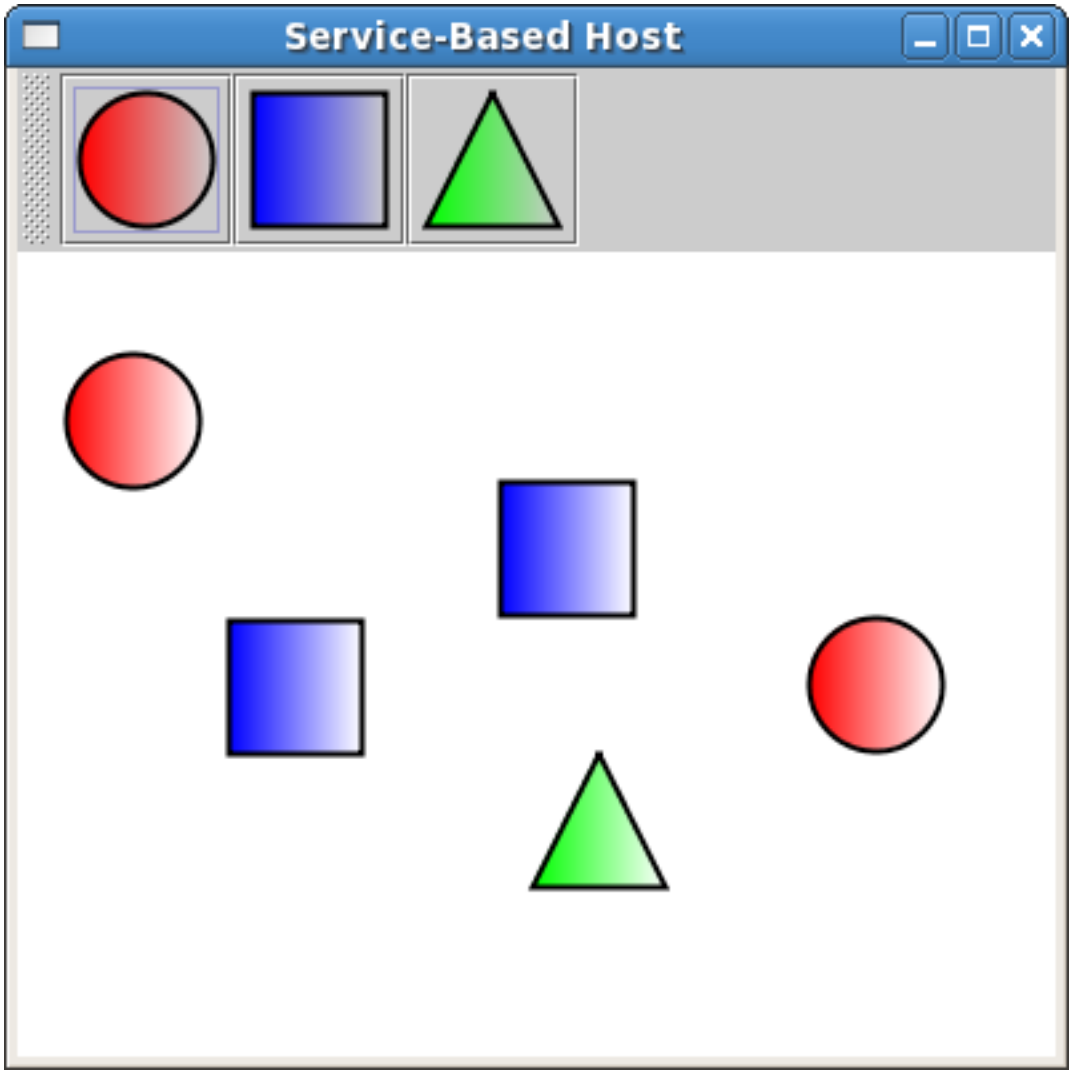
Shape Abstraction

- Conceptual `SimpleShape` interface

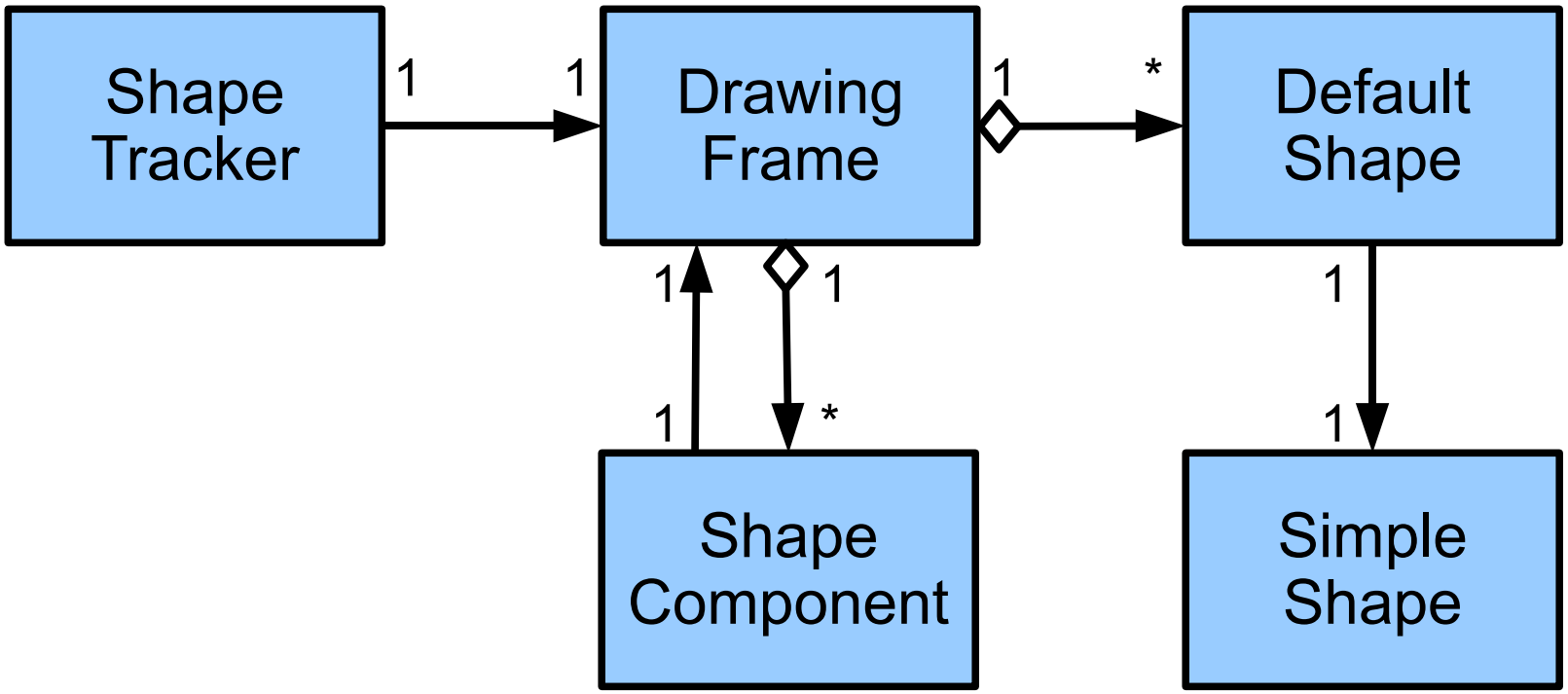
```
public interface SimpleShape
{
    /**
     * Method to draw the shape of the service.
     * @param g2 The graphics object used for
     *           painting.
     * @param p The position to paint the shape.
    **/
    public void draw(Graphics2D g2, Point p);
}
```

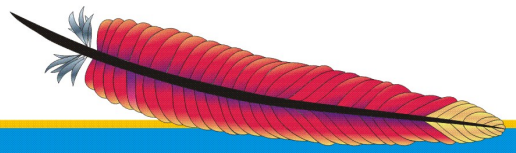


Paint Program Mock Up

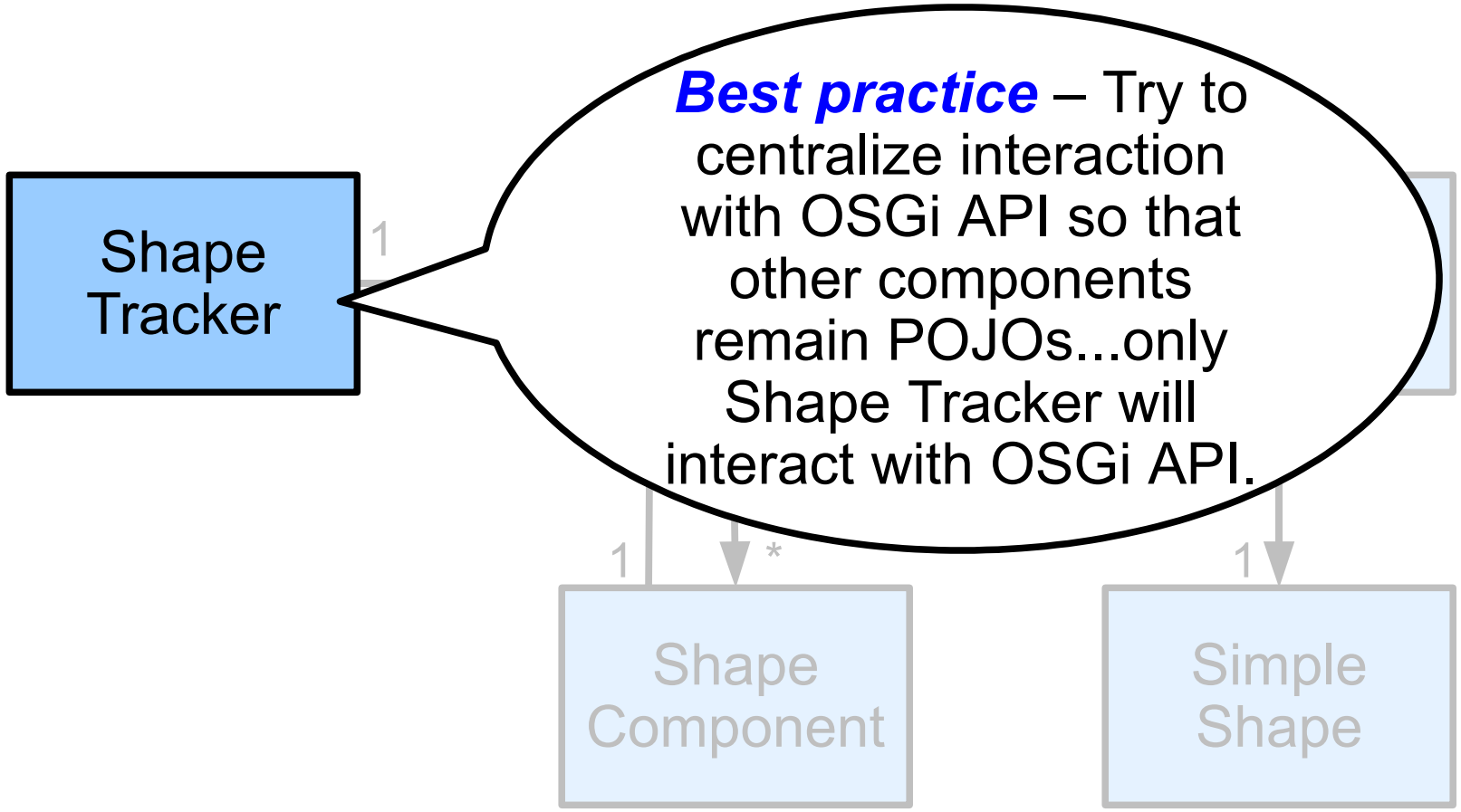


High-Level Architecture

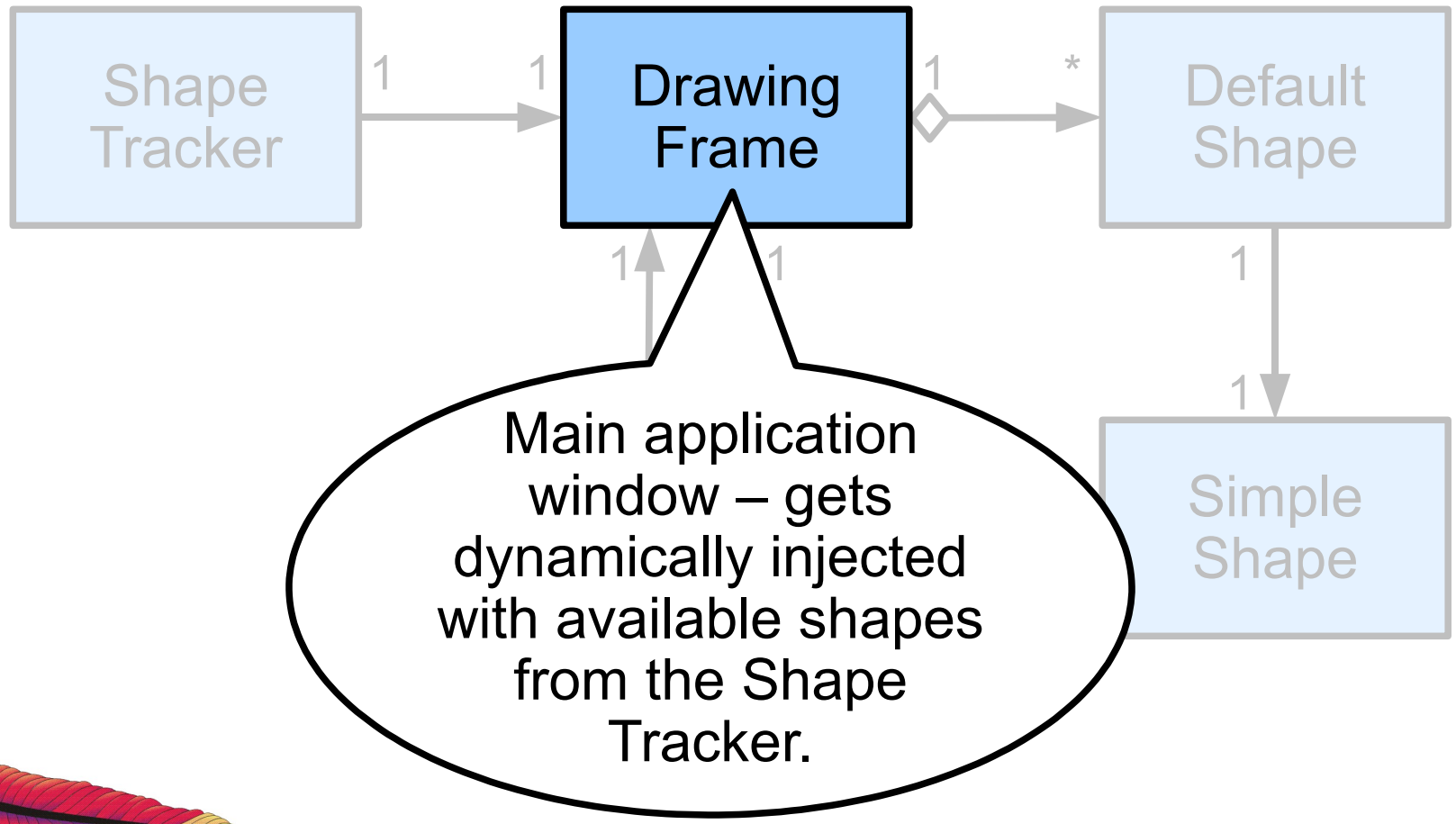




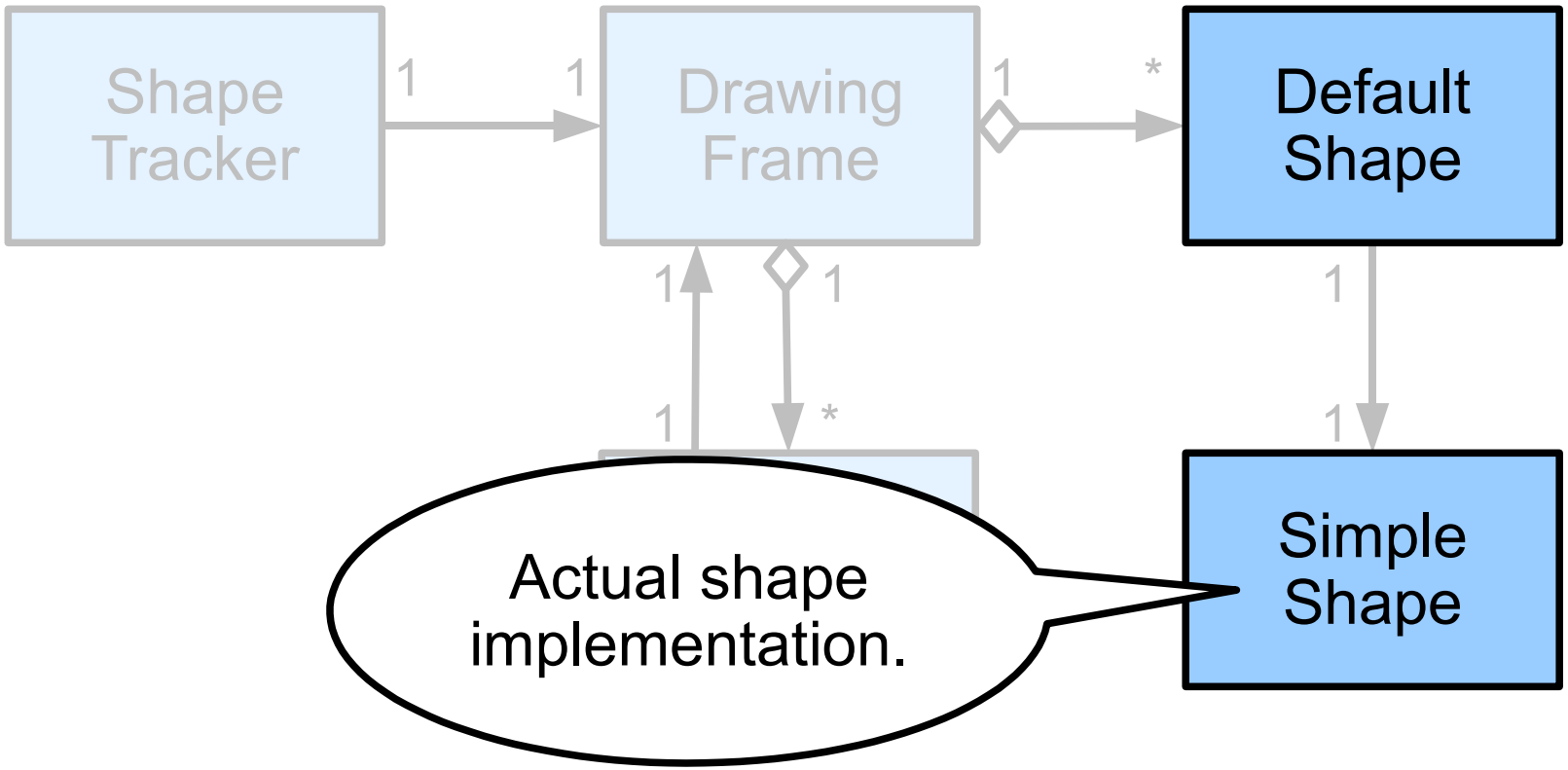
High-Level Architecture



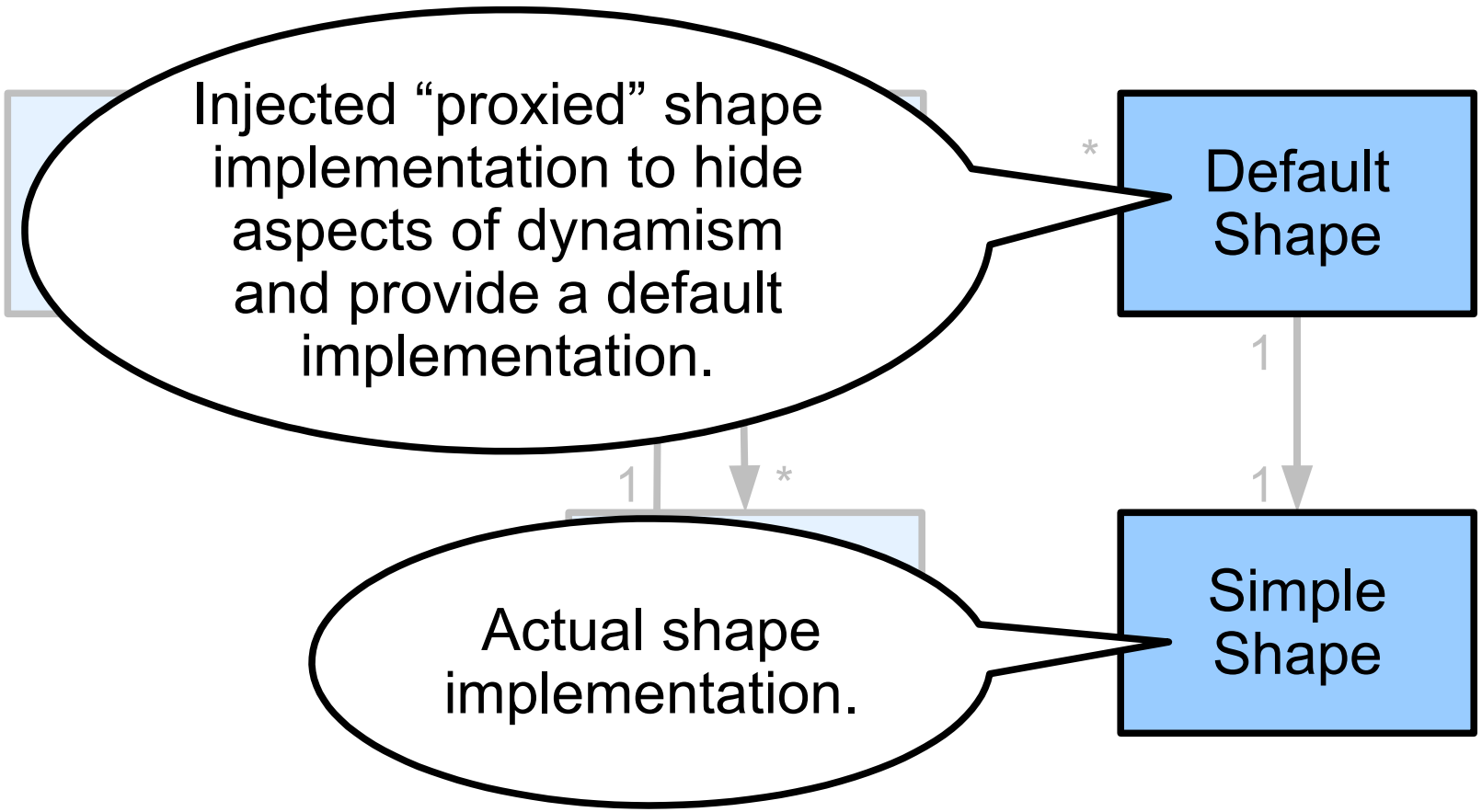
High-Level Architecture



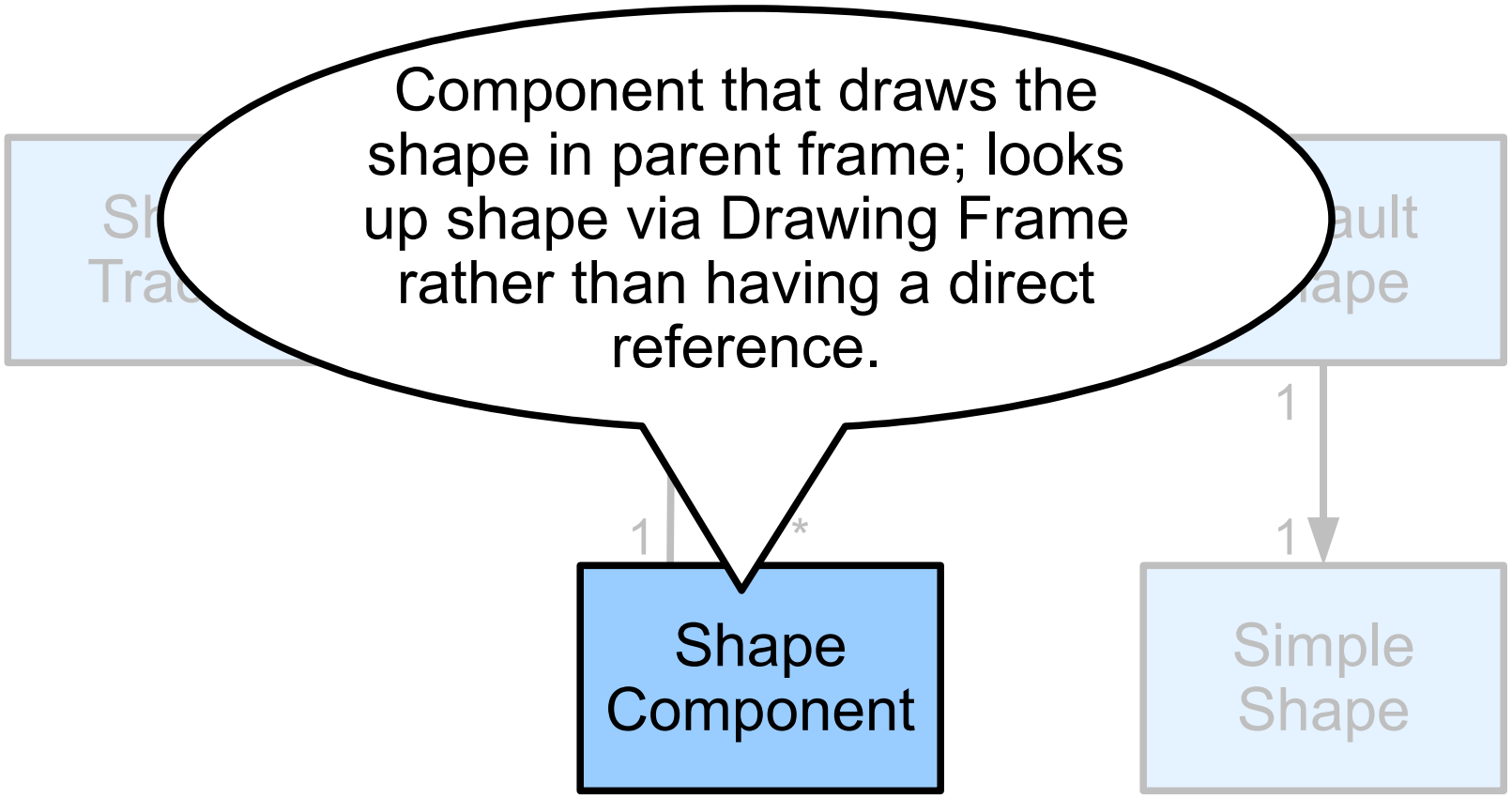
High-Level Architecture



High-Level Architecture

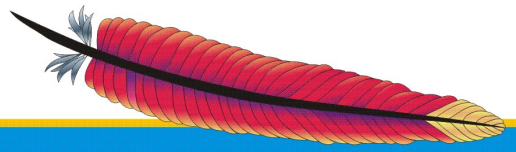


High-Level Architecture

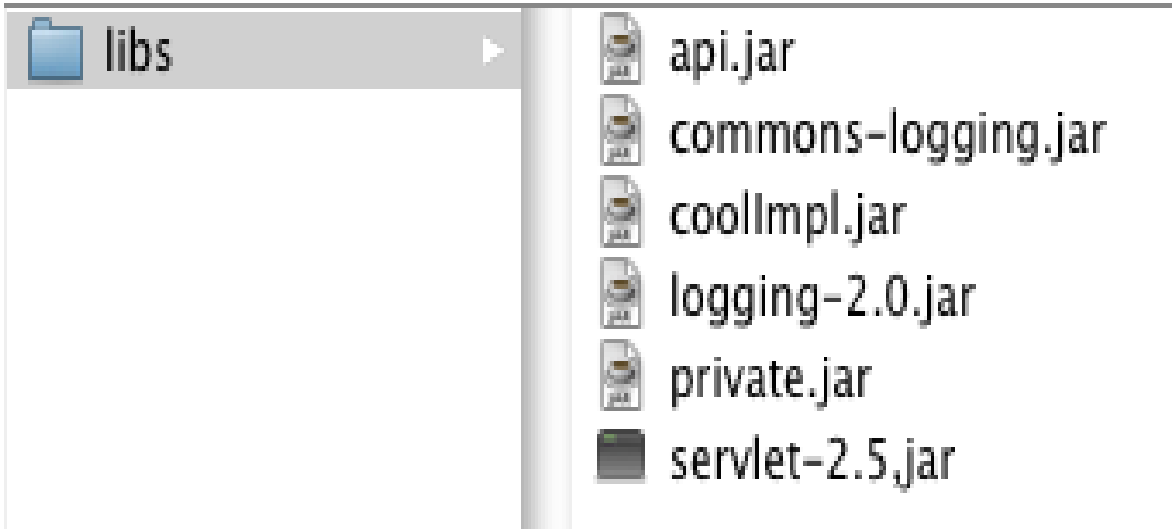


LIVE DEMO

3 Why OSGi?

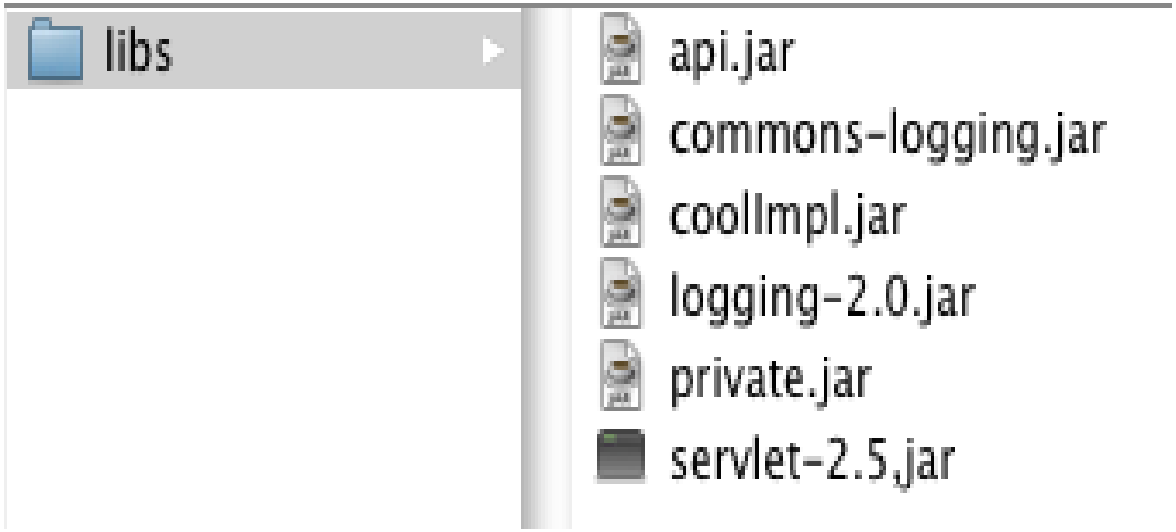


Class Path Hell



- Can you spot some potential problems?

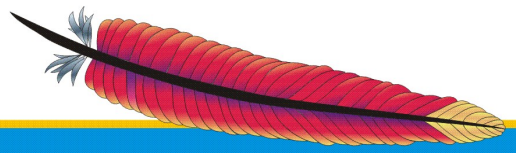
Class Path Hell



- What libs are used? Versions?
- Which jar is used? Version?
- No difference between private and public classes

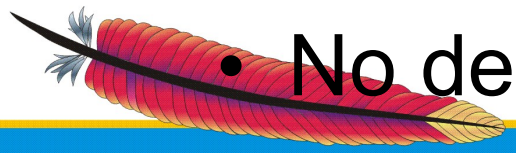
Java's Shortcomings

- Simplistic version handling
  - “First” class from class path
  - JAR files assume backwards compatibility at best
- Implicit dependencies
  - Dependencies are implicit in class path ordering
  - JAR files add improvements for extensions, but cannot control visibility



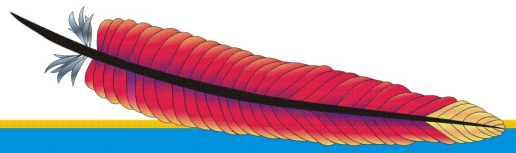
Java's Shortcomings

- Split packages by default
  - Class path approach searches until it finds, which leads to shadowing or version mixing
- Limited scoping mechanisms
  - No module access modifier
  - Impossible to declare all private stuff as private
- Missing module concept
  - Classes are too fine grained, packages are too simplistic, class loaders are too low level
- No deployment/lifecycle support



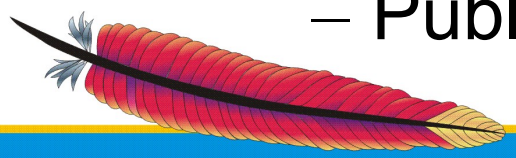
Java Dynamism Limitations

- Low-level support for dynamics
  - Class loaders are complicated to use and error prone
- Support for dynamics is still purely manual
  - Must be completely managed by the programmer
  - Leads to many ad hoc, incompatible solutions
- Limited deployment support

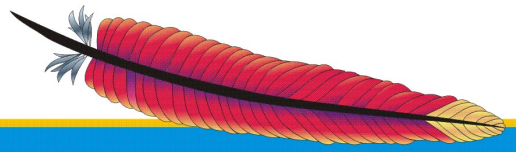


OSGi Technology

- Adds modularity and dynamics
  - Module concept
    - Explicit sharing (importing and exporting)
  - Automatic management of code dependencies
    - Enforces sophisticated consistency rules for class loading
  - Life-cycle management
    - Manages dynamic deployment and configuration
- Service Registry
  - Publish/find/bind

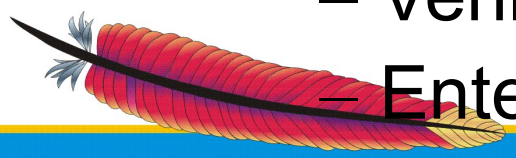


4 Apache Felix



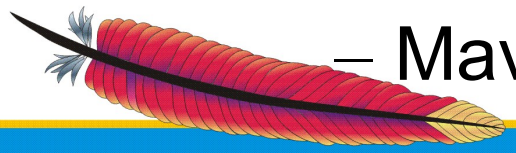
OSGi Alliance

- Industry consortium
- ***OSGi Service Platform*** specification
  - Framework specification for hosting dynamically downloadable services
  - Standard service specifications
- Several expert groups define the specifications
  - Core Platform Expert Group (CPEG)
  - Mobile Expert Group (MEG)
  - Vehicle Expert Group (VEG)
  - Enterprise Expert Group (EEG)



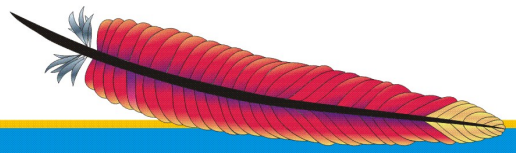
Apache Felix

- Top-level project (March 2007)
- Healthy and diverse community
- OSGi R4 (R4.1) implementation
  - Framework (frequent releases)
  - Services (continued development)
    - Log, Package Admin, Event Admin, Configuration Admin, Declarative Services, Meta Type, Deployment Admin (and more)
  - Moving towards upcoming R4.2
- Tools
  - Maven Plugins, Web Console, iPojo

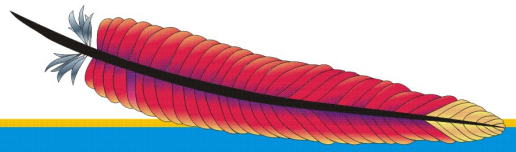


Apache Felix

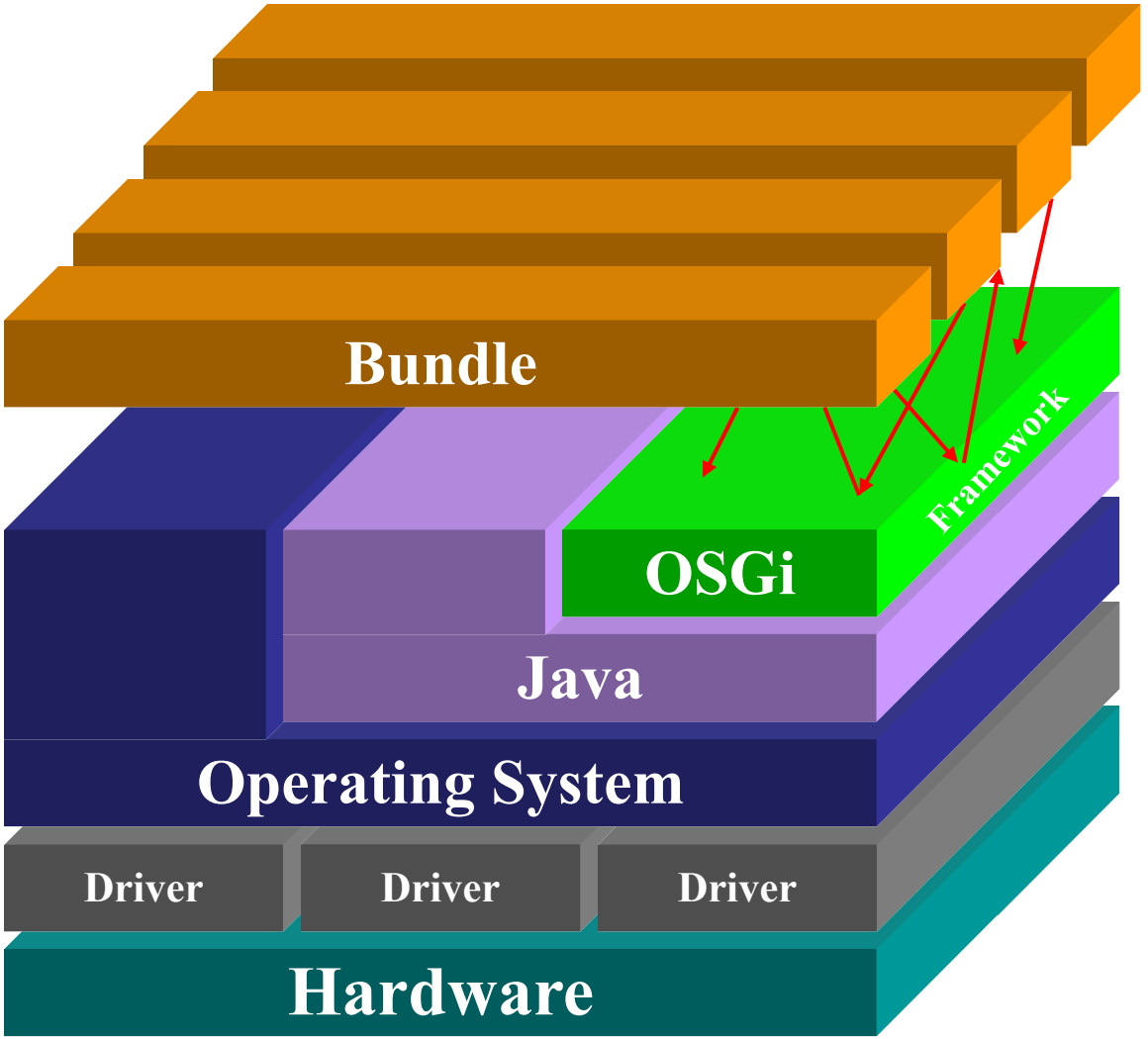
- Growing community
  - Several code grants and contributions
  - Various (Apache) projects use Felix / have expressed interest in Felix and/or OSGi
    - e.g., ServiceMix, Directory, **Sling**, Tuscany
- Roadmap
  - Continue toward R4 and R4.1 compliance
    - some parts consider pre R4.2 already



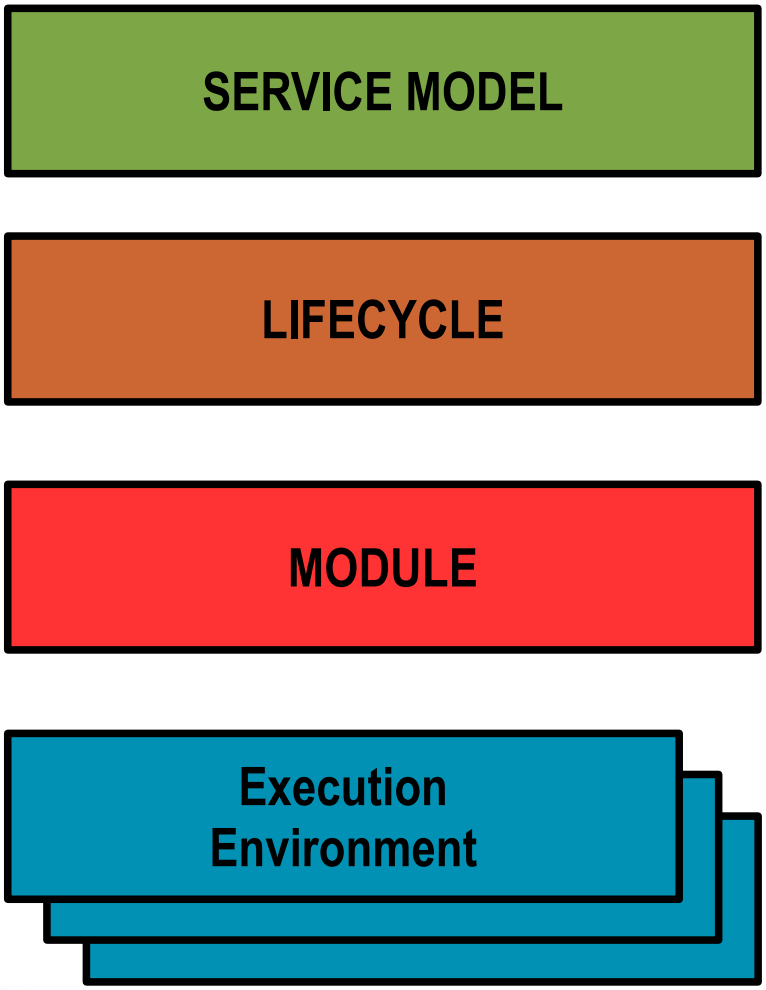
5 OSGi – Part 1
Bundles



OSGi Architectural Overview



OSGi Framework Layering



L3 – Provides a publish/find/bind service model to decouple bundles

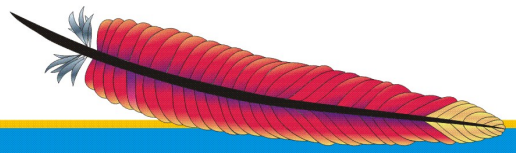
L2 - Manages the life cycle of bundle in a bundle repository without requiring the VM be restarted

L1 - Creates the concept of modules (aka. bundles) that use classes from each other in a controlled way according to system and bundle constraints

L0 -
- OSGi Minimum Execution Environment
- CDC/Foundation
- JavaSE

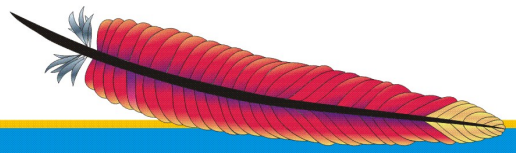
OSGi Framework

- Component-oriented framework
- Module concept: Bundles
  - Separate class loader -> graph
  - Package sharing and version management
  - Life-cycle management and notification
- Dynamic!
  - Install, update, and uninstall at runtime
- Runs multiple applications and services in a single VM



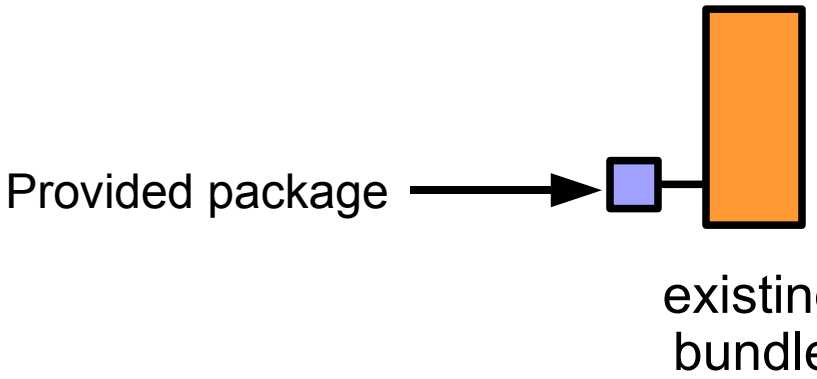
OSGi Modularity

- Explicit code boundaries and dependencies
  - Package imports and exports
- Multi-version support
  - Version ranges for dependencies
- Class space is managed by OSGi
- Managed life cycle
  - Dynamic install, update, uninstall



OSGi Modularity - Example

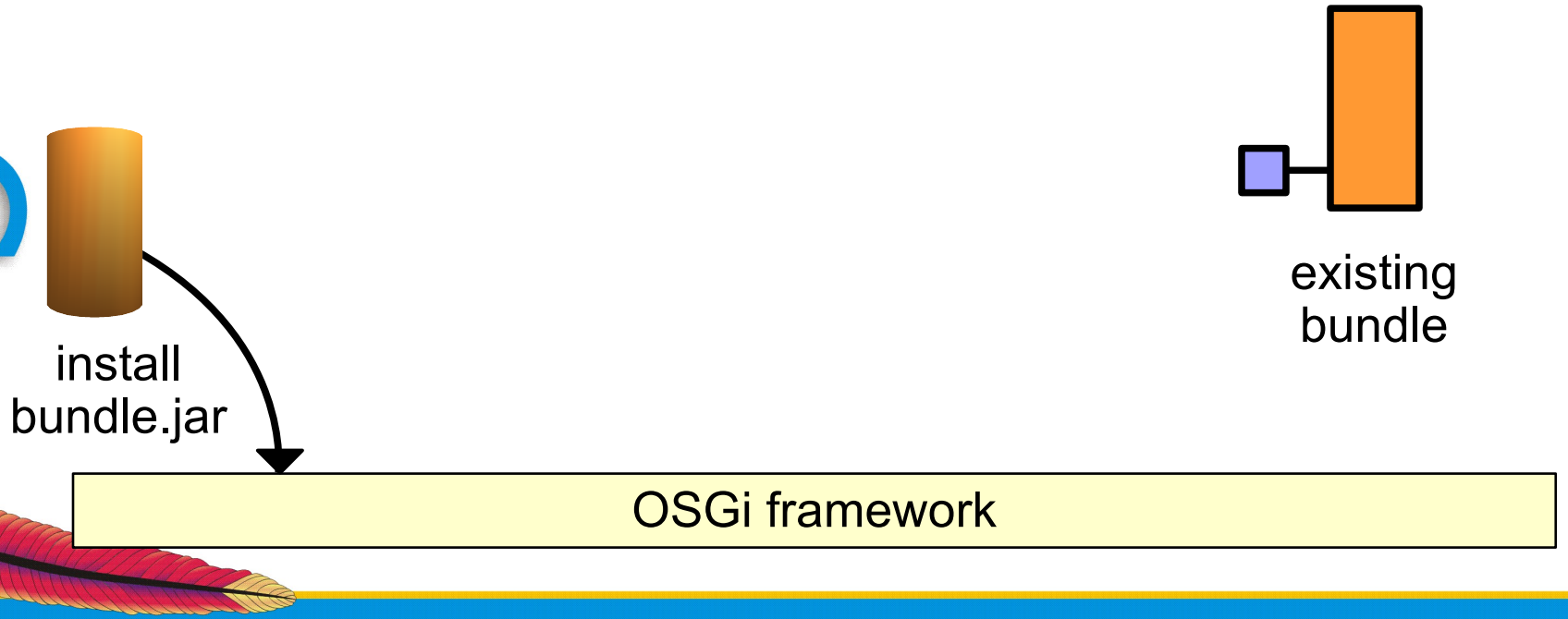
- Dynamic module deployment and dependency resolution



OSGi framework

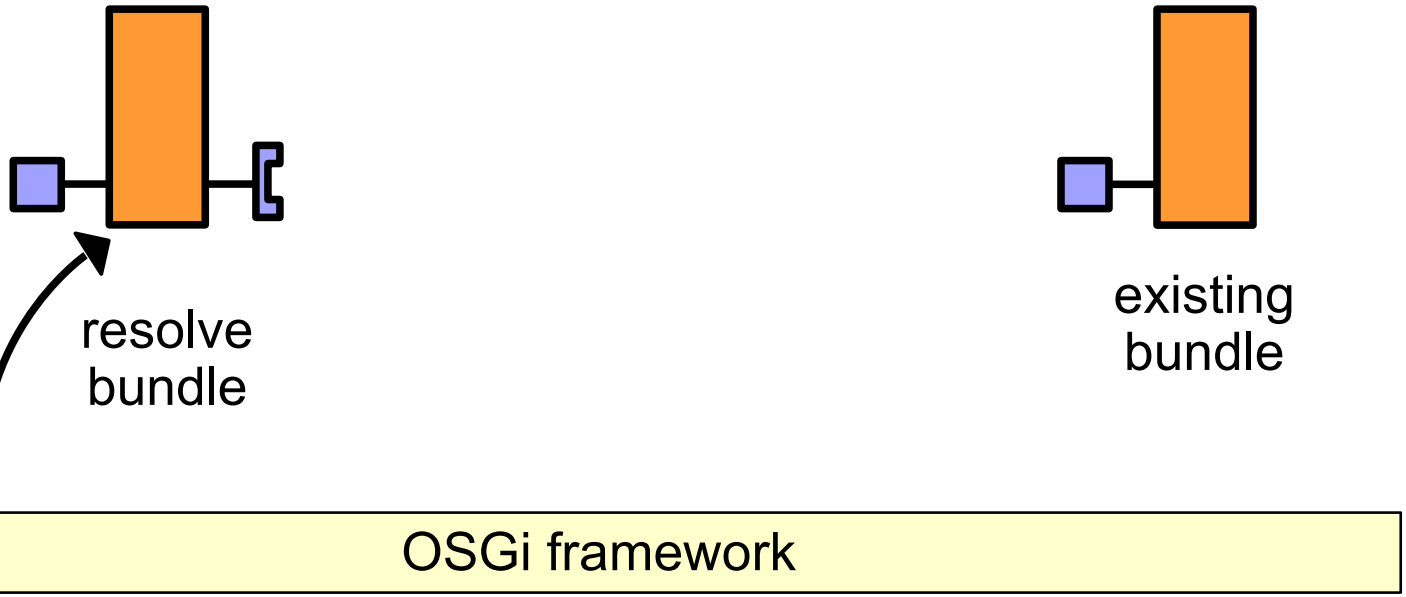
OSGi Modularity - Example

- Dynamic module deployment and dependency resolution



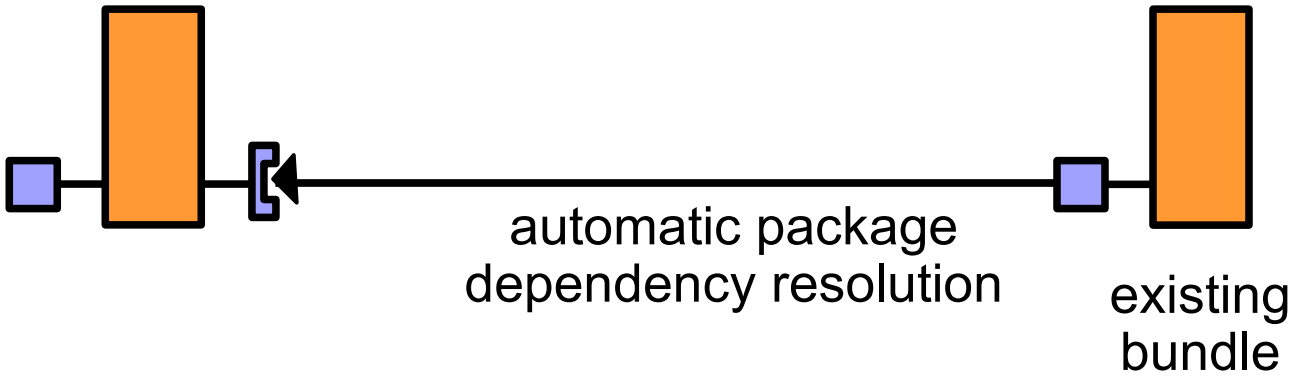
OSGi Modularity - Example

- Dynamic module deployment and dependency resolution



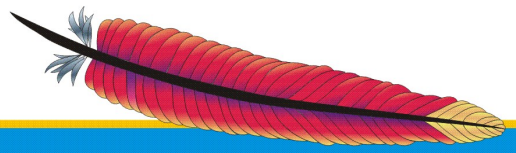
OSGi Modularity - Example

- Dynamic module deployment and dependency resolution



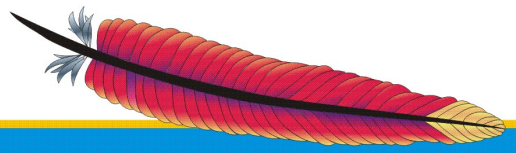
Creating a Bundle

- Plain old JAR with additional metadata in the manifest
  - Bundle identifier, version, exports, imports
- Tools
  - Text editor (Manifest)
  - Eclipse (PDE)
  - Bundle packaging tools
    - **BND** from Peter Kriens
    - Apache Felix ***maven-bundle-plugin*** based on BND



Maven is Your Friend

- Apache Felix Maven Bundle Plugin
- Creates metadata based on POM
  - Automatically: import packages
  - Manually: export and private packages
- Analyses classes for consistency
- Allows to include dependencies
- Creates final bundle JAR file



Maven Bundle Plugin Sample

```
<artifactId>org.apache.sling.engine</artifactId>
<packaging>bundle</packaging>
<version>2.0.3-incubator-SNAPSHOT</version>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.felix</groupId>
      <artifactId>maven-bundle-plugin</artifactId>
      <extensions>true</extensions>
      <configuration>
        <instructions>
          <Export-Package>
            org.apache.sling.engine;version=${pom.version}
          </Export-Package>
          <Private-Package>
            org.apache.sling.engine.impl
          </Private-Package>
          <Embed-Dependency>
            commons-fileupload
          </Embed-Dependency>
        </instructions>
```

Maven Bundle Plugin Sample

```
<artifactId>org.apache.sling.engine</artifactId>
<packaging>bundle</packaging>
<version>2.0.3-incubator-SNAPSHOT</version>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.felix</groupId>
      <artifactId>maven-bundle-plugin</artifactId>
      <extensions>true</extensions>
      <configuration>
        <instructions>
          <Export-Package>
            org.apache.sling.engine;version=${pom.version}
          </Export-Package>
          <Private-Package>
            org.apache.sling.engine.impl
          </Private-Package>
          <Embed-Dependency>
            commons-fileupload
          </Embed-Dependency>
        </instructions>
```

Maven Bundle Plugin Sample

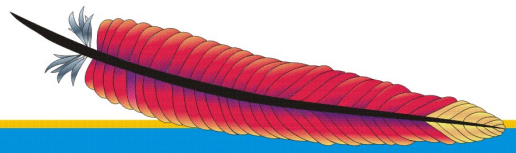
```
<artifactId>org.apache.sling.engine</artifactId>
<packaging>bundle</packaging>
<version>2.0.3-incubator-SNAPSHOT</version>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.felix</groupId>
      <artifactId>maven-bundle-plugin</artifactId>
      <extensions>true</extensions>
      <configuration>
        <instructions>
          <Export-Package>
            org.apache.sling.engine;version=${pom.version}
          </Export-Package>
          <Private-Package>
            org.apache.sling.engine.impl
          </Private-Package>
          <Embed-Dependency>
            commons-fileupload
          </Embed-Dependency>
        </instructions>
```

Maven Bundle Plugin Sample

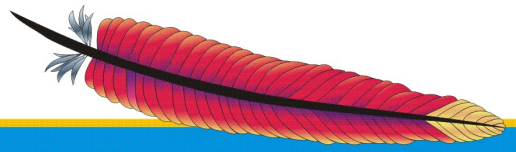
```
<artifactId>org.apache.sling.engine</artifactId>
<packaging>bundle</packaging>
<version>2.0.3-incubator-SNAPSHOT</version>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.felix</groupId>
      <artifactId>maven-bundle-plugin</artifactId>
      <extensions>true</extensions>
      <configuration>
        <instructions>
          <Export-Package>
            org.apache.sling.engine;version=${pom.version}
          </Export-Package>
          <Private-Package>
            org.apache.sling.engine.impl
          </Private-Package>
          <Embed-Dependency>
            commons-fileupload
          </Embed-Dependency>
        </instructions>
```

Be Modular!

- Create clean package spaces
  - public vs private
- Provide Bundles
  - Add manifest information
- Think about dependencies
  - Additional bundle vs include
  - Optional
  - Version ranges
- Benefits even without OSGi

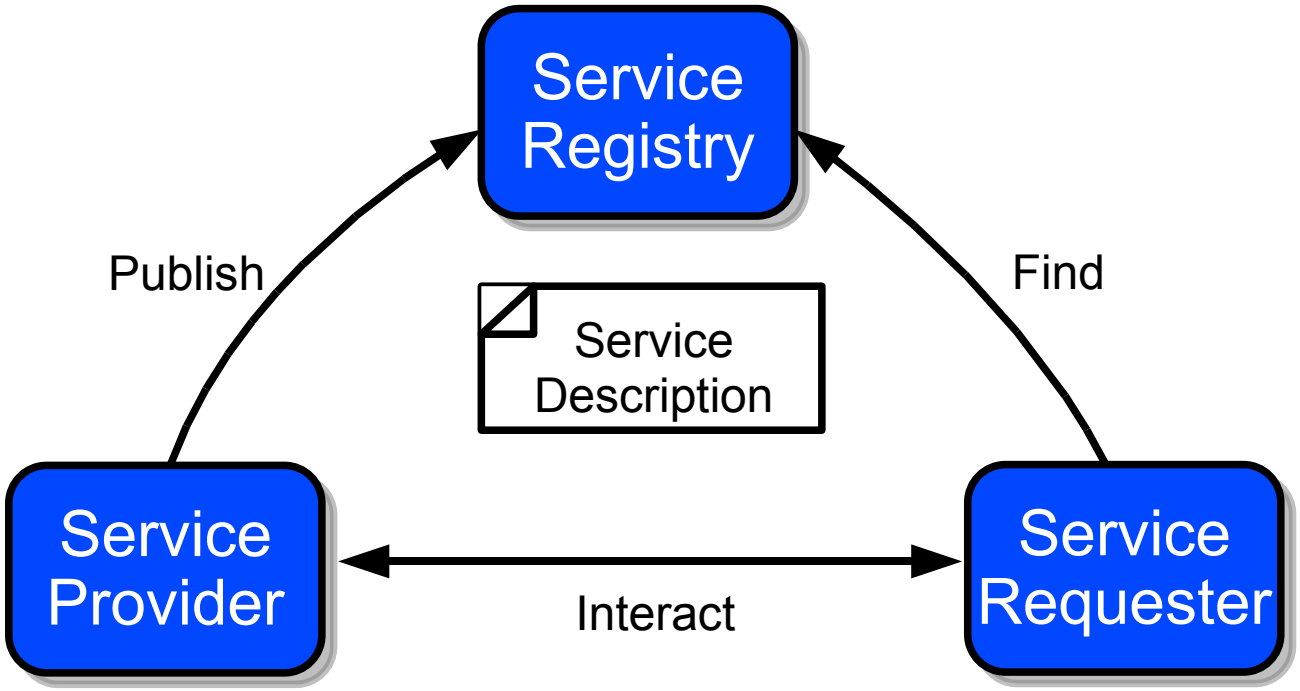


6 OSGi – Part 2
Services



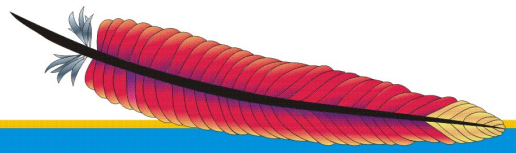
OSGi Services (1/3)

- Service-oriented architecture
  - Publish/find/bind
  - Possible to use modules without services



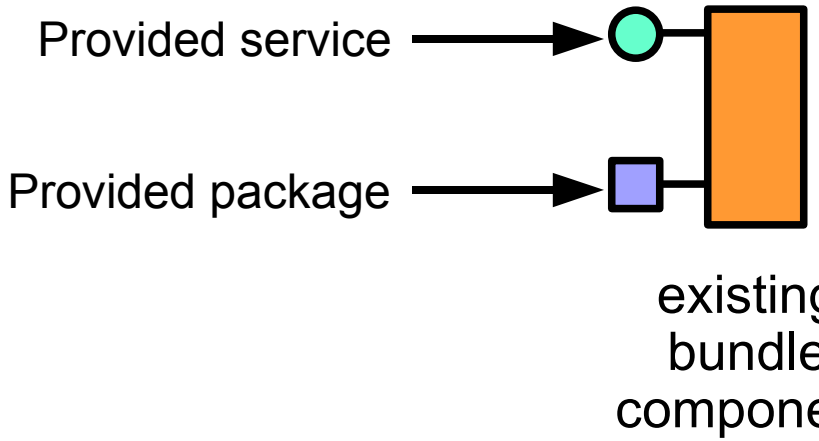
OSGi Services (2/3)

- An OSGi application is...
  - A collection of bundles that interact via service interfaces
  - Bundles may be independently developed and deployed
  - Bundles and their associated services may appear or disappear at any time
- Resulting application follows a ***Service-Oriented Component Model*** approach



OSGi Services (3/3)

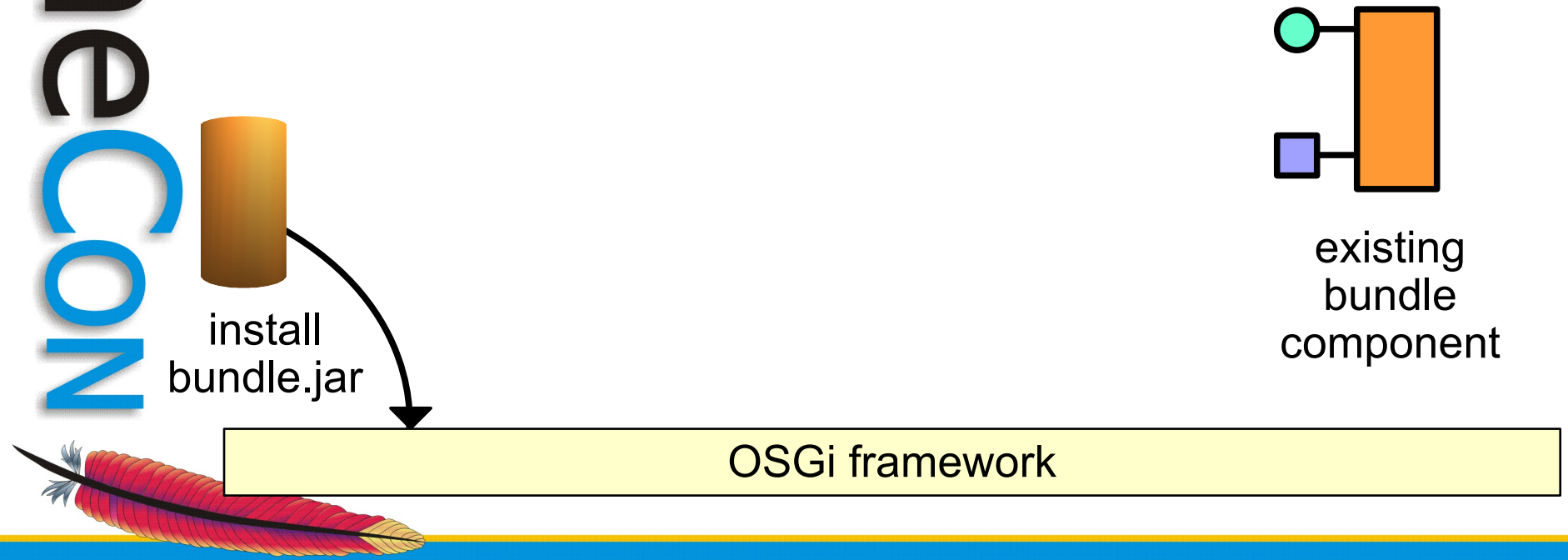
- Dynamic service lookup



OSGi framework

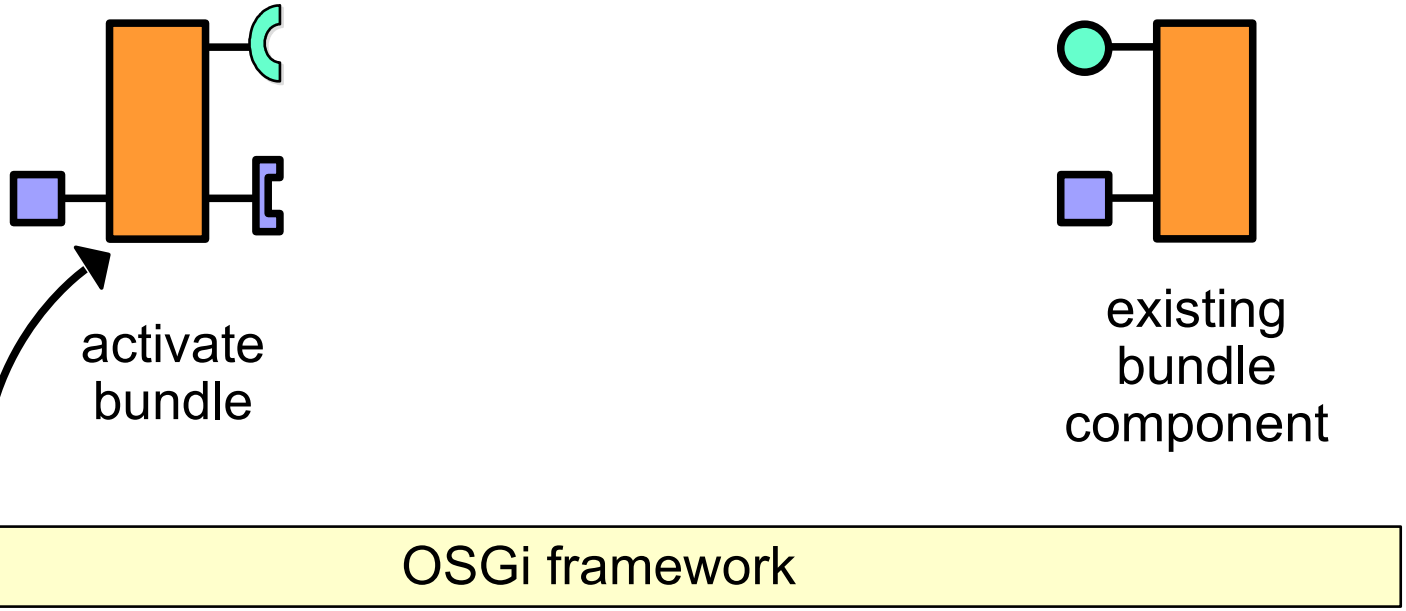
OSGi Services (3/3)

- Dynamic service lookup



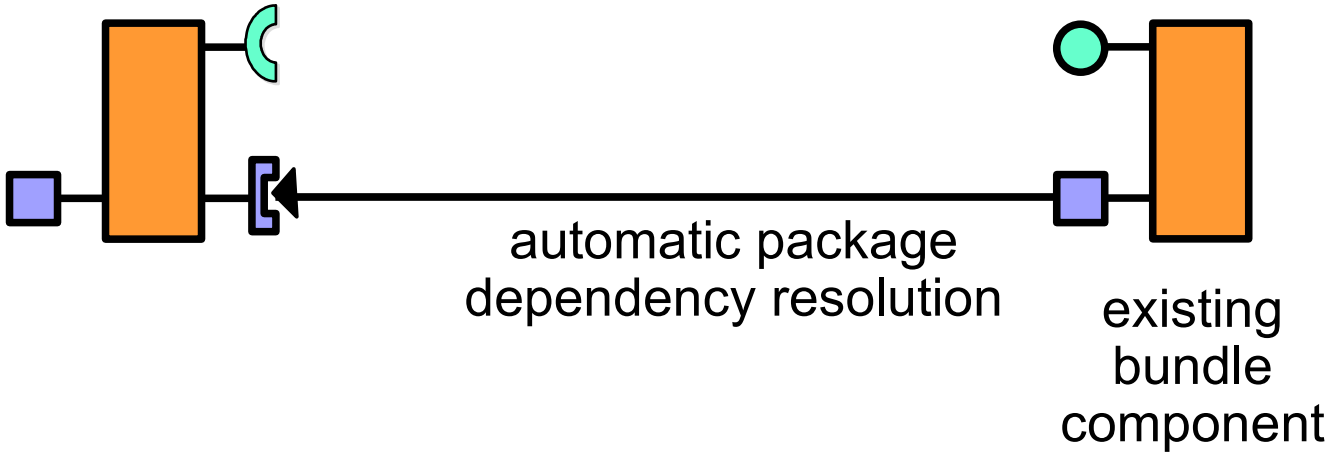
OSGi Services (3/3)

- Dynamic service lookup



OSGi Services (3/3)

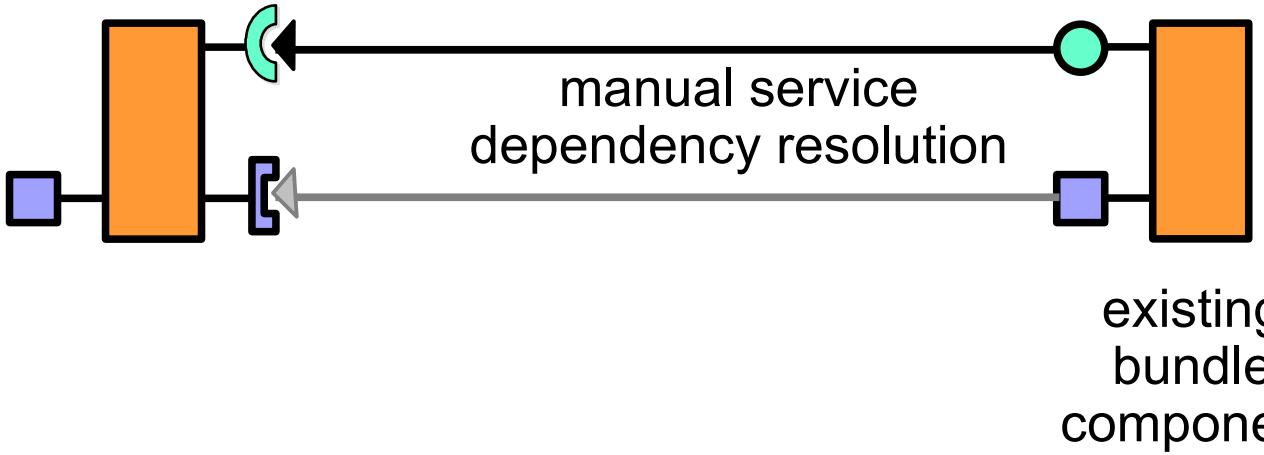
- Dynamic service lookup



OSGi framework

OSGi Services (3/3)

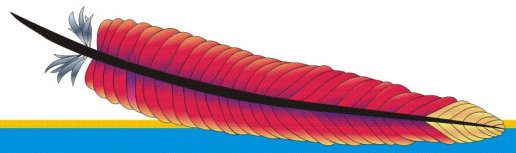
- Dynamic service lookup



OSGi framework

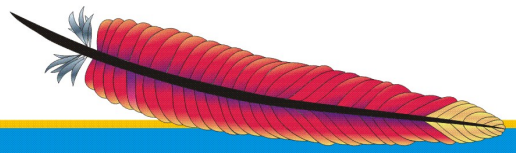
OSGi Services Advantages

- Lightweight services
  - Lookup is based on interface name
  - Direct method invocation
- Good design practice
  - Separates interface from implementation
  - Enables reuse, substitutability, loose coupling, and late binding



OSGi Services Advantages

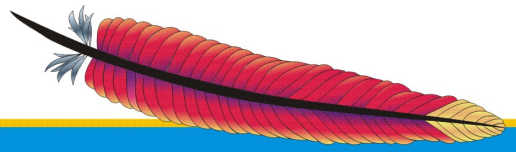
- Dynamic
  - Loose coupling and late binding
- Application's configuration is simply the set of deployed bundles
  - Deploy only the bundles that you need



OSGi Services Issues

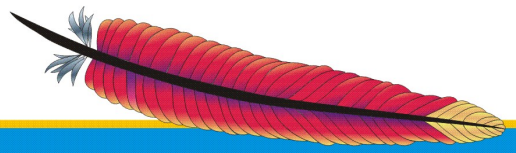
- More sophisticated, but more complicated
  - Requires a different way of thinking
    - Things might appear/disappear at any moment
  - Must manually resolve and track services
- There is help
  - Service Tracker
    - Still somewhat of a manual approach
  - Declarative Services, Spring DM, iPOJO
    - Sophisticated service-oriented component frameworks
    - Automated dependency injection and more
    - More modern, POJO-oriented approaches

7 OSGi – Part 3 Dynamics



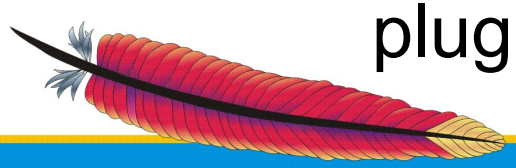
Everything is a Bundle

- How to structure bundles?
  - API vs implementation bundle
  - Fine-grained vs coarse-grained
  - No “One Size Fits All”
- Simple Rules
  - Stable code vs changing code
  - Optional parts



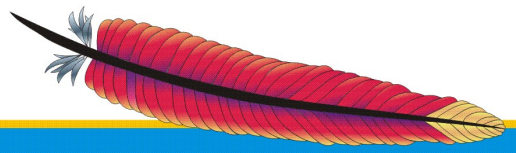
Third Party Libraries

- Use as bundles
  - Project delivers already a bundle
    - Apache Commons, Apache Sling etc.
  - Use special bundle repositories
    - Felix Commons, Spring etc.
    - But check included metadata!
  - Create your own wrapper
    - Easy with the Felix maven bundle plugin
- Include in your bundle
  - Again: easy with the Felix maven bundle plugin



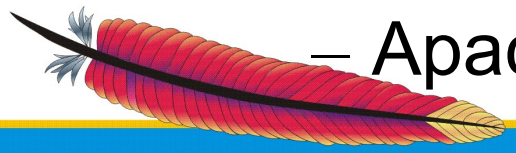
Everything is Dynamic

- Bundles can come and go!
  - Packages
  - Services
- Services can come and go!
- Be prepaired!
  - Application code must handle dynamics!



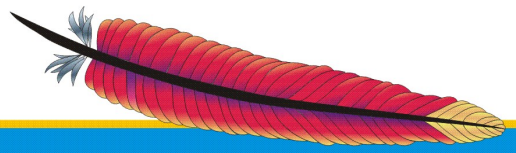
Dynamic Services

- OSGi Declarative Services Specification
  - XML Configuration
    - Contained in bundle
    - Manifest entry pointing to config(s)
  - Publishing services
  - Consuming services
    - Policy (static,dynamic), cardinality (0..1, 1..1, 0..n)
  - Default configuration
  - Service Lifecycle management
- Various Implementations
  - Apache Felix SCR



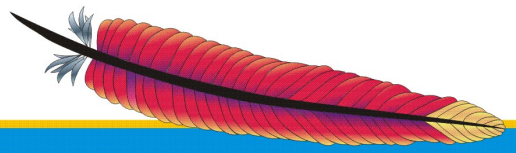
Dynamic Services Configuration

```
<scr:component enabled="true"
        name="org.apache.sling.event.impl.DistributingEventHandler">

  <implementation
      class="org.apache.sling.event.impl.DistributingEventHandler"/>

  <service servicefactory="false">
    <provide interface="org.osgi.service.event.EventHandler"/>
  </service>

  <property name="repository.path" value="/var/eventing/distribution"/>
  <property name="cleanup.period" type="Integer" value="15"/>

  <reference name="threadPool"
             interface="org.apache.sling.event.ThreadPool"
             cardinality="1..1" policy="static"
             bind="bindThreadPool" unbind="unbindThreadPool"/>
```



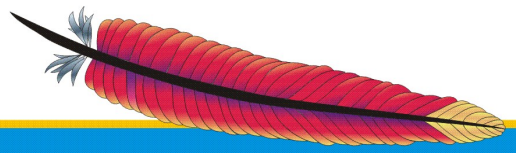
Dynamic Services Configuration

```
<scr:component enabled="true"
        name="org.apache.sling.event.impl.DistributingEventHandler">

  <implementation
       class="org.apache.sling.event.impl.DistributingEventHandler"/>

  <service servicefactory="false">
    <provide interface="org.osgi.service.event.EventHandler"/>
  </service>

  <property name="repository.path" value="/var/eventing/distribution"/>
  <property name="cleanup.period" type="Integer" value="15"/>

  <reference name="threadPool"
             interface="org.apache.sling.event.ThreadPool"
             cardinality="1..1" policy="static"
             bind="bindThreadPool" unbind="unbindThreadPool"/>
```



Dynamic Services Configuration

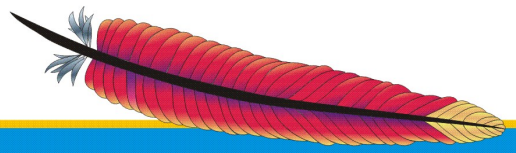
```
<scr:component enabled="true"
        name="org.apache.sling.event.impl.DistributingEventHandler">

  <implementation
       class="org.apache.sling.event.impl.DistributingEventHandler"/>

  <service servicefactory="false">
    <provide interface="org.osgi.service.event.EventHandler"/>
  </service>

  <property name="repository.path" value="/var/eventing/distribution"/>
  <property name="cleanup.period" type="Integer" value="15"/>

  <reference name="threadPool"
             interface="org.apache.sling.event.ThreadPool"
             cardinality="1..1" policy="static"
             bind="bindThreadPool" unbind="unbindThreadPool"/>
```



Dynamic Services Configuration

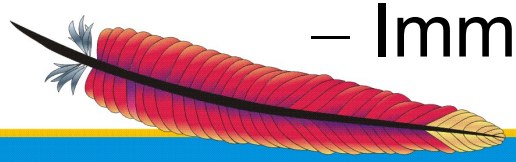
```
<scr:component enabled="true"
        name="org.apache.sling.event.impl.DistributingEventHandler">

  <implementation
       class="org.apache.sling.event.impl.DistributingEventHandler"/>

  <service servicefactory="false">
    <provide interface="org.osgi.service.event.EventHandler"/>
  </service>

  <property name="repository.path" value="/var/eventing/distribution"/>
  <property name="cleanup.period" type="Integer" value="15"/>
```

```
<reference name="threadPool"
           interface="org.apache.sling.event.ThreadPool"
           cardinality="1..1" policy="static"
           bind="bindThreadPool" unbind="unbindThreadPool"/>
```



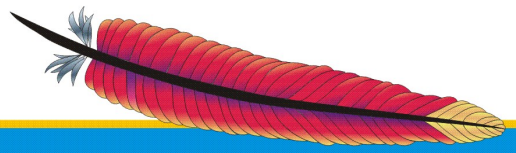
Declarative Services

- Reads XML configs on bundle start
- Registers services
- Keeps track of dependencies
  - Starts/stops services
- Invokes optional activation and deactivation method
  - Provides access to configuration
- Caution: A service is by default only started if someone else uses it!
  - Immediate flag forces a service start



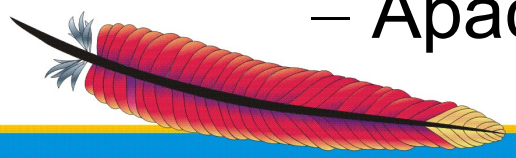
Example Service

```
protected ThreadPool threadPool;

protected void activate(ComponentContext context)
throws Exception {
    @SuppressWarnings("unchecked")
    final Dictionary<String, Object> props = context.getProperties();
    this.cleanupPeriod = (Integer)props.get("cleanup.period");
    super.activate(context);
}

protected void bindThreadPool(ThreadPool p) {
    this.threadPool = p;
}

protected void unbindThreadPool(ThreadPool p) {
    if ( this.threadPool == p ) {
        this.threadPool = null;
    }
}
```



Config Admin and Metatype

- OSGi Config Admin
  - Configuration Manager
  - Persistence storage
  - API to retrieve/update/remove configs
  - Works with Declarative Services
- OSGi Metatype Service
  - Description of bundle metadata
  - Description of service configurations
- Various Implementations
  - Apache Felix



Maven SCR Plugin

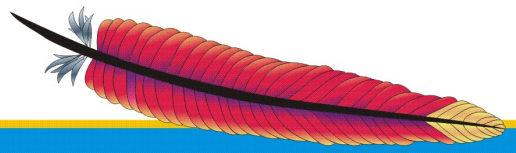
- Combines everything (DS, ConfigAdmin, Metatype, Maven)
- Annotation-based (works for 1.4+)
- Annotate components
  - Properties with default values
  - Service providers
  - Services references (policy and cardinality)
- Generates DS XML
- Generates Metatype config
- Generates Java code

SCR Plugin Sample

```
/**
 * @scr.component
 * @scr.property name="repository.path"
                 value="/var/eventing/distribution" private="true"
 * @scr.service interface="EventHandler"
 */
public class DistributingEventHandler
    implements EventHandler {

    protected static final int DEFAULT_CLEANUP_PERIOD = 15;

    /** @scr.property valueRef="DEFAULT_CLEANUP_PERIOD" type="Integer" */
    protected static final String PROP_CLEANUP_PERIOD = "cleanup.period";

    /** @scr.reference */
    protected ThreadPool threadPool;

    protected void activate(ComponentContext context)
    throws Exception {
        final Dictionary<String, Object> props = context.getProperties();
        this.cleanupPeriod = (Integer)props.get(PROP_CLEANUP_PERIOD);
    }
```

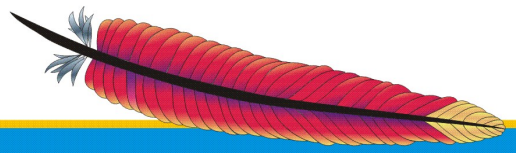
Alternatives

- Manually through bundle activator
- Apache Felix iPojo
- Spring Dynamic Modules



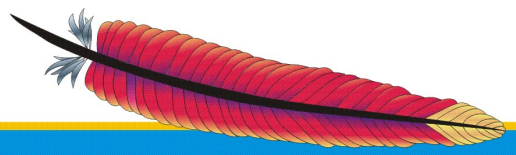
Handling extensibility

- Two basic implementation strategies
  - Service-based approach
  - Extender model



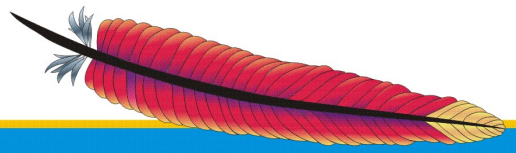
Service Whiteboard Pattern

- Clients register a service interface
- Service tracker for registered services
- Simple, more robust, leverages the OSGi service model
- Service whiteboard pattern
  - It is an *Inversion of Control* pattern



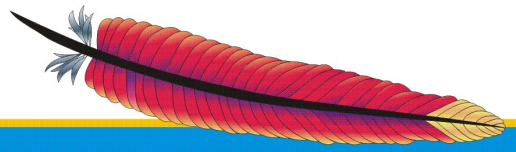
Externder Model

- Bundles contain manifest entries
  - Like available service classes
- Custom bundle tracker
  - Keeps track of bundles
  - Specifically, `STARTED` and `STOPPED` events
  - Checks bundles manifest data
    - Creates/removes services



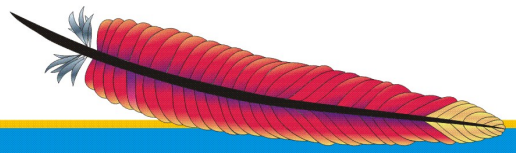
ApacheCon

8 Famous Final Words



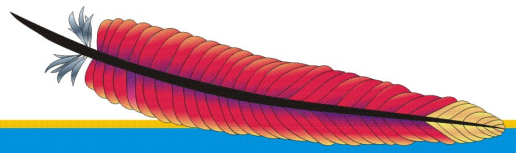
Conclusion

- Modulary and dynamics are required by todays applications
- OSGi technology addresses Java's limitations in these areas
  - Available today and growing in importance
- Development is straightforward and provides immediate benefits
- Apache Felix is ready when you are!



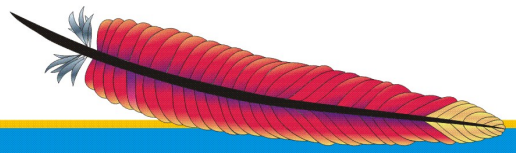
Suggestions for Development

- Think about modularity!
  - Clean package space
- Think about dynamics!
- Consider OSGi
- Check out the spec and other projects
- Minimize dependencies to OSGi
  - but only if it makes sense



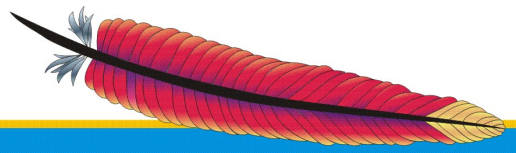
Suggestions for Using OSGi

- Think about dynamics
  - Optional bundles
  - Optional services
  - Handle these cases
- Use your preferred logging library
  - LogManager takes care
- Use available tooling
- Be part of the community!



Check It Out

- Read the OSGi spec
  - Framework
  - Config Admin, Metatype, Declarative Services
  - Deployment Admin, OBR
- Download Apache Felix
  - Try tutorials and samples
- Download Apache Sling :)
- Explore the web – **embrace OSGi**



Embrace ~~Change~~ OSGi

Questions?