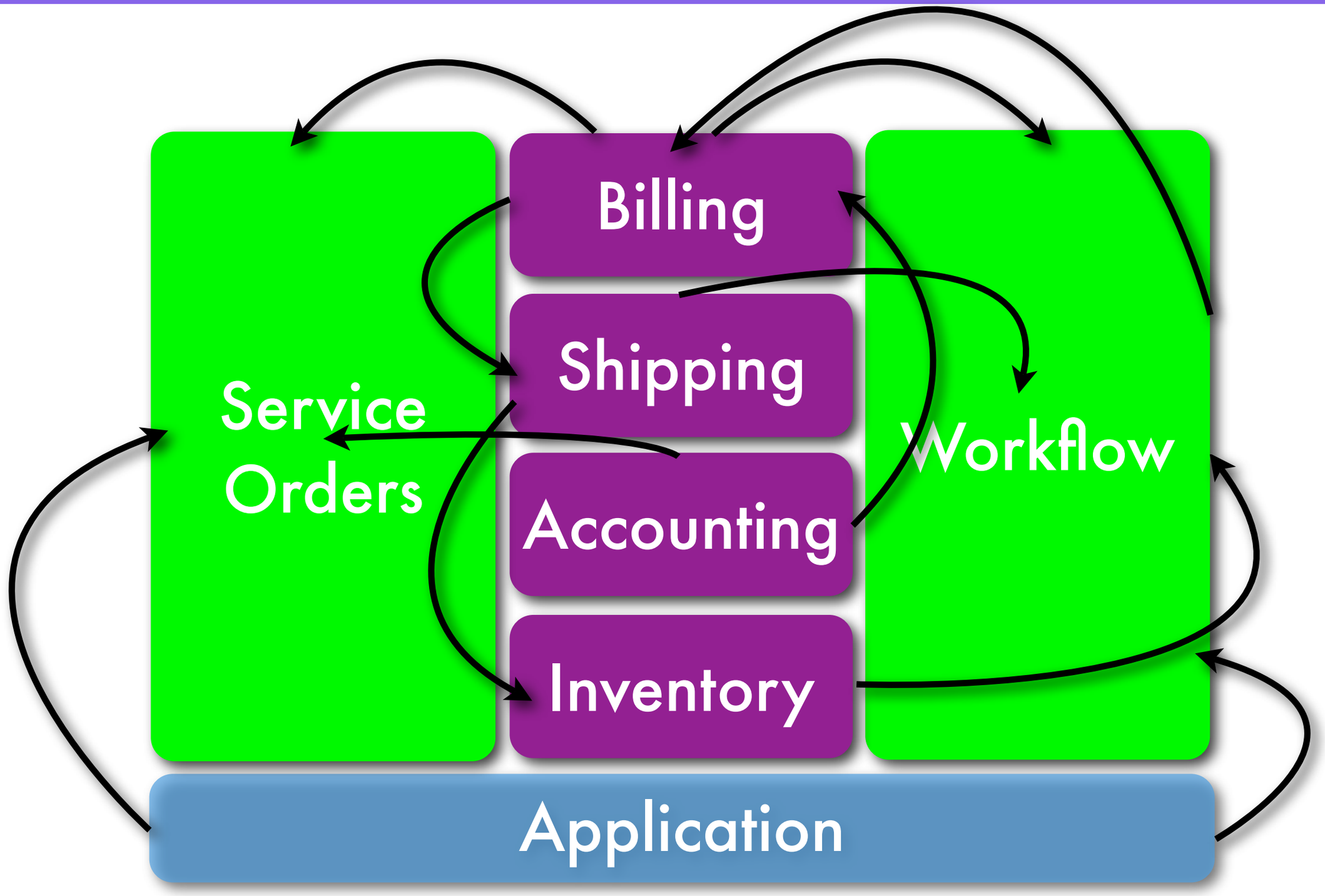
OSGi

Building and Managing Pluggable Applications

What A Mess


Billing
Shipping
Accounting
Inventory
Service Orders
Workflow
Application

From The View Of...

- Building monolithic applications is evil
  - 'nuf said
- Strict dependency management is critical
  - Practice developing decoupled applications
- Determine if OSGi is a capable technology for developing a service-oriented, pluggable Swing application

OSGi Overview

- **O**pen **S**ervices **G**ateway **I**nitiative
- Founded in 1999
- Java-specific
- Mission

"is to create open specifications for the network delivery of managed services to local networks and devices"

OSGi R3 Specification, Section 1.1

OSGi Services Platform

- Component model
- Lifecycle management
- Dependency Management
  - clean separation of specification and implementation
- Security
- Service Registry

Enables dynamic delivery of multiple services

Dependency Management is key to developing dynamic, pluggable applications

What Can I Do With It?

- Create applications that are
  - dynamic
  - service-oriented
  - pluggable
  - substitutable
- Who's Using It?
  - BMW
  - Eclipse 3
  - JSR 232 (Mobile Operations Management) is keeping tabs on OSGi

What's Missing?

- The OSGi specification does not manage service dependencies
- Must manually manage service dependencies
  - service listeners
  - the OSGi service tracker utility class

How Do I Get Started?

- Need a development environment to manage the bundles
- Need to practice loose coupling
  - expose all public functionality as interfaces in a separate JAR file
- Need an OSGi framework
- Need a "management system" to discover and manage bundles

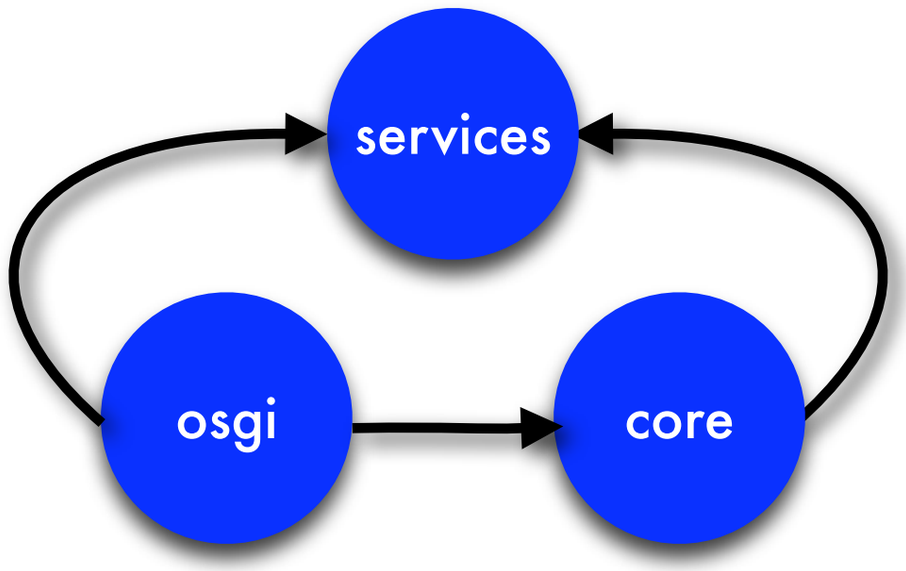
Development Environment

- Ant
  - Flexible but requires a lot of work to manage inter-module dependencies
  - Maven 2 Ant tasks *may* help ease some pain
- Maven
  - Never used it
  - Supposedly manages inter-module dependencies

Maven has an OSGi goal that may help

Dependency Management

- Each Bundle has these internal dependencies
- Each module creates a JAR file
- The **osgi** module is the deployable artifact




```
+ my-bundle
   + core
      + src
   + osgi
      + src
   + services
      + src
```

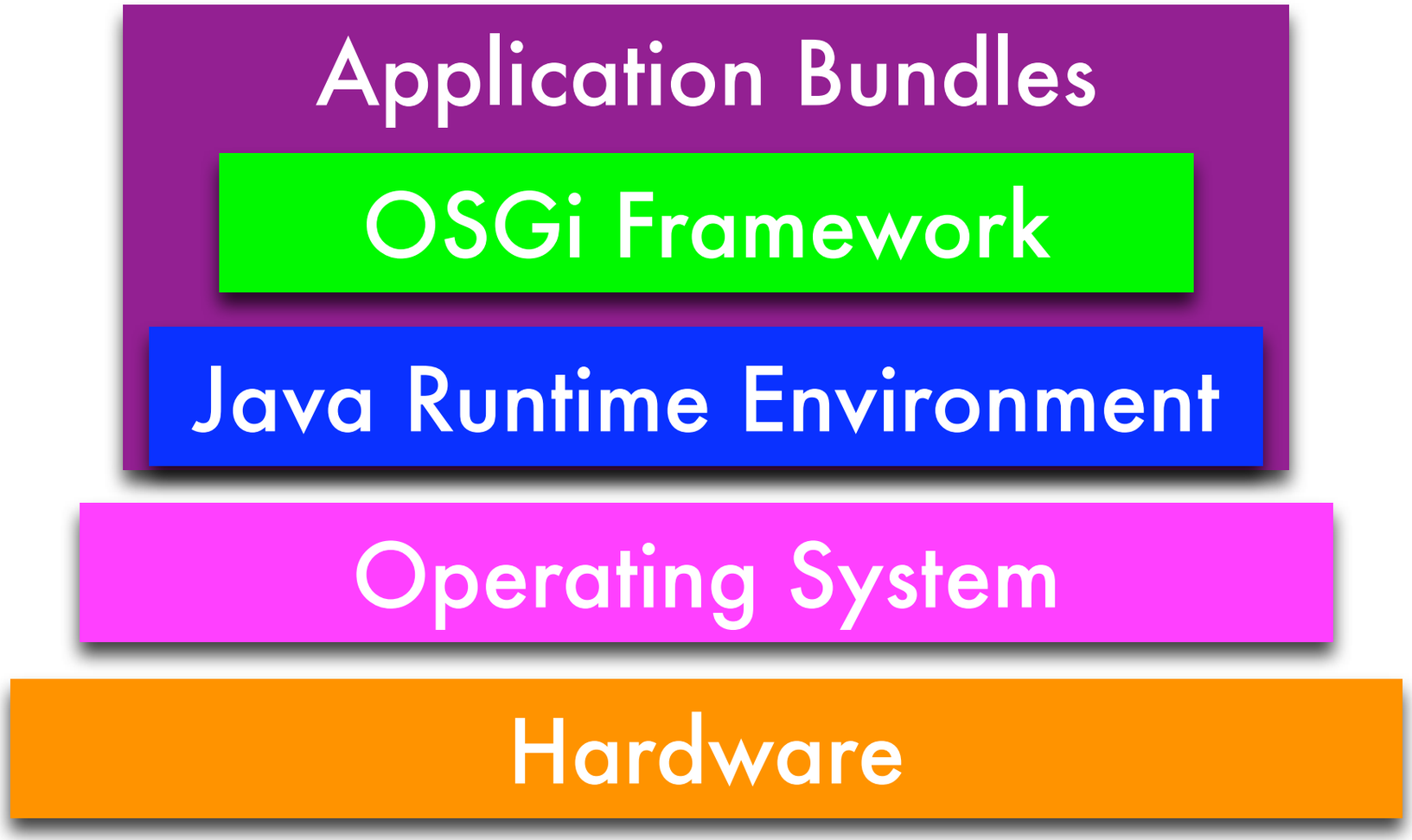
OSGi Frameworks

	Oscar	Knopflerfish
Download	http://oscar.objectweb.org/	http://www.knopflerfish.org/
Active?	Active	Active
Configurability	Pretty simple	Seems involved
OSGi Version	3	3
Which One?	You should try both and see what you like better	

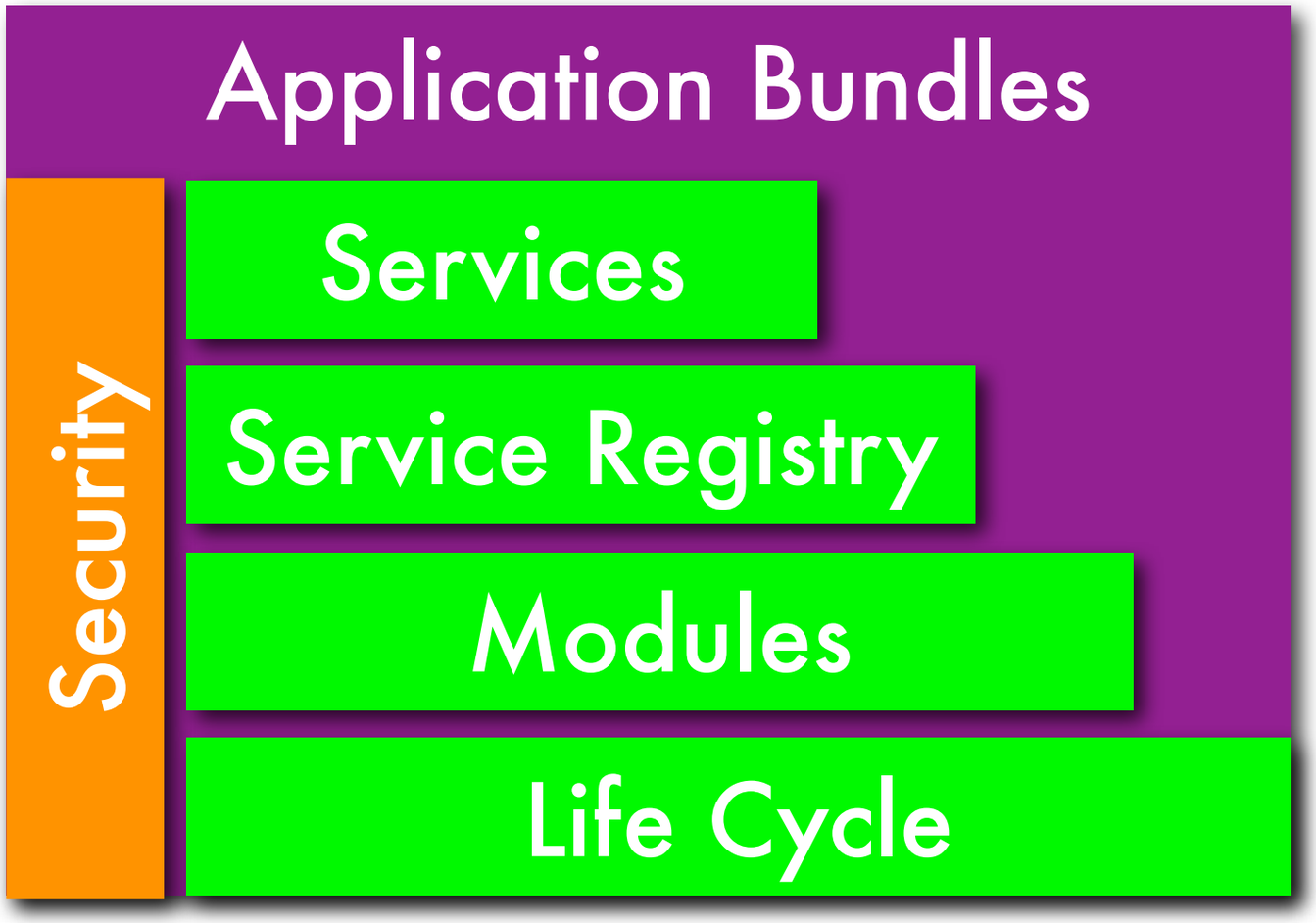
Discovery Service

- Broadcast System
- JINI
  - There is work being done in this area
- File System Scanner
  - Look in a well-known directory for new bundles

Technology Stack



OSGi Framework



OSGi Terminology

- Service
- Service Oriented Architecture
  - not just for “web” services
- Service Registry
- Device
- Bundle

OSGi Service

- Service
  - Java interface describing public behavior (POJI)
  - Services are registered with a service registry
- Example:


```
public interface Page {
    JPanel getView();
    String getDescription();
}
```

Service Oriented Architecture

- Developing applications with a well-defined, **published** API
- Typically thought of as Web Services
  - SOAP, REST, XML-RPC
- Can be POJOs executing in the same JVM
  - Applications communicate by sending messages to well-defined Java interfaces

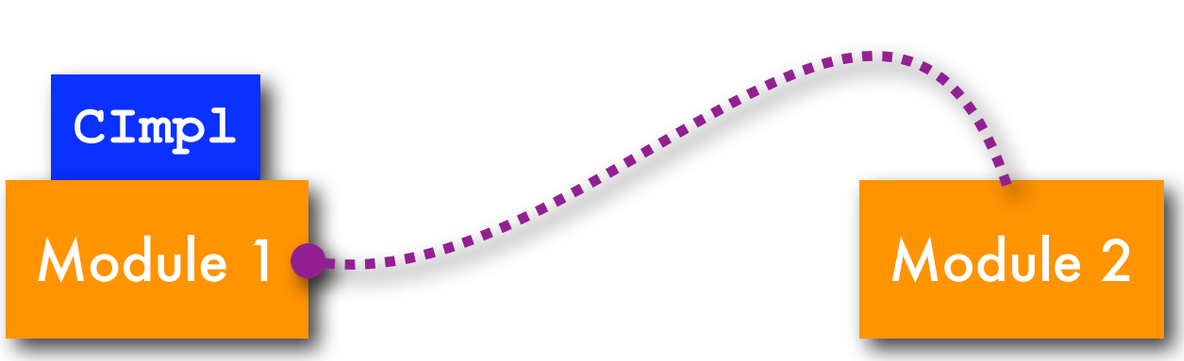
Let's look at a simple OSGi application

Unnecessary Coupling

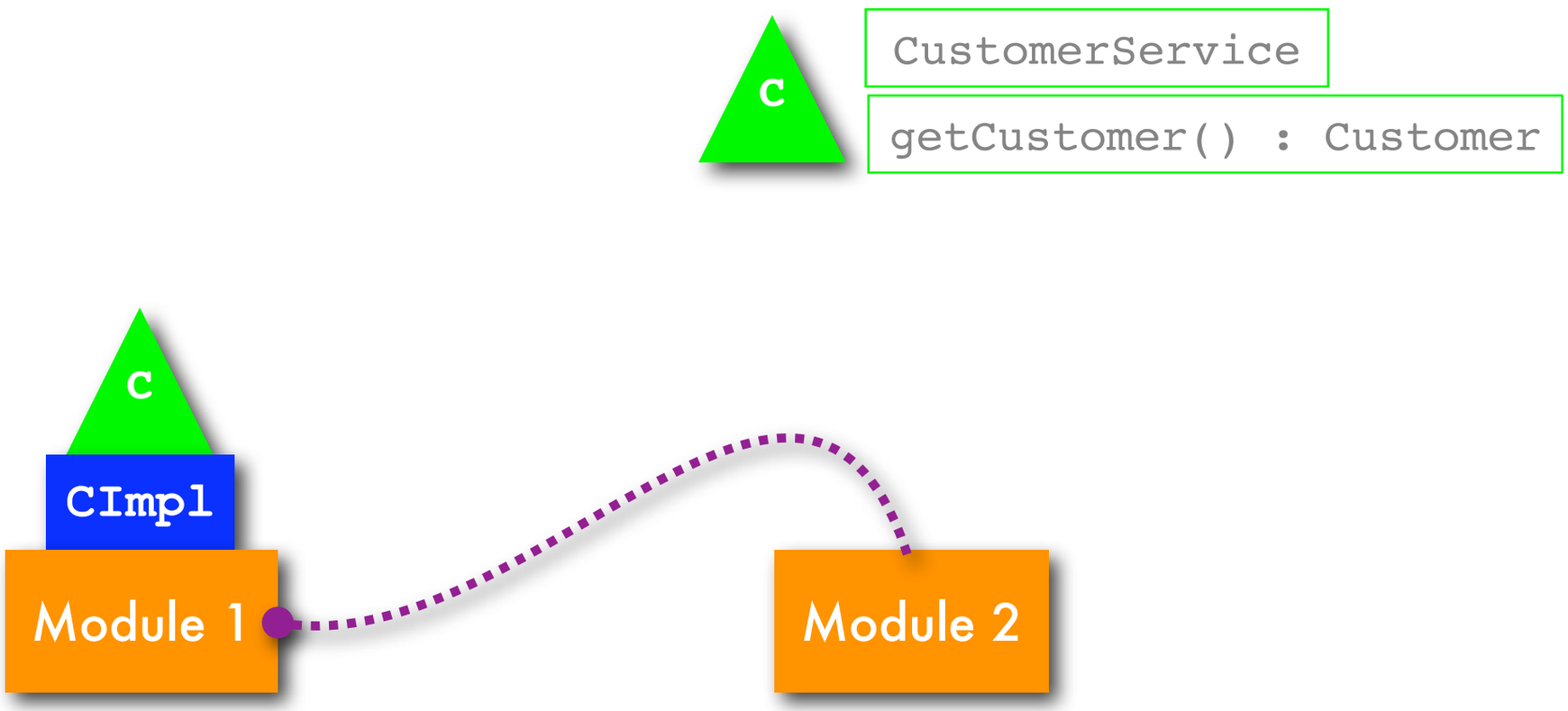
CImpl

CustomerService

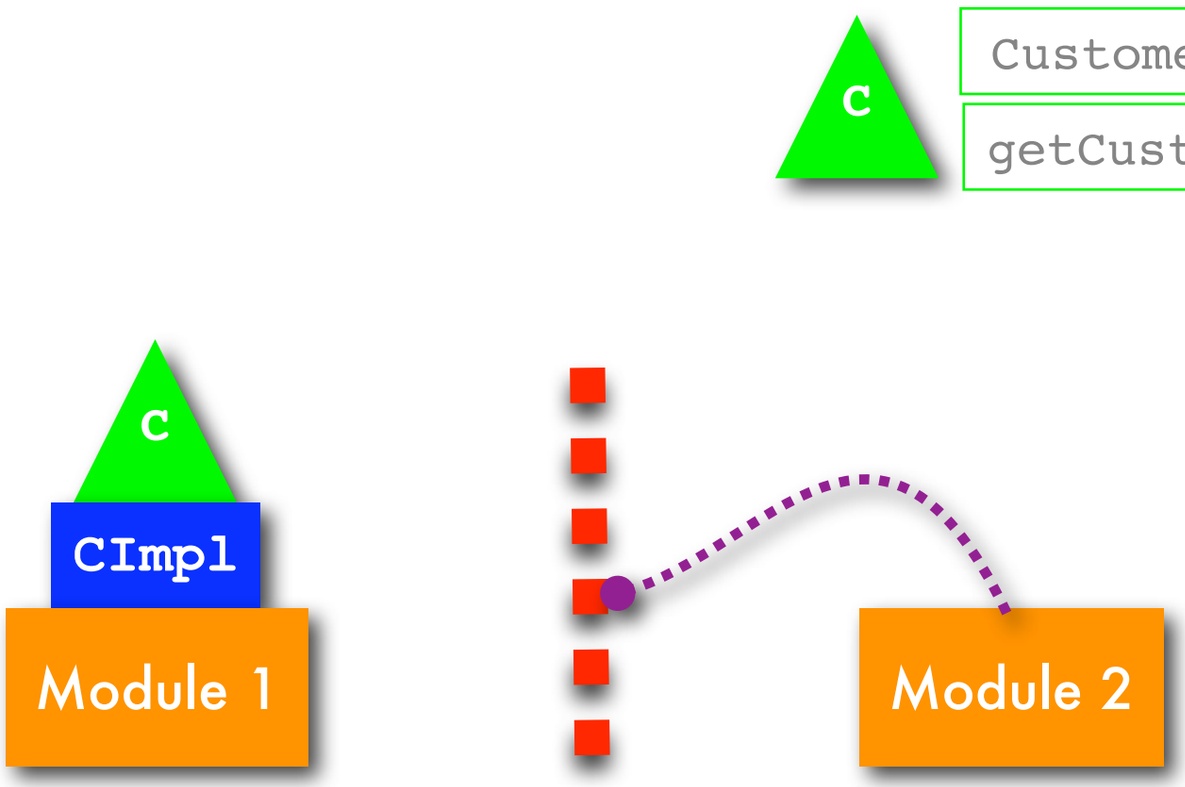
getCustomer() : Customer



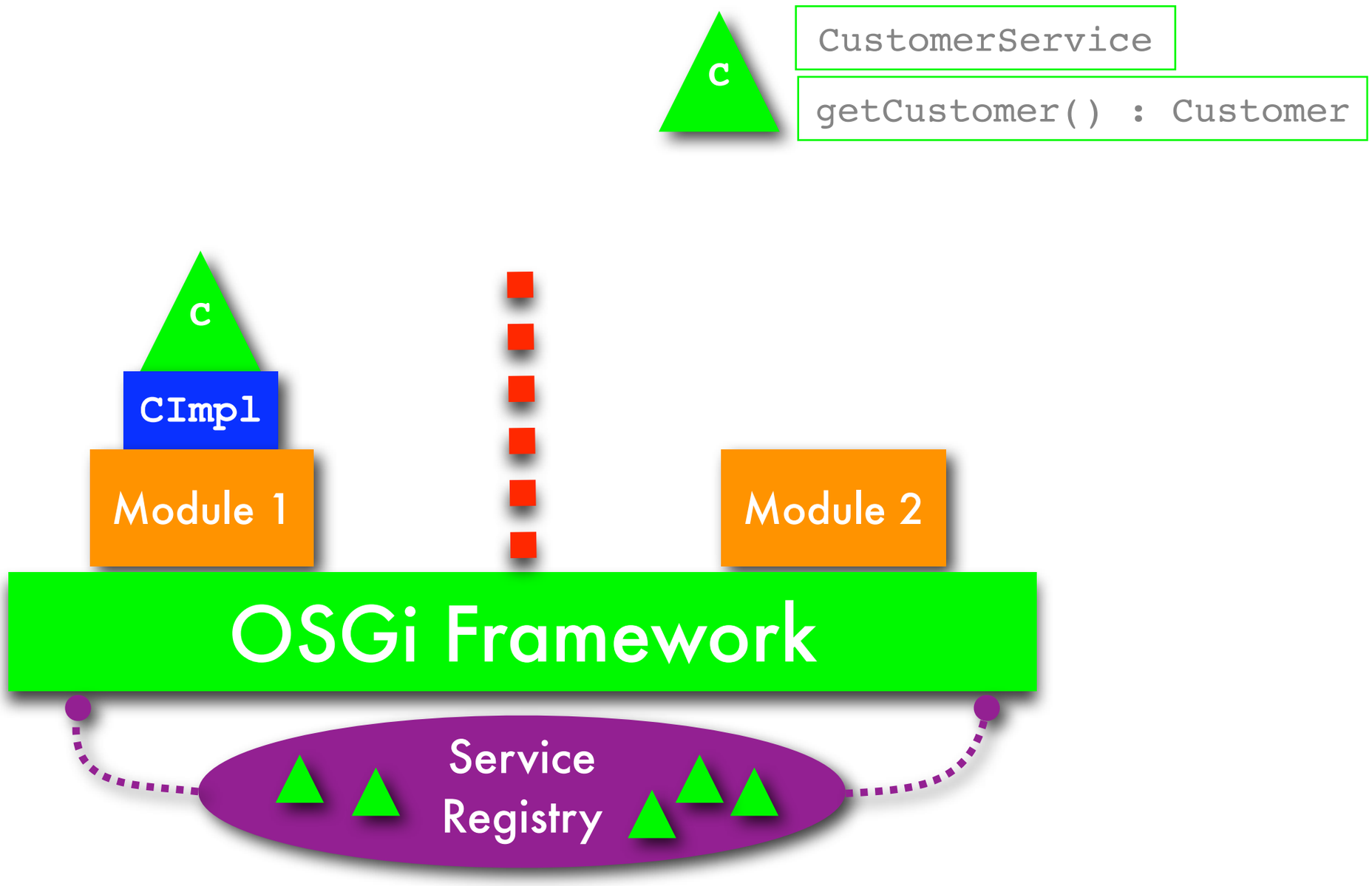
Extract An Interface



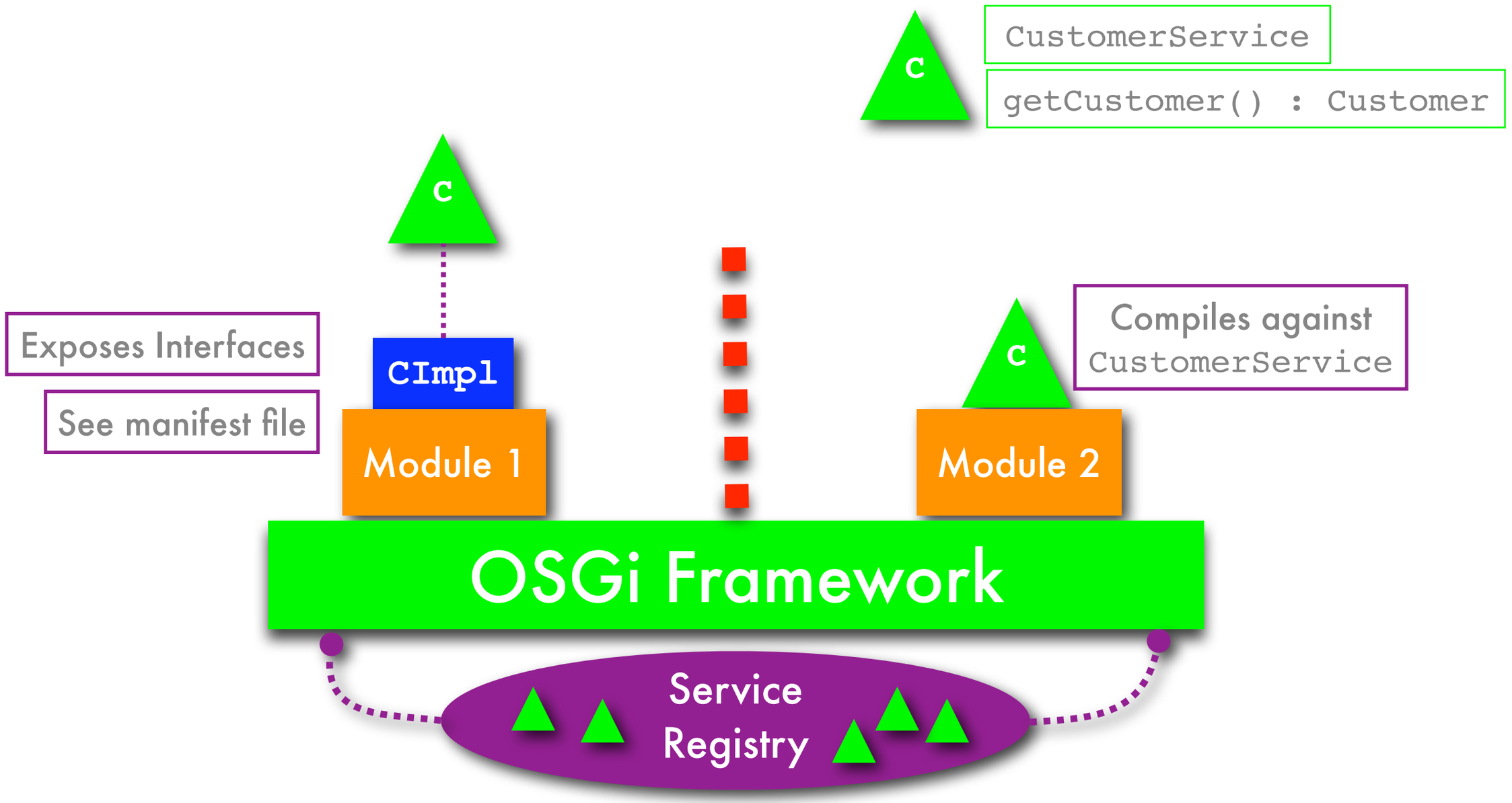
Remove Coupling



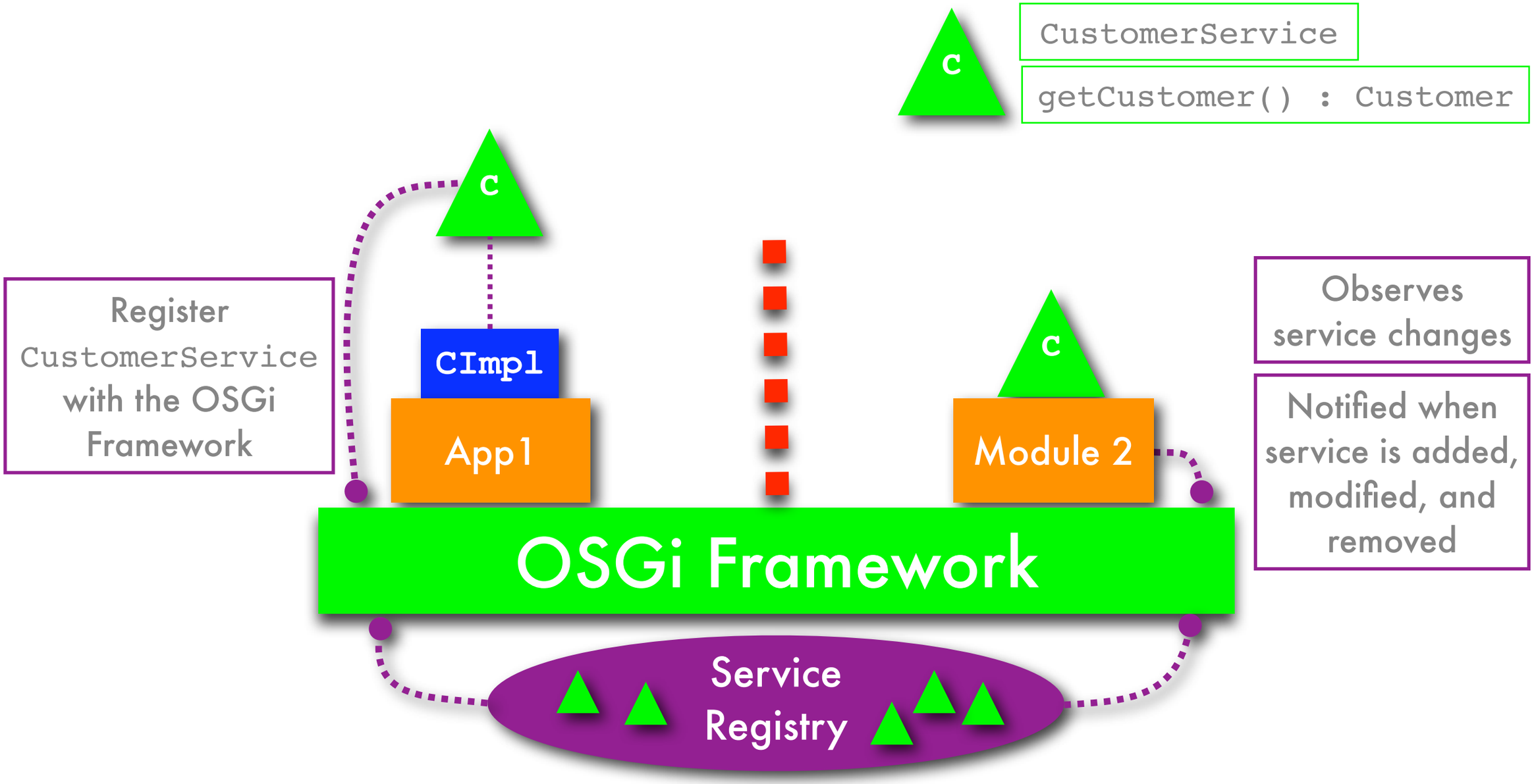
Add Service Registry



Expose Services



React To Service



OSGi Device

- A device is
  - anything that has access to a network AND
  - is capable of running an OSGi implementation
- Examples
  - phones, consumer electronics, PDAs
  - PCs, industrial computers
  - cars

We will focus our examples on the PC desktop

Bundle

- JAR file containing
  - Manifest file describing the bundle
    - We'll look at the OSGi Manifest headers soon
  - Application code (.*class* files)
    - Optional Java libraries as embedded JAR files
      - specified by a special OSGi manifest header
      - `Bundle-Classpath: .,lib/customer-services.jar`
- Optional native code

Bundle

- "Deployed" to an OSGi framework
- When "started" a bundle registers services with the service registry

- Bundles "should" only interact with other Bundles via registered services
  - See OSGi Service Architecture Overview

- Why?

Dynamism

- Communication via dynamically registered services
  - Promotes de-coupling
  - Promotes dynamism

"continuous change, activity, or progress"

Bundle Manifest

- Every OSGi Bundle (JAR) has a standard JAR Manifest file
  - Located in *META-INF/MANIFEST.MF*
- The OSGi specification adds custom headers that follows the Manifest format
  - *http://java.sun.com/j2se/1.4.2/docs/guide/jar/jar.html*
- Contains metadata about the bundle
  - Used by the OSGi framework to manage the bundle's life cycle, classpath, dependencies, etc

Example Manifest

META-INF/MANIFEST.MF

```
Manifest-Version: 1.0
Bundle-Name: shell
Bundle-Version: 1.0.0
Bundle-Description: OSGi Shell
Bundle-DocURL: http://www.briancoyner.com/shell.html
Bundle-Vendor: CoyTech
Bundle-Activator: com.briancoyner.shell.osgi.ShellActivator
Bundle-Category: Framework
Bundle-Classpath: .,lib/shell-core.jar,lib/shell-service.jar
Import-Package: org.osgi.framework
```

Bundle Manifest

META-INF/MANIFEST.MF

```
Manifest-Version: 1.0
Bundle-Name: shell
Bundle-Version: 1.0.0
Bundle-Description: OSGi Shell
Bundle-DocURL: http://www.briancoyner.com/shell.html
```

Attribute	Description
Bundle-Name	Short Name. No spaces
Bundle-Version	Version of the bundle. Free format.
Bundle-Description	Short description.
Bundle-DocURL	URL containing documentation about this bundle

Bundle Manifest

META-INF/MANIFEST.MF

```
Bundle-Activator: com.briancoyner.shell.osgi.ShellActivator
Bundle-Classpath: .,lib/shell-core.jar,lib/shell-service.jar
Import-Package: org.osgi.framework
Export-Package: com.briancoyner.shell.service
```

Attribute	Description
Bundle-Activator	The name of the class implementing the `BundleActivator` interface. The `BundleActivator` is called by the Framework to start and stop the bundle.
Bundle-Classpath	A CSV list of JAR file path names (inside the bundle). The '.' specifies the bundle itself.
Import-Package	CSV list of package names that **must be** imported.
Export-Package	CSV list of package names that can be exported.

Bundle Activator

- All bundles must have a
  - `org.osgi.framework.BundleActivator`
- Declared in the manifest file
  - *Bundle-Activator* Header
- Simple interface allowing the Framework to start and stop a bundle

```
public interface BundleActivator {
    void start(BundleContext context) throws Exception;
    void stop(BundleContext context) throws Exception;
}
```

Bundle Context

- The bundle context is a bundle's view into the Framework
- Allows a bundle to interact with the Framework
- There is a one-to-one relationship between a `Bundle` and a `BundleContext`
- Your code should **never** reference the framework implementation classes specific classes

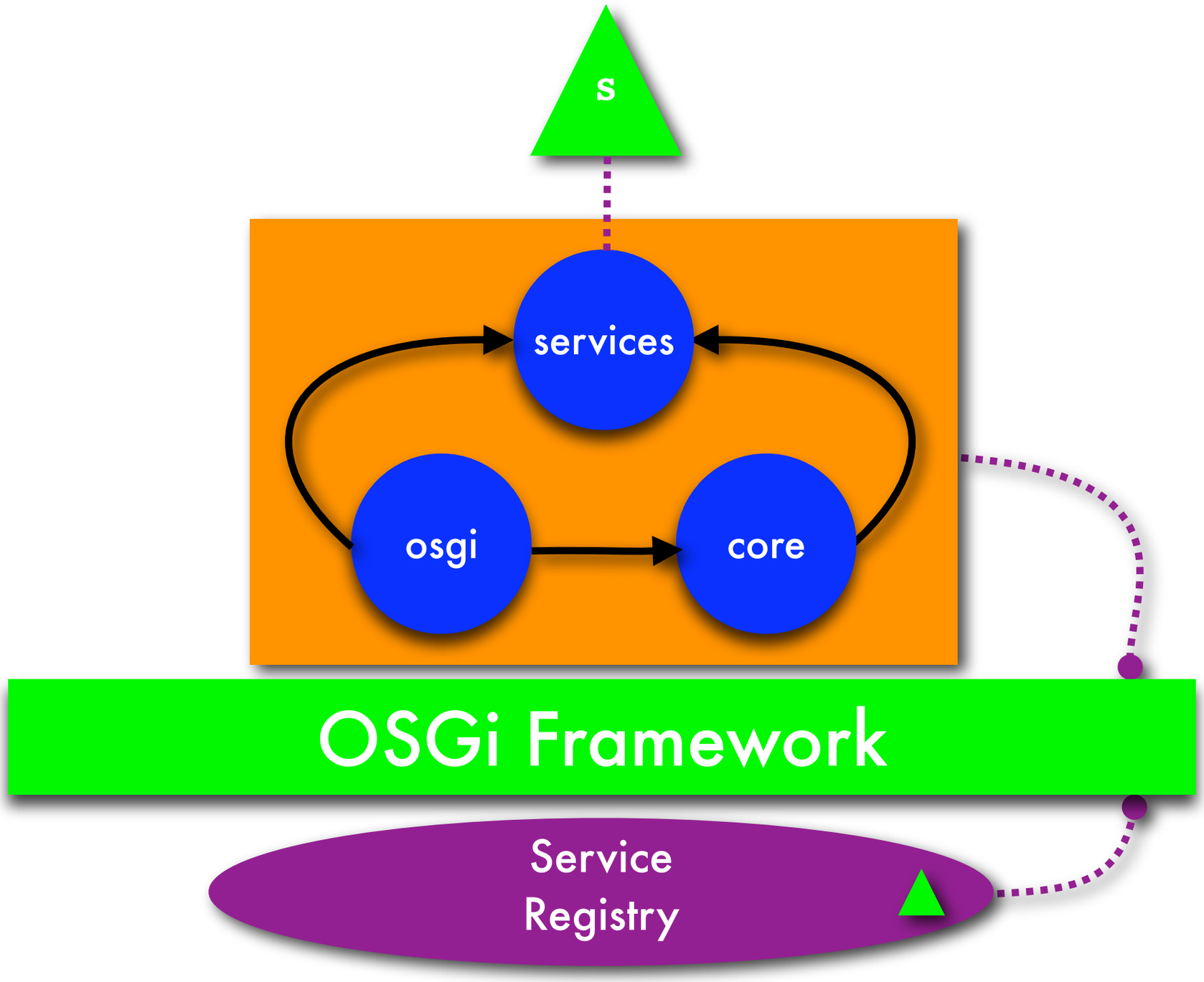
Bundle Context Behaviors

- A bundle can use the context to
  - Subscribe to events published by the Framework
  - Install new bundles
  - Get the bundles installed in the Framework
  - Register services with the Service Registry
  - Query for services in the Service Registry

Bundle Namespace

- The Framework loads each bundle in its own class loader
- What does this do?
  - Avoids naming conflicts between different versions of a class (libraries) loaded in different bundles
  - Enables sharing packages between bundles
  - Reduces the memory footprint
    - only need one version of a library loaded

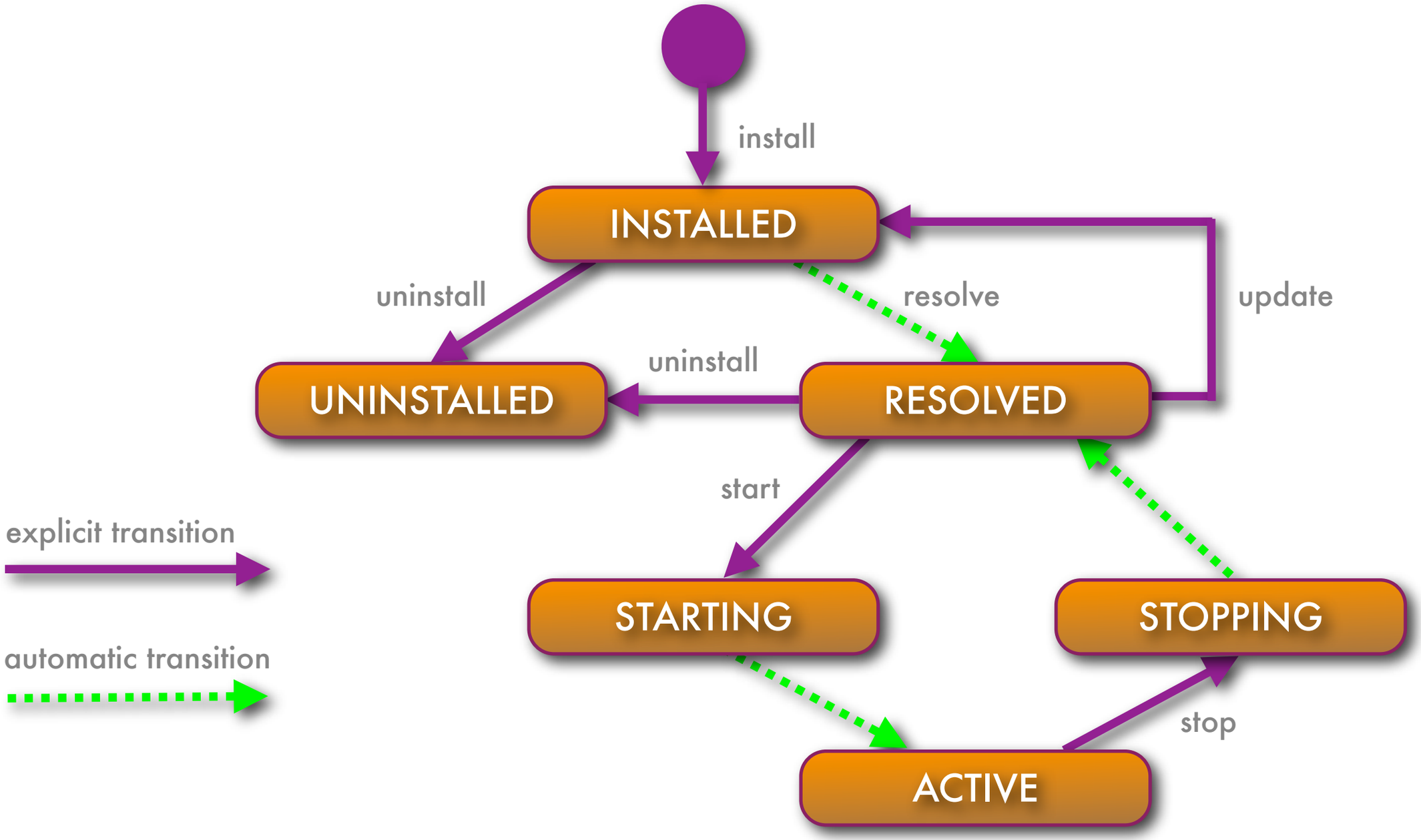
What's Inside?



Bundle States

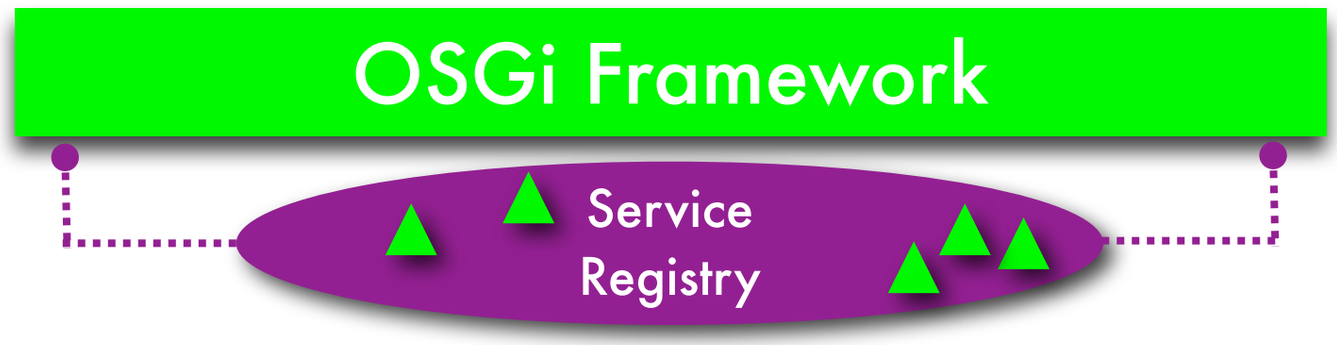
- INSTALLED (ready to start executing)
- RESOLVED
  - all classes the bundle needs have been found
- STARTING
  - inside the `BundleActivator.start()` method
- STOPPING
  - inside the `BundleActivator.stop()` method
- ACTIVE
- UNINSTALLED (stick a fork in it)

State Transitions



Service Registration

- The Framework provides a shared registry for bundles



- Here is how to register a `Page` service

```
public class AddressBookActivator implements BundleActivator {

    public void start(BundleContext context) throws Exception {
         context.registerService(Page.class.getName(),
                  new AddressBookPage(), null);
    }
}
```

Service Tracking

- When a service is registered with the Framework, the Framework fires a `ServiceEvent` to all registered listeners
- Here is a service listener

```
public class PageServiceListener implements ServiceListener {

    public void serviceChanged(ServiceEvent event) {
        if (ServiceEvent.REGISTERED == event.getType() {
        } else if (ServiceEvent.MODIFIED == event.getType() {
        } else if (ServiceEvent.UNREGISTERING == event.getType() {
        }
    }
}
```

Listening For Services

- Here is how to hook up the `PageServiceListener`

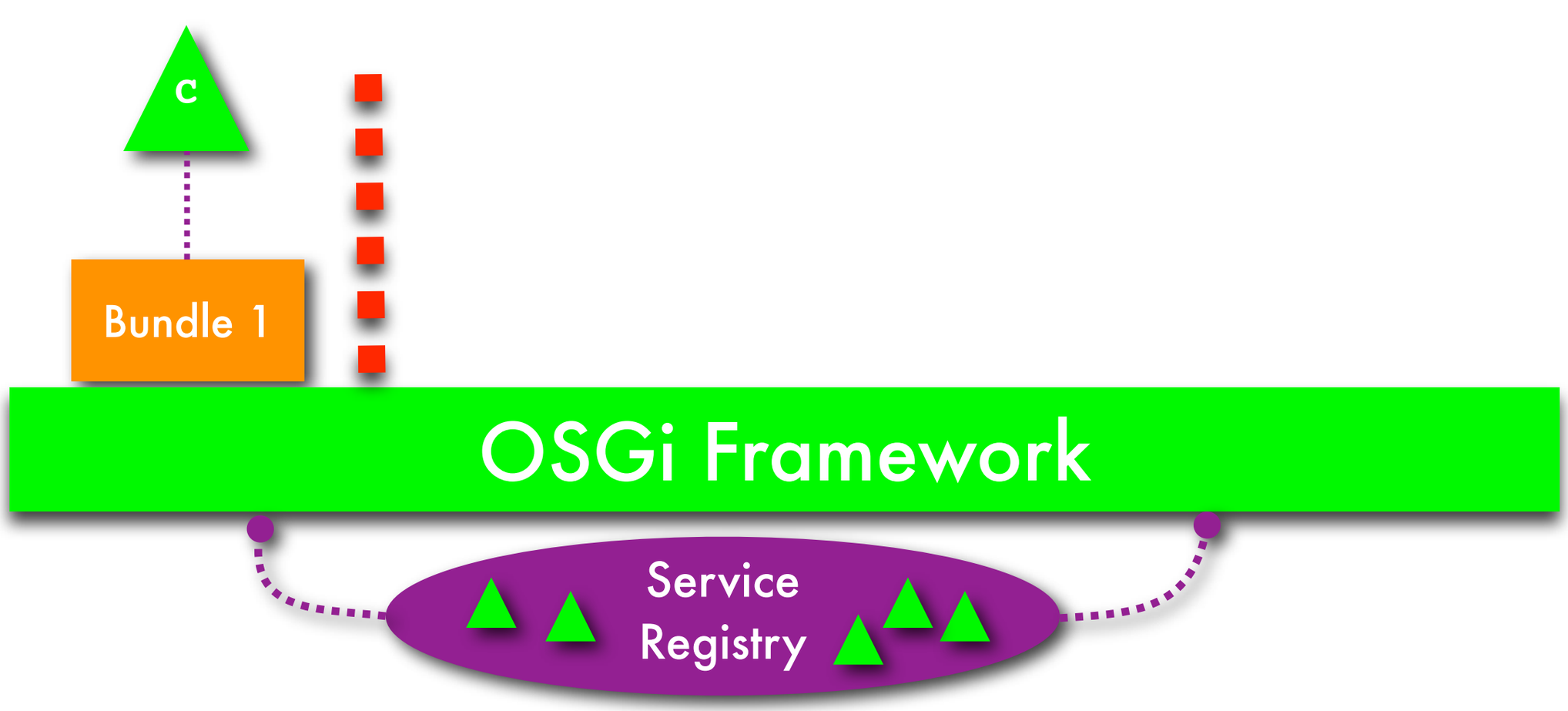
```
public class ShellActivator implements BundleActivator {

    public void start(BundleContext context) throws Exception {
        ServiceListener listener = new PageServiceListener();
        String ldapStyleFilter = “(objectclass=” +
                Page.class.getName() + “)”;
        bundleContext.addServiceListener(listener,
                ldapStyleFilter);
    }
}
```

Dynamic Nature Of OSGi

- The dynamic nature of OSGi makes service registration difficult to manage
- Why?
  - We cannot rely on the availability of service at any given time
  - Our application must be able to handle a highly dynamic environment
- Of course, things work fine if the bundle listening for service events is already active

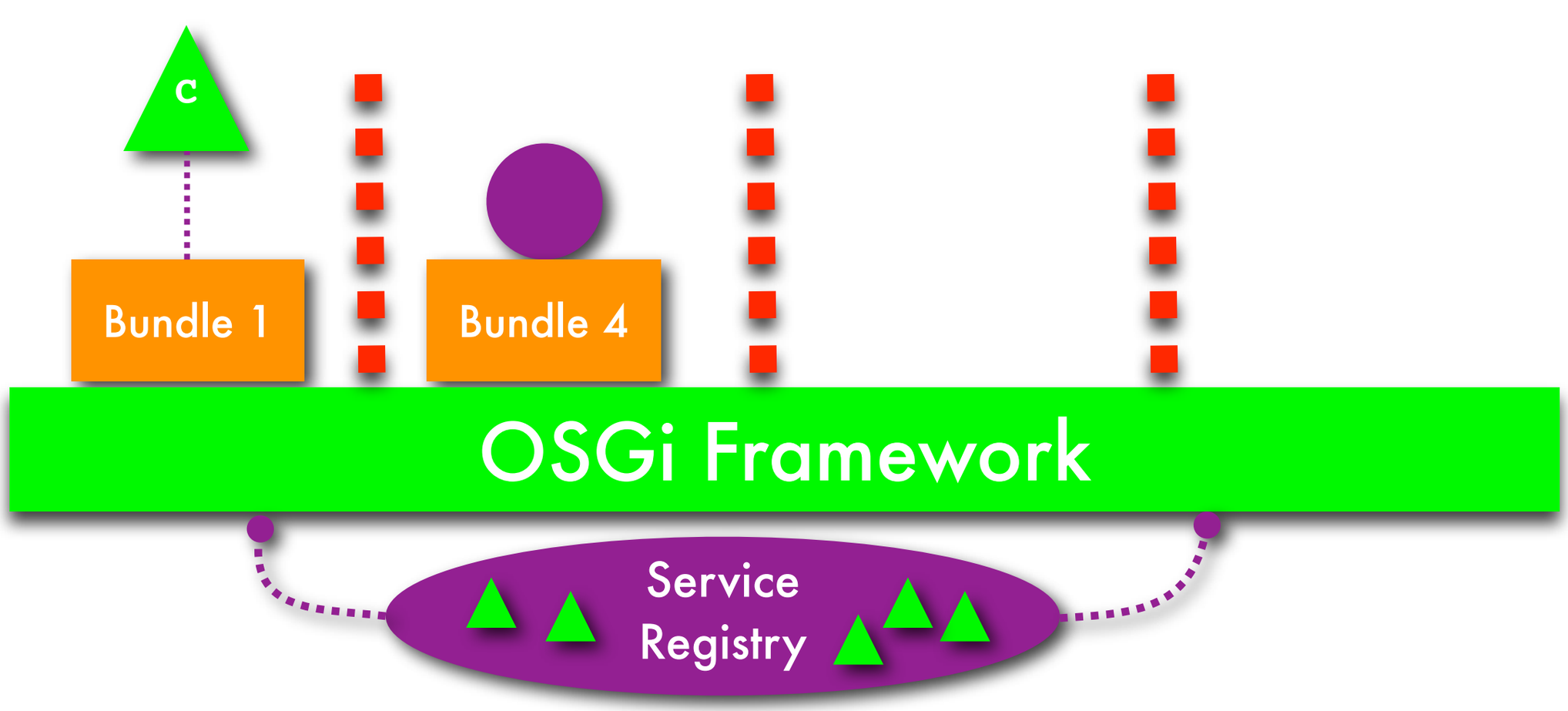
Expose An Interface

- Bundle 1 exposes an interface "C" to other bundles through a "services" JAR



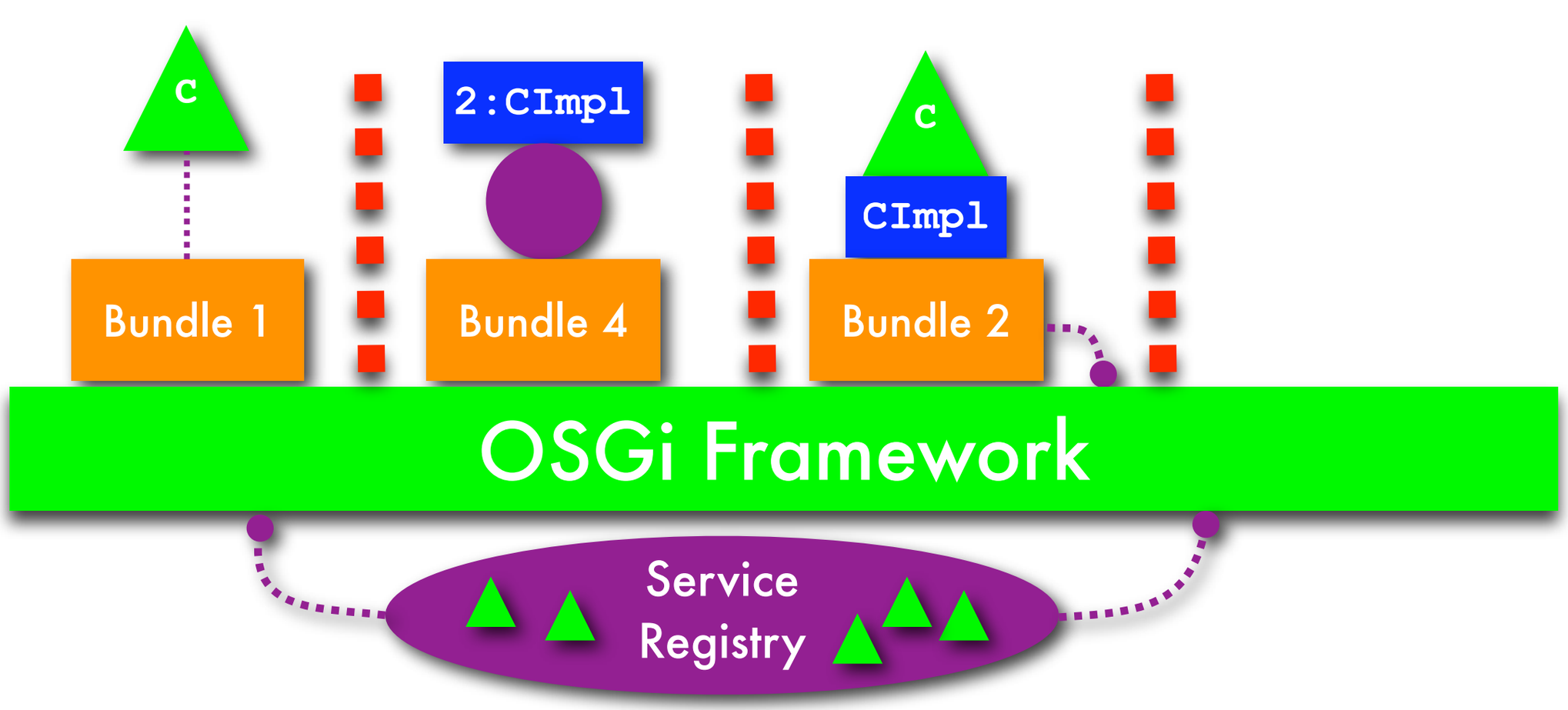
Bundle Listens For Events

- Bundle 4, our service listener, is active



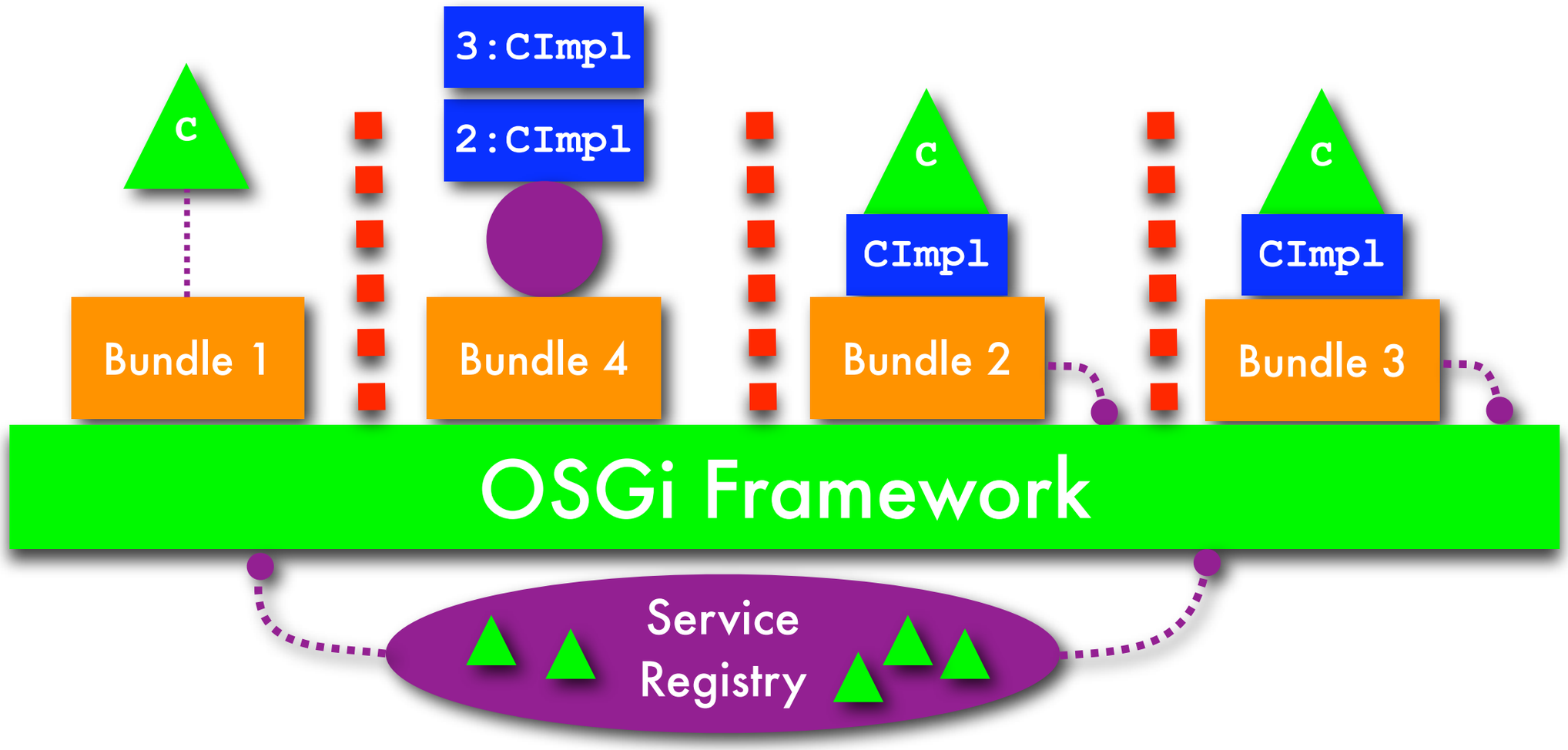
Dynamic Nature Of OSGi

- Bundle 2 starts and registers an implementation of "C"



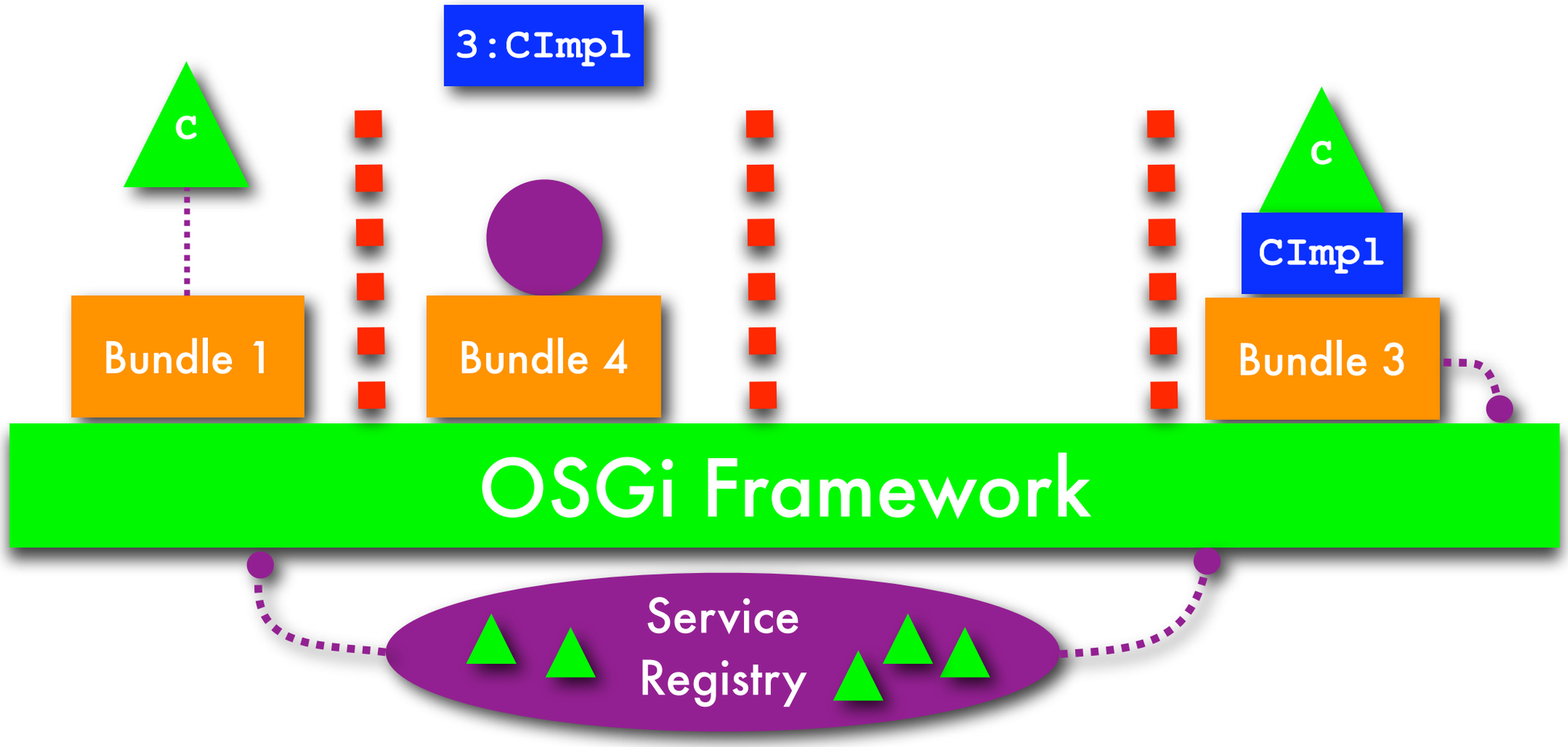
Dynamic Nature Of OSGi

- Bundle 3 starts and registers an implementation of "C"



Stop A Bundle

- Bundle 2 stops and its service is removed
- Bundle 4 receives the service event

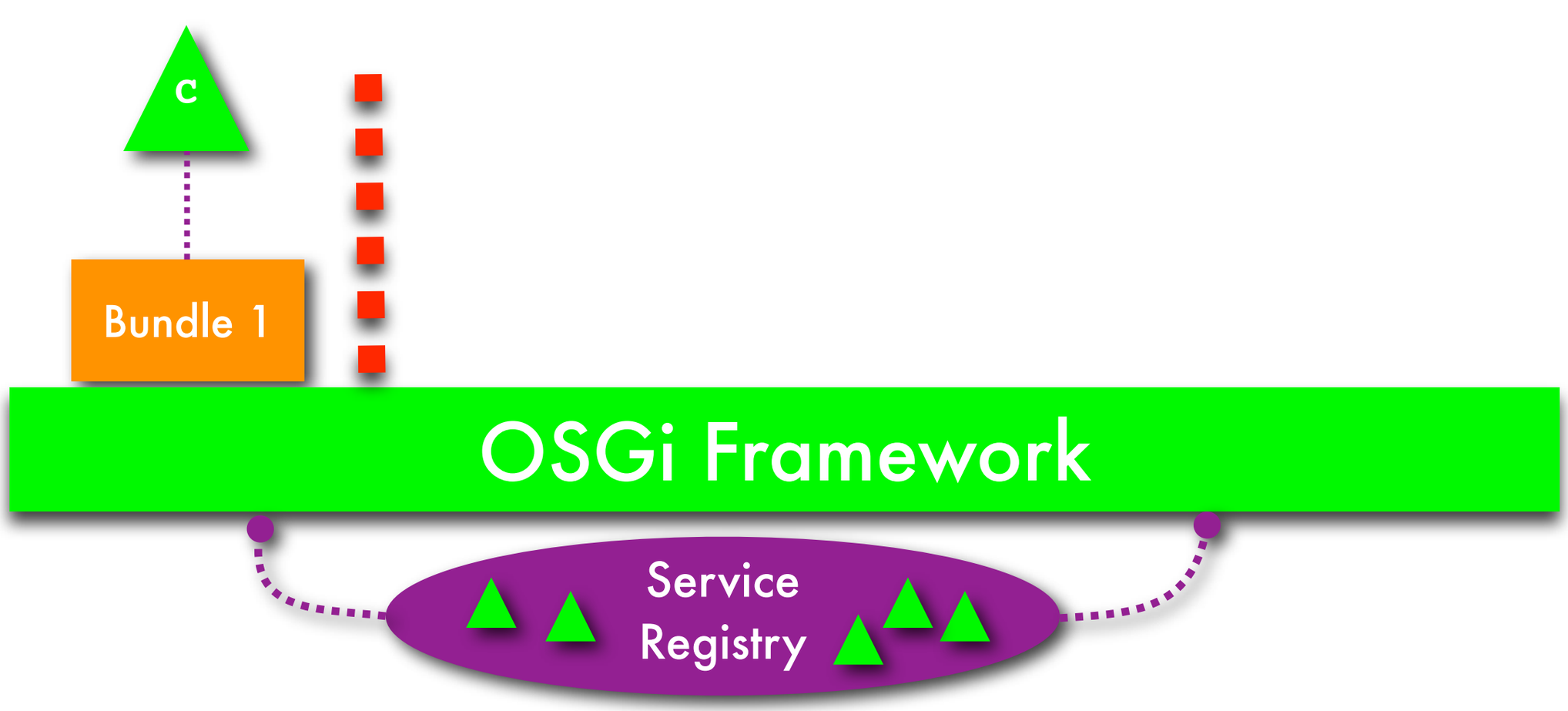


Dynamic Nature Of OSGi

- Service registration becomes more interesting when listeners are not active when a service is registered

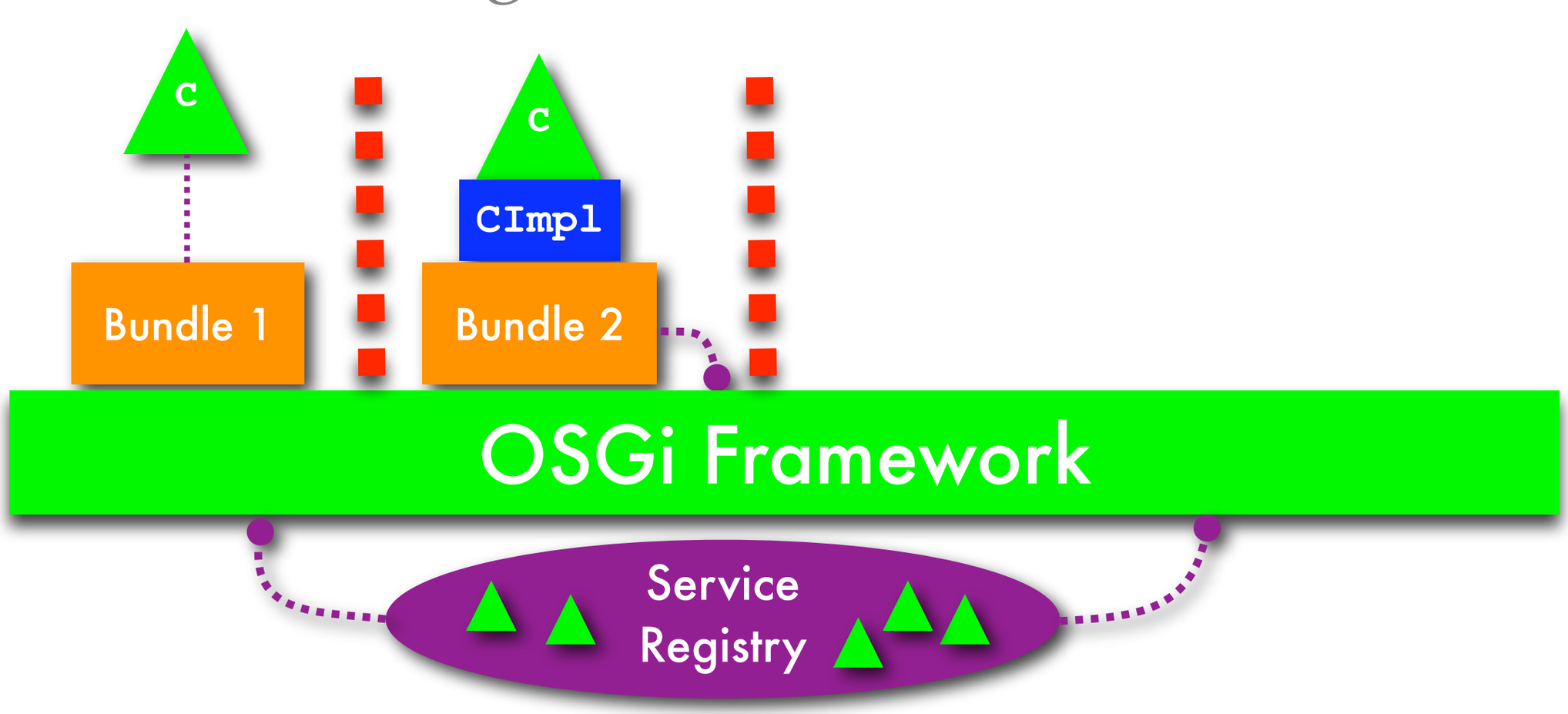
Expose An Interface

- Bundle 1 exposes an interface "C" to other bundles through a "services" JAR



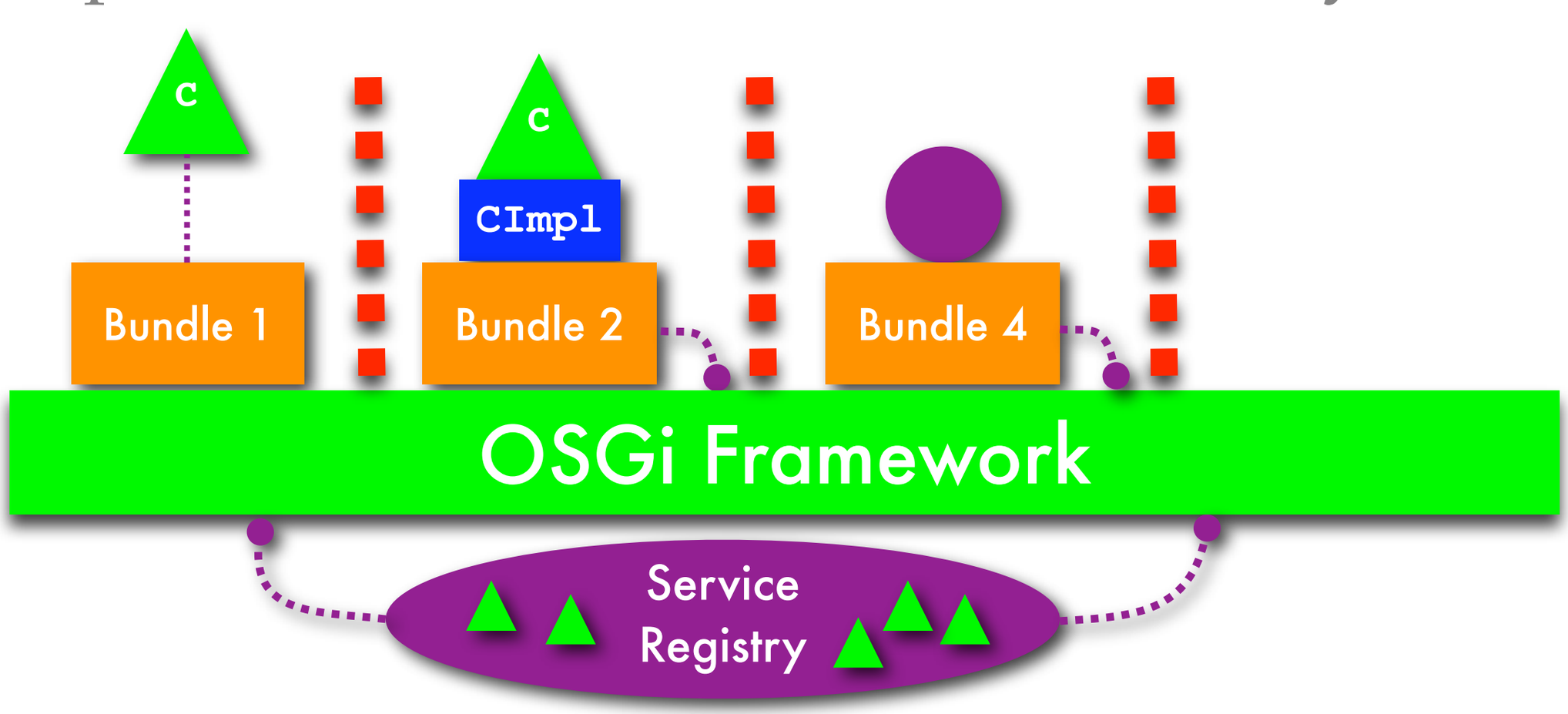
Register Services

- Bundle 2 starts and registers an implementation of "C"
- Who's listening? No one!



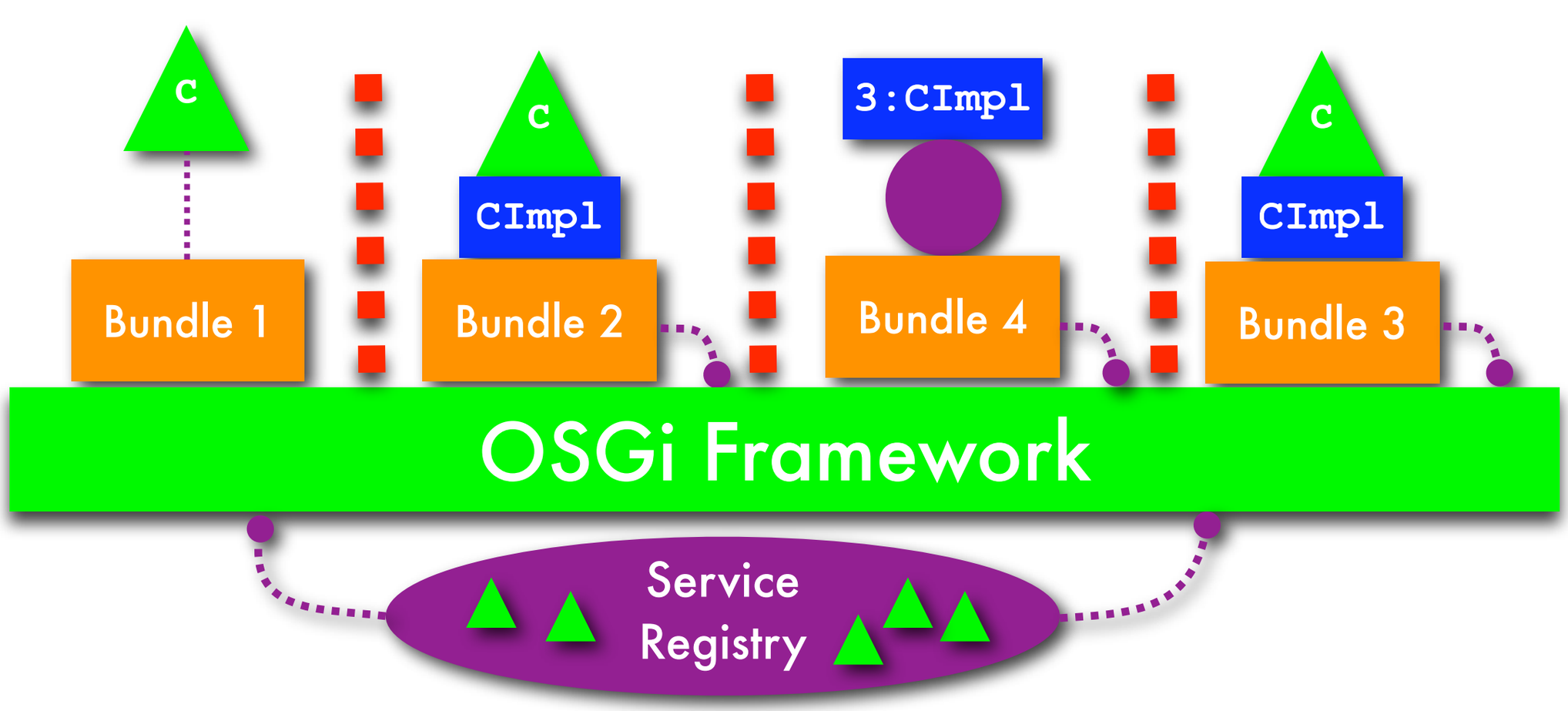
Services Not Delivered

- Bundle 4 is our "C" service listener
- He started after Bundle 2 registered its "C" implementation and did not receive any events



Almost There…

- Bundle 4 is our "C" service listener
- Bundle 4 receives a service event from Bundle 3



Querying For Services

- A possible solution is to query for services your bundle is interested in when the bundle starts

See next slide for example code

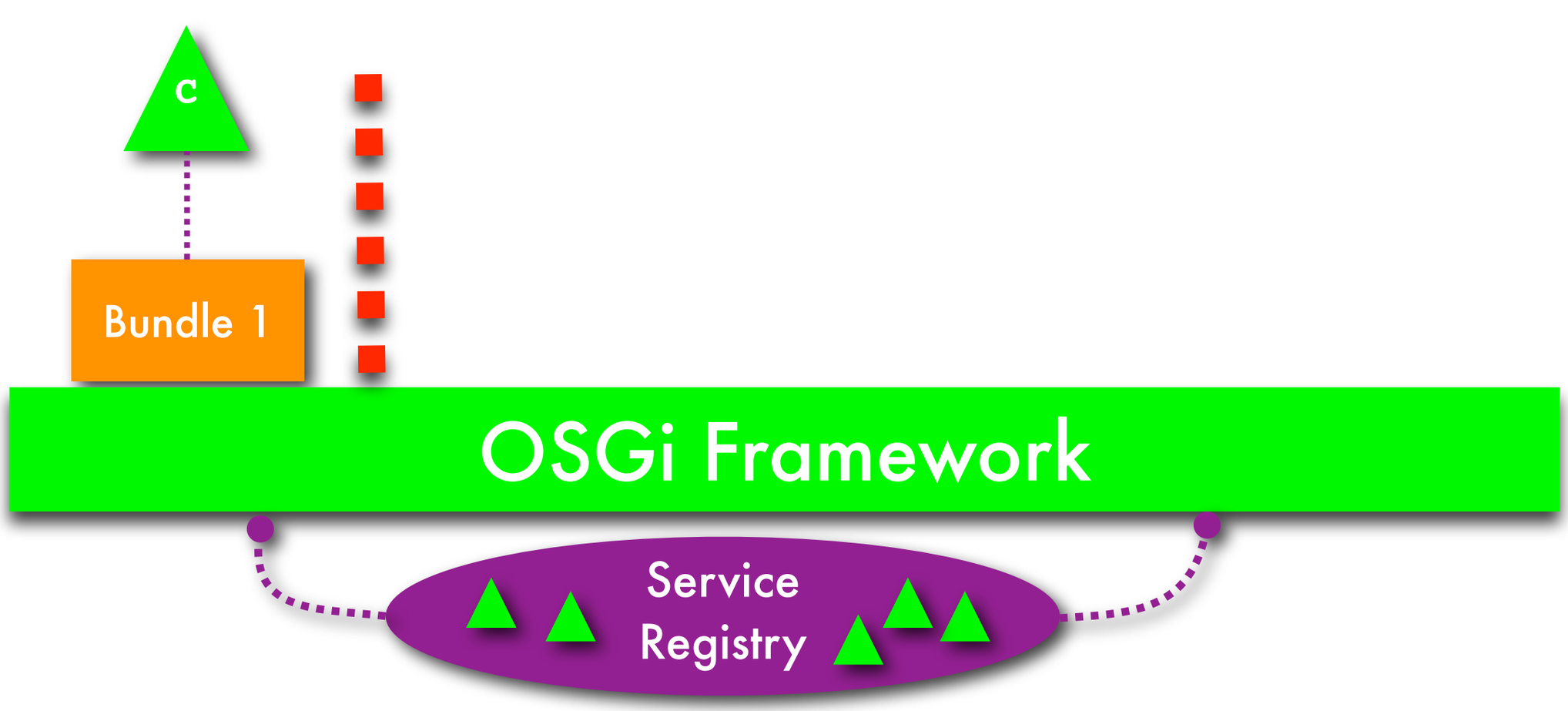
- We'll see that the solution requires too much work and is error prone
- We'll see how to *easily* track services using an OSGi utility

Querying For Services

```
public class ShellActivator implements BundleActivator {

    public void start(BundleContext context) throws Exception {
        ServiceListener listener = new PageServiceListener();
        String ldapStyleFilter = "(objectclass=" +
                Page.class.getName() + ")";
        bundleContext.addServiceListener(listener, ldapStyleFilter);

        String className = null;
        // match all services against filter
        ServiceReference[] references = context.getServiceReferences(
                className, ldapStyleFilter);
        if (references != null) {
            for (int i = 0; i < references.length; i++) {
                listener.serviceChanged(new ServiceEvent.REGISTERED,
                        references[i]);
            }
        }
    }
}
```

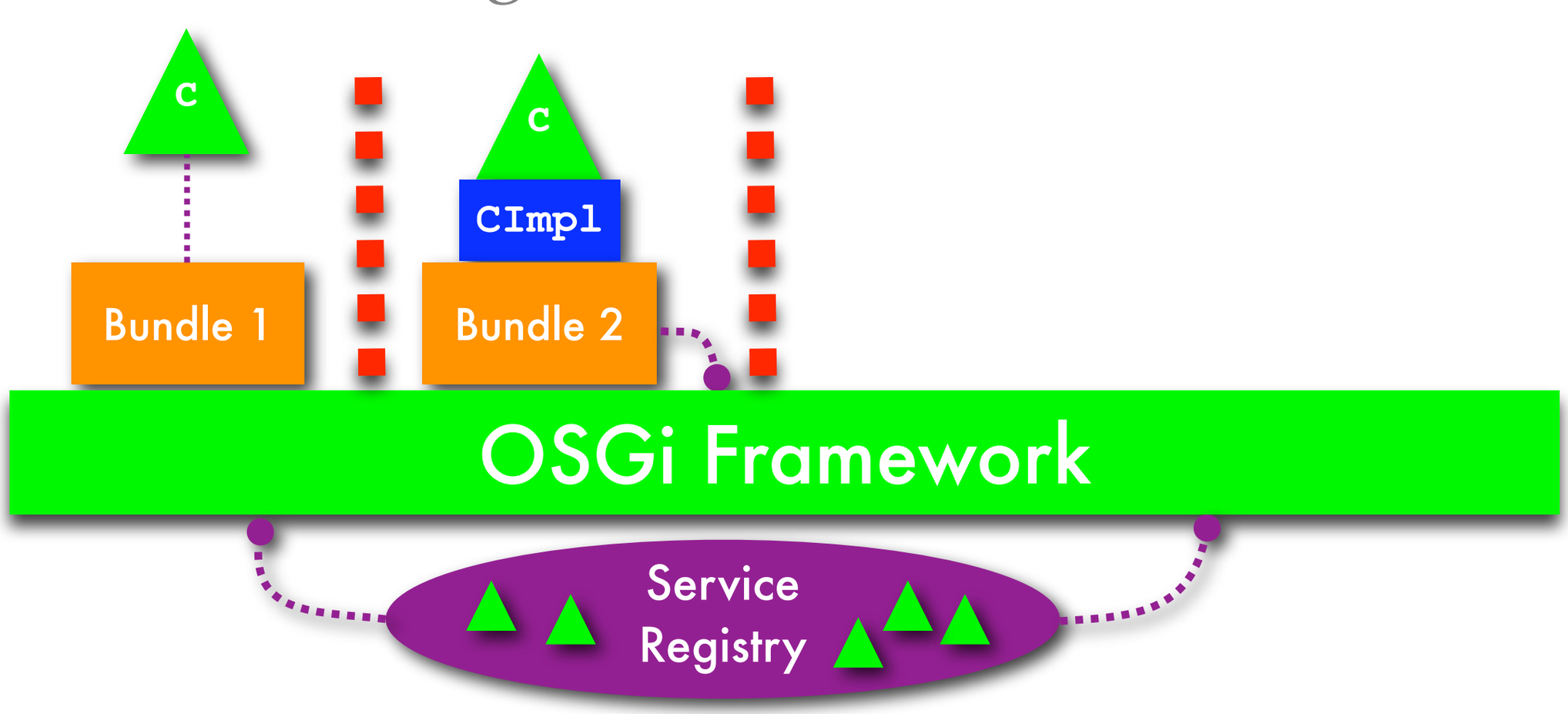
Expose An Interface

- Bundle 1 exposes an interface "C" to other bundles through a "services" JAR



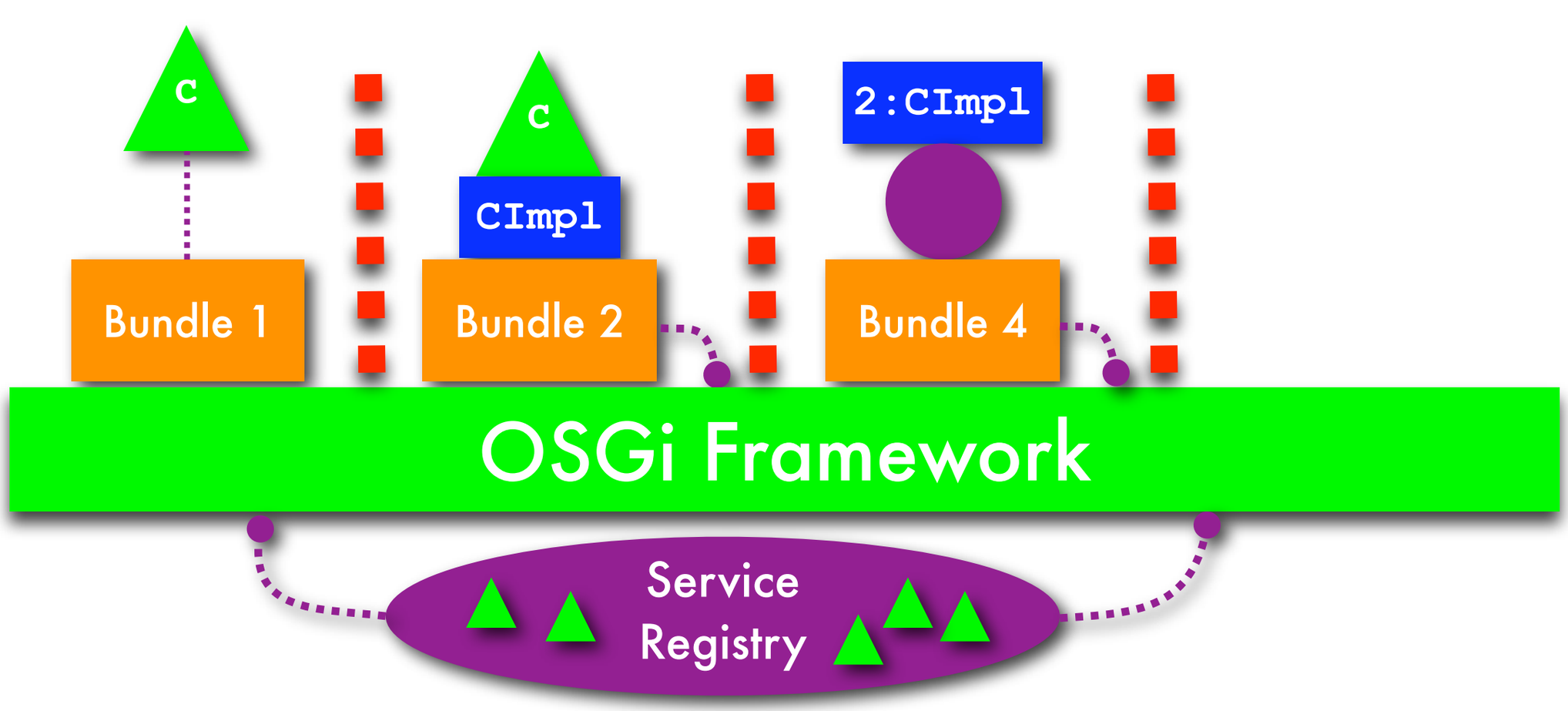
Register Service

- Bundle 2 starts and register an implementation of "C"
- Who's listening? No one!



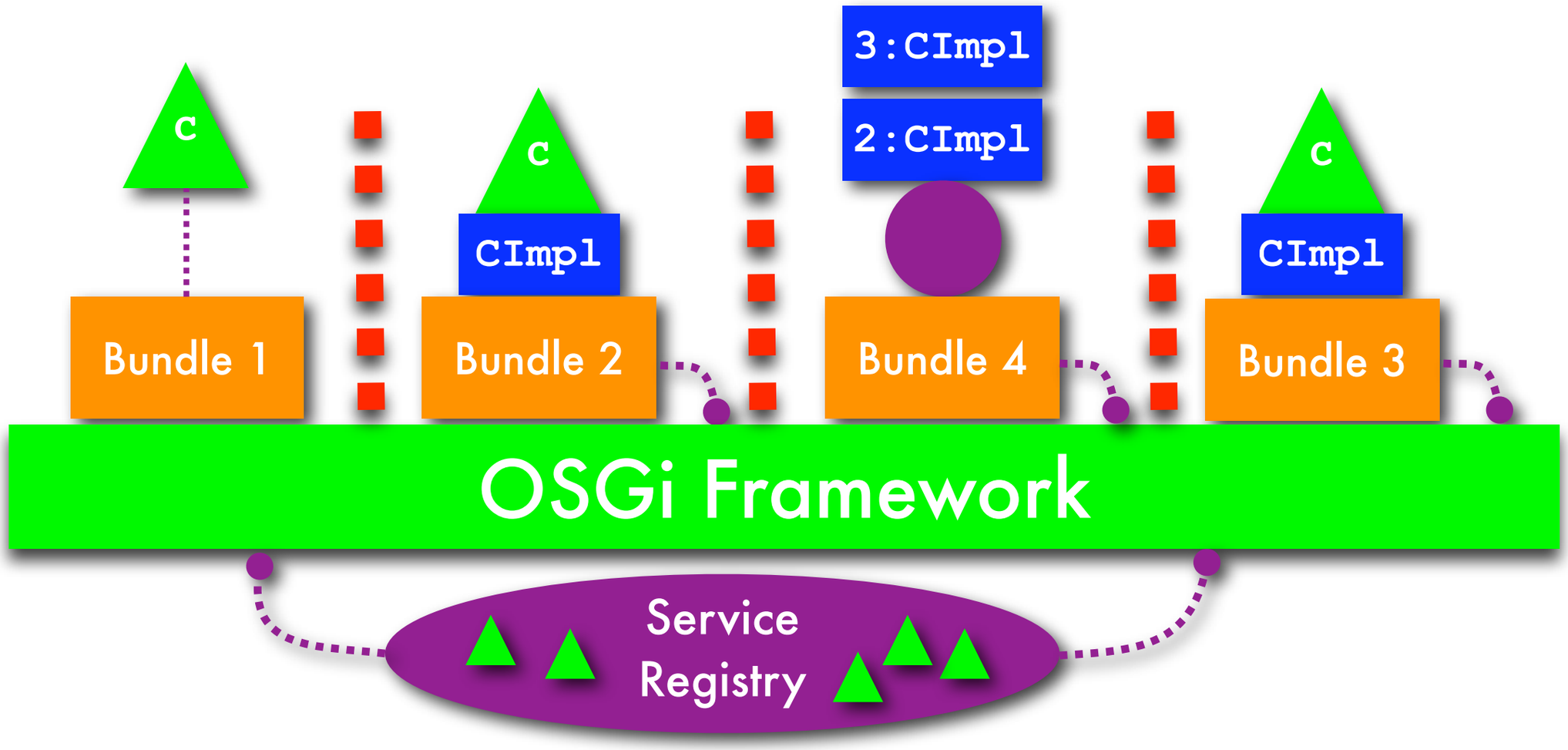
Query The Registry

- Bundle 4 is our "C" service listener
- He queries the service registry for "C"



All Is Well

- Bundle 3 starts and registers an implementation of "C"



OSGi Service Tracker

- Manually tracking services is *too hard*
- Luckily there is an OSGi utility called
  - `org.osgi.util.tracker.ServiceTracker`
- The service tracker utility simplifies listening for services from the Framework's service registry
- The service tracker takes care of keeping track of services as they come and go

Service Tracker Example

```
public class ShellActivator implements BundleActivator {

    public void start(BundleContext context) throws Exception {

        // we could use a filter or a class name
        String serviceName = Page.class.getName();

        ServiceTrackerCustomizer serviceStrategy = ...;

        ServiceTracker tracker = new ServiceTracker(context,
                serviceName, serviceStrategy);

        // start tracking... call close() to stop tracking
        tracker.open();
    }
}
```

Service Tracker Example

```
public class PageServiceTrackerCustomizer
        implements ServiceTrackerCustomizer {

    private BundleContext context;

    public PageServiceTrackerCustomizer(BundleContext context) {
        this.context = context;
    }

    public Object addingService(ServiceReference reference) {
        return context.getService(reference);
    }

    public void modifiedService(ServiceReference ref, Object service) {
    }

    public void removedService(ServiceReference ref, Object service) {
    }
}
```

OSGi Services

- OSGi R3 defines numerous service specifications
- The core services are
  - Package Admin
    - manage exported dependencies
  - Start Level
    - manage the order of starting and stopping bundles
  - Permission Admin

The Splash bundle (part of demo) utilizes the start level service and Framework events

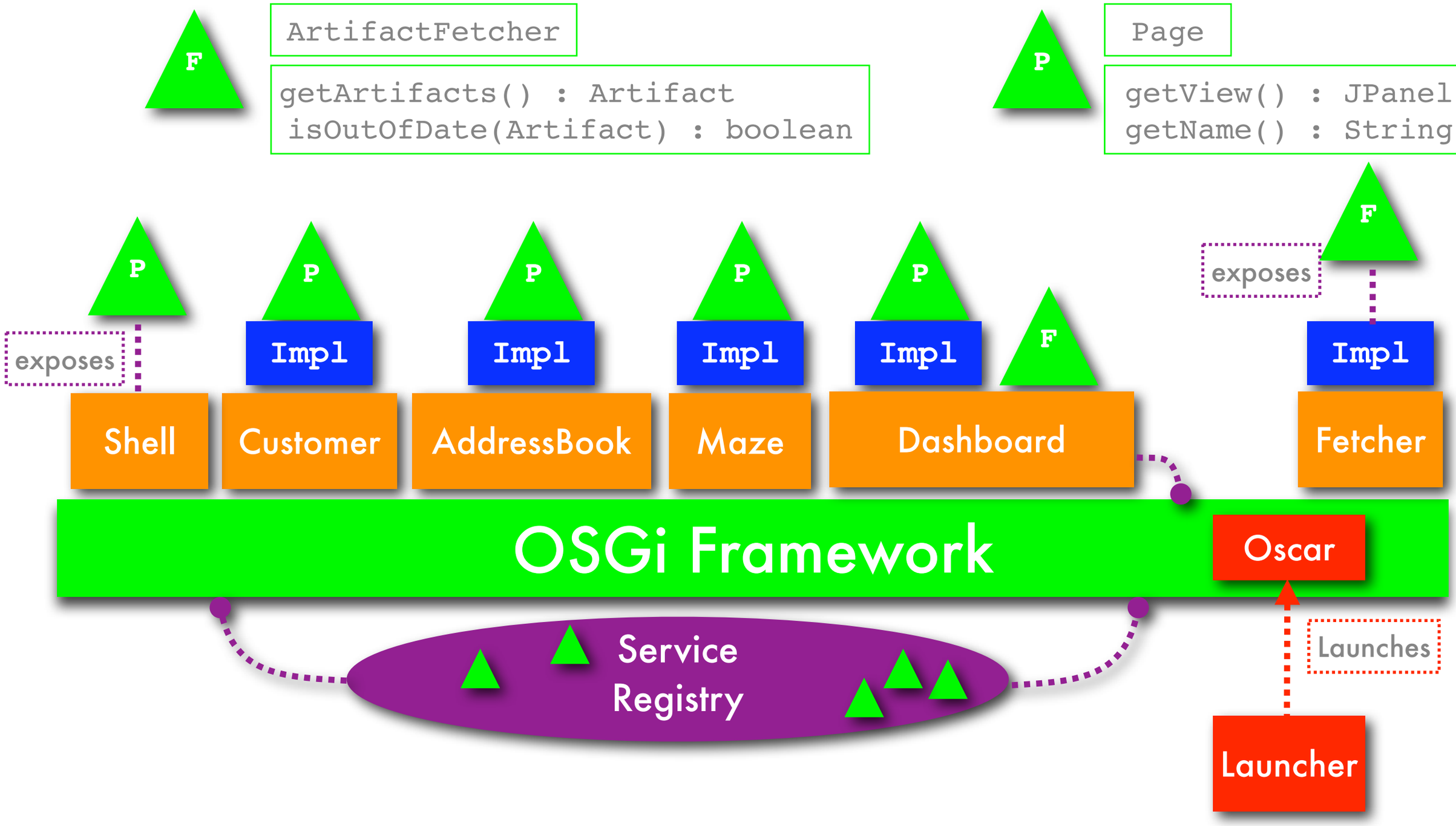
OSGi Services

- URL Handlers
- Log
- Configuration Admin
- HTTP
- Preferences
- Wire Admin
- Service Tracker
- Metatype

OSGi Services

- Jini Driver
- UPnP
- XML Parser
- IO Connector
- User Admin
- Device Access
- Measurement and State
- Position

Example Application



The Demo

- My application demonstrates
  - Dynamically installing bundles to a running "Shell"
  - Dynamically updating bundles that are out-of-date
  - Using different versions of a library
  - Using the OSGi start-level service to display a splash screen

There are bugs