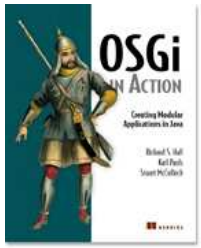
akquinet



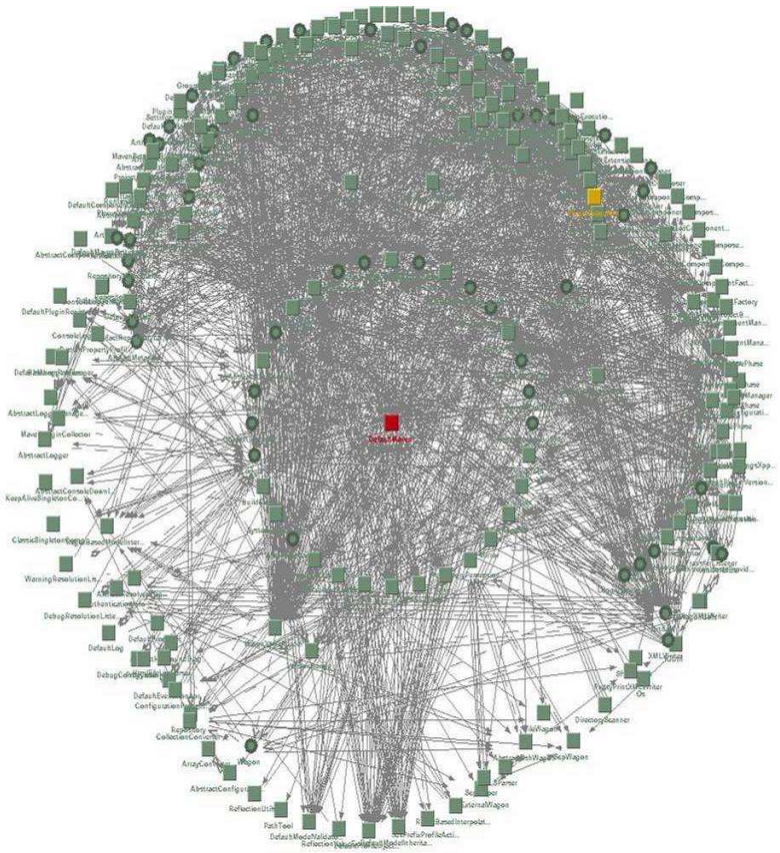
OSGi in Action

Karl Pauls karl.pauls@akquinet.de
Clement Escoffier clement.escoffier@akquinet.de

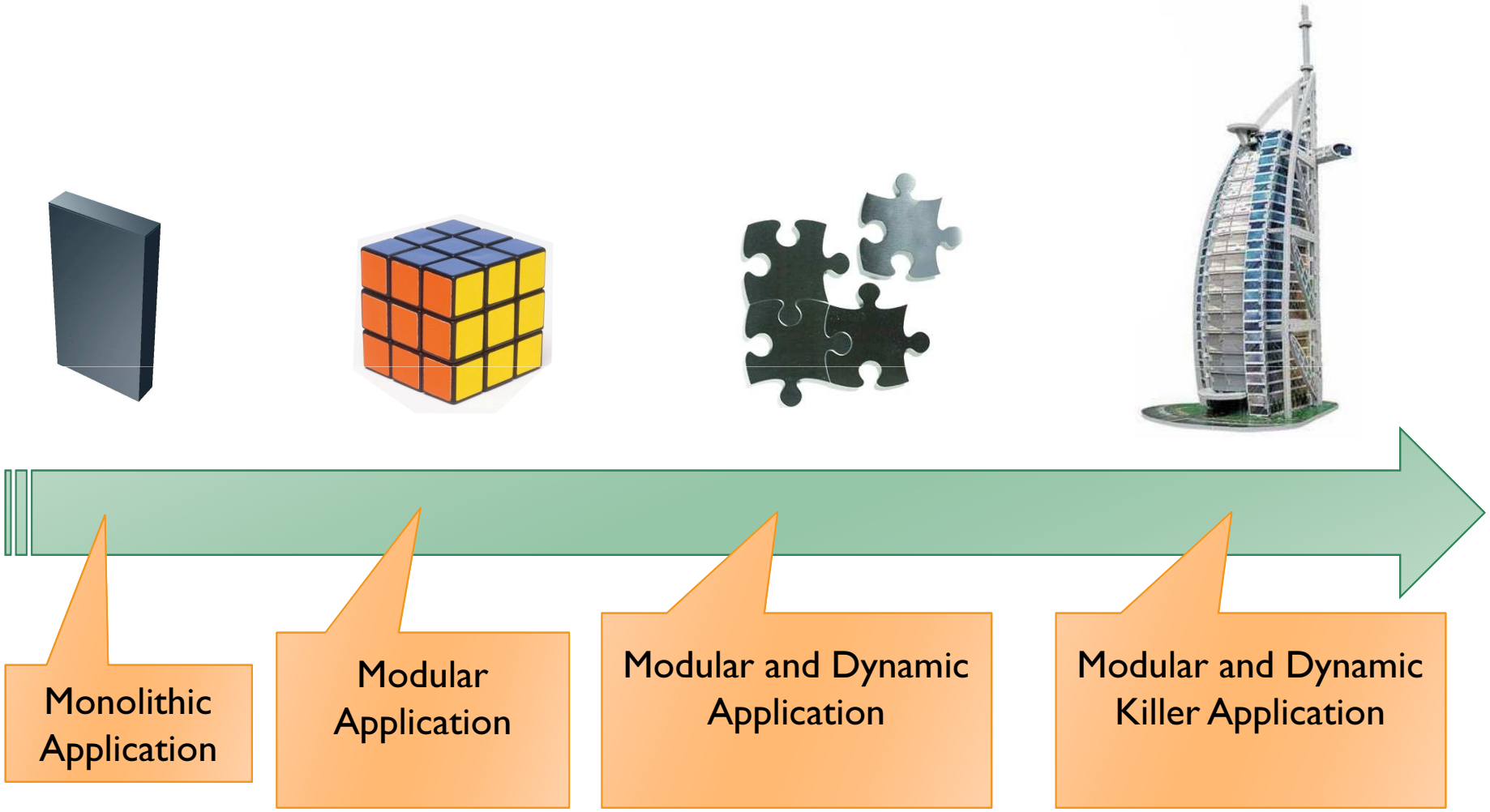
Why OSGi ?

Limits of Object-Oriented Programming (OOP)

- Coupling severely limits reusability
  - Using a generic object, can drag in a large number of other objects
- Creates overly large systems after a certain complexity is reached
- Flexibility must be built in by the programmer
  - Plug-in architectures
  - Factories, Dependency Injection



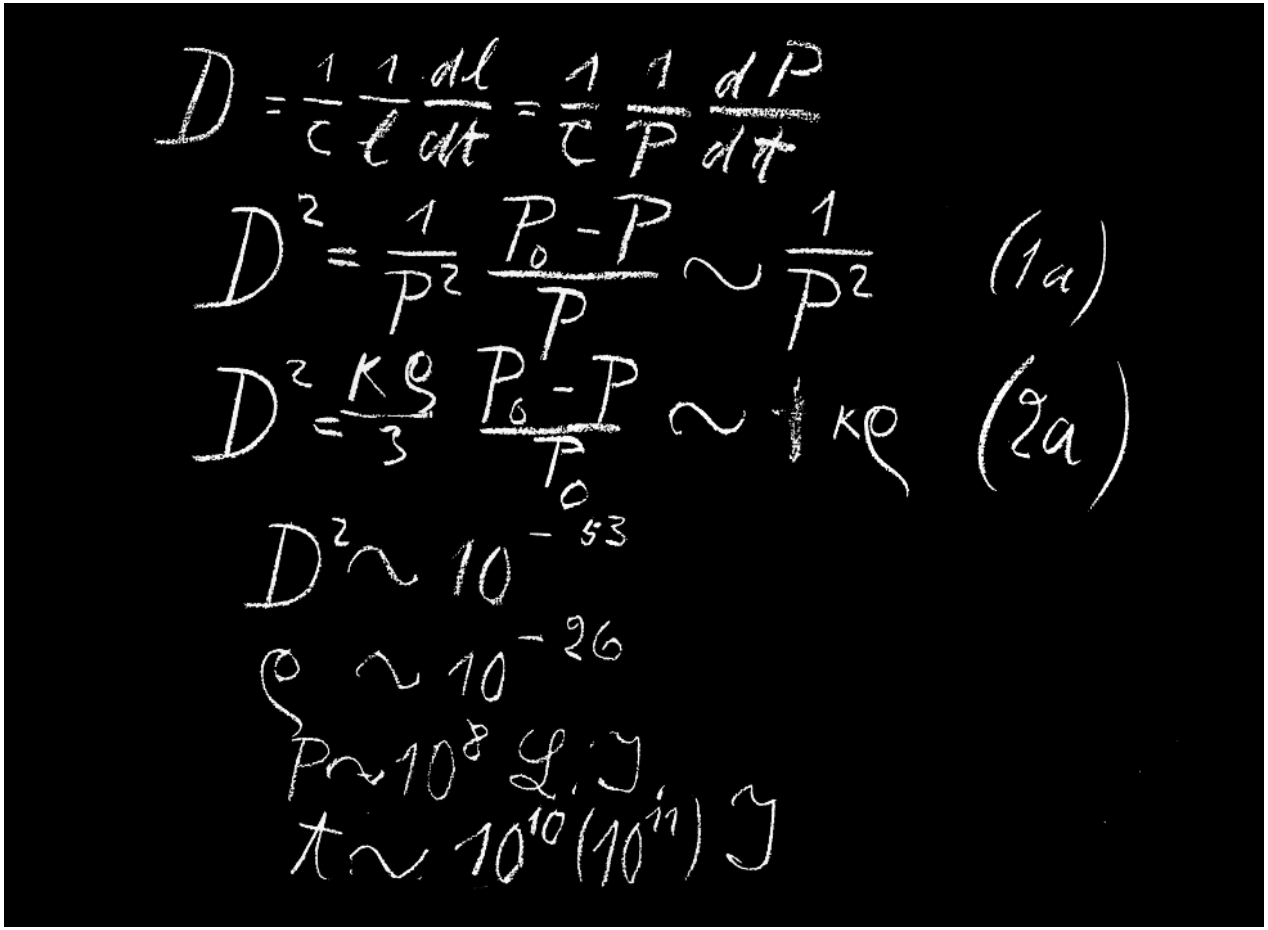
Modularize !


Monolithic Application
Modular Application
Modular and Dynamic Application
Modular and Dynamic Killer Application

Why OSGi ?

- Need simpler ways to construct software systems

- OSGi is about:
  - Software construction: building systems out of smaller components …
  - Components that work together …
  - Managing components …
  - "Universal Middleware"


$D = \frac{1}{c}\frac{1}{\ell}\frac{d\ell}{dt} = \frac{1}{c}\frac{1}{P}\frac{dP}{dt}$
$D^2 = \frac{1}{P^2}\frac{P_0 - P}{P} \sim \frac{1}{P^2}$ (1a)
$D^2 = \frac{\kappa\rho}{3}\frac{P_0 - P}{P_0} \sim \kappa\rho$ (2a)
$D^2 \sim 10^{-53}$
$\rho \sim 10^{-26}$
$P \sim 10^{8}$ Ly
$t \sim 10^{10}(10^{11})$ Ly

What is OSGi ?

OSGi Technology / Service Platform

- The dynamic module system for Java ™
- Provides :
  - Standardized primitives that allow applications to be constructed from small, reusable and collaborative components
  - Functions to dynamically change the composition, without requiring restarts
  - A service-oriented architecture that enables components to dynamically discover each other for collaboration => minimize coupling and render it manageable

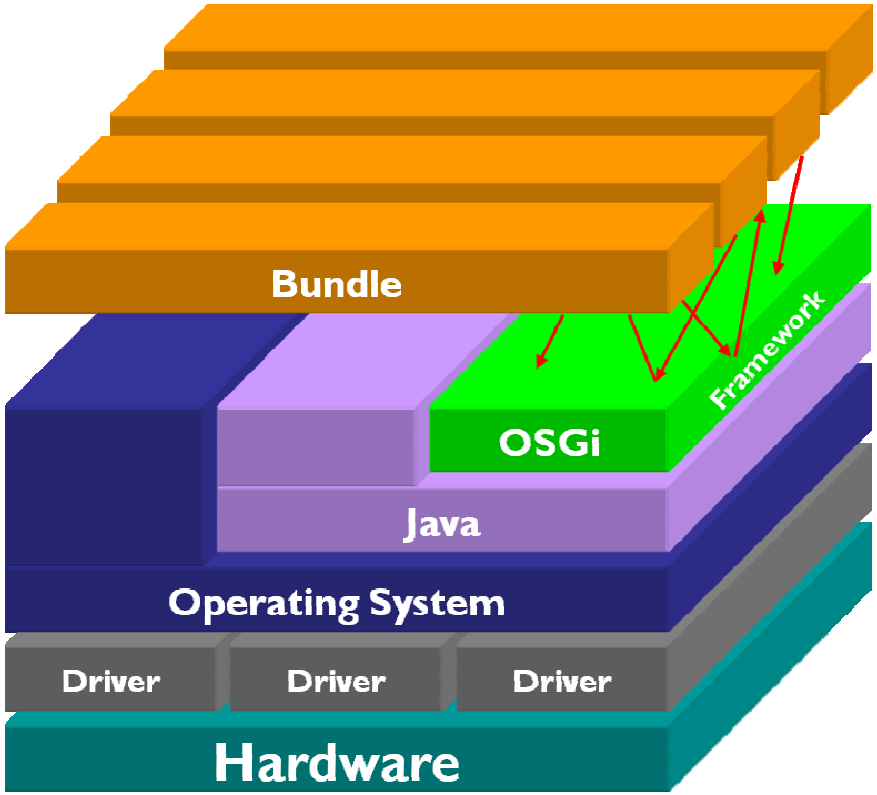
OSGi Alliance

- Industry consortium

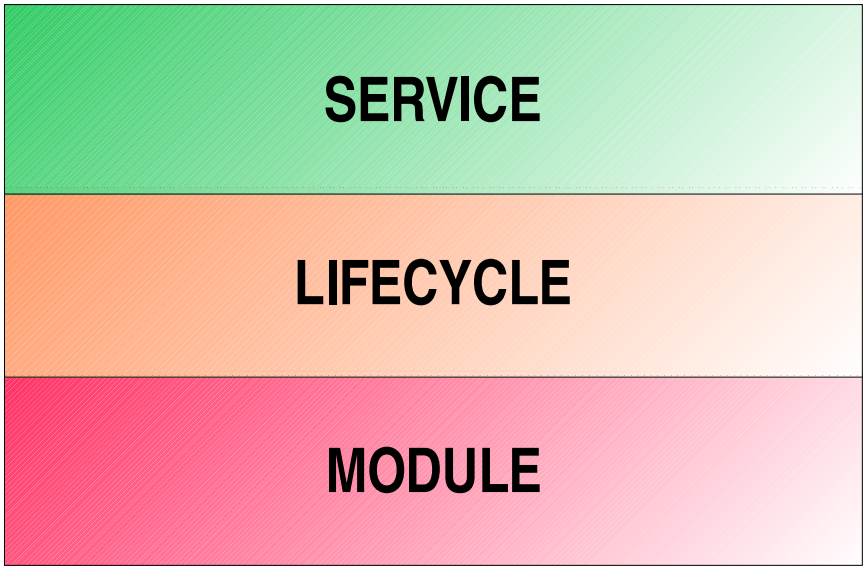
- OSGi Service Platform specification
  - Framework specification for hosting dynamically downloadable services
  - Standard service specifications

- Several expert groups define the specifications
  - Core Platform Expert Group (CPEG)
  - Mobile Expert Group (MEG)
  - Vehicle Expert Group (VEG)
  - Enterprise Expert Group (EEG)

OSGi Architectural Overview



OSGi Framework Layering



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

L2 - Manages the lifecycle of bundles in a bundle repository without requiring the Virtual Machine (VM) to be restarted

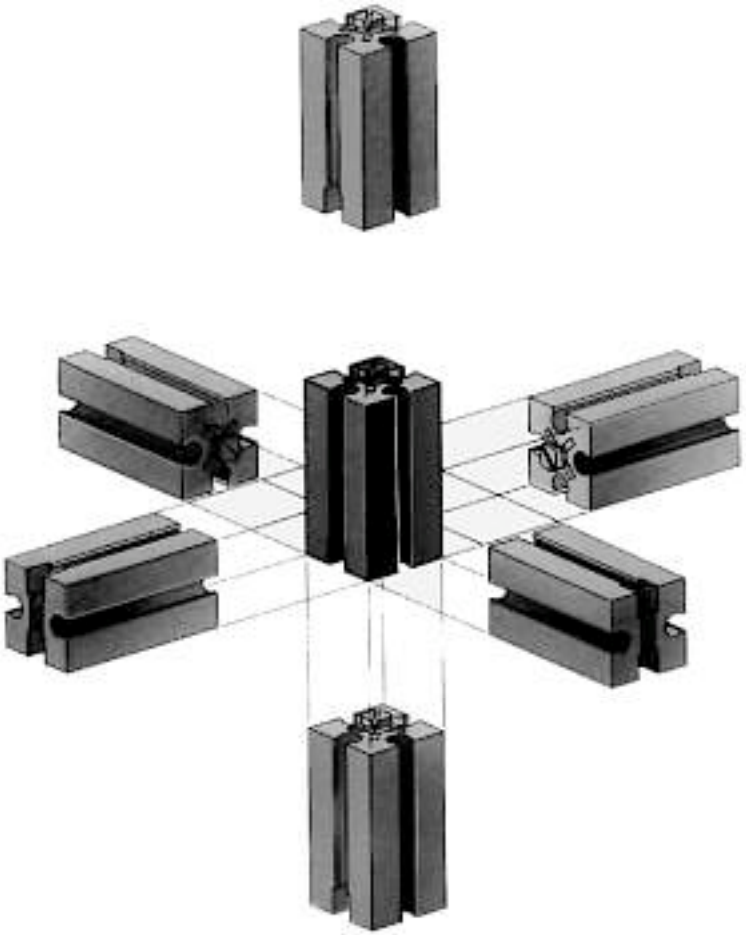
L1 - Creates the concept of bundles that use classes from each other in a controlled way according to constraints

OSGi Framework (1/2)

- Component-oriented framework
  - Bundles - i.e., modules/components
  - Package sharing and version management
  - Life-cycle management and notification
- Service-oriented architecture
  - Publish/find/bind intra-VM service model
- Open remote management architecture
  - No prescribed policy or protocol

OSGi Framework (2/2)

- Runs multiple applications and services
- Single Virtual Machine (VM) instance
- Separate Class-Loader per Bundle
  - Class-Loader graph
  - Independent namespaces
  - Class sharing at the Java package level
- Java Permissions to secure framework
- Explicitly considers dynamic scenarios
  - Run-time install, update and uninstall of Bundles



The Module Layer

Modularity

- What?

Modularity

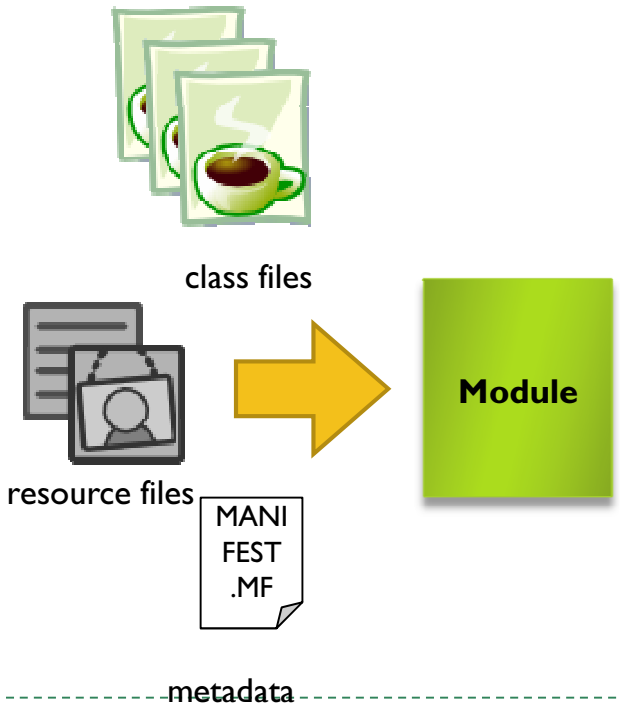
- What?
  - Separation of concerns
  - Structure
  - Encapsulation
  - Focuses on
    - Cohesion (low is bad, high is good)
    - Coupling (low is good, high is bad)
- Why?

Modularity

- What?
  - Separation of concerns
  - Structure
  - Encapsulation
  - Focuses on
    - Cohesion (low is bad, high is good)
    - Coupling (low is good, high is bad)
- Why?
  - Independent development
  - Independent maintenance and evolution
  - Improve reusability

OSGi Bundle

- A bundle is a module in OSGi terminology
- A bundle is a JAR file containing
  - Code
  - Resources
  - Metadata



Code Visibility Metadata

- A bundle is a JAR file containing code
  - What code in the JAR file is visible to other code in the JAR file?
  - What code in the JAR file is visible to code outside the JAR file?
  - What code outside the JAR file is visible to code inside the JAR file?

- Unlike standard JAR files, OSGi metadata explicitly answers all of these questions

Internal Code Visibility

- Internal code in standard JARs can see all root-relative packages
  - Not the case with bundles

Internal Code Visibility

- Internal code in standard JARs can see all root-relative packages
  - Not the case with bundles
- Bundles must specify Bundle-ClassPath
  - Comma-delimited list indicating where to search in the JAR file when looking for classes

Internal Code Visibility

- Internal code in standard JARs can see all root-relative packages
  - Not the case with bundles
- Bundles must specify Bundle-ClassPath
  - Comma-delimited list indicating where to search in the JAR file when looking for classes
- To get standard JAR behavior
  - `Bundle-ClassPath: .`

Internal Code Visibility

- Internal code in standard JARs can see all root-relative packages
  - Not the case with bundles
- Bundles must specify Bundle-ClassPath
  - Comma-delimited list indicating where to search in the JAR file when looking for classes
- To get standard JAR behavior
  - `Bundle-ClassPath: .`
- May also include embedded JARs and directories
- Examples
  - `Bundle-ClassPath: lib/foo.jar,classes/`
  - `Bundle-ClassPath: lib/foo.jar,.`

Exposing Internal Code (1/2)

- Standard JAR files expose all internal root-relative packages
  - Not the case with bundles

Exposing Internal Code (1/2)

- Standard JAR files expose all internal root-relative packages
  - Not the case with bundles
- Bundles must specify Export-Package
  - List of packages from the bundle class path to expose
  - Uses common OSGi syntax mentioned earlier
- Why do this?

Exposing Internal Code (1/2)

- Standard JAR files expose all internal root-relative packages
  - Not the case with bundles
- Bundles must specify Export-Package
  - List of packages from the bundle class path to expose
  - Uses common OSGi syntax mentioned earlier
- Why do this?
  - It separates internal visibility from external visibility
  - In other words, it allows bundles to have private content

Accessing External Code (1/2)

- Standard JARs implicitly see everything in the class path
  - Not the case with bundles

Accessing External Code (1/2)

- Standard JARs implicitly see everything in the class path
  - Not the case with bundles
- Bundles must specify Import-Package
  - List of packages needed from other bundles
  - Uses common OSGi syntax mentioned earlier

Accessing External Code (1/2)

- Standard JARs implicitly see everything other class on the class path
  - Not the case with bundles
- Bundles must specify Import-Package
  - List of packages needed from other bundles
  - Uses common OSGi syntax mentioned earlier
- Bundles must import every needed package not contained in the bundle itself, except java.*
- Why do this?

Accessing External Code (1/2)

- Standard JARs implicitly see everything other class on the class path
  - Not the case with bundles
- Bundles must specify Import-Package
  - List of packages needed from other bundles
  - Uses common OSGi syntax mentioned earlier
- Bundles must import every needed package not contained in the bundle itself, except java.*
- Why do this?
  - Make dependencies explicit
  - Make dependencies manageable

Bundle Manifest Example

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: org.foo.simplebundle
Bundle-Version: 1.0.0
Bundle-ClassPath: .,org/foo/embedded.jar
Import-Package:
 osgi.service.log; version="[1.0.0,1.1.0)",
 org.foo.service; version="1.1"
Export-Package:
 org.foo.service; version="1.1";
   vendor="org.foo",
 org.foo.service.bar; version="1.1";
   uses:="org.foo.service"
```

Bundle Manifest Example

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: org.foo.simplebundle
Bundle-Version: 1.0.0
Bundle-ClassPath: /foo/embedded.jar
Import-Package:
 osgi.service.[illegible] [1.0.0,1.1.0)",
 org.foo.service, version="1.1"
Export-Package:
 org.foo.service; version="1.1";
   vendor="org.foo",
 org.foo.service.bar; version="1.1";
   uses:="org.foo.service"
```

Indicates R4 semantics and syntax

Bundle Manifest Example

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: org.foo.simplebundle
Bundle-Version: 1.0.0
Bundle-ClassPath: ./org/foo/embedded.jar
Import-Package:
 osgi.service.log; version="[1.0.0,1.1.0)",
 org.foo.service;
Export-Package:
 org.foo.service; version="1.1";
   vendor="org.foo",
 org.foo.service.bar; version="1.1";
   uses:="org.foo.service"
```

Globally unique ID

Bundle Manifest Example

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: org.foo.simplebundle
Bundle-Version: 1.0.0
Bundle-ClassPath: .,org/foo/embedded.jar
Import-Package:
 osgi.service.log; version="[1.0.0,1.1.0)",
 org.foo.s[illegible]on="1.1"
Export-Pa[illegible]
 org.foo.service; version="1.1";
   vendor="org.foo",
 org.foo.service.bar; version="1.1";
   uses:="org.foo.service"
```

Internal bundle class path

Bundle Manifest Example

```
Bundle-ManifestVersion: 2
Bundle-SymbolicN                    mplebundle
Bundle-Version:
Bundle-ClassPath:                   mbedded.jar
Import-Package:
 osgi.service.log; version="[1.0.0,1.1.0)",
 org.foo.service; version="1.1"
Export-Package:
 org.foo.service; version="1.1";
   vendor="org.foo",
 org.foo.service.bar; version="1.1";
   uses:="org.foo.service"
```

Import of a
package version range

Bundle Manifest Example

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: org.foo.simplebundle
Bundle-Version: 1
Bundle-ClassPat                    dded.jar
Import-Package:
 osgi.service.log; version="[1.0.0,1.1.0)",
 org.foo.service; version="1.1"
Export-Package:
 org.foo.service; version="1.1";
   vendor="org.foo",
 org.foo.service.bar; version="1.1";
   uses:="org.foo.service"
```

Importing an exported package

Bundle Manifest Example

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: org.foo.simplebundle
Bundle-Version: 1.0.0
Bundle-ClassPath:                    mbedded.jar
Import-Package:
 osgi.service.                       0.0,1.1.0)",
 org.foo.service;                    1.1"
Export-Package:
 org.foo.service; version="1.1";
   vendor="org.foo",
 org.foo.service.bar; version="1.1";
   uses:="org.foo.service"
```

Exported package with version and arbitrary attribute

Bundle Manifest Example

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: org.foo.simplebundle
Bundle-Version: 1.0.0
Bundle-ClassPath: .,org/foo/embedded.jar
Import-Package:
 osgi.service.log; vers 1.1.0)",
 org.foo.service; ve
Export-Package:
 org.foo.service; versi
   vendor="org.foo",
 org.foo.service.bar; version="1.1";
   uses:="org.foo.service"
```

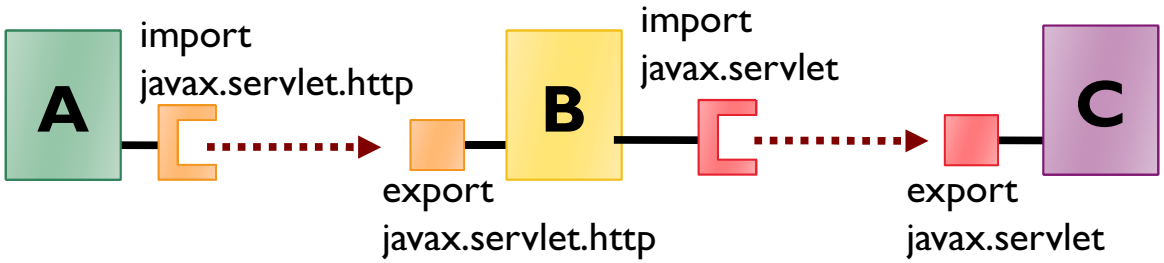
Provided package with dependency on exported package

Dependency Resolution (1/2)

- Automatically managed by the OSGi framework
  - Ensures a bundle's dependencies are satisfied before the bundle can be used

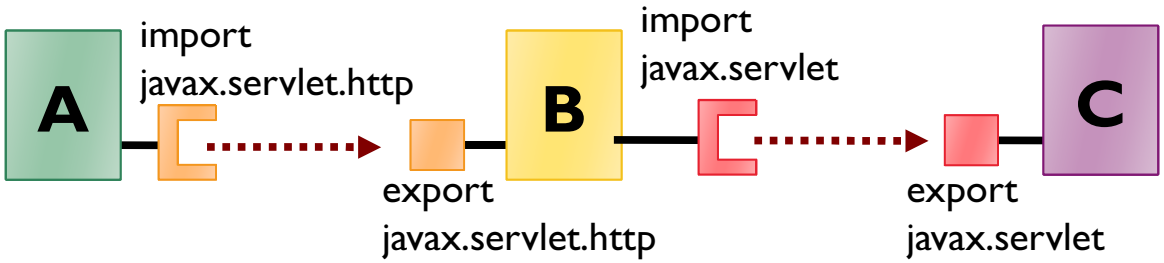
Dependency Resolution (1/2)

- Automatically managed by the OSGi framework
  - Ensures a bundle's dependencies are satisfied before the bundle can be used
- In simple terms, resolving a bundle matches its imported packages to bundles providing them



Dependency Resolution (1/2)

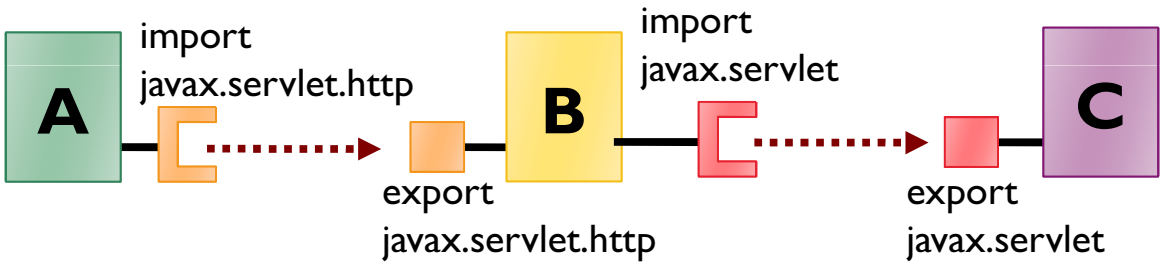
- Automatically managed by the OSGi framework
  - Ensures a bundle's dependencies are satisfied before the bundle can be used
- In simple terms, resolving a bundle matches its imported packages to bundles providing them



  - Typically, resolving a bundle will result in other bundles being transitively resolved

Dependency Resolution (1/2)

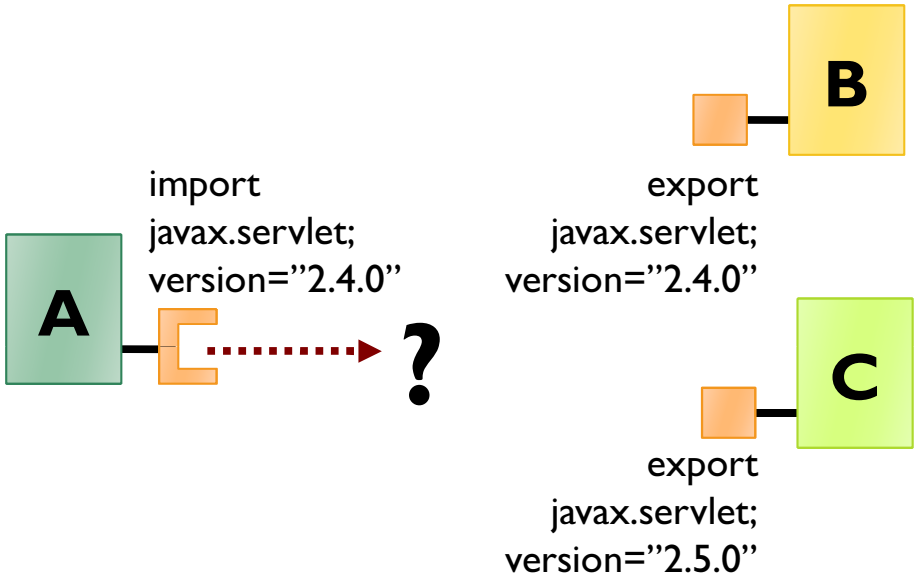
- Automatically managed by the OSGi framework
  - Ensures a bundle's dependencies are satisfied before the bundle can be used
- In simple terms, resolving a bundle matches its imported packages to bundles providing them



  - Typically, resolving a bundle will result in other bundles being transitively resolved
  - If a version or arbitrary attributes are specified on imports, then exports must match
    - Multiple attributes on an import are logically ANDed

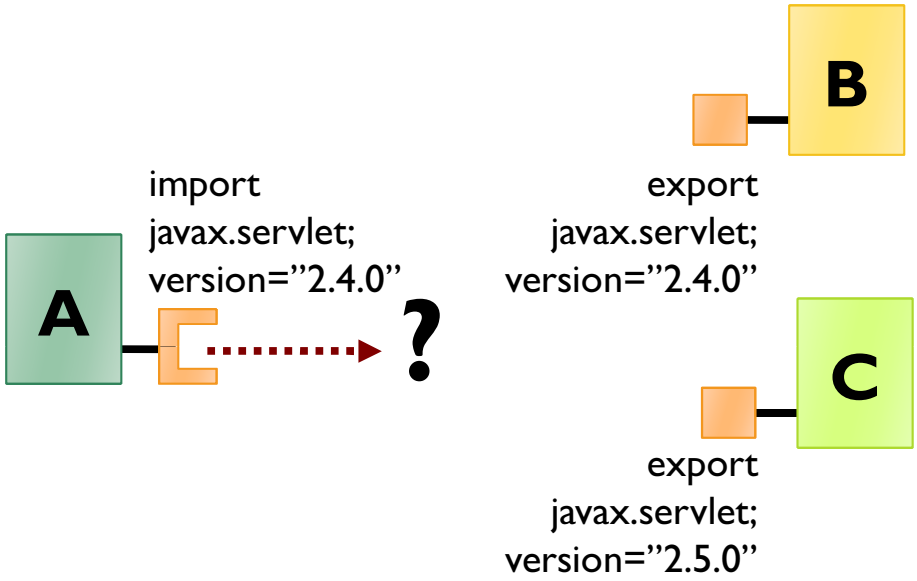
Dependency Resolution (2/2)

- Multiple matching providers



Dependency Resolution (2/2)

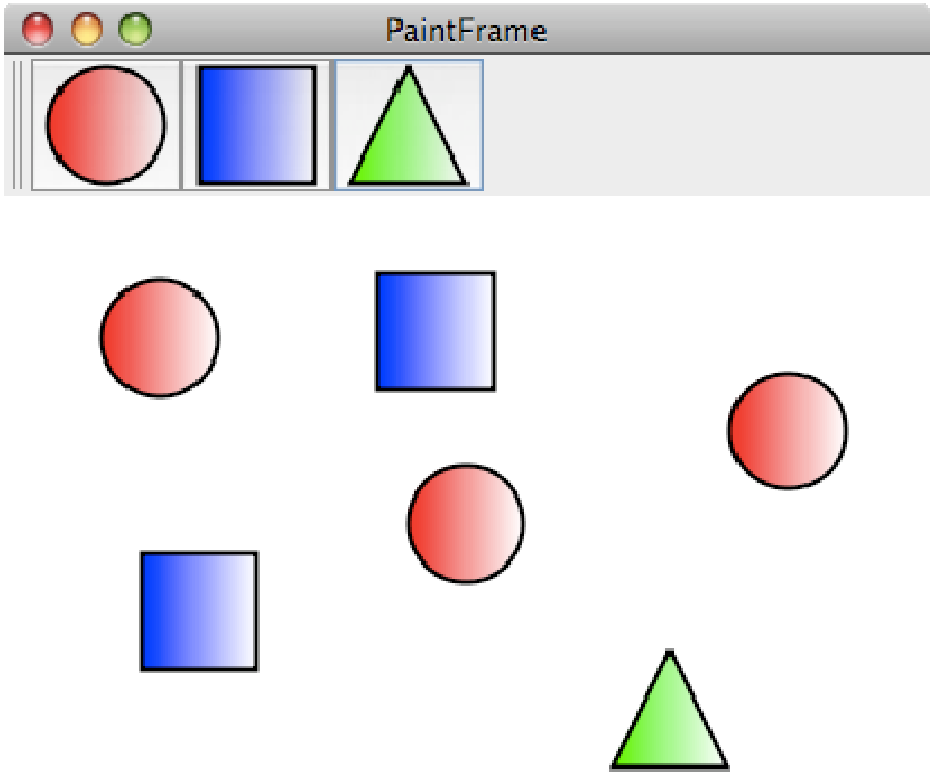
- Multiple matching providers



- Resolution algorithm orders matching providers
  - Already resolved providers ordered by decreasing version
  - Unresolved providers ordered by decreasing version
  - If versions are equal, matching providers are ordered based on installation order

Non-Modular Paint Program (1/3)

- We have a simple paint program



Non-Modular Paint Program (2/3)

- It is packaged as a single JAR file with the following contents:

```
META-INF/
META-INF/MANIFEST.MF
org/
org/foo/
org/foo/paint/
org/foo/paint/PaintFrame$1$1.class
org/foo/paint/PaintFrame$1.class
org/foo/paint/PaintFrame$ShapeActionListener.class
org/foo/paint/PaintFrame.class
org/foo/paint/SimpleShape.class
org/foo/paint/ShapeComponent.class
org/foo/shape/
org/foo/shape/Circle.class
org/foo/shape/circle.png
org/foo/shape/Square.class
org/foo/shape/square.png
org/foo/shape/Triangle.class
org/foo/shape/triangle.png
```

Non-Modular Paint Program (2/3)

- It is packaged as a single JAR file with the following contents:

```
META-INF/
META-INF/MANIFEST.MF
org/
org/foo/
org/foo/paint/
org/foo/paint/PaintFrame$1$1.class
org/foo/paint/PaintFrame$1.class
org/foo/paint/PaintFrame$ShapeActionListener.class
org/foo/paint/PaintFrame.class
org/foo/paint/SimpleShape.class
org/foo/paint/ShapeComponent.class
org/foo/shape/
org/foo/shape/Circle.class
org/foo/shape/circle.png
org/foo/shape/Square.class
org/foo/shape/square.png
org/foo/shape/Triangle.class
org/foo/shape/triangle.png
```

Main implementation package is `org.foo.paint`

Non-Modular Paint Program (2/3)

- It is packaged as a single JAR file with the following contents:

```
META-INF/
META-INF/MANIFEST.MF
org/
org/foo/
org/foo/paint/
org/foo/paint/PaintFrame$1$1.class
org/foo/paint/PaintFrame$1.class
org/foo/paint/PaintFrame$ShapeActi
org/foo/paint/PaintFrame.class
org/foo/paint/SimpleShape.class
org/foo/paint/ShapeComponent.class
org/foo/shape/
org/foo/shape/Circle.class
org/foo/shape/circle.png
org/foo/shape/Square.class
org/foo/shape/square.png
org/foo/shape/Triangle.class
org/foo/shape/triangle.png
```

Static main method in `PaintFrame`

Non-Modular Paint Program (2/3)

- It is packaged as a single JAR file with the following contents:

```
META-INF/
META-INF/MANIFEST.MF
org/
org/foo/
org/foo/paint/
org/foo/paint/PaintFrame$1$1.class
org/foo/paint/PaintFrame$1.class
org/foo/paint/PaintFrame$ShapeAction
org/foo/paint/PaintFrame.class
org/foo/paint/SimpleShape.class
org/foo/paint/ShapeComponent.class
org/foo/shape/
org/foo/shape/Circle.class
org/foo/shape/circle.png
org/foo/shape/Square.class
org/foo/shape/square.png
org/foo/shape/Triangle.class
org/foo/shape/triangle.png
```

Interface `SimpleShape` supports multiple shape implementations

Non-Modular Paint Program (2/3)

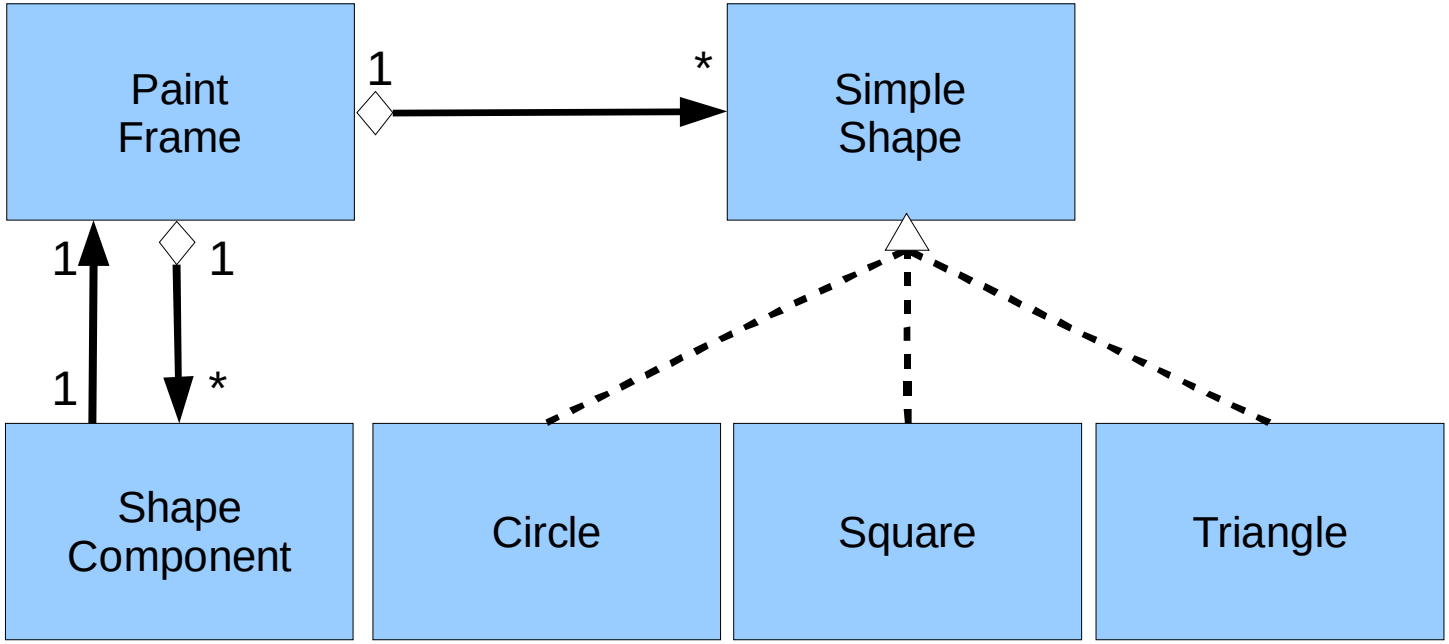
- It is packaged as a single JAR file with the following contents:

```
META-INF/
META-INF/MANIFEST.MF
org/
org/foo/
org/foo/paint/
org/foo/paint/PaintFrame$1$1.class
org/foo/paint/PaintFrame$1.class
org/foo/paint/PaintFrame$ShapeActionListener.class
org/foo/paint/PaintFrame.class
org/foo/paint/SimpleShape.class
org/foo/paint/ShapeComponent.class
org/foo/shape/
org/foo/shape/Circle.class
org/foo/shape/circle.png
org/foo/shape/Square.class
org/foo/shape/square.png
org/foo/shape/Triangle.class
org/foo/shape/triangle.png
```

Shape implementations defined in `org.foo.shape`

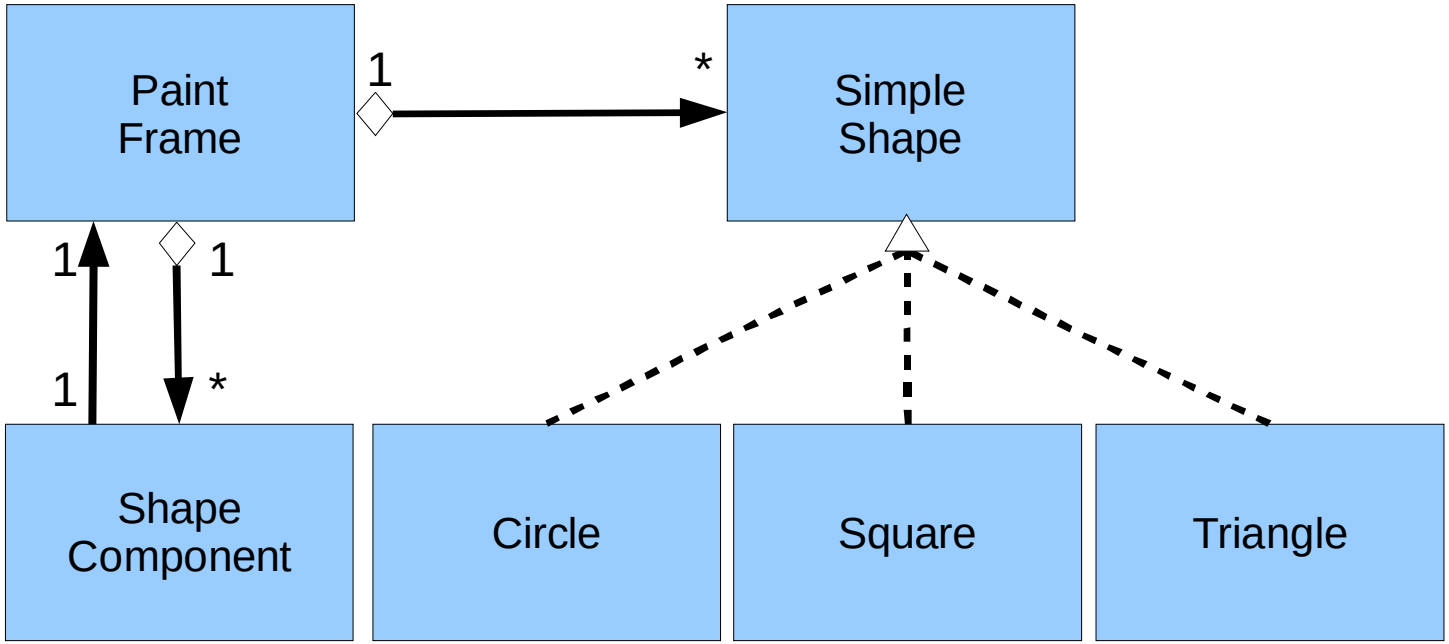
Non-Modular Paint Program (3/3)

- Relationship among classes



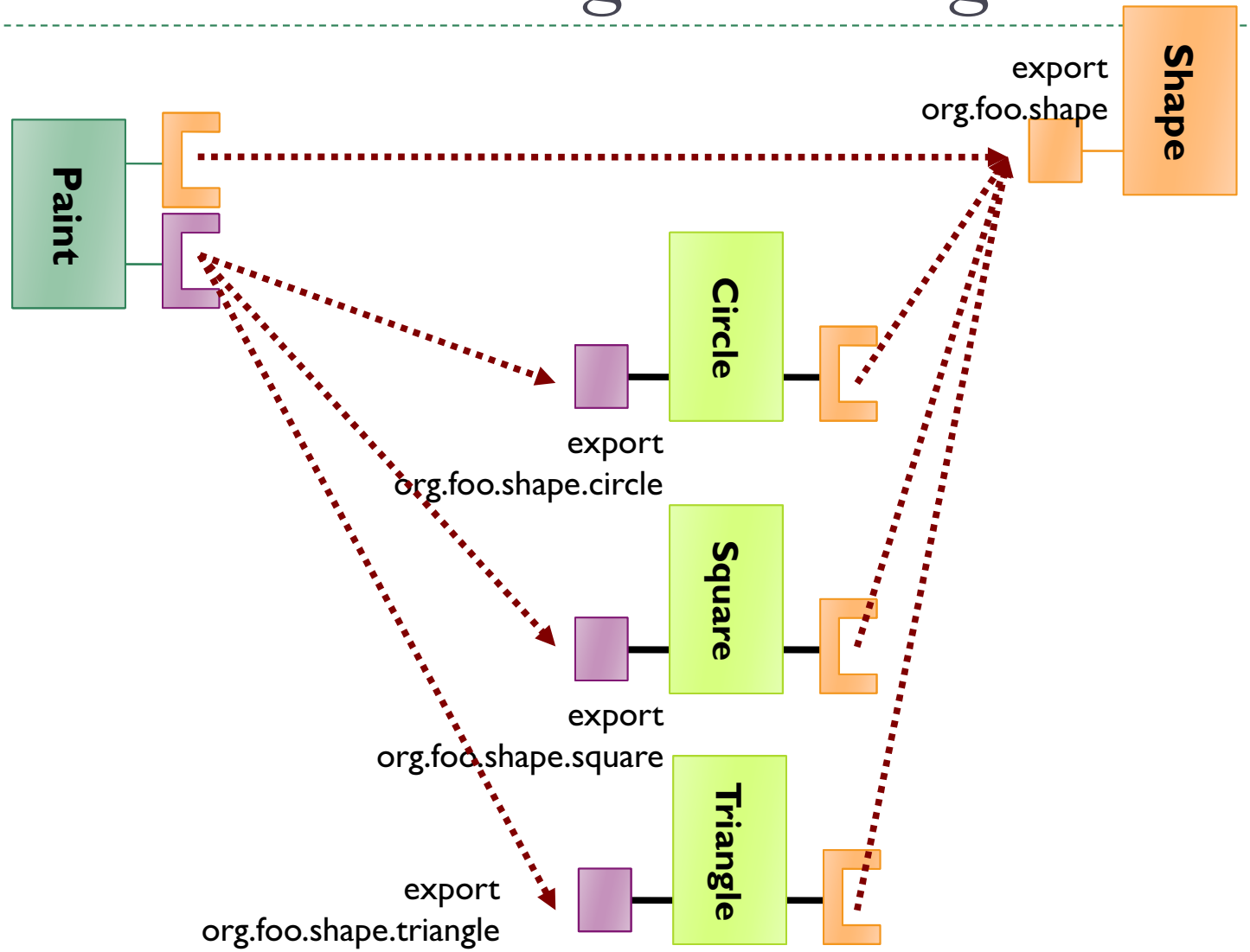
Non-Modular Paint Program (3/3)

- Relationship among classes



How to decompose into bundles?

Modular Paint Program Design



Benefits of Modularized Paint Program

- Enforced logical boundaries
- Automatic dependency resolution
  - Ensures proper configuration
- Improves reusability of code
- Improves ability to create different configurations

The Lifecycle Layer

What & Why of Lifecycle Layer

- Once we have a bundle, what do we do with it?
  - We need to somehow tell the OSGi framework about it

What & Why of Lifecycle Layer

- Once we have a bundle, what do we do with it?
  - We need to somehow tell the OSGi framework about it
- What if our bundle needs to be initialized somehow?
  - We need some sort of hook in the framework

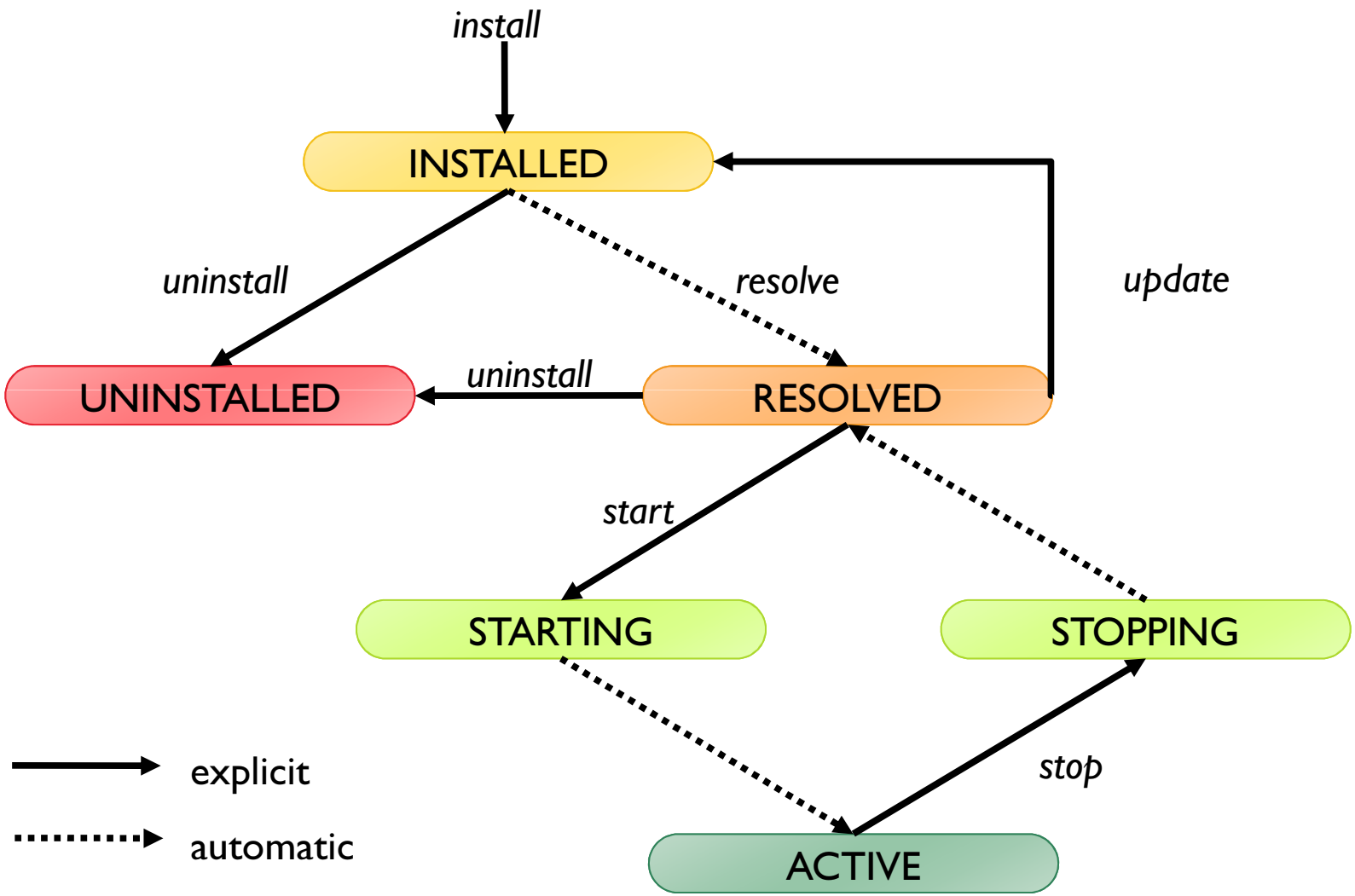
What & Why of Lifecycle Layer

- Once we have a bundle, what do we do with it?
  - We need to somehow tell the OSGi framework about it
- What if our bundle needs to be initialized somehow?
  - We need some sort of hook in the framework
- What if we want to add and remove bundles at run time?
  - We need someway to access the underlying framework

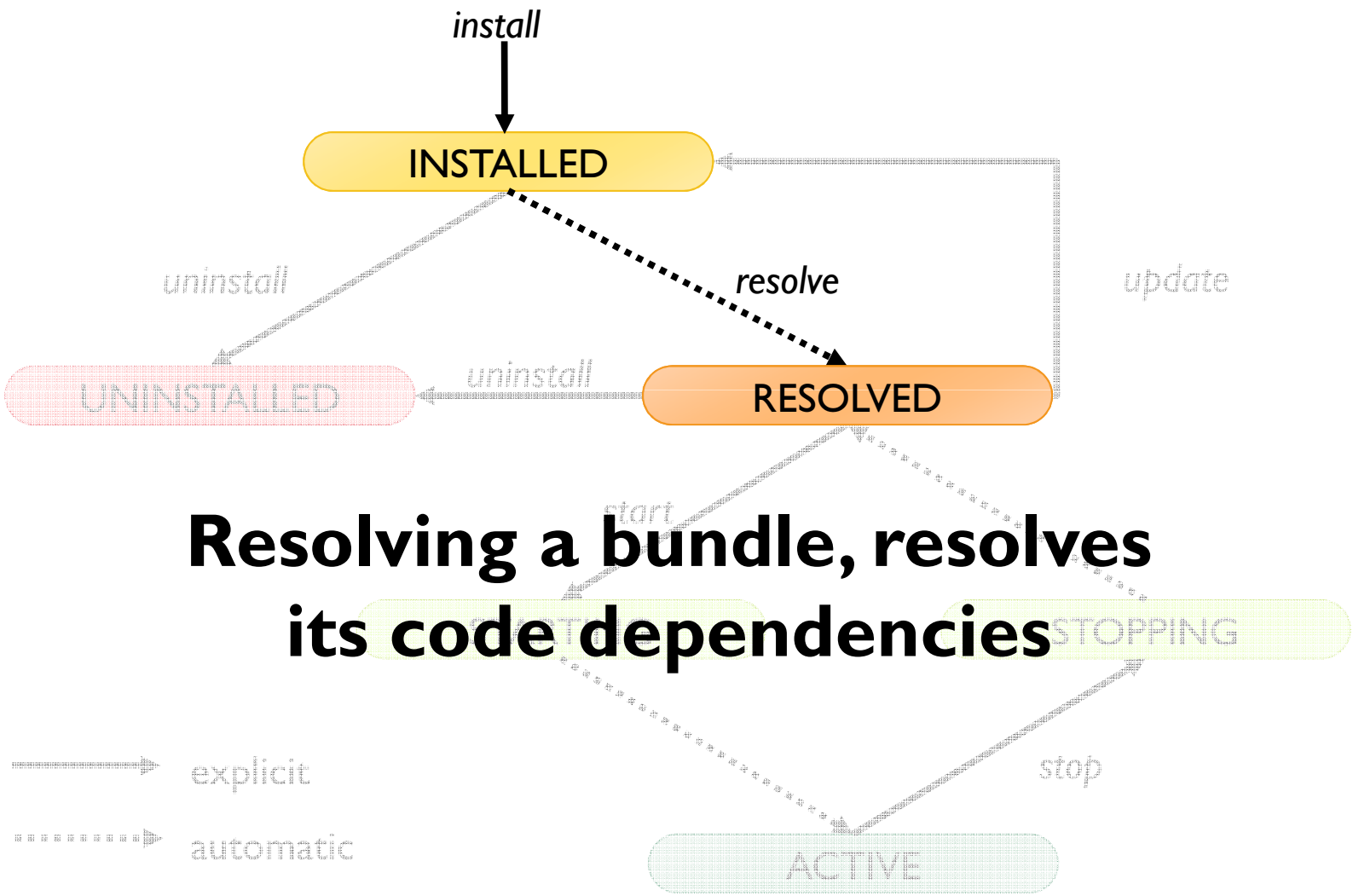
What & Why of Lifecycle Layer

- Once we have a bundle, what do we do with it?
  - We need to somehow tell the OSGi framework about it
- What if our bundle needs to be initialized somehow?
  - We need some sort of hook in the framework
- What if we want to add and remove bundles at run time?
  - We need someway to access the underlying framework
- We can do all of these things with a well-defined lifecycle for bundles
  - A lifecycle defines the stages of a bundle's lifetime
    - The framework associates a lifecycle state with each bundle

Bundle Life Cycle



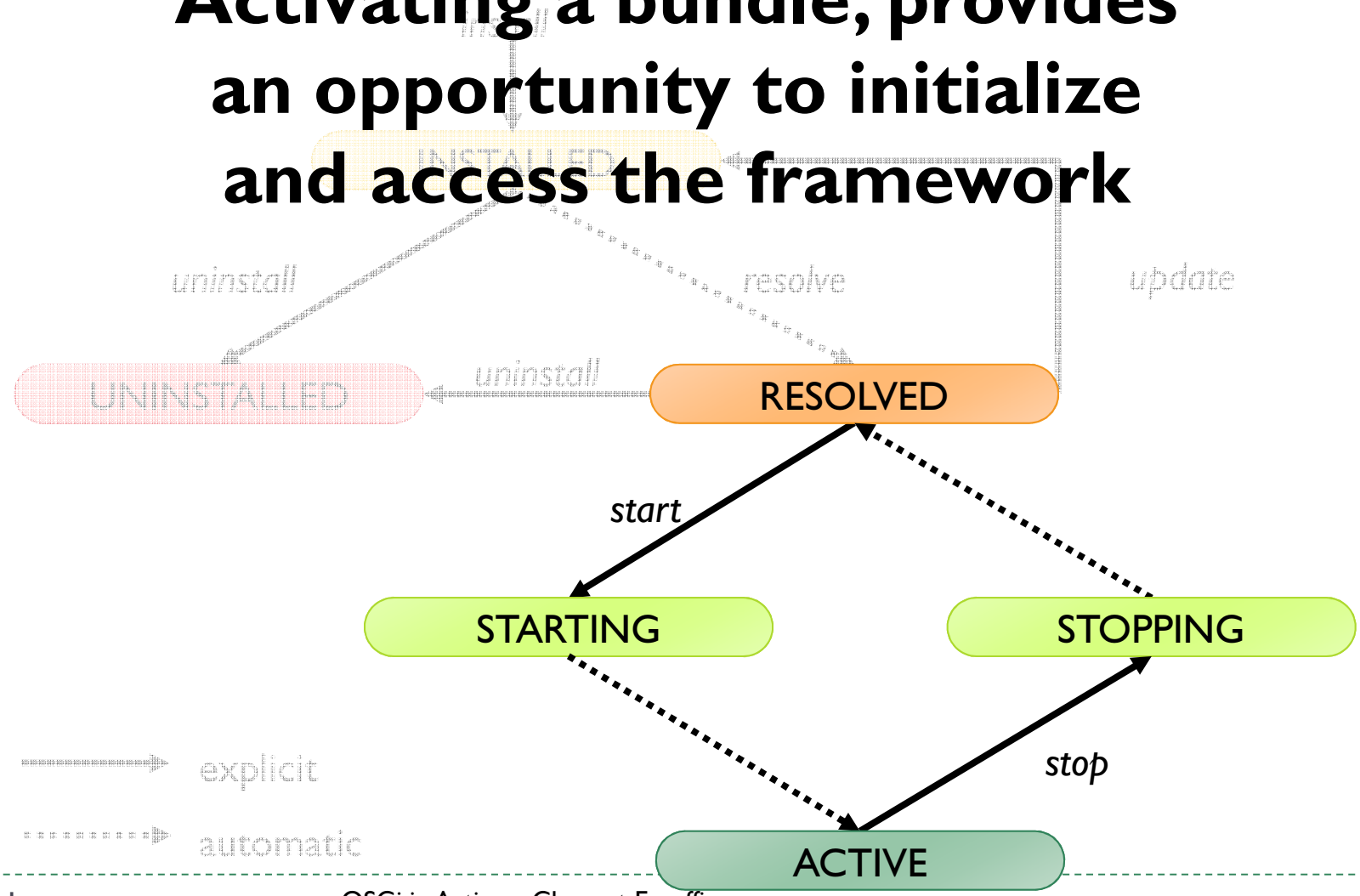
Bundle Life Cycle



Resolving a bundle, resolves its code dependencies

Bundle Life Cycle

Activating a bundle, provides an opportunity to initialize and access the framework



Bundle Activator

- The bundle activator is a framework hook to allow bundles to startup and shutdown

Bundle Activator

- The bundle activator is a framework hook to allow bundles to startup and shutdown
  - The hook is invoked in the STARTING/STOPPING states

Bundle Activator

- The bundle activator is a framework hook to allow bundles to startup and shutdown
  - The hook is invoked in the STARTING/STOPPING states
  - An activator implements a simple interface and is included in the bundle JAR file

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

Bundle Activator

- The bundle activator is a framework hook to allow bundles to startup and shutdown
  - The hook is invoked in the STARTING/STOPPING states
  - An activator implements a simple interface and is included in the bundle JAR file

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

  - Additional manifest metadata is needed to declare the activator

```
Bundle-Activator: <fully-qualified-class-name>
```

e.g.:

```
Bundle-Activator: org.foo.MyActivator
```

Bundle Activator

- The bundle activator is a framework hook to allow bundles to startup and shutdown
  - The hook is invoked in the STARTING/STOPPING states
  - An activator implements a simple interface and is included in the bundle JAR file

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

  - Additional manifest meta is needed to declare the activator

```
Bundle-Activa lass-name>
```

e.g.:

```
Bundle-Activator: ivator
```

What are these?

Bundle Context

- Represents the bundle's execution context

```
public interface BundleContext {
  String getProperty(String key);
  Bundle getBundle();
  Bundle installBundle(String location) throws BundleException;
  Bundle installBundle(String location, InputStream input)
    throws BundleException;
  Bundle getBundle(long id);
  Bundle[] getBundles();
  ...
  void addBundleListener(BundleListener listener);
  void removeBundleListener(BundleListener listener);
  void addFrameworkListener(FrameworkListener listener);
  void removeFrameworkListener(FrameworkListener listener);
  ...
  File getDataFile(String filename);
  ...
}
```

Bundle Context

- Represents the bundle's execution context

Lifecycle method
to install other bundles

```
public interface BundleContext {
  String getProperty(String key);
  Bundle getBundle();
  Bundle installBundle(String location) throws BundleException;
  Bundle installBundle(String location, InputStream input)
    throws BundleException;
  Bundle getBundle(long id);
  Bundle[] getBundles();
  ...
  void addBundleListener(BundleListener listener);
  void removeBundleListener(BundleListener listener);
  void addFrameworkListener(FrameworkListener listener);
  void removeFrameworkListener(FrameworkListener listener);
  ...
  File getDataFile(String filename);
  ...
}
```

Bundle Context

- Represents the bundle's execution context

```
public interface BundleContext {
  String getProperty(String key);
  Bundle getBundle();
  Bundle installBundle(String location) throws BundleException;
  Bundle installBundle(String location, InputStream input)
    throws BundleException;
  Bundle getBundle(long id);
  Bundle[] getBundles();
  ...
  void addBundleListener(BundleListener listener);
  void removeBundleListener(BundleListener listener);
  void addFrameworkListener(FrameworkListener listener);
  void removeFrameworkListener(FrameworkListener listener);
  ...
  File getDataFile(String filename);
  ...
}
```

Access to other installed bundles

Bundle Context

- Represents the bundle's execution context

```
public interface BundleContext {
  String getProperty(Strin
  Bundle getBundle();
  Bundle installBundle(Str                    ndleException;
  Bundle installBundle(String                 tStream input)
    throws BundleException;
  Bundle getBundle(long id);
  Bundle[] getBundles();
  ...
  void addBundleListener(BundleListener listener);
  void removeBundleListener(BundleListener listener);
  void addFrameworkListener(FrameworkListener listener);
  void removeFrameworkListener(FrameworkListener listener);
  ...
  File getDataFile(String filename);
  ...
}
```

Access to our own bundle...
what's that?

Bundle

- Run-time representation of a bundle

```
public interface Bundle {
  ...
  int getState();
  void start(int options) throws BundleException;
  void start() throws BundleException;
  void stop(int options) throws BundleException;
  void stop() throws BundleException;
  void update() throws BundleException;
  void update(InputStream in) throws BundleException;
  void uninstall() throws BundleException;
  Dictionary getHeaders();
  String getSymbolicName();
  long getBundleId();
  String getLocation();
  ...
  URL getResource(String name);
  Enumeration getResources(String name) throws IOException;
  Class loadClass(String name) throws ClassNotFoundException;
  ...
  BundleContext getBundleContext();
}
```

Bundle

- Run-time representation of a bundle

```
public interface Bundle {
  ...
  int getState();
  void start(int options) throws BundleException;
  void start() throws BundleException;
  void stop(int options) throws BundleException;
  void stop() throws BundleException;
  void update() throws BundleException;
  void update(InputStream in) throws BundleException;
  void uninstall() throws BundleException;
  Dictionary getHeaders();
  String getSymbolicName();
  long getBundleId();
  String getLocation();
  ...
  URL getResource(String name);
  Enumeration getResources(String name) throws IOException;
  Class loadClass(String name) throws ClassNotFoundException;
  ...
  BundleContext getBundleContext();
}
```

Lifecycle method to start bundle

Bundle

- Run-time representation of a bundle

```
public interface Bundle {
  ...
  int getState();
  void start(int options) throws BundleException;
  void start() throws BundleException;
  void stop(int options) throws BundleException;
  void stop() throws BundleException;
  void update() throws BundleException;
  void update(InputStream in) throws BundleException;
  void uninstall() throws BundleException;
  Dictionary getHeaders();
  String getSymbolicName();
  long getBundleId();
  String getLocation();
  ...
  URL getResource(String name);
  Enumeration getResources(String name) throws IOException;
  Class loadClass(String name) throws ClassNotFoundException;
  ...
  BundleContext getBundleContext();
}
```

Lifecycle method to stop bundle

Bundle

- Run-time representation of a bundle

```
public interface Bundle {
  ...
  int getState();
  void start(int options) throws BundleException;
  void start() throws BundleException;
  void stop(int options) throws BundleException;
  void stop() throws BundleException;
  void update() throws BundleException;
  void update(InputStream in) throws BundleException;
  void uninstall() throws BundleException;
  Dictionary getHeaders();
  String getSymbolicName();
  long getBundleId();
  String getLocation();
  ...
  URL getResource(String name);
  Enumeration getResources(String name) throws IOException;
  Class loadClass(String name) throws ClassNotFoundException;
  ...
  BundleContext getBundleContext();
}
```

Lifecycle method to update bundle

Bundle

- Run-time representation of a bundle

```
public interface Bundle {
  ...
  int getState();
  void start(int options) throws BundleException;
  void start() throws BundleException;
  void stop(int options) throws BundleException;
  void stop() throws BundleException;
  void update() throws BundleException;
  void update(InputStream in) throws BundleException;
  void uninstall() throws BundleException;
  Dictionary getHeaders();
  String getSymbolicName();
  long getBundleId();
  String getLocation();
  ...
  URL getResource(String name);
  Enumeration getResources(String name) throws IOException;
  Class loadClass(String name) throws ClassNotFoundException;
  ...
  BundleContext getBundleContext();
}
```

Lifecycle method to uninstall bundle

Bundle

- Run-time representation of a bundle

```
public interface Bundle {
  ...
  int getState();
  void start(int options) throws BundleException;
  void start() throws BundleException;
  void stop(int options) throws BundleException;
  void stop() throws BundleException;
  void update() throws BundleException;
  void update(InputStream in) throws BundleException;
  void uninstall() throws BundleException;
  Dictionary getHeaders();
  String getSymbolicName();
  long getBundleId();
  String getLocation();
  ...
  URL getResource(String name);
  Enumeration getResources(String name) throws IOException;
  Class loadClass(String name) throws ClassNotFoundException;
  ...
  BundleContext getBundleContext();
}
```

Access to bundle resources

Bundle

- Run-time representation of a bundle

```
public interface Bundle {
  ...
  int getState();
  void start(int options) throws BundleException;
  void start() throws BundleException;
  void stop(int options) throws BundleException;
  void stop() throws BundleException;
  void update() throws BundleException;
  void update(InputStream in) throws BundleException;
  void uninstall() throws BundleException;
  Dictionary getHeaders();
  String getSymbolicName();
  long getBundleId();
  String getLocation();
  ...
  URL getResource(String name);
  Enumeration getResources(String name) throws IOException;
  Class loadClass(String name) throws ClassNotFoundException;
  ...
  BundleContext getBundleContext();
}
```

Access to accessible classes

Bundle Dynamism

- Bundles can be installed, started, stopped, updated, and uninstalled at run time
  - Bundle events signal lifecycle changes

To listen for events

```
BundleContext.addBundleListener()
```

Bundle Dynamism

- Bundles can be installed, started, stopped, updated, and uninstalled at run time
  - Bundle events signal lifecycle changes

Implement listener interface

```
public interface BundleListener extends EventListener {
  public void bundleChanged(BundleEvent event);
}
```

Bundle Dynamism

- Bundles can be installed, started, stopped, updated, and uninstalled at run time
  - Bundle events signal lifecycle changes

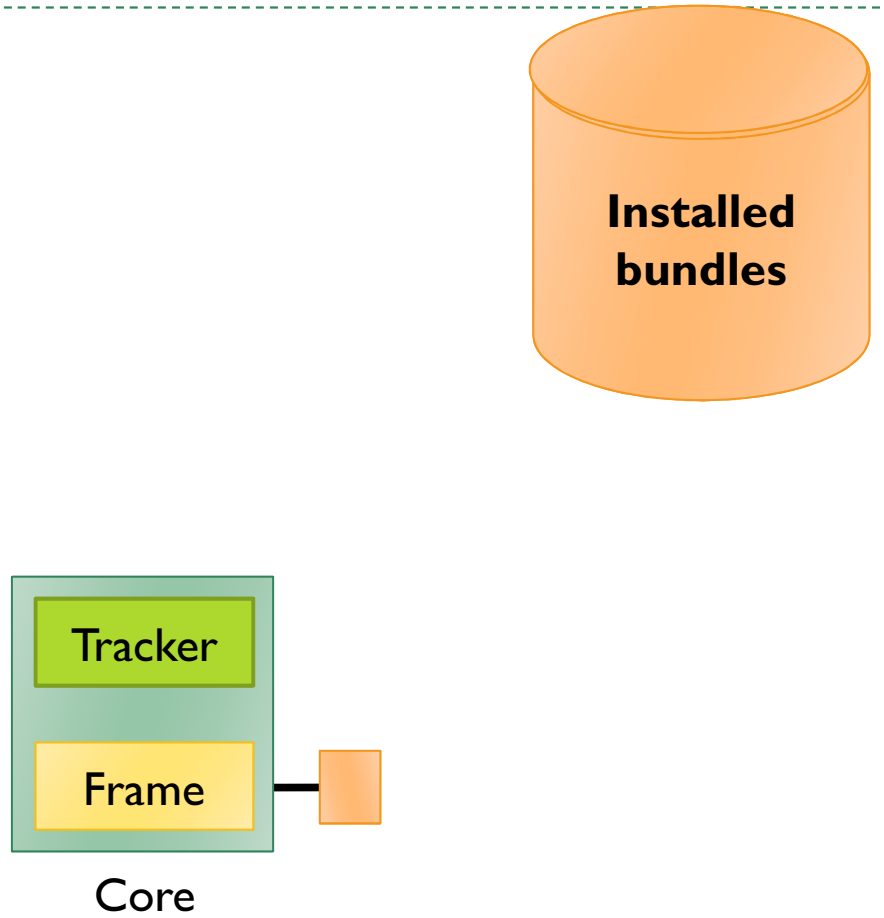
Received event

```
public class BundleEvent extends EventObject {
  public final static int    INSTALLED   = 0x00000001;
  public final static int    STARTED     = 0x00000002;
  public final static int    STOPPED     = 0x00000004;
  public final static int    UPDATED     = 0x00000008;
  public final static int    UNINSTALLED = 0x00000010;
  public final static int    RESOLVED    = 0x00000020;
  public final static int    UNRESOLVED  = 0x00000040;
  public final static int    STARTING    = 0x00000080;
  public final static int    STOPPING    = 0x00000100;
  …
  public Bundle getBundle() { … }
  public int getType() { … }
}
```

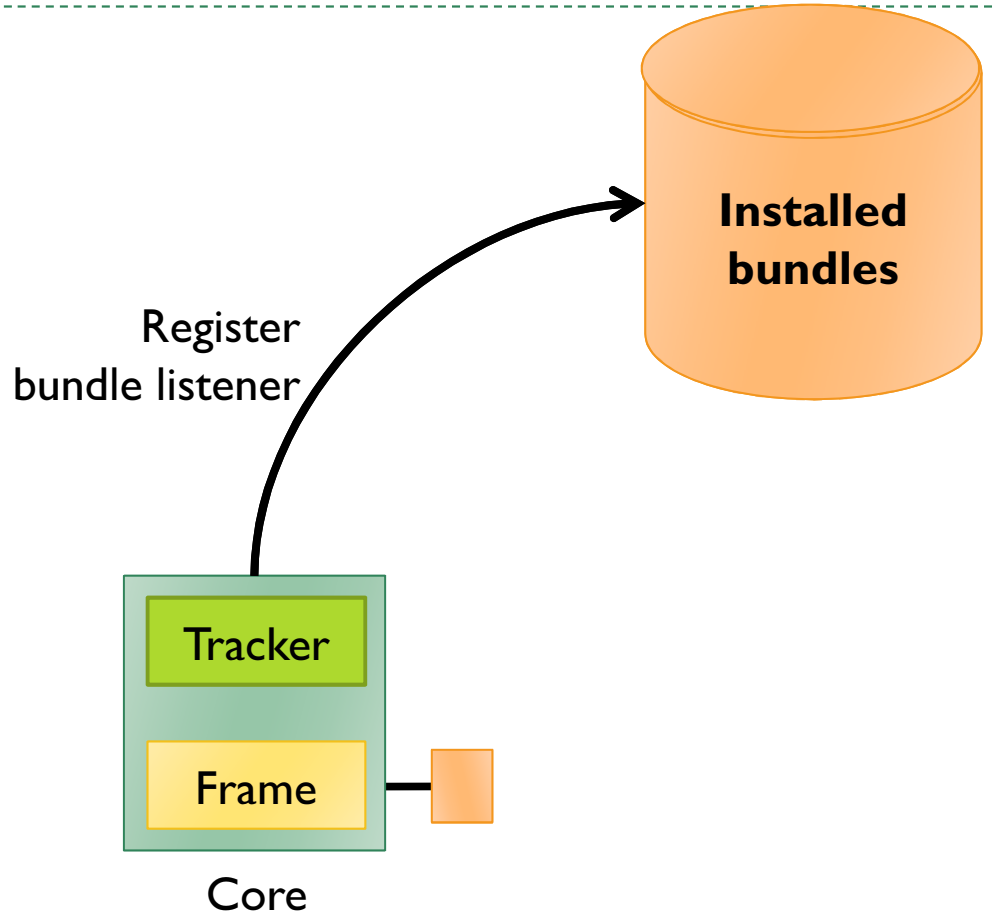
Bundle-Based Dynamic Extensibility

- Bundle lifecycle events provide a mechanism for dynamic extensibility
- The extender pattern
  - An application component, called the extender, listens for bundles to be installed, started, and stopped
  - On install, the extender probes bundles to see if they are extensions
    - Typically, extension contain special metadata or resources to indicate they provide an extension
  - When started, the extender performs some action to integrate the extension into the application
  - When stopped, the extender performs some action to remove the extension from the application

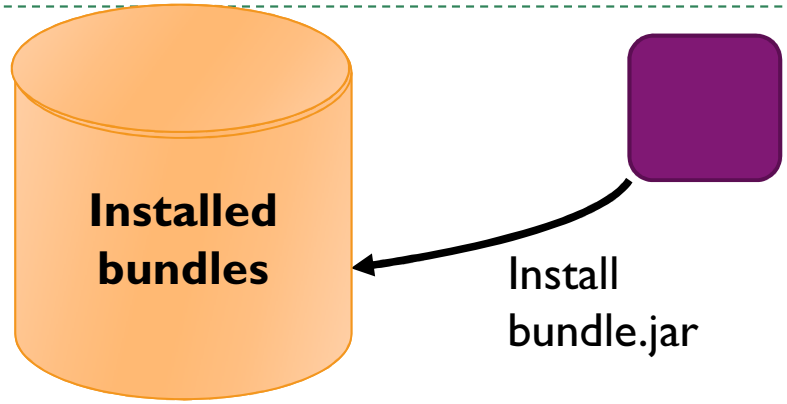
Extender Pattern

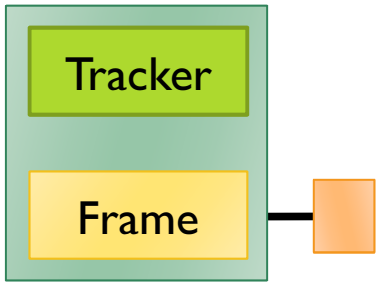


Extender Pattern



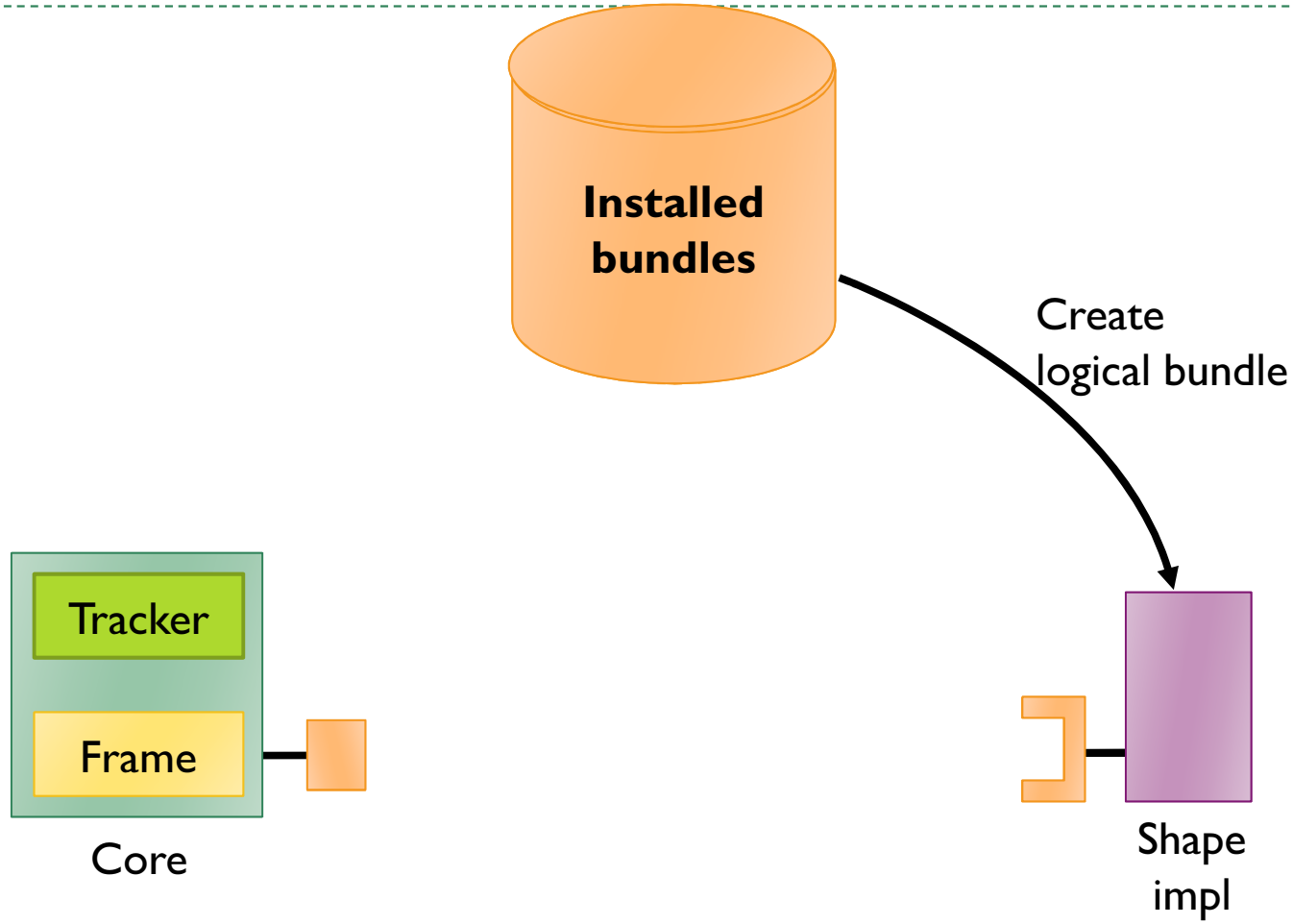
Extender Pattern



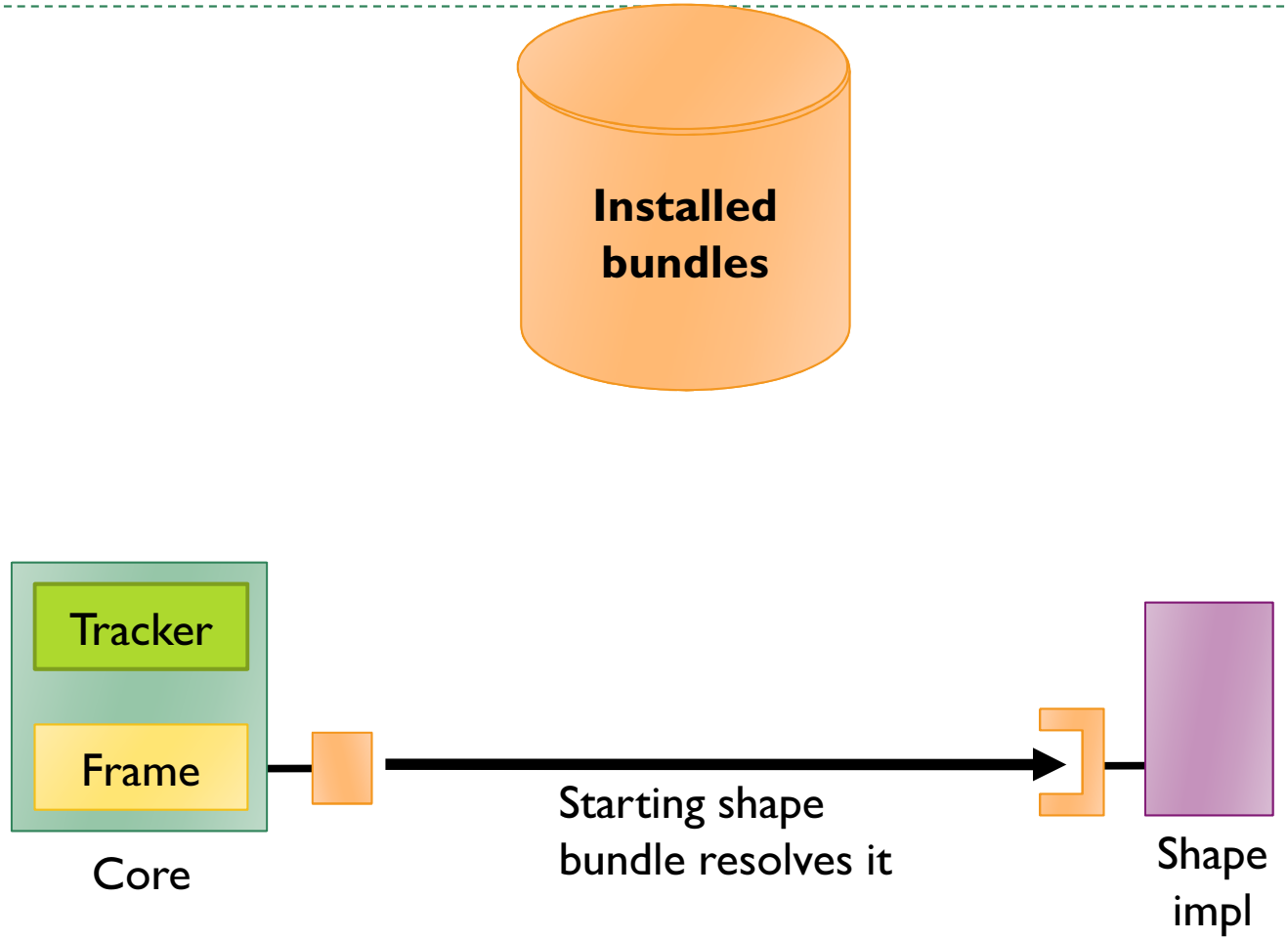


Core

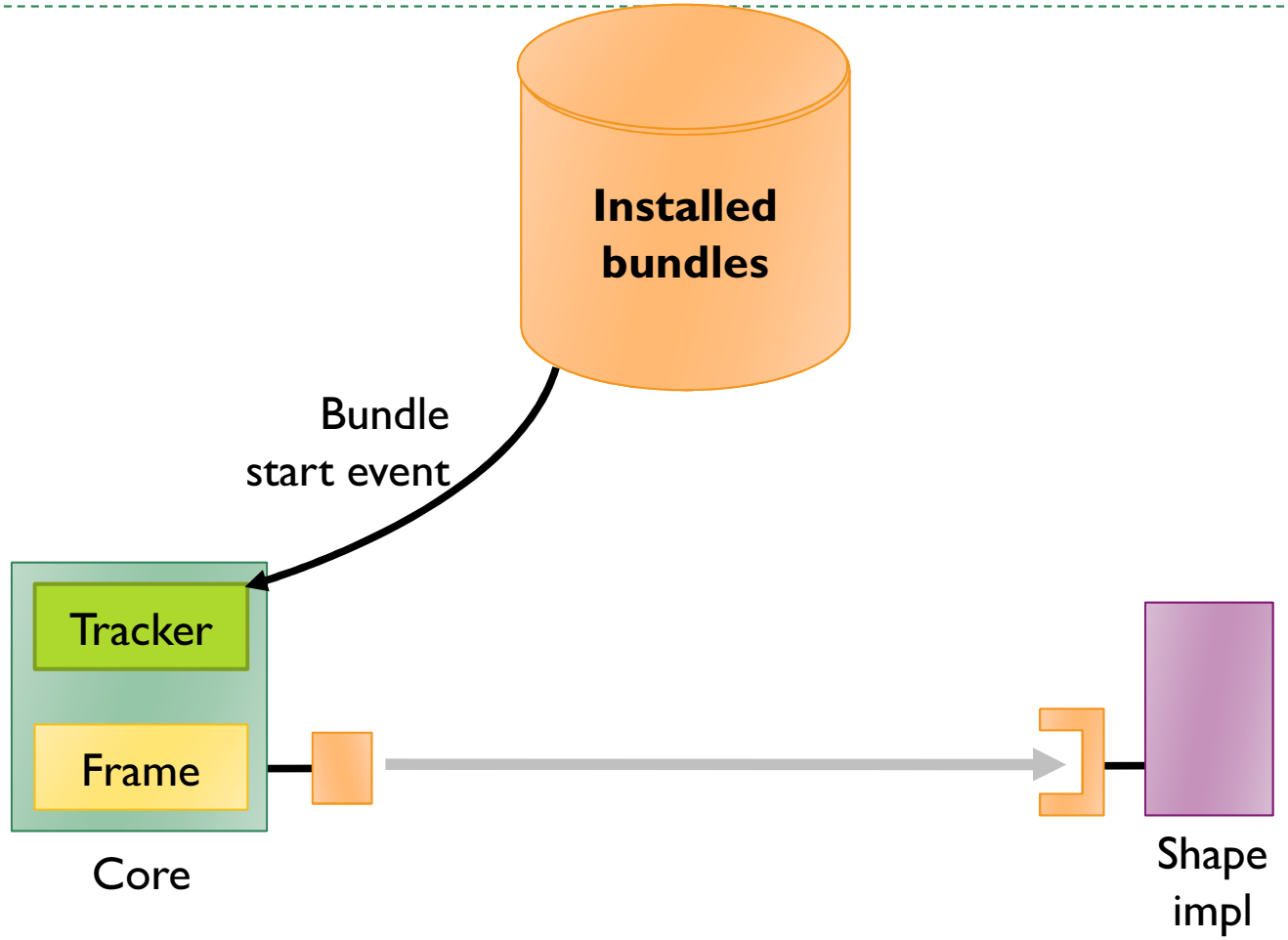
Extender Pattern



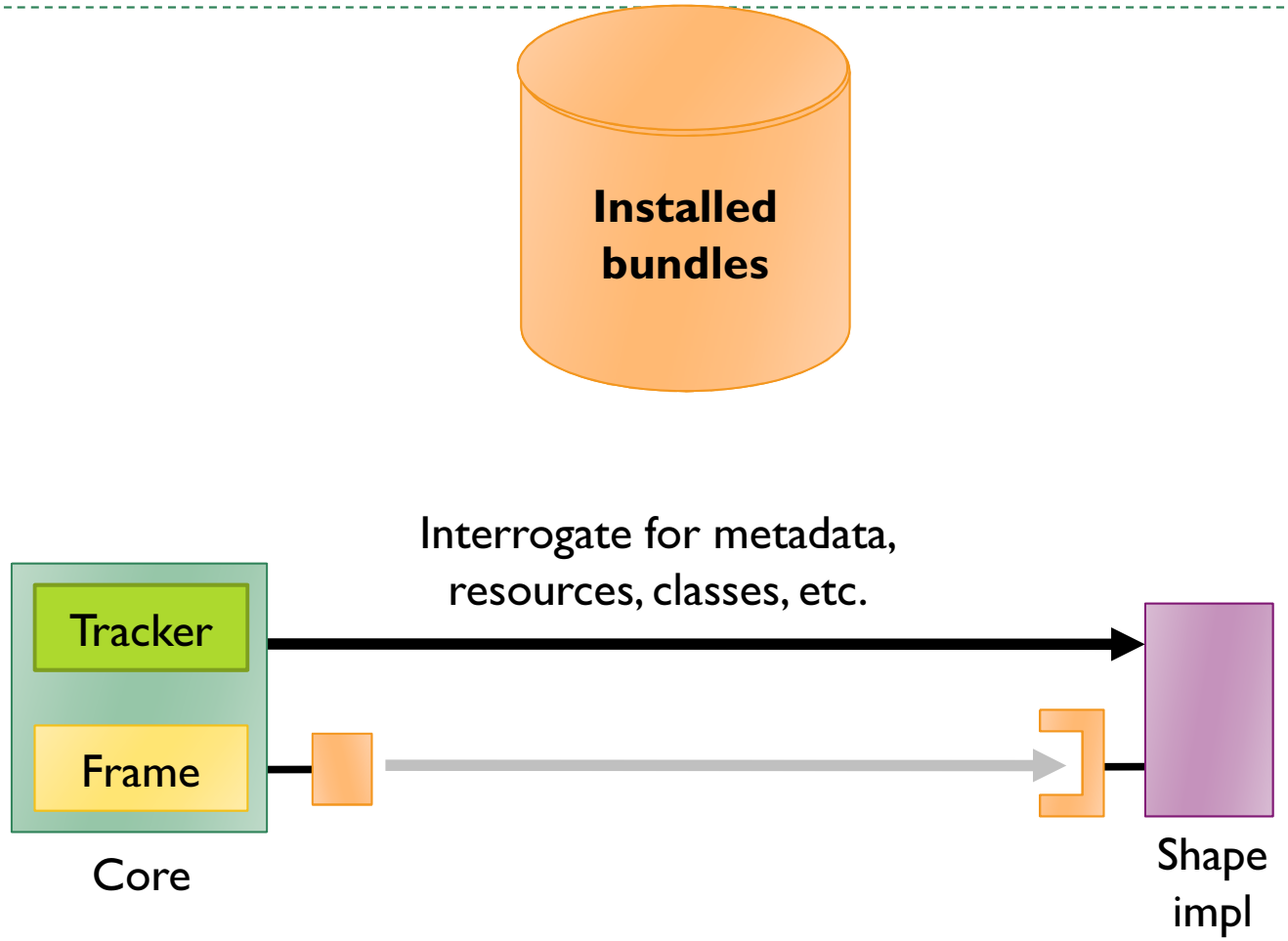
Extender Pattern



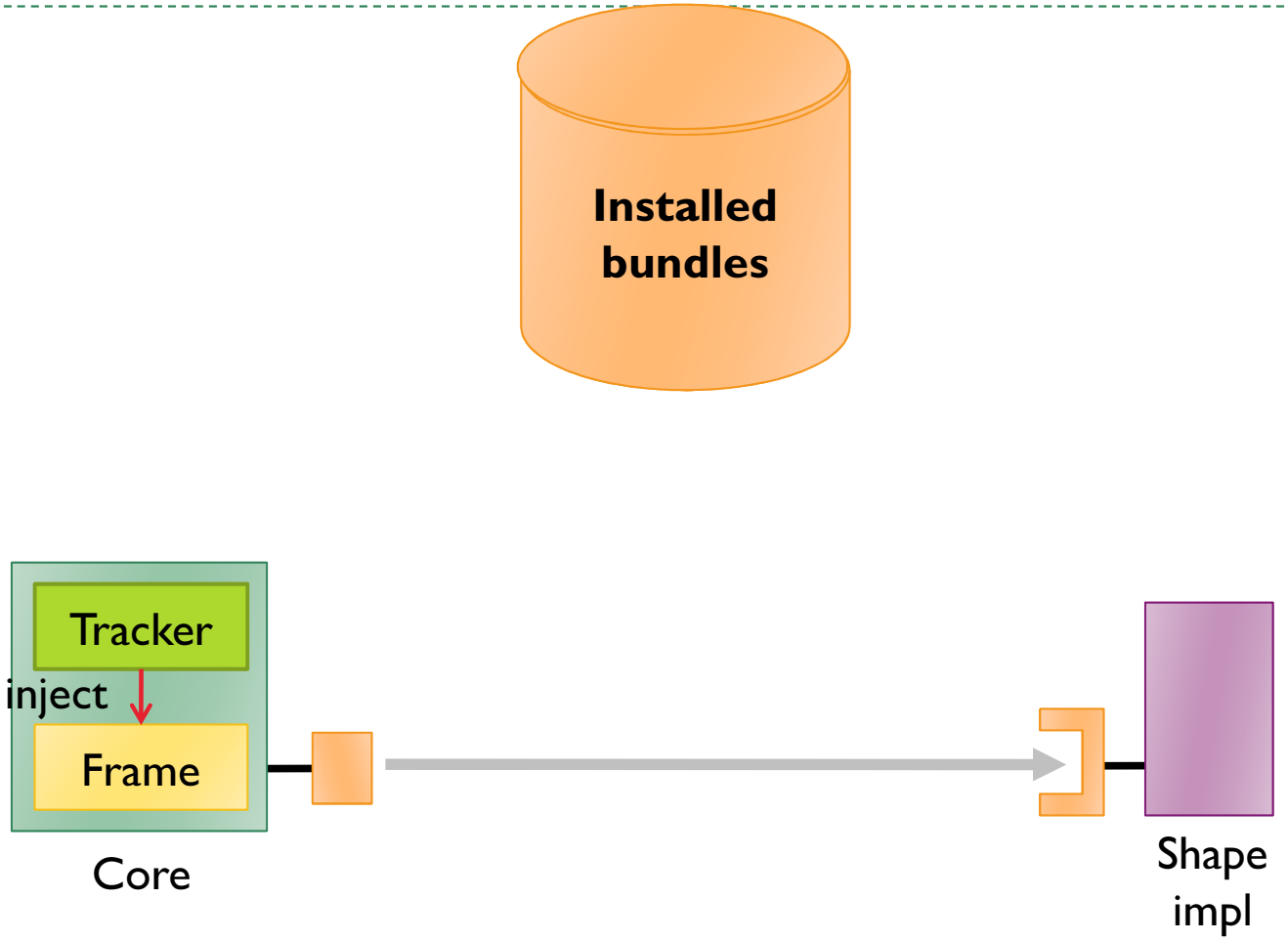
Extender Pattern



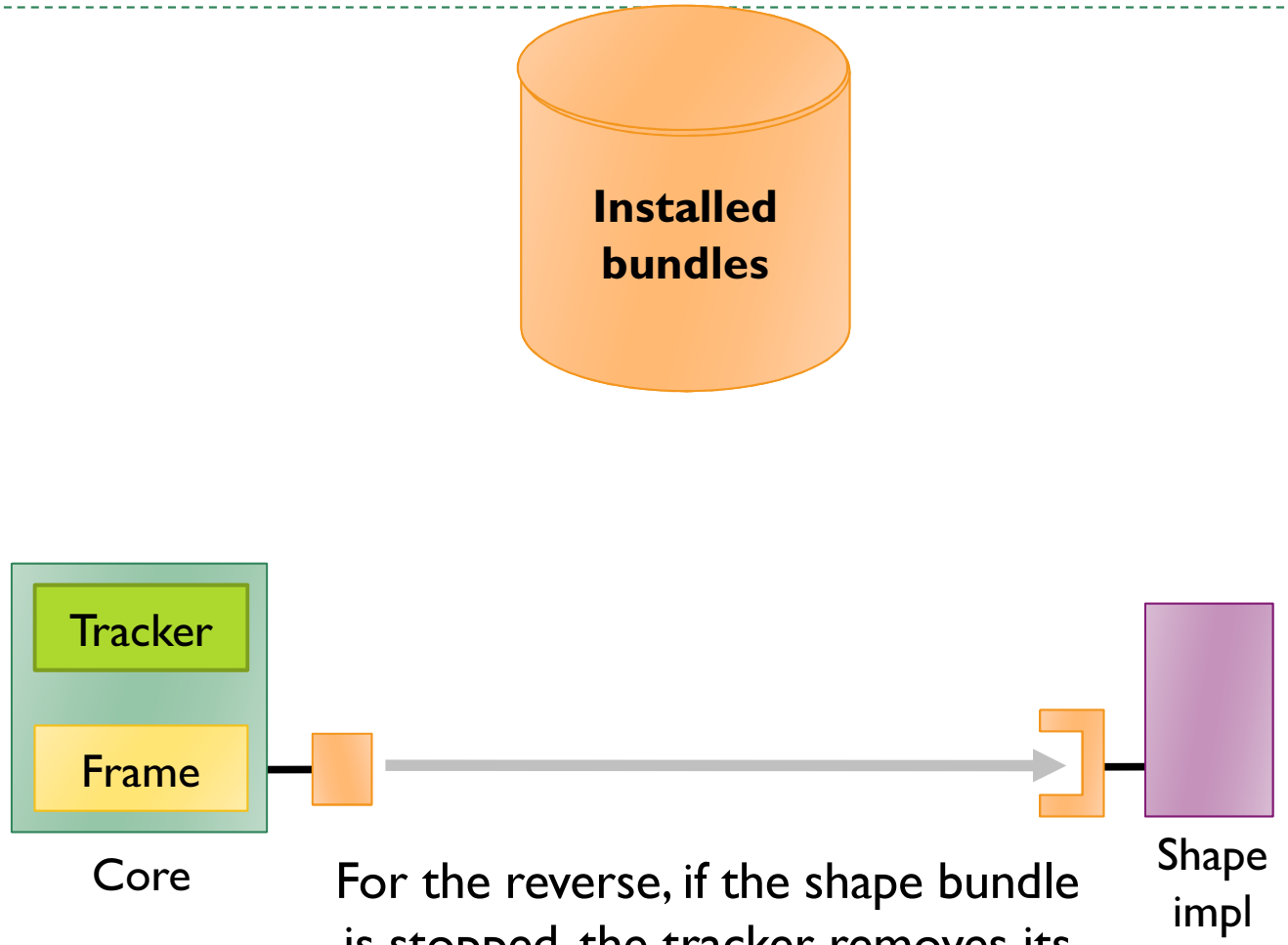
Extender Pattern



Extender Pattern



Extender Pattern



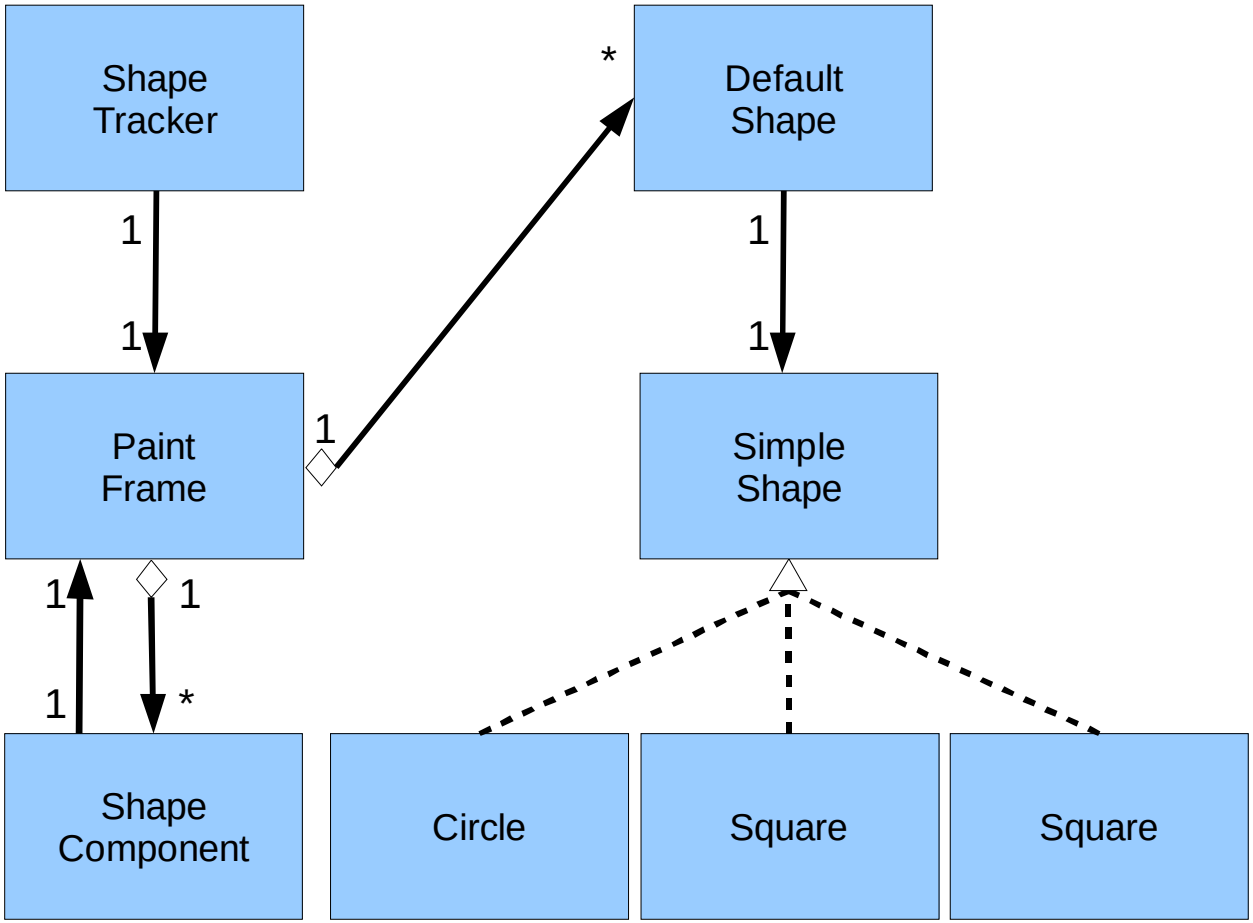
For the reverse, if the shape bundle is stopped, the tracker removes its associated shape.

Extender Paint Program Overview (1/2)

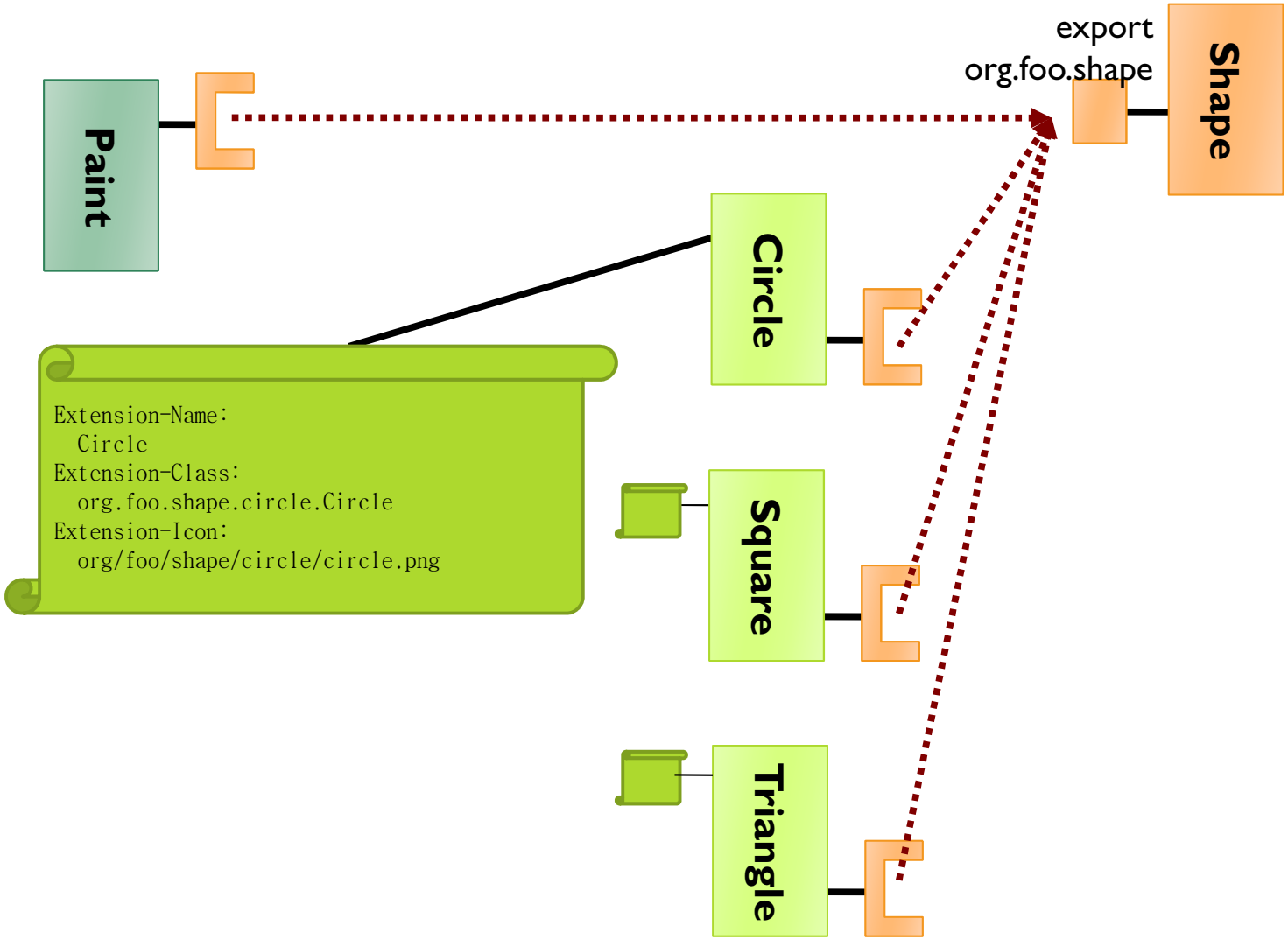
- Dynamically extensible paint program
  - Uses the extender pattern to deliver shapes
  - The paint bundle is the extender, i.e., it listens for bundles containing shapes
  - On install, the extender probes bundles to see if they are extensions
    - Special metadata in the manifest denotes the name, class, and icon of the shape
- Uses placeholder when shape has been used, but currently unavailable because the bundle is not active

Extender Paint Program Design (1/2)

- Relationship among classes



Extender Paint Program Design (2/2)



Package Admin

- Framework provides special API to deal with bundles interactions

```
public interface PackageAdmin {
  static final int BUNDLE_TYPE_FRAGMENT = 0x00000001;
  Bundle getBundle(Class clazz);
  Bundle[] getBundles(String symbolicName, String
versionRange);
  int getBundleType(Bundle bundle);
  ExportedPackage getExportedPackage(String name);
  ExportedPackage[] getExportedPackages(Bundle bundle);
  ExportedPackage[] getExportedPackages(String name);
  Bundle[] getFragments(Bundle bundle);
  RequiredBundle[] getRequiredBundles(String symbolicName);
  Bundle[] getHosts(Bundle bundle);
  void refreshPackages(Bundle[] bundles);
  boolean resolveBundles(Bundle[] bundles);
}
```

Package Admin

- Framework provides special API to deal with bundles interactions

```
public interface PackageAdmin {
  static final int BUNDLE_TYPE_FRAGMENT = 0x00000001;
  Bundle getBundle(Class clazz);
  Bundle[] getBundles(String symbolicName, String
versionRange);
  int getBundleType(Bundle bundle);
  ExportedPackage getExportedPackage(String name);
  ExportedPackage[] getExportedPackages(Bundle bundle);
  ExportedPackage[] getExportedPackages(String name);
  Bundle[] getFragments(Bundle bundle);
  RequiredBundle[] getRequiredBundles(String symbolicName);
  Bundle[] getHosts(Bundle bundle);
  void refreshPackages(Bundle[] bundles);
  boolean resolveBundles(Bundle[] bu
}
```

Provides various methods to introspect bundle dependencies

Package Admin

- Framework provides special API to deal with bundles interactions

```
public interface PackageAdmin {
  static final int BUNDLE_TYPE_FRAGMENT = 0x00000001;
  Bundle getBundle(Class clazz);
  Bundle[] getBundles(String symbolicName, String
versionRange);
  int getBundleType(Bundle bundle);
  ExportedPackage getExportedPackage(String name);
  ExportedPackage[] getExportedPackages(
  ExportedPackage[] getExportedPacka
  Bundle[] getFragments(Bundle bund
  RequiredBundle[] getRequiredBundles
  Bundle[] getHosts(Bundle bundle);
  void refreshPackages(Bundle[] bundles);
  boolean resolveBundles(Bundle[] bundles);
}
```

So, how do we gain access to this API?

The Service Layer

Service Orientation

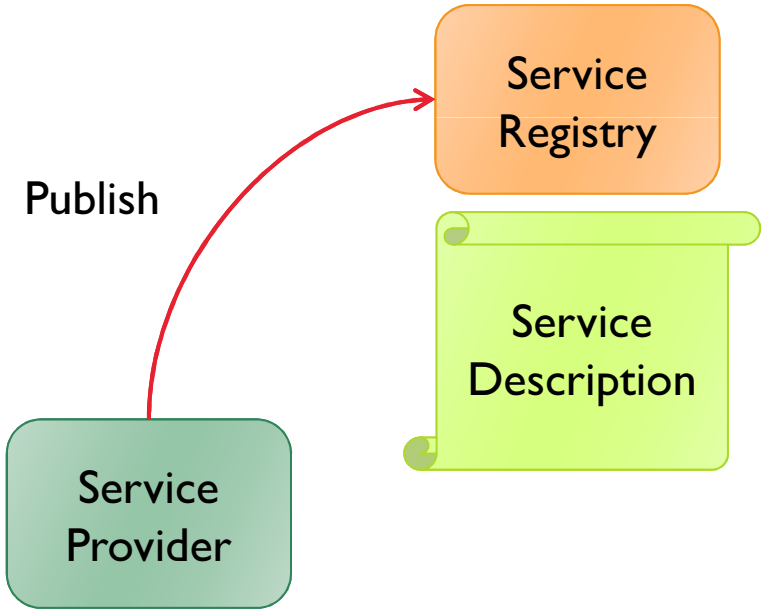
- The OSGi framework promotes a service-oriented interaction pattern among bundles

Service Orientation

- The OSGi framework promotes a service-oriented interaction pattern among bundles

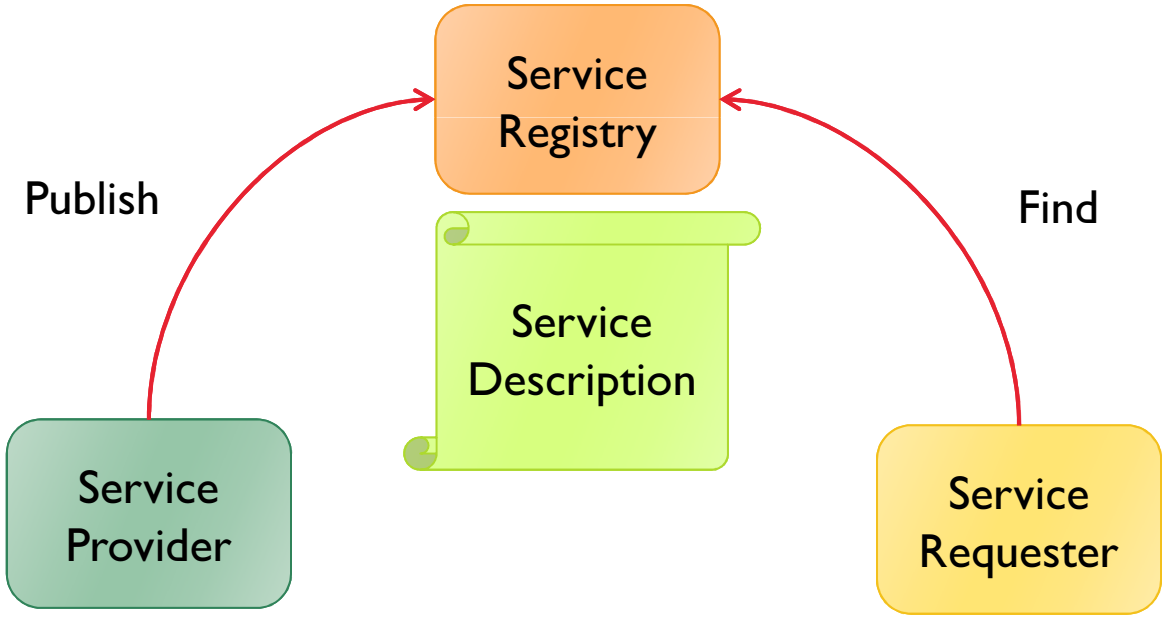
Service Orientation

- The OSGi framework promotes a service-oriented interaction pattern among bundles



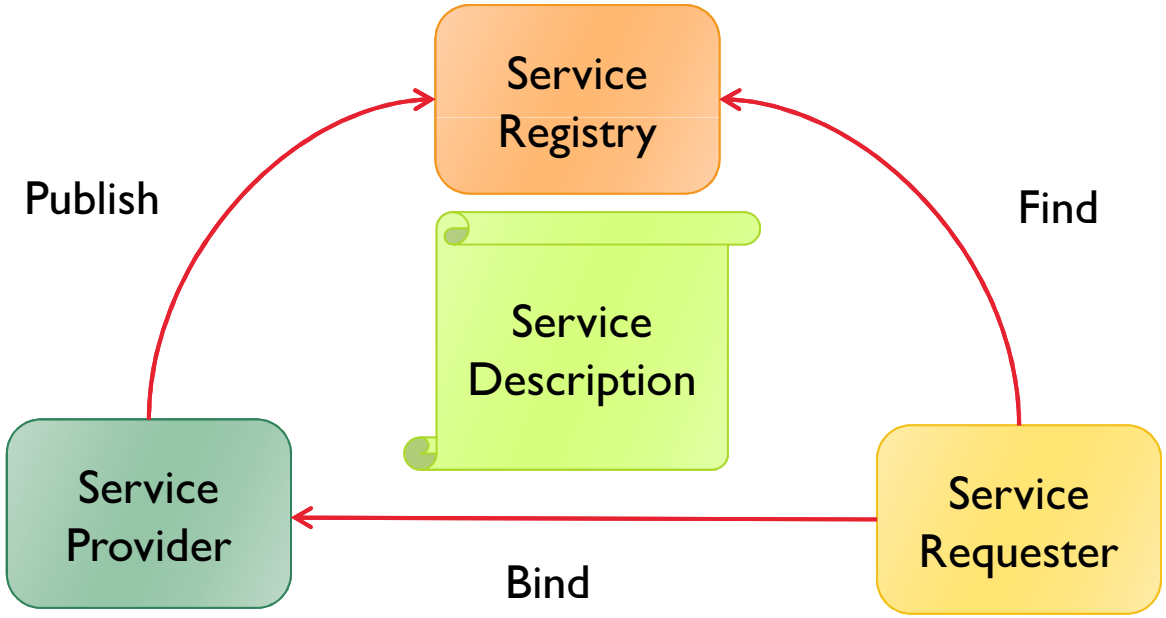
Service Orientation

- The OSGi framework promotes a service-oriented interaction pattern among bundles



Service Orientation

- The OSGi framework promotes a service-oriented interaction pattern among bundles



OSGi Service Approach Advantages

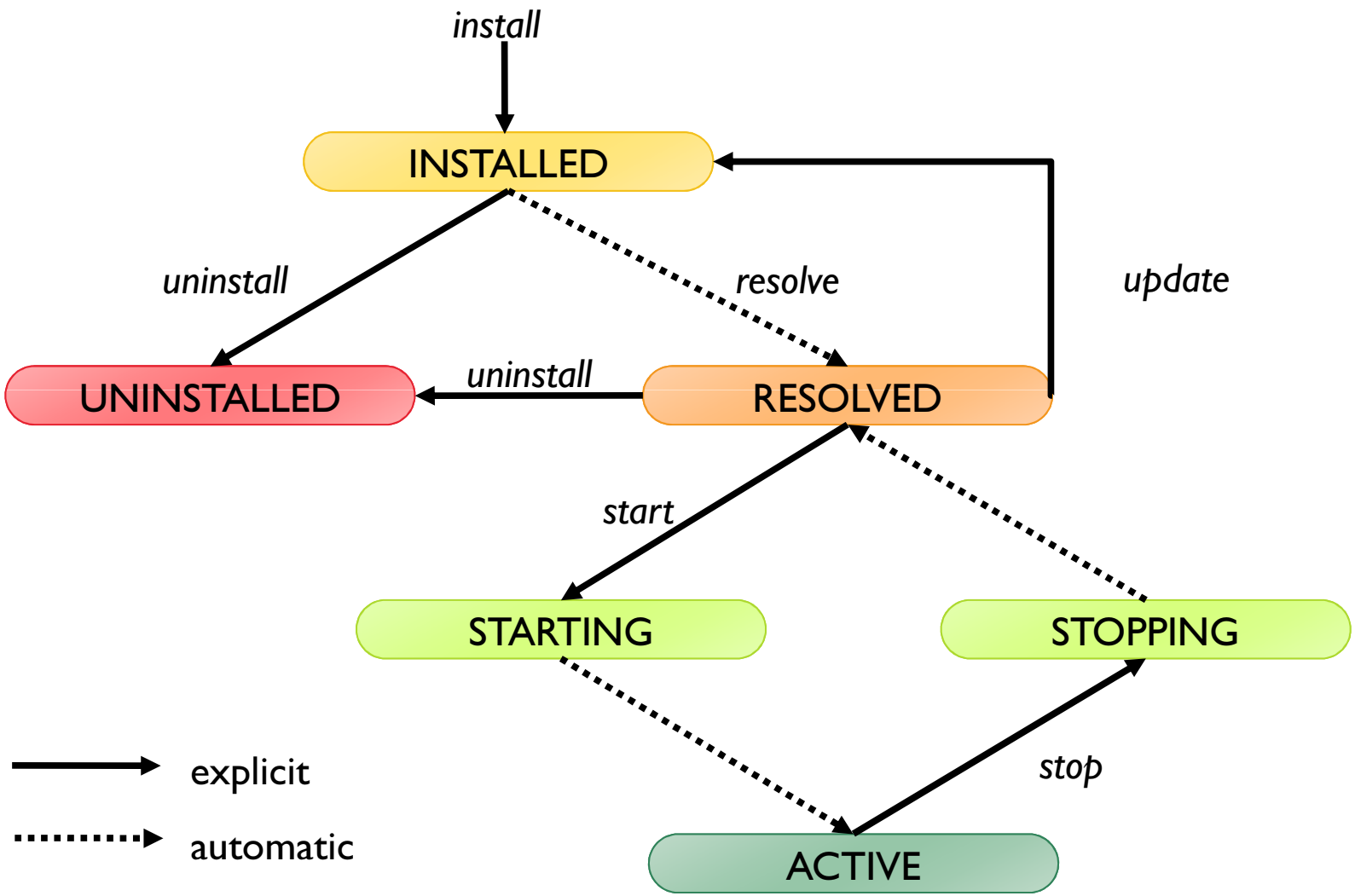
- Lightweight services
  - Direct method invocation
- Structured code
  - Promotes separation of interface from implementation
  - Enables reuse, substitutability, loose coupling, and late binding
- Dynamics
  - Loose coupling and late binding make it possible to support run-time management of module

OSGi Applications

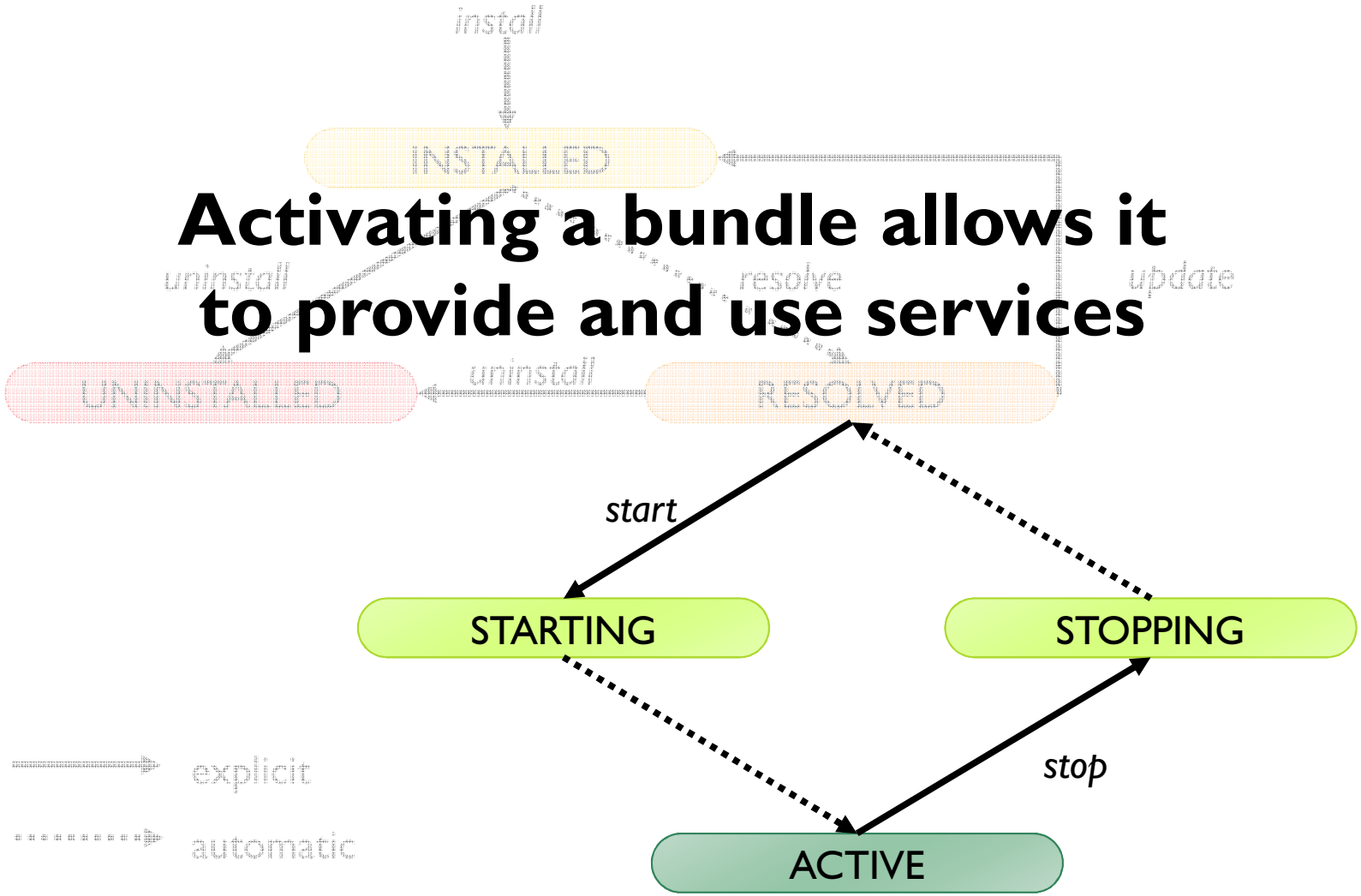
- A collection of bundles that interact via service interfaces
  - Bundles may be independently developed and deployed
  - Bundles and their associated services may appear or disappear at any time

- Resulting application follows a Service-Oriented Component Model approach
  - Combines ideas from both component and service orientation

Bundle Life Cycle (Revisited)



Bundle Life Cycle (Revisited)



Activating a bundle allows it to provide and use services

What's a Service? (in OSGi)

- Just a simple Java object
- Typically described by a Java interface
  - Allows for multiple providers
- Using a service is just like using any object

Hello World Service Example

- Let's assume we have this service interface

```
package com.foo.hello;
public interface Hello {
  void sayHello(String name);
}
```

Hello World Service Example

- Let's assume we have this service interface

```
package com.foo.hello;
public interface Hello {
  void sayHello(String name);
}
```

- And this implementation

```
package com.foo.hello.impl;
import com.foo.hello;
public class HelloImpl implements Hello {
  public void sayHello(String name) {
    System.out.println("Hello " + name + "!");
  }
}
```

Publishing a Service (1/2)

- BundleContext allows bundles to publish services

```
public interface BundleContext {
  …
  void addServiceListener(ServiceListener listener, String
filter)
  throws InvalidSyntaxException;
  void addServiceListener(ServiceListener listener);
  void removeServiceListener(ServiceListener listener);
  ServiceRegistration registerService(
    String[] clazzes, Object service, Dictionary props);
  ServiceRegistration registerService(
    String clazz, Object service, Dictionary props);
  ServiceReference[] getServiceReferences(String clazz, String
filter)
    throws InvalidSyntaxException;
  ServiceReference getServiceReference(String clazz);
  Object getService(ServiceReference reference);
  boolean ungetService(ServiceReference reference);
}
```

Publishing a Service (1/2)

- BundleContext allows bundles to publish services

```
public interface BundleContext {
  …
  void addServiceListener(ServiceListener listener, String
filter)
  throws InvalidSyntaxException;
  void addServiceListener(ServiceListener listener);
  void removeServiceListener(ServiceListener listener);
  ServiceRegistration registerService(
    String[] clazzes, Object service, Dictionary props);
  ServiceRegistration registerService(
    String clazz, Object service, Dictionary props);
  ServiceReference[] getServiceReferences(String clazz, String
filter)
    throws InvalidSyntaxException;
  ServiceReference getServiceReference(
  Object getService(ServiceReference re
  boolean ungetService(ServiceReference re
}
```

We have two methods for publishing services

Publishing a Service (2/2)

- Bundles often publish services in their activator

```
package com.foo.hello.impl;
import org.osgi.framework.*;
public class Activator implements BundleActivator {
  private ServiceRegistration m_reg = null;
  public void start(BundleContext context) {
    m_reg = context.registerService(
      com.foo.hello.Hello.class.getName(),
          new HelloImpl(), null);
  }

  public void stop(BundleContext context) {
    m_reg.unregister();
  }
}
```

Publishing a Service (2/2)

- Bundles often publish services in their activator

```
package com.foo.hello.impl;
import org.osgi.framework.*;
public class Activator implements BundleActivator {
  private ServiceRegistration m_reg = null;
  public void start(BundleContext context) {
    m_reg = context.registerService(
      com.foo.hello.Hello.class.getName(),
          new HelloImpl(), null);
  }

  public void stop(BundleContext context)
    m_reg.unregister();
  }
}
```

We register the service when starting, which makes it available to other bundles

Publishing a Service (2/2)

- Bundles often publish services in their activator

```
package com.foo.hello.impl;
import org.osgi.framework.*;
public class Activator implements BundleActivator {
  private ServiceRegistration m_reg = null;
  public void start(BundleContext context) {
    m_reg = context.registerService(
      com.foo.hello.Hello.class.getName(), new HelloImpl(),
null);
  }

  public void stop(BundleContext context) {
    m_reg.unregister();
  }
}
```

We unregister it when stopping

Packaging the Hello World Service

- Our service implementation bundle contains these packages
  - com.foo.hello
  - com.foo.hello.impl

Packaging the Hello World Service

- Our service implementation bundle contains these packages
  - com.foo.hello
  - com.foo.hello.impl
- And the following manifest metadata

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: com.foo.hello.impl
Export-Package: com.foo.hello
Import-Package: org.osgi.framework, com.foo.hello
Bundle-Activator: com.foo.hello.impl.Activator
```

Using a Service (1/2)

- BundleContext allows bundles to find services

```
public interface BundleContext {
  …
  void addServiceListener(ServiceListener listener, String
filter)
  throws InvalidSyntaxException;
  void addServiceListener(ServiceListener listener);
  void removeServiceListener(ServiceListener listener);
  ServiceRegistration registerService(
    String[] clazzes, Object service, Dictionary props);
  ServiceRegistration registerService(
    String clazz, Object service, Dictionary props);
  ServiceReference[] getServiceReferences(String clazz, String
filter)
    throws InvalidSyntaxException;
  ServiceReference getServiceReference(String clazz);
  Object getService(ServiceReference reference);
  boolean ungetService(ServiceReference reference);
}
```


OSGi in Action - Clement Escoffier
(clement.escoffier@akquinet.de)

Using a Service (1/2)

- BundleContext allows bundles to find services

```
public interface BundleContext {
  …
  void addServiceListener(ServiceListener listener, String
filter)
  throws InvalidSyntaxException;
  void addServiceListener(ServiceListener lis
  void removeServiceListener(ServiceListen
  ServiceRegistration registerService(
    String[] clazzes, Object service, Dictio
  ServiceRegistration registerService(
    String clazz, Object service, Dictionary props);
  ServiceReference[] getServiceReferences(String clazz, String
filter)
    throws InvalidSyntaxException;
  ServiceReference getServiceReference(String clazz);
  Object getService(ServiceReference reference);
  boolean ungetService(ServiceReference reference);
}
```

We have methods to find service references and get service objects

Using a Service (2/2)

- Bundles retrieve service references
  - Indirect references to service object

```
package com.foo.hello.client;
import org.osgi.framework.*;
import com.foo.hello.Hello;
public class HelloClient implements BundleActivator {
  public void start(BundleContext context) {
    ServiceReference ref = context.getServiceReference(
      com.foo.hello.Hello.class.getName());
    if (ref != null) {
      Hello h = (Hello) context.getService(ref);
      if (h != null) {
        h.sayHello("World");
        context.ungetService(h);
      }
    }
  }
…
}
```

Using a Service (2/2)

- Bundles retrieve service references
  - Indirect references to service object

```
package com.foo.hello.client;
import org.osgi.framework.*;
import com.foo.hello.Hello;
public class HelloClient implements BundleActivator {
  public void start(BundleContext context) {
    ServiceReference ref = context.getServiceReference(
      com.foo.hello.Hello.class.getName());
    if (ref != null) {
      Hello h = (Hello) context.getSe vice(ref);
      if (h != null) {
        h.sayHello("World");
        context.ungetService(h);
      }
    }
  }
...
}
```

We first get the service reference

Using a Service (2/2)

- Bundles retrieve service references
  - Indirect references to service object

```
package com.foo.hello.client;
import org.osgi.framework.*;
import com.foo.hello.Hello;
public class HelloClient implements BundleActivator {
  public void start(BundleContext context) {
    ServiceReference ref = context.getServiceReference(
      com.foo.hello.Hello.class.getName());
    if (ref != null) {
      Hello h = (Hello) context.getService(ref);
      if (h != null) {
        h.sayHello("World");
        context.ungetService(h);
      }
    }
  }
…
}
```

From the service reference we get the service object

Using a Service (2/2)

- Bundles retrieve service references
  - Indirect references to service object

```
package com.foo.hello.client;
import org.osgi.framework.*;
import com.foo.hello.Hello;
public class HelloClient implements BundleActivator {
  public void start(BundleContext context) {
    ServiceReference ref = context.getServiceReference(
      com.foo.hello.Hello.class.getName());
    if (ref != null) {
      Hello h = (Hello) context.getService(ref);
      if (h != null) {
        h.sayHello("World");
        context.ungetService(ref);
      }
    }
  }
...
}
```

We use the service like a normal object

Using a Service (2/2)

- Bundles retrieve service references
  - Indirect references to service object

```
package com.foo.hello.client;
import org.osgi.framework.*;
import com.foo.hello.Hello;
public class HelloClient implements BundleActivator {
  public void start(BundleContext context) {
    ServiceReference ref = context.getServiceReference(
      com.foo.hello.Hello.class.getName());
    if (ref != null) {
      Hello h = (Hello) context.getService(ref);
      if (h != null) {
        h.sayHello("World");
        context.ungetService(h);
      }
    }
  }
...
}
```

And release the service object when we are done with it

Packaging the Hello World Service

- Our client implementation bundle contains this package
  - com.foo.hello.client
- And the following manifest metadata

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: com.foo.hello.client
Import-Package: com.foo.hello, org.osgi.framework
Bundle-Activator: com.foo.hello.client.Activator
```

Service Dynamism

- Services can be published and revoked at run time
  - Service events signal service changes
    - Must track events for any services being used

<u>To listen for events</u>

```
BundleContext.addServiceListener()
```

Services Dynamism

- Services can be published and revoked at run time
  - Service events signal service changes
    - Must track events for any services being used

Implement listener interface

```
public interface ServiceListener extends EventListener {
  public void serviceChanged(ServiceEvent event);
}
```

Services Dynamism

- Services can be published and revoked at run time
  - Service events signal service changes
    - Must track events for any services being used

Received event

```
public class ServiceEvent extends EventObject {
  public final static int REGISTERED    = 0x00000001;
  public final static int MODIFIED      = 0x00000002;
  public final static int UNREGISTERING = 0x00000004;
  …
  public ServiceReference getServiceReference() { … }
  public int getType() { … }
}
```

Services Dynamism

- Services can be published and revoked at run time
  - Service events signal service changes
    - Must track events for any services being used

Received event

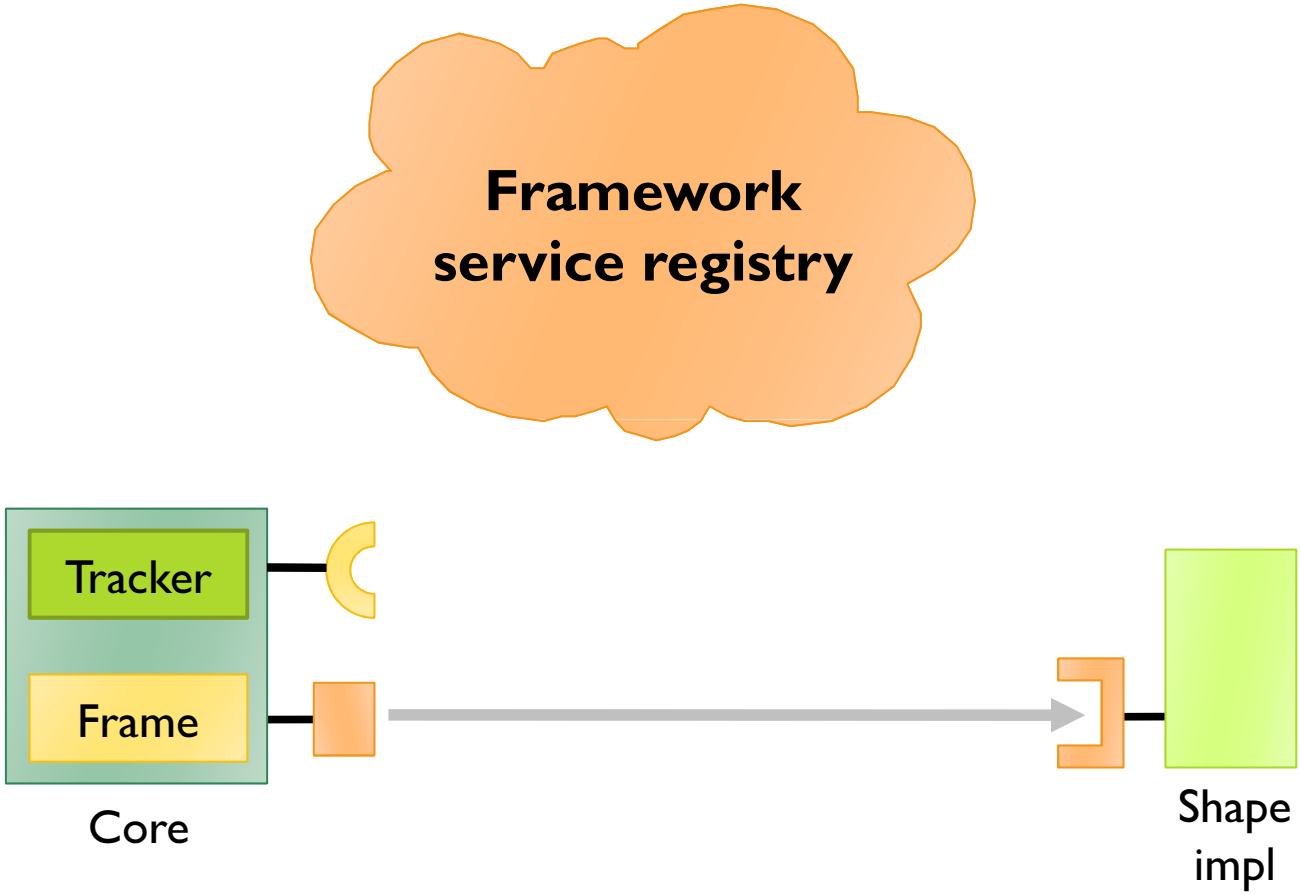
```
public class ServiceEvent extends EventObject {
  public final static int REGISTERED    = 0x00000001;
  public final static int MODIFIED      = 0x00000002;
  public final static int UNREGISTERING = 0x00000004;
  …
  public ServiceReference getServiceReference() { … }
  public int getType() { … }
}
```

Even though services are just normal objects, they are potentially much more volatile, so service events are very important

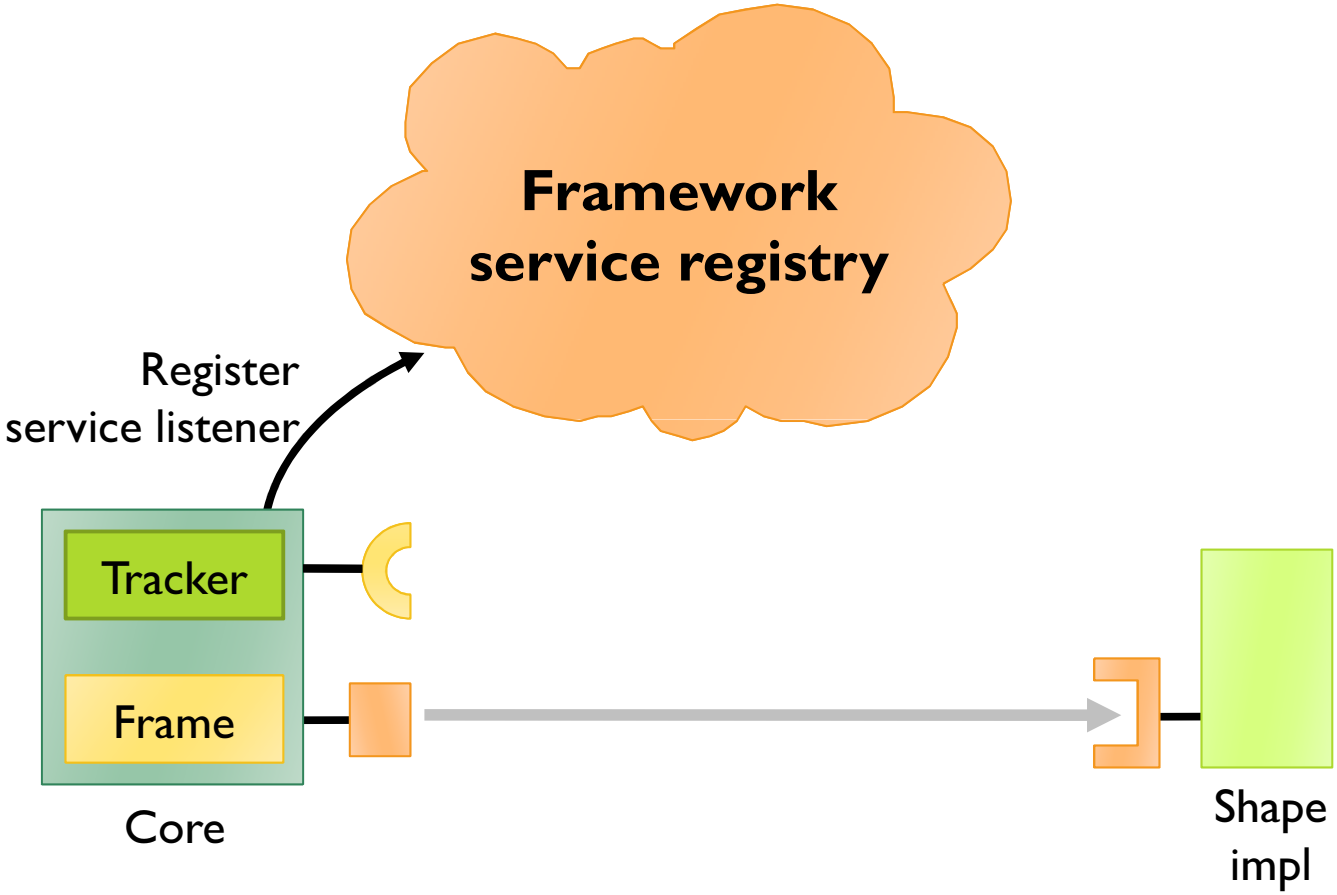
Service-Based Dynamic Extensibility

- Service events provide a mechanism for dynamic extensibility
- The whiteboard pattern
  - Treats the service registry as a whiteboard
    - A reverse way to create a service
  - An application component listens for services of a particular type to be added and removed
  - On addition, the service is integrated into the application
  - On removal, the service is removed from the application

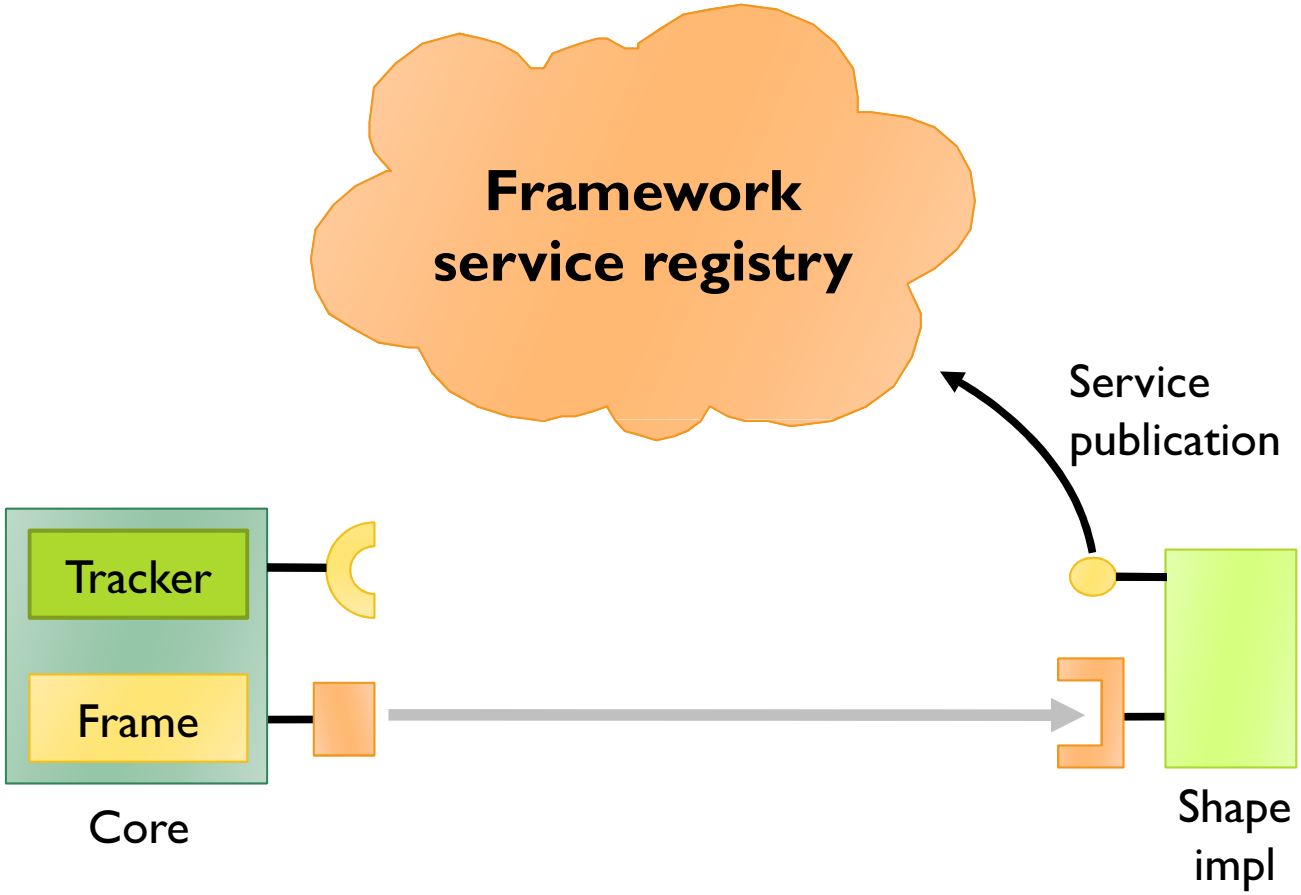
Whiteboard Pattern



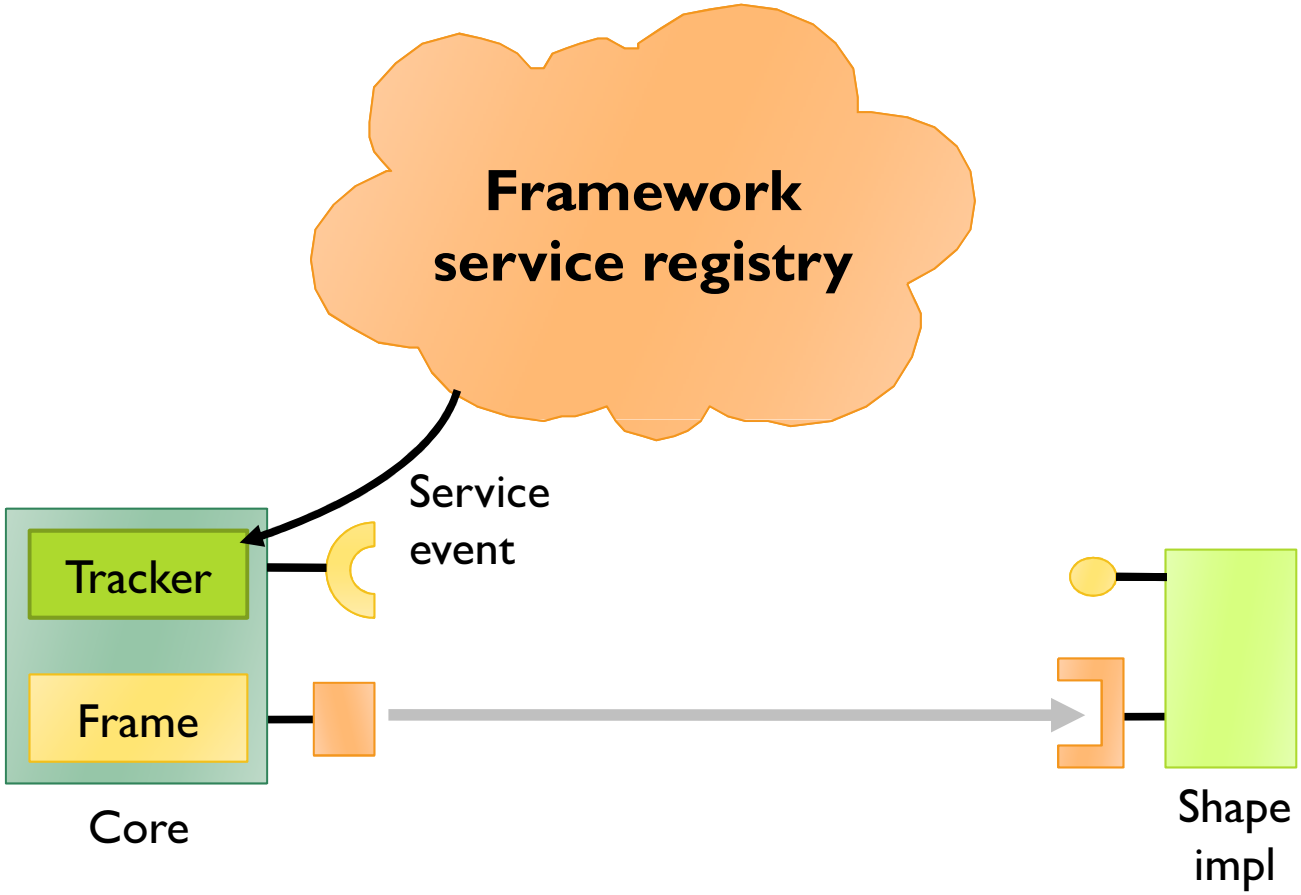
Whiteboard Pattern



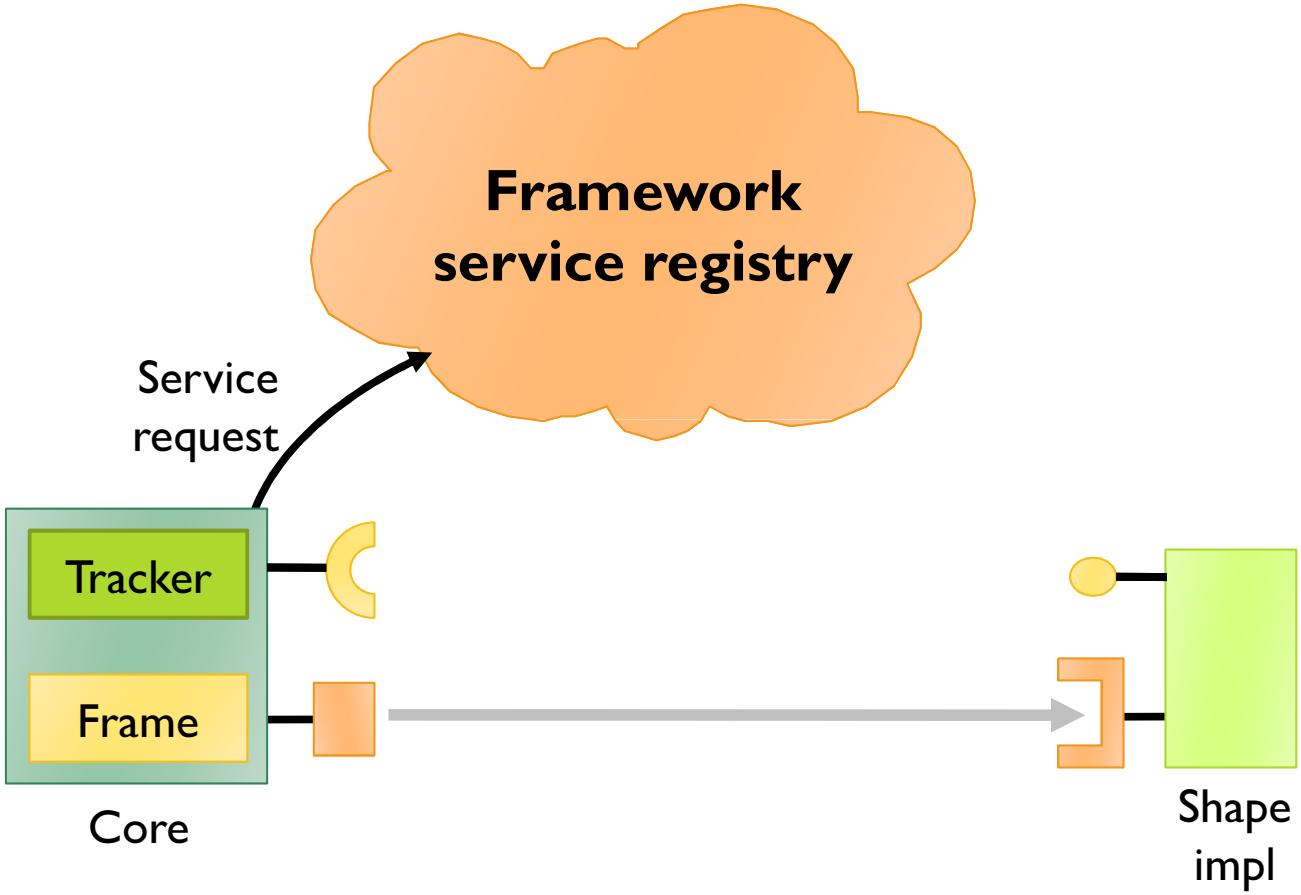
Whiteboard Pattern



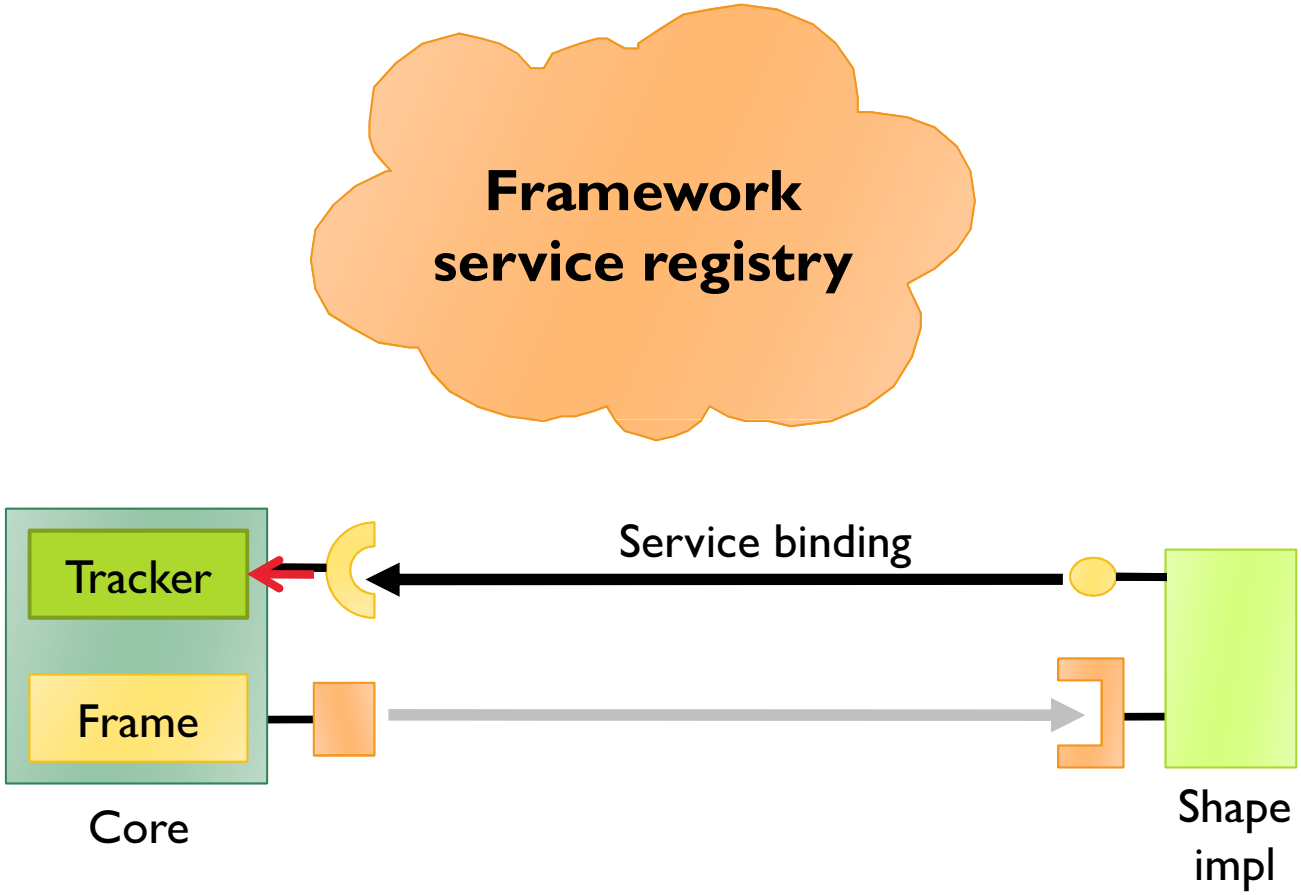
Whiteboard Pattern



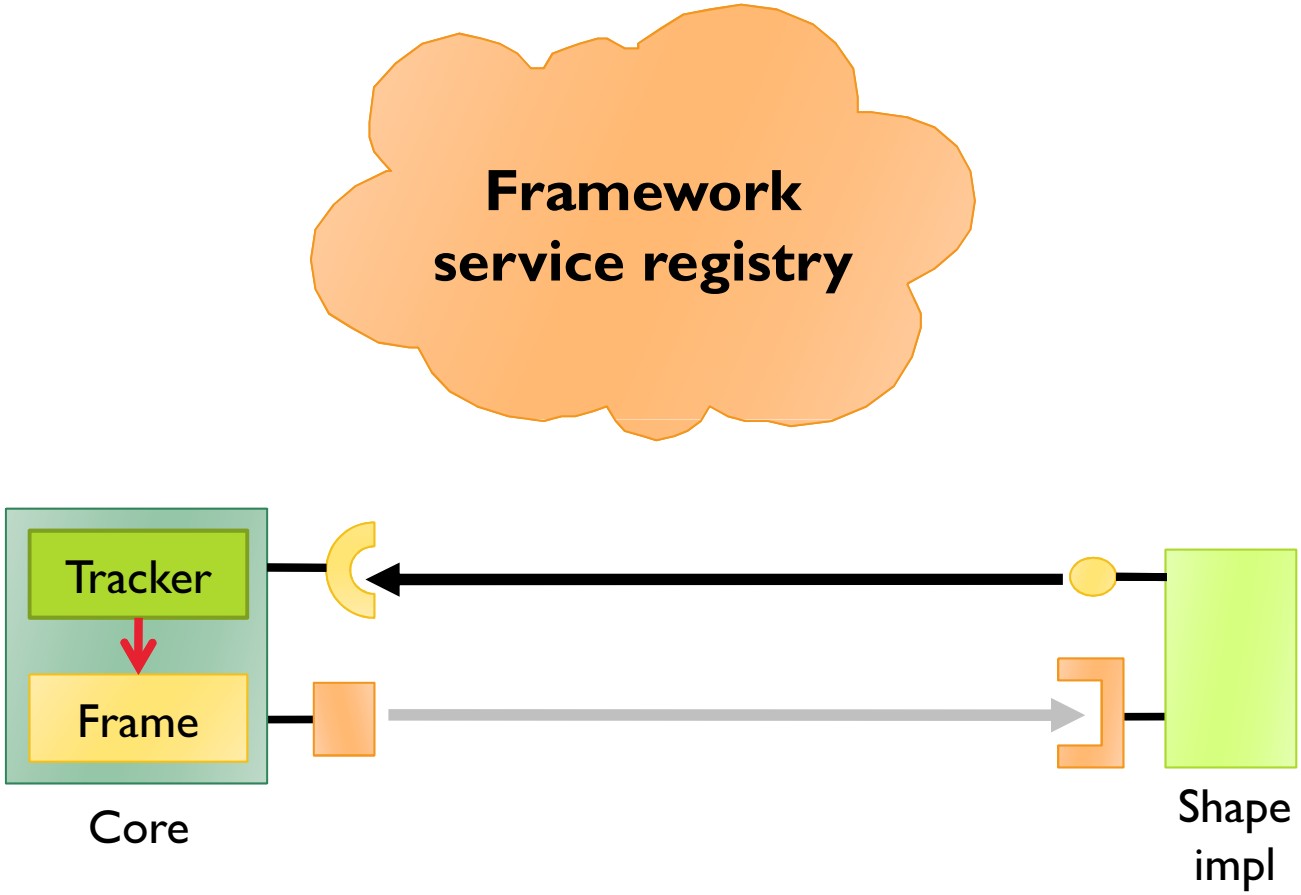
Whiteboard Pattern



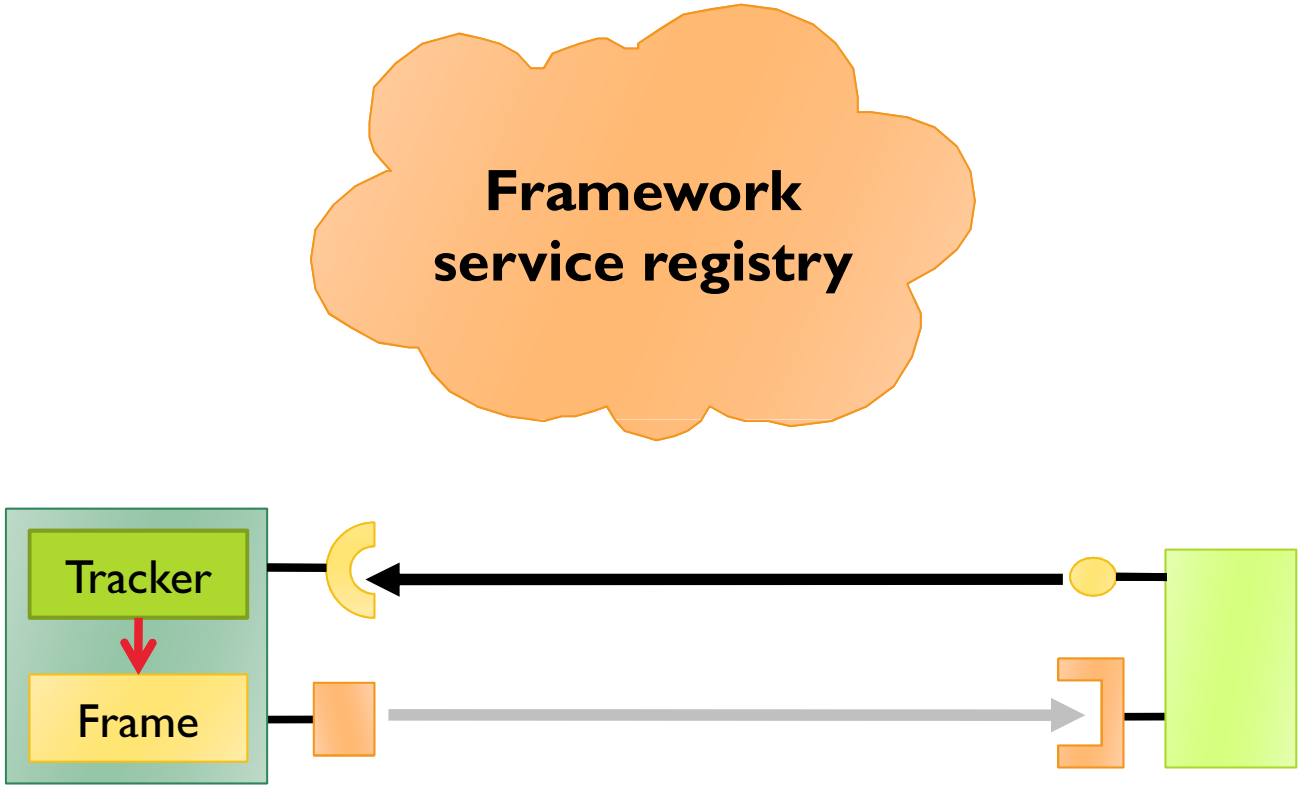
Whiteboard Pattern



Whiteboard Pattern



Whiteboard Pattern



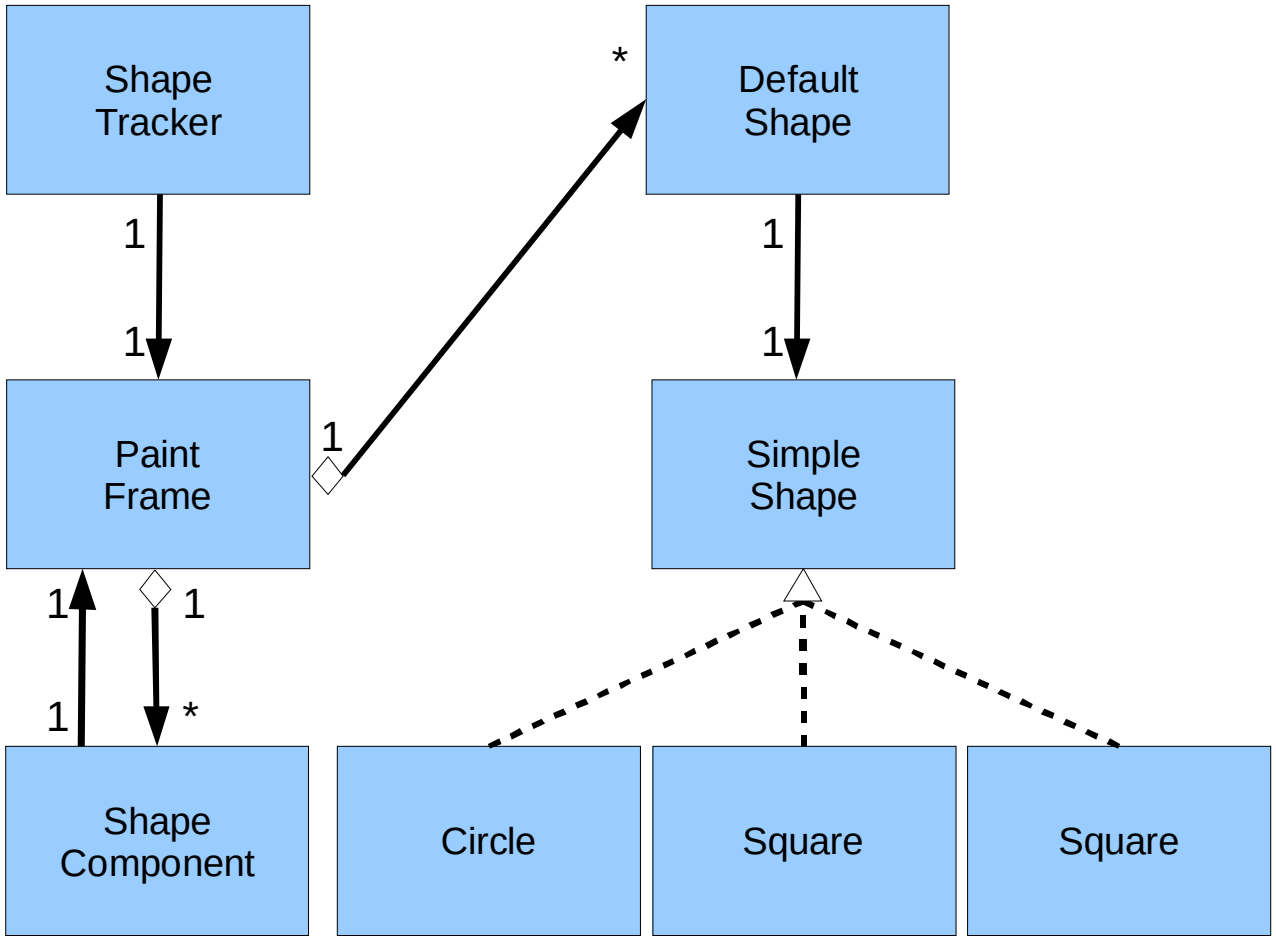
For the reverse, if the shape service is removed, the tracker removes its associated shape.

Service Paint Program Overview

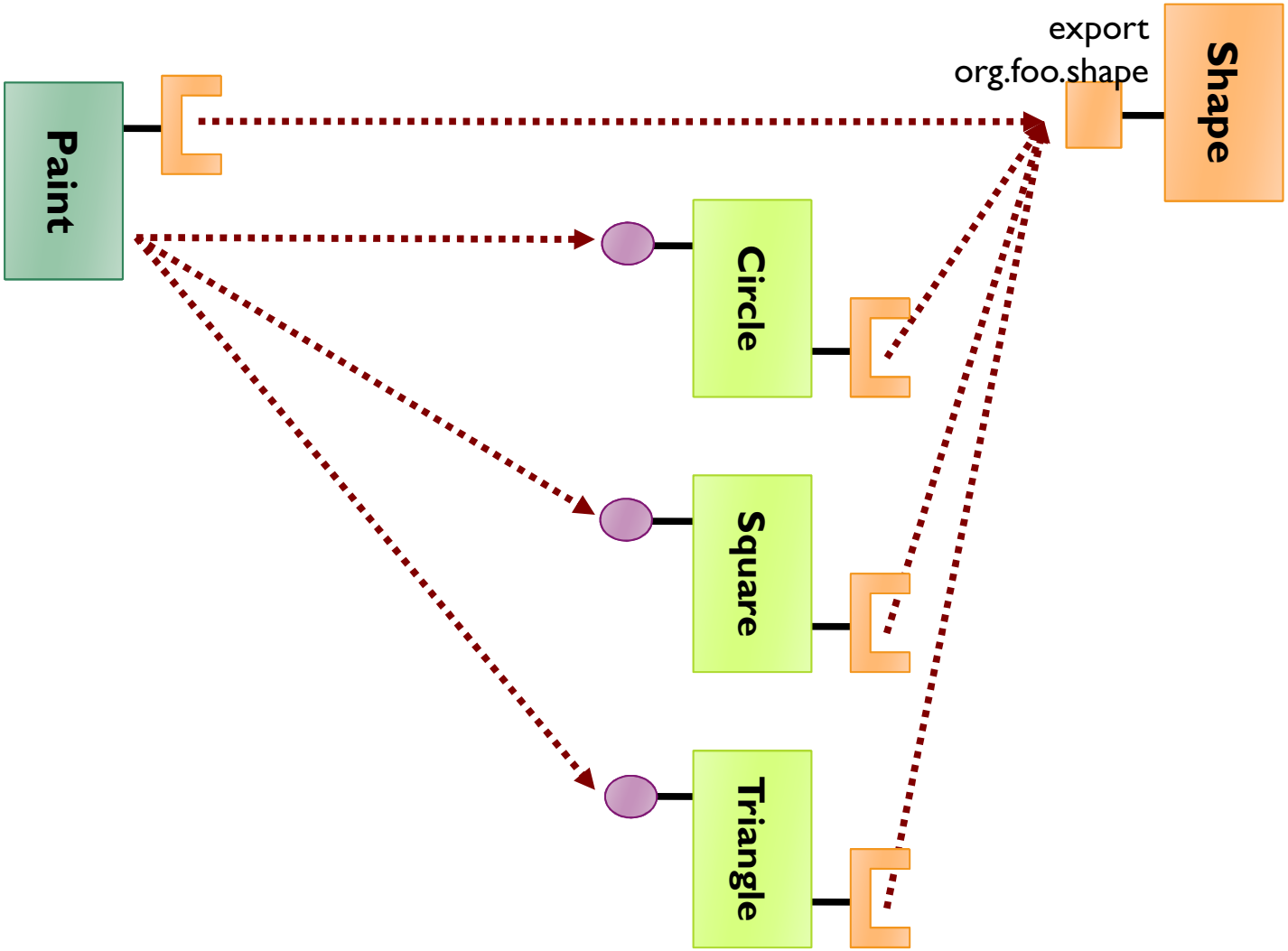
- Dynamically extensible paint program
  - Uses the whiteboard pattern to deliver shapes
  - The paint bundle listens for shape services that come and go
  - Uses service properties for the name and icon of the shape
- Uses placeholder when shape has been used, but is currently unavailable because the service is not available

Service Paint Program Design (1/2)

- Relationship between classes



Service Paint Program Design (2/2)



Challenges of Dynamism

- Both bundles and services are dynamic
- OSGi is inherently multi-threaded
- This means you have to deal with the fact that
  - Your application will likely see multiple threads
  - Application components can appear or disappear at any time
- There is help
  - Service Tracker
  - Declarative Services

Advanced Service Handling

Service-Component Model

- Why ?
  - Simplification of the development model
    - Dynamism
    - Management
    - Reconfiguration
  - Architectural view
  - Allow to easily create sophisticated applications

Service-Component Model

- Why ?
  - Simplification of the development model
    - Dynamism
    - Management
    - Reconfiguration
  - Architectural view
  - Allow to easily create sophisticated applications

- Service-Component models
  - Infuse service-oriented mechanisms in a component model
  - Provide
    - Simple development model
    - Architectural views, composition mechanisms

Existing Service Component Models

- Declarative Services
  - Specified in OSGi R4
  - Define a declarative component model to deal with the service dynamism
- Spring Dynamic Modules
  - Spring on the top of OSGi
  - Beans can use services and be exposed as services
- Apache Felix iPOJO
  - POJO-based component model
  - Extensible
    - Is not limited to dynamism
  - Supports annotations
  - The most advanced today

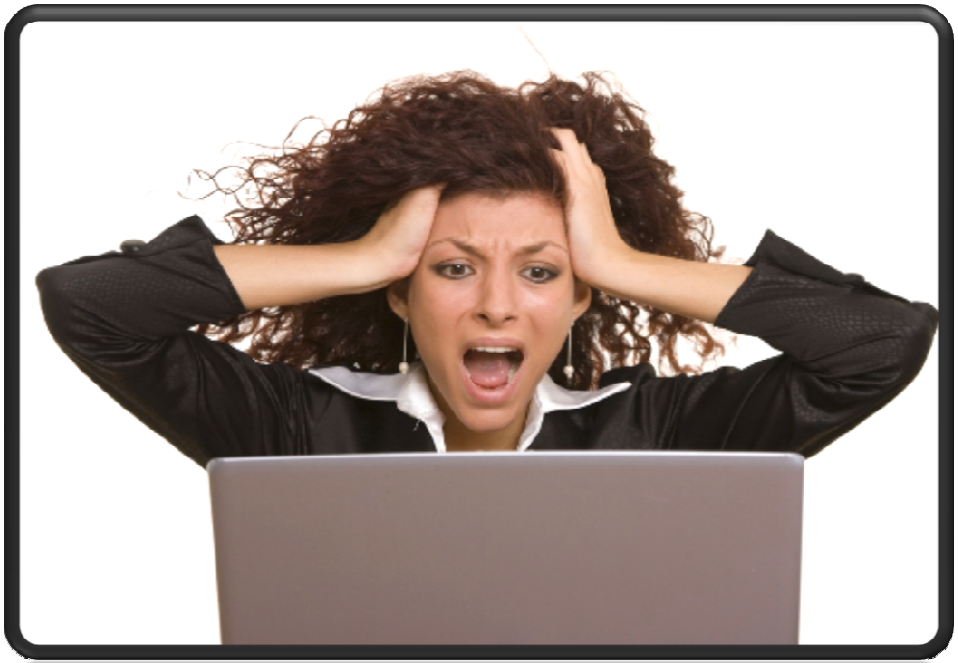
Conclusion

Conclusions

- We've seen all OSGi has to offer
  - Module layer
  - Lifecycle layer
  - Service layer
  - Advanced service handling

- While there are plenty of more details to these layers, you should now be familiar with the most important parts
  - The most commonly used/needed features
  - The most commonly used patterns

Questions ?



Ada Diaconescu: ada.diaconescu_at_telecom-paristech.fr