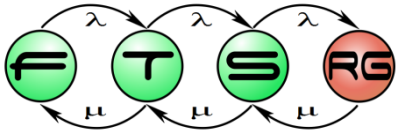
OSGi



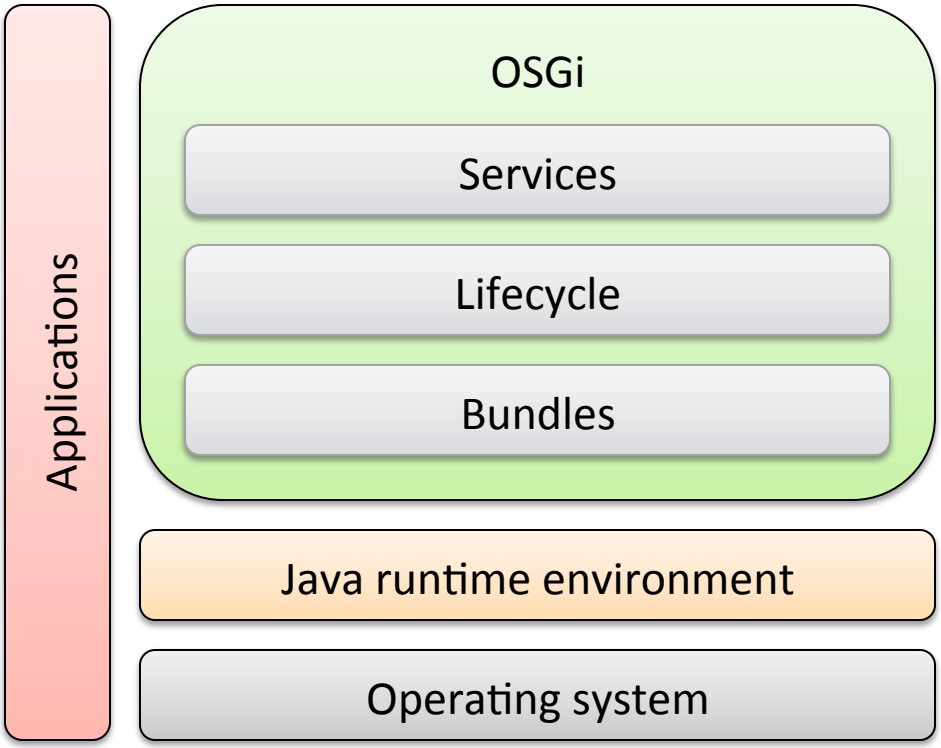
OSGi

- „A dynamic module system for Java”
- OSGI Alliance (www.osgi.org)
  - → ~30 members (Nokia, IBM, NTT, Motorola, etc.)

OSGi

- Common problems
  - Integration
  - Versioning
  - Lifecycle
- Common specification
  - Component based
  - Common integration primitives
  - Version 5.0 (2012 June)
    - Version R6 finalized in March 2014

OSGi

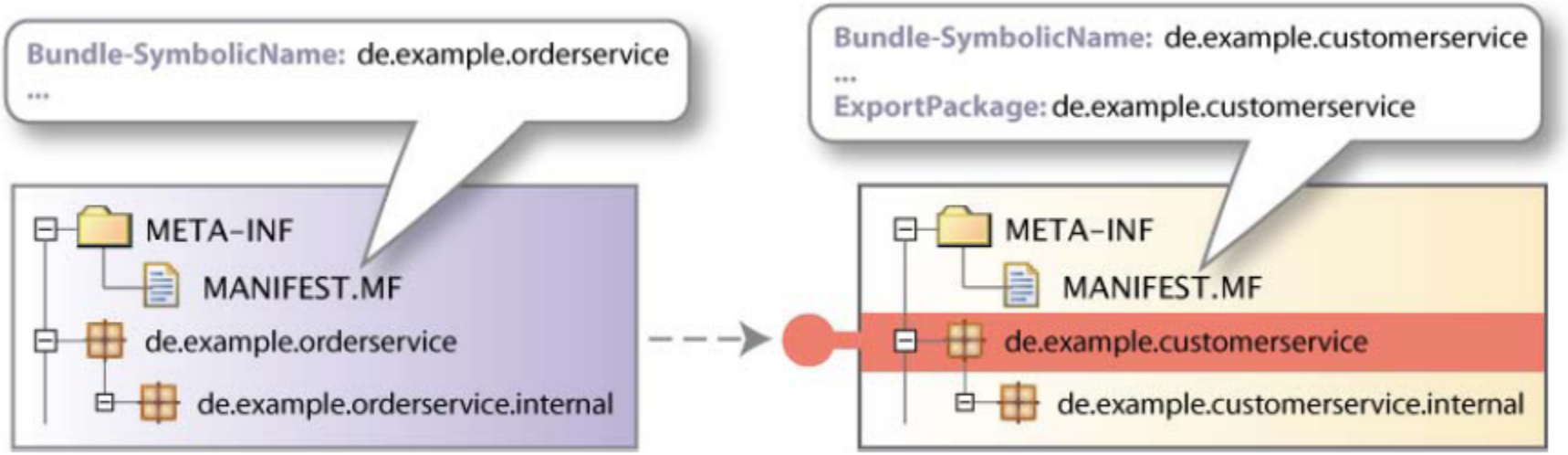

Applications
OSGi
Services
Lifecycle
Bundles
Java runtime environment
Operating system

OSGi Basics: Bundles

- Public and Private API
- Dependency Management
- Versioning

OSGi Basics: Bundles

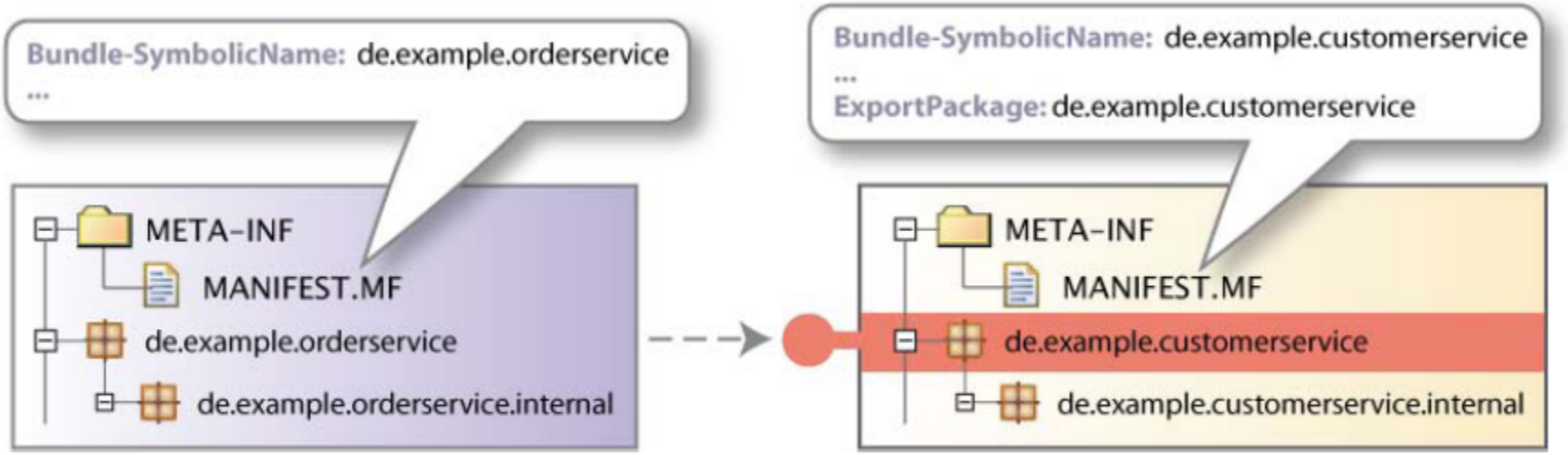
- Public and Private API
- Dependency Management
- Versioning



OSGi Basics: Bundles

- Public and Private API
- Dependency Management
- Versioning

The same as Eclipse plug-ins

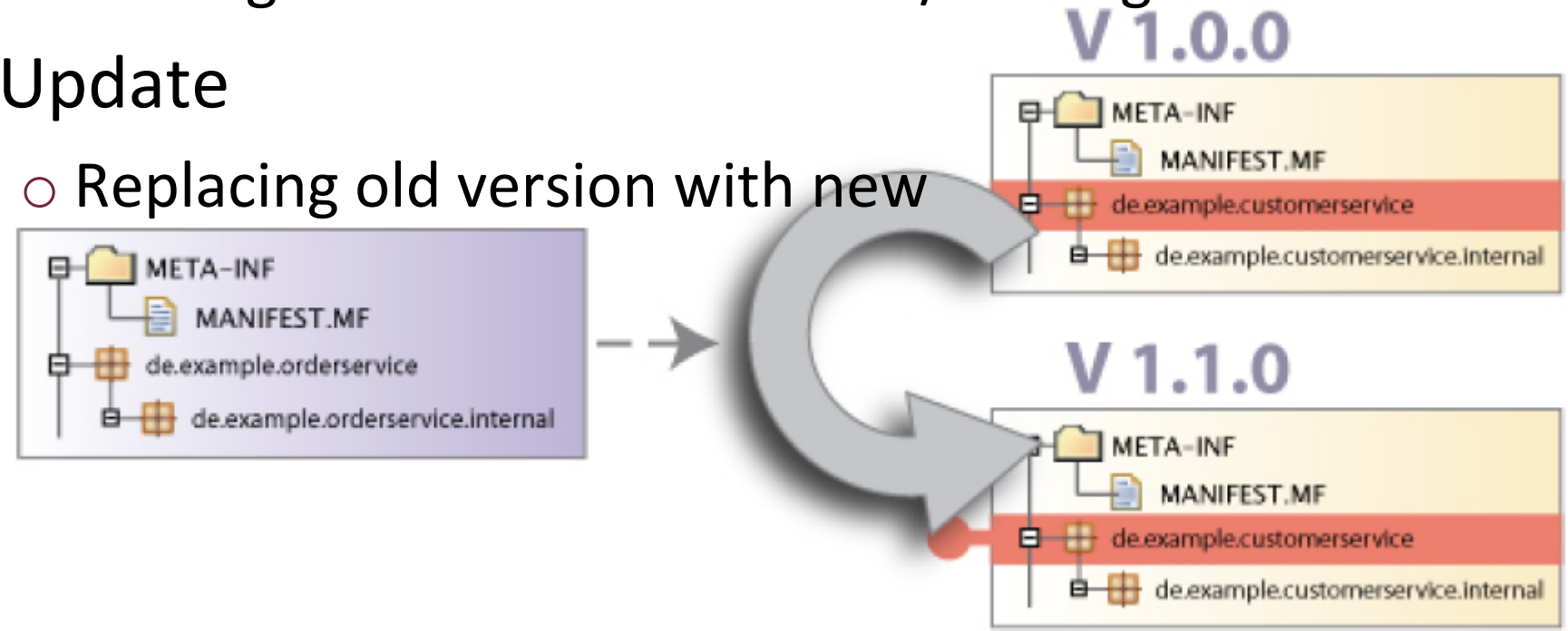


OSGi Basics: Lifecycle

- Install
  - Making bundle available
- Start/stop
  - Making contributions available/missing
- Update
  - Replacing old version with new

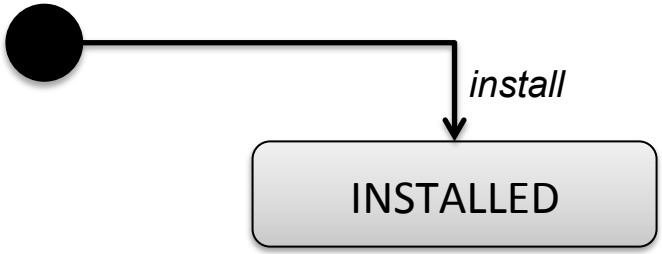
OSGi Basics: Lifecycle

- Install
  - Making bundle available
- Start/stop
  - Making contributions available/missing
- Update
  - Replacing old version with new

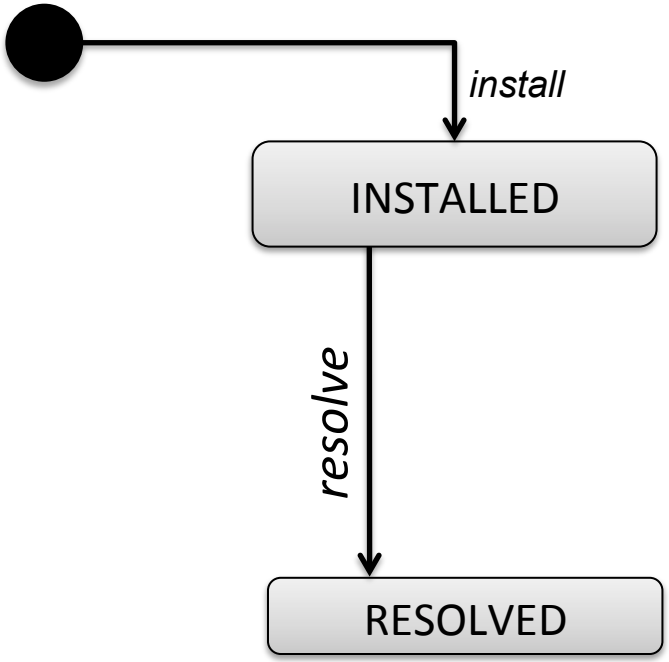


OSGi Basics: Bundle Lifecycle

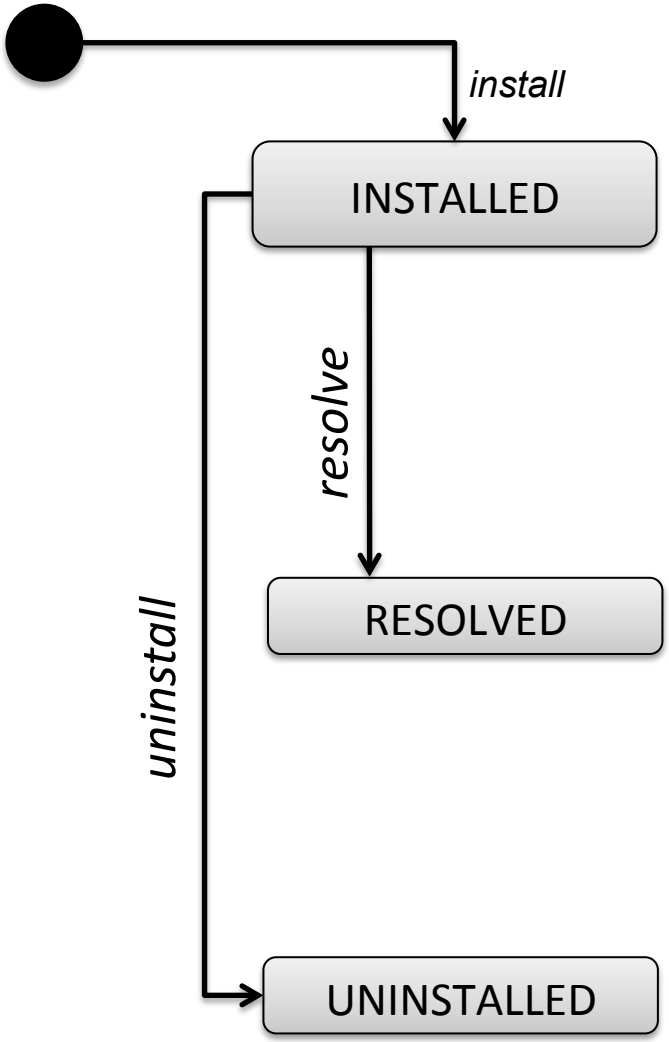
OSGi Basics: Bundle Lifecycle



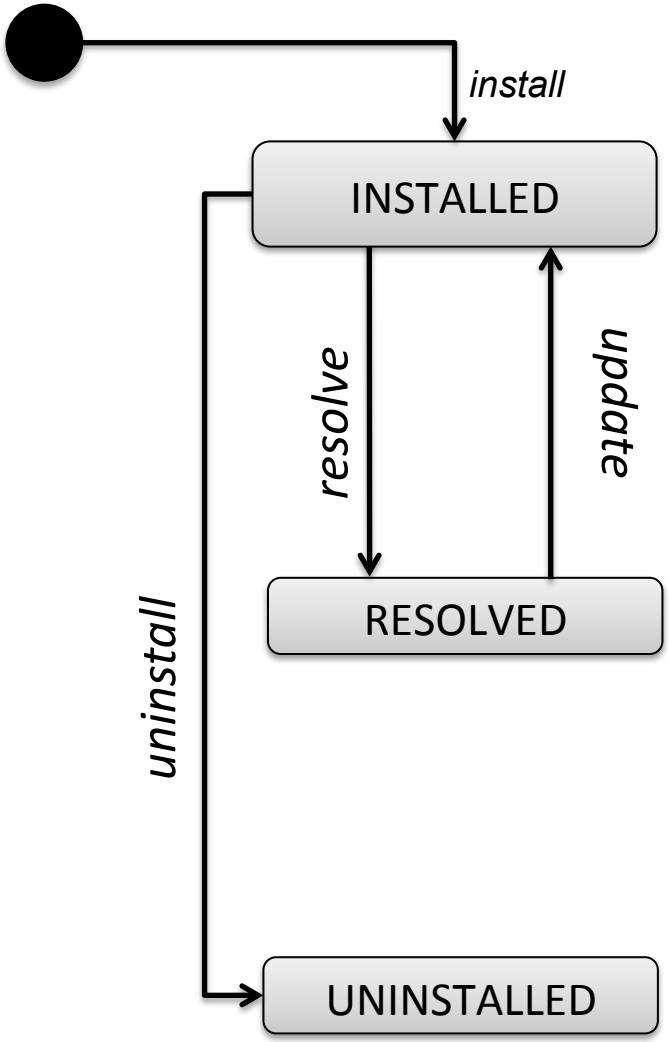
OSGi Basics: Bundle Lifecycle



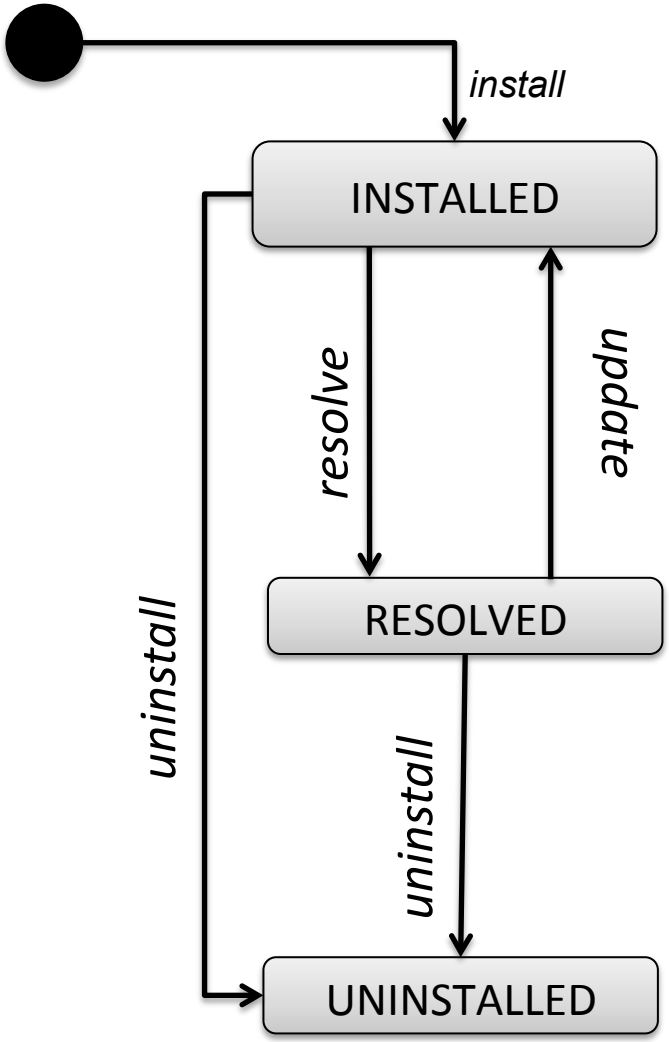
OSGi Basics: Bundle Lifecycle



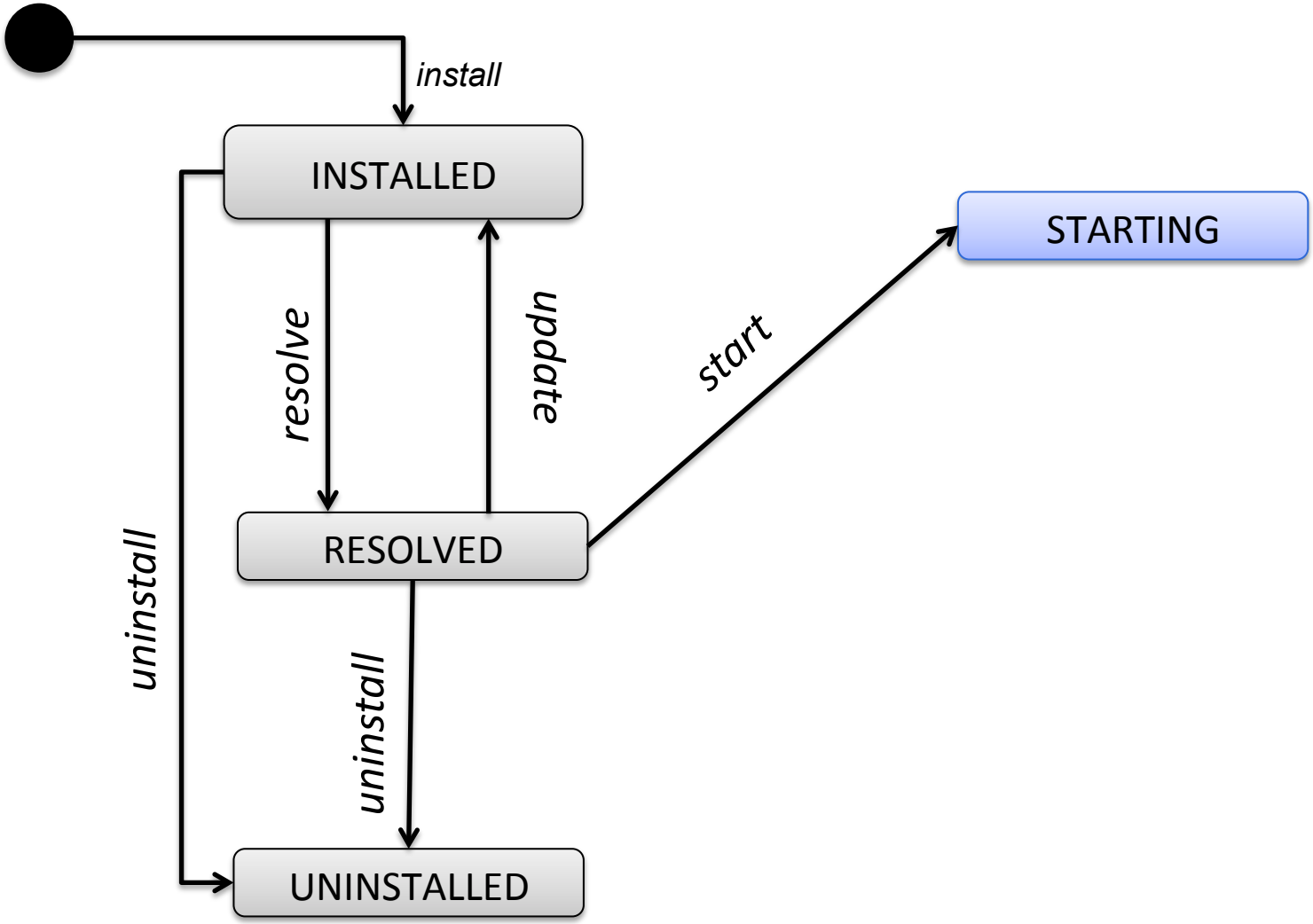
OSGi Basics: Bundle Lifecycle



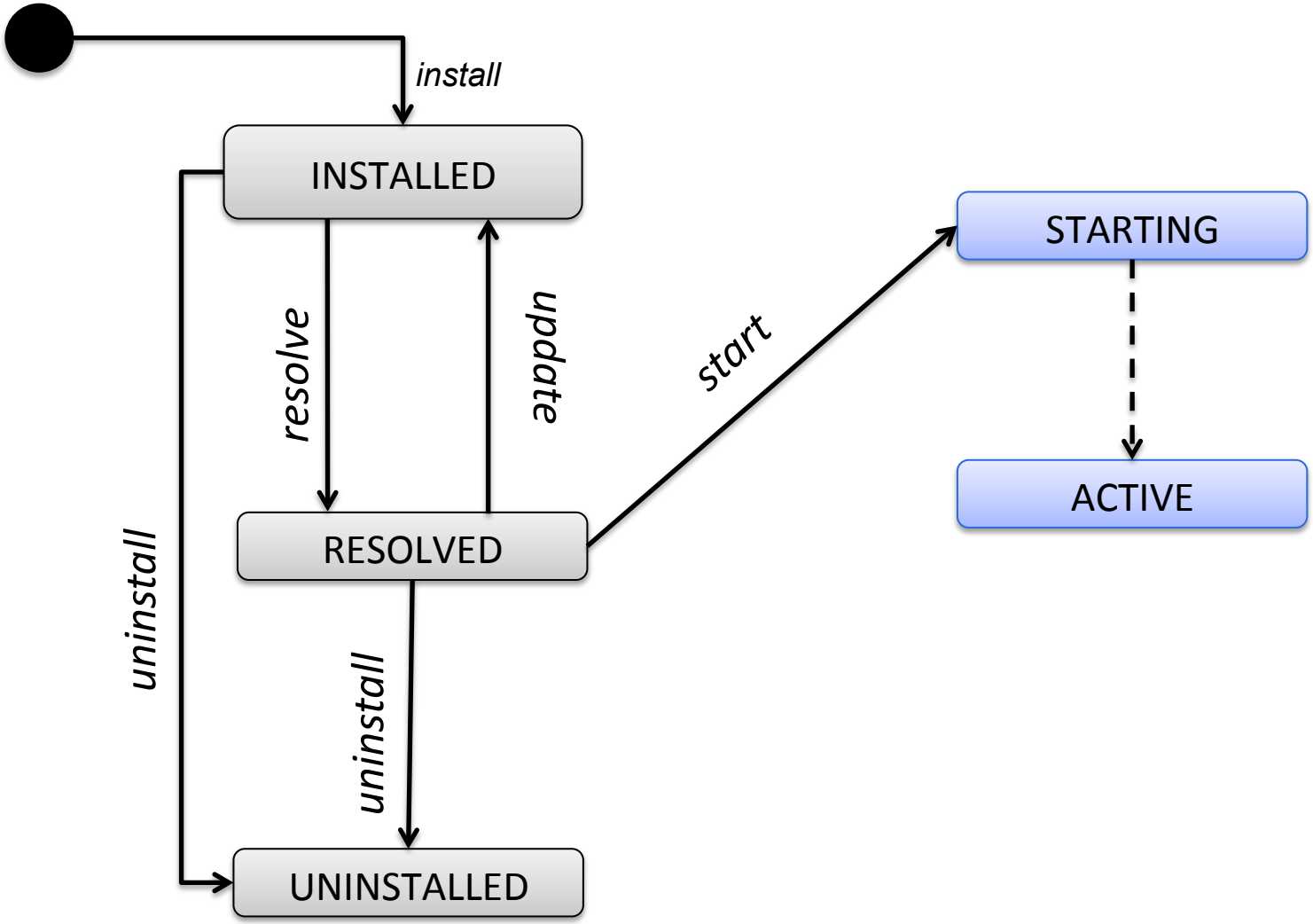
OSGi Basics: Bundle Lifecycle



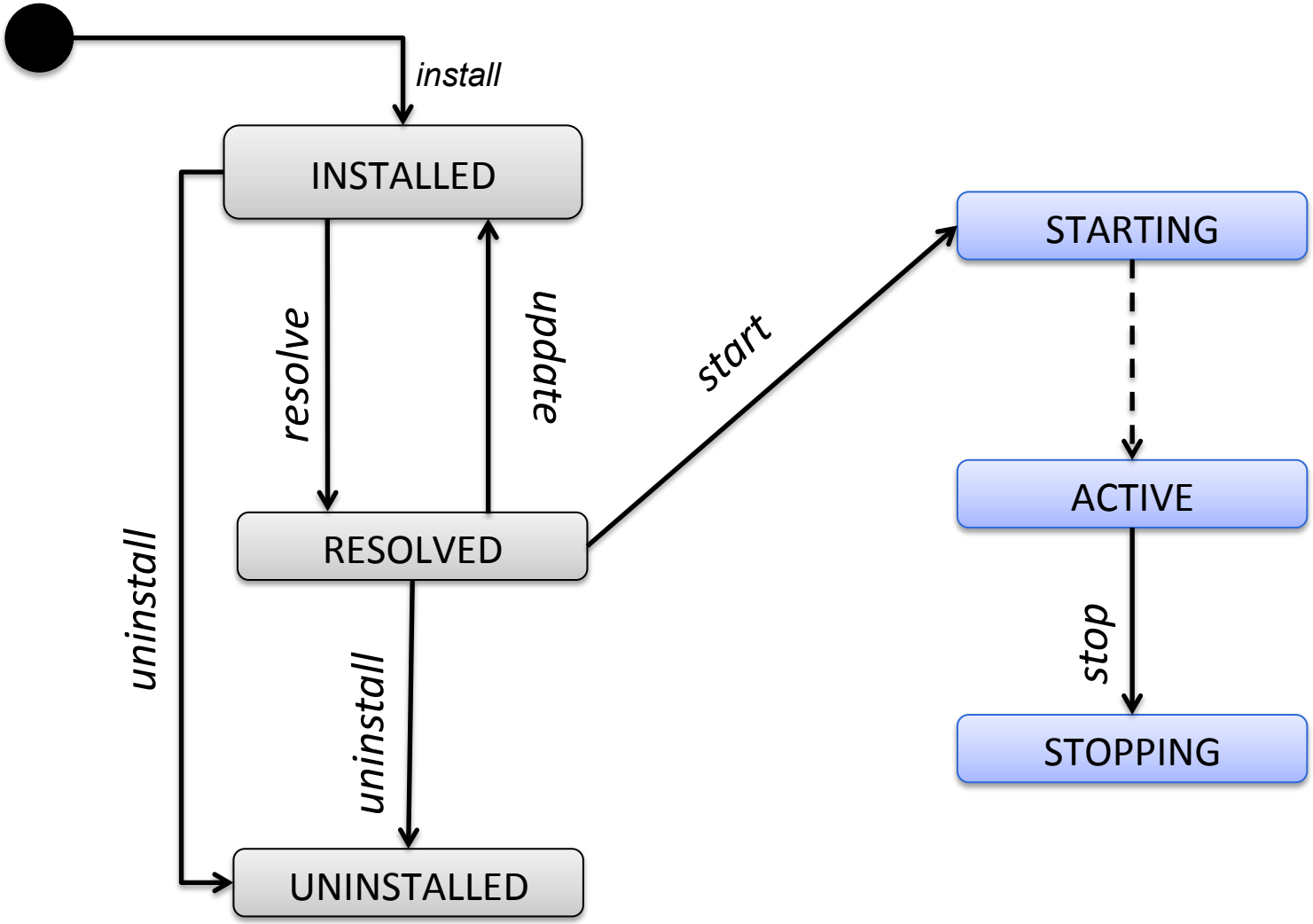
OSGi Basics: Bundle Lifecycle



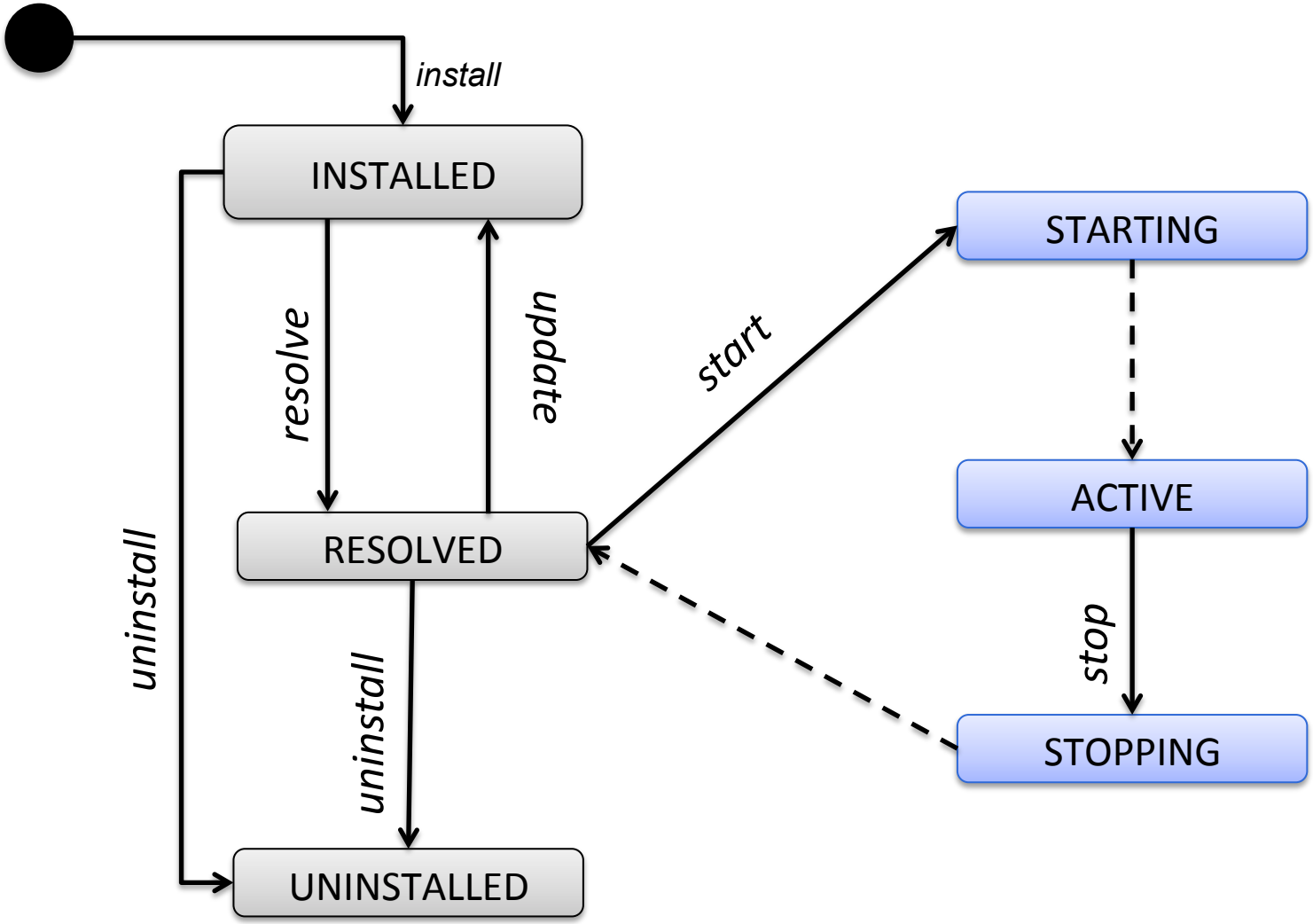
OSGi Basics: Bundle Lifecycle



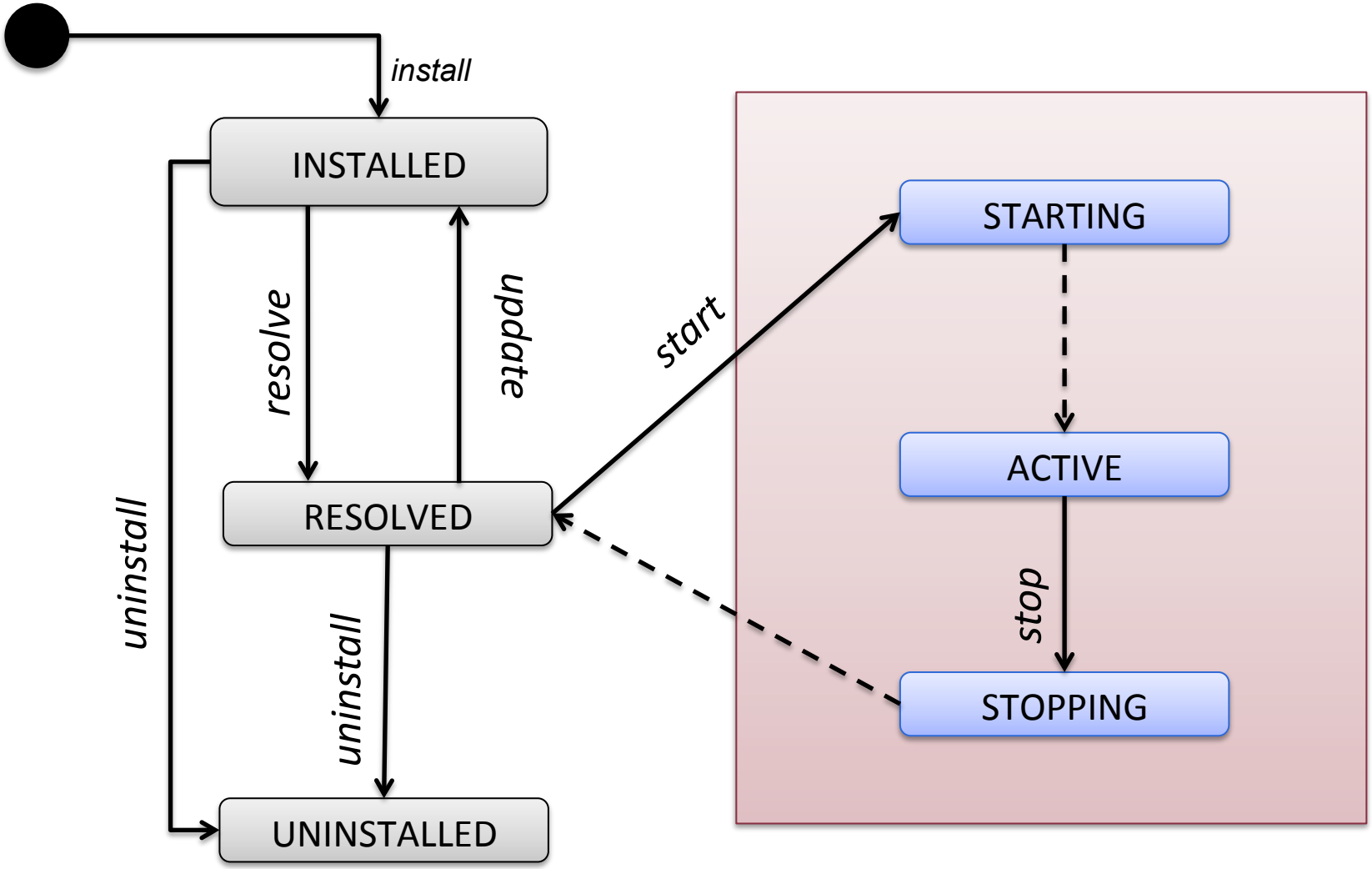
OSGi Basics: Bundle Lifecycle



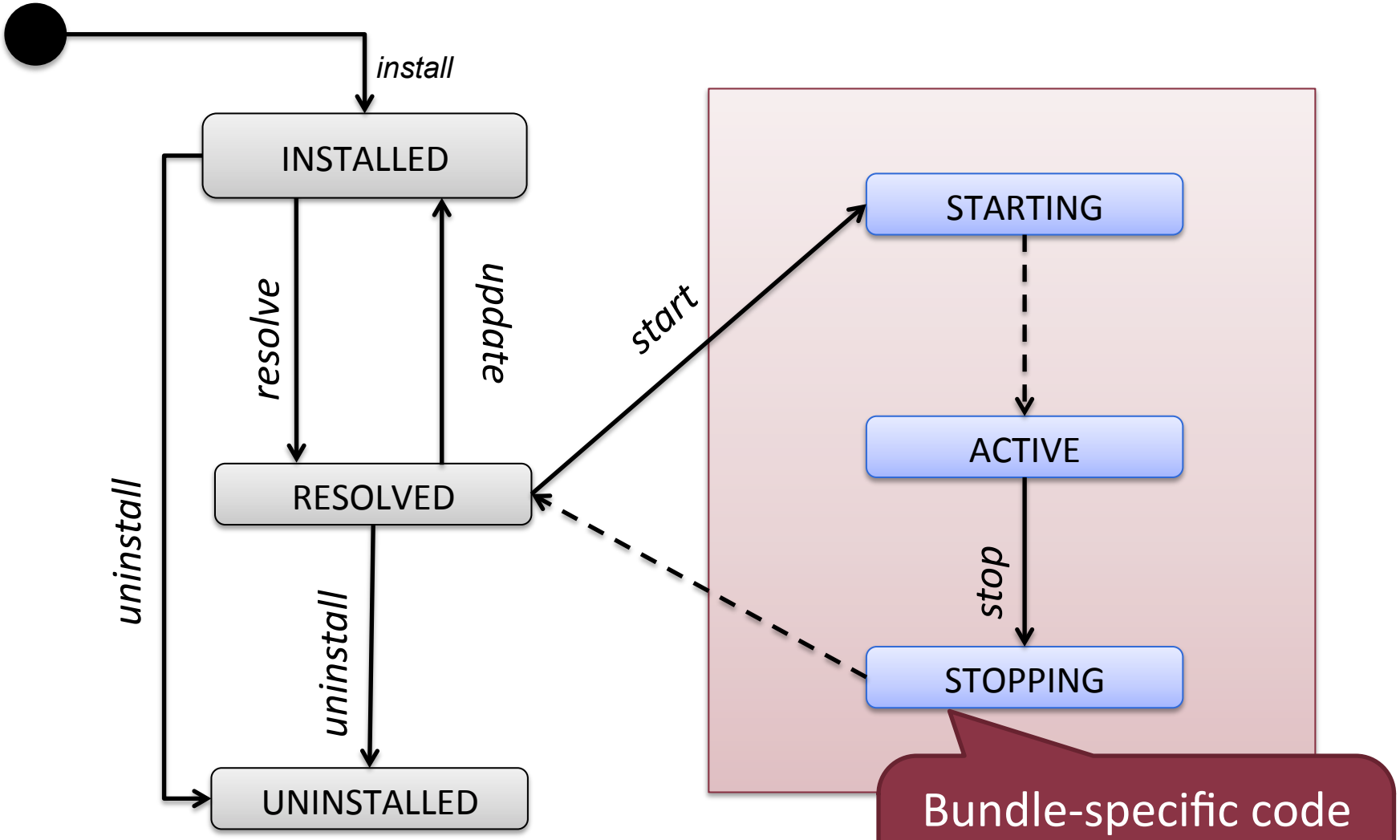
OSGi Basics: Bundle Lifecycle



OSGi Basics: Bundle Lifecycle

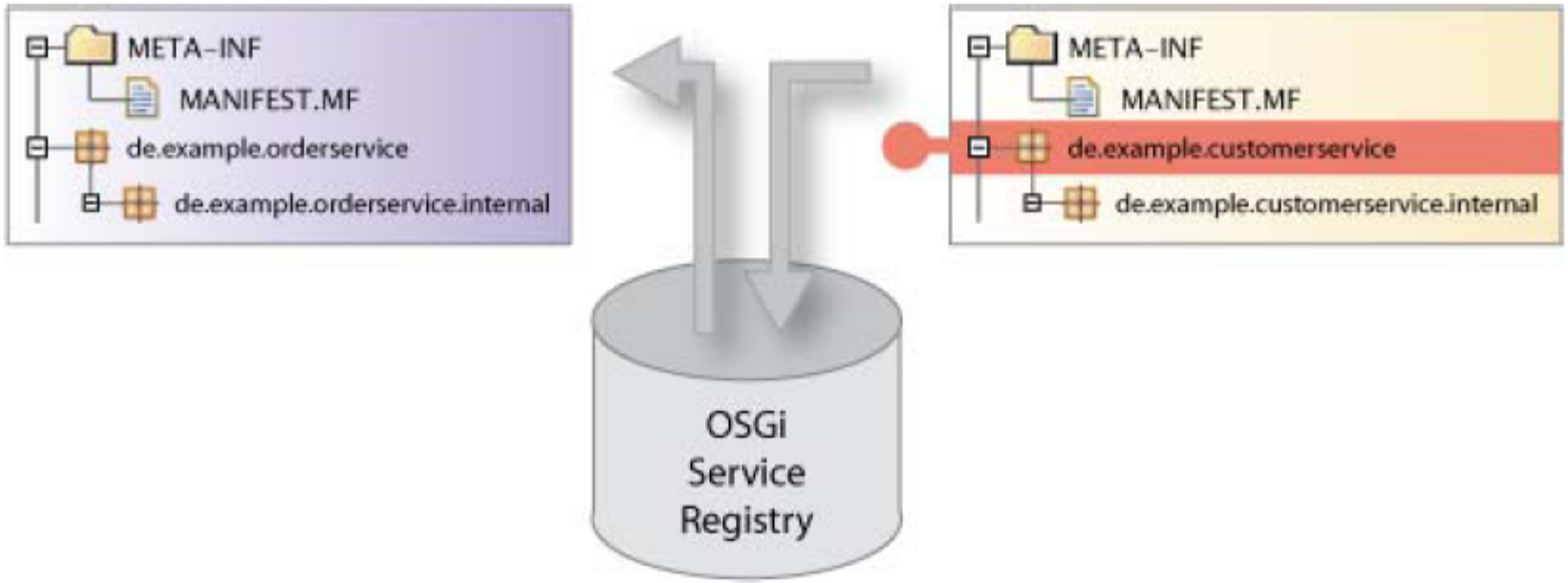


OSGi Basics: Bundle Lifecycle



OSGi Basics: Services

- Bundles can offer services
  - Service registry -> searchable
  - Services start/stop during runtime



OSGi Bundles

Bundles

- Bundles can be started/stopped
  - Remember lifecycle
  - Code can only be referenced if bundle is running
    - On-demand start of bundles
- Running bundles often export services

Modules – Data in manifest.mf

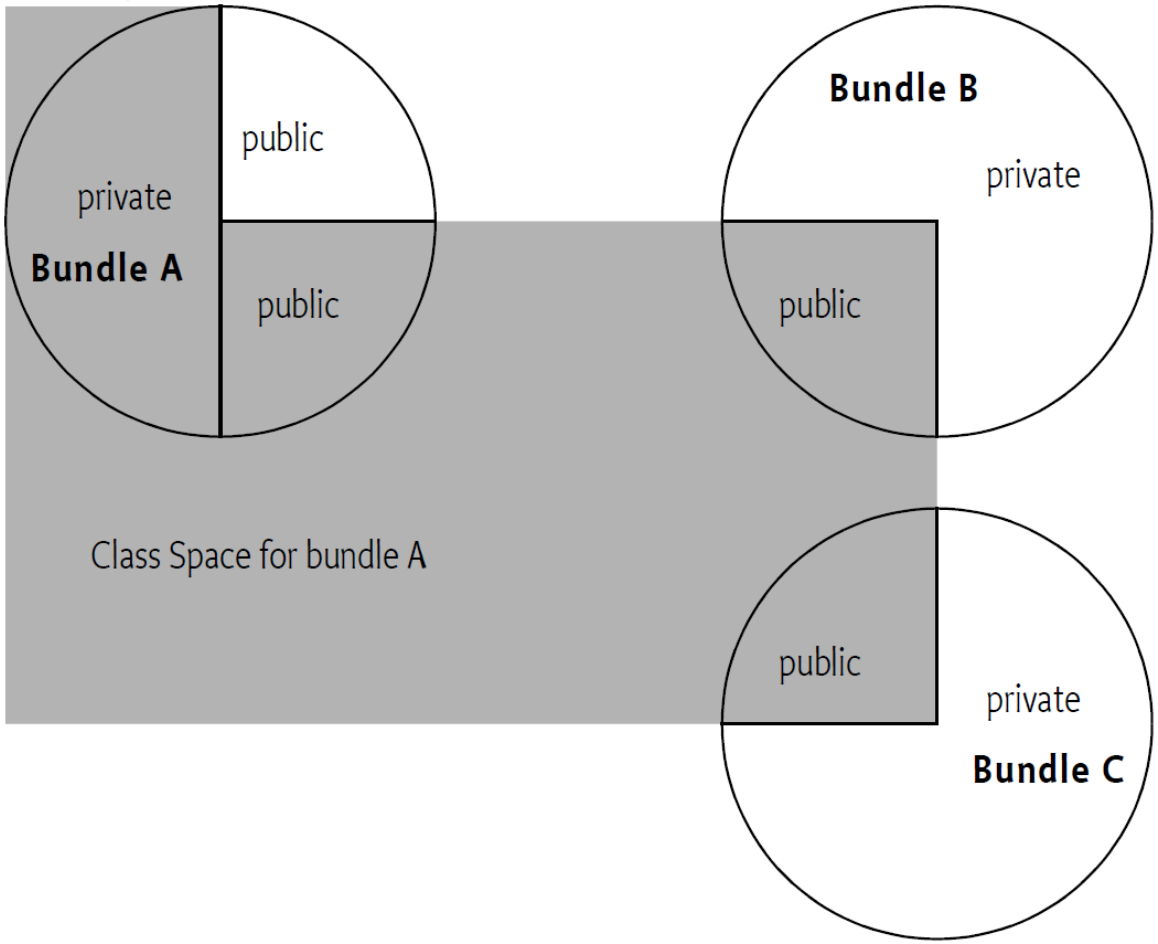
- Bundle-activator
  - Lifecycle callbacks
- Bundle-classpath
  - List of classpath entries. By default: . (bundle root)
- Bundle-name
  - Human-readable name
- Bundle-SymbolicName
  - Bundle identifier
- Bundle-UpdateLocation
  - Update download URL
- Export-Package
  - List of exported packages
- Import-Package
  - List of imported packages
- Require-Bundle
  - List of required bundles

Class loading

- Every bundle is in a single JVM
- Every bundle owns a classloader
  - Three places to load classes from
    1. Boot class path: java.* packages
    2. Framework class path: top level class loader for framework services
    3. Bundle space: bundle-specific class loader

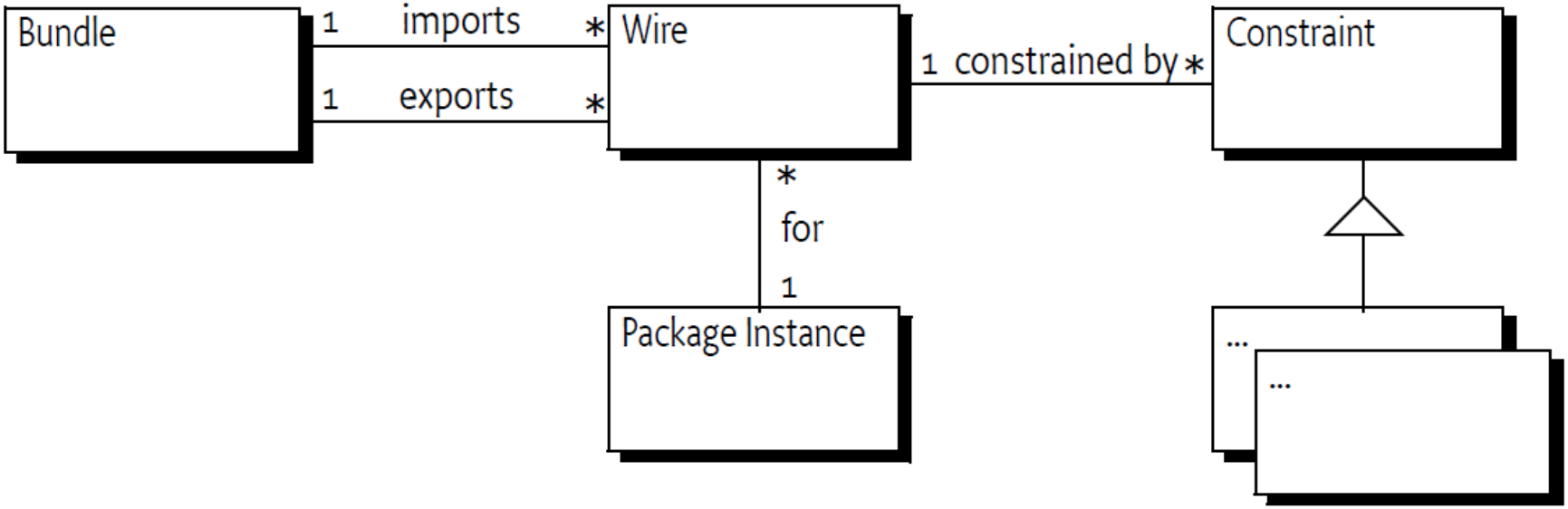
Class loading – Class space

- The class space for a bundle
  - Parent class loader (java.*)
  - Dependencies
    - Imported packages
  - Private classpath
  - Attached fragments



Bundle Resolution

- Resolution: connects import-export declarations
  - Constraints
- Wire: a connection between import and export
  - Valid – fulfills all constraints



Bundle Metadata

- Bundle-SymbolicName
  - „Unique” name
    - Same name and version cannot be loaded multiple times
  - Parameters
    - Singleton: only a single version can be loaded
    - Fragment-attached: fragment attachment settings
      - Always, Never, Resolve-time
  - Példa:

```
Bundle-SymbolicName: com.acme.foo;singleton:=true
```

- Bundle-Version
  - Semantic versioning
  - Same as Eclipse plug-ins
  - Példa:

```
Bundle-Version: 22.3.58.build-345678
```

Metadata Resolution

Metadata Resolution

- Imported-packages
  - List of imported packages
  - Resolution – if required import cannot be resolved, bundle cannot be loaded
  - Version – version interval, e.g. [1.0.0,2.0.0)
  - Bundle-version: export version
  - Bundle-symbolic-name: exporting bundle anem

- Exported-packages
  - List of exported pacakges
  - Similar to Import-Package
  -

Metadata Resolution

- Imported-packages
  - List of imported packages
  - Resolution – if required import cannot be resolved, bundle cannot be loaded
  - Version – version interval, e.g. [1.0.0,2.0.0)
  - Bundle-version: export version
  - Bundle-symbolic-name: exporting bundle anem

```
Import-Package: com.acme.foo;com.acme.bar;
        version="[1.23,1.24]";
        resolution:=mandatory
```

- Exported-packages
  - List of exported pacakges
  - Similar to Import-Package
  -

Metadata Resolution

- Imported-packages
  - List of imported packages
  - Resolution – if required import cannot be resolved, bundle cannot be loaded
  - Version – version interval, e.g. [1.0.0,2.0.0)
  - Bundle-version: export version
  - Bundle-symbolic-name: exporting bundle anem

```
Import-Package: com.acme.foo;com.acme.bar;
        version="[1.23,1.24]";
        resolution:=mandatory
```

- Exported-packages
  - List of exported pacakges
  - Similar to Import-Package
  -

Metadata Resolution

- Imported-packages
  - List of imported packages
  - Resolution – if required import cannot be resolved, bundle cannot be loaded
  - Version – version interval, e.g. [1.0.0,2.0.0)
  - Bundle-version: export version
  - Bundle-symbolic-name: exporting bundle anem

```
Import-Package: com.acme.foo;com.acme.bar;
        version="[1.23,1.24]";
        resolution:=mandatory
```

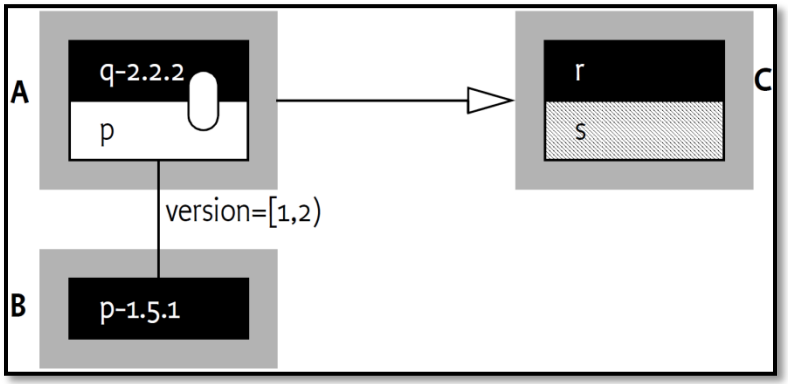
- Exported-packages
  - List of exported pacakges
  - Similar to Import-Package
  -

```
Export-Package: com.acme.foo;com.acme.bar;version=1.23
```

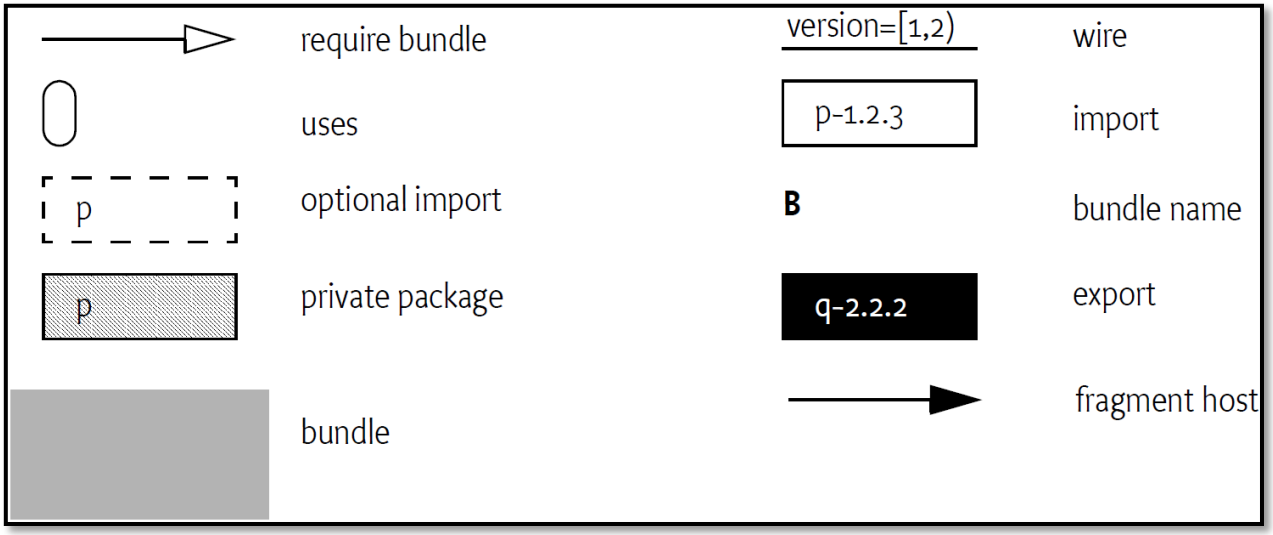
Bundle diagram

```
A: Import-Package: p; version="[1,2)"
   Export-Package: q; version=2.2.2; uses:=p
   Require-Bundle: C
B: Export-Package: p; version=1.5.1
C: Export-Package: r
```

Textual form



Graphical form



Legend

Capabilities (Since v4.3)

- Capability
  - Generalization of dependency relations
  - Property with textual description
  - Versionable
- Require-Capability
  - Setting up dependency
- Provide-Capability
  - Declaring capability

OSGi Services

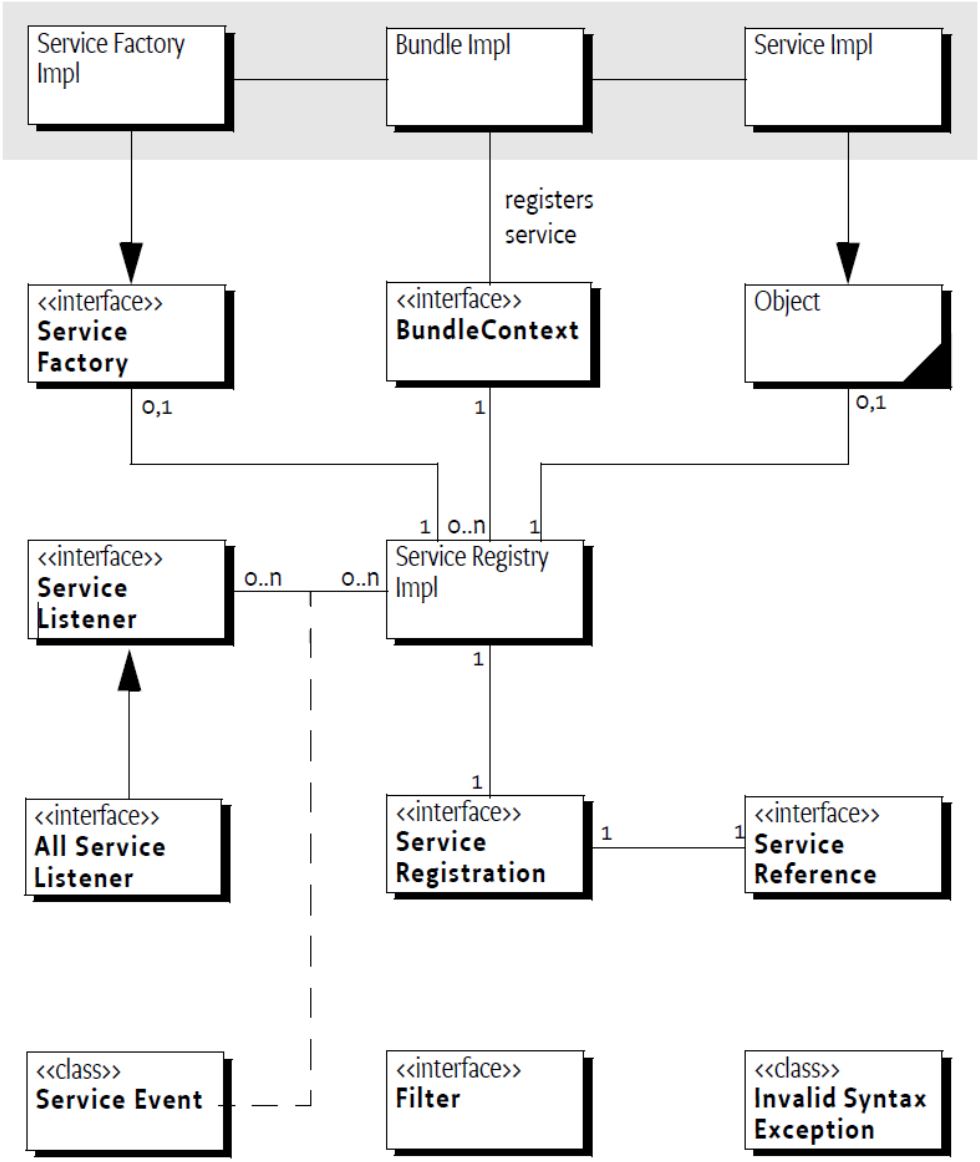
Service Layer

- Specifying cooperation mode
  - „Publish, find, and bind”
  - Service
    - Plain old Java object (POJO)
    - Registered using (one or more) interfaces
- Bundles
  - Register services
  - Look up services
  - Use services
  - Handle related events

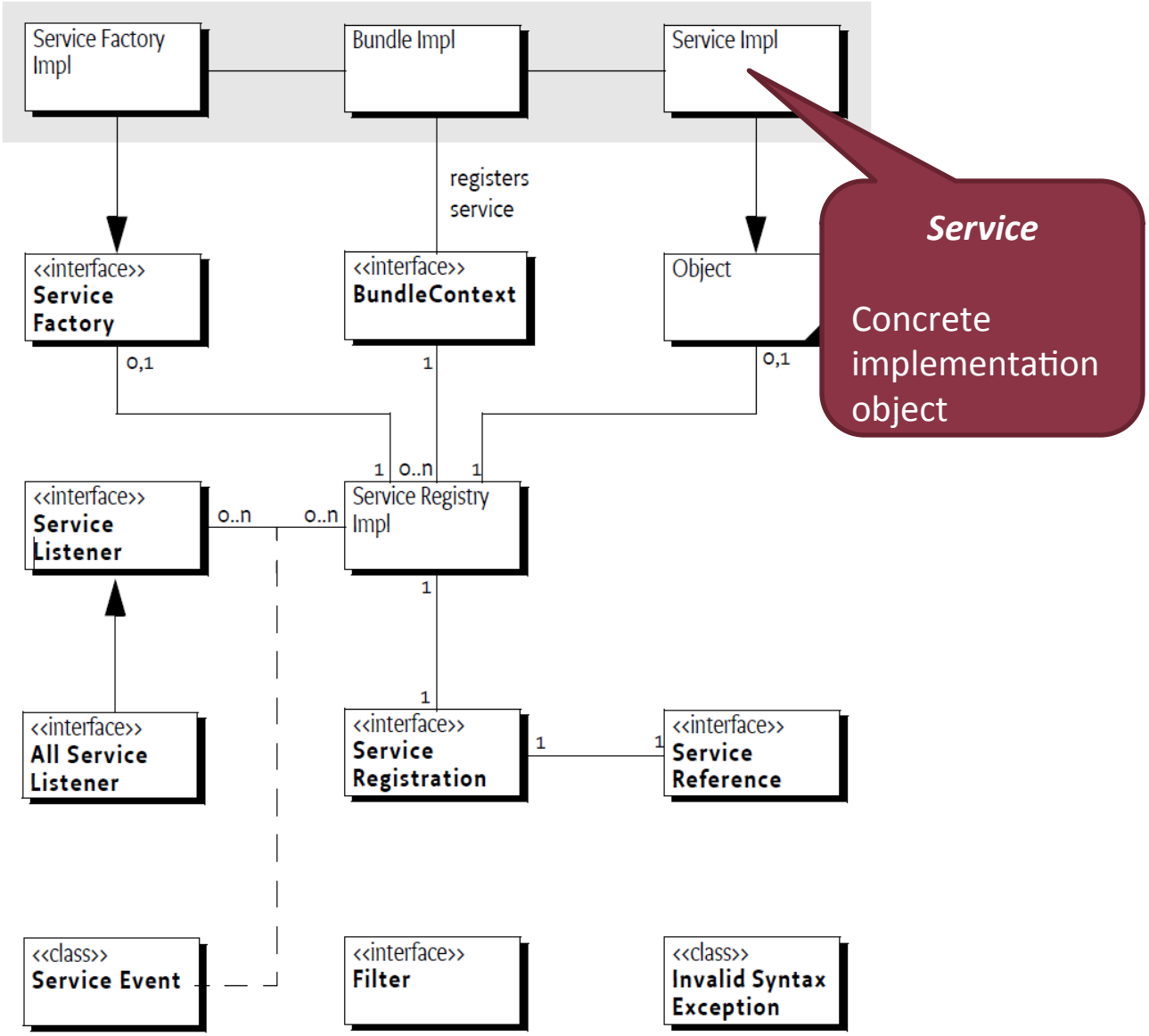
Service Layer

- Dynamic services:
  - New services appear
  - Old services disappear
- Versioning:
  - Services can be updated
- Reflective access:
  - Every property can be accessed generically
- Perzisztent id:
  - Service tracking between framework restarts

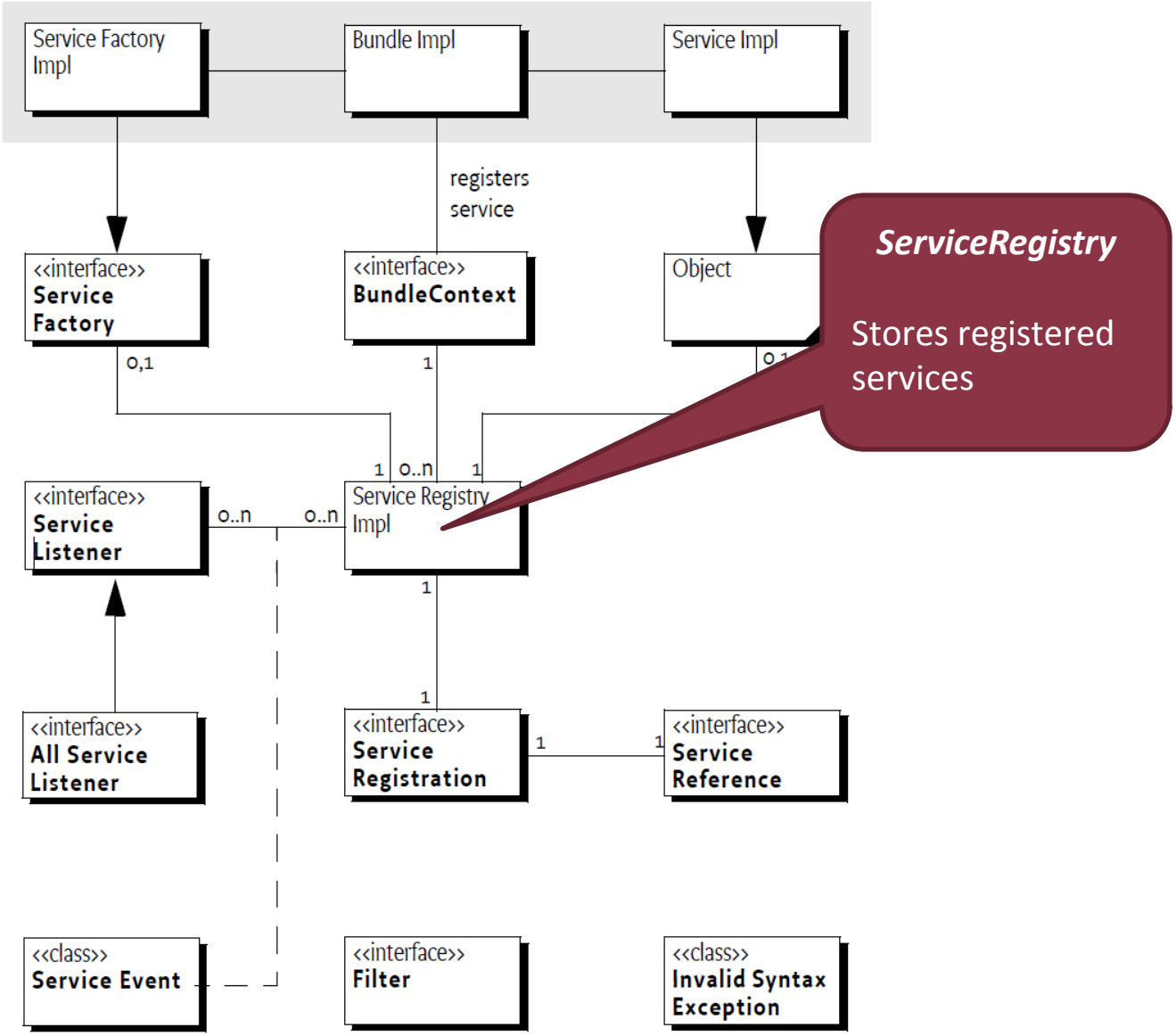
Service Layer



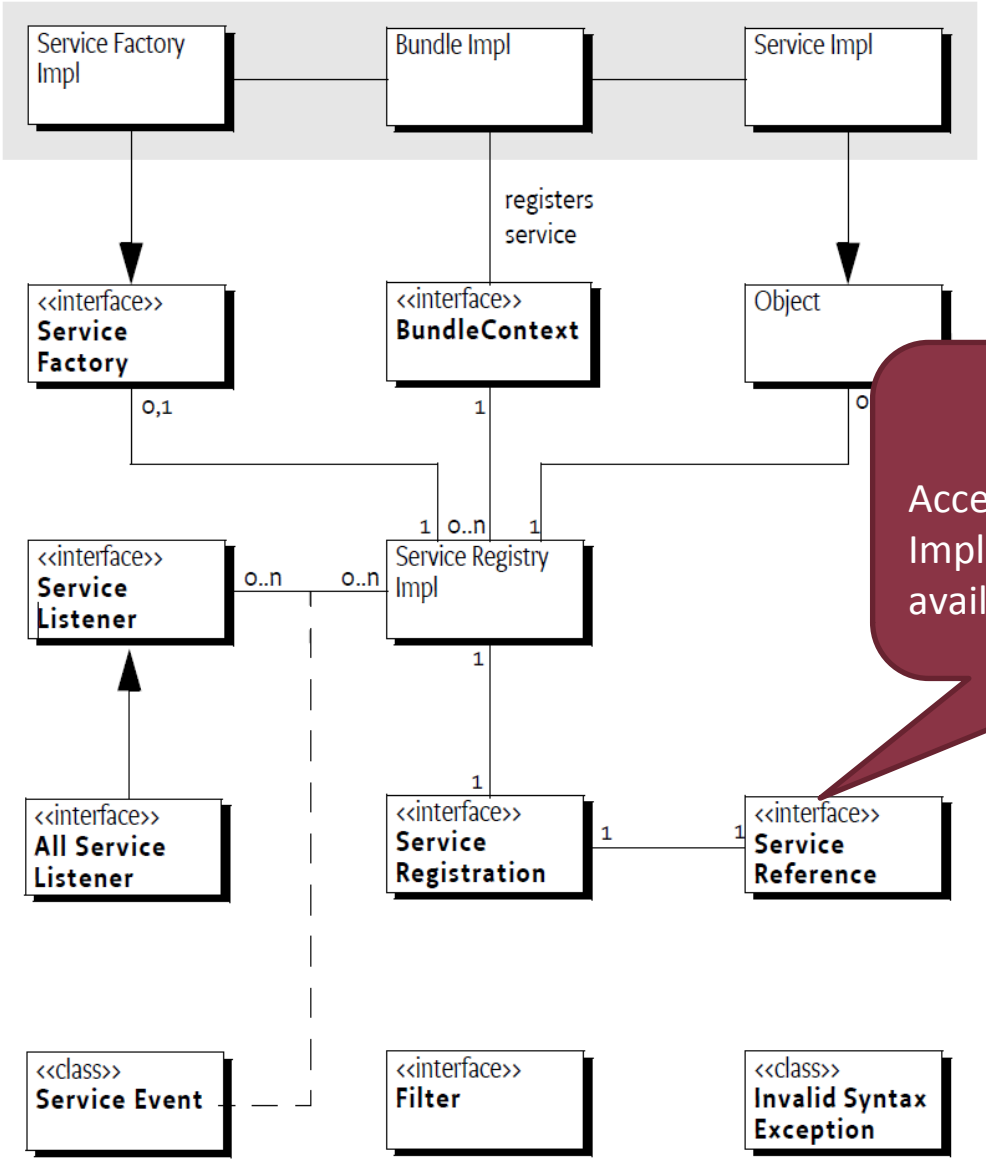
Service Layer



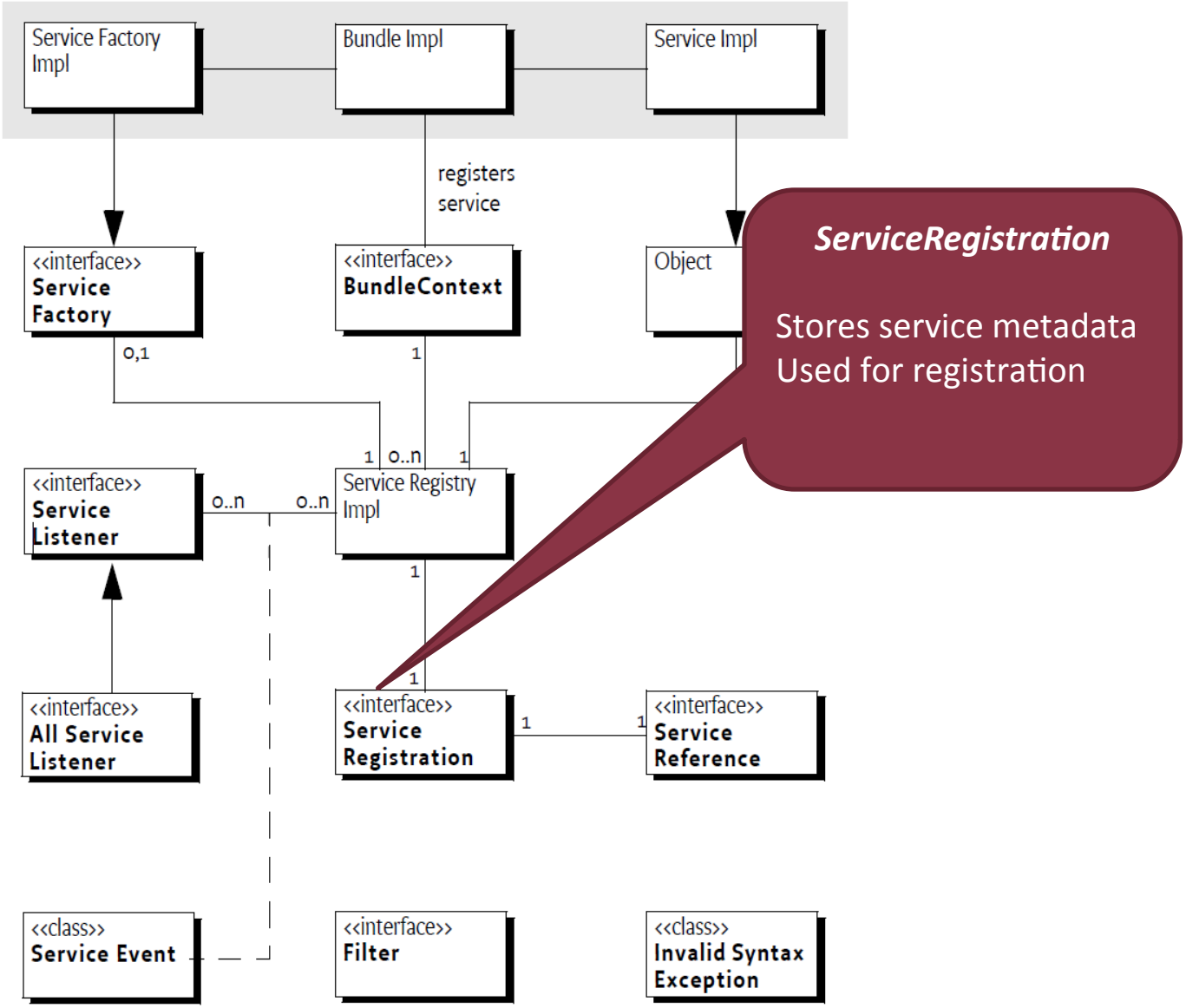
Service Layer



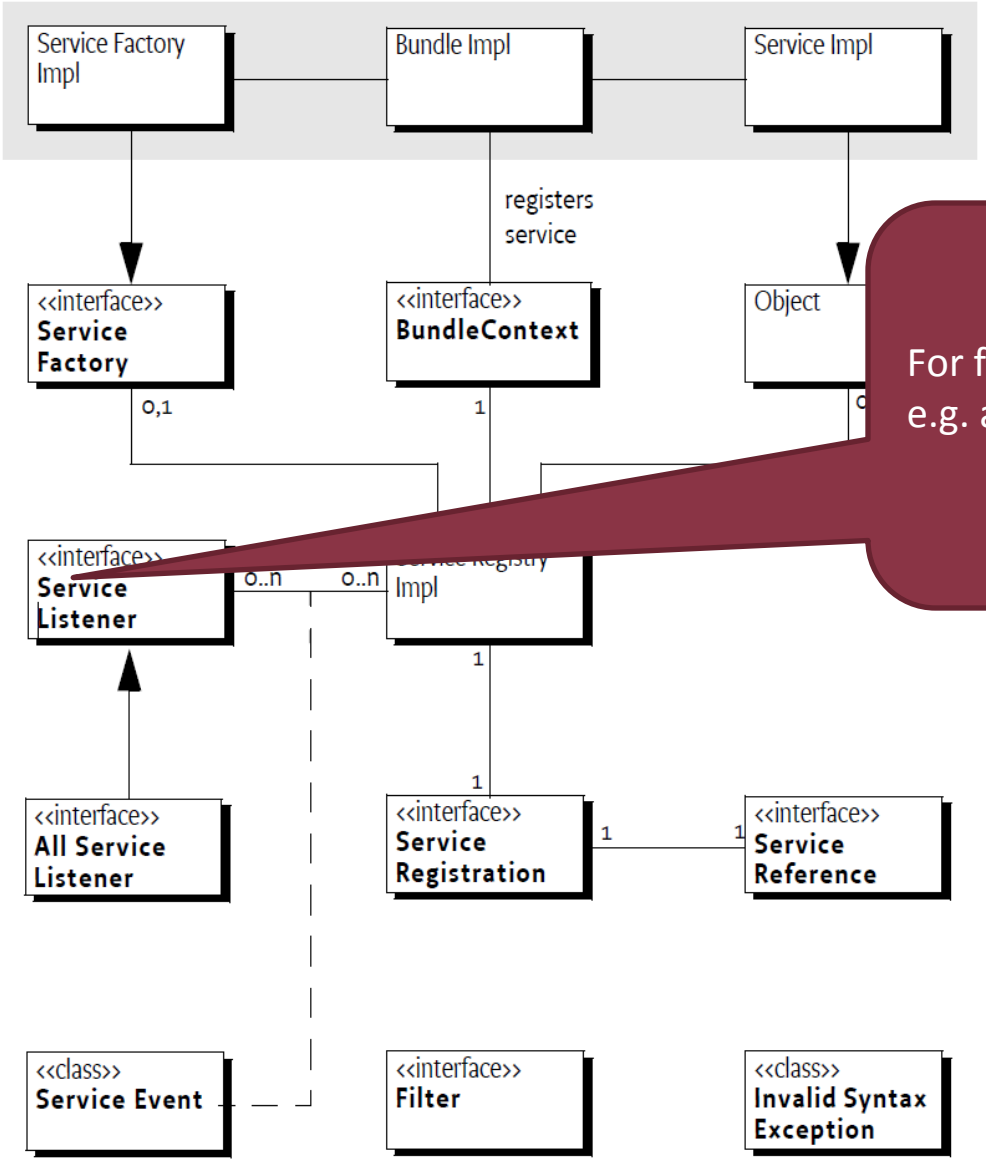
Service Layer



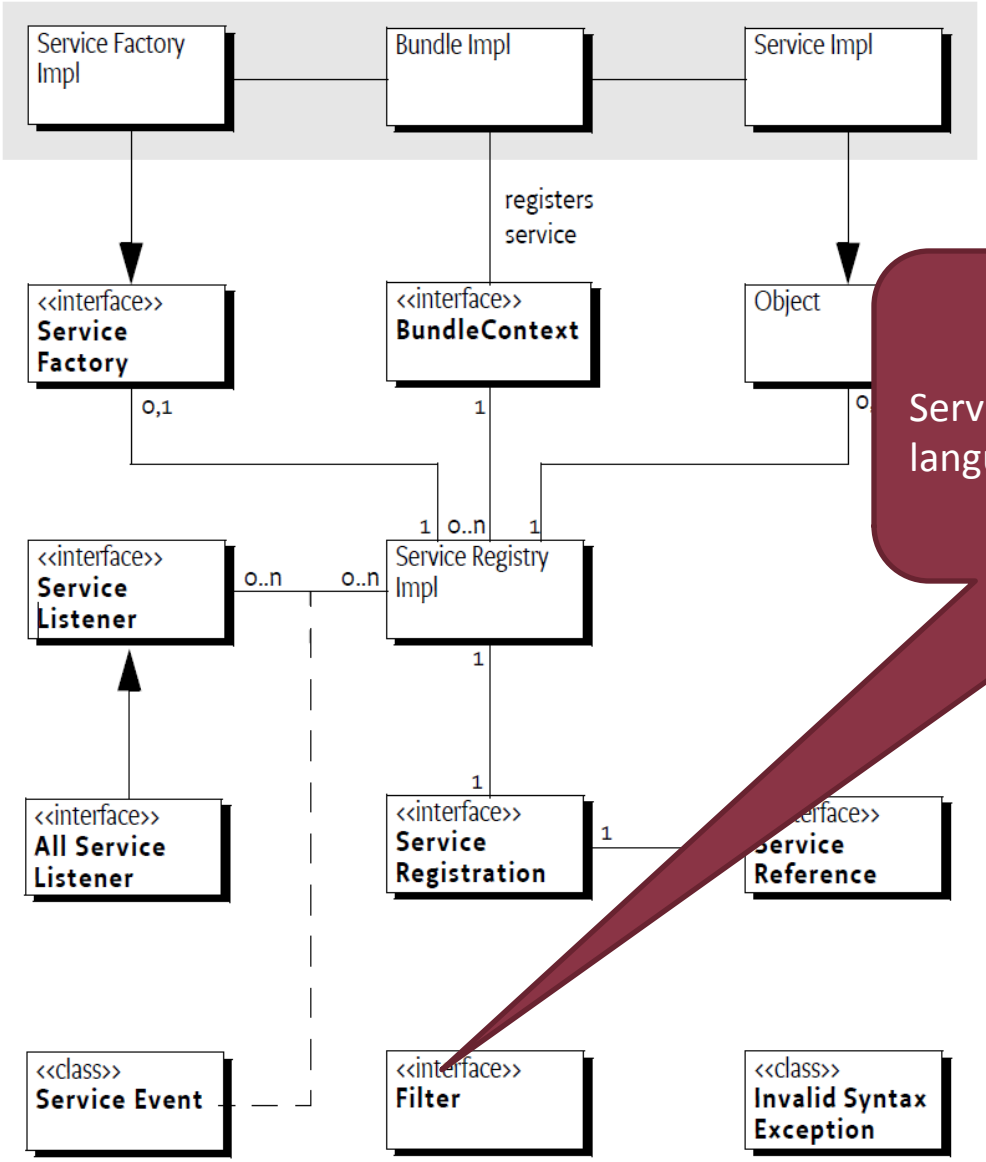
Service Layer



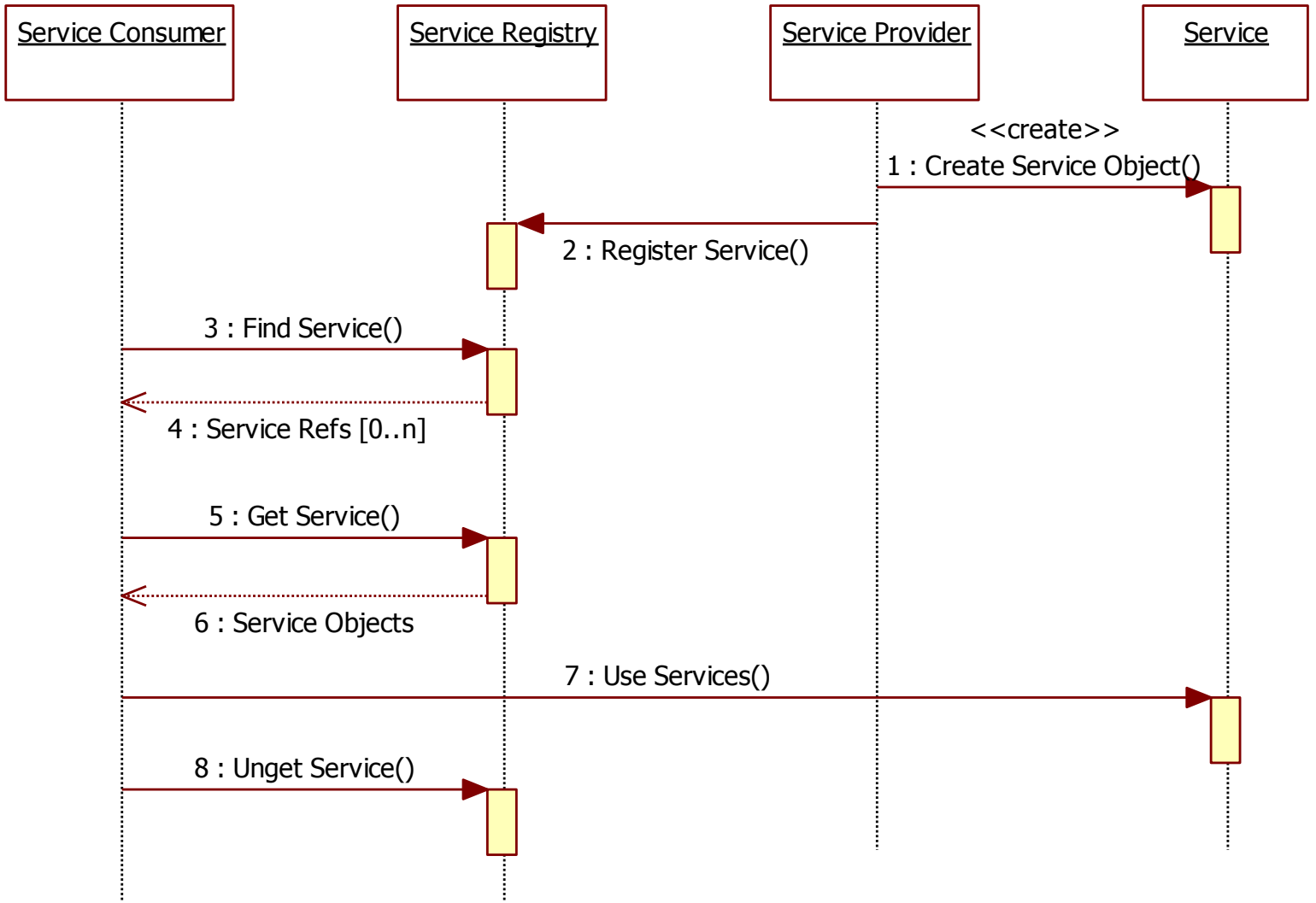
Service Layer



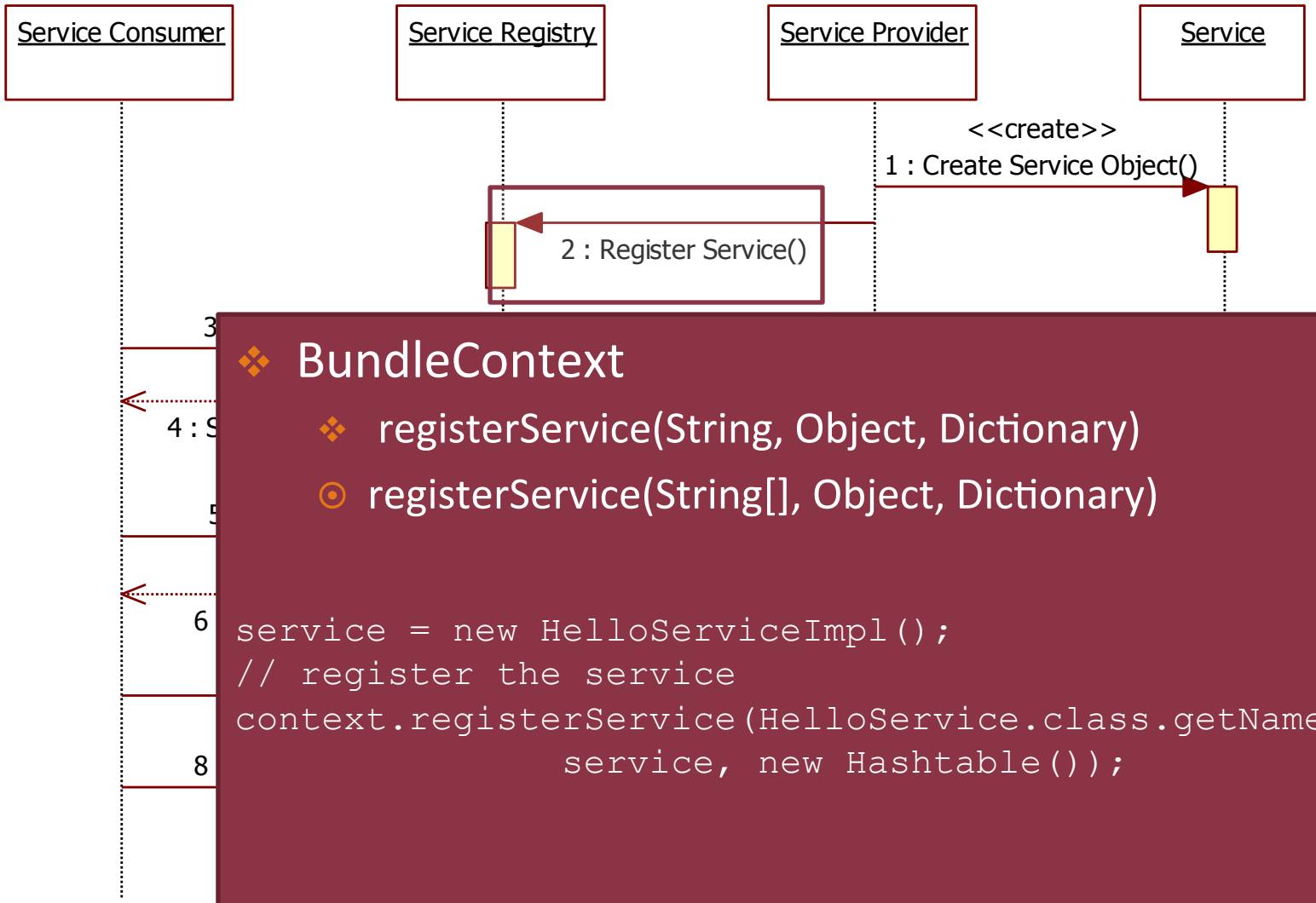
Service Layer



Service Access



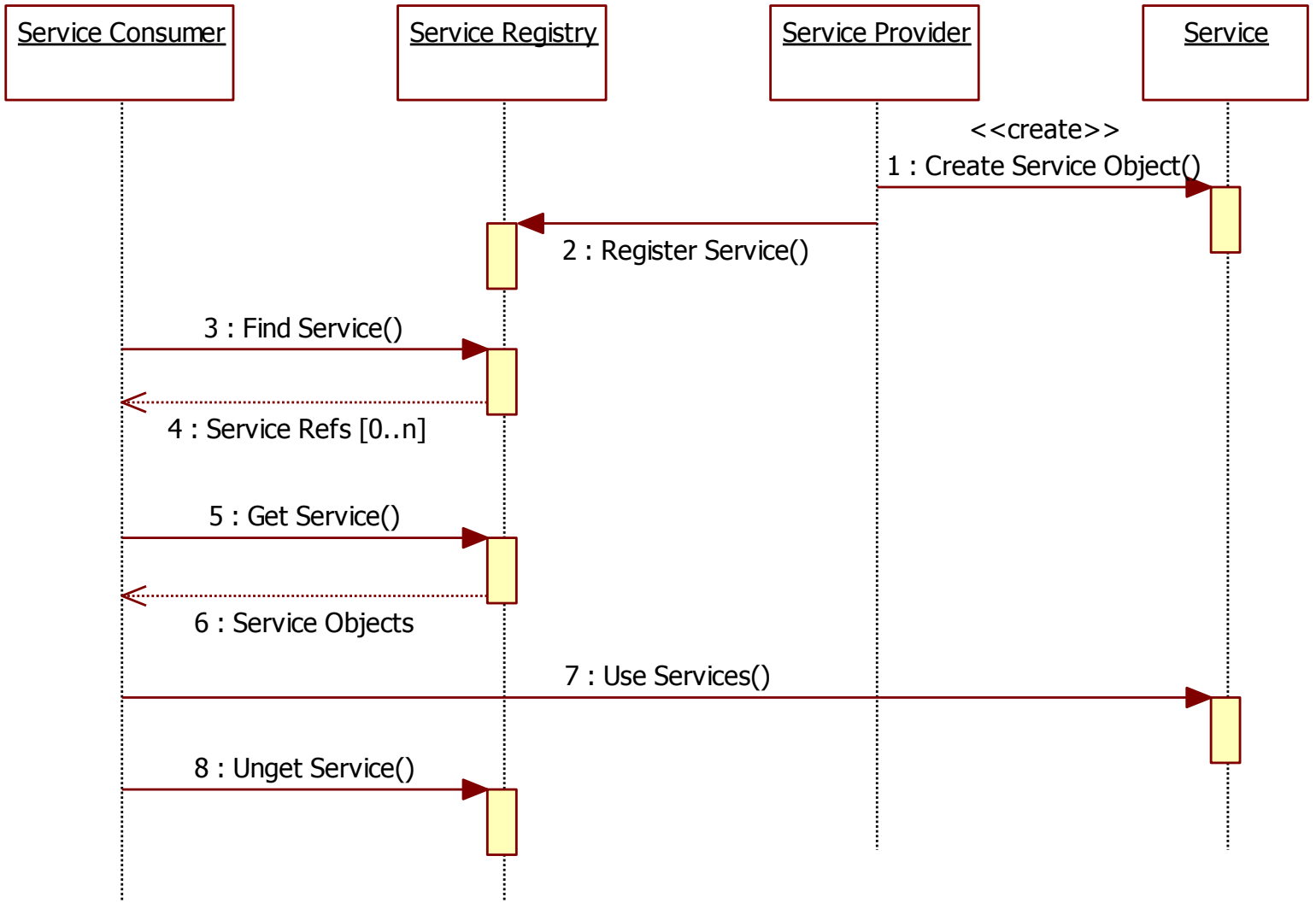
Service Access



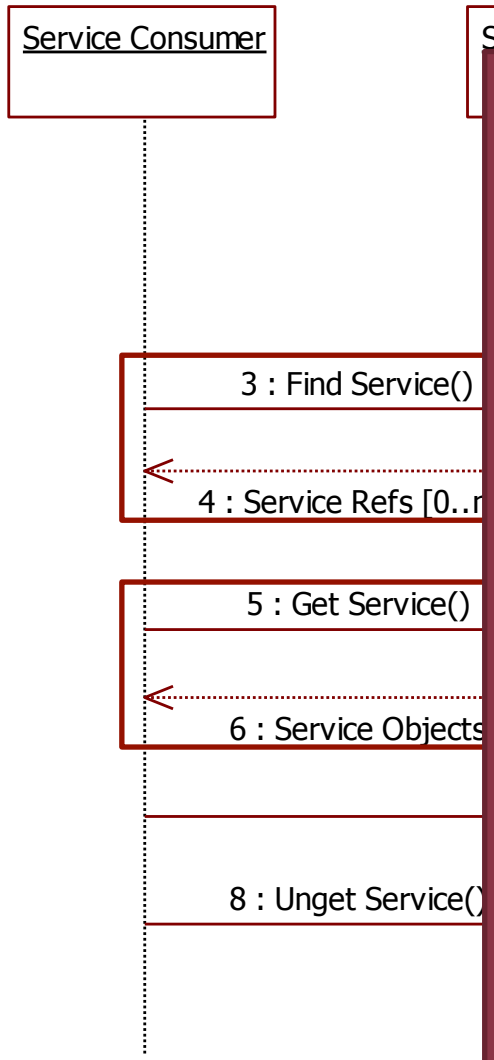
- BundleContext
  - registerService(String, Object, Dictionary)
  - registerService(String[], Object, Dictionary)

```
service = new HelloServiceImpl();
// register the service
context.registerService(HelloService.class.getName(),
                        service, new Hashtable());
```

Service Access



Service Access



- BundleContext
  - getServiceReference(String)
    - Use Ranking for selection
  - getServiceReferences(String,String)
    - All references
  - getService(ServiceReference)
    - Returns service object
  - ungetService(ServiceReference)

```
ServiceReference reference =
context.getServiceReference(
  HelloService.class.getName());
HelloService service = (HelloService)
  context.getService(reference);
```

Direct Service Access - Problems

- No notification for service disappearance
  - For each use ask for the service instance
  - Service listener: easy to make mistakes
- Low-level API
  - Cumbersome to use
  - Large amount of Java code needed

Service Tracker

- ServiceTracker instance
  - Register one for a service interface
  - Notifies if an instance of a tracked interface
    - Appears
    - Is removed
    - Is modified

OSGi compendium

OSGi compendium

- Additional features for OSGi frameworks
- Important additional services
  - Declarative Services:
    - High-level service definitions
  - Http Service
    - HTTP related services
  - Remote Services
    - Cooperation between frameworks
  - Log Service
    - Generic log service
  - …

Declarative Services

- In Compendium services since R4
- Based on XML component descriptors
  - Declarative „Publish, find, and bind”
  - Splitting responsibilities
    - Bundle only implements services
    - Service Component Runtime (SCR) manages registration
  - Components also dynamic
  - „On demand” loading
    - As Eclipse Extensions

DS – Component descriptor

- XML based
- Declarative
  - Service declarations
  - Service access
- Multiple components in a single bundle
  - Enumerate them in MANIFEST.MF
    - Service-Component keyword

Component Descriptor Example

```
<scr:component xmlns:scr="http://www.osgi.org/
  xmlns/scr/v1.1.0"
   name="sample.component">
   <implementation
  class="org.sample.HelloServiceImpl"/>
   <service>
     <provide interface="org.sample.HelloService"/>
   </service>
   <reference
      bind="setService"
      unbind="unsetService"
      cardinality="0..1"
      interface="org.sample.ServiceForHello"
  name=„SERVICEFORHELLO"
   policy="dynamic"/>
</scr:component>
```

Component Descriptor Example

```
<scr:component xmlns:scr="http://www.osgi.org/
  xmlns/scr/v1.1.0"
   name="sample.component">
   <implementation
  class="org.sample.HelloServiceImpl"/>
   <service>
     <provide interface="org.sample.HelloService"/>
   </service>
   <reference
      bind="setService"
      unbind="unsetService"
      cardinality="0..1"
      interface="org.sample.ServiceForHello"
  name=„SERVICEFORHELLO"
   policy="dynamic"/>
</scr:component>
```

Provided service declarations

Component Descriptor Example

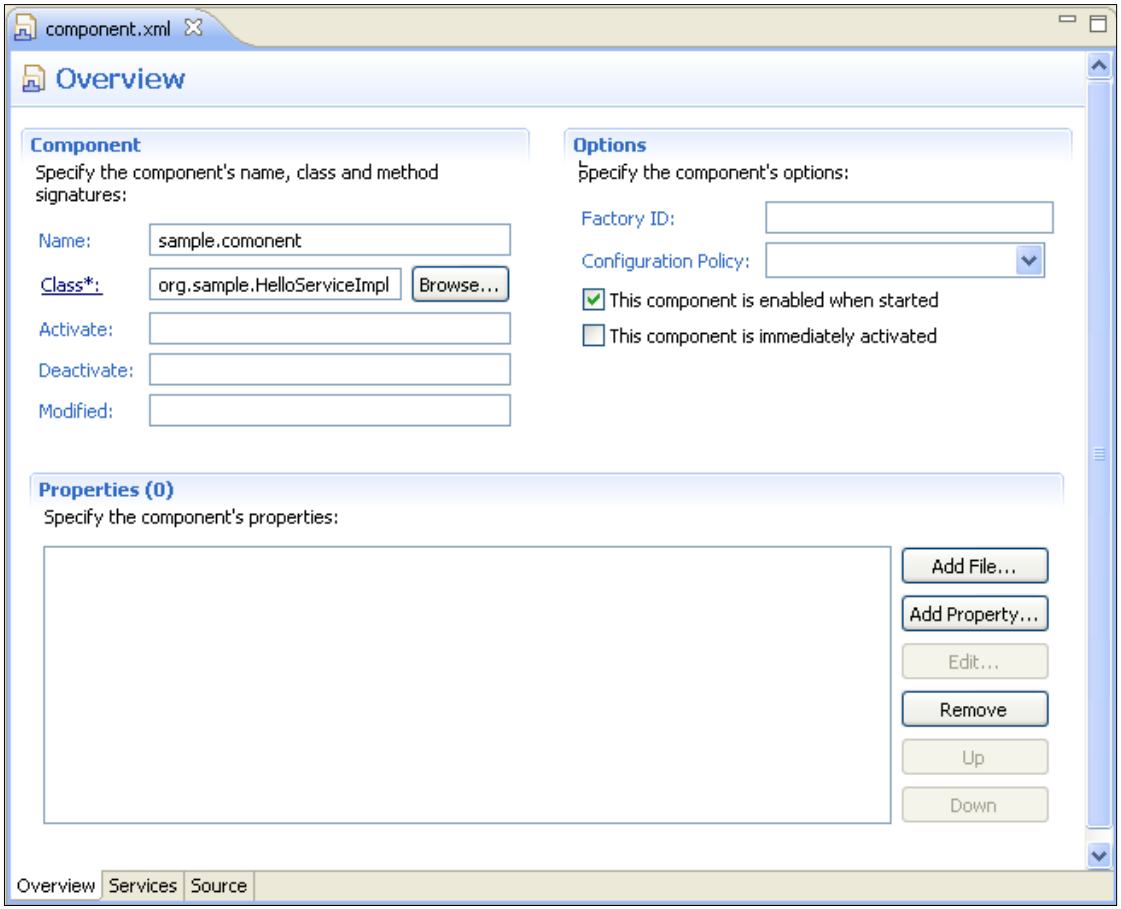
```
<scr:component xmlns:scr="http://www.osgi.org/
   xmlns/scr/v1.1.0"
    name="sample.component">
    <implementation
   class="org.sample.HelloServiceImpl"/>
    <service>
      <provide interface="org.sample.HelloService"/>
    </service>
    <reference
       bind="setService"
       unbind="unsetService"
       cardinality="0..1"
       interface="org.sample.ServiceForHello"
   name=„SERVICEFORHELLO"
    policy="dynamic"/>
</scr:component>
```

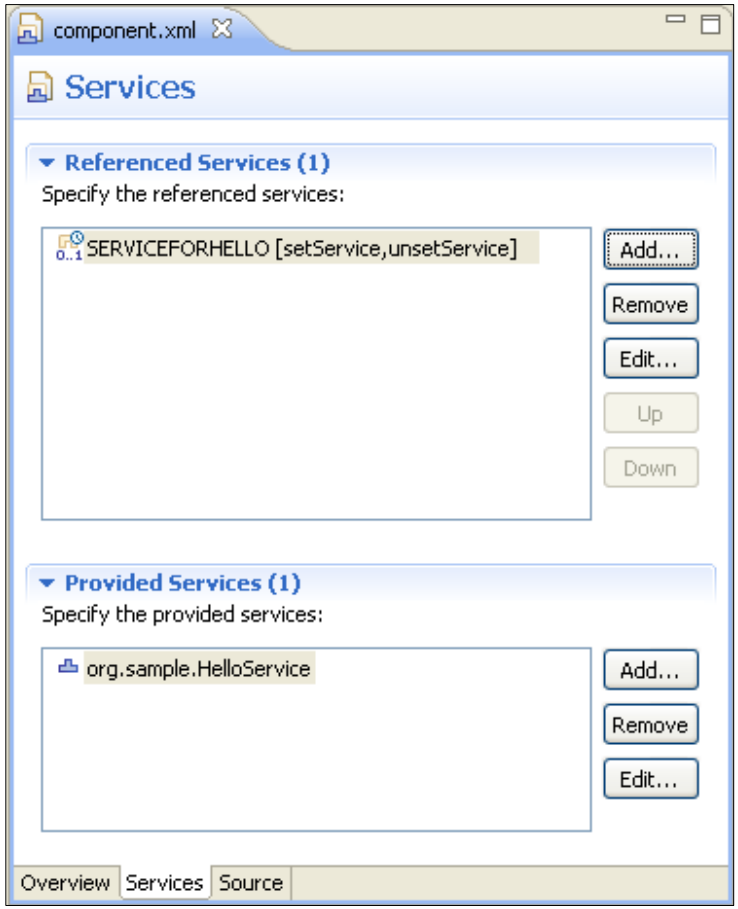
Provided service declarations

Referenced services

DS – Component Descriptor

- Eclipse Support: Declarative Service Tooling
  - Component Definition Editor





DS – Referring Services

- Cardinality:
  - 0..1 → optional, single service can be handled
  - 1..1 → mandatory, single service can be handled
  - 1..n → mandatory, multiple services handled
  - 0..n → optional, multiple services handled
- Policy:
  - dynamic: service may be re-created by SCR
  - static: no service replacement by SCR (stateful services)

DS – Referring Services

- Cardinality:
  - 0..1 → optional, single service can be handled
  - 1..1 → mandatory, single service can be handled
  - 1..n → mandatory, multiple services handled
  - 0..n → optional, multiple services handled
- Policy:
  - dynamic: service may be re-created by SCR
  - static: no service replacement by SCR (stateful services)

```
<reference
      bind="setService"
      unbind="unsetService"
      cardinality="0..1"
      interface="org.sample.ServiceForHello" name=„SERVICEFORHELLO"
   policy="dynamic"/>
```

DS – Referring Services

- Cardinality:
  - 0..1 → optional, single service can be handled
  - 1..1 → mandatory, single service can be handled
  - 1..n → mandatory, multiple services handled
  - 0..n → optional, multiple services handled
- Policy:
  - dynamic: service may be re-cre
  - static: no service replace ent by SCR (stateful services)

Method to call when service appears

```
<reference
      bind="setService"
      unbind="unsetService"
      cardinality="0..1"
      interface="org.sample.ServiceForHello" name=„SERVICEFORHELLO"
   policy="dynamic"/>
```

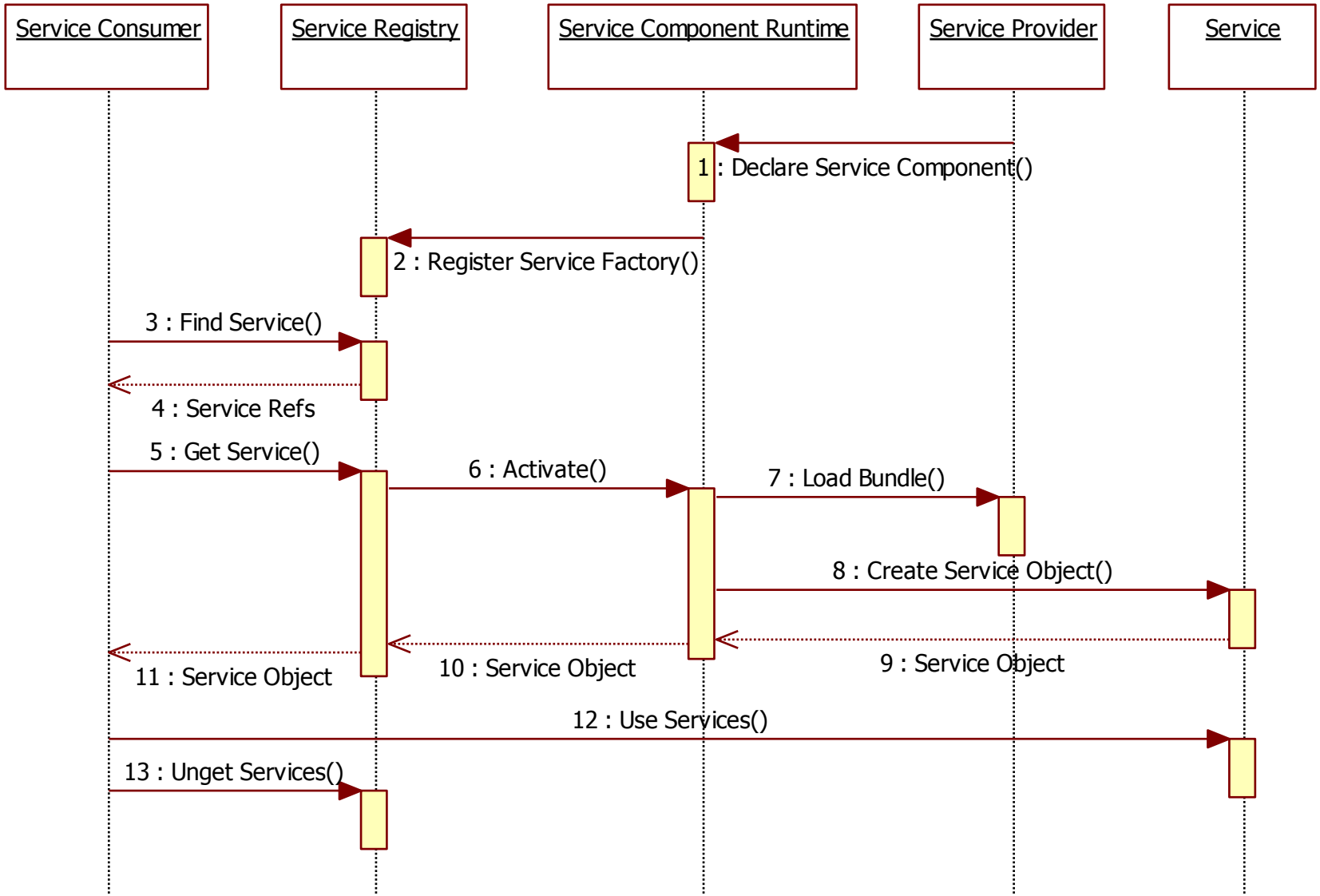
DS – Referring Services

- Cardinality:
  - 0..1 → optional, single service can be handled
  - 1..1 → mandatory, single service can be handled
  - 1..n → mandatory, multiple services handled
  - 0..n → optional, multiple services handled
- Policy:
  - dynamic: service may be re-created by SCR
  - static: no service replacement by SCR (stateful services)

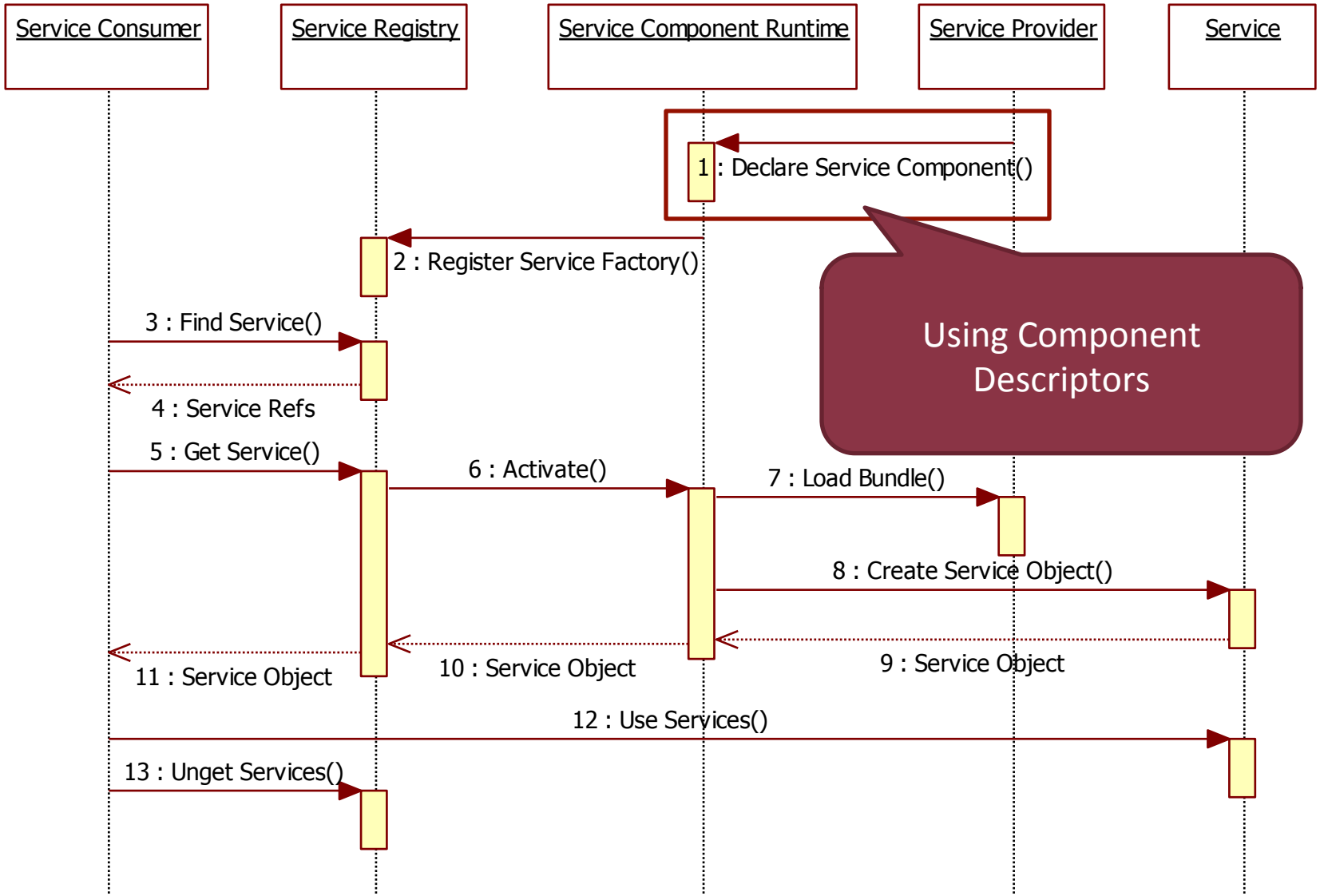
```
<reference
      bind="setService"
      unbind="unsetService"
      cardinality="0..1"
      interface="org.sample.ServiceForHello" name=„SERVICEFORHELLO"
   policy="dynamic"/>
```

Method to call when service disappears

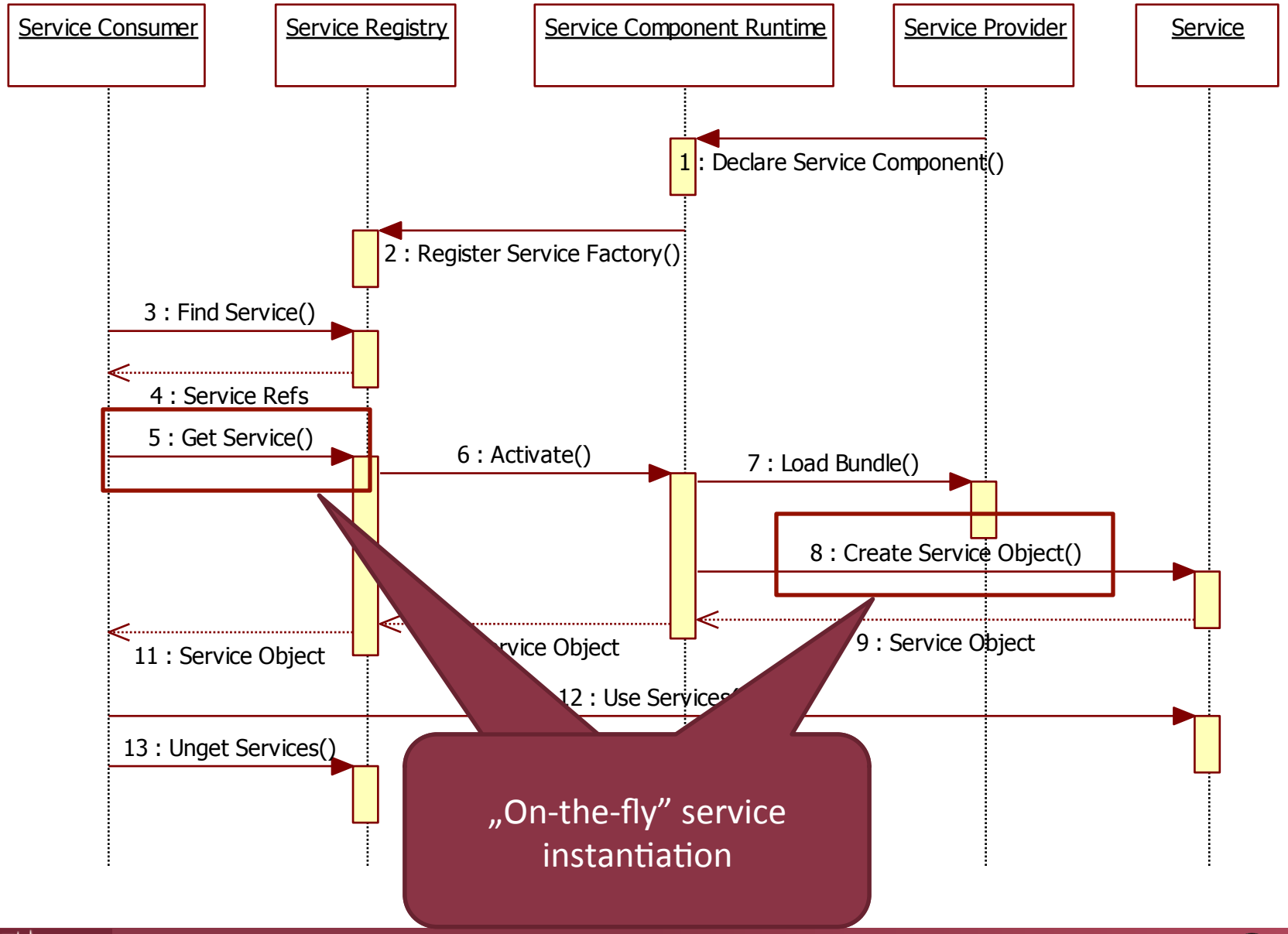
DS – Lifecycle Management



DS – Lifecycle Management



DS – Lifecycle Management



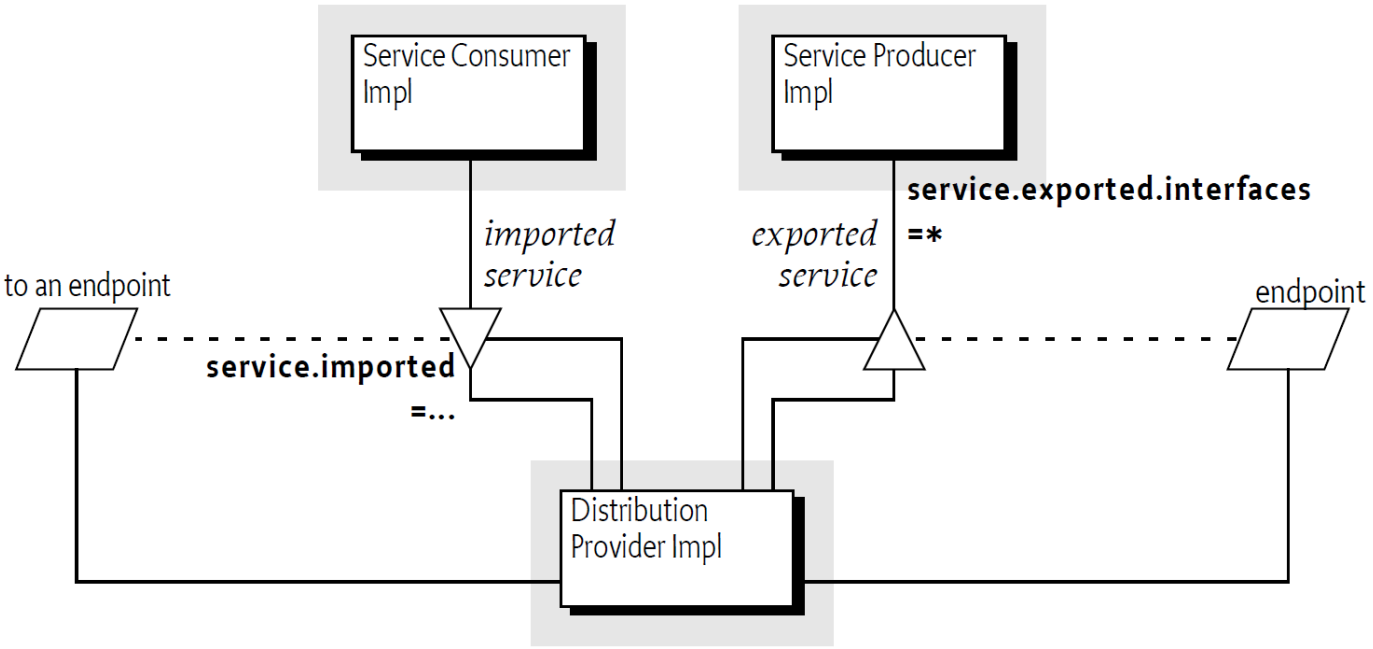
Http Services

- Very common service
- Used for OSGi-based HTTP services
- Currently supports
  - Servlet registration
    - On-the-fly
    - Allows executing OSGi in a servlet
  - Resource registration (HTML files, images, etc…)
- Application
  - Apache Felix Web Console
    - Monitoring OSGi containers

Remote Services

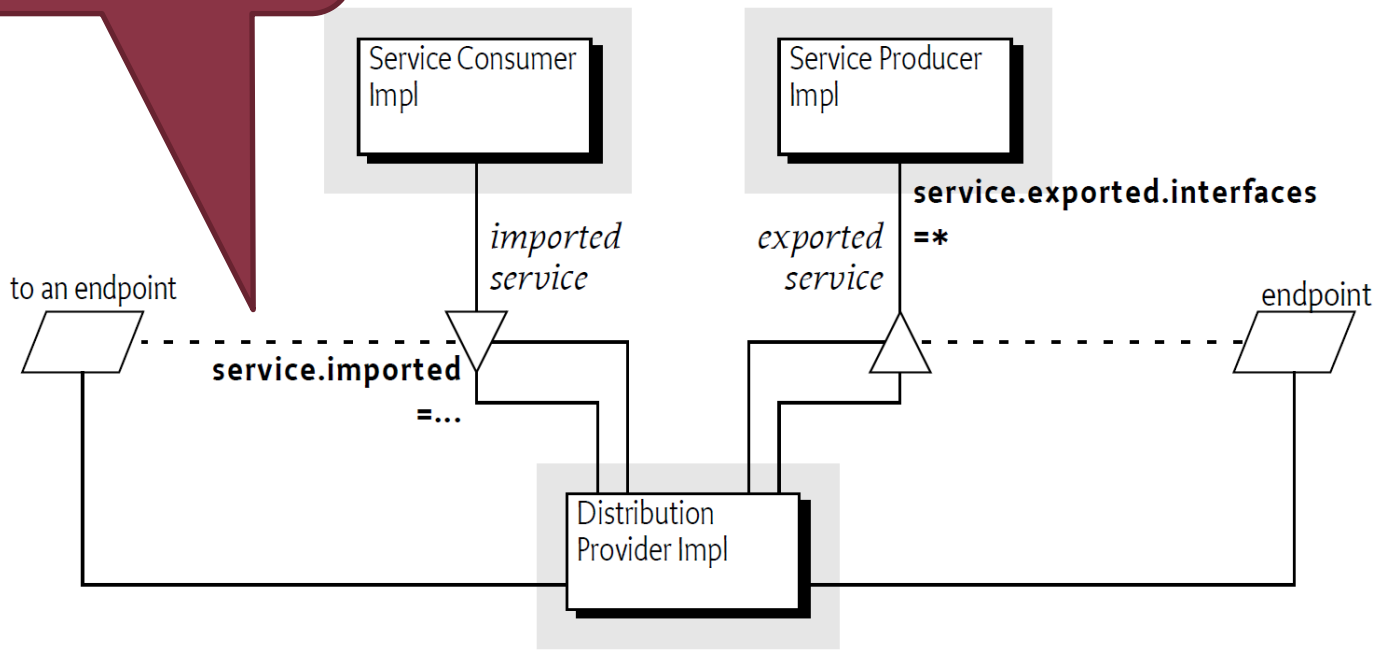
- Requirements
  - Transparency
    - Remote services are available the same way as local
  - Genericity
    - No constraints for distributed applications
  - Consistent behaviour:
    - Local and remote works the same way

Remote Services



Remote Services

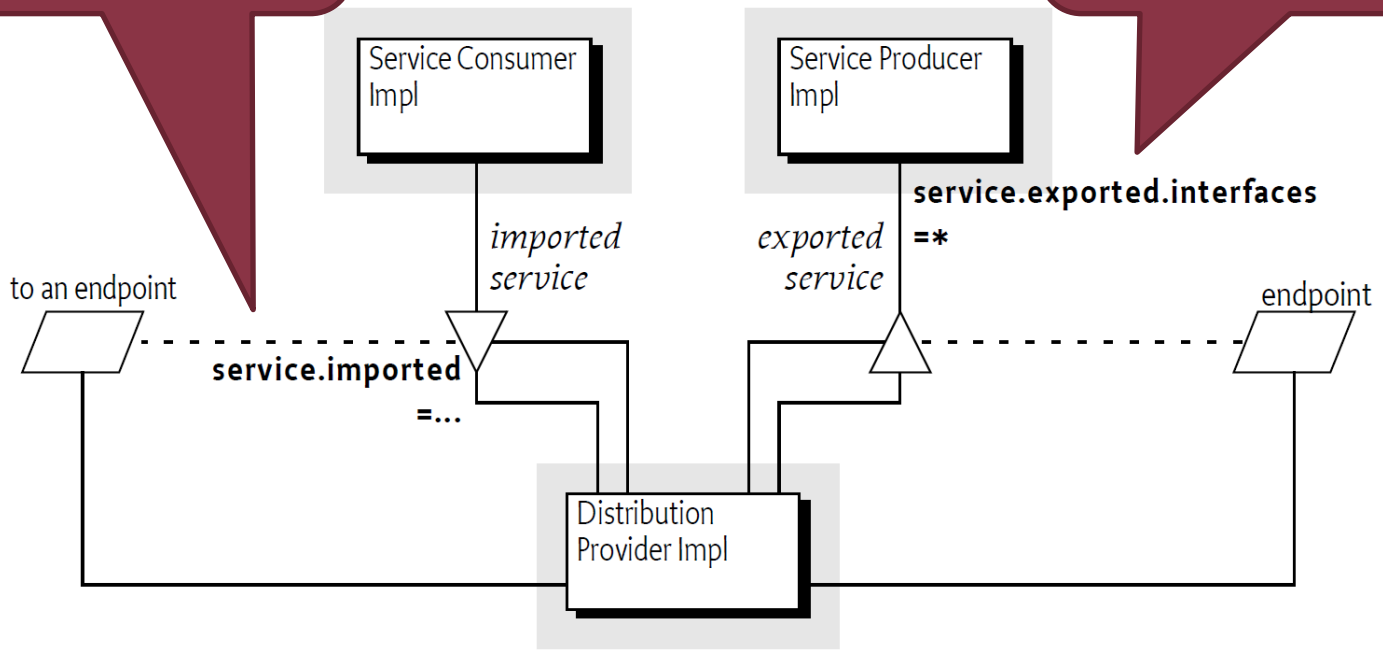
Property for imported services



Remote Services

Property for imported services

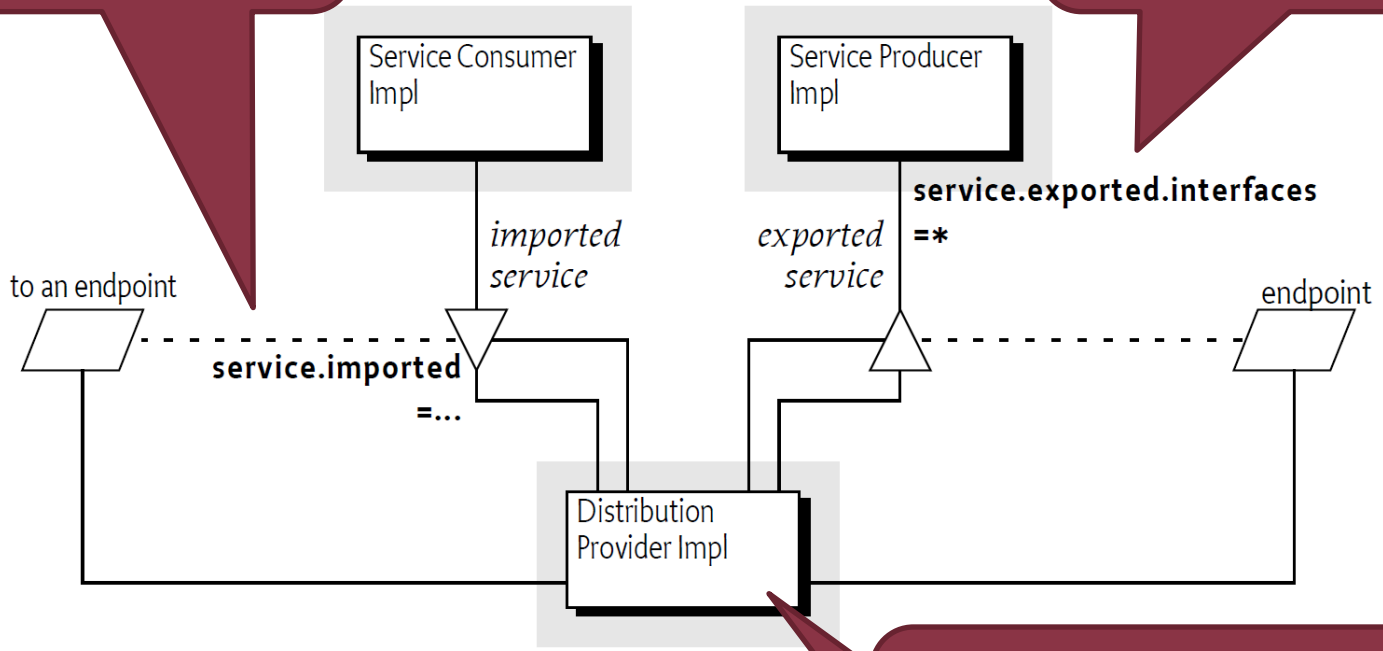
Property for exported services



Remote Services

Property for imported services

Property for exported services



- Transparent behaviour
- Service proxies

Eclipse Equinox

OSGi Implementations

- Open Source
  - **Eclipse Equinox (http://www.eclipse.org/equinox/)**
  - Apache Felix (http://felix.apache.org/)
  - Knopflerfish (http://www.knopflerfish.org/)
  - ProSyst mBedded Server Equinox Edition (http://www.prosyst.com/products/osgi_se_equi_ed.html)
- Proprietary:
  - ProSyst (http://www.prosyst.com/)
  - Knopflerfish Pro (http://www.gatespacetelematics.com/)

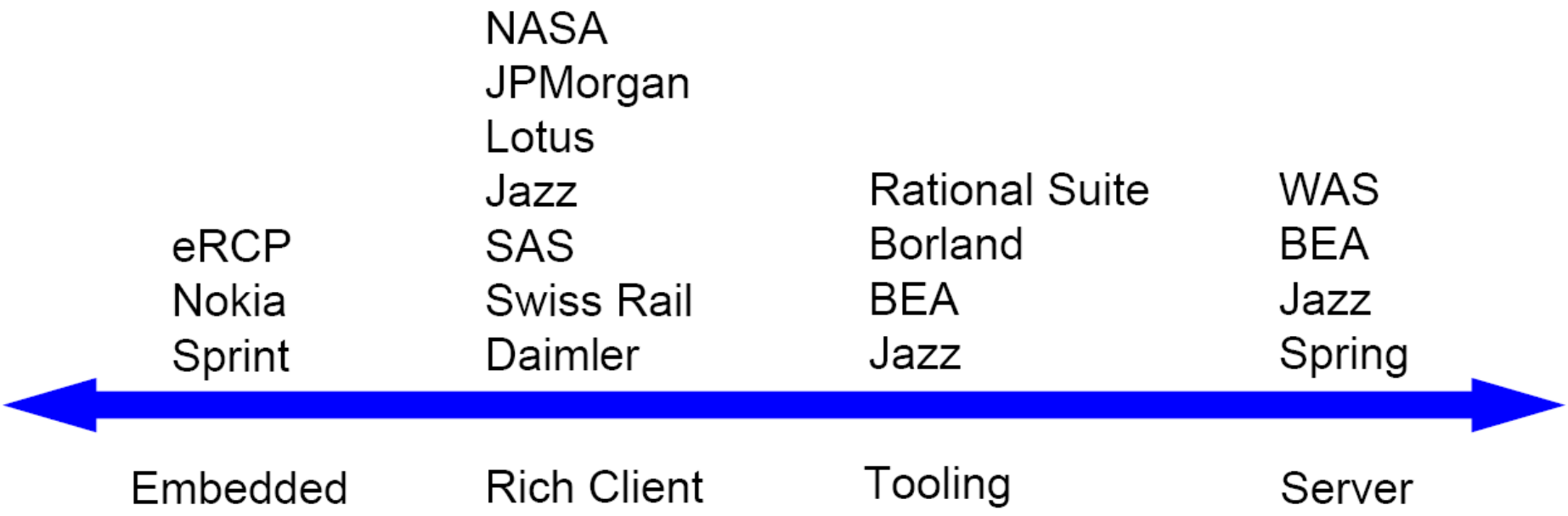
OSGi, Eclipse and Equinox

- OSGi
  - Open standard
  - Component-based applications
  - Wide application area
    - Mobile, server, desktop, enterprise, embedded
- Eclipse
  - RCP framework
  - Eclipse runtime based on OSGi (since 3.0)
- Equinox
  - Eclipse OSGi implementation (since 3.3)
  - OSGi 4.0 and 4.1 reference implementation

Equinox „Runtime”

- Combines
  - OSGi reference implementation
  - Eclipse Extension mechanism
- Also usable as application server!

Usage


NASA
JPMorgan
Lotus
Jazz
SAS
Swiss Rail
Daimler
Rational Suite
Borland
BEA
Jazz
WAS
BEA
Jazz
Spring
eRCP
Nokia
Sprint
Embedded
Rich Client
Tooling
Server

Equinox as a Server

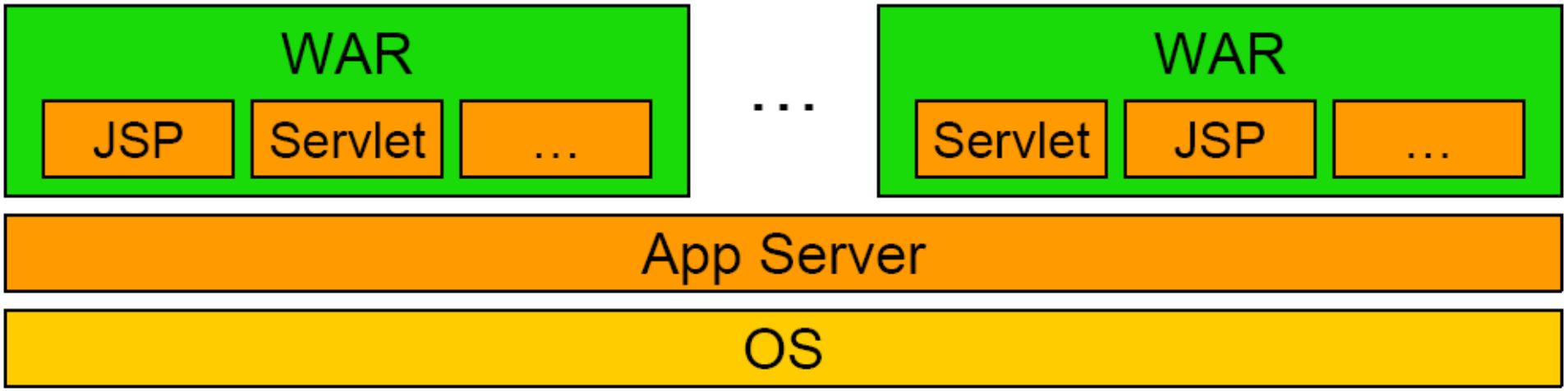
- Equinox runtime + server side addons
  - HTTP service/registry
  - Jetty: lightweight webserver
  - Integration bundles (ServletBridge=Servlet/JSP, etc.)

Equinox in an app server

- Traditional Application Server
- Equinox in Application Server
- Pure Equinox

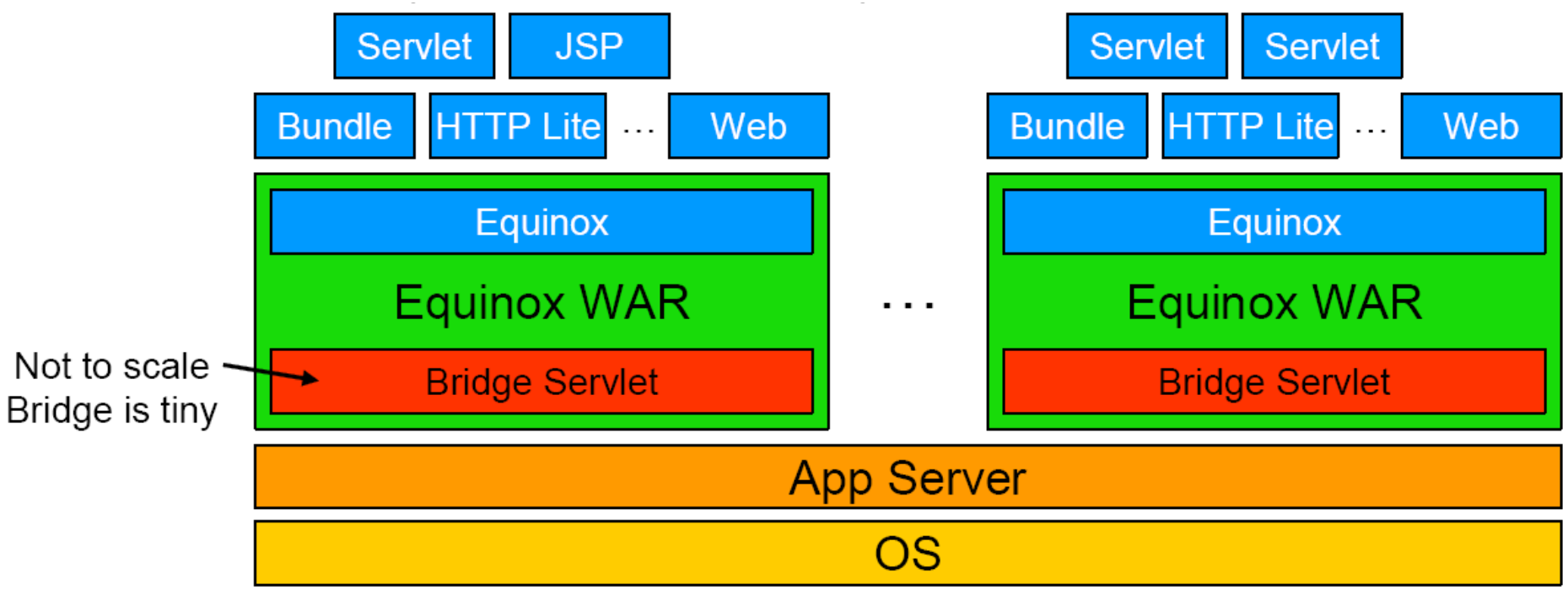
Traditional Application Server

- Self-contained „war” applications
  - Isolation
- Application management
  - Only on application level



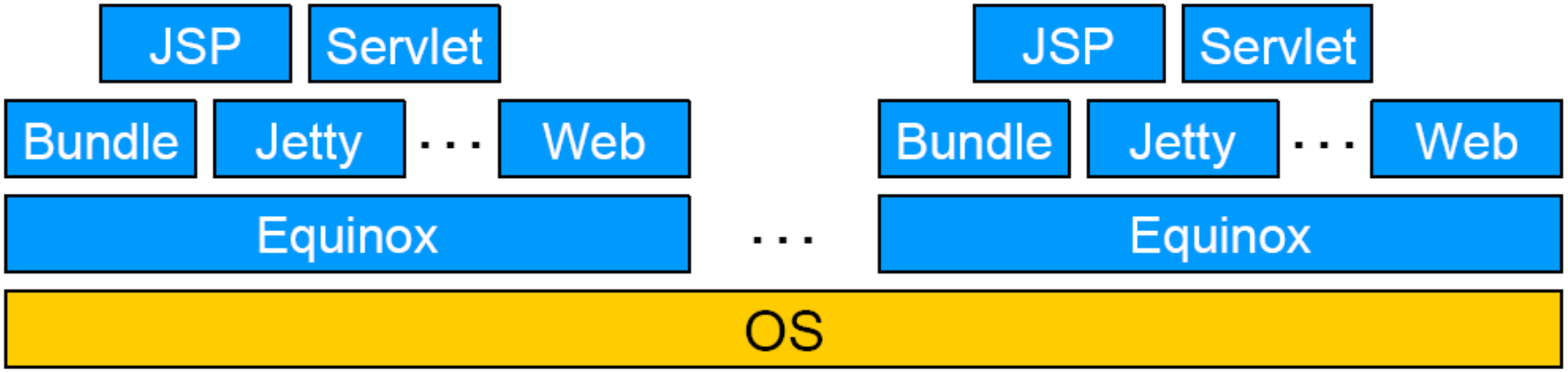
Equinox in Application Server

- „Bridge Server”
- Isolated applications
- Reuses existing infrastructure



Pure Equinox

- Execute Equinox directly
- Processes are isolated
- HTTP via Jetty
- App install/update/… via bundle management



Benefits

- Incremental updates
- Multiple instances parallel → HA / performance
- Management on different leveles
- Easy to differentiate
- Better class loading performance
- Component sharing between client and server

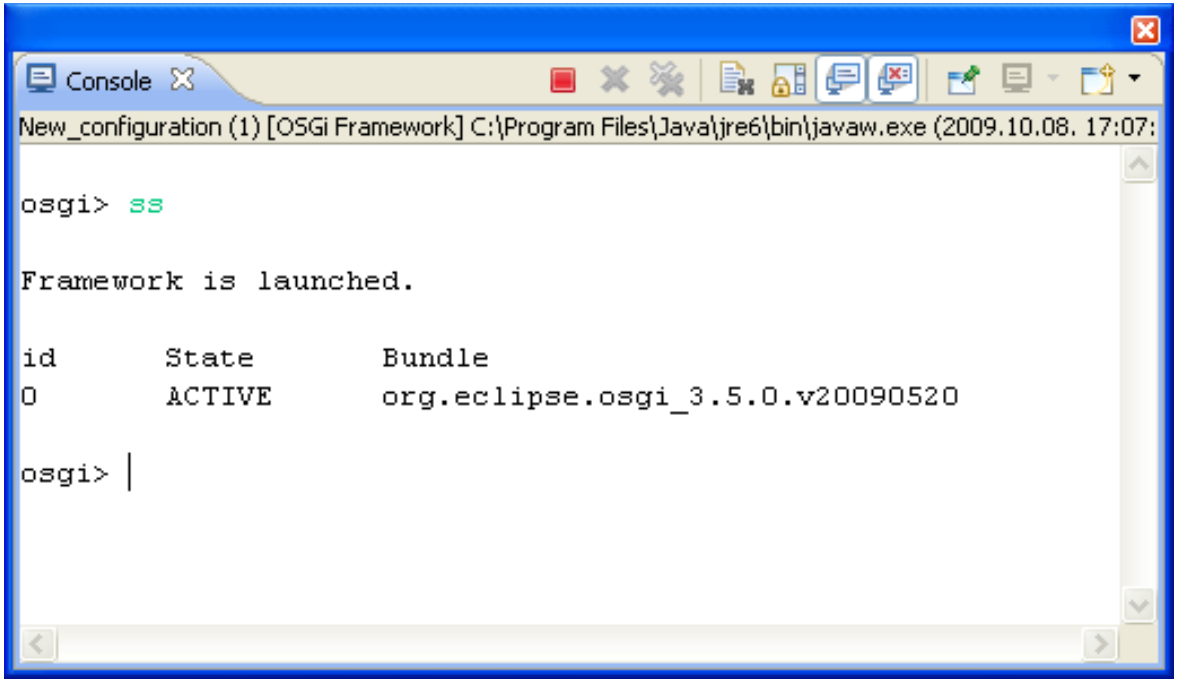
Eclipse RT

- Runtime components in Eclipse
  - Eclipse Communication Framework
    - Common protocol implementations
  - Jetty
    - Servlet Engine and Http Server
  - Virgo
    - Application Server
  - Riena
    - Client-server development focus
  - …

OSGi Console

OSGi Console

- OSGi prompt
  - Similar to DOS/Bash shells
- Eclipse support
  - Console view
  - Highlighting

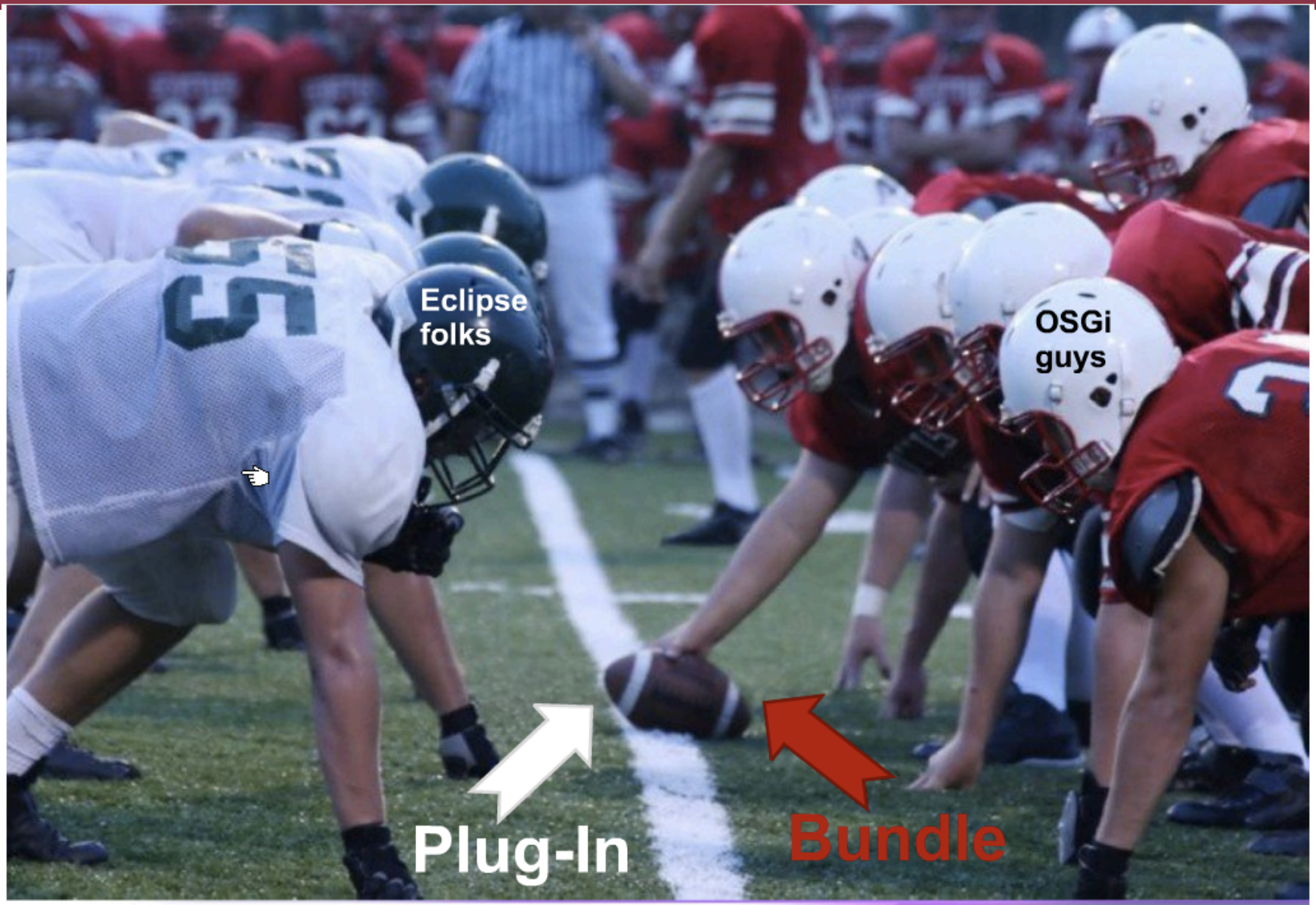


OSGi Console - Commands

- Useful commands
  - ss: lists installed bundles
  - start <id>: starts a bundle
  - stop <id>: stops a bundle
  - install file:<path>: install a bundle from a file
  - uninstall <id>: uninstalls a bundle
  - update <id>: updates a bundle
  - services <filter>: lists all services
    - E.g. osgi> services (objectClass=*HelloService)
  - shutdown: shuts down framework
  - close: shutdown and exit
  - exit: ~ System.exit

Eclipse vs OSGi

Bundle vs plug-in



Eclipse extensions vs. OSGi services

- Eclipse extension
  - Giving others option to contribute (master-slave)
  - Usually UI extensions (too small for OSGi services)
    - Often not code-based
      - Themes
      - Command framework
- OSGi service
  - Anybody can define services
  - Anybody can use services
  - Very dynamic framework
  - Loose coupling

Eclipse extensions vs. OSGi services

	Extensions	Services	Declative Services
Registration	XML declaration	Java objects	Java Objektumok (+ Proxy until first use)
What to register	Every **extension** from each **plugin.xml,** *automatically*	**BundleContext** API, *manually*	**Service-Component** descriptors, *automatically*
Usage	**Extension point ID**	Interface name + property filters	As services; SCR fills in the objects
Multiplicity	*One-to-many*: one **Extension point** for each **Extension**	*Many-to-many*: no limitation	Ua., mint services
Loading	XML declaration on-start, for classes *lazy-loading*	Before usage	On-demand

Required-Bundle vs. Import-Package

- Required-Bundle
  - Eclipse uses this
  - Imports every package
    - exported by the bundle
    - handles reexports as well
  - Stronger coupling
- Import-Package
  - OSGi mostly relies on them
  - Only the selected packages imported
  - Loose coupling