OSGi Design Patterns and Best Practices

Marcel Offermans

luminis TECHNOLOGIES

Marcel

- Member Apache Software Foundation
  - Committer and PMC member at Apache Felix and Apache ACE
- marrs@apache.org
- Fellow and Software Architect at Luminis Technologies
- marcel.offermans@luminis.nl

Agenda

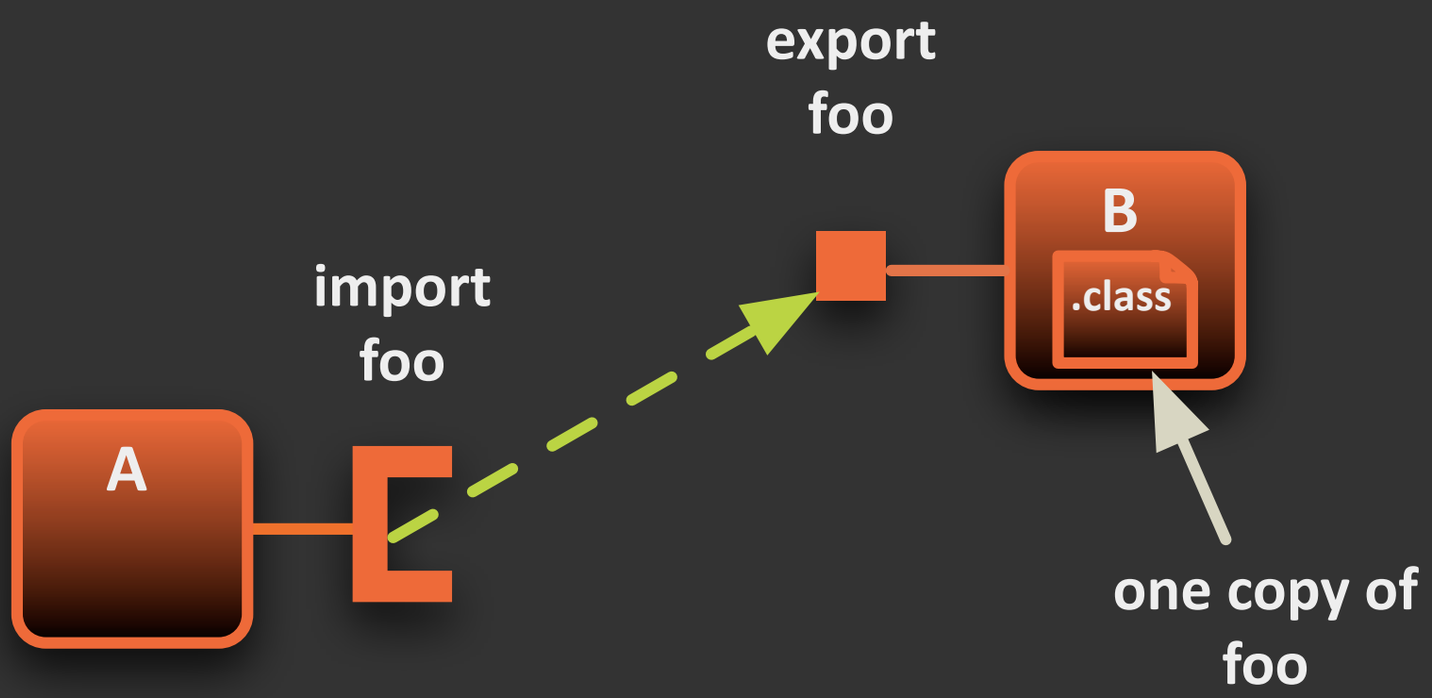
- Modularity
- Versioning
- Life Cycle
- Lean Development
- Service Interface Design
- Using the OSGi Container
- Design Patterns

Modularity

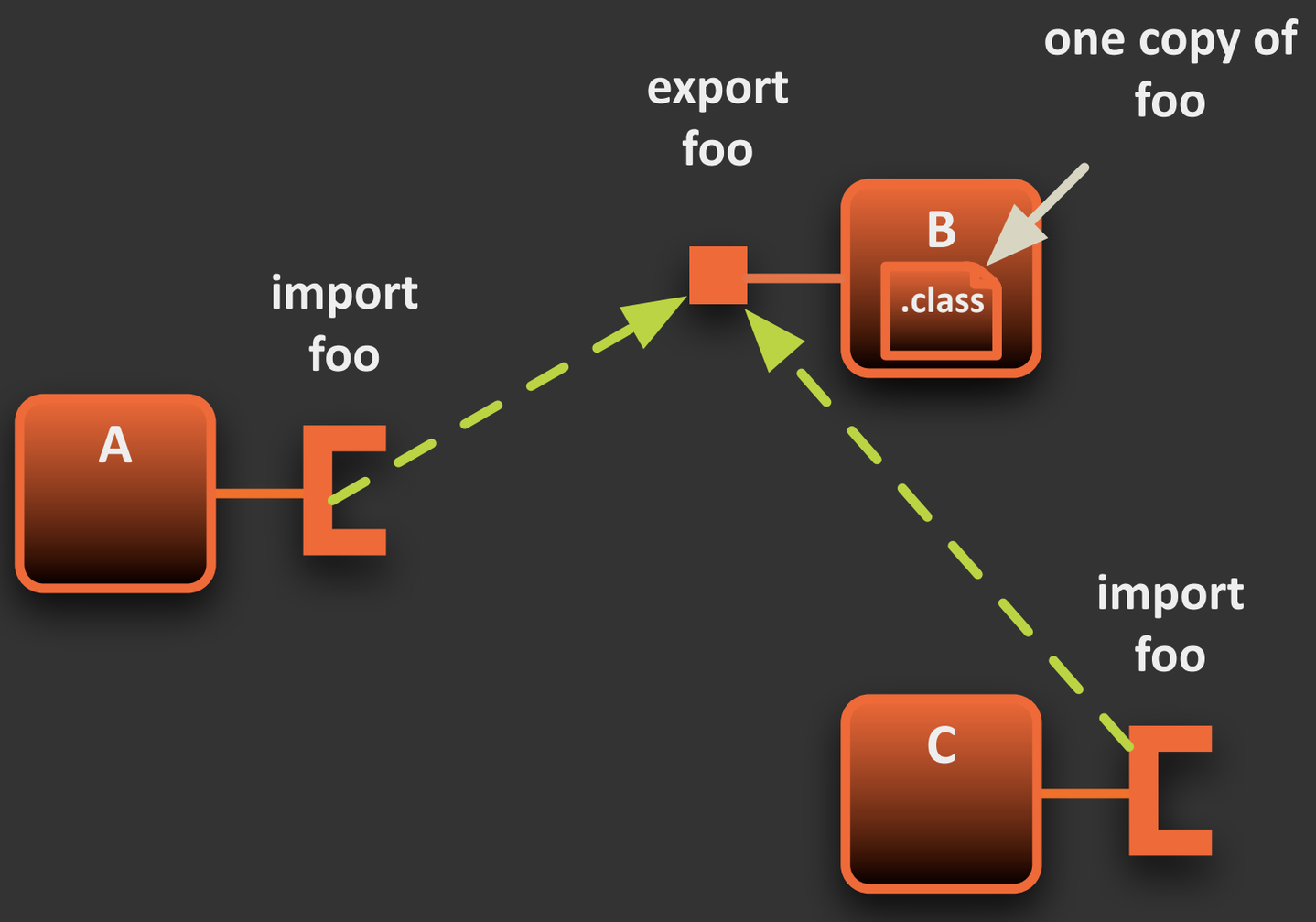
Design Considerations

- Minimize dependencies
- Consider rate of change
- Minimize complexity
- Hide your implementation, including libraries

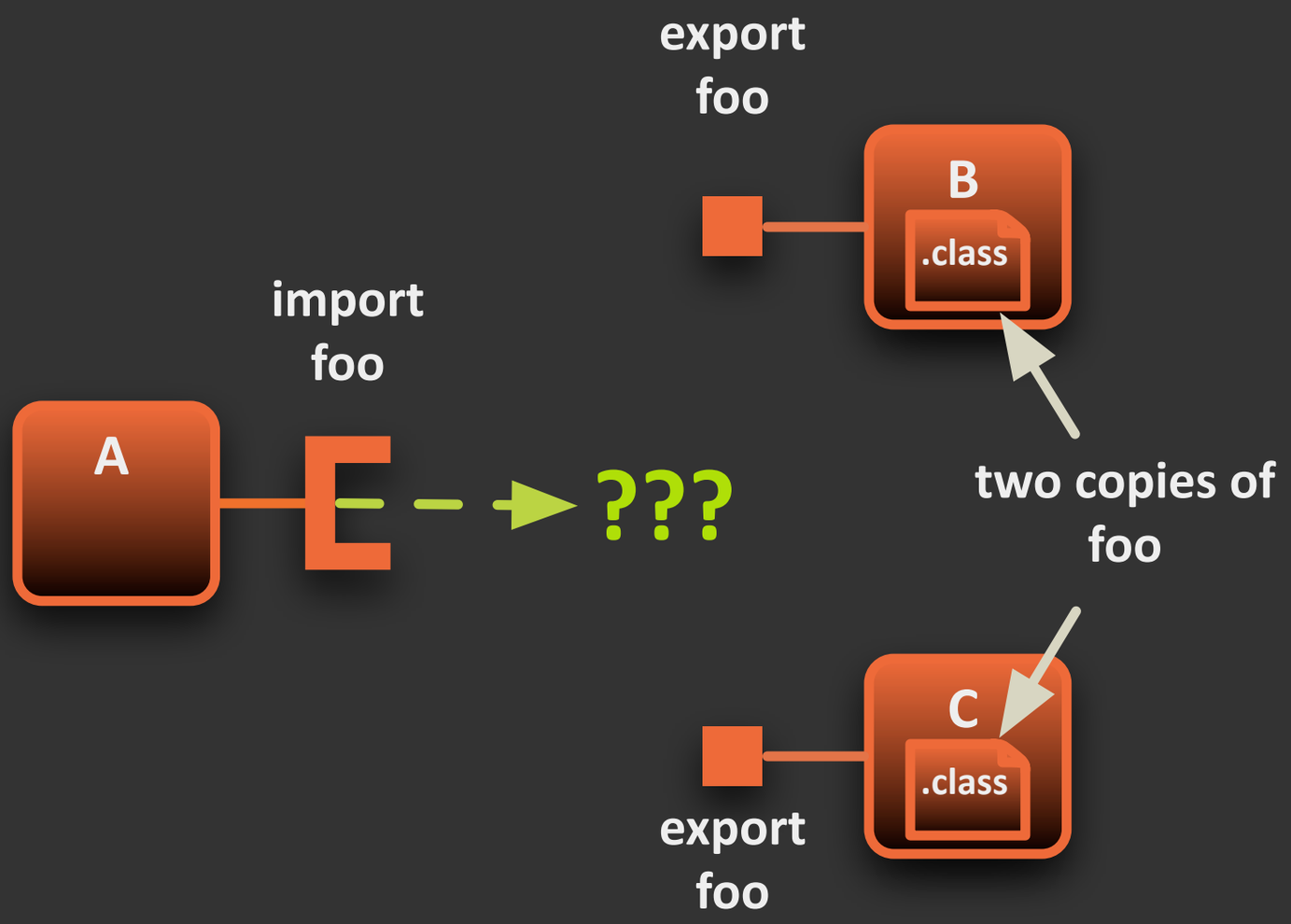
Import what you export



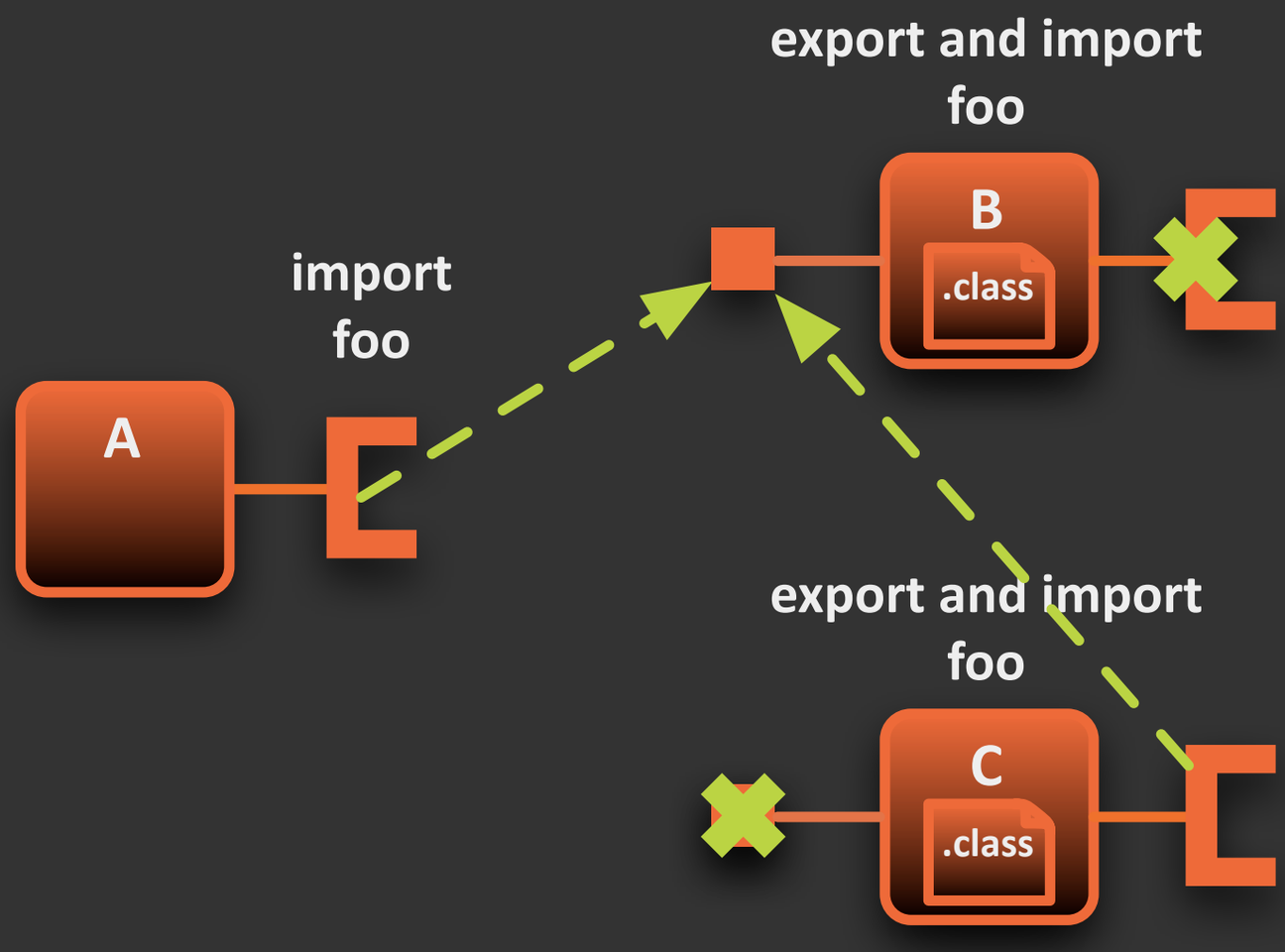
Import what you export



Import what you export



Import what you export



Versioning

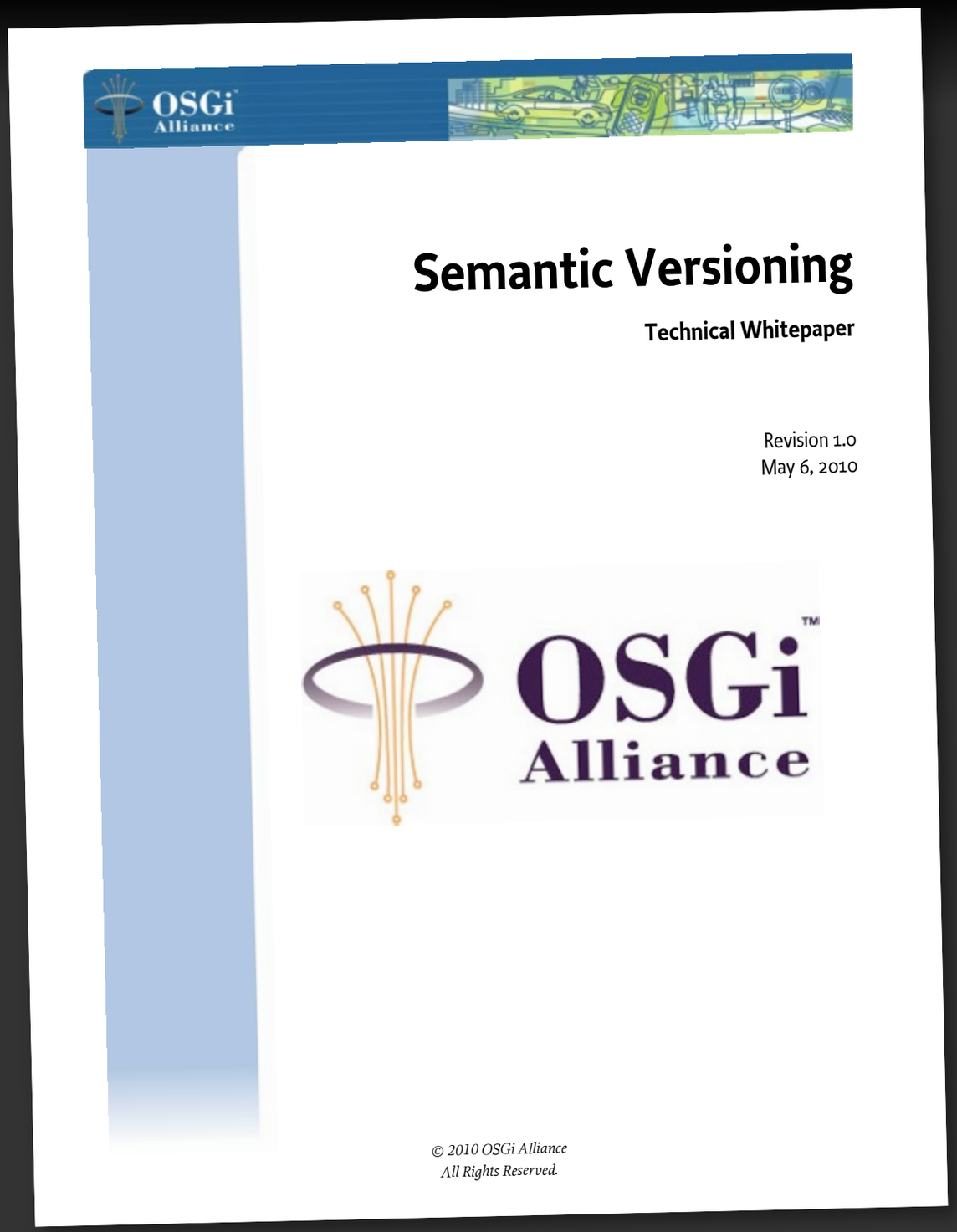
Version Format

- OSGi uses a 4-part version numbering scheme
  - <major>.<minor>.<micro>.<qualifier>
  - Major, minor, and micro are numeric values
  - Qualifier is a string value
  - Valid examples:
    1, 1.10, 1.9.9.alpha, 0.2.0.SNAPSHOT
  - Invalid examples:
    0.2.SNAPSHOT, 1.9.9-alpha
  - Comparison (not always intuitive):
    1.0.0 < 1.9.9 < 1.10, 1.0.0.beta > 1.0.0.alpha > 1.0.0

Version Range Format

- Interval notation is used for version ranges
  - Use '[' or ']' for inclusive values
  - Use '(' or ')' for exclusive values
  - Example (quotes needed)
    - "[1.0.0,2.0.0)" == (1.0.0 <= v < 2.0.0)
- When a version range is expected, a single value represents an infinite range
  - "1.0.0" == (1.0.0 <= v < ∞)
- To specify a precise version
  - "[1.0.0,1.0.0]"

Tip: Semantic Versioning whitepaper



Source: http://www.osgi.org/wiki/uploads/Links/SemanticVersioning.pdf

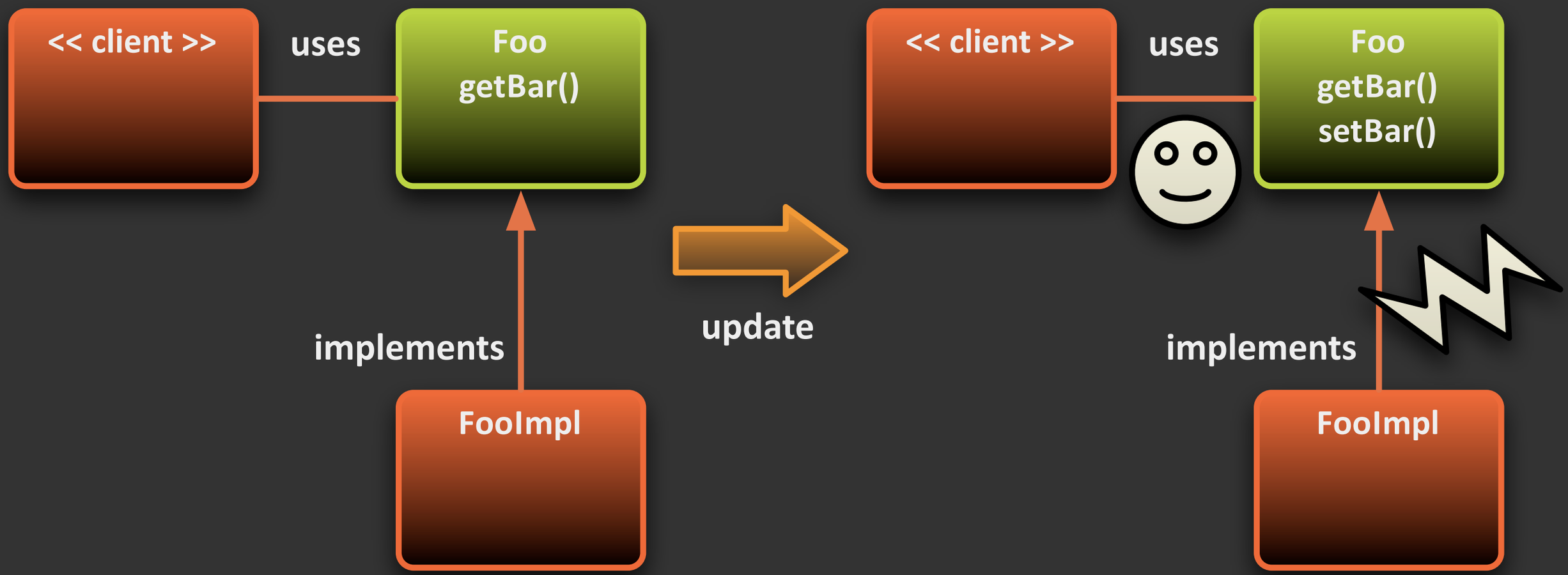
Version Policy

- Major number change – signifies an incompatible update.
- Minor number change – signifies a backward compatible update.
- Micro number change – signifies an internal update (e.g., a bug fix or performance improvement).
- Qualifier change – signifies a trivial internal change with “outward” noticeable difference, but nonetheless is a new artifact (e.g., line number refactoring).

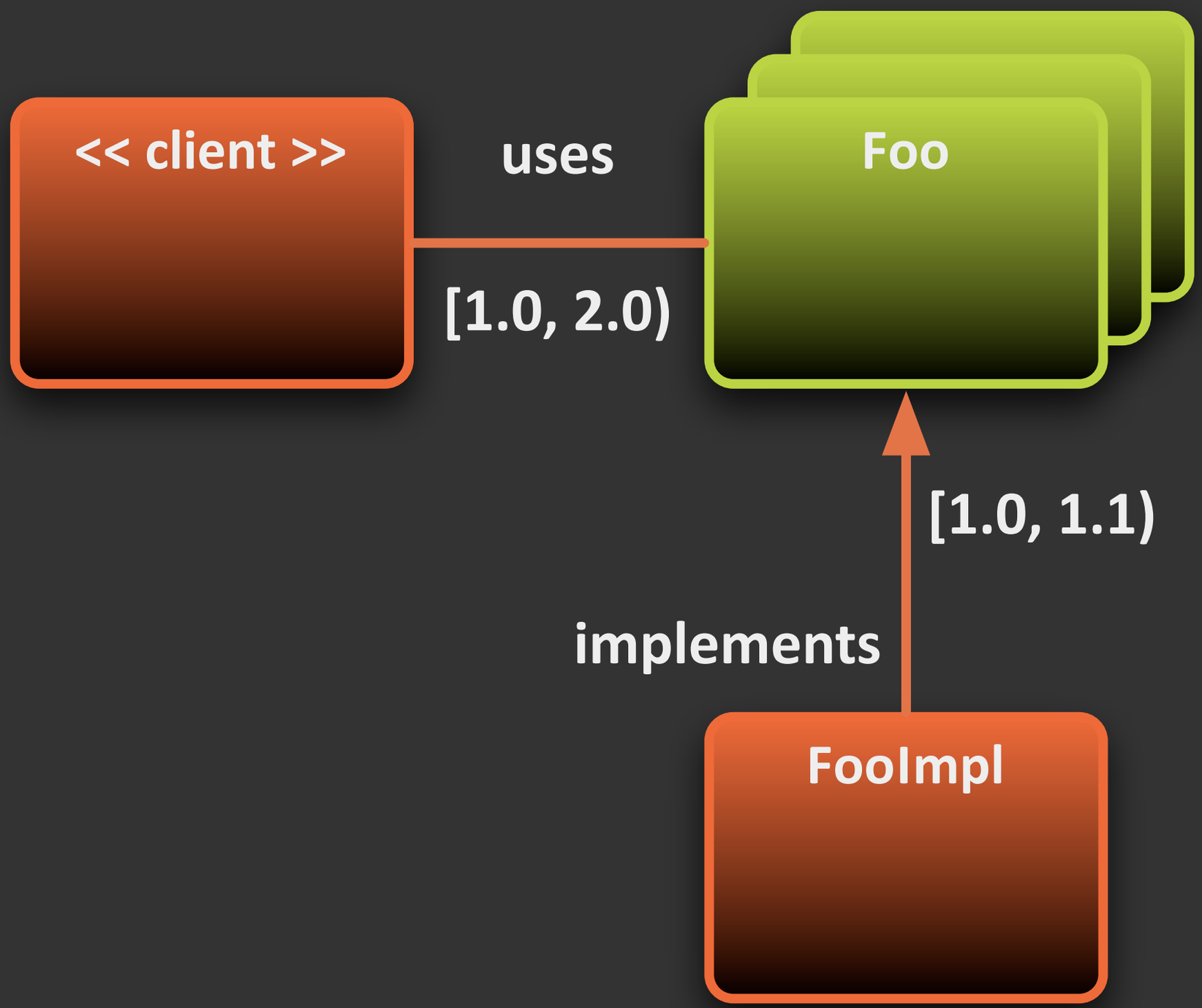
Experience: versioning

- Choose a versioning policy and stick with it
- Version all packages you export
- Use version ranges on import that are consistent with policy
- If you FORGET this you will REGRET it
- Do not be tempted to mis-use commercial versions internally
- Only bump versions on actual changes

Backward Compatibility Semantics



Best Practice



Life Cycle

Launcher API

- Standardized way of instantiating and starting an OSGi framework
- Present since R4.2
- Launching is a three step process:
  - First obtain an implementation of FrameworkFactory
  - Then use the factory to instantiate a framework
  - Finally, start the created framework

Launcher Example

```
void launch(String factoryName, File[] bundles) throws Exception {
  Map p = new HashMap();
  p.put("org.osgi.framework.storage",
      System.getProperties("user.home") + File.separator + "osgi");
  FrameworkFactory factory =
      (FrameworkFactory) Class.forName(factoryName).newInstance();
  Framework framework = factory.newFramework(p);
  framework.init();
  BundleContext context = framework.getBundleContext();
  for (File bundle : bundles) {
    context.installBundle(bundle.toURL().toString());
  }
  framework.start();
  framework.waitForStop();
}
```

Launcher using ServiceLoader

- **Java 6 feature**
- **Needs a JAR with**
  `META-INF/services/org.osgi.framework.launch.FrameworkFactory`
  containing
  `org.foo.osgi.Factory`

```
ServiceLoader<FrameworkFactory> sl =
    ServiceLoader.load(FrameworkFactory.class);
Iterator<FrameworkFactory> it = sl.iterator();
if (it.hasNext()) {
  FrameworkFactory factory = it.next();
  Framework fw = factory.newFramework(null);
  // ...
}
```

Startup and Ordering

- Do not rely on startup ordering
- Do not use start levels for startup ordering
- Test starting your bundles in a random order
- Do not clear your bundle cache on every (re) start

Deploying and Updating

- Make everything deployable
- Use a proper management agent to do updates

Lean Development

Anticipate Change

Lean Development

"Non-Functionals"

Great functionality! And can you do that, in a ... way

Based on Luke Hohmann's book Beyond Software Architecture

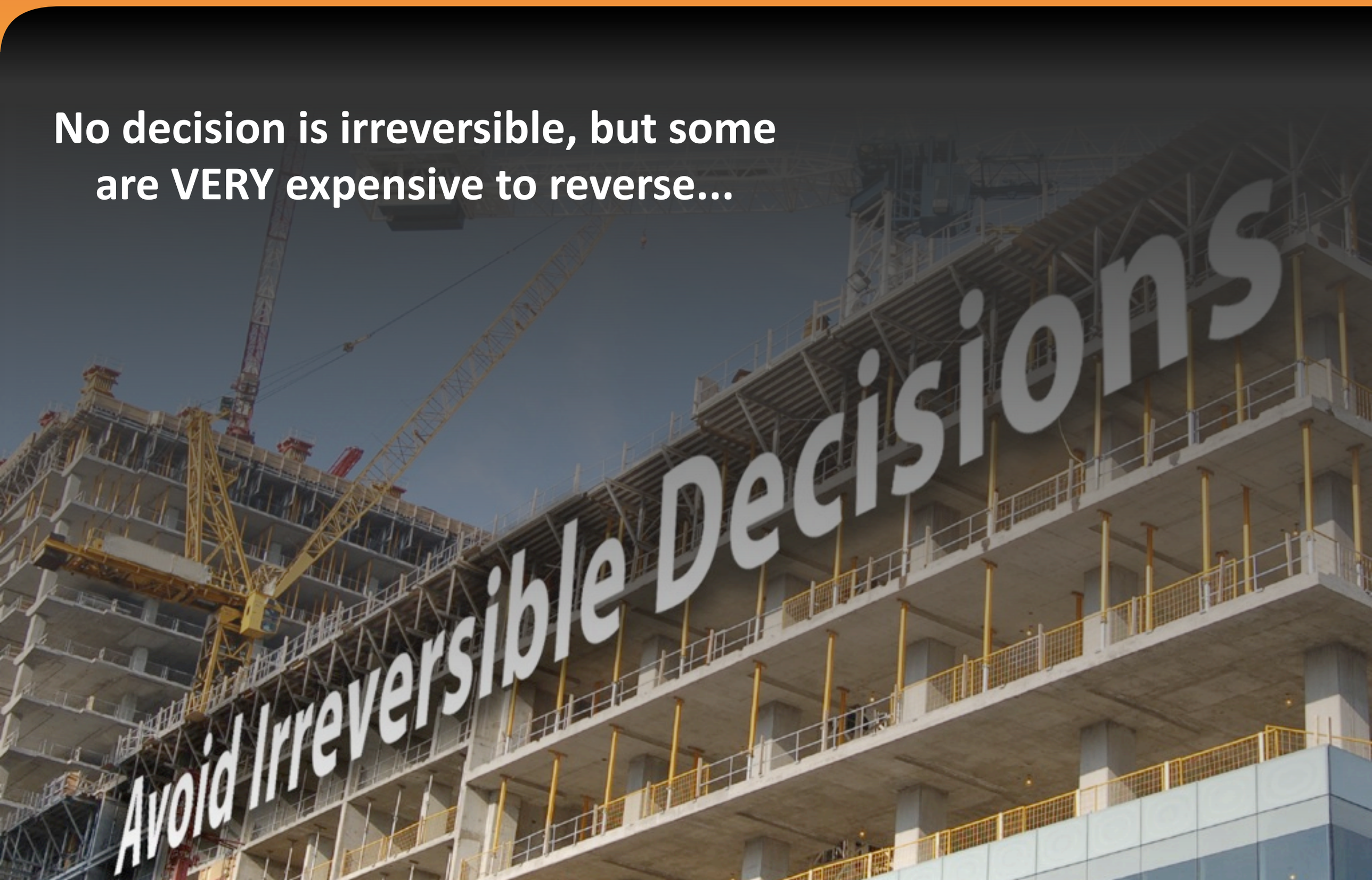
Lean Development

Anticipate, don't over dimension

Do not try to support the things now that MIGHT be required next...

Lean Development

No decision is irreversible, but some are VERY expensive to reverse...



Service Interface Design

Design for Change

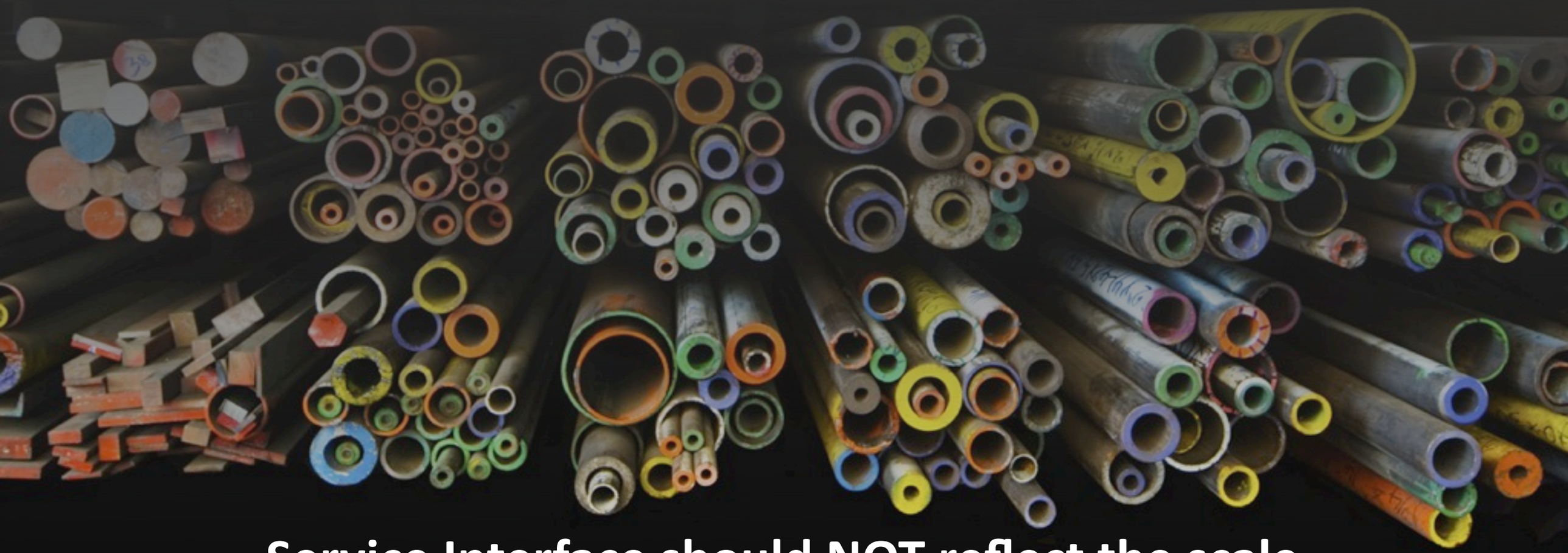
Behavior Only

Do NOT disclose parameters and implementation details

Do NOT shape a Service after your first/only implementation

Design for Change

Scale Invariance



Service Interface should NOT reflect the scale your (first) implementation will support

Design for Change

Decoupling

If your implementations require other services,
access these services through interfaces only.

Design for Change

Think BIG

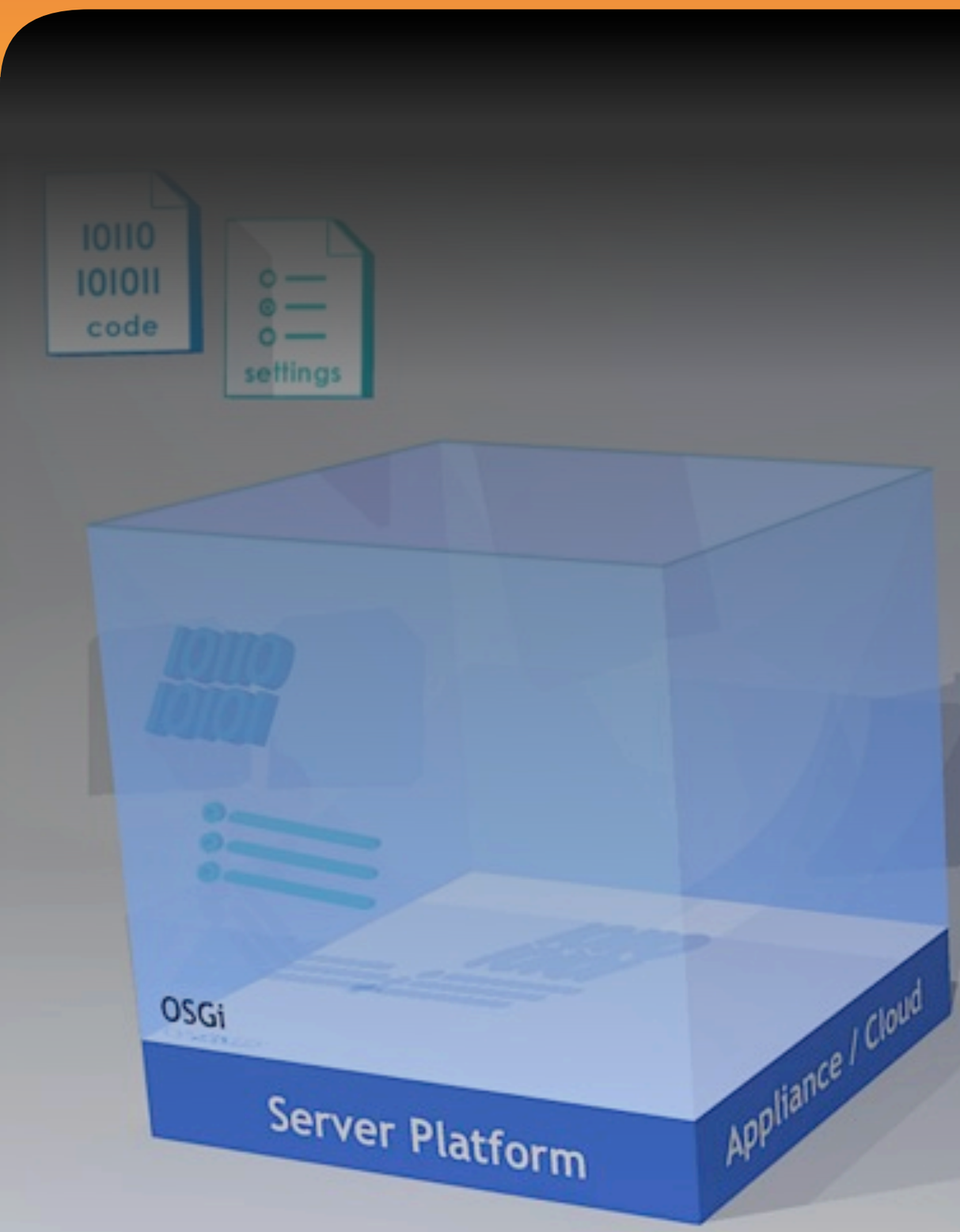
act small!

Your service interface should be future proof,
your implementation should fit your current needs.

Keep many possible implementations
(functionality, scale) in mind when designing interfaces.

Using the OSGi Container

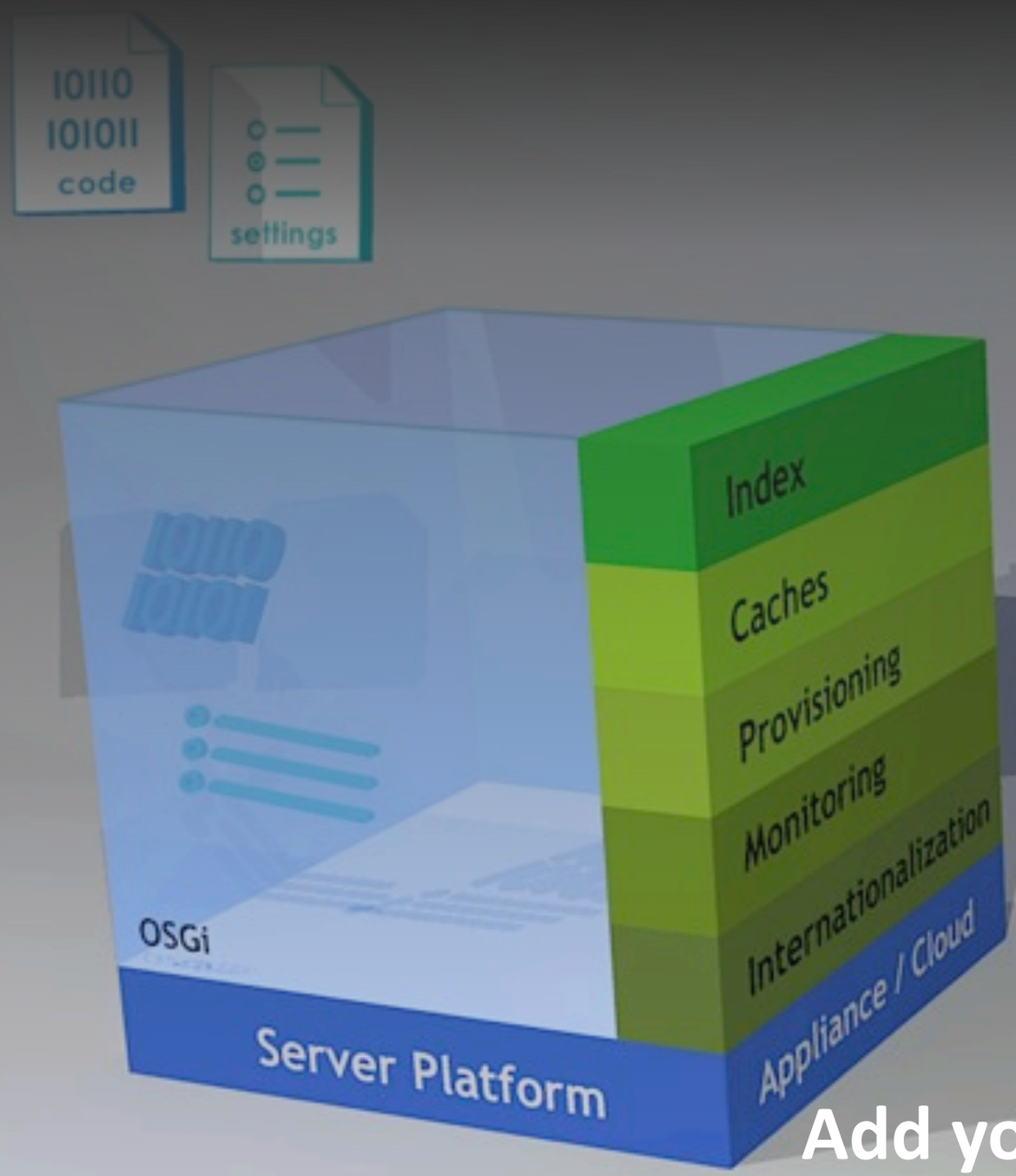
Container Style Development



Application consists of interdependent services

Public services are accessed through UI/REST/SOAP/JMS/...

Be a Good Host

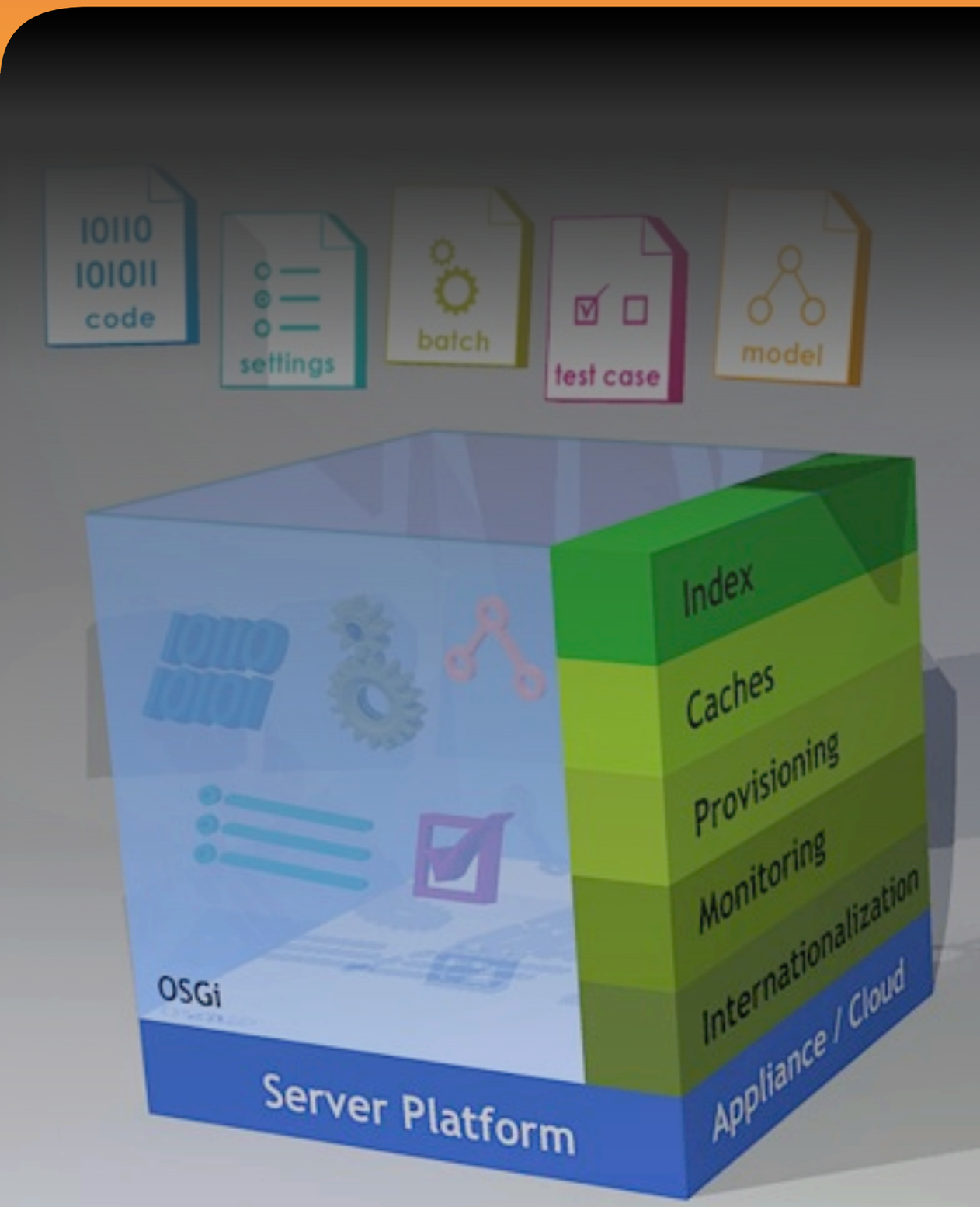


Offer facilities to your components

OSGi Compendium is a Catalog of generic Facilities

Add your own to the container if needed

Deploy Everything



Not just code!

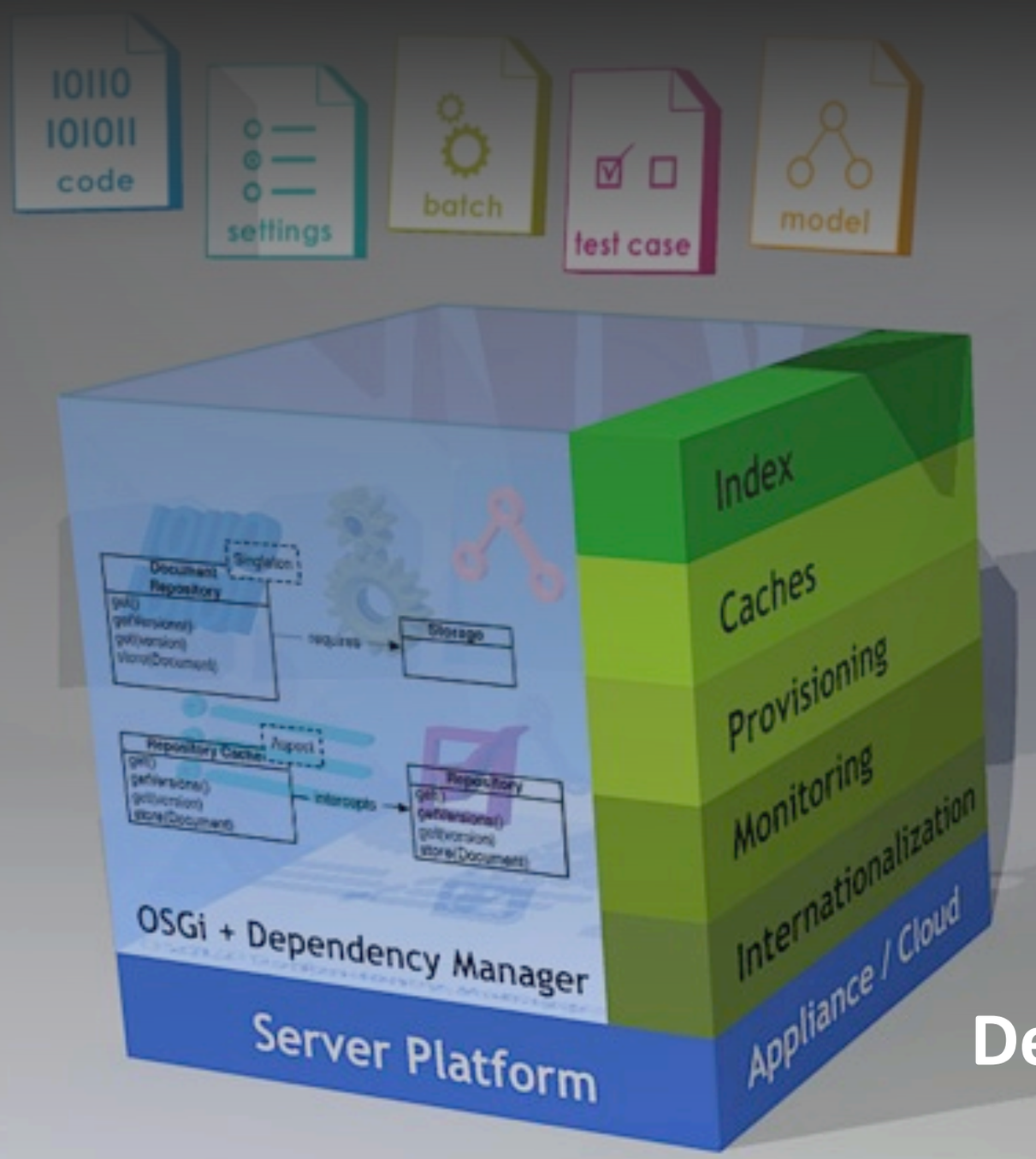
Configuration
Content
Test Cases

Inversion of Control

Use declarative life cycle and Dependency Management

Develop POJO's
Delegate injection

Add Chemistry



Deployed services react on each other

Resulting in behavior and new services

Define these reactions in terms of life cycle and dependencies

Design Patterns

Design Patterns for OSGi

- Whiteboard Pattern
- Null Object Pattern
- Singleton Services
- Aspect Services
- Adapter Services
- Resource Adapter Services

Whiteboard Pattern

“don’t call us...

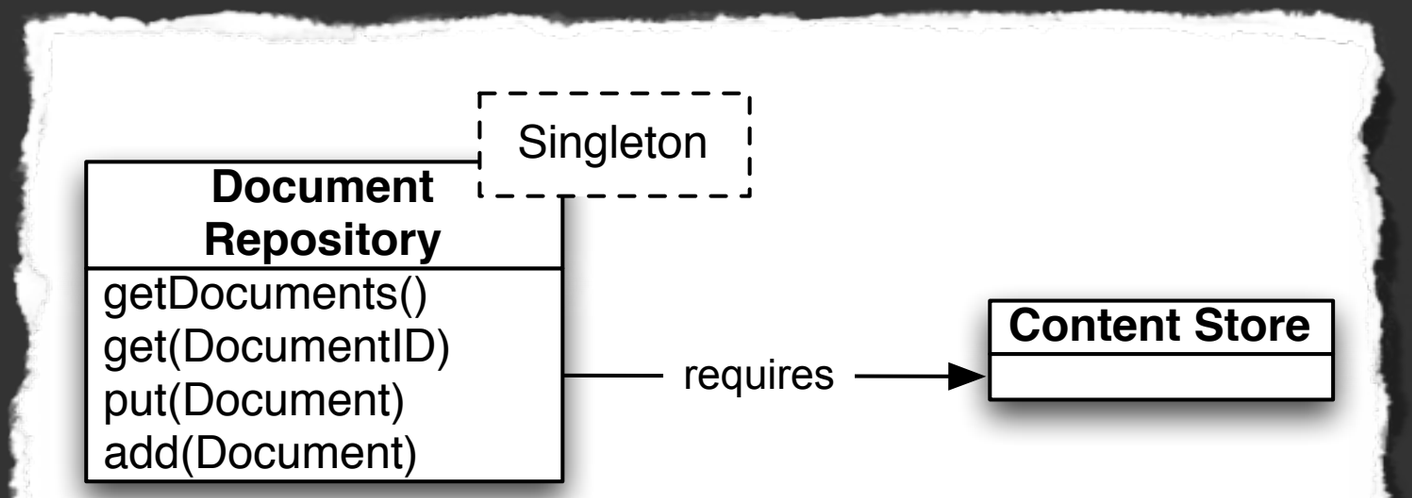
we’ll call you”

Null Object Pattern

An object that implements a certain interface, can be safely invoked and does nothing

Singleton Services

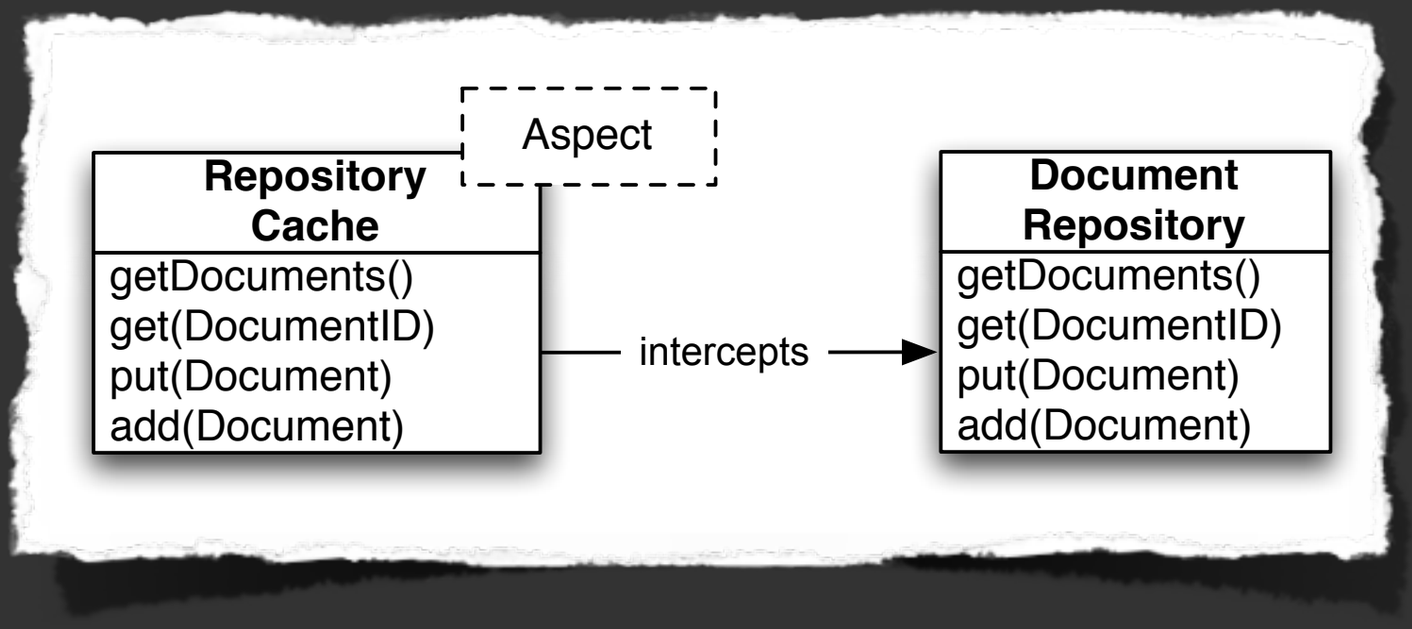
Bind logic in a single instance




```
class Activator {
    public void init(context, manager) {
        manager.add(createComponent()
            .setInterface(DocumentRepository.class.getName(), null)
            .setImplementation(DocumentRepositoryImpl.class)
            .add(createServiceDependency()
                .setService(ContentStore.class)
            ));
    }
}
```

Aspect Services

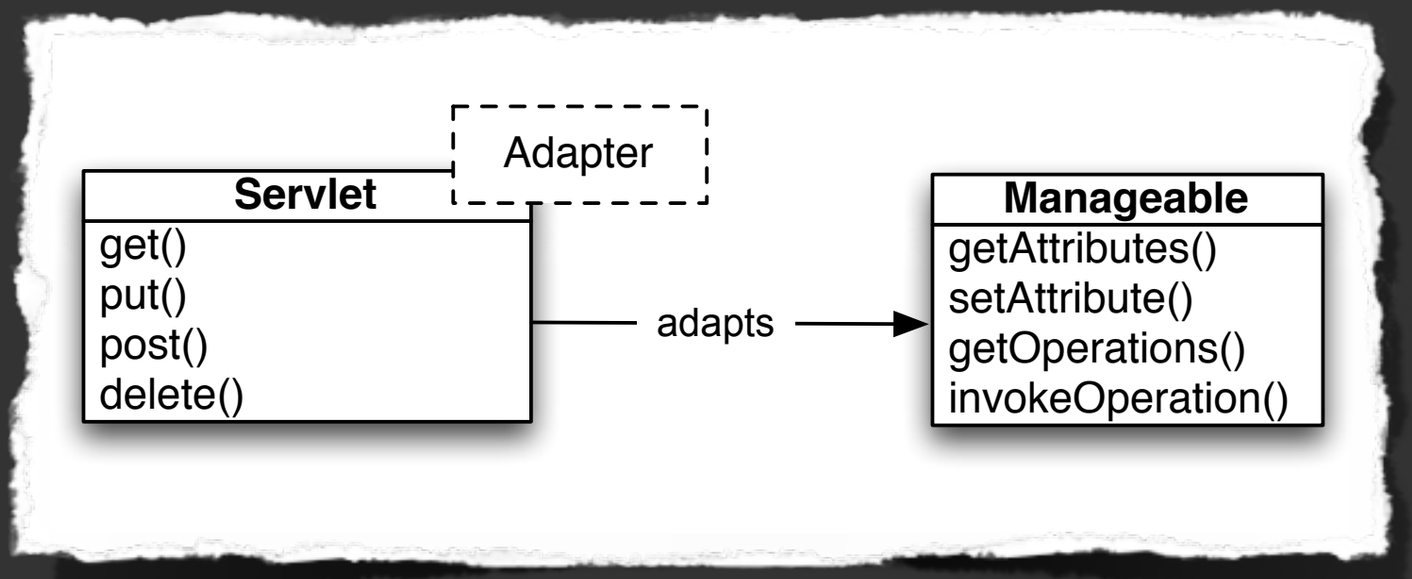
Transparently inject an interceptor service "in front of" all services matching a filter




```
class Activator {
   public void init(context, manager) {
         manager.add(createAspectService(
            DocumentRepository.class, null, 10, null)
            .setImplementation(DocumentRepositoryCache.class)
         );
   }
}
```

Adapter Services

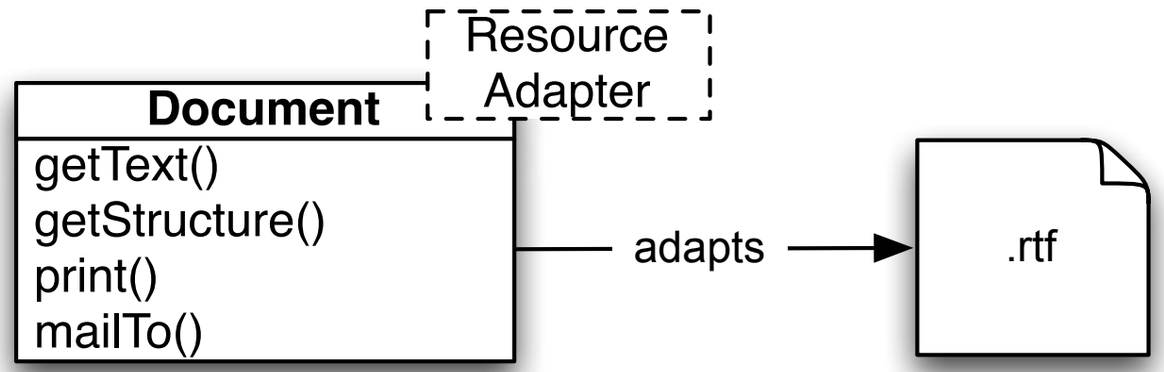
Start an instance of the adapter service for any "adaptee" service matching a filter




```
class Activator {
   public void init(context, manager) {
      manager.add(createAdapterService(Manageable.class, null),
            .setInterface(Servlet.class.getName(), null),
            .setImplementation(ServletAdapter.class)
      ));
   }
}
```

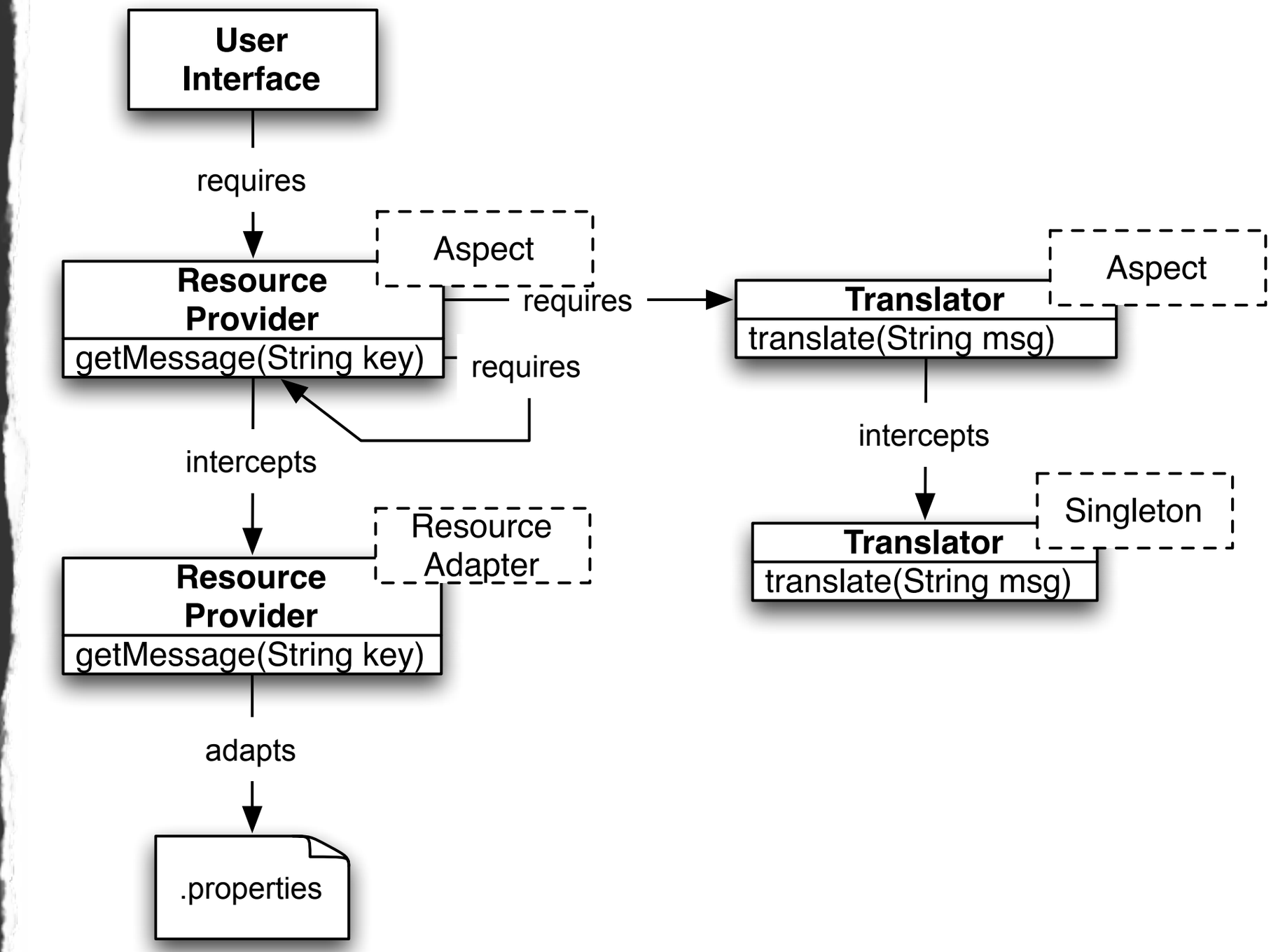
Resource Adapter Services

Start an instance of a Resource Driven Service for all resources matching a filter




```
class Activator {
    public void init(context, manager) {
        manager.add(createResourceAdapter(
            "(path=*.mp3)", false, null, "changed")
            .setInterface(AudioTrack.class, null)
            .setImplementation(MP3AudioTrack.class)
        ));
    }
}
```

An example...


User Interface
requires
Resource Provider
getMessage(String key)
Aspect
requires
Translator
translate(String msg)
Aspect
requires
intercepts
Resource Provider
getMessage(String key)
Resource Adapter
adapts
.properties
intercepts
Translator
translate(String msg)
Singleton

Wrapup

Jan 18-21, Dublin/Pleasanton, CA

Masterclass on OSGi

http://njbartlett.name/2010/11/02/masterclass-usa.html

code: APC11 -10%

Mar 21-24, Santa Clara, CA

OSGi DevCon and EclipseCon

http://www.osgi.org/DevCon2011/HomePage

Luminis Technologies

Training, support and consultancy

http://luminis-technologies.com/