Spring Application Plattform

- das neue Rad?

Papick Garcia Taboada | pgt/adminSight

Kristian Köhler | kkoehler.com

W-JAX 08
Konferenz für Java™, Enterprise Architekturen, SOA

Hello World

Kristian Köhler
Diplom-Wirtschaftsinformatiker (BA)
Freiberuflicher Software Architekt, Berater und Trainer

Papick G. Taboada
Diplom-Wirtschaftsingenieur (TH)
Freiberuflicher Technology Scout

Agenda

Einleitung

Motivation
Modularisierung

Technologisches Umfeld

OSGi
Spring DM

SpringSource dm Server

Überblick und Features
Ausblick

Motivation

Warum ein Application-Server?

Warum OSGi und nicht Spezifikationskonform?

Architektur?

Eine Architektur (von griech. αρχή = Anfang, Ursprung und lat. tectum = Haus, Dach) beschreibt in der Informatik im allgemeinen das **Zusammenspiel der Komponenten** eines komplexen Systems. Der Begriff wird in unterschiedlichen Bereichen angewendet.

Architektur?

Zusammenspiel der Komponenten

Architektur?

Komponenten ?

Divide and Conquer

Kennen wir!

- Component based development
- Layers
- Modularization
- AOP...

Component based development?

Können wir...

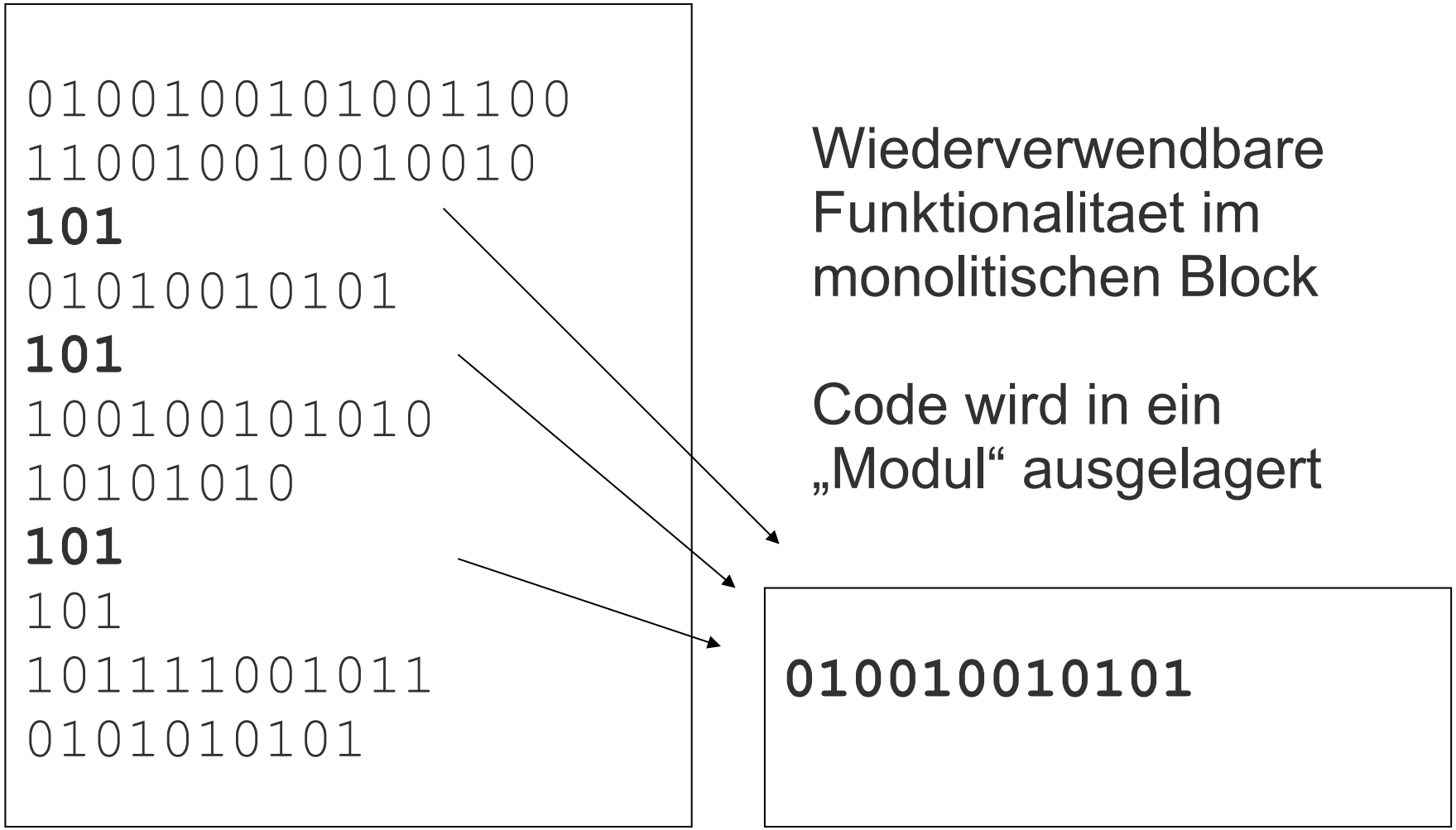
- Spring Beans
- EJB Komponenten
- Servlets
- 1000 verschiedene GUI-Komponentenmodelle
- …

Komponentenbildung...

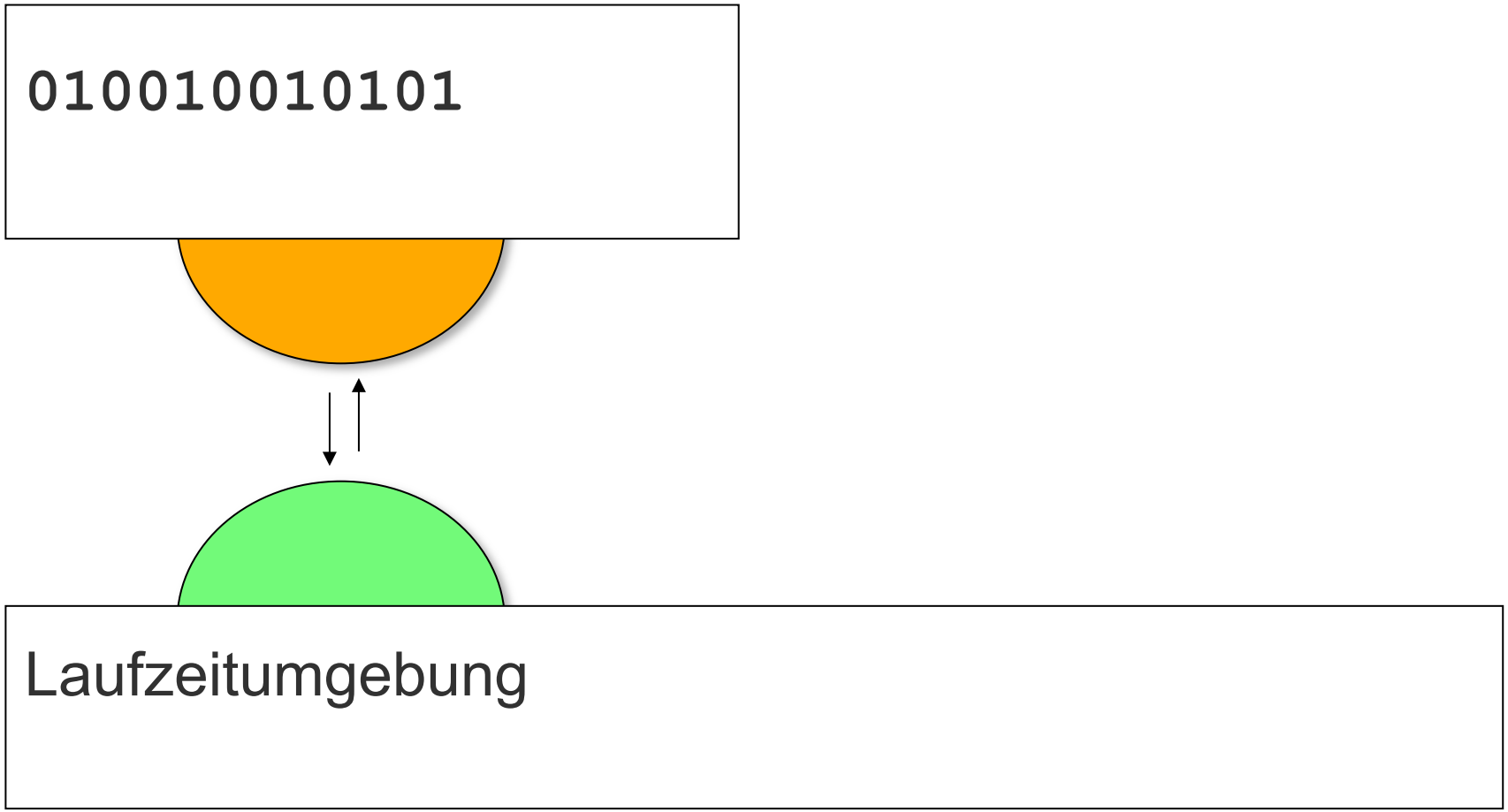
```
0100100101001100
1100100100100010
010010010101
01010010101
010010010101
100100101010
10101010
010010010101
101
101111001011
0101010101
```

Wiederverwendbare Funktionalität im monolitischen Block

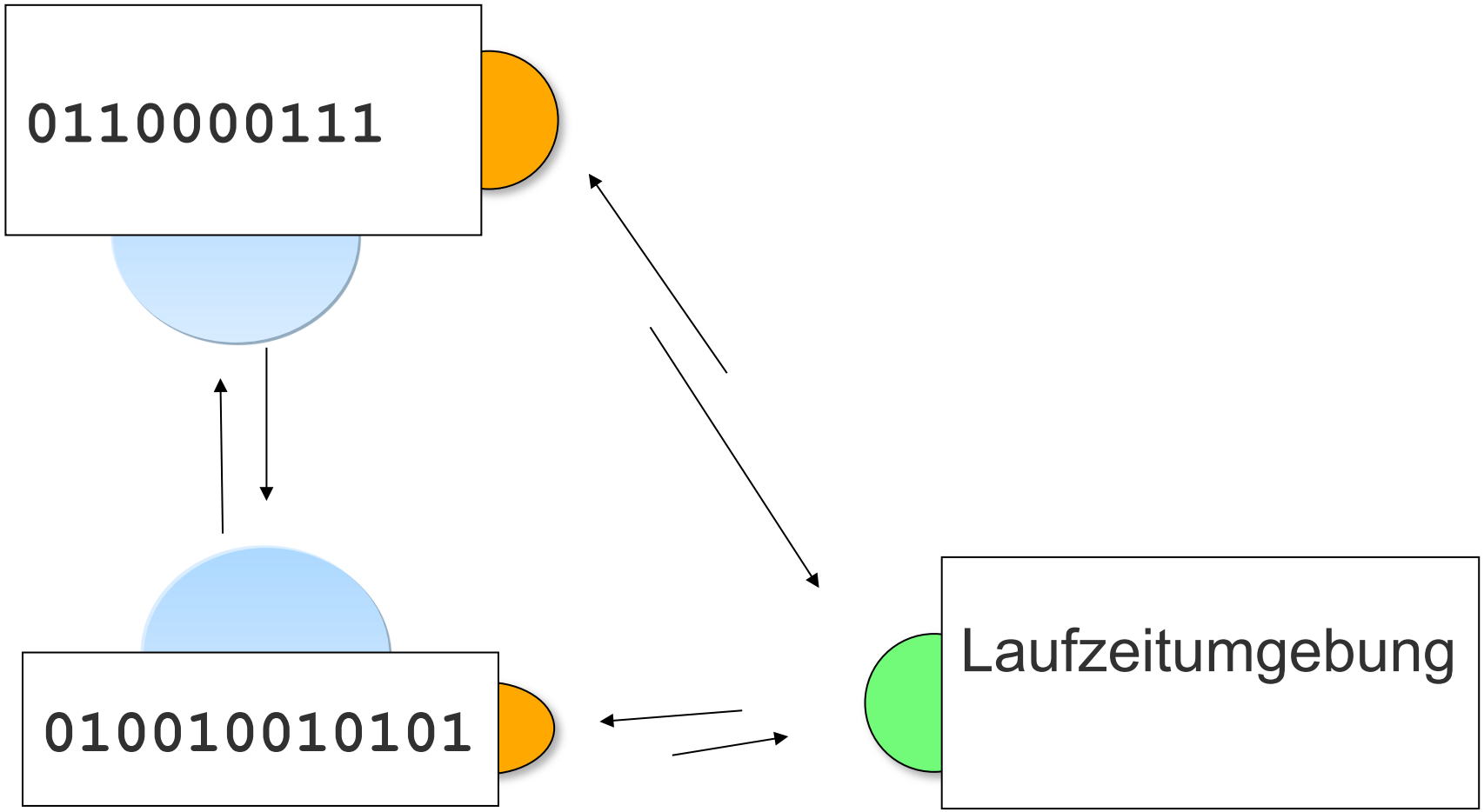
Komponentenbildung...



Laufzeitumgebung



Komponenten im System



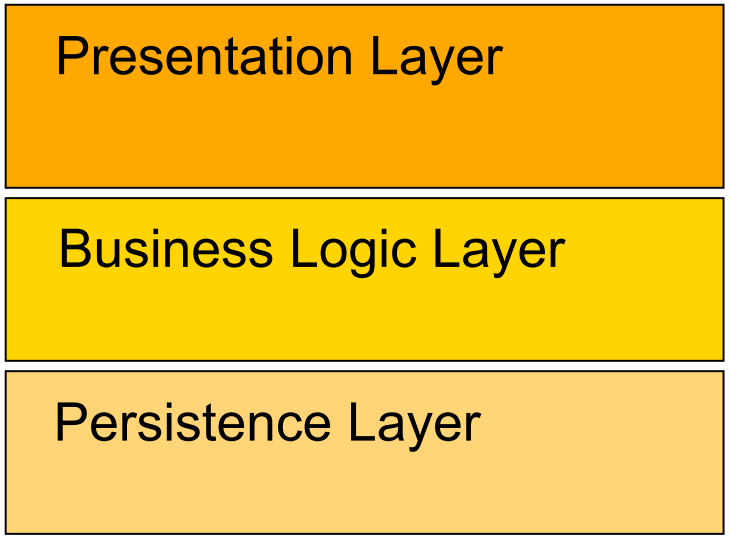
Noch zu definieren…

- Zustand einer Komponente
- Lebenszyklus
- Abhängigkeiten
- Gemeinsame Dienste
- usw.

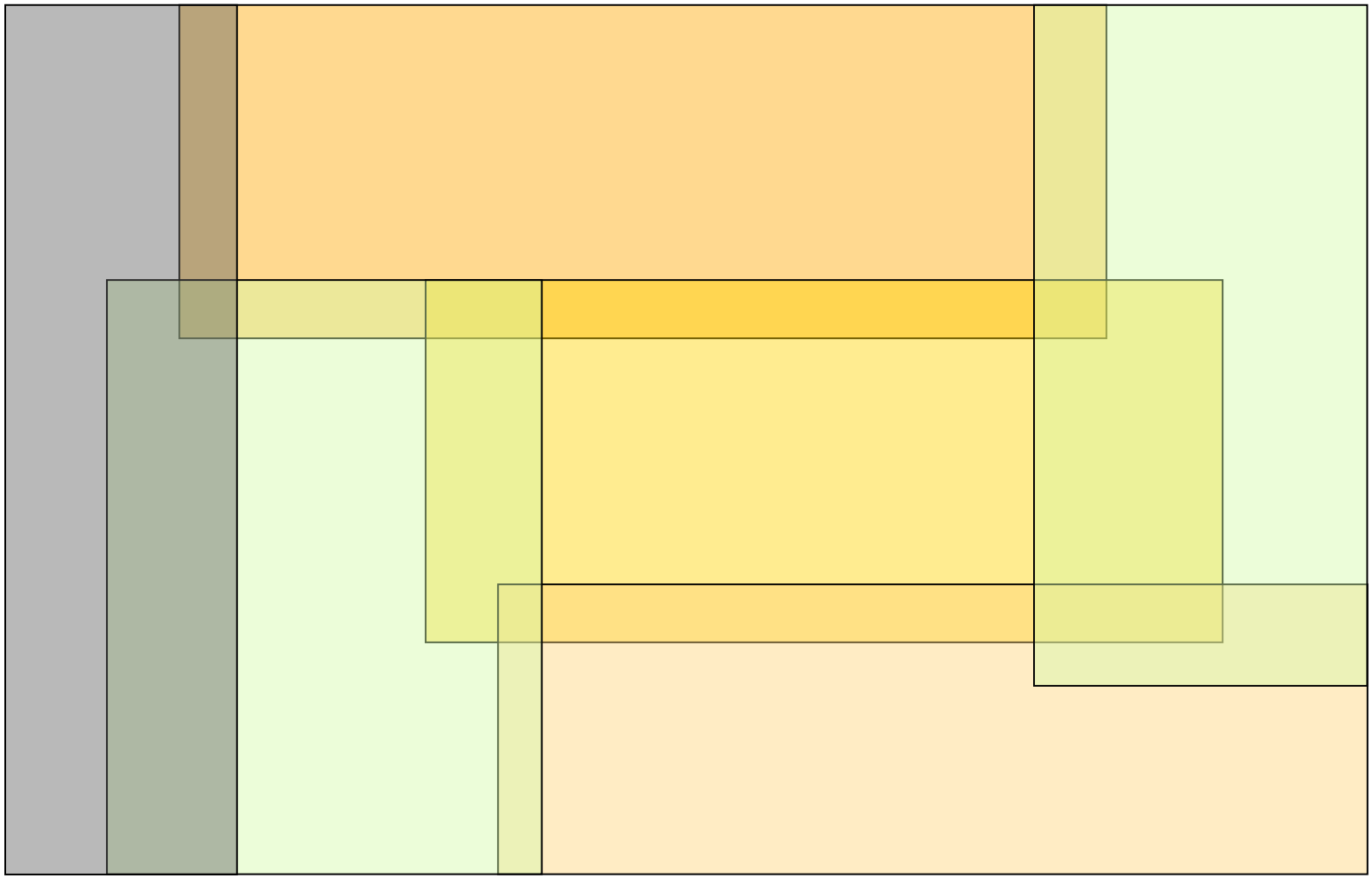
Layers

Können wir...

- EJB Klassisch
  - Business Delegate
  - Service Facade
  - Business Logic
  - Dao
- Spring Klassisch
  - Wie oben, evtl. die eine oder andere Schicht weniger... ;-)



Nur am Rande: „Layers in practice“



AOP – Aspekt Orientierte Programmierung

- Können wir... manchmal...
  - AOP ist ein Konzept
    - Keine Spezifikation
    - Verschiedene Lösungsansätze
      - Dynamic Proxies
      - Bytecode-Manipulation
    - Verschiedene Anbieter
      - Spring AOP
      - EJB Interceptoren
      - AspectJ
      - ...

Aber jetzt: Modularisierung...

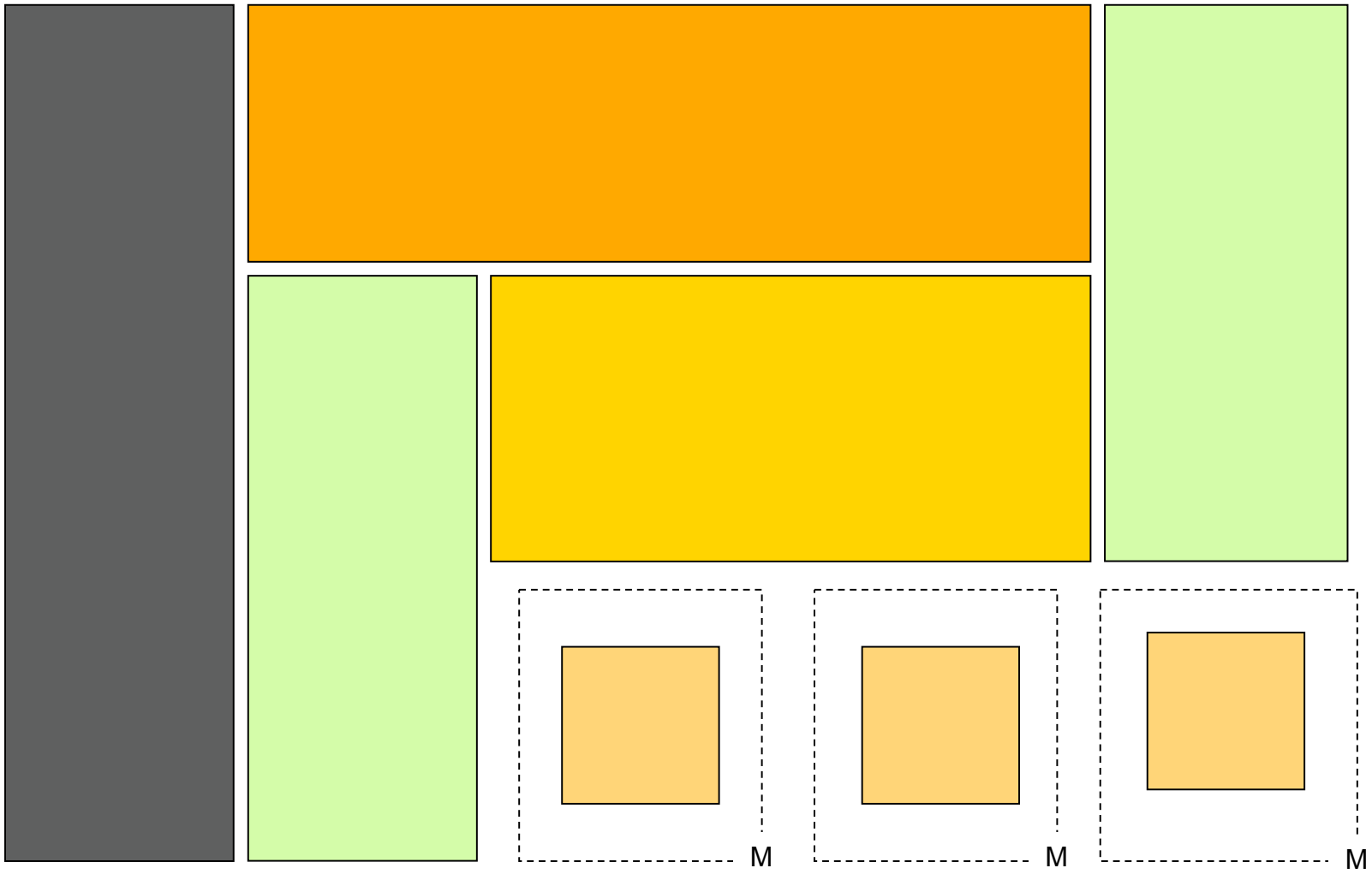
Können wir?

Was ist eigentlich Modularisierung?

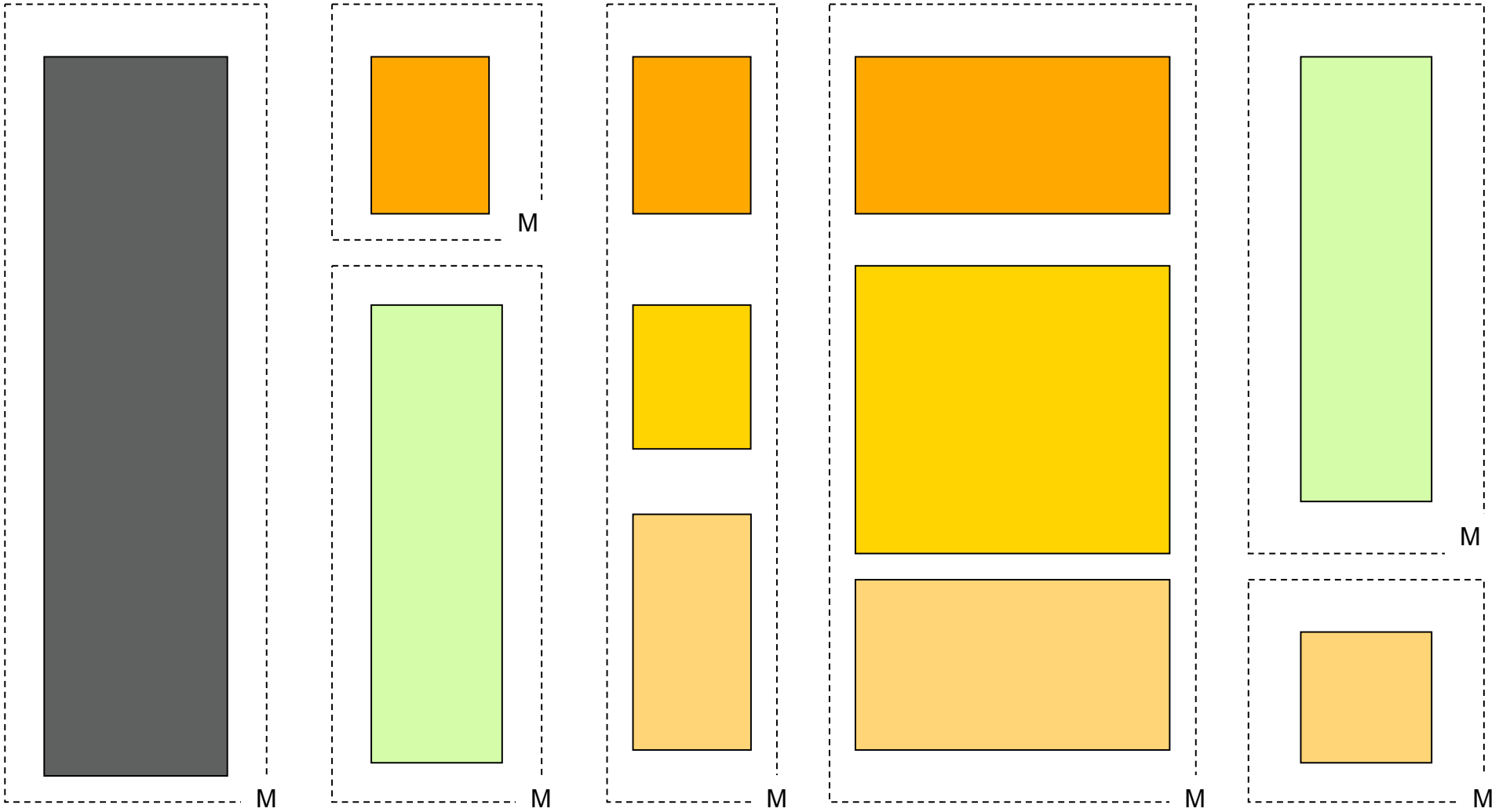
Modularisierung...

"A module is a self-contained component of a system, which has a **well-defined interface to the other components**; something is modular if it includes or uses modules which can be **interchanged as units without disassembly** of the module. Design, manufacture, repair, etc. of the modules may be complex, but this is not relevant; once the module exists, it can easily be connected to or disconnected from the system."

Etwa so?



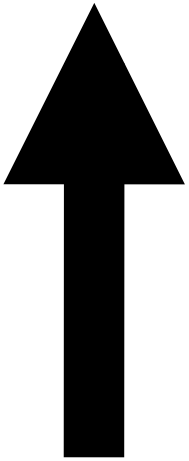
Oder so?



Granularität

„Fine and coarse grained components“

Beispiel aus Architekturdefinition:



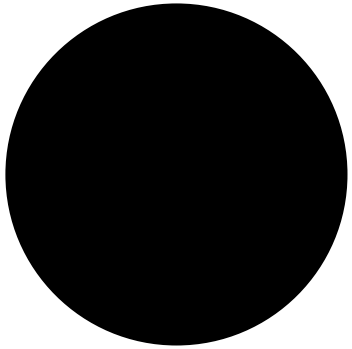
- System
- Subsystem
- Module
  - Smallest composing part
  - Will be to mapped
    - Maven artifacts
    - IDE project

The term „component“ is used by many technologies. To avoid misinterpretation I do not use it here…

Java EE Granularität

„fine grained“ „coarse grained“

HelloWorldServlet



EJB-JAR
&
WAR

Archives

EAR Archive

JAR
„Libraries“

Spezifikationsloch

Modularisierung
zur Compilezeit?

Classloader-Monsters

Wer Wo Was

Compile-Time

- Abhängigkeiten zu Schnittstellen
- Problembereich Build-Infrastrutkur

Runtime

- Abhängigkeiten zu Implementierungen
- Problembereich Laufzeitumgebung

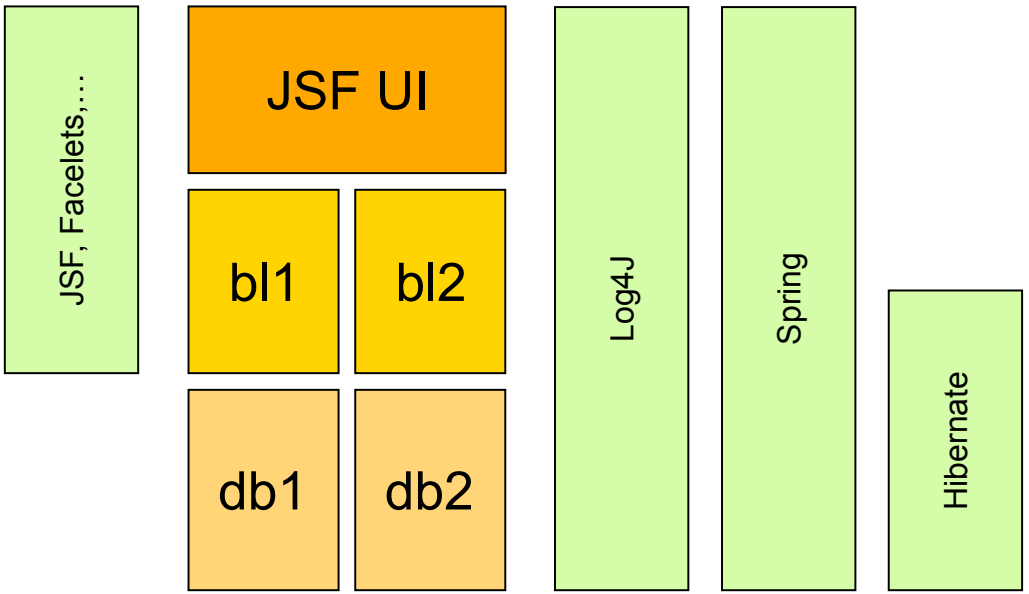
Kleines Experiment

Divide & Conquer applied

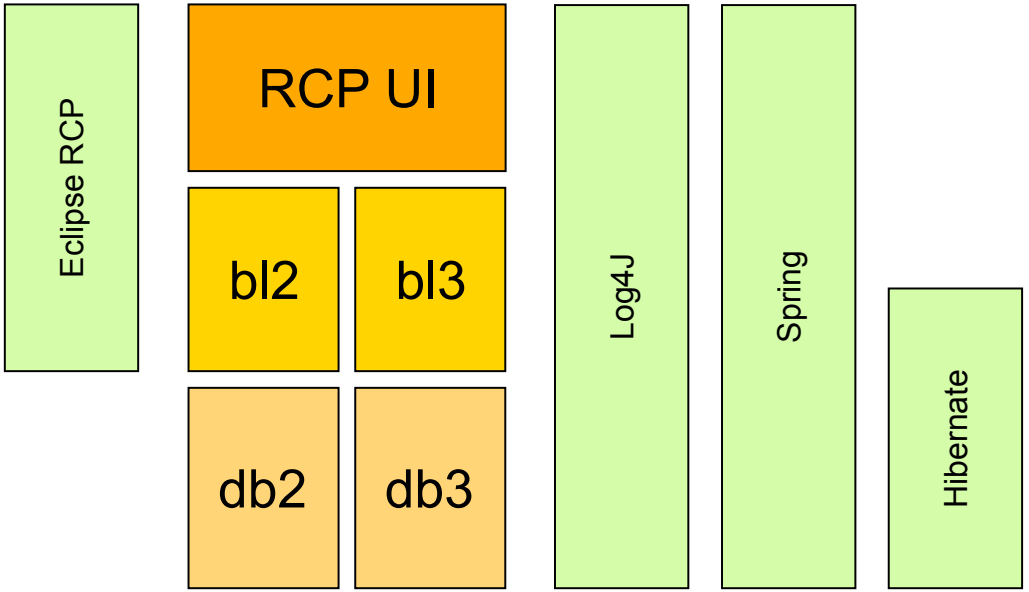
Eine kleine Anwendung

- 1x Web-Client
- 1x Rich-Client
- 1x Gute Laune und Motivation

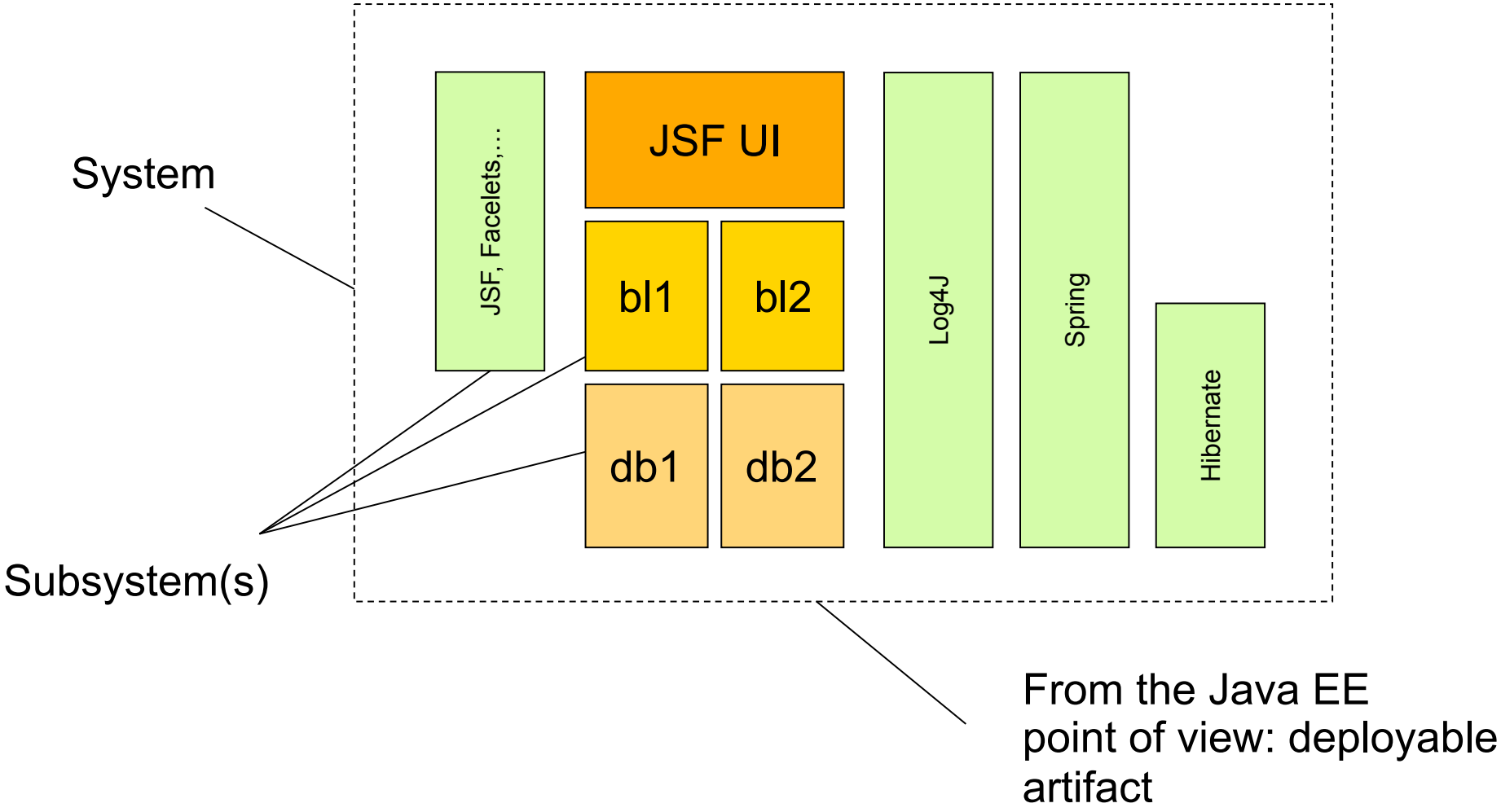
Simple outline – WebApp A



Simple outline – Rich client B



Web A: boundaries and names

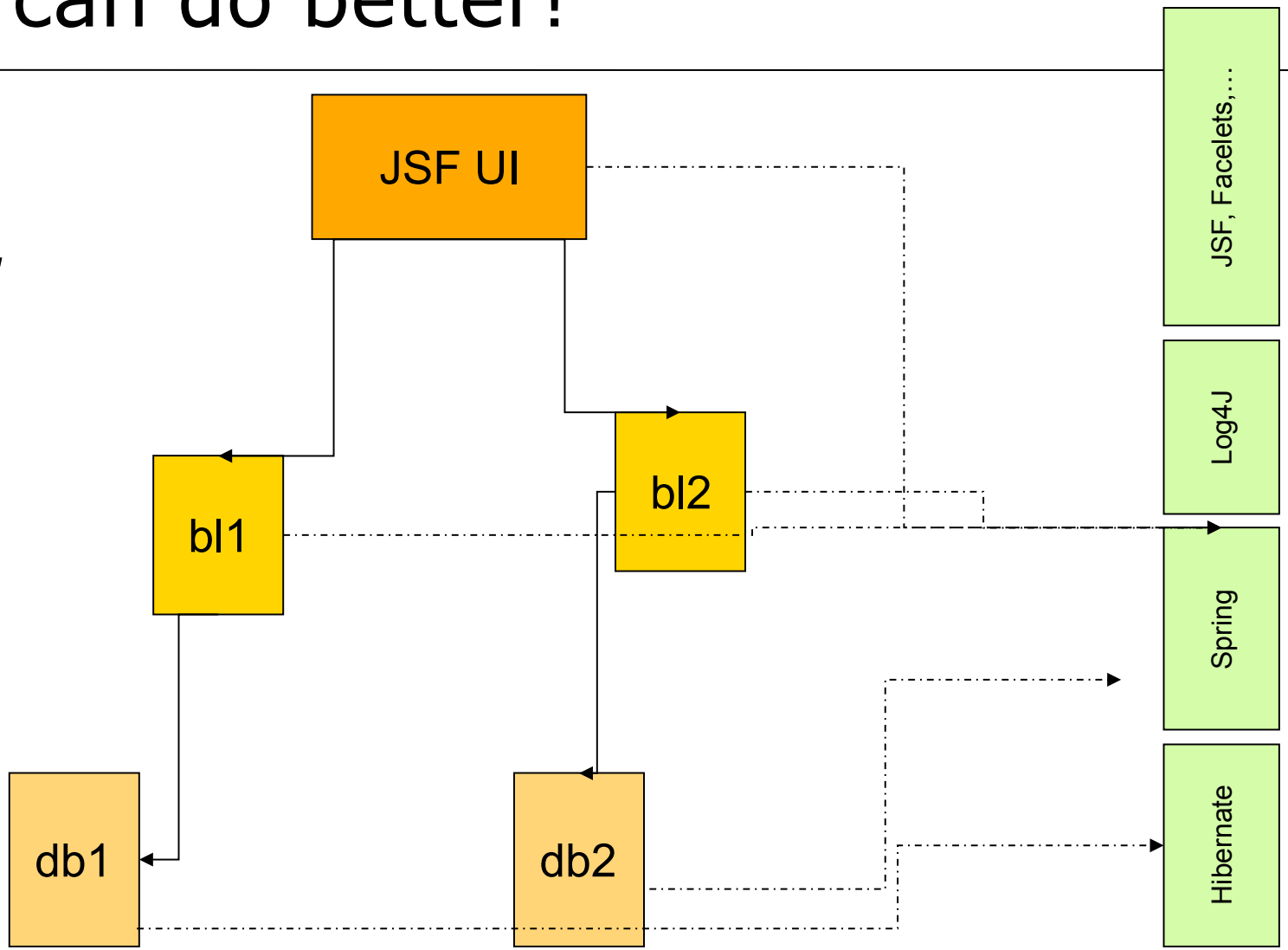


Web A: we can do better!

Dependencies!

Between modules

Between modules and required technologies



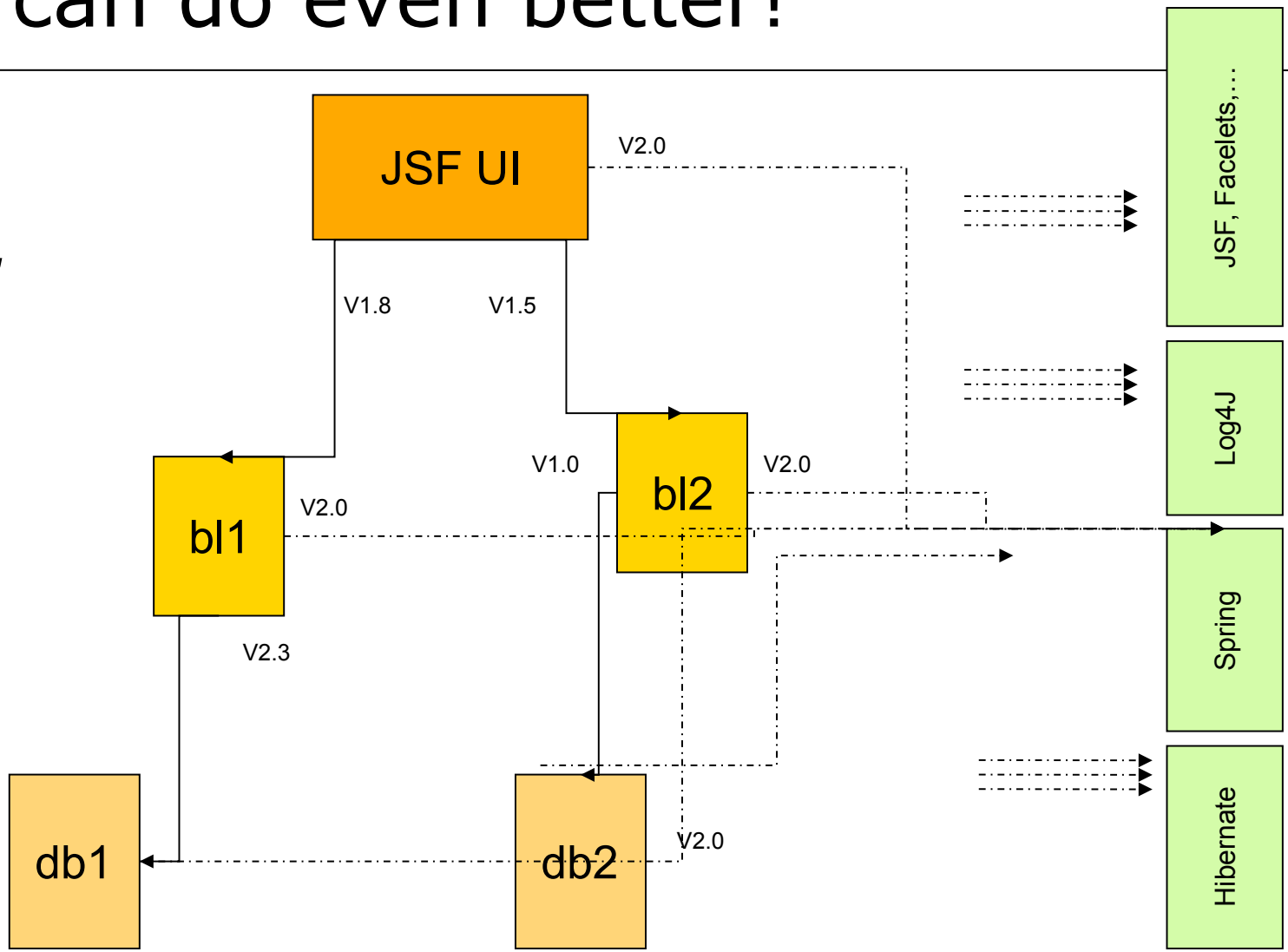
Web A: we can do even better!

Dependencies!

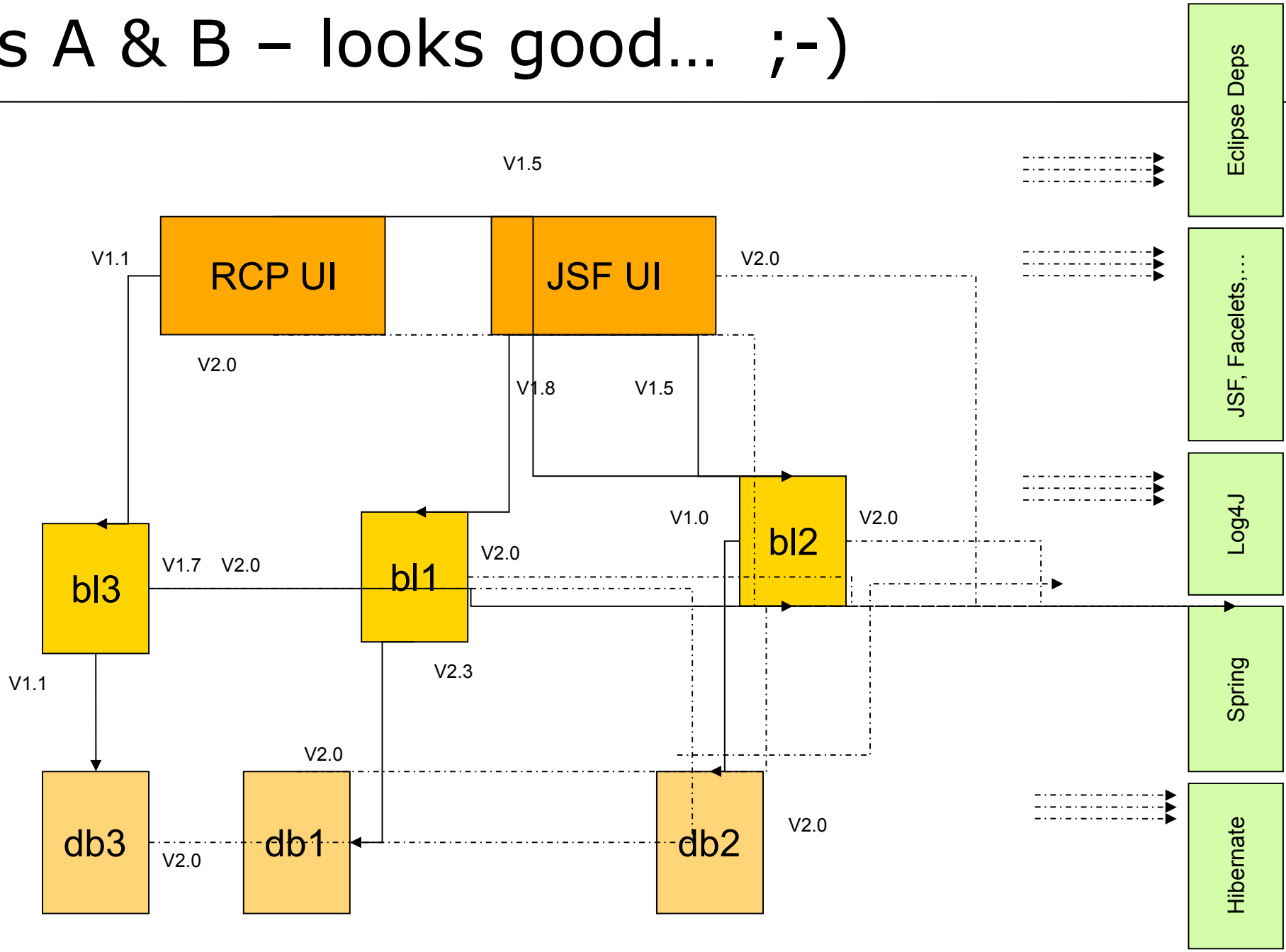
Between modules

Between modules and required technologies

Versions!!!



Apps A & B – looks good… ;-)



Probleme

- Modularisierung
  - Größer Komponenten
  - Kleiner als Deployment Artefakt
- Abhängigkeiten zwischen Module
  - Definieren
  - Compiletime
  - Laufzeitverhalten
- Versionierung

Die Lösungsansätze

Wir lösen es in der Build-Infrastruktur
und haben zur Laufzeit ein Monolith...

Schonmal mit Maven, Ivy & Co. gelöst.

oder...

OSGi

Das Problem mit den Lösungen

OSGi und Java EE Paketierung

Passt nicht zusammen...

Lösung über Buildinfrastrutkur liefert keine Antworten auf Fragen in der Laufzeitumgebung

Also entweder oder...

- OSGi
  - Module
  - Abhängigkeiten
  - Versionen
  - Kein Server
  - Keine Paketierung
- Java EE
  - Spezifikation...
  - Lösung nur „beim Bauen“

Agenda

Einleitung

Motivation
Modularisierung

Technologisches Umfeld

OSGi
Spring DM

SpringSource dm Server

Überblick und Features
Ausblick

OSGi
Open Services Gateway initiative (I)

- Kein JavaEE Standard
  - Definiert von OSGi Alliance
    - Mitglieder z. B. IBM, Nokia und Oracle
    - http://www.osgi.org
- Komponentenmodell
  - Anwendung wird in „bundles“ unterteilt
  - Bundles können einzeln verwaltet werden
    - install/start/stop/uninstall/update

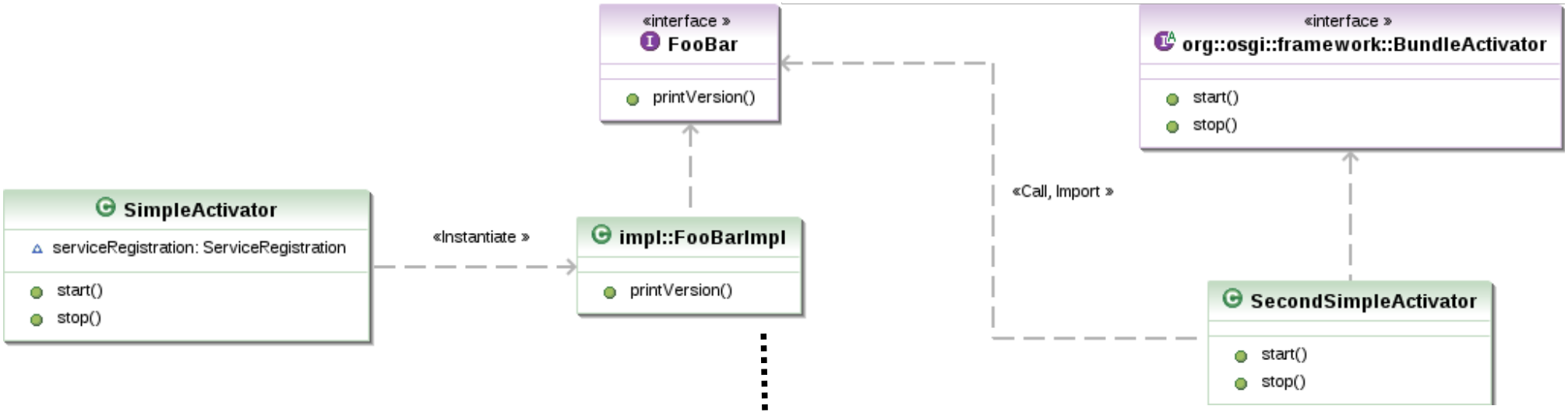
OSGi
Open Services Gateway initiative (II)

- Sichtbarkeitsregeln für Bundles
  - Definition von Im- und Exports
    - Eintrag in META-INF/MANIFEST
- Abhängigkeitsverwaltung zwischen Bundles
  - Auflösung von Abhängigkeiten zur Laufzeit
  - Versionsinformationen vorhanden
- „SOA in a JVM“
  - Service Registry verwaltet Dienste

OSGi Service Platform

- Ausführungsplatform für OSGi Bundles
  - Lightweight Container
  - Containerdienste
  - Lifecyle für Bundles
  - Standalone oder Embedded einsetzbar
- OSS Implementierungen
  - Equinox (http://www.eclipse.org/equinox/)
  - Apache Felix (http://felix.apache.org/)
  - Knopflerfish (http://knopflerfish.org/)

Einfaches OSGi Beispiel (I)



META-INF/MANIFEST

Manifest-Version: 1.0
Export-Package: com.kkoehler.wjax08.bundle1;version="0.1"
;uses:="org.osgi.framework"
Private-Package: com.kkoehler.wjax08.bundle1.impl
Ignore-Package: com.kkoehler.wjax08.bundle1.impl
Built-By: kkoehler
Tool: Bnd-0.0.238
Bundle-Name: Sample OSGi Bundle 1 - v0.1
Created-By: Apache Maven Bundle Plugin
Build-Jdk: 1.6.0_10
Bundle-Version: 0.1
Bnd-LastModified: 1225437612220
Bundle-ManifestVersion: 2
Bundle-Activator: com.kkoehler.wjax08.bundle1.SimpleActivator
Import-Package: com.kkoehler.wjax08.bundle1;version="0.1",org.osgi.fra
mework;version="1.3"
Bundle-SymbolicName: com.kkoehler.wjax08.my-bundle1

META-INF/MANIFEST

Manifest-Version: 1.0
Export-Package: com.kkoehler.wjax08.bundle2;uses:="com.kkoehler.wjax08
.bundle1,org.osgi.framework"
Built-By: kkoehler
Tool: Bnd-0.0.238
Bundle-Name: Sample OSGi Bundle 2
Created-By: Apache Maven Bundle Plugin
Bundle-Version: 0.1
Build-Jdk: 1.6.0_10
Bnd-LastModified: 1225437523386
Bundle-ManifestVersion: 2
Bundle-Activator: com.kkoehler.wjax08.bundle2.SecondSimpleActivator
Import-Package:
com.kkoehler.wjax08.bundle1;version="0.1",com.kkoehler
.wjax08.bundle2,org.osgi.framework;version="1.3"
Bundle-SymbolicName: com.kkoehler.wjax08.my-bundle2

Einfaches OSGi Beispiel (II) - Registry

```
package com.kkoehler.wjax08;

import org.osgi.framework.*;

public class SimpleActivator  implements BundleActivator
{
  ServiceRegistration registration;

  public void start(BundleContext bundleContext)
  {
        registration = bundleContext.registerService(
                    FooBar.class.getName(),
                    new FooBarImpl(), null);
  }
  public void stop( BundleContext bc )
  {
        registration.unregister();
        registration = null;
  }
}
```

Für Interaktion mit OSGI Platform Abhängigkeit zu OSGI Klassen.

Anmelden, Suchen, Nutzen von Diensten sind wir mittlerweile anders gewohnt ;-)

Infrastruktur-Code

Überlegungen zu „raw OSGi“

- Abhängigkeit zu OSGi Framework
  - siehe Activator
- Referenzen zwischen Bundles „umständlich“
  - Anmelden/ Abmelden programmatisch
- Bundles müssen konfiguriert werden
  - Anwendungskonfiguration
- Enterprise Features
  - z. B. Transaktionen (kommt mit OSGi 4.2)
- Unit und Integration Testing

Spring Dynamic Modules

Ehemals Spring-OSGi

Keine OSGi Service Platform

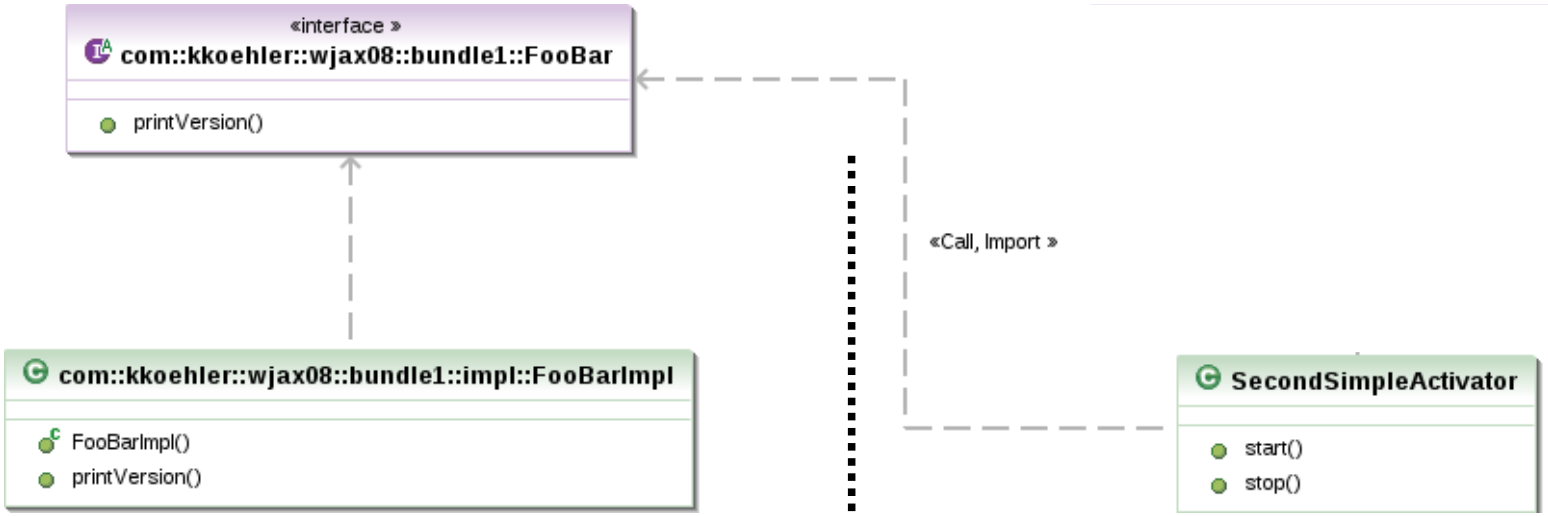
Brücke zwischen Spring und OSGi

- Nicht Framework bezogen

Spring Dynamic Modules

- Spring Konzepte in OSGi Welt bringen
  - Programmiermodell (POJO)
  - Dependency Injection
  - AOP
- OSGi Ideen für Spring Anwendungen nutzbar machen
  - Modularisierung
  - Dependency Management (Sichtbarkeit, Versionierung, etc)

Einfaches Spring dm Beispiel (I)



META-INF/spring/context.xml

```
...
<bean id="foo"
      class="....impl.FooBarImpl"/>

<osgi:service id="fooBar"
      ref="foo"
      interface="...FooBar"/>

...
```

META-INF/spring/context.xml

```
...
<bean id="activator"
   class="....SecondSimpleActivator"
   init-method="init">
   <property name="fooBar">
        <ref bean="fooOsgi"/>
   </property>
</bean>

<osgi:reference id="fooOsgi"
      interface="....FooBar"/>
...
```

Überlegungen zu Spring dm

- Kein Anschluss an JavaEE Komponentenmodelle (z.B. Web-Apps)
- Abhängigkeitsverwaltung
  - Bundles müssen manuell installiert werden
  - Ein Bundle kommt selten allein...
- Kein Konzept für „Anwendung“
  - Deployment
  - Sichtbarkeit/ Grenzen
  - Logging

Java EE Granularität

„coarse grained“

„fine grained“

HelloWorldServlet

EJB-JAR
&
WAR

Archives

EAR Archive

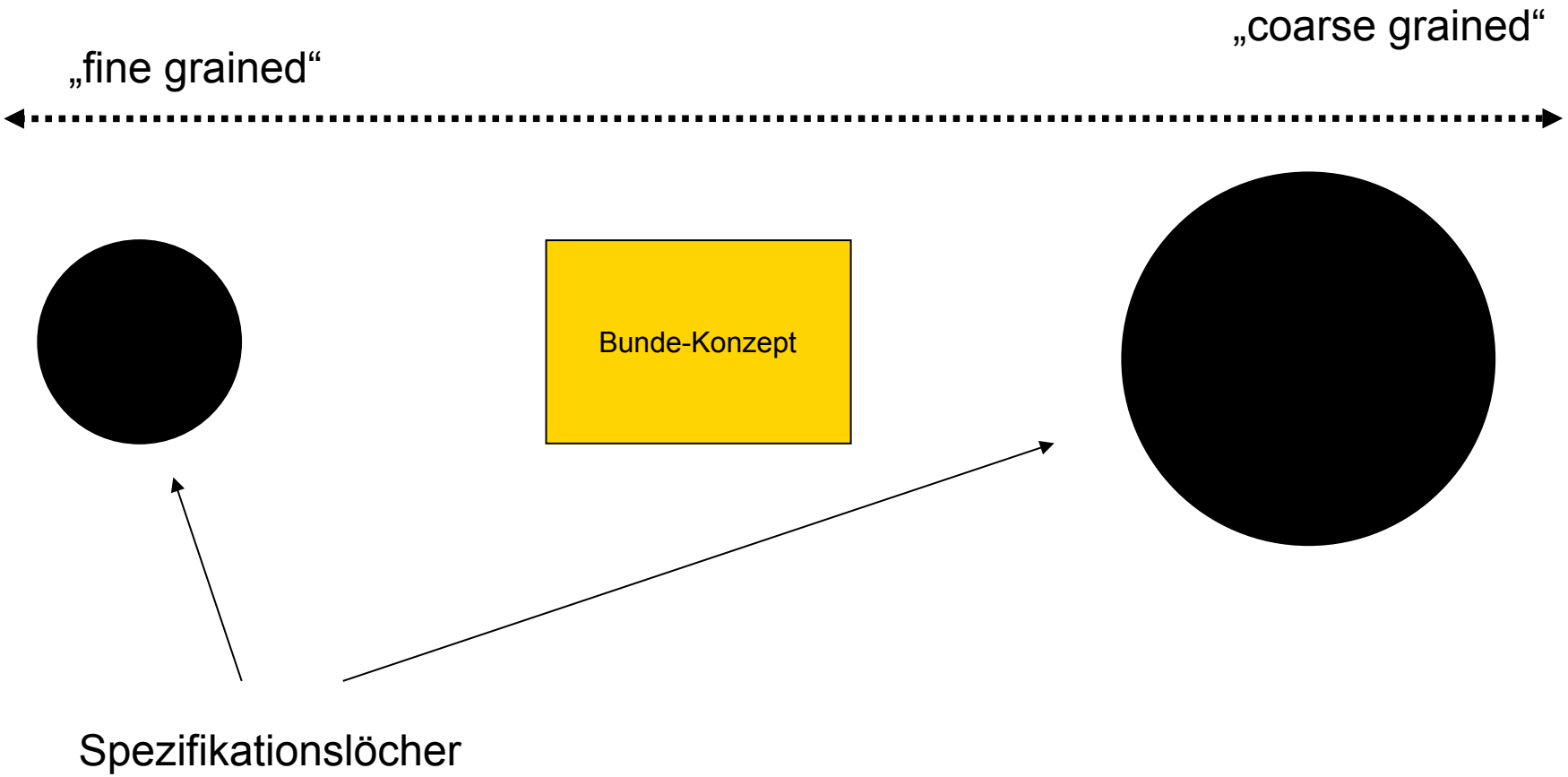
JAR
„Libraries“

Spezifikationsloch

Modularisierung
zur Compilezeit?

Classloader-Monsters

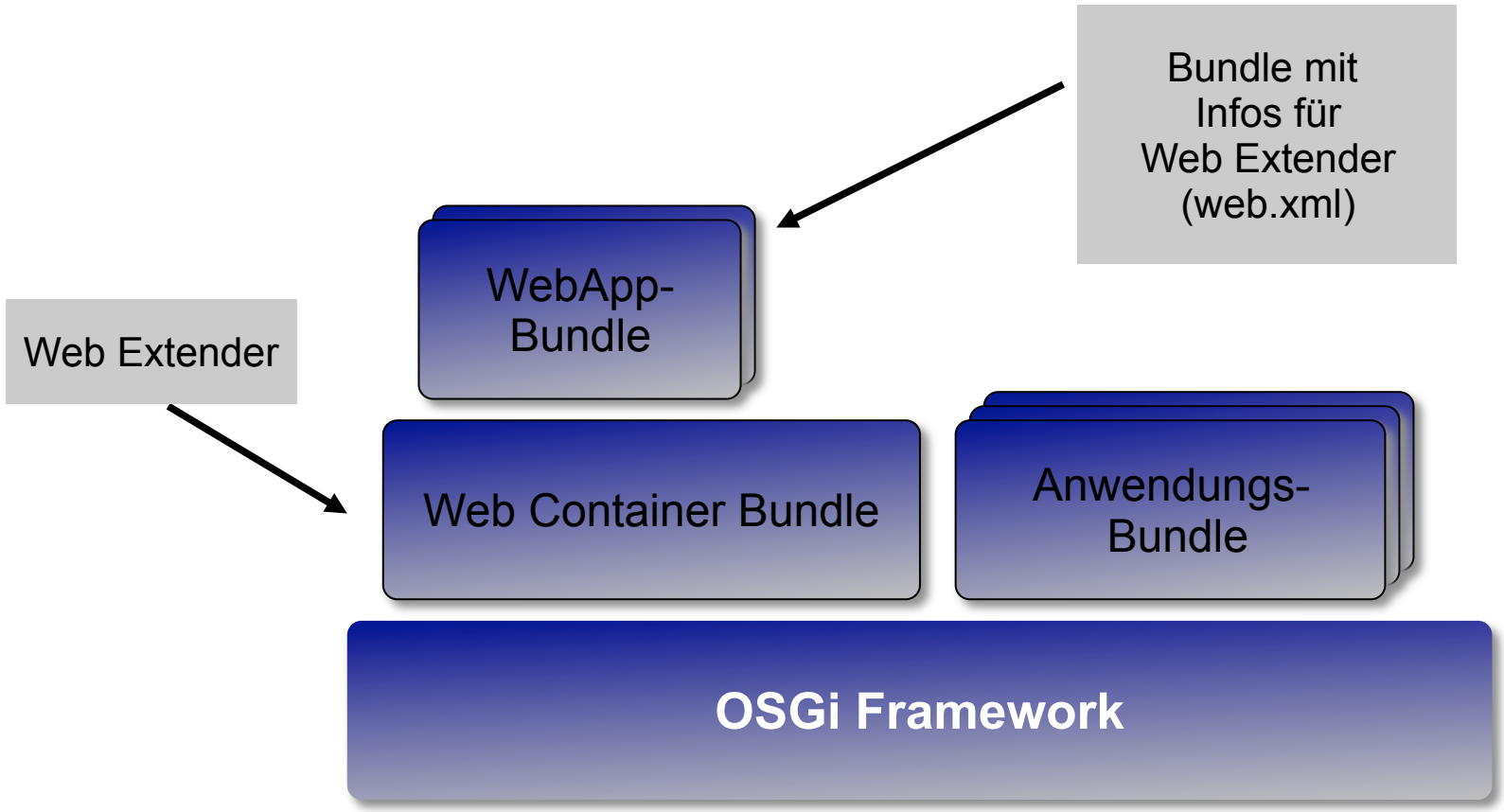
Gleiches Problem, aber „inside-out“



Extender Pattern

- „Inversion of control“ bei Bundle-Installation
  - Ausnutzen des OSGi Lifecycles
    - Synchronous Bundle Listener
  - Installiertes Bundle wird untersucht
    - z. B. Dateien oder Manifest Eintrag
- Weniger Verantwortung bei „neuem“ Bundle
  - Kein „Boilerplate Code“ in jedem Bundle
  - Fehlerresistenter

Extender Pattern bei OSGi



Agenda

Einleitung

Motivation
Modularisierung

Technologisches Umfeld

OSGi
Spring DM

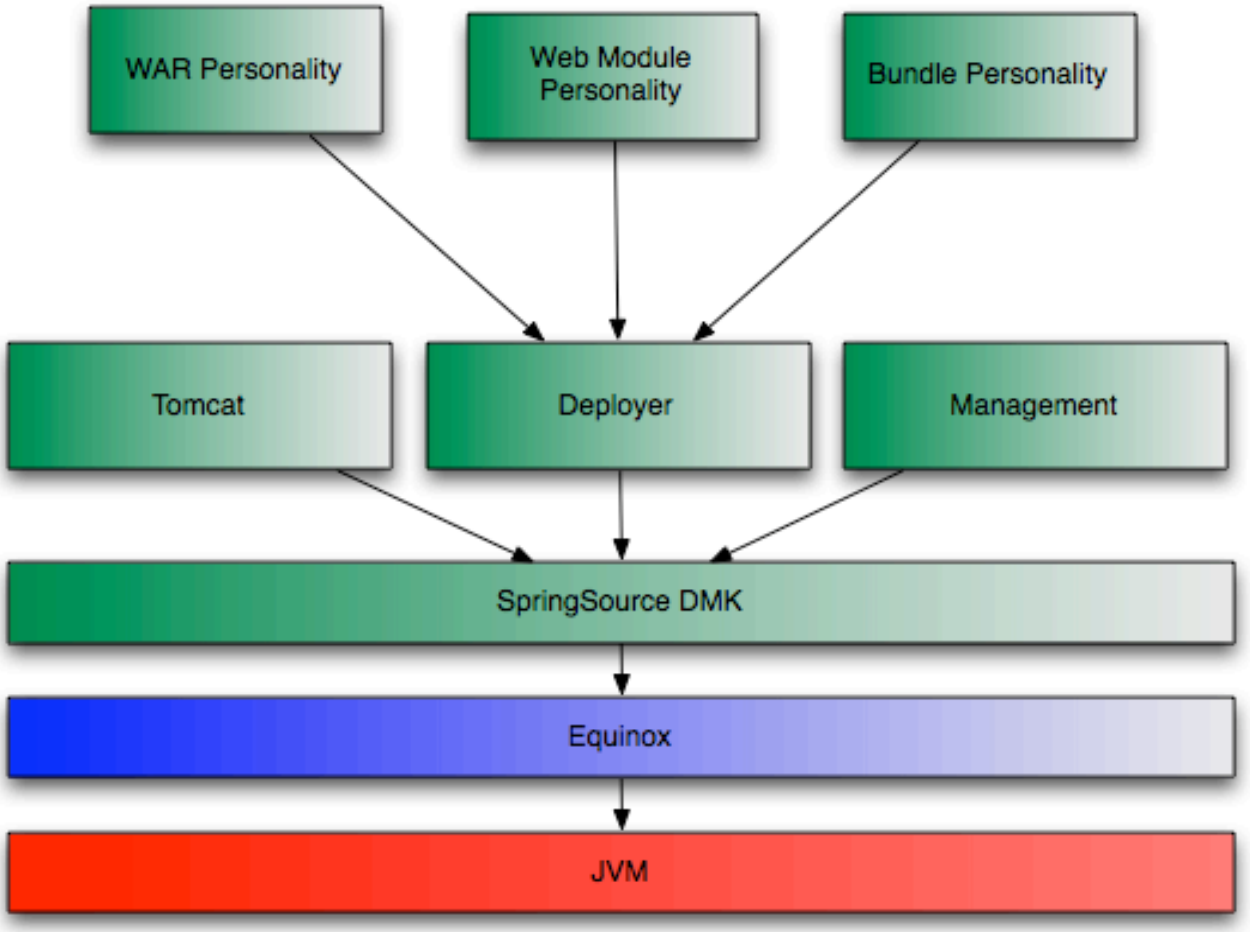
SpringSource dm Server

Überblick und Features
Ausblick

SpringSource dm Server

- OSGi basierter Java Application Server
  - (K)ein Java EE Standard konformer Server ;-)
    - WAR (Web-Profil Java EE 6)
  - OSGi wird nicht nur intern eingesetzt
    - Eigene Anwendungen können OSGi nutzen
    - OSGi Deployment möglich
- Wird von SpringSource entwickelt
  - Community und Enterprise Version
    - Community GPL lizensiert

Spring Application Plattform



DMK = Dynamic Module Kernel

Quelle: http://blog.springsource.com/main/wp-content/uploads/2008/04/architecture.png

SpringSource dm Server - Features

Abhängigkeitsmanagement

Deployment

Runtime Provisioning/ Repository

Logging

Clustering-Unterstützung

...

„Neues“ Abhängigkeitsmanagement

- Import-Bundle
  - ClassLoader Problem bei RequireBundle
    - Reihenfolge beim Laden eventuell problematisch
    - „Split packages“
- Import-Library
  - Referenziert alle exportierten Packages einer Gruppe von Bundles
- Werden intern zu Import-Package
  - OSGi Standardverhalten

Deployment/ Personalities

- Web Applications
  - Standard JavaEE WARs
  - Shared Libraries WAR
  - Shared Services WAR
  - Plattform spezifische Web Modules
- RAW OSGI Bundles
- PAR - Anwendungsarchiv
  - Proprietäres Format (bevorzugt)

Shared Libraries WAR

- Standard WAR ohne Bibliotheken
  - Kein unnötig großes Web-Archiv
    - WEB-INF/lib ist leer
- Abhängigkeiten werden über OSGi importiert
  - Import-Package
  - Require-Bundle

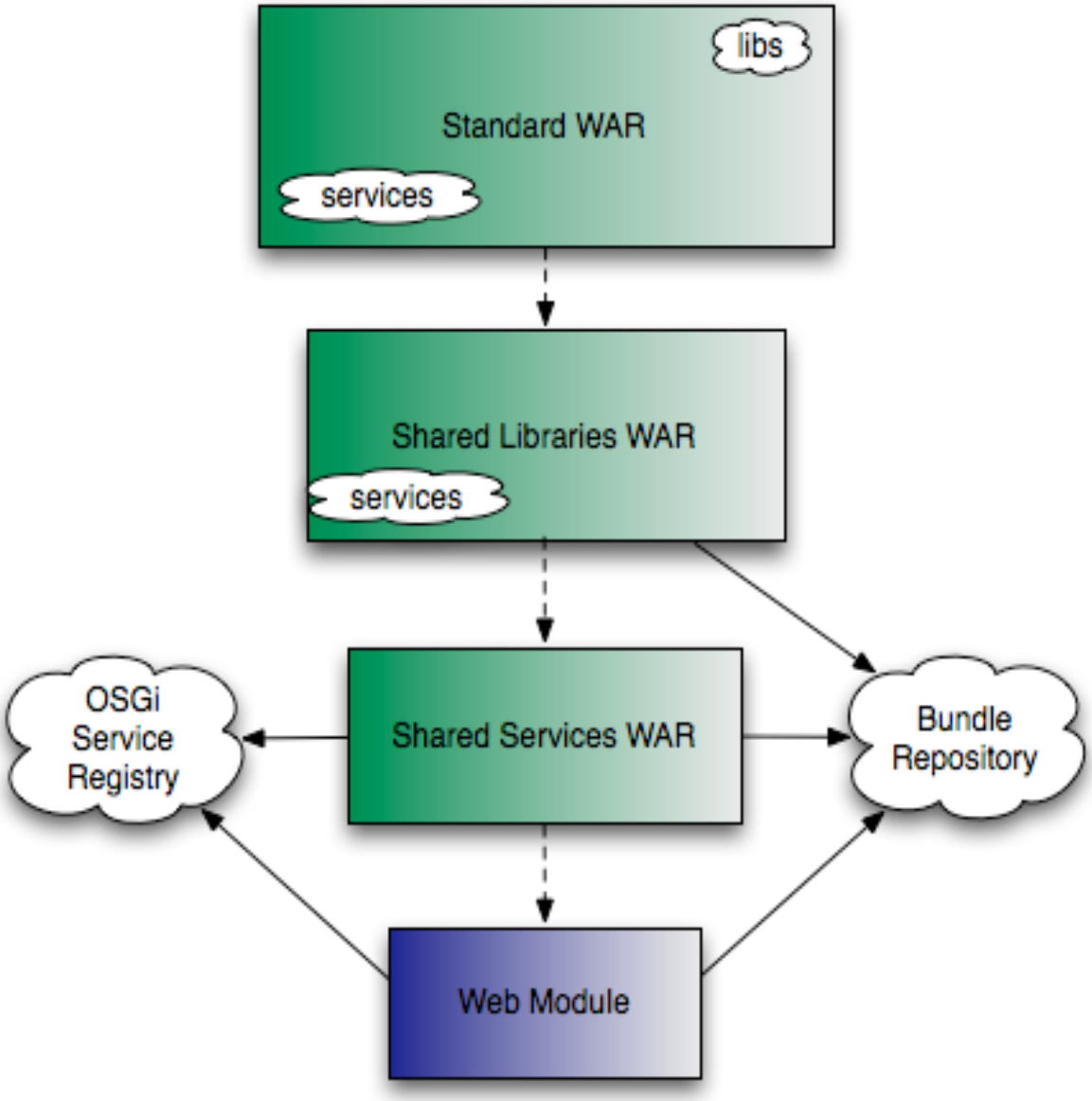
Shared Services WAR

- Standard WAR ohne Bibliotheken + Services
  - WEB-INF/lib ist leer
  - Keine Services in Archiv
- Services in „externen“ Bundles
  - <osgi:reference.../>

Plattform spezifische Web Modules

- Ähnliche Struktur wie „Shared Services WAR“
- Vereinfachte Spring MVC Konfiguration
  - Einträge in MANIFEST.MF
    - Web-DispatcherServletUrlPatterns
    - Web-ContextPath
    - …
  - web.xml wird zur Laufzeit erzeugt
    - Merging mit Fragementen

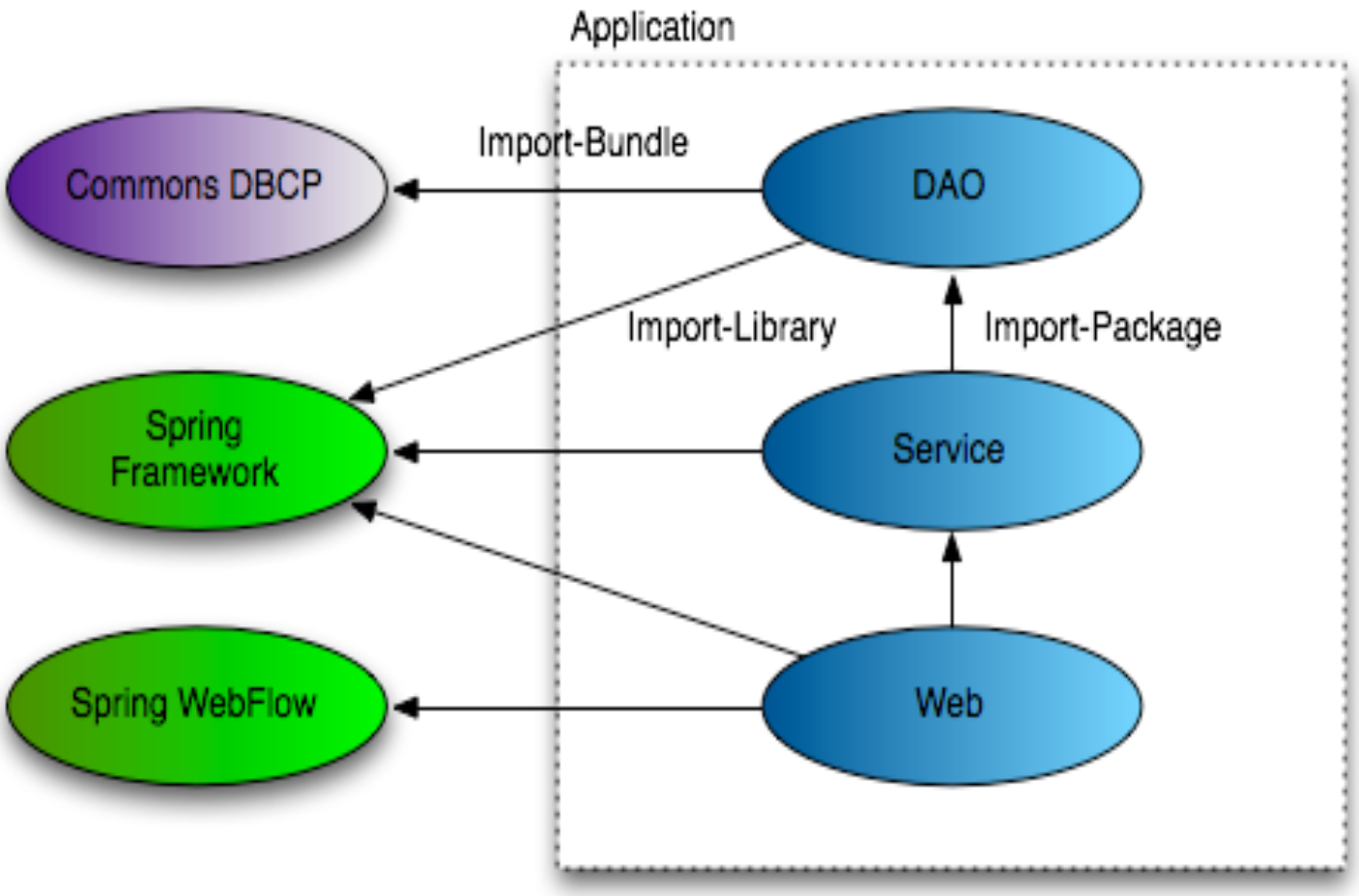
Migration von WAR to Web Module



Platform Archives (PAR)

- Wird als „Ersetzung“ eines EAR gesehen
- Enthält alle Bundles einer Anwendung
  - Einfachere Insallation
  - Teile der Anwendung können unabhängig voneinander entwickelt/verwaltet werden
- Anwendungen werden abgegrenzt
  - Classloading, AOP load-time weaving, etc
- Einfacheres Management
  - „Was gehört zusammen“

Typische Anwendung



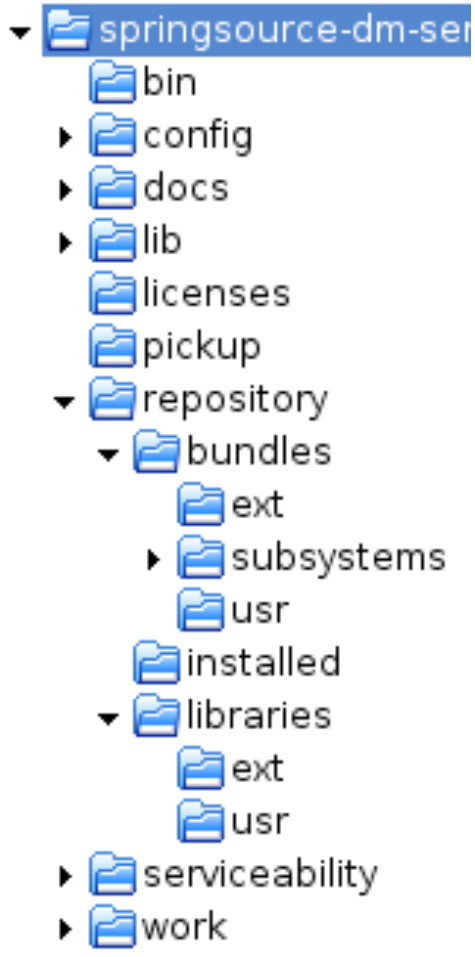
Quelle: http://blog.springsource.com/main/wp-content/uploads/2008/04/par-structure.png

Runtime provisioning und Repository (I)

- Server Repository hält Bibliotheken vor
  - Bibliotheken als OSGi Bundles
- Nicht aufgelöste Abhängigkeiten werden zur Laufzeit nachgeladen („as-needed“)
  - Memory Footprint
  - Kleinere Anwendungsarchive

Runtime provisioning und Repository (II)

- Hinzufügen von Bibliotheken zur Laufzeit mgl.
  - Kopieren in Verzeichnis
  - Eclipse Tooling
- Online Repository für Bibliotheken
  - http://www.springsource.com/repository
  - Kein automatischer Download!
- Konfiguration Repo über JSON-File
  - z. B. für Multi-Instance-Usage



Beispiel

Zeit für Beispiel?

Server directory layout...

Server Starten...

PAR Deployment...

Personalities Roadmap...

„In the 1.0 Platform release we support the web and bundle personalities, which enable you to build sophisticated web applications. Future releases will include support for more personalities (…)

„The 2.0 release will introduce additional personalities to cover batch, web services and SOA applications.“

http://blog.springsource.com/2008/04/30/introducing-the-springsource-application-platform/

Serviceability/ Logging

Ziel: Einfacher Betrieb

- Leicht zu lesende Logs ;-)
- Trennung zwischen Trace, Log und Dump

Anwendungslogs werden separiert

- Spezielle Verzeichnisse pro App
- Konfiguration mittels MANIFEST
  - Application-TraceLevels: *=info

Product/ Standards Roadmap...

- Version 1.2
  - Administration configuration
  - Feature complete
- Version 2.0
  - OSGi Blueprint Service (RFC 124)
    - Spring DM als RI für OSGi
    - Spring DM Server als Plattform für OSGi Blueprint

Standard? OSGi 4.2 (early draft)

RFC 120: Security Enhancements

RFC 121: Bundle Tracker

RFC 125: Bundle License

RFC 126: Service Registry Hooks

RFC 128: Accessing Exit Values from Applications

RFC 129: Initial Provisioning Update

RFC 132: Command Line Interface and Launching

RFC 134: Declarative Services Update

RFC 98: Transactions in OSGi

RFC 119: Distributed OSGi

RFC 124: A Component Model for OSGi

RFC 124: A Component Model for OSGi

The OSGi platform provides an attractive foundation for building enterprise applications. However it **lacks a rich component model for declaring components within a bundle** and for instantiating, configuring, assembling and decorating such components when a bundle is started. This RFC describes a set of core features required in an enterprise programming model and that are widely used outside of OSGi today when building enterprise (Java) applications. **These features need to be provided on the OSGi platform for it to become a viable solution for the deployment of enterprise applications**.

SPRING DM

lacks a rich component model for declaring components within a bundle

These features need to be provided on the OSGi platform for it to become a viable solution for the deployment of enterprise applications.

May we live in interesting times...

- OOP 2001 Vergleich zwischen Corba Component Model (CCM) und EJB
  - CCM? Was ist das?
- Java EE Spezifikation muss sich bewegen
  - Reichen die angepeilten Profile?
  - OSGi 4.2: ernstzunehmende Konkurrenz?
  - Krachts nicht spätestens mit
    - Java Superpackages und
    - Java Dynamic Modules?

Vielen Dank für Ihre Aufmerksamkeit

Fragen?

Meet...

@ OIO Stand 1. Etage
@ W-Jax Ballroom

Stand1. Etage

Kristian Köhler
Diplom-Wirtschaftsinformatiker (BA)
Freiberuflicher Software Architekt,
Berater und Trainer
http://www.kkoehler.com

Papick G. Taboada
Diplom-Wirtschaftsingenieur (TH)
Freiberuflicher Technology Scout
http://pgt.de