OSGi DevCon 2014

Application Isolation

Is there an alternative to Subsystems?

Tim Diekmann
Principal Architect
TIBCO Software Inc.

Disclaimer

Any of the TIBCO ActiveMatrix BusinessWorks materials presented here is subject to change without notice. This document is TIBCO proprietary information.

This document (including, without limitation, any product roadmap or statement of direction data) illustrates the planned testing, release and availability dates for TIBCO products and services. This document is provided for informational purposes only and its contents are subject to change without notice.

TIBCO makes no warranties, express or implied, in or relating to this document or any information in it, including, without limitation, that this document, or any information in it, is error-free or meets any conditions of merchantability or fitness for a particular purpose.

Overview

- why multiple applications
- brief history of approaches
- OSGi solution
- why another solution
- Alternative OSGi solution

Why multiple applications?

… in a single JVM that is

- save memory (objects and classes)
- reduce management and monitoring overhead
- save CPU on startup and load of applications
- *multiple applications vs multi-tenancy*

very brief History

- web resources
- web applications
- JEE application servers
- application servers like Apache Karaf, Eclipse Virgo, Paremus Service Fabric, IBM WebSphere, etc.
- *what are applications?*

Application Definition

*"**Application software** is all the computer software that causes a computer to perform useful tasks beyond the running of the computer itself."*

Wikipedia - http://en.wikipedia.org/wiki/Application_software

my Application Definition*

* in the context of this presentation, Java, and OSGi

- packaged software deployed to an app server:
  - versioned unit of release/maintenance
  - providing coherent business function
  - with a common life cycle
  - possibly modular in composition

Problems

- no isolation
  - same OS process, OutOfMemoryErrors, in-memory access to other application data
- trade-off between sharing versus protecting
- no accounting
- unwanted side effects
- unit of certified deliverable
  - cannot possibly test all potential interactions

Approaches

- servlet bridge
- nested / child OSGi frameworks
- Apache Karaf
- Eclipse features
- Eclipse Virgo
- IBM WebSphere, Paremus Service Fabric

OSGi Approaches

- Deployment Admin Service, Application Admin Service (older)
- nested OSGi framework
- Subsystems
  - part of Enterprise Specification, introduced in R5
  - based on experience in Eclipse Equinox, Virgo, Apache Aries, and others
  - explicit model of sharing policies: share all, share selected, share nothing
  - visibility boundaries for services, packages, events
  - implementation boundary already provided by OSGi service model

Advantages

- rich metadata supports large number of use cases
- common life cycle for constituents
- declarative approach
- independent of OSGi implementation
- reduces memory consumption by sharing
- supports dynamic resolution and installation of applications
  - unresolved deployment
  - SUBSYSTEM.MF vs DEPLOYMENT.MF

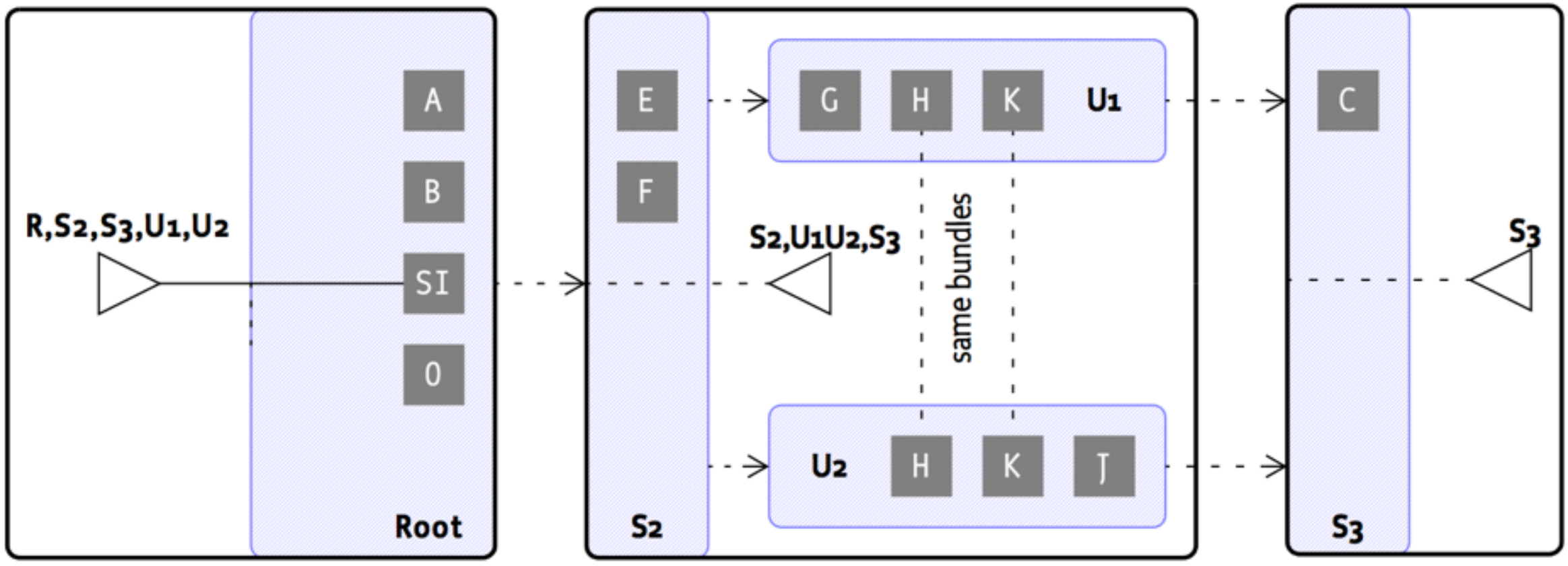
Example of Subsystems Metadata

SUBSYSTEM.MF

```
Manifest-Version: 1.0
Subsystem-ManifestVersion: 1.0
Subsystem-Name: Bank Account
Subsystem-SymbolicName: com.mybank.account.app
Subsystem-Version: 1.0
Subsystem-Type: osgi.subsystem.application
Subsystem-Content:
com.mybank.account.bankWeb; version=1.0.0,
com.mybank.account.bankAccount; version=1.0.0,
com.mybank.account.common; version=1.0.0,
com.mybank.account.utility; version=1.0.0
Use-Bundle: com.mybank.account.admin;version="[1.0.0,2.0.0)"
Subsystem-ExportService: com.mybank.account.service.AccountService
Subsystem-ImportService:
com.mybank.security.UserAuthService;filter="(security=strong)"
```

Drawbacks

- complicated
- complex policy definitions



Drawbacks

- complicated
- complex policy definitions
- configuration vs convention
- missing tooling: details exposed to users, application developers
- issues with common infrastructure services
  - e.g. Configuration Admin, Event Admin, Declarative Services, Blueprint and other extenders
- application is aware of subsystem environment, not functional in OSGi environment without it
- modular applications require updates of metadata in multiple places
  - think distributed application development
  - dynamic changes to the application
- subsystem needs to be started to share code

Drawbacks

SUBSYSTEM.MF

```
Manifest-Version: 1.0
Subsystem-ManifestVersion: 1.0
Subsystem-Name: Bank Account
Subsystem-SymbolicName: com.mybank.account.app
Subsystem-Version: 1.0
Subsystem-Type: osgi.subsystem.application
Subsystem-Content:
com.mybank.account.bankWeb; version=1.0.0,
com.mybank.account.bankAccount; version=1.0.0,
com.mybank.account.common; version=1.0.0,
com.mybank.account.utility; version=1.0.0
Use-Bundle: com.mybank.account.admin;version="[1.0.0,2.0.0)"
Subsystem-ExportService: com.mybank.account.service.AccountService
Subsystem-ImportService:
com.mybank.security.UserAuthService;filter="(security=strong)"
```

Subsystems Improvements

- enhancements worked out in RFC 201 scheduled for a later release of the Enterprise Specification
  - header localization
  - weaving hook integration / interaction
  - provide deployment manifest at install time
  - improve API for management agent
  - determine service dependencies of applications
  - revisit rules for preferred provider and application resolution

Alternative

10.1.8.19 public Bundle installBundle (String location) throws BundleException

location The location identifier of the bundle to install.

- Installs a bundle from the specified location identifier.

  This method performs the same function as calling installBundle(String,InputStream) with the specified location identifier and a null InputStream.

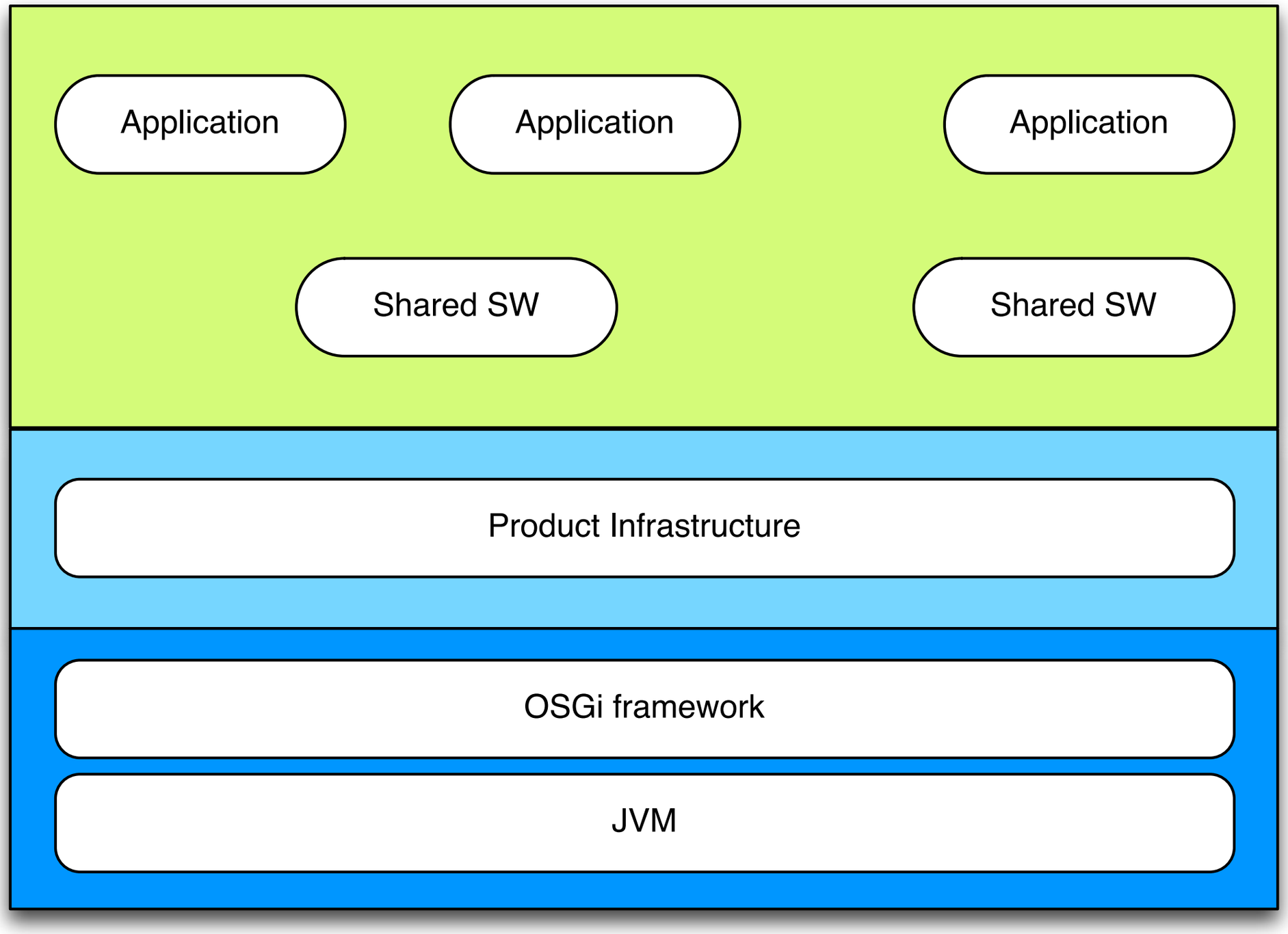
Returns The Bundle object of the installed bundle.

Throws BundleException – If the installation failed. BundleException types thrown by this method include: BundleException.READ_ERROR , BundleException.DUPLICATE_BUNDLE_ERROR, BundleException.MANIFEST_ERROR, and BundleException.REJECTED_BY_HOOK.

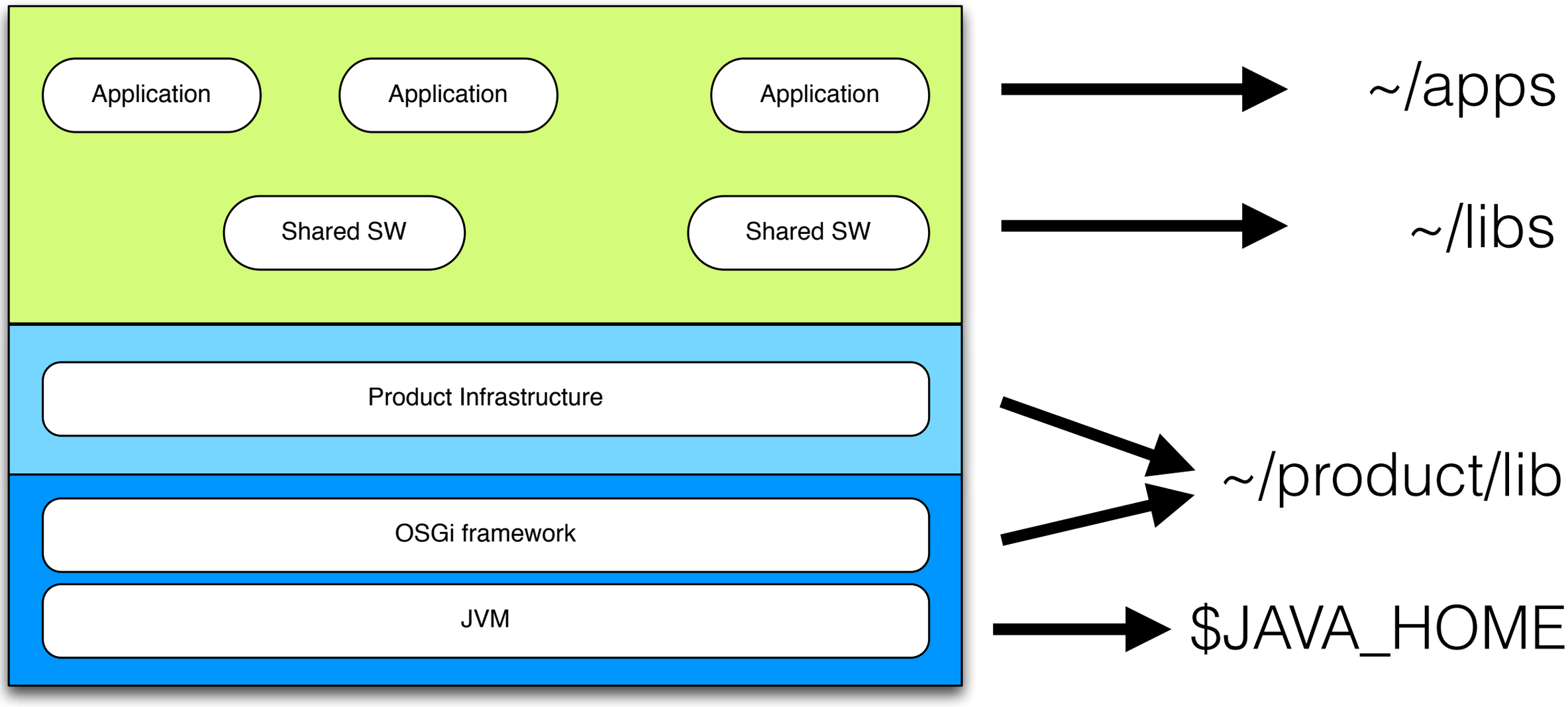
SecurityException – If the caller does not have the appropriate AdminPermission[installed bundle, LIFECYCLE], and the Java Runtime Environment supports permissions.

IllegalStateException – If this BundleContext is no longer valid.

Layering



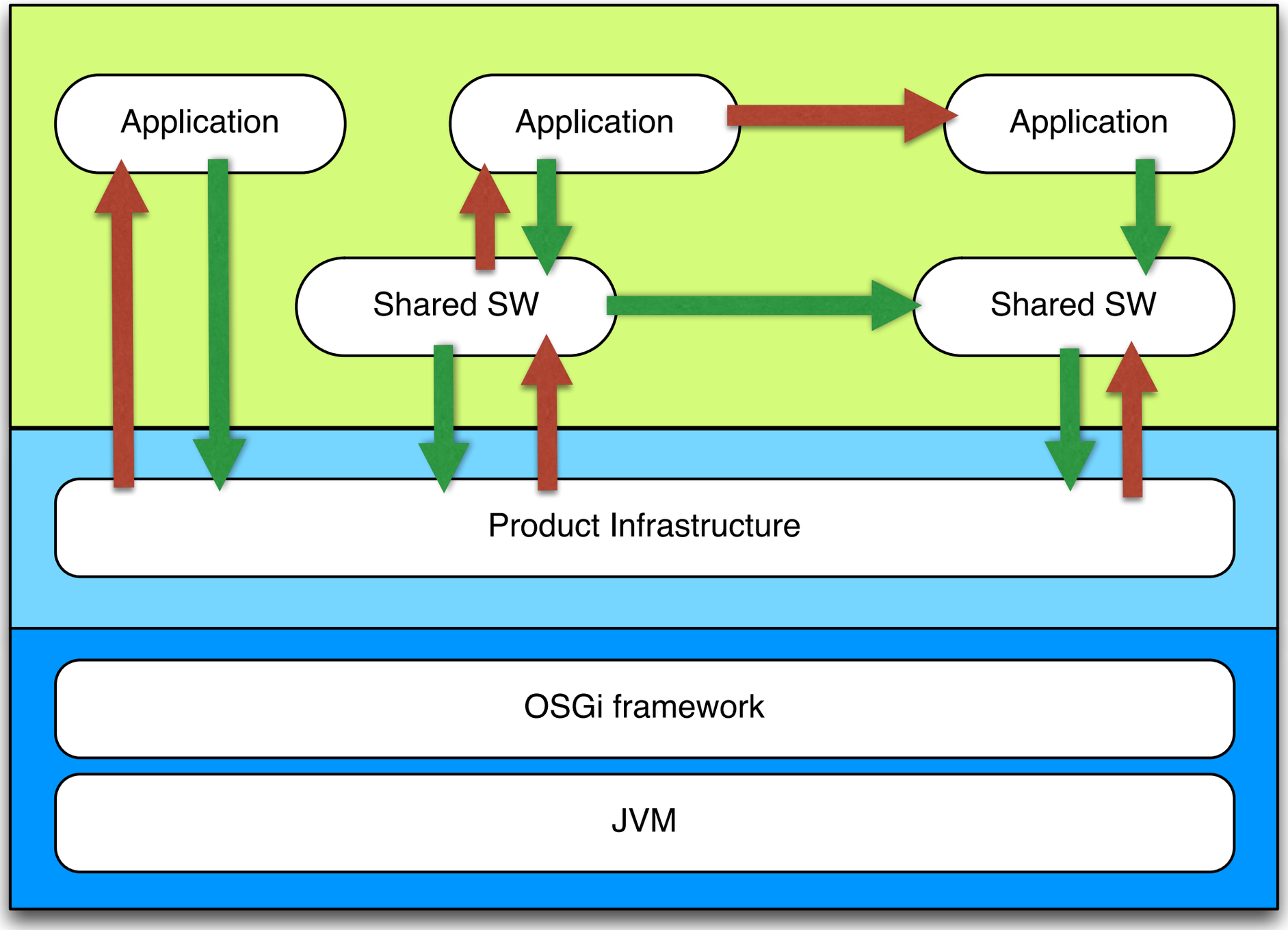
File System Layout



Solution

- file system is source of truth for deployment
- use the JVM as-is with its boot and ext class loaders
- place product code in one folder or structure
  - e.g. system or product
- place common infrastructure code in separate folder
- place shared libraries in separate folders
  - not necessarily part of the product, separate lifecycle
- place applications in separate folders

Visibility Boundaries



Advantages

- simple
- understandable
- changeable, flexible
- structured by location in file system
- no other metadata required
  - unless you want to model other dependencies and visibility boundaries
- compatible

Implementation

- different strategies possible
  - multi-tier level visibility
  - sensitive vs public services and code
    - visibility boundaries enforced via ResolverHooks, EventHooks, FindHooks, even WaevingHooks
- change strategy via configuration
- support multiple singletons if required by business logic, e.g. static variables in common classes

Disadvantages

- may not support all complex use cases
- requires additional metadata to support finer grain sharing policies
- too simple?

Real World Example

TIBCO ActiveMatrix BusinessWorks™ 6

TIBCO BusinessWorks™ 6

- ESB product, hosting multiple applications in single OSGi based runtime environment

- install product in one folder

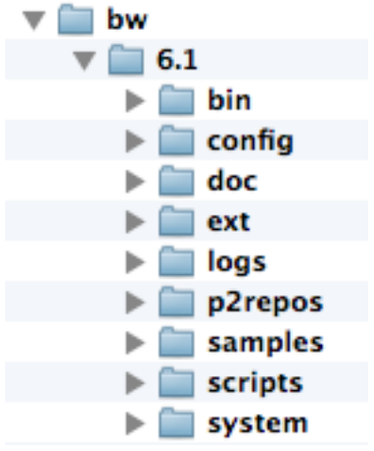
- install applications in a different folder

- determine visibility and accessibility based on folder location

Entitites

- product
  - BusinessWorks 6
- palettes (part of the product)
  - ~16 Palettes
- product extensions
  - palettes like salesforce.com, Twitter, etc.
- applications

Folder Layout



product layout:

system contains all product code

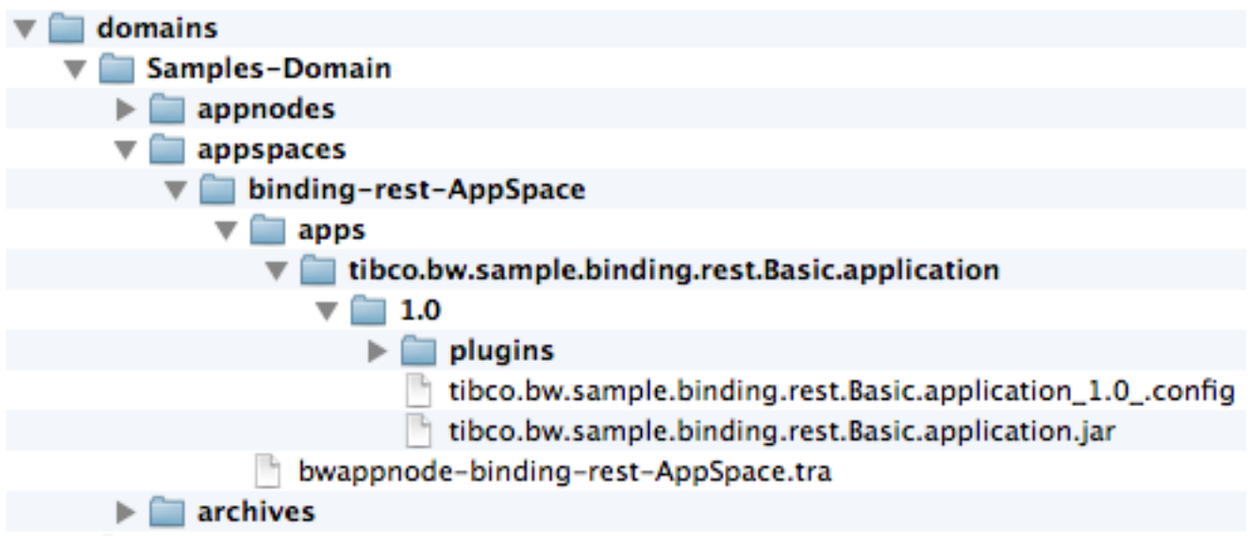
ext contains links to product extensions

application layout:

grouped in AppSpaces

versioned

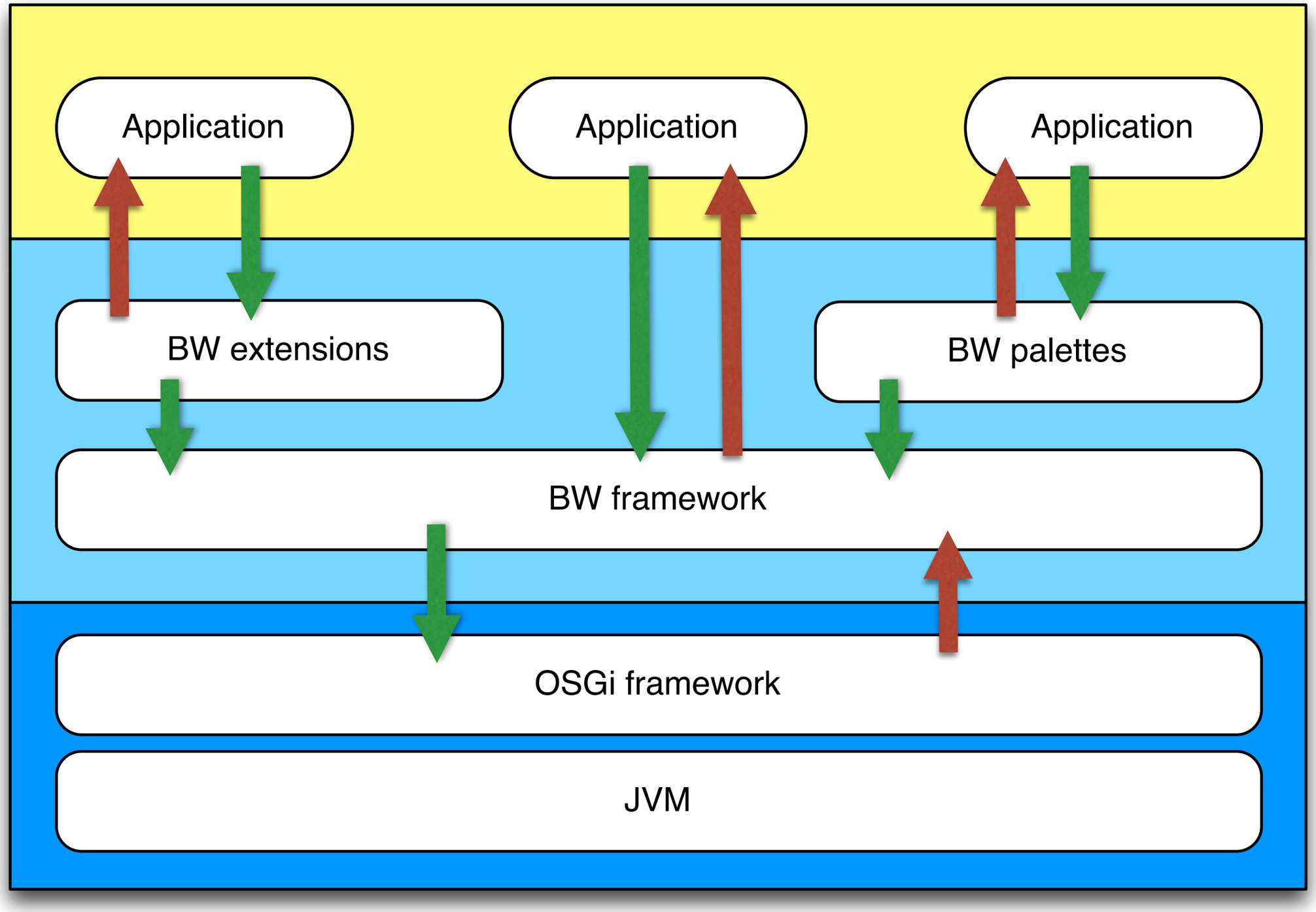
contains code and configuration



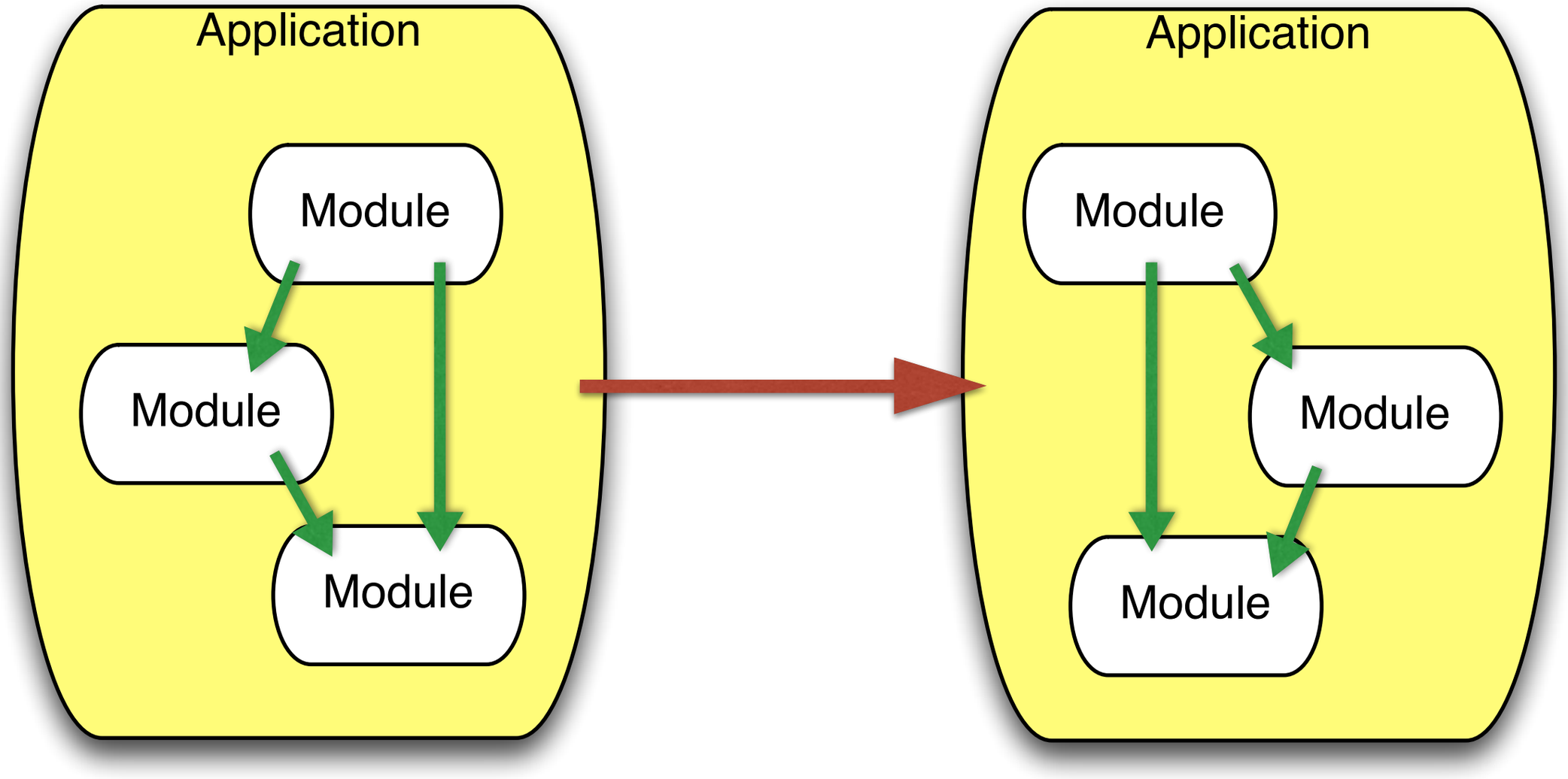
Visibility Boundaries

- product is self-contained in *system* folder
  - does not get wired outside of system folder
- product extensions are able to see and use anything in product folder
  - can also supply their own versions of infrastructure bundles
  - visible to all applications
- palettes sit above product and product extensions
- applications sit above palettes
  - can provide infrastructure that is confined to application

Visibility Boundaries



Applications



Specifics

- separate public from internal APIs
  - product internals are hidden from upper layers
- application support provided by Eclipse based designer
- additional application metadata
  - application identification and versioning
  - dependency management
- same hotfix mechanism for product as well as applications and extensions

Hotfixing

- simple, file system based
- system/hotfix folder repeats structure under system
- any files in hotfix override files in original location
- works for outer flat class path as well as OSGi class path environments

the end

questions?

References

- OSGi Alliance, http://www.osgi.org
- OSGi Core Specification, http://www.osgi.org/Download/File?url=/download/r5/osgi.core-5.0.0.pdf
- Enterprise Specification, http://www.osgi.org/Download/File?url=/download/r5/osgi.enterprise-5.0.0.pdf
- Introduction to Java Multitenancy, http://www.ibm.com/developerworks/library/j-multitenant-java/
- Apache Aries, https://aries.apache.org
- wikipedia.org, http://en.wikipedia.org/wiki/OSGi
- Equinox Regions, Mind the Gap Blog, http://underlap.blogspot.com/2011/02/stumbling-towards-better-design.html
- Paremus Service Fabric, http://www.paremus.com/products/products_psf.html
- TIBCO ActiveMatrix BusinessWorks 6, http://www.tibco.com/products/automation/application-integration/activematrix-businessworks/default.jsp