Liferay Architecture

Understanding the inside of Liferay

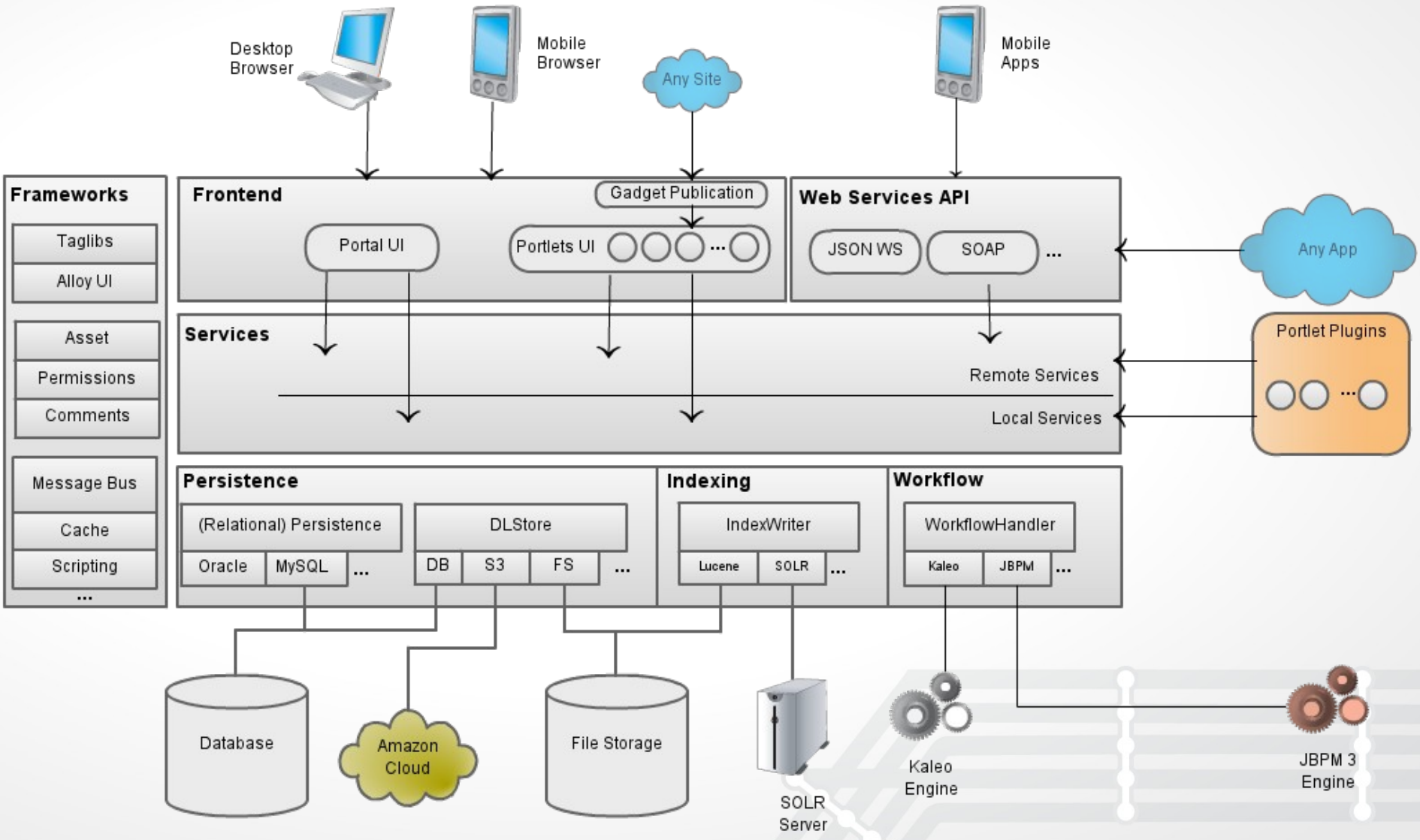
Jorge Ferrer

Vice President

Engineering

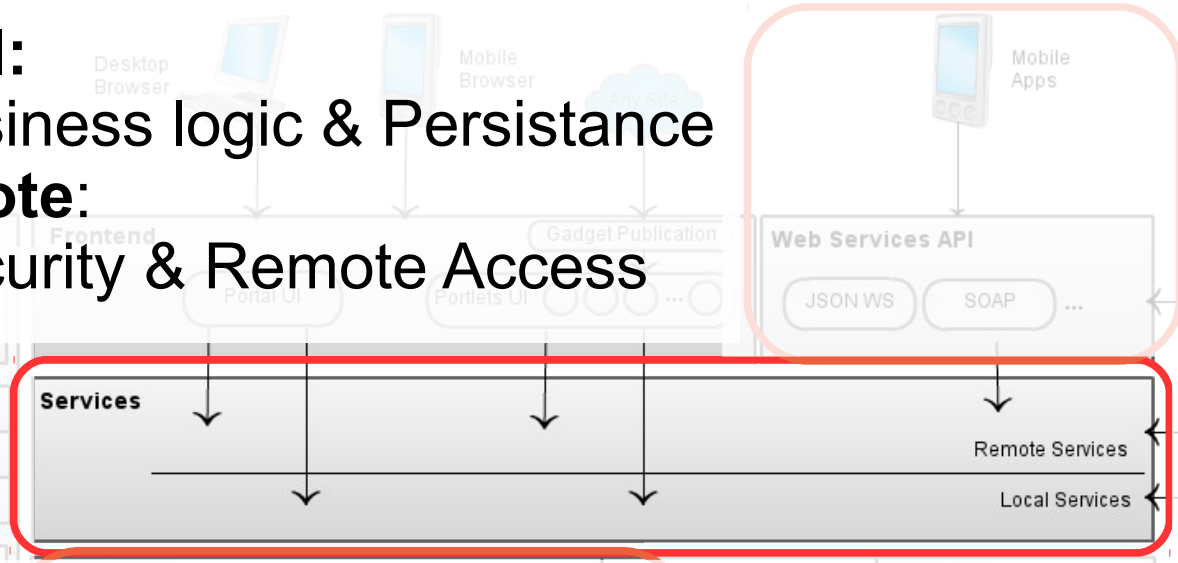
Table of Contents

Architecture Overview


Desktop Browser
Mobile Browser
Any Site
Mobile Apps
Frameworks
Taglibs
Alloy UI
Asset
Permissions
Comments
Message Bus
Cache
Scripting
...
Frontend
Gadget Publication
Portal UI
Portlets UI
...
Web Services API
JSON WS
SOAP
...
Any App
Portlet Plugins
Services
Remote Services
Local Services
Persistence
(Relational) Persistence
Oracle
MySQL
...
DLStore
DB
S3
FS
...
Indexing
IndexWriter
Lucene
SOLR
...
Workflow
WorkflowHandler
Kaleo
JBPM
...
Database
Amazon Cloud
File Storage
SOLR Server
Kaleo Engine
JBPM 3 Engine

Services (and persistence)

- **Local:**
  Business logic & Persistance
- **Remote**:
  Security & Remote Access



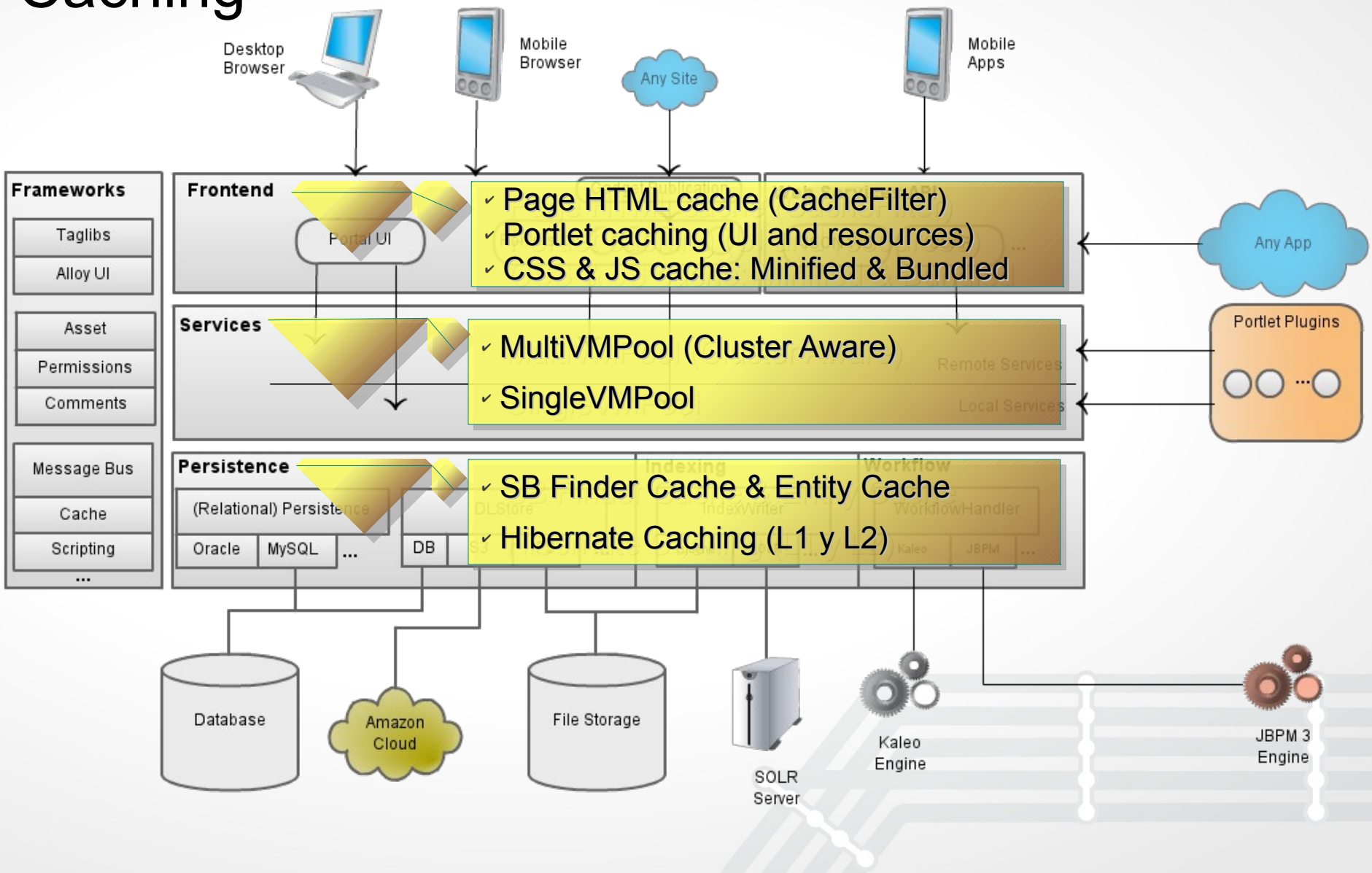
Services Patterns

- Service per model entity
- Always return void, <Entity> o List<Entity>
- Grouped as:
  - Portal services
  - Per “Portlet” services
- Encapsulate all access to Persistence and Finders
- Security must be in remote layer

Service Builder

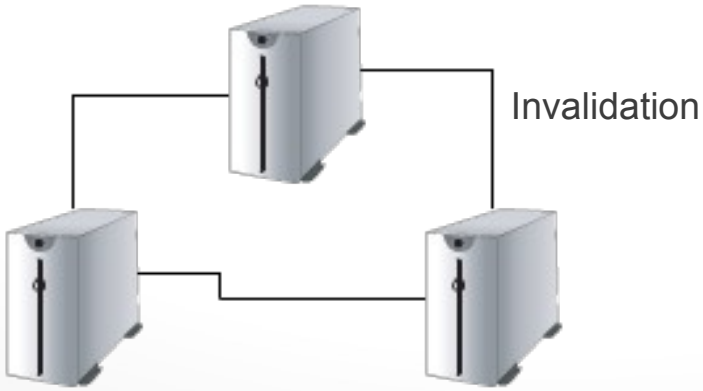
- Takes care of all the plumbing and heavy lifting:
  - Persistence (Hibernate and custom SQL)
  - Caching (Persistence layer)
  - Services and remote protocols
  - Spring wiring
- Define your models and services in service.xml
  - See DTD to find out about all options
- Code generated via Freemarker Templates

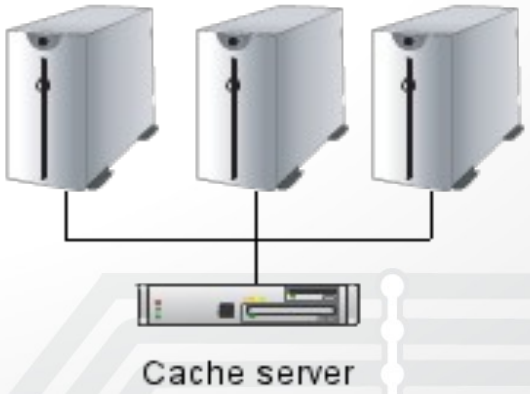
Caching


Desktop Browser
Mobile Browser
Any Site
Mobile Apps
Frameworks
Taglibs
Alloy UI
Asset
Permissions
Comments
Message Bus
Cache
Scripting
...
Frontend
Portal UI
Page HTML cache (CacheFilter)
Portlet caching (UI and resources)
CSS & JS cache: Minified & Bundled
Any App
Services
MultiVMPool (Cluster Aware)
SingleVMPool
Portlet Plugins
Persistence
(Relational) Persistence
Oracle
MySQL
DB
SB Finder Cache & Entity Cache
Hibernate Caching (L1 y L2)
Database
Amazon Cloud
File Storage
SOLR Server
Kaleo Engine
JBPM 3 Engine

Cache Distribution

- One to one
  - RMI (Default <6.1)
  - Multicast (Default 6.1+)
- Centralized
  - Terracota (Terracota Edition)
  - Memcache



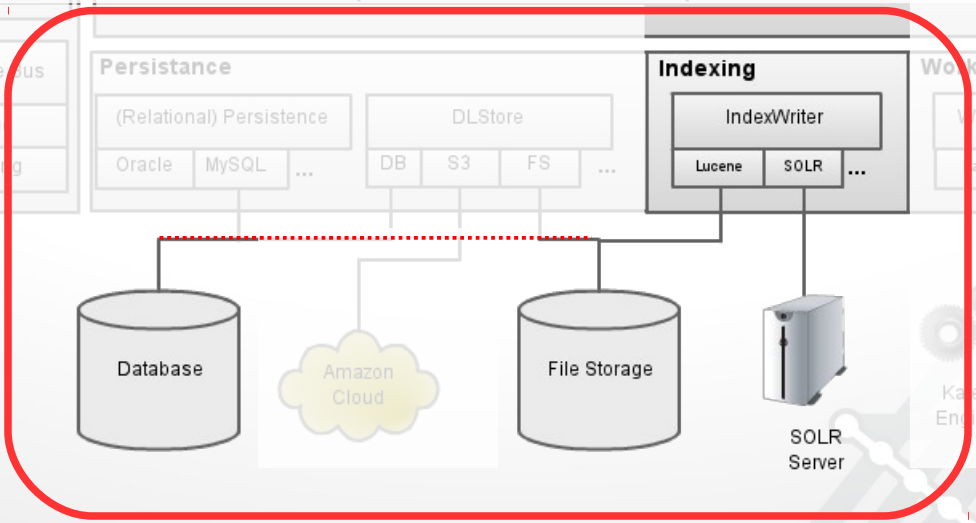


Transactions

- Definition: Executing operations atomically
- Service methods happen within a transaction (Based on Spring)
  - ServiceBuilder generates annotations:
    - `add*, check*, clear*, delete*, set*, and update*`: require propagation, read-write
    - All others: supported, read-only
- How about transactionality in plugins that invoke the portal?
  - Automatic support for “distributed” transactions across plugins by reusing Hibernate's connection

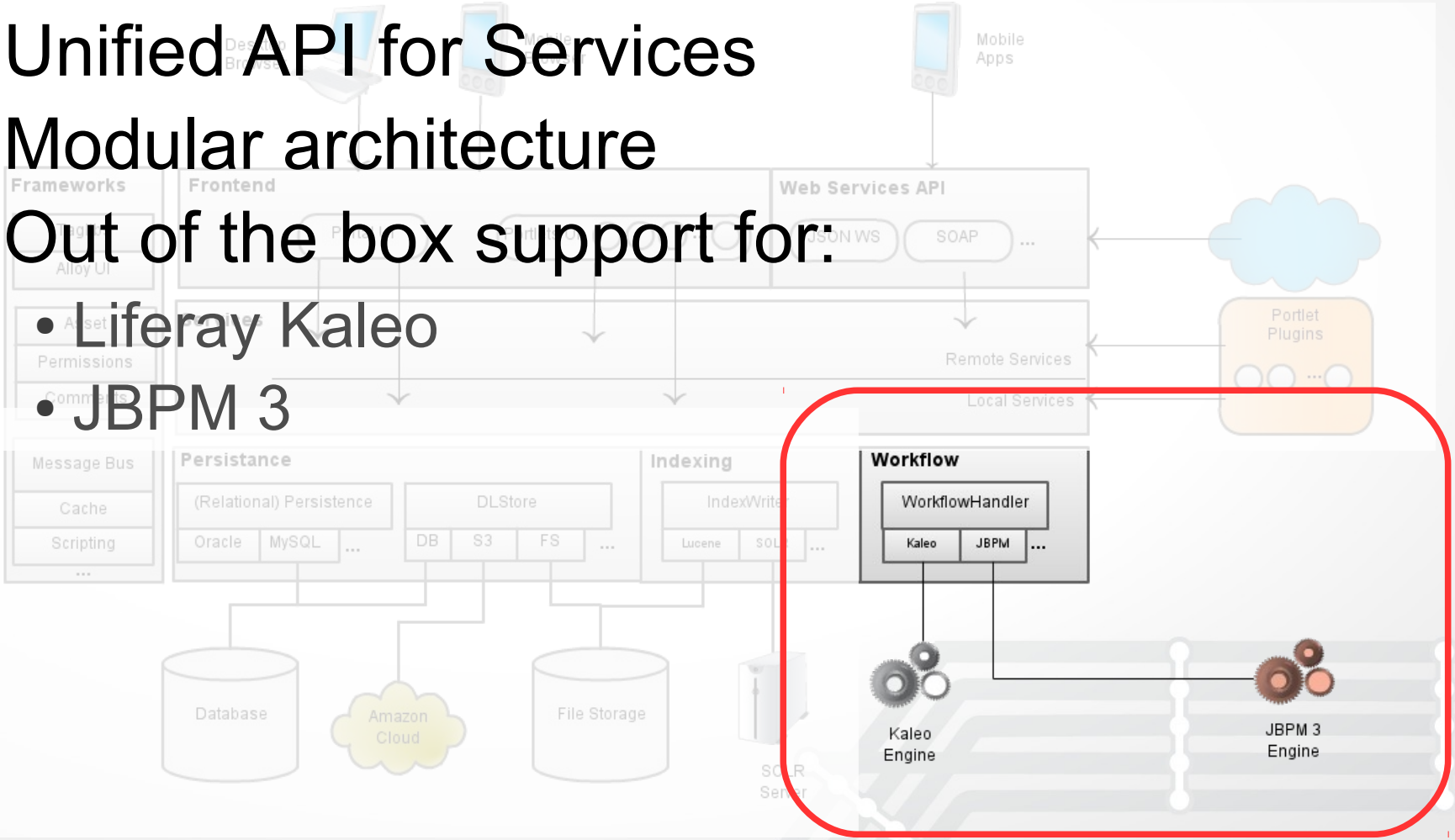
Indexing

- Modular architecture
- Out of the box support for:
  - Lucene (Clustering with ClusterLink via jGroups)
  - SOLR
- Asynchronous indexation (Message bus via AOP)



Workflow Framework

- Unified API for Services
- Modular architecture
- Out of the box support for:
  - Liferay Kaleo
  - JBPM 3



Frameworks

- **Permissions**: role based, resource oriented
- **Asset** Framework: common functionality for all content: tags, categories, counts, ...
- **Message Bus**: Used mainly for Asynchronous Invocations
  - Used for: indexing, metadata extraction, notifications, ...
- **Scripting**:
  - Supports: JavaScript, Ruby, Python, …
  - Forbidden classes
- **Audit**: auditing user actions
- **Expando**: allows dynamic relational storage
- More...

The secret ingredients

Consistency

- Never reinvent the wheel
  - See if it's already coded somewhere else
- Always follow existing patterns & standards
  - Use Copy & Paste intelligently when needed
- Reuse and code to allow reusing

Enforced through **Peer Reviews** and a code **Gatekeeper**: Brian

Facilitate maintainability

- Use modern de-facto standard libraries: spring, hibernate, lucene, …
  - But use them only for what is really needed
- Always find the best place for your code
  - (Even) If it's not broken ~~don't fix it~~ change it
- Never leave legacy behind

Enforced through **Peer Reviews** and a code **Gatekeeper**: Brian

Contributions

- Great ideas come from advanced developers using the product
- Some examples:
  - Service Builder Freemarker templates
  - Auto Deploy
  - Clp

Quality guaranteed through Peer Reviews and a code Gatekeeper: Brian

Coming next

- Module Framework (**OSGi**)
- Application **Resiliency**
- More **hook-ability** to facilitate combination of hooks

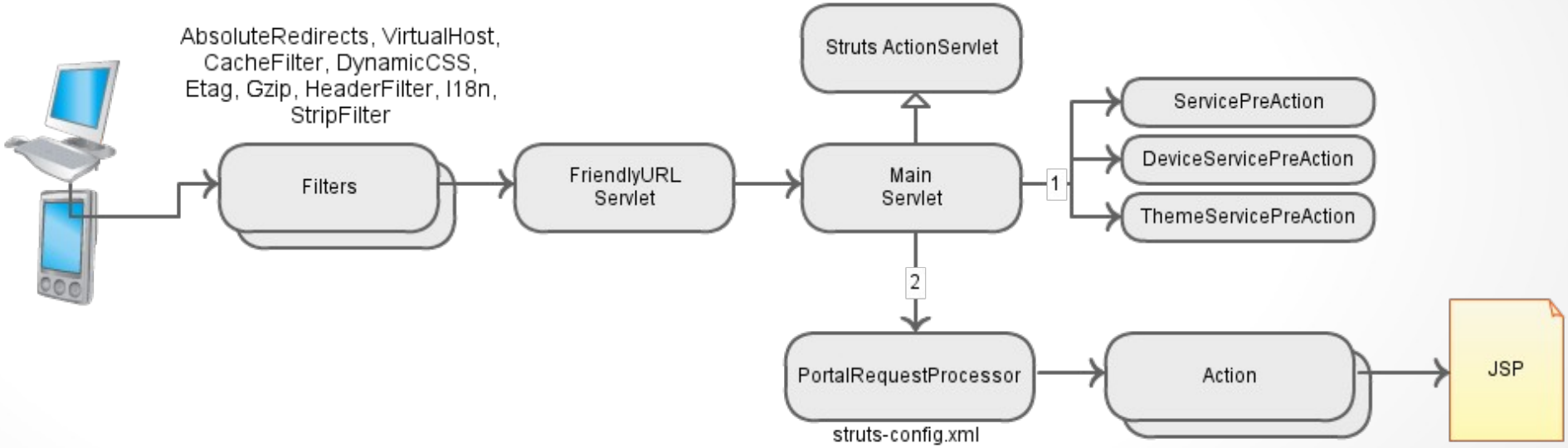
Thanks!

Bonus content in the downloadable slides:

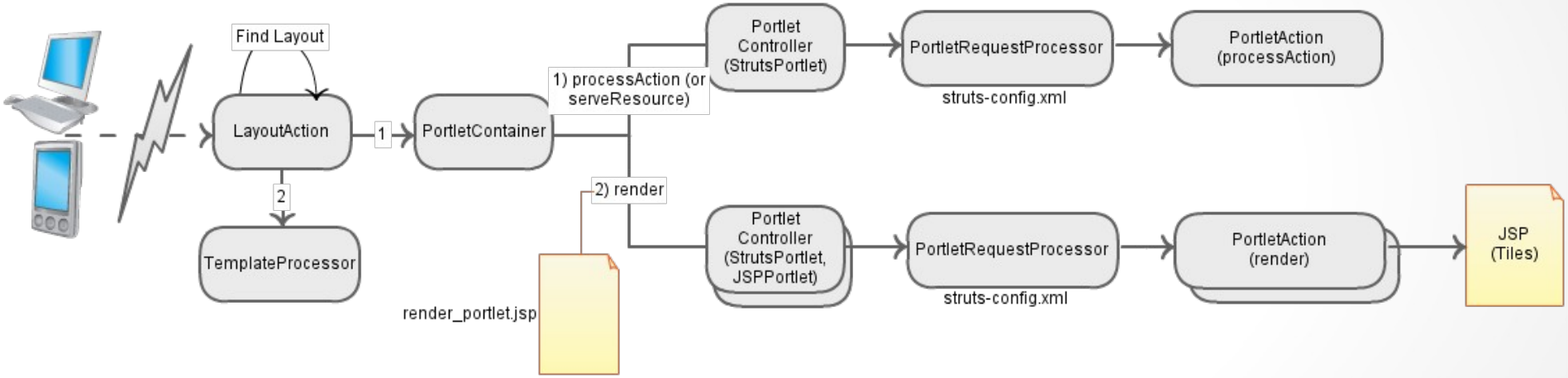
- Request Handling
- Plugin Architecture
  - Deployment
  - Class loading

Request Handling

Request Cycle – Portal URL



Request Cycle – Portlet URL

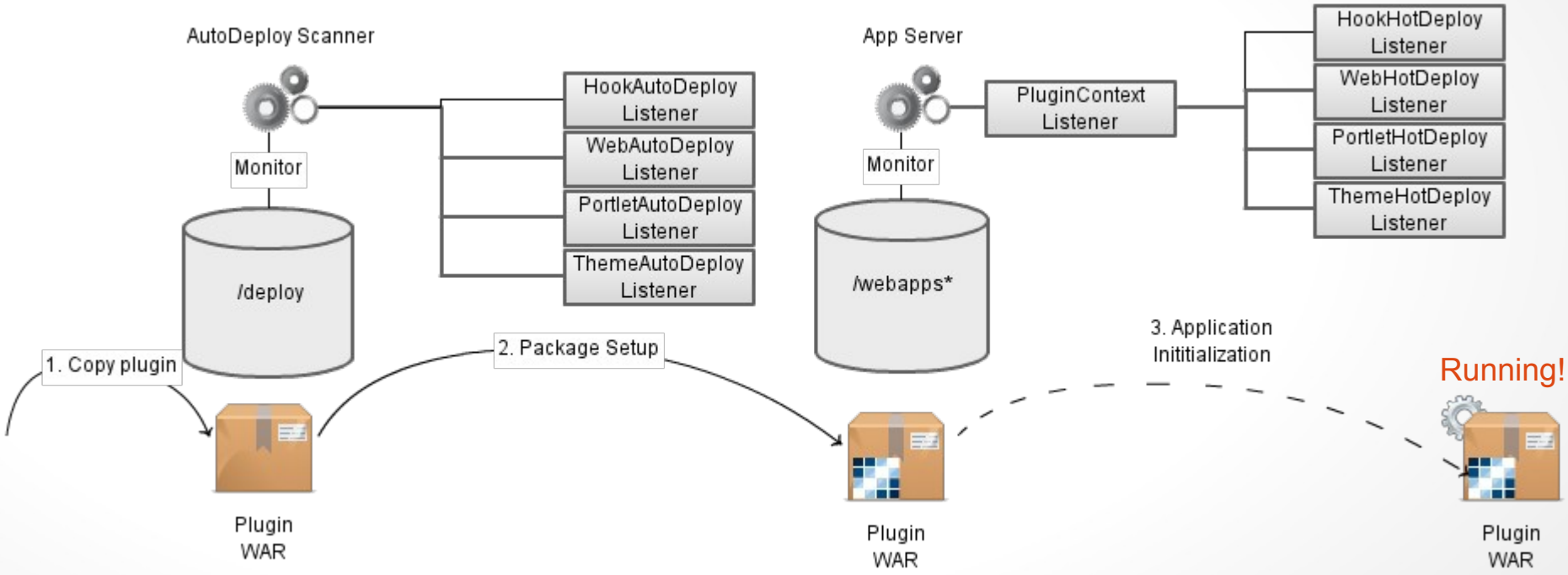


Plugin Architecture

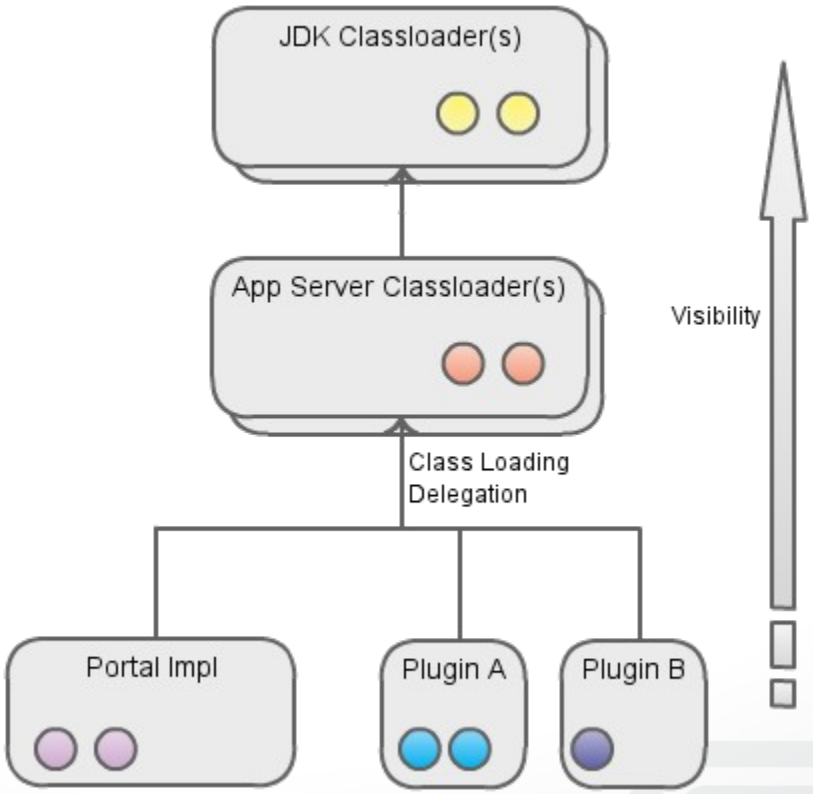
Application Containers

- Portlet Container: included in the core
- OpenSocial Container
- Other frontend integration solutions:
  - PortletBridge: Integrated in Liferay as a portlet, WebProxy.
  - Iframe Portlet: Smart resizing and tracking
  - WSRP: Provided through a plugin

Plugin Deployment



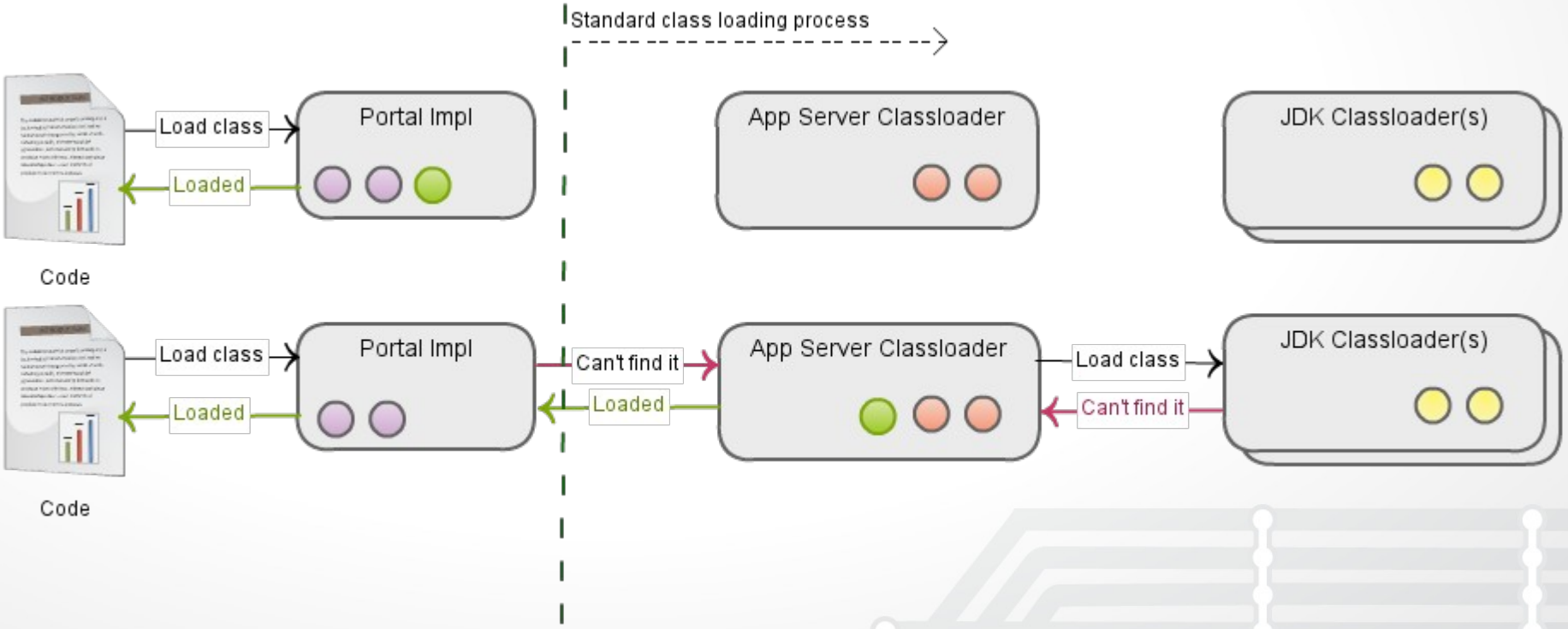
Class Loading Hierarchy



Classes should be no higher in the hierarchy than they are supposed to exist

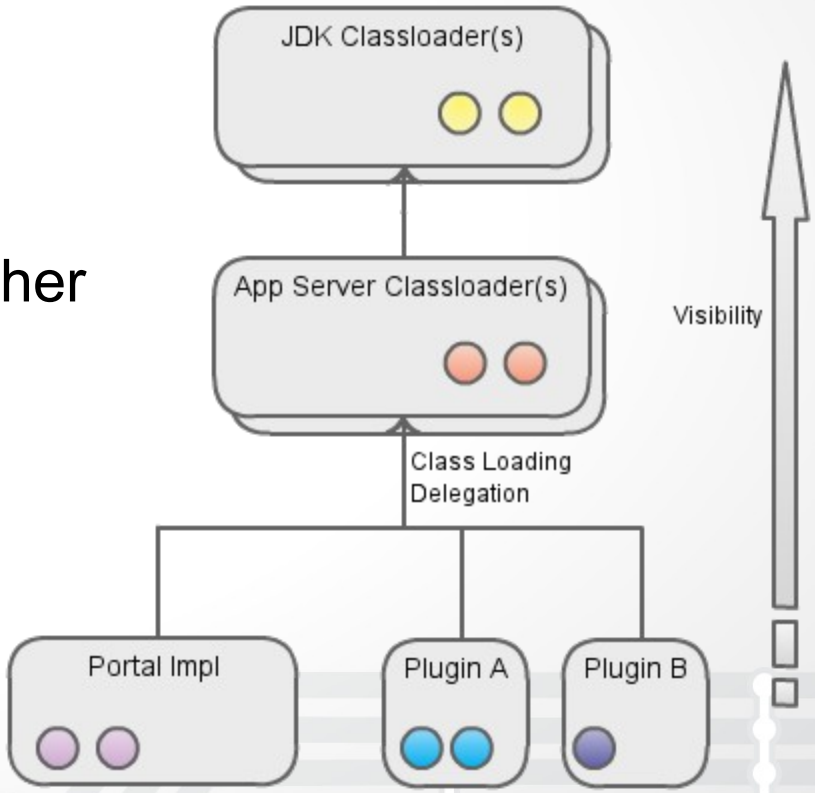
Class Loading Process

Classloaders delegate the requests to load that class to their parent classloaders



Class Loading Manipulation

1. Interface-based Dependency Injection
   - Invocation of services
   - Invocation of PortalImpl and other utils
2. Explicit class loader invocation
   - CLP: inter-portlet invocation
   - PortalClassLoader



Class Loading for Ext, Hook,Web

1. Ext
    1. All code executes in Portal context
2. Hook
    1. Java classes execute in plugin context
    2. JSPs execute in Portal context
3. Web
    1. All code executes in plugin context