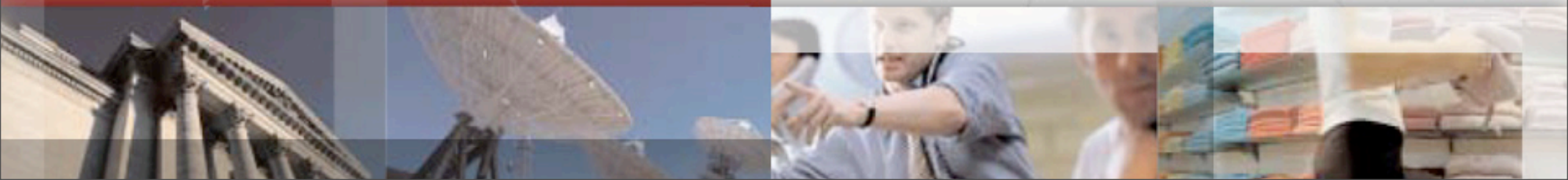
TIBCO®
The Power of Now®

Using OSGi Metadata with a Standard Class Loader

David Kemper
Principal Architect
djk@tibco.com

TIBCO Software Inc.
http://www.tibco.com



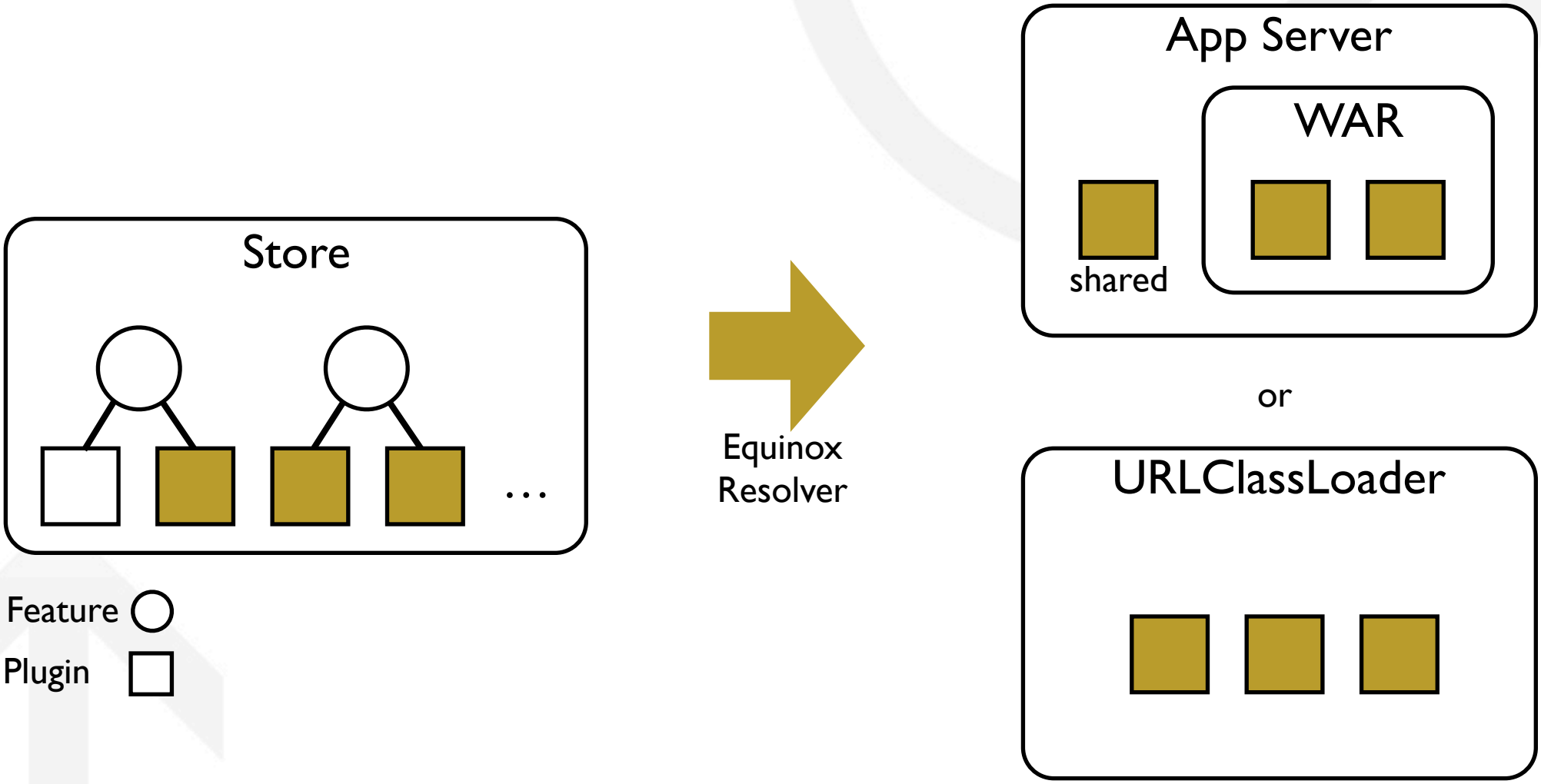
Why use OSGi metadata without an OSGi runtime?

- **Why not use an OSGi runtime?**
  - Implementing a specific standard with its own class loader model (for example J2EE)
  - Simple use case like a command-line tool where OSGi is overkill
  - Required to use legacy runtime (for example based on a URLClassLoader)
- **But when you are not in OSGi…**
  - Duplicate packages will collide
  - Un-exported packages are visible
  - You can't run against multiple versions of a component
  - Fragments and singletons have no special meaning in the runtime
  - You often cannot use "packed" plugins (a jar-within-a-jar)
- **Why use OSGi metadata?**
  - Rich description of the dependencies between jars
  - Co-located with the component itself, so easy to carry through to runtime
  - Tools can use the metadata for validation and runtime provisioning
  - Eclipse tooling supports the creation and management of this metadata

How do you provision with OSGi metadata?

- **Given a store of plugins and features and a set of initial constraints, determine the *optimum*, *minimal* set of plugins that satisfy the constraints**
- **It is helpful to define initial constraints using Eclipse features**
  - Reuse Eclipse packaging infrastructure
  - Customers exposed to less detail
- **Use Equinox for the heavy lifting**
  - Leverage infrastructure already available to the Eclipse design time; don't reinvent the wheel
  - Equinox externalizes and supports its resolver APIs
  - Feature dependencies look like Require-Bundle constraints

Examples



What are some of the challenges?

- **You'll want to filter your inputs and outputs**
  - You don't want to resolve all metadata from your store because it doesn't scale well and is overly constrained
  - Support for a shared class path or screening specific versions add complexity
- **Going from a set of plugins to a Java class path is more work**
  - Java class path is not generated by the Equinox Resolver
  - You have to take fragments into account
  - Native library paths are available in version 3.4
- **Interpreting Equinox Resolver errors is not straightforward**
  - You have to dig for the root cause of an unresolved plugin and correlate it back to the input constraint

Questions

- David Kemper
TIBCO Software Inc.
djk@tibco.com