DEVCON
LIFERAY DEVELOPER CONFERENCE

MOD CONF

Modules from Front to Back

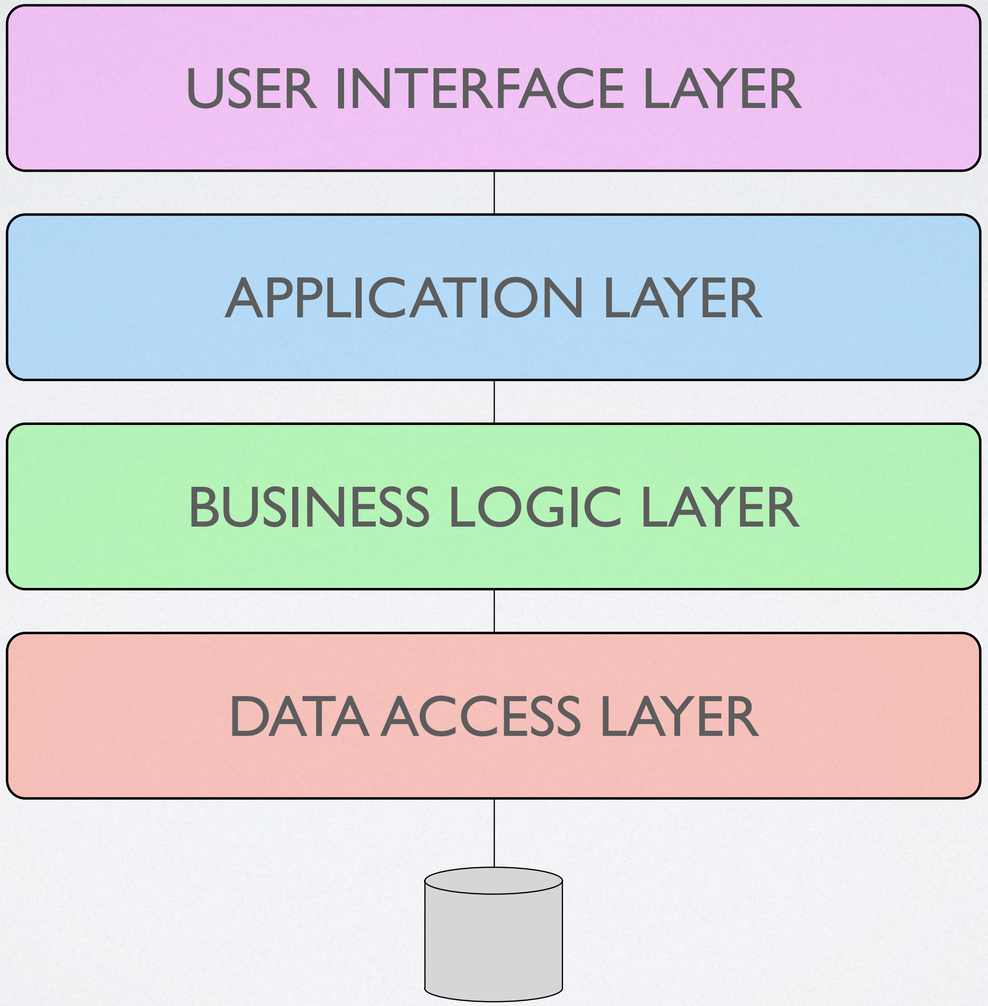
Java-OSGi and Web Components

Neil Bartlett

bndtools

Introduction

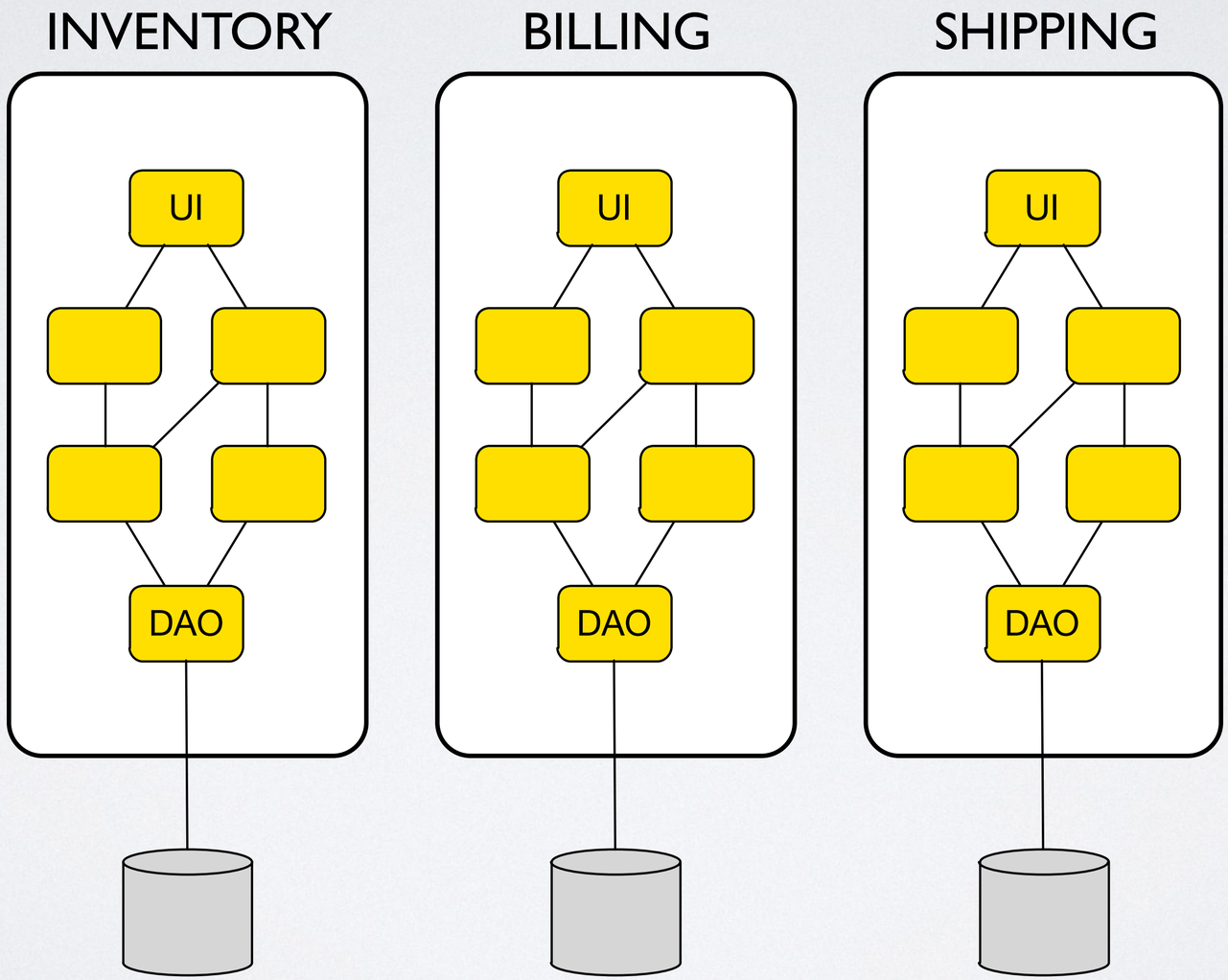
Problem Statement



Problem Statement

- New features cut across all layers.
- IT organises around technologies rather than business areas.
- Blame shifting: "The database is fine so it's not **our** problem!"
- **Not Modular**… just try leaving out a layer!

Cutting With the Grain



Benefits

- IT organisation aligned with business.
- Add new features without coordinating multiple teams.
- **Modular**: slices can be omitted, added, evolved independently.

www.ebay.co.uk

Hello Neil. | Daily Deals | Sell | Help & Contact | SAVE UP TO 50% ON REFURBISHED TECH > | My eBay

ebay Shop by category | running shoes | All Categories | Search | Advanced

Related: mens running shoes size 9 | running shoes size 5 | running shoes size 9 | mens running shoes | asics gel | mens running shoes size 10… | Include description

Categories
All
Clothes, Shoes & Accessories
Men's Trainers
Women's Trainers
Men's Casual Shoes
More
Show more

Shoe Size see all

All listings | Auction | Buy it now | Sort: Best Match | View:

383,526 results for running shoes | Follow this search | Postage to EN6 2DD

Shop by category

Men's Trainers | Women's Shoes | Men's Casual Shoes | Sporting Goods

pay.ebay.co.uk

ebay Checkout

How do you like our checkout?
Tell us what you think

Pay with

PayPal

Credit or debit card VISA

Post to

Neil Bartlett

Item (1) £89.49
Postage Free

Order total **£89.49**

I agree to the PayPal terms and Privacy Policy. PayPal provides this payment service. PayPal account not required.

Confirm and pay

Granularity

- The eBay model is good but very large-grained.
- Each app is monolithic and suffers many problems.
- Can we design web applications that are truly modular, top to bottom?

OSGi

OSGi Overview

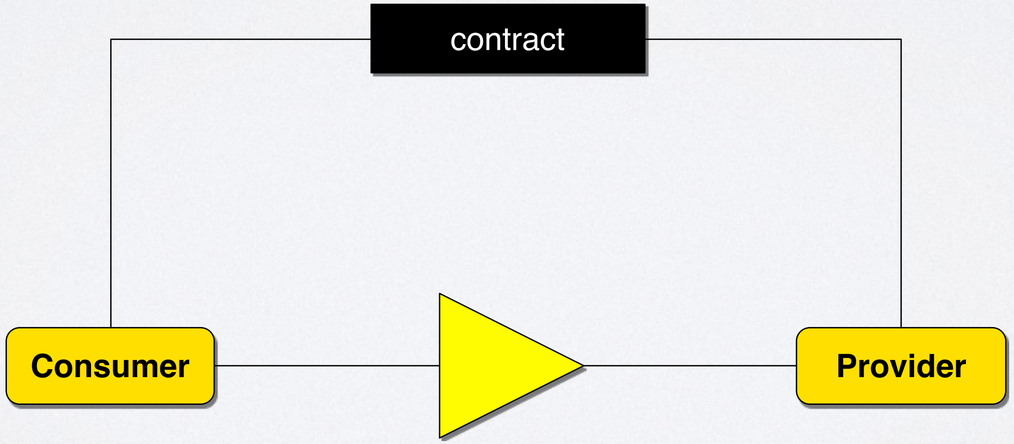
- The **most comprehensive** solution for Java modularity.
- 1999 – JSR 8: Open Services Gateway Specification
- 2000 – OSGi Alliance formed, OSGi Release 1 published.
- Today: Release 6, working towards Release 7.
- **We've got the back-end covered**.

What is OSGi *Really*?

- Sometimes called "ClassLoaders on Steroids"
- I'm **not** going to talk about ClassLoaders, Bundles, or Package Imports/Exports! These things are important but **boring**.
- Services are they key to OSGi ("Open **Services** Gateway").

Services

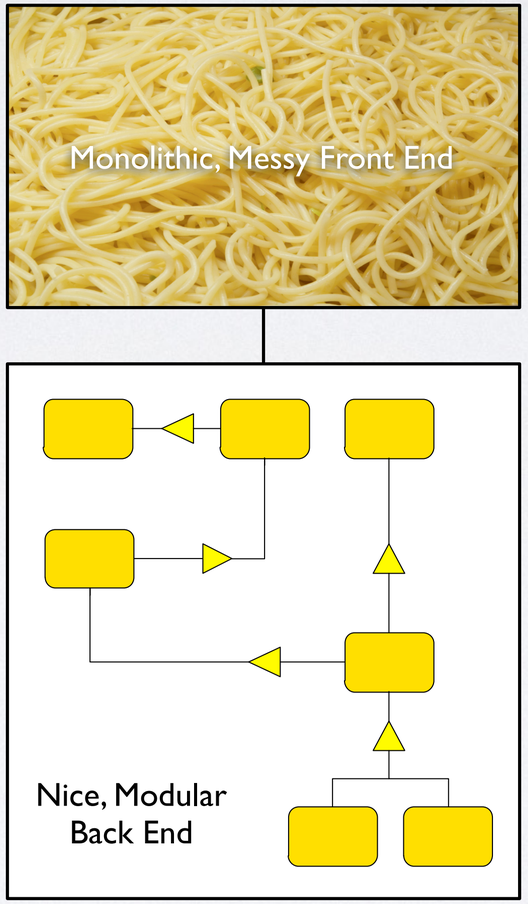
- Enabler for Component-Based Software Engineering.
- Robust, dynamic, adaptable.



Component Implementation

- Many implementation technologies: Declarative Services, Blueprint, iPOJO, Peaberry, or "raw" API.
- All interoperate through standardised abstraction.
- Consumers and Providers are **oblivious**.

Problem



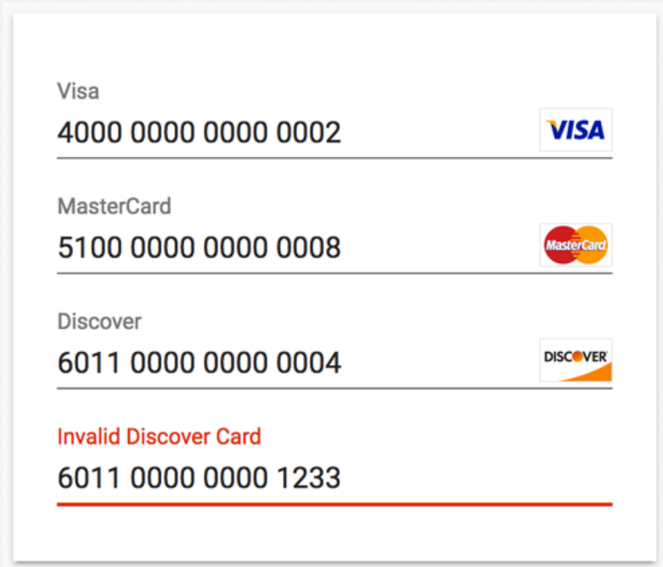
Web Components

What Are Web Components?

- A family of specifications/features being added to HTML and DOM.
- Bringing Component-Based Software Engineering to WWW.
- Key features: Custom Elements, Shadow DOM, HTML Imports, HTML Templates.
- **Assemble** a web app from off-the-shelf & custom components.

Example

```
<gold-cc-input label="Visa" auto-validate></gold-cc-input>
<gold-cc-input label="MasterCard" auto-validate></gold-cc-input>
<gold-cc-input label="Discover" auto-validate></gold-cc-input>
<gold-cc-input label="Invalid Discover Card" auto-validate></gold-cc-input>
```



Web Component Implementation

- Many implementation technologies: Polymer, Bosonic, X-Tag, Mozilla Bricks, or "raw" API (aka vanilla JS).
- All interoperate through standardised abstraction.
- Composable – components made of smaller components.

Problem

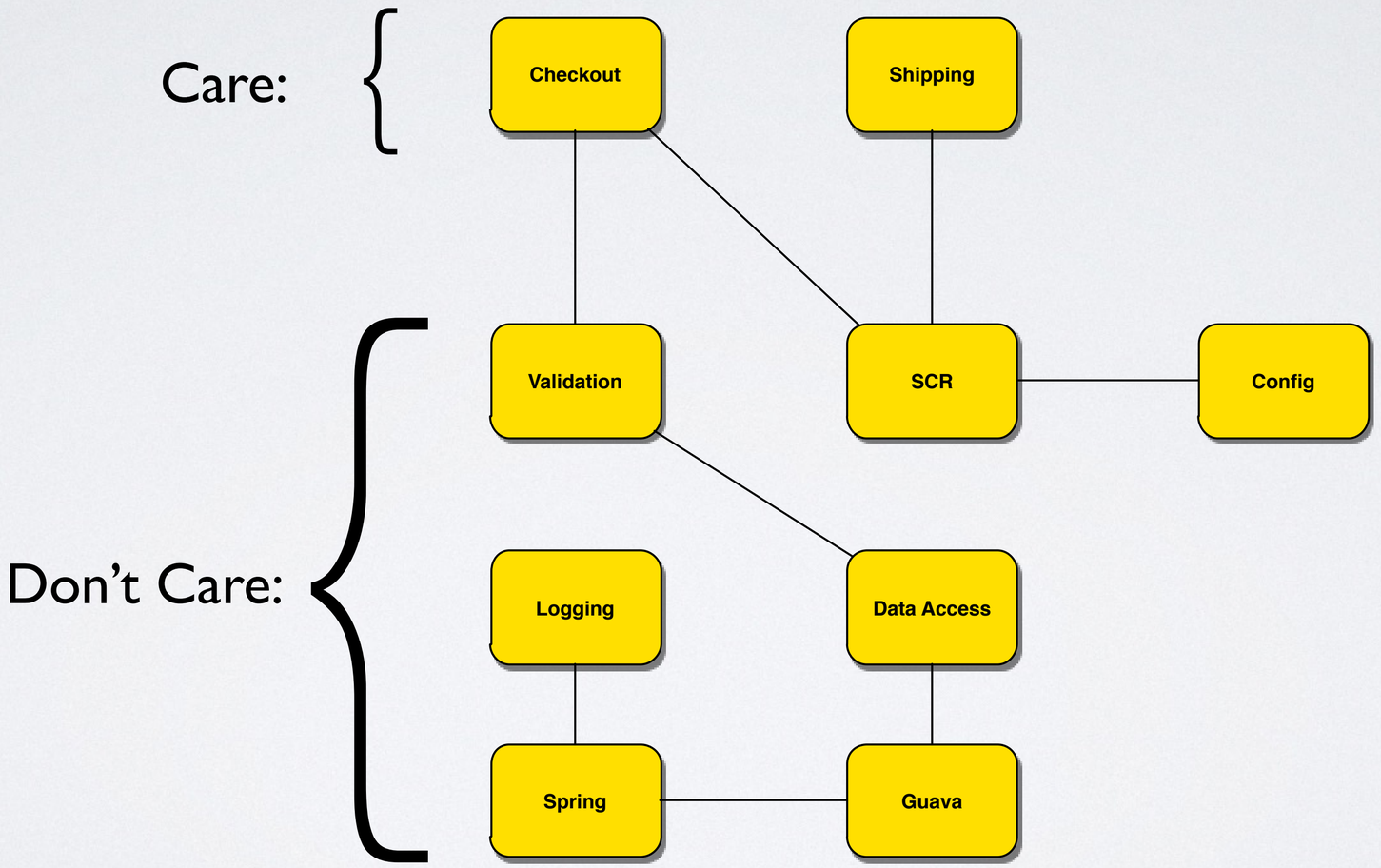
- Off-the-shelf Web Components tend to be tiny UI widgets.
- E.g.: carousels, credit card capture, address capture, charts…
- Larger components will require back-end integration and data.
- How do we ensure that front- and back-end components are always deployed?

Application Assembly

Application Assembly

- In OSGi, the "application" **emerges** from the installed bundles.
- But not all bundles are equally interesting.

Application Assembly



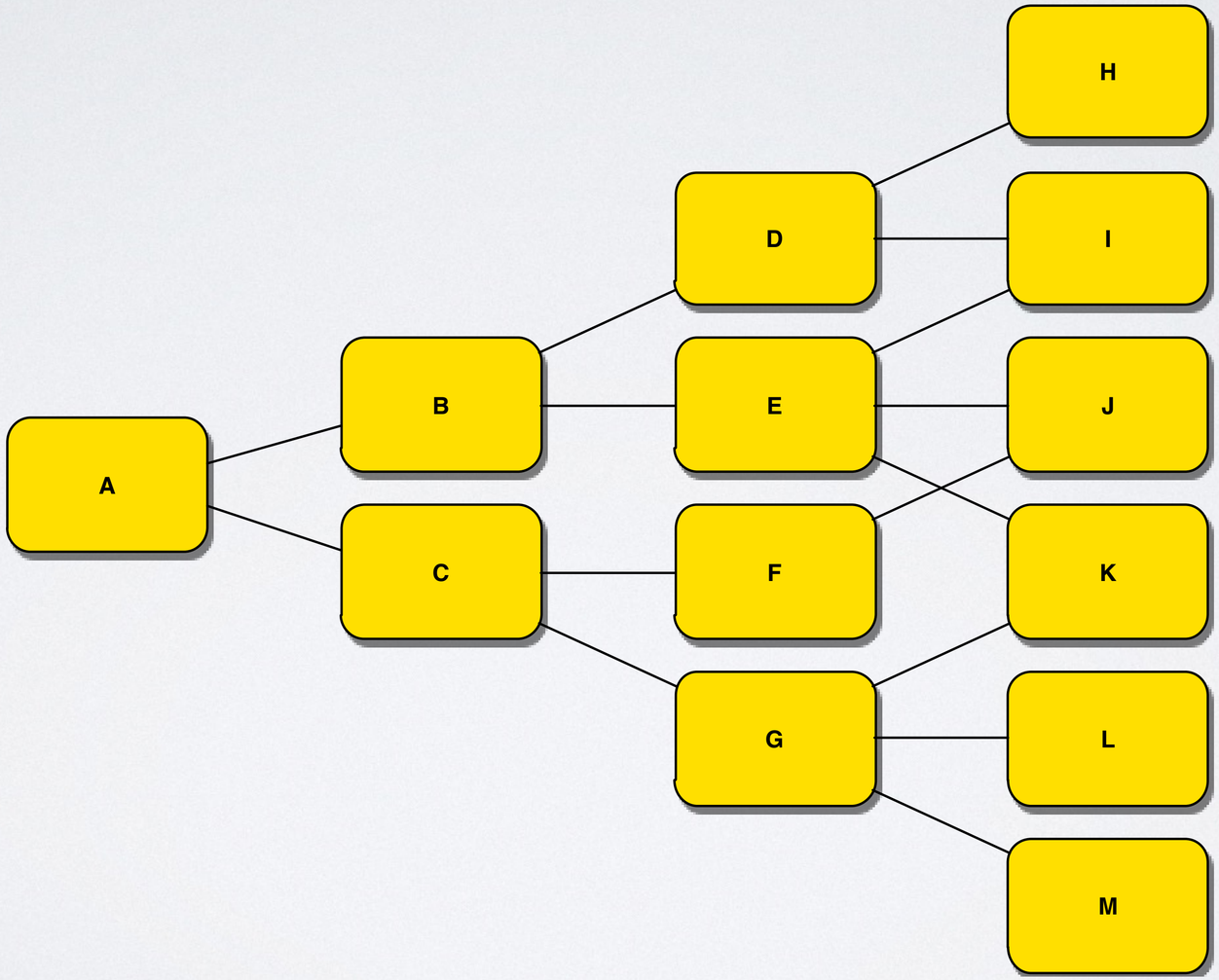
Application Assembly

- All are necessary, but I don't want to keep track of them all.
- Solution: dependency resolution.
- State what you **want**. Tool tells you what you **need**.
- Examples: Maven, npm, Bower…

The Trouble Is…

- Existing package managers do a **terrible** job.
- These package manager use identity for dependencies.
- The result is **fan-out**.

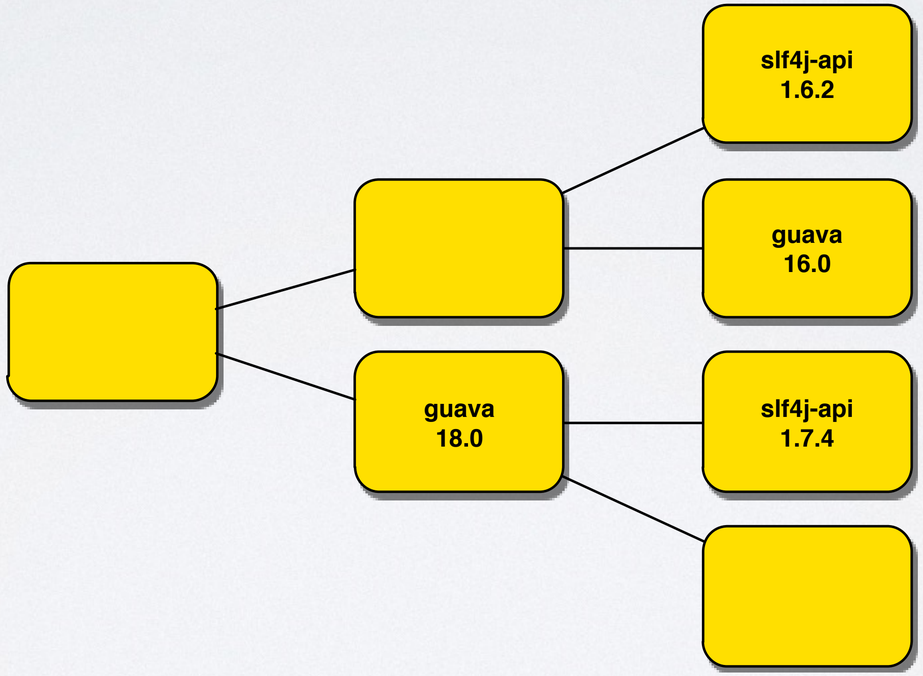
Fan-Out



Oh Maven!

```
[INFO] ------------------------------------------------------------------
[INFO] Building org.example.foo 0.0.1-SNAPSHOT
[INFO] ------------------------------------------------------------------
[INFO]
[INFO] Downloading the Internet.....................................
.....................................
```

Conflicts



Capabilities & Requirements


```
Provide-Capability: acme.display;
        width=1024;
        height=800
```

```
Require-Capability: acme.display; filter:=“(&
        (width>=800)
        (height>=600))”
```

Capabilities & Requirements

- Namespace is arbitrary – reserved namespaces begin with "`osgi.`"
- Completely generic model, not limited to OSGi or even Java.
- Don't depend on "who" a module is. Depend on what it **can do**.
- (**Can** still depend on identity: `osgi.identity` namespace.)

Capabilities & Requirements


```
Provide-Capability: modconf.webservice;
    modconf.webservice=ShoppingCart;
    region=EU
```

```
Require-Capability: modconf.webservice; filter:=“(&
        (modconf.webservice=ShoppingCart)
        (|(region=EU)(region=UK))
        )”
```

Problem

- Provide- and Require-Capabilities headers are tricky & error-prone to write by hand.
- bnd 3.3 will generate them from Java annotations:

```
@WebComponent(name = "cart-list")
public class CartComponent { … }

@RequireWebComponent(name = "cart-list")
public class ShoppingComponent { … }
```

Final Pieces

- A web application is not just a pile of components.
- Need a top-level app to bring them together.

```
@RequireWebComponent(name = “shop-browse”)
@RequireWebComponent(name = “cart-list”)
@RequireWebComponent(name = “shop-browse”)
@RequireWebComponent(name = “user-manager”)
public class ShopApp { … }
```

- NB: requires Java 8 repeating annotations, not supported (yet) in bnd.

Final Pieces

- Component libraries like Polymer are too big for individual capabilities.
- OSGi RFP-171 Web Resources – can be used for other assets e.g. Bootstrap, JQuery, Angular…
- Maps to predictable URL paths:

```
@RequirePolymerIronWebresource(resource = "*.html")
@RequirePolymerPaperWebresource(resource = "*.html")
@RequireBootstrapWebResource(resource="css/bootstrap.css")
public class ShopApp { … }
```

```
<link rel="stylesheet" type="text/css"
 href="/osgi.enroute.webresource/${bsn}/${Bundle-Version}/bootstrap-min.css">
```

Workshop

Requirements

- Eclipse – Neon recommended (Luna/Mars acceptable)
- Bndtools 3.3
- Java 8
- Google Chrome

Files

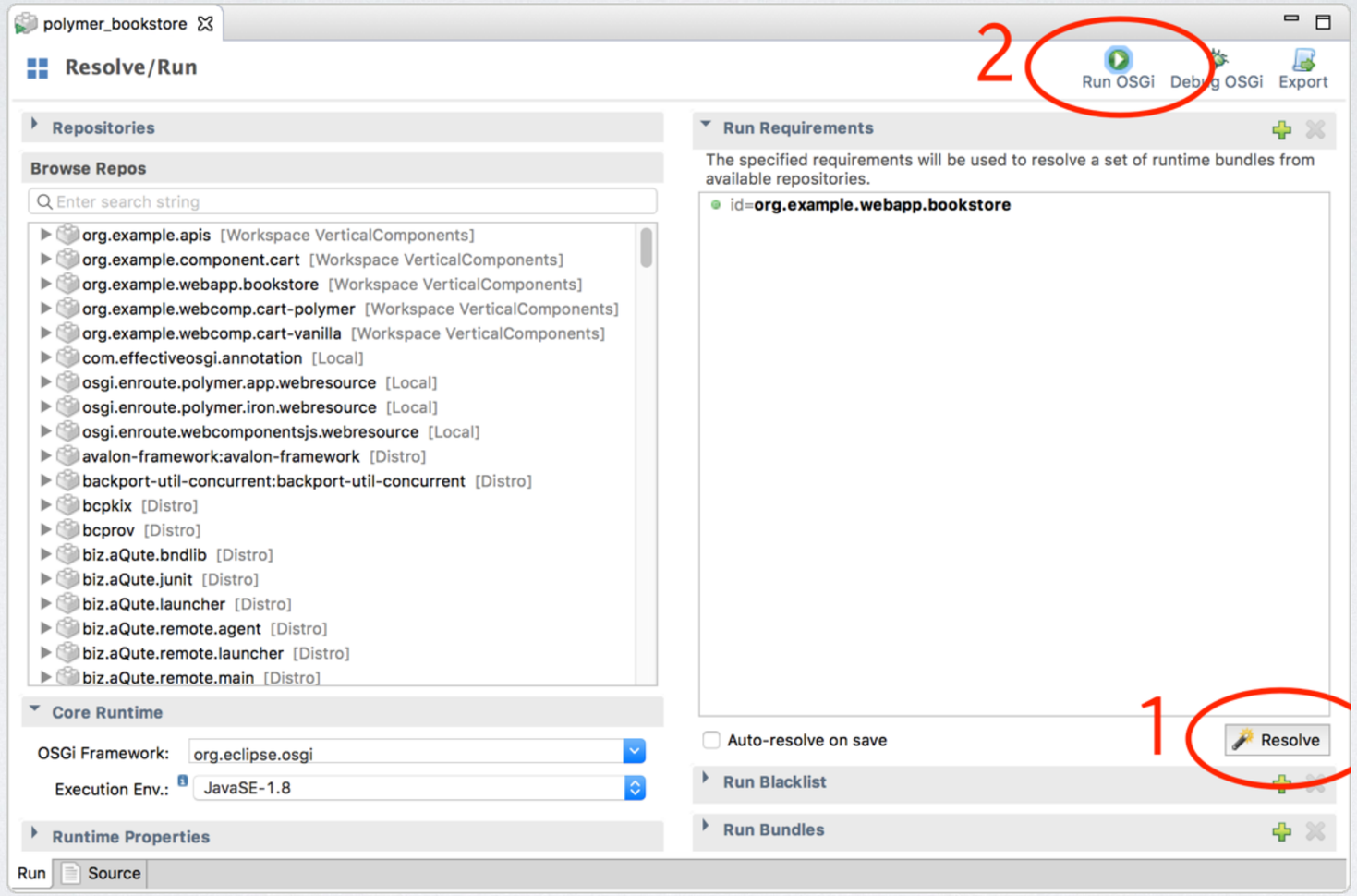
- `modconf-nbartlett-initial.zip`: Initial Workspace
- `modconf-nbartlett-stepN.zip`: Completed Exercise *N*
- `modconf-nbartlett-completed.zip`: Full Completed Workspace

- See also: `github.com/njbartlett/modconf2016-workshop`

Exercise 0: Setup

1. Start with empty Eclipse workspace.
2. File menu → Import → General → Existing Projects
3. “Select Archive”. Browse for initial.zip. Ensure all projects checked → Finish.
4. Open: org.example.webapp.bookstore/polymer_bookstore.bndrun
5. Resolve and Run.
6. Open in Chrome: http://localhost:8080/

Hint: Running the App



Exercise 1: Declare Web Component

1. Open org.example.webcomp.cart-vanilla/src/org/example/webcomp/cart/CartServlet.java. Add:

```
@WebComponent(name = "cart-list")
```

2. Examine static/cart-list/component.js for example of “vanilla JavaScript” Web Component.

Exercise 2: Add Web Comp. to App.

1. Add to BookstoreApp.java:

```
@RequireWebComponent(name = "cart-list")
```

2. Add to index.html inside <head>:

```
<script src="/cart-list/component.js"></script>
```

3. Add to index.html inside <body>:

```
<cart-list></cart-list>
```

4. Resolve then Run OSGi.

Exercise 3: Add 3rd Party Components

1. Add to BookstoreApp.java:

```
@RequirePolymerIronWebresource(resource = "*.html")
@RequirePolymerAppWebresource(resource = "*.html")
```

2. Add to index.html inside <head>:

```
<link rel="import" href="/osgi.enroute.webresource/${bsn}/${Bundle-Version}/iron-icons/iron-icons.html">
<link rel="import" href="/osgi.enroute.webresource/${bsn}/${Bundle-Version}/app-layout/app-header/app-header.html">
<link rel="import" href="/osgi.enroute.webresource/${bsn}/${Bundle-Version}/app-layout/app-toolbar
                        /app-toolbar.html">
```

3. Add to index.html inside <body>:

```
<app-header><app-toolbar>
    <iron-icon class="big" icon="icons:polymer"></iron-icon>
    <div title>Modconf Bookstore</div>
    <iron-icon icon="icons:menu" ></iron-icon>
</app-toolbar></app-header>
```

4. Resolve and Run.

Exercise 4: Declare Polymer Component

1. Open org.example.webcomp.cart-polymer/src/org/example/webcomp/cart/CartServlet.java. Add:

```
@WebComponent(name = "polymer-cart-list")
@RequirePolymerIronWebresource(resource = "*.html")
```

2. Examine static/cart-list/*.html for example of Polymer-based Web Component.

Exercise 5: Use Polymer Comp. in App.

1. Update BookstoreApp.java:

   ```
   @RequireWebComponent(name = "polymer-cart-list") // was “cart-list”
   ```

2. Update index.html inside <head>. Remove this line:

   ~~<script src=“/cart-list/component.js”></script>~~

3. … and add this line:

   ```
   <link rel="import" href="/cart-list/cart-list.html">
   ```

4. Resolve and Run.

Conclusion

Conclusion

- A promising model for front-to-back modularity.
- Perhaps a better approach to web packaging than existing tools: npm, Bower, Webpack etc?
- Annotations and runtime extenders are a work-in-progress.
- Ongoing development under OSGi enRoute and Effective OSGi.