AOP-ing your JavaScript

Brian Cavalier / @briancavalier
SpringJS at Pivotal / cujoJS Co-lead

Aspect Oriented Programming

- AOP refresher
- How to do it in JavaScript (hint: it’s easy)
- Application composition

AOP

- Program transformation that combines separate, and possibly unrelated, concerns
- Huh?

AOP

- Non-invasively augment or modify the behavior of existing code
- AOP is a composition strategy
- “Advice” is a common approach
  - before, around, afterReturning, afterThrowing, after

Stereotypical AOP Examples

- Logging
- Profiling
- Transaction boundaries
- Security

Composition strategies

- Inheritance
- Delegation
- AOP

Composition strategies

- Add profiling to all instances of class X
  - –Using inheritance breaks the “is-a” mental model
  - –Using inheritance means changing the code that creates instances of X to create instances of ProfiledX
  - –Using either delegation or inheritance -> must account for profiling code in unit tests!
- Add profiling to all instances of classes X, Y, and Z
  - –Multiply all above problems by 3

Composition strategies

- AOP can apply behavior *from the outside*
- controlled - guarantees about *not breaking your stuff*
- non-invasive - without changing the *source code*

Typical AOP approaches

- Can require some sophisticated machinery
- Source code transformation
- Byte code transformation
- Language-level Proxies
- VM or runtime support

AOP in JavaScript

AOP in JavaScript

- JavaScript doesn’t have Proxies*, byte code access, or VM level support for AOP
- Source code transformation
- AST transformation
- Or something *easier* ...

* ECMAScript 6 will have language-level Proxies

Method replacement

```
// Save the original function
var orig = thing.method;

// Replace it with one that does what we want
thing.method = function() {
     doAdditionalStuff();
     return orig.apply(this, arguments);
}
```

Method replacement

```
var orig = thing.method;

thing.method = function() {
    try {
        return orig.apply(this, arguments);
    } catch(e) {
        doAdditionalStuff(e);
        throw e;
    }
};
```

Method replacement

- Pros
  - Easy to implement
  - Dynamic - add and remove advice at runtime
- Cons
  - Changes the **`hasOwnProperty`** landscape
  - Harder to do app-wide weaving (e.g. classpath scanning and global pointcuts)

Examples

- Logging
  - https://github.com/briancavalier/aop-s2gx-2013/blob/master/examples/logging.js
- Profiling
  - https://github.com/briancavalier/aop-s2gx-2013/blob/master/examples/around.js
- Memoization
  - https://github.com/briancavalier/aop-s2gx-2013/blob/master/examples/around.js#L170

If it’s so easy ...

- why isn’t it more common in JS?
  - –Don't know AOP exists
  - –Apply AOP without knowing it
  - –Know about AOP, but don't know how to apply it in JS

AOP in JavaScript

- AOP in 50 LOC - https://github.com/briancavalier/aop-s2gx-2013/tree/master/src
- cujoJS's meld - https://github.com/cujojs/meld
- Dojo's dojo/aspect - http://dojotoolkit.org
- Twitter Flight - http://twitter.github.io/flight/) -
- javascript-hooker - https://github.com/cowboy/javascript-hooker)
- dcl - https://github.com/uhop/dcl

Neato, but yawn

- Guess what? Users don’t actually care about logging, profiling, or memoization.
- If that’s all we could do, this would be *lame*

Can we ...

- use this kind of approach to connect more interesting things together?
- What about Views, Controllers, Models, or *any* application components?

Application composition

- Connecting reusable components together to make a particular application
- Now *that* sounds useful
- It also sounds a lot like AOP: "composing units of behavior"

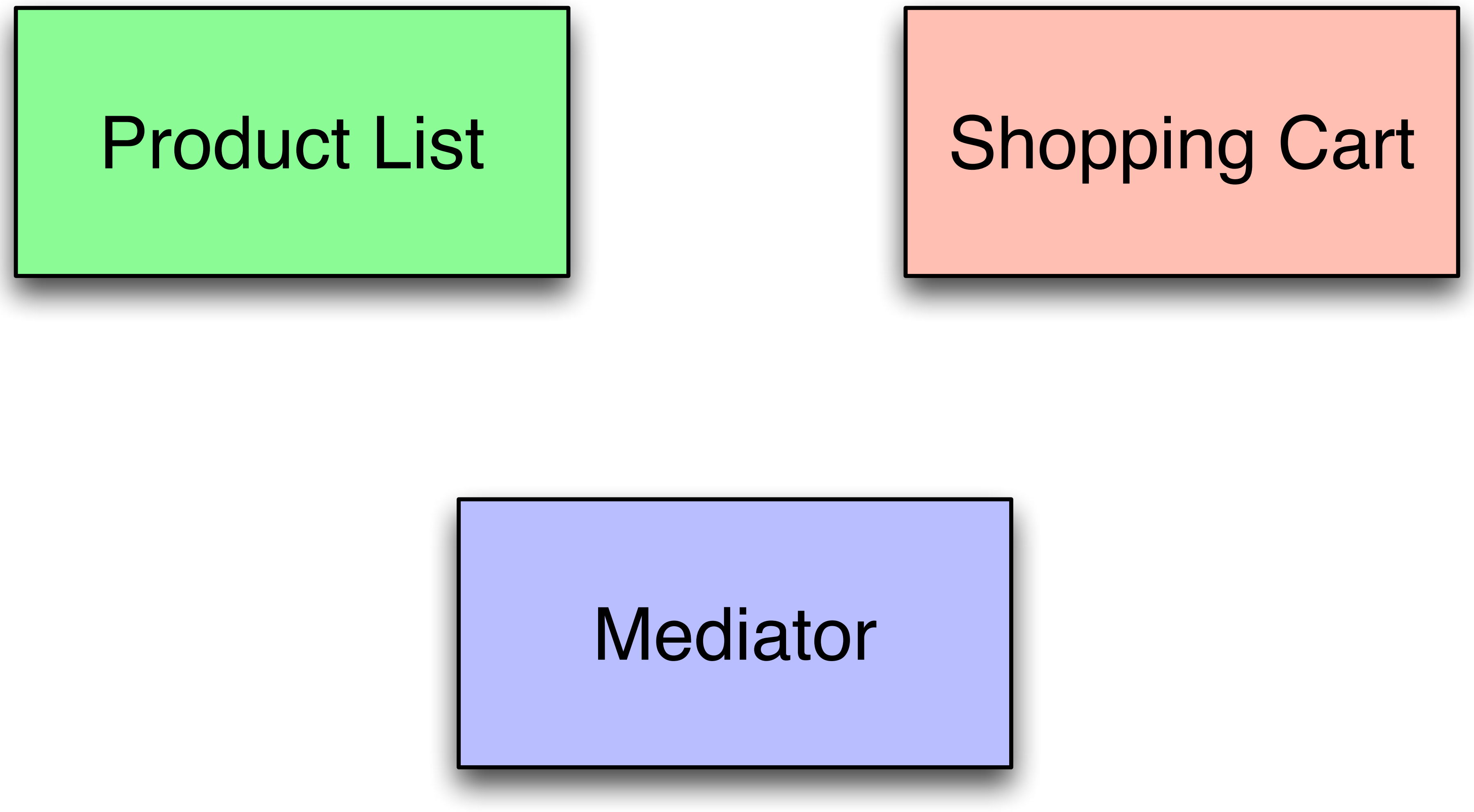
Let’s make a simple app

- Product list and shopping cart
- https://github.com/briancavalier/aop-s2gx-2013/tree/master/demo-app

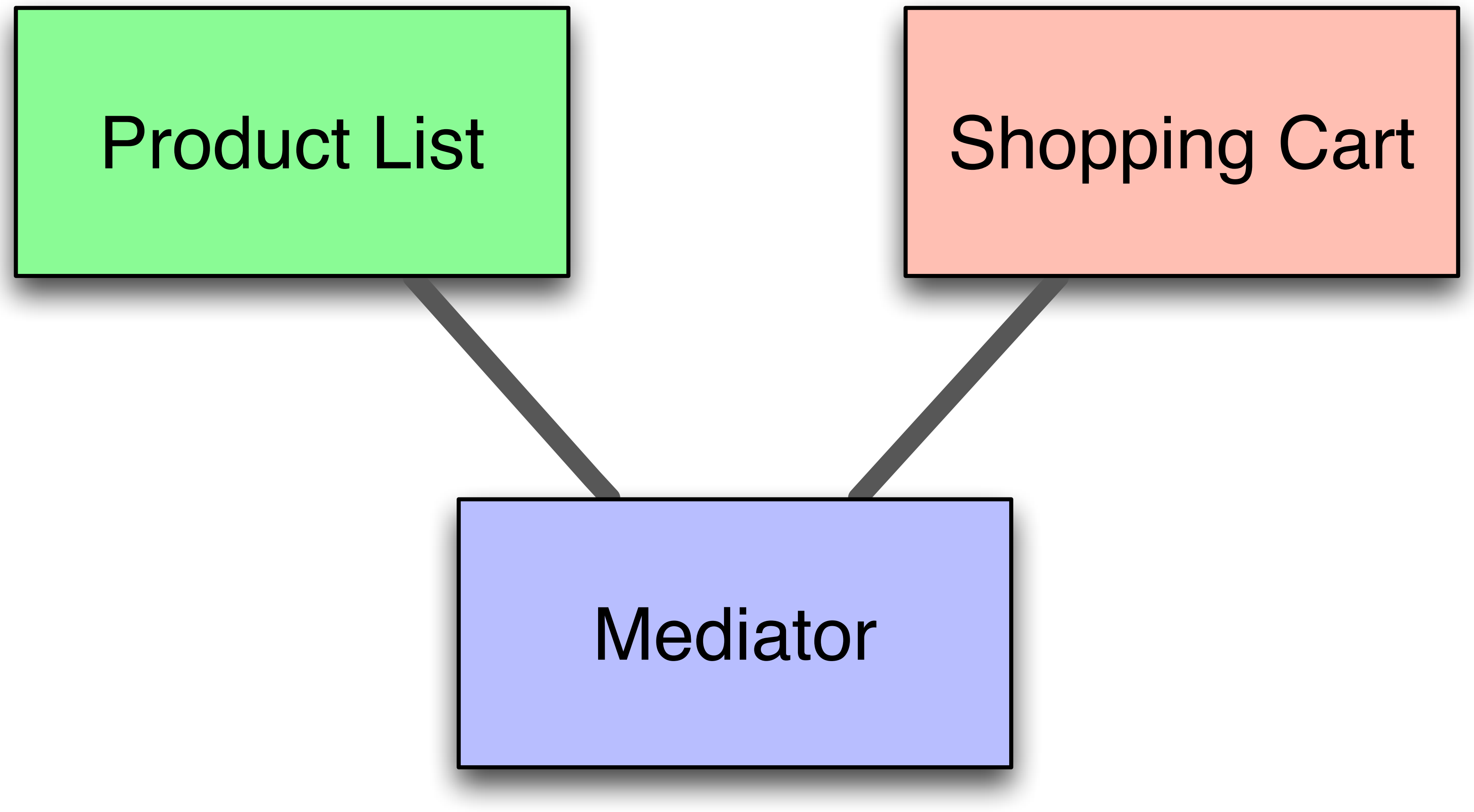
Let’s make a simple app

Product List
Shopping Cart

Let’s make a simple app



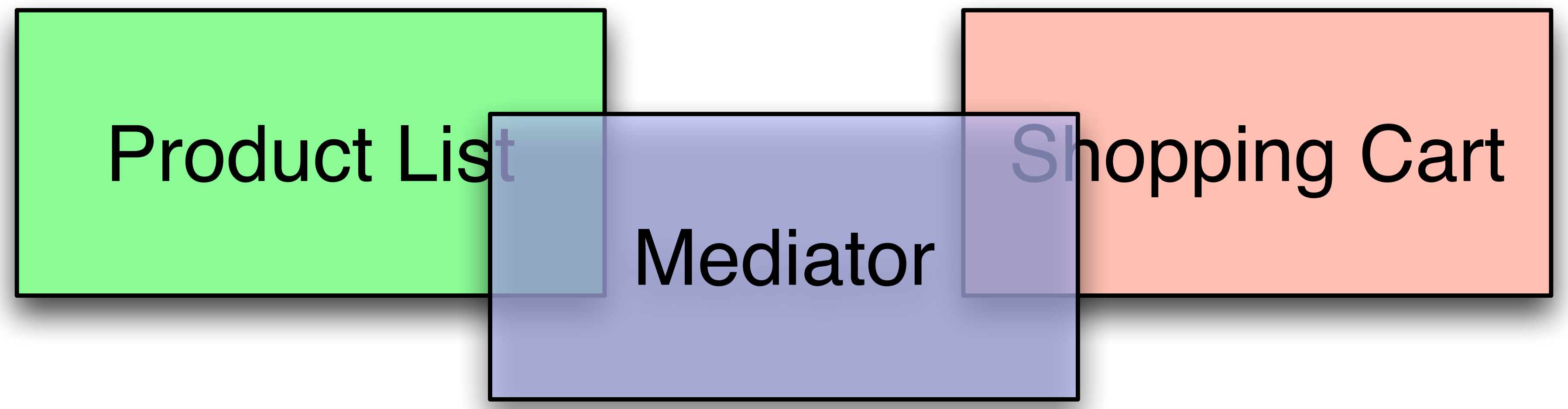
Let’s make a simple app



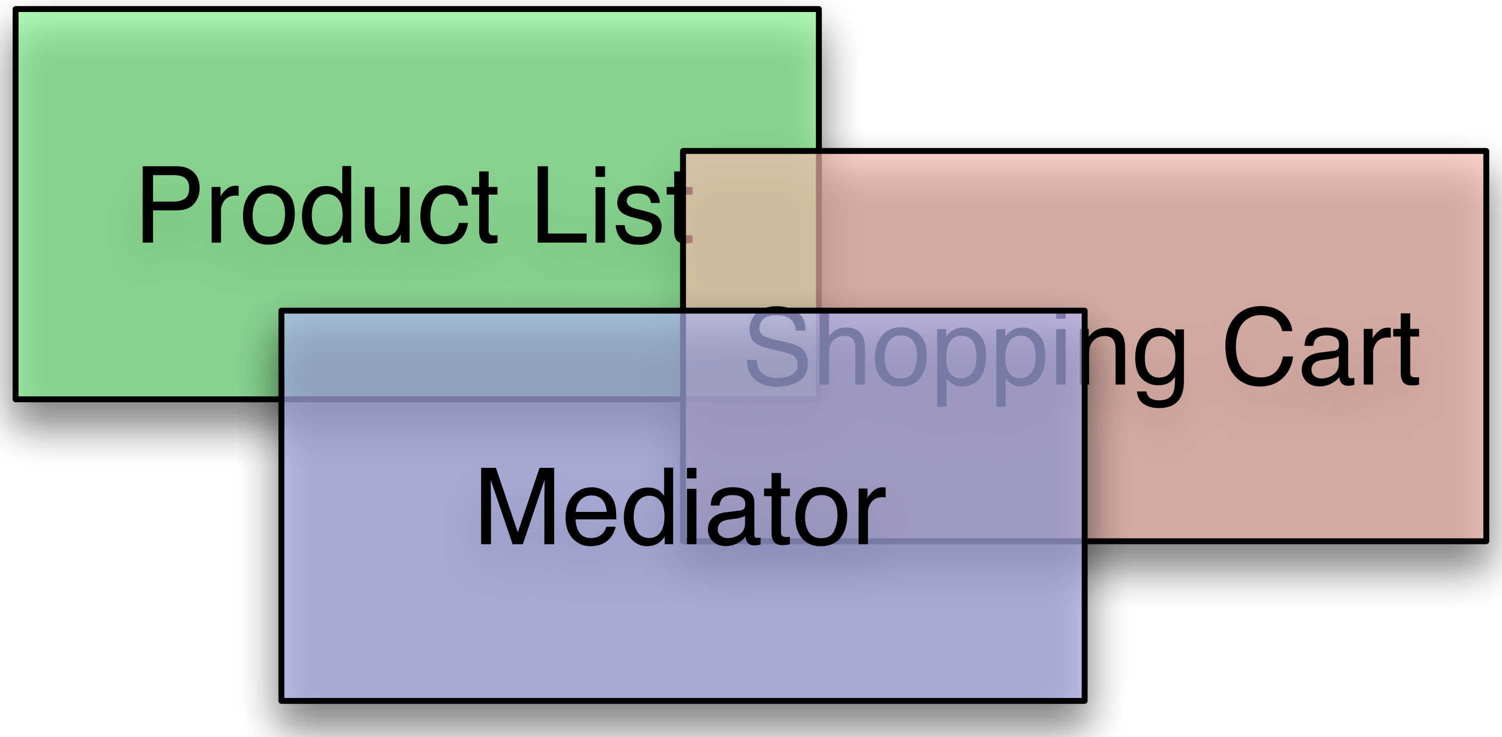
Let’s make a simple app

- Delegation - https://github.com/briancavalier/aop-s2gx-2013/blob/master/demo-app/vanilla
- Events - https://github.com/briancavalier/aop-s2gx-2013/blob/master/demo-app/events
- Pubsub - https://github.com/briancavalier/aop-s2gx-2013/blob/master/demo-app/pubsub

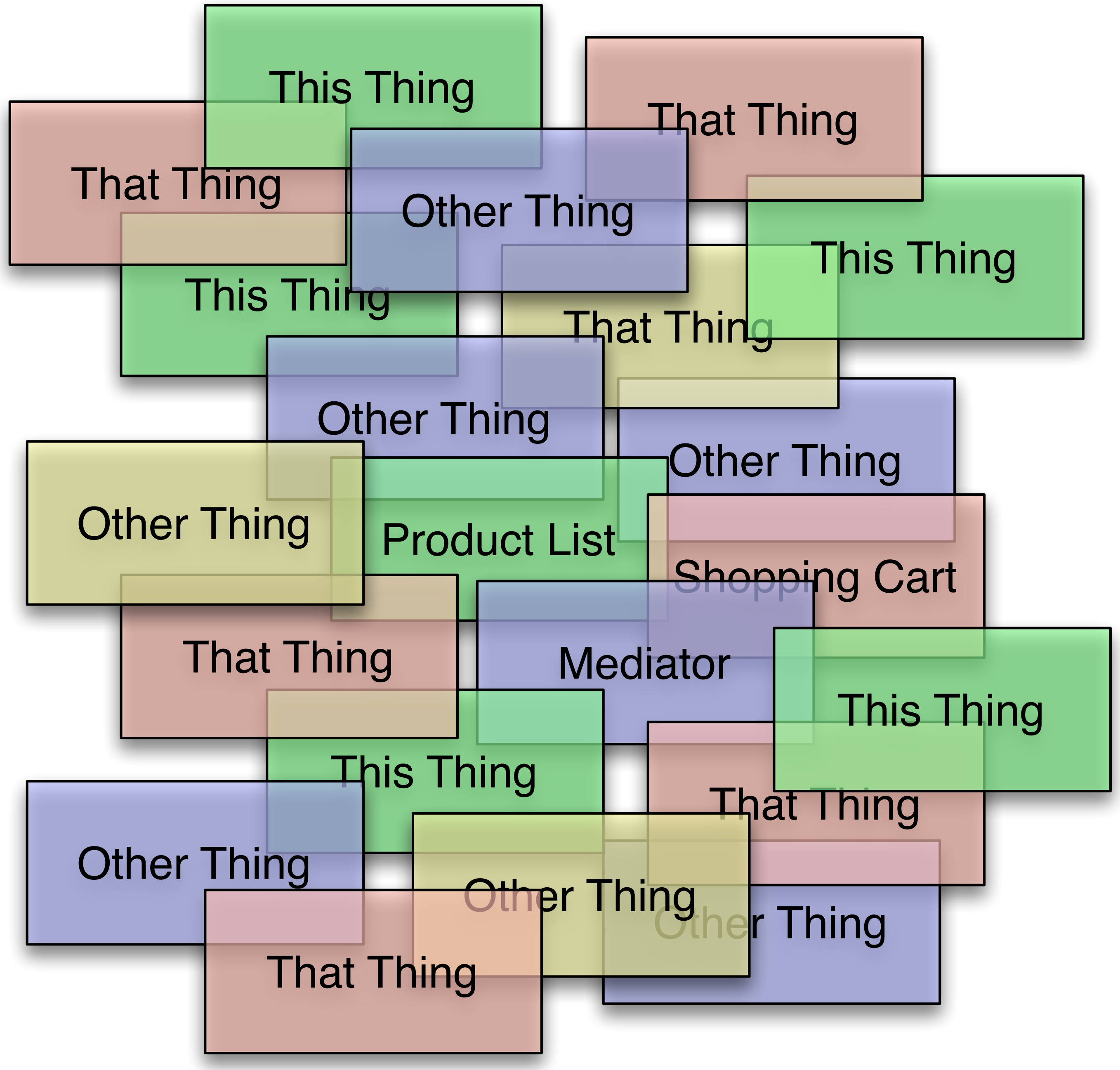
Coupled



Inseparable



Noooooo!


This Thing
That Thing
That Thing
Other Thing
This Thing
This Thing
That Thing
Other Thing
Other Thing
Other Thing
Product List
Shopping Cart
That Thing
Mediator
This Thing
This Thing
That Thing
Other Thing
Other Thing
Other Thing
That Thing

Bad

- Components coupled directly to each other, or directly to a connection lib API, or both.

Bad

- Lots of mocking to unit test
- Components easily break one another
- Adding new components -> changing source code of existing components
- Changing one component may require
  - updating many mocks
  - re-unit testing all components!

Application Composition

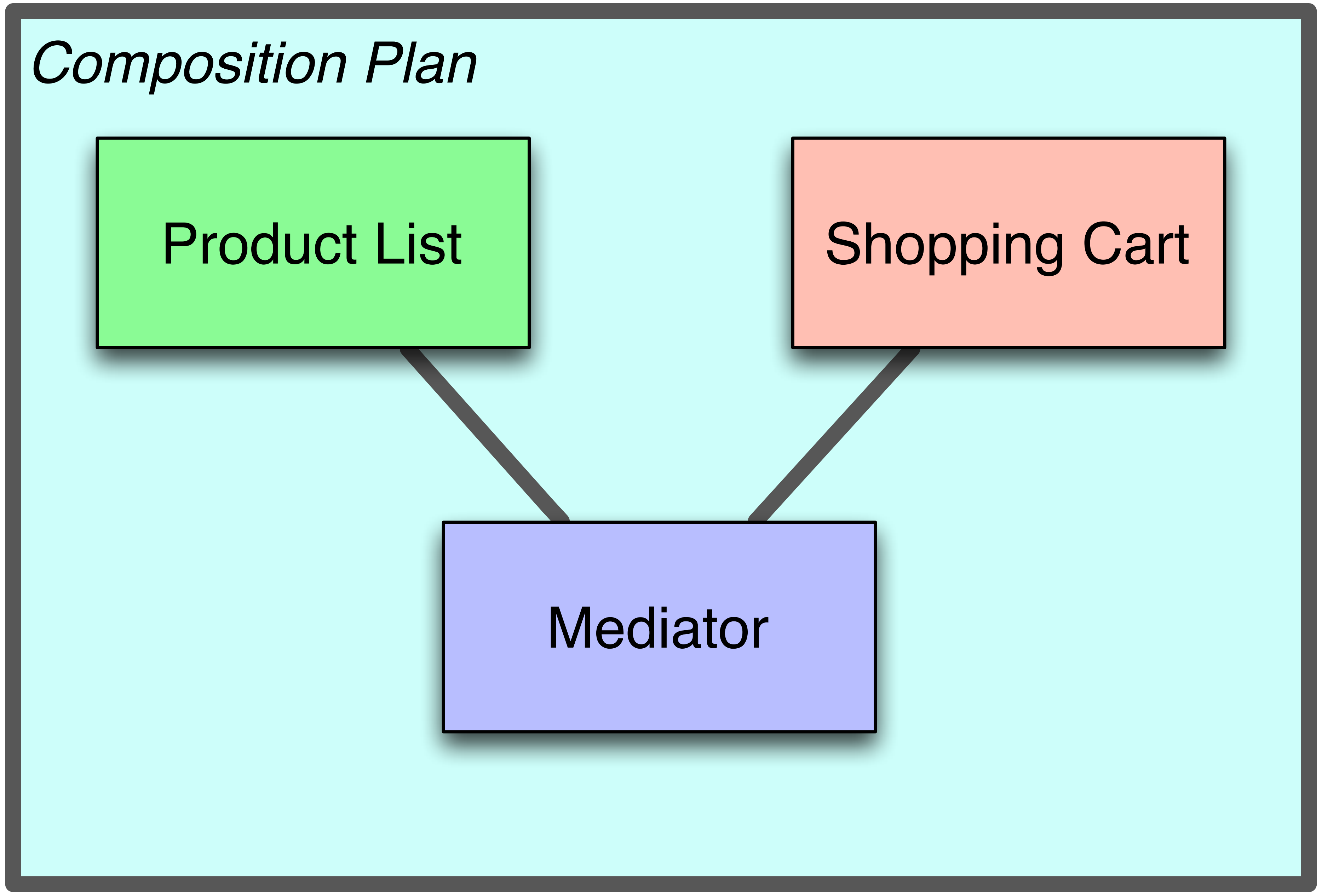
Application composition

- The act of connecting components together to make a complete application
- Often a separate, and *very different activity* than implementing the stuff inside components

Composition plan

- A *dedicated place* to compose application components
- It *owns the lines* in your box and line diagrams
- Example: Spring Application Context

Composition plan



Composition plan

- Let’s re-make our app using AOP and composition
- Simple AOP - https://github.com/briancavalier/aop-s2gx-2013/blob/master/demo-app/aop-simple
- meld AOP - https://github.com/briancavalier/aop-s2gx-2013/blob/master/demo-app/aop-meld

Good

- Components have no knowledge of each other
  - –unit tests are easy, less mocking
- Change the plan w/o changing the components' source
  - –no need to re-run unit tests
- Add new behavior to existing applications
  - –minimize regressions
- Create a new plan (i.e. app variant) easily
  - –build faster

Composition

- If we're always connecting components in similar ways, can we create a *DSL* to do it?

Yes

- Let’s re-make our simple app again
- cujoJS 1 (w/Controller) - https://github.com/briancavalier/aop-s2gx-2013/tree/master/demo-app/cujojs-1
- cujoJS 2 (Controller-less) - https://github.com/briancavalier/aop-s2gx-2013/tree/master/demo-app/cujojs-2

AOP

- Add/modify behavior
- Compose components
- Controlled, non-invasive
- Don't need a lib, but they help!

Application composition

- Separate connection from components
- Make a composition plan
- Test & refactor components easily
- Reduce collateral damage
- Build faster

Links - AOP

- AOP @ Wikipedia: http://en.wikipedia.org/wiki/Aspect-oriented_programming
- Spring AOP: http://static.springsource.org/spring/docs/2.5.5/reference/aop.html
- meld docs: https://github.com/cujojs/meld/blob/master/docs/TOC.md

Links - AOP in JavaScript

- AOP in 50 LOC - https://github.com/briancavalier/aop-s2gx-2013/tree/master/src
- cujoJS's meld - https://github.com/cujojs/meld
- Dojo's dojo/aspect - http://dojotoolkit.org
- Twitter Flight - http://twitter.github.io/flight/) -
- javascript-hooker - https://github.com/cowboy/javascript-hooker)
- dcl - https://github.com/uhop/dcl

Links - Application composition

- cujoJS wire - http://github.com/cujojs/wire
- Other JS IOC containers popping up recently

Links - Examples

- Examples from this talk - http://github.com/briancavalier/aop-s2gx-2013
- cujoJS.com - http://cujojs.com
- cujoJS sample apps - http://know.cujojs.com/samples

Learn More. Stay Connected.

Talk to us on Twitter: @springcentral

Find session replays on YouTube: spring.io/video