



Advanced Case Study:

Building The Allurent Display Architecture

Joe Berkovitz
VP Engineering
Allurent, Inc.

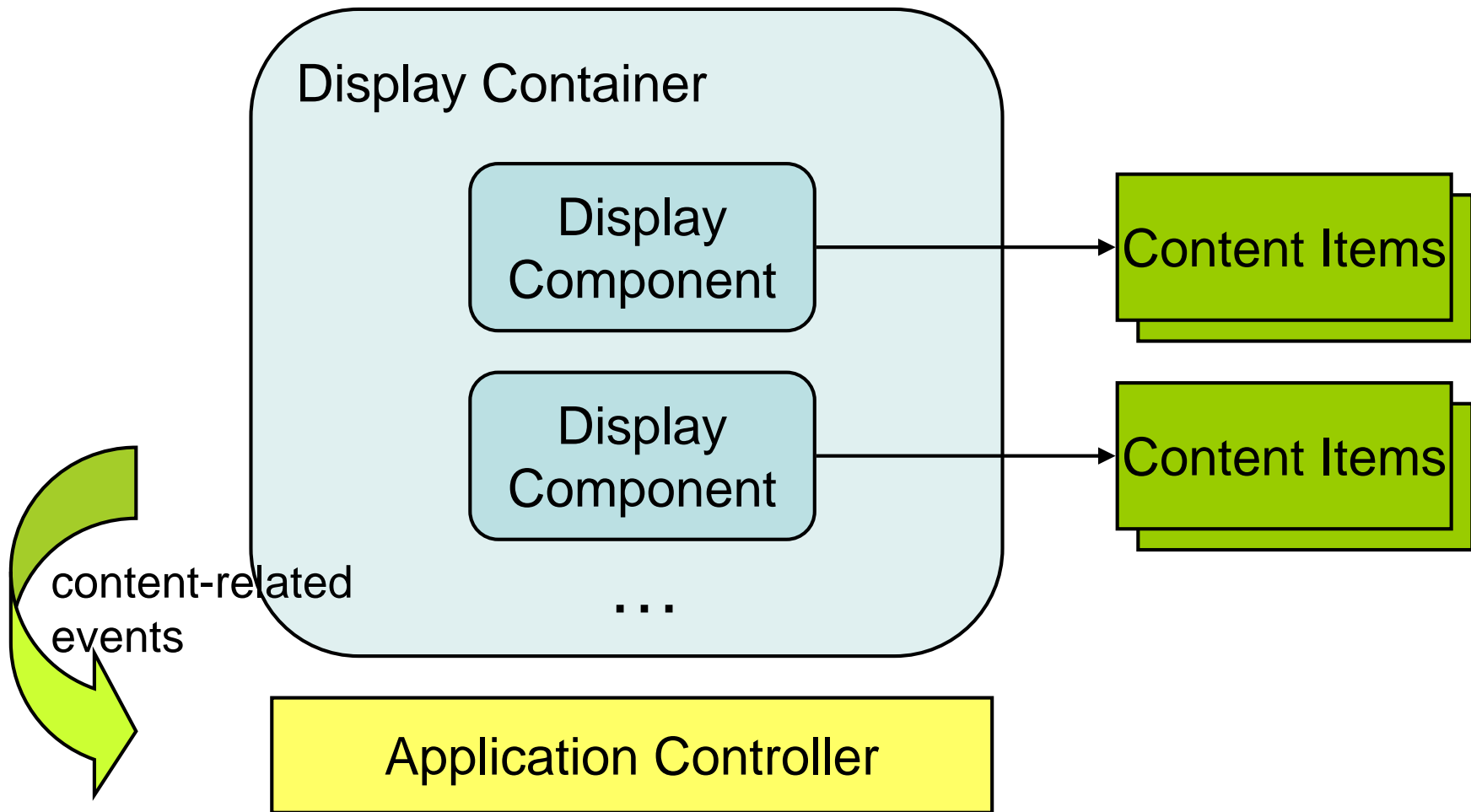


360|Flex Atlanta

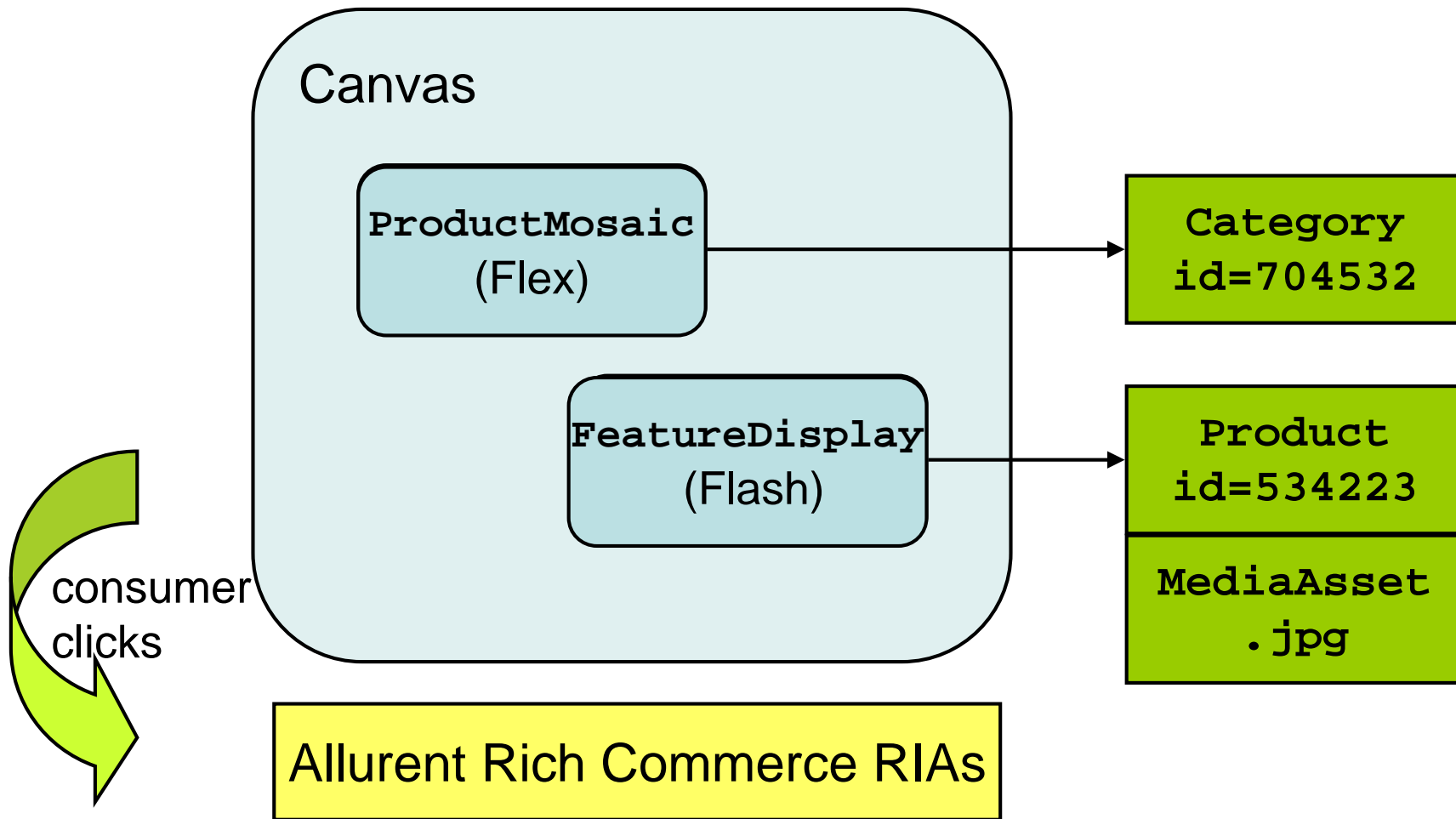
Overview

- What is Allurent Display?
- Why is it worth talking about here?
- Brief Demonstrations
- For each major subsystem:
 - Flex Framework Highlights
 - Architecture and Implementation Ideas

Allurent Display In Essence



In more concrete terms...



Allurent Display Demonstration (Consumer and Authoring)

How to Represent Displays?

- An obvious idea: MXML
- Advantages:
 - Simple syntax
 - Known quantity and spec
 - Interpret at design time
 - Compile for production

How to Represent Displays?

- MXML has disadvantages, though...
 - Allows complex programming/binding that is "indigestible" for a visual editor
 - Event handling requires yet more code
 - Hard to represent content or media as pure "logical references"
 - No on-demand loading of components

ASML: The Display Markup Language

```
<mx:Canvas width="765" height="540"
  xmlns:asml="..." xmlns:mx="..." xmlns:flexref="..."
  xmlns:flashref="...">

  <flexref:ProductMosaic width="700" height="302"
    x="10" y="10">
    <asml:Content
      asml:uri="arc://catalog/category/1234"/>
  </flexref:ProductMosaic>

  <flashref:FeatureDisplay x="0" y="0">
    <asml:Content
      asml:uri="arc://catalog/product/71561"
      asml:image="arc://media/italianbed.jpg"/>
  </flashref:FeatureDisplay>

</mx:Canvas>
```

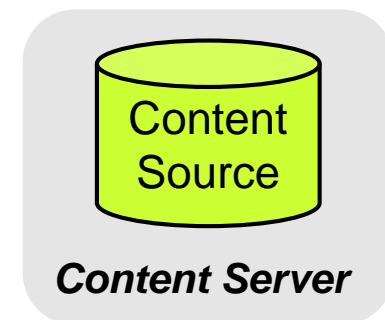
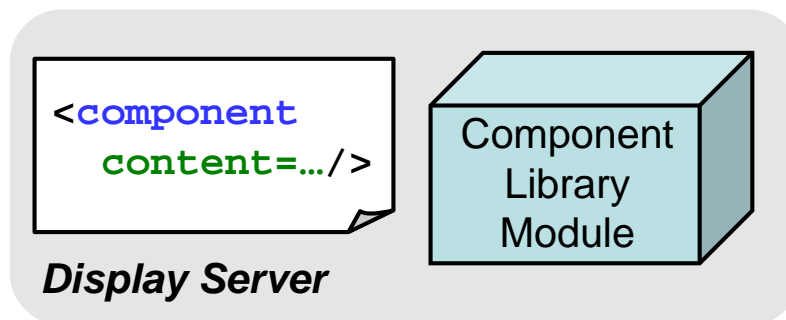
ASML Highlights

- Component instantiation and containment just like MXML
- Explicit constructs for content and media references
- All references are logical, not physical!
- No scripting allowed, ever
- All events dispatched through publish/subscribe system to listening Controllers

A System Interaction View...

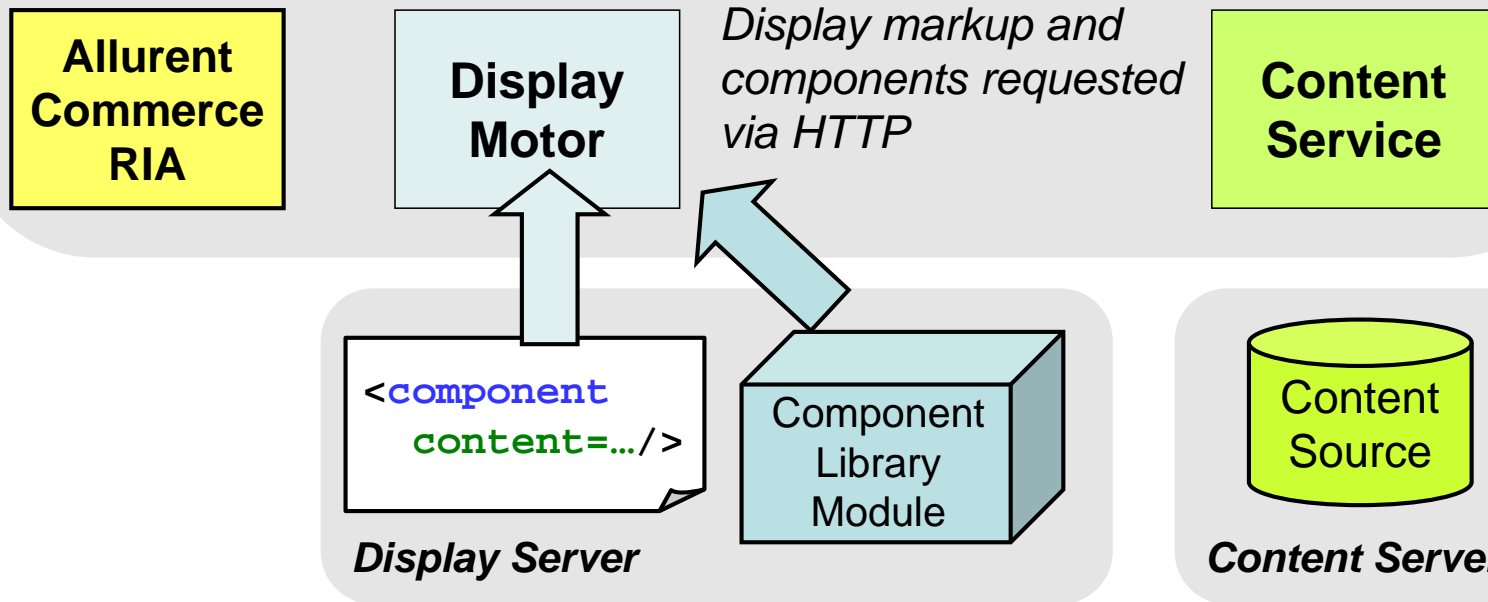
Flash Player/Client

Application requests rendering of Allurent Display markup

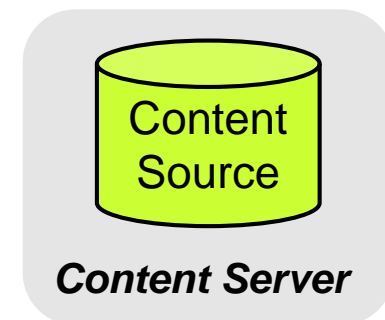
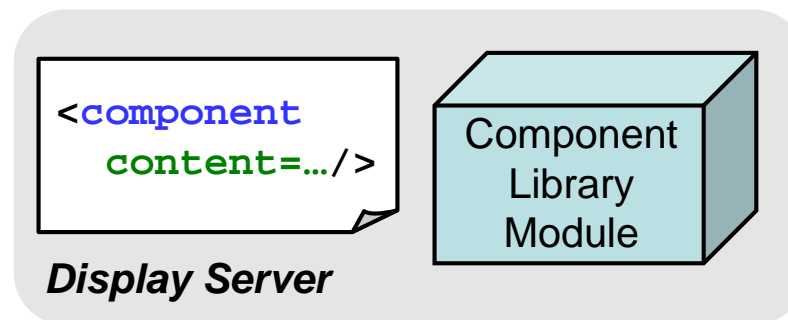
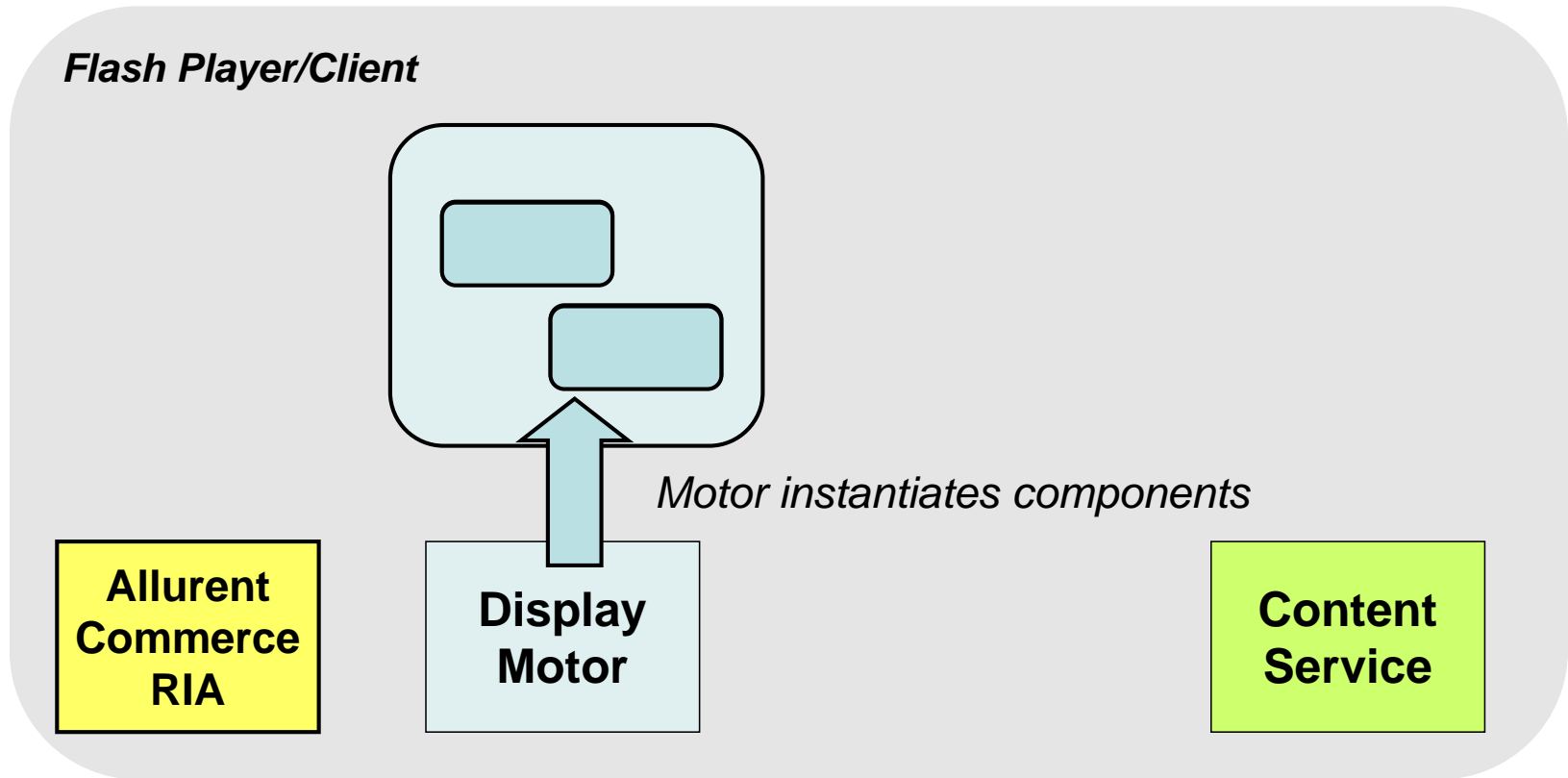


A System Interaction View...

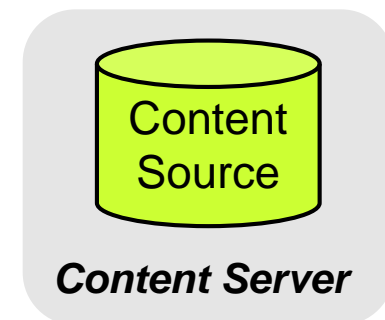
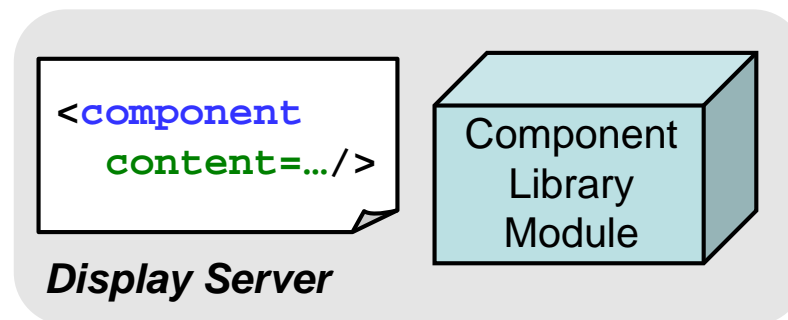
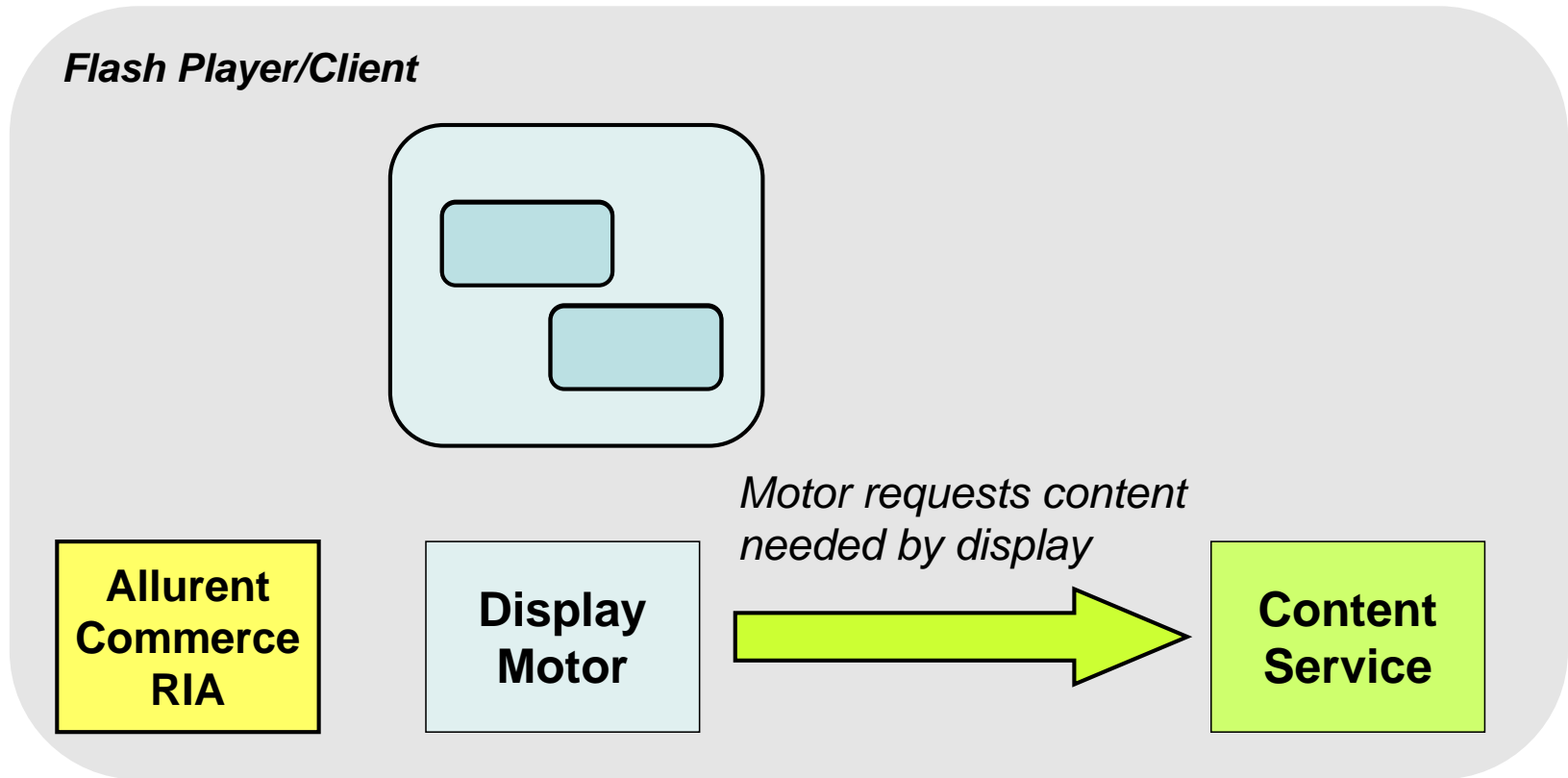
Flash Player/Client



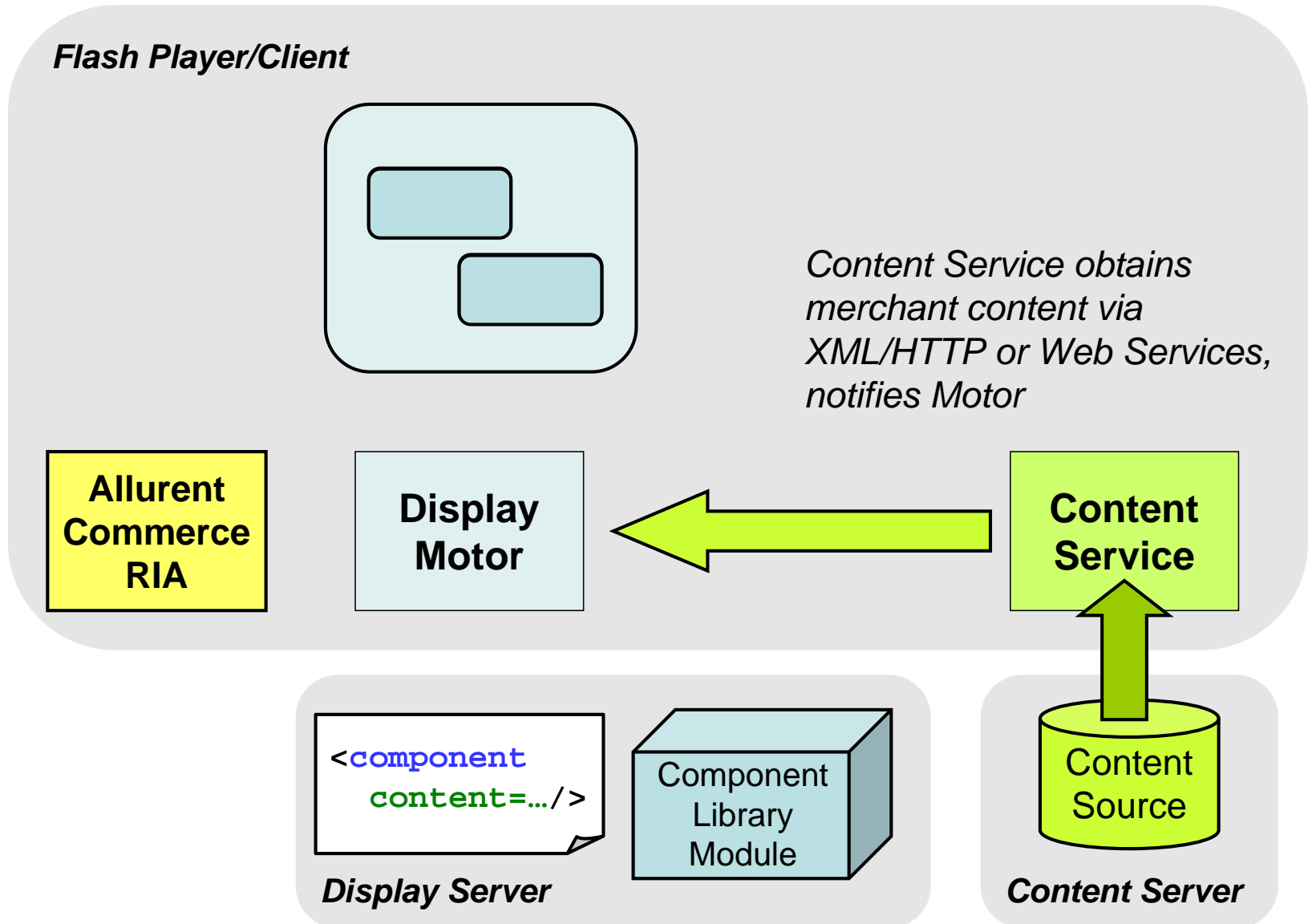
A System Interaction View...



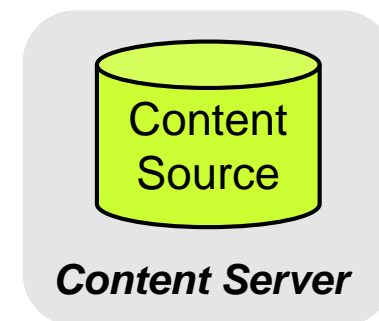
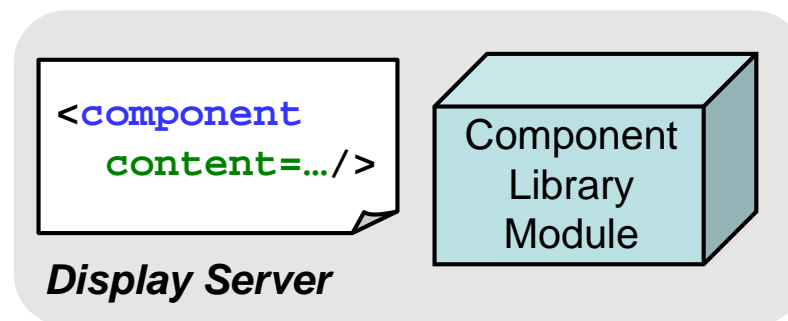
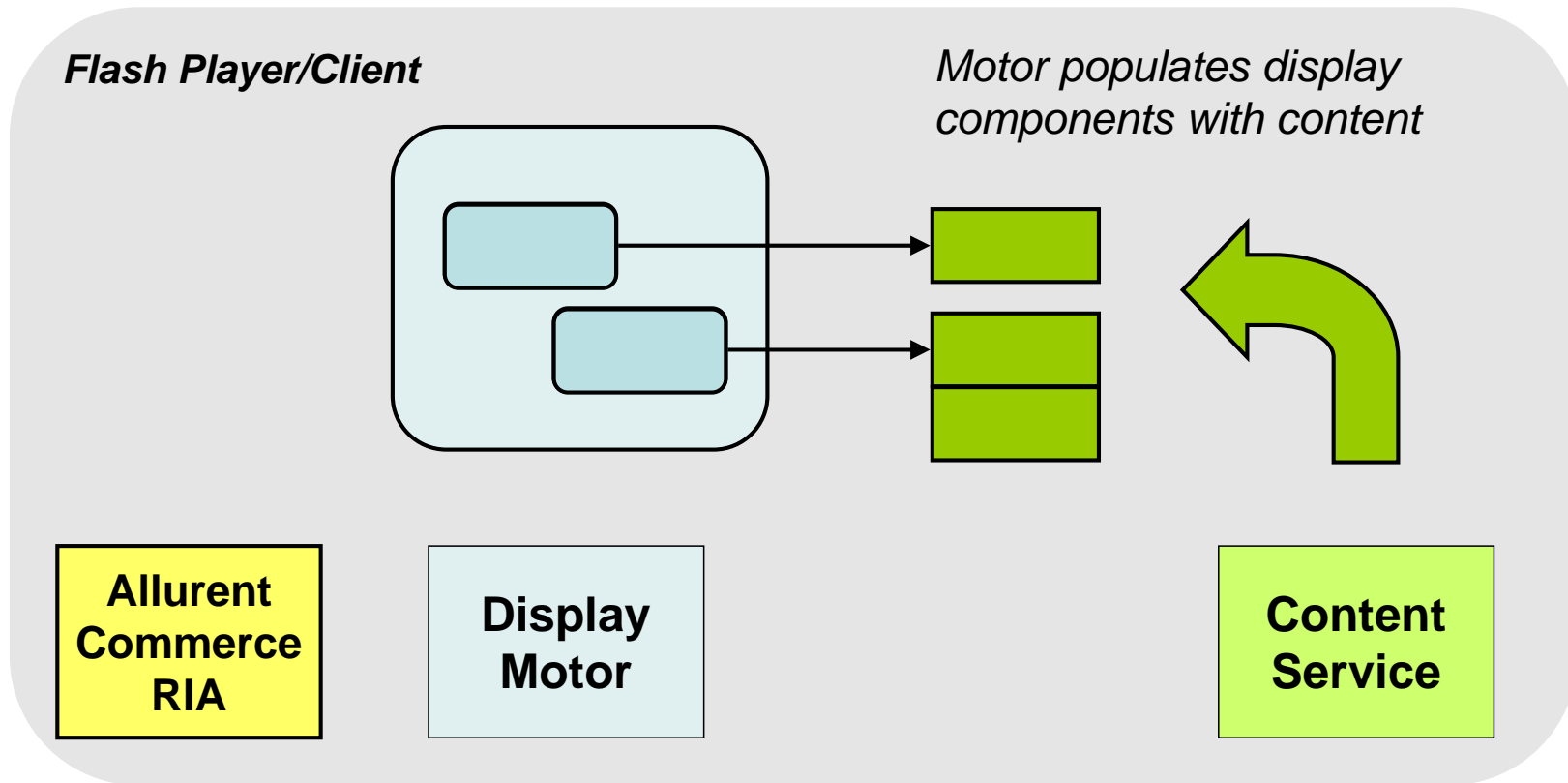
A System Interaction View...



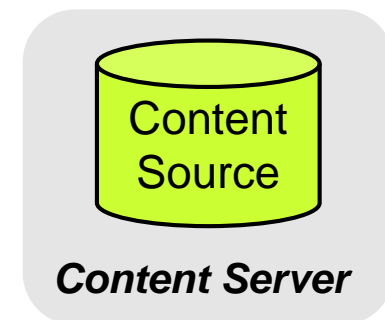
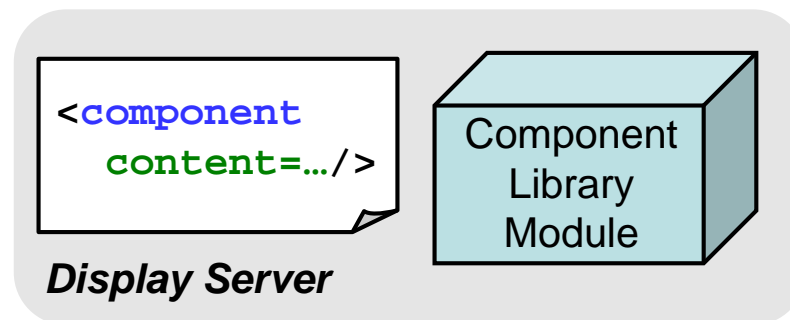
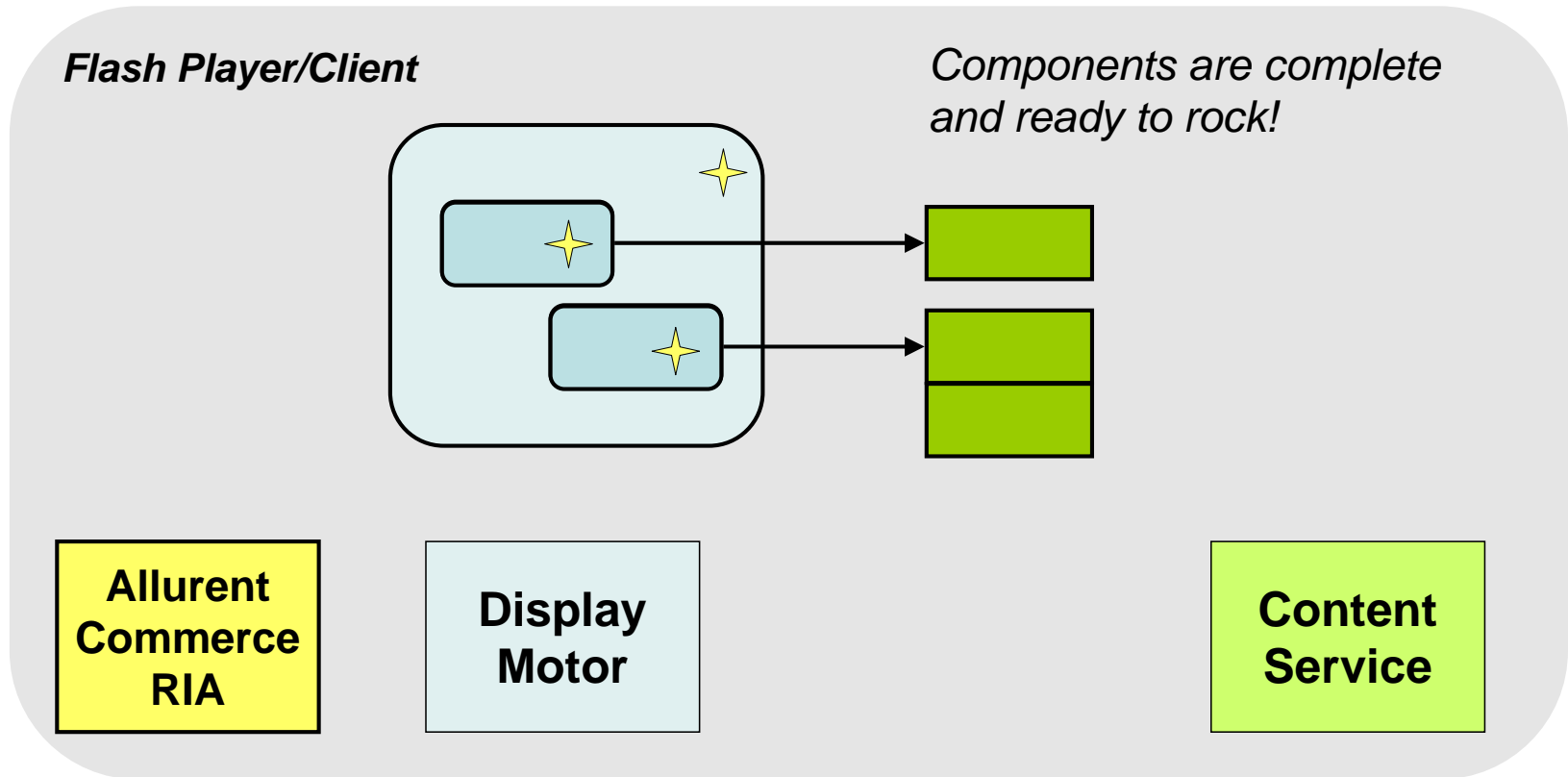
A System Interaction View...



A System Interaction View...



A System Interaction View...



Major Subsystems

- Display Motor
- Visual Merchandiser authoring tool
- Component Libraries
- Content Service

Display Motor: Design Drivers

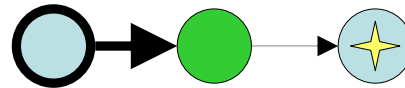
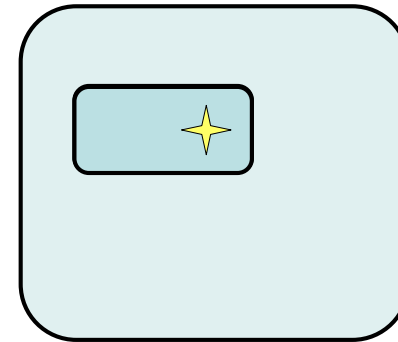
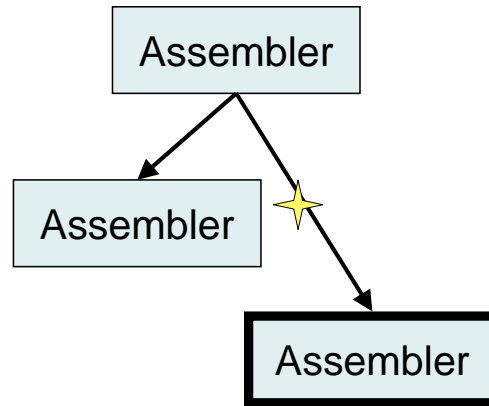
- Simple XML syntax, easy to parse
- Each component requires individual assembly, based on its markup and component metadata
- *Many* asynchronous aspects possible:
 - Loading of ASML markup document
 - Loading the Module containing a component's code and metadata
 - Loading content for a component
 - Delayed completion of child components
- Components *cannot* include code for this junk!

Display Motor: Basic Flow

- Build a graph of Assembler objects based on XML nodes in markup, using top-down parser. Each one manages a single component.
- An Assembler has its own state machine with transitions and conditions. The `assemble()` function proceeds from a "start" state to "finish", working (instantiate component, load its content) until it blocks.
- When an Assembler is blocked, it sets up a callback. All callbacks are the same function: `assemble()`! This function continues work until completion or another blocking wait.
- A completed Assembler informs its parent.

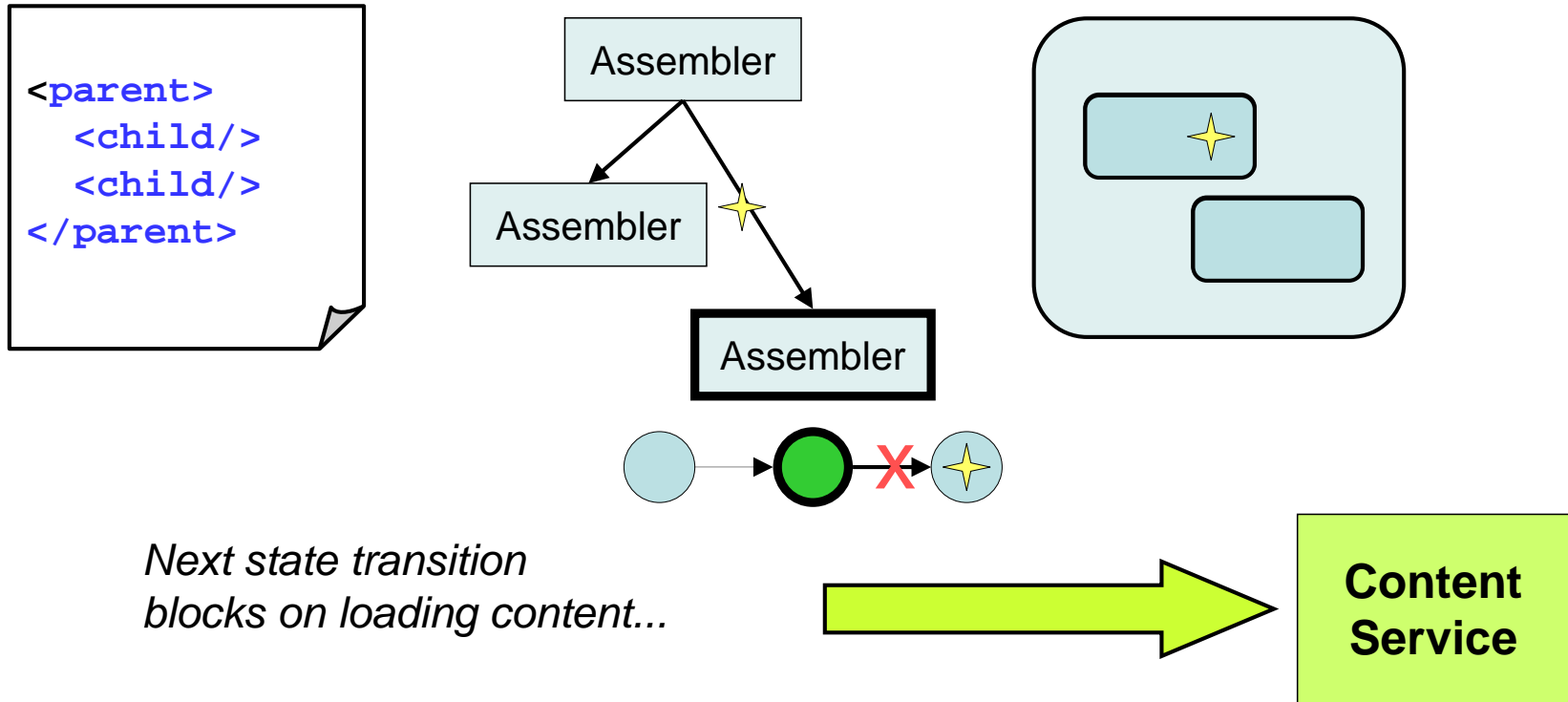
Motor Flow

```
<parent>  
  <child/>  
  <child/>  
</parent>
```

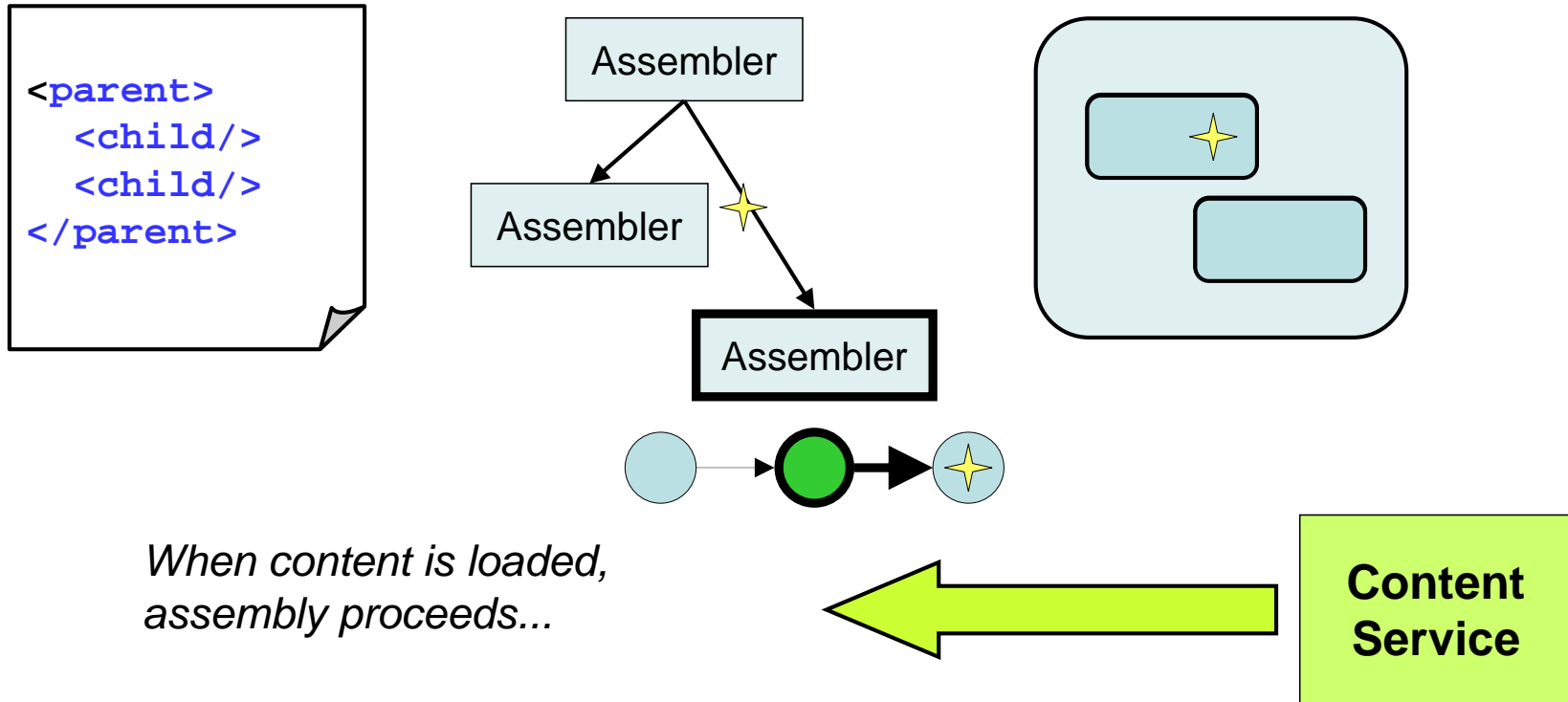


*Initial state transition
instantiates the component...*

Motor Flow

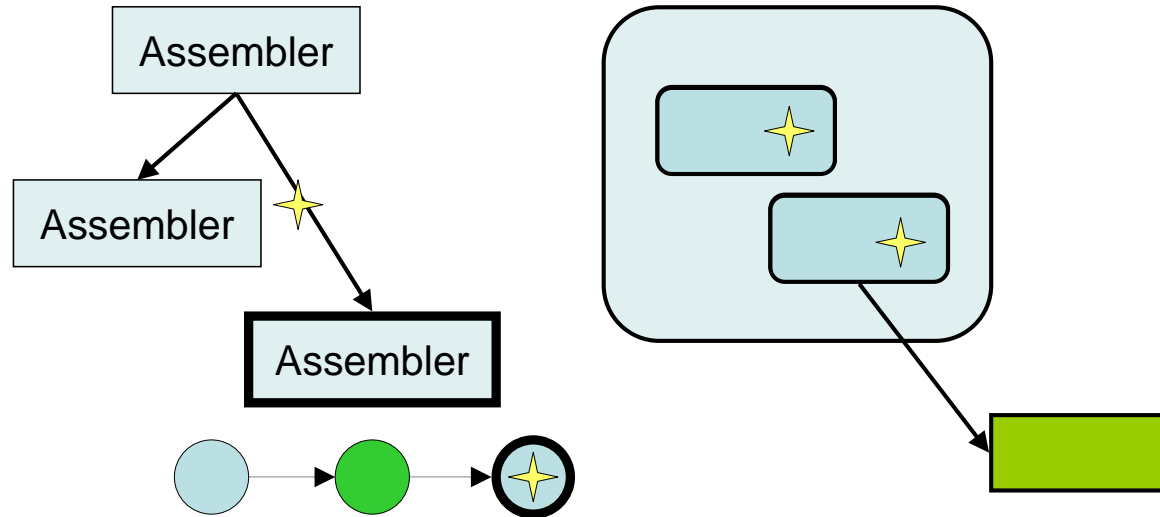


Motor Flow



Motor Flow

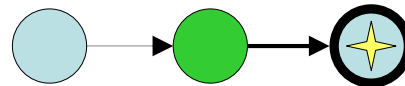
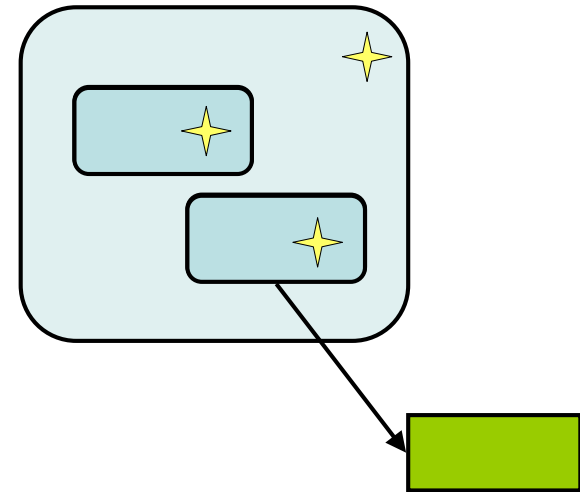
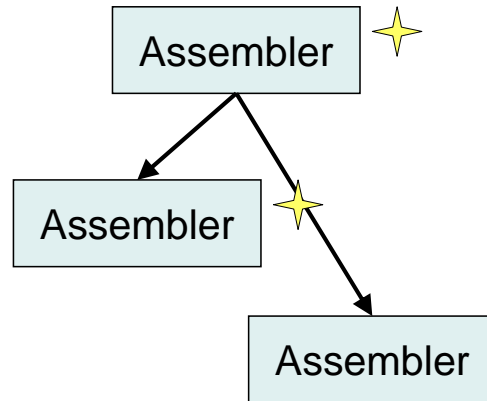
```
<parent>  
  <child/>  
  <child/>  
</parent>
```



*Component is complete,
notifies its parent...*

Motor Flow

```
<parent>  
  <child/>  
  <child/>  
</parent>
```



Finally, entire document is complete!

Stateful Continuation Pattern

```
public function assemble(e:Event = null):void {
    while (currentState != finalState)
        if (!gotoNextState())
            return;    // we're blocked, wait to restart
    dispatchEvent(new Event(Event.COMPLETE));
}
private function gotoNextState():Boolean {
    for each (var t:Transition in currentState.transitions)
        if (t.actionFunction.apply(this, [])) {
            currentState = t.targetState;
            return true;
        }
    return false;
}
private function someAction():Boolean {
    if (needToLoadSomething) {
        someLoader.load(something);
        someLoader.addEventListener(LoadEvent.DONE, assemble);
        return false;
    }
    return true;
}
```

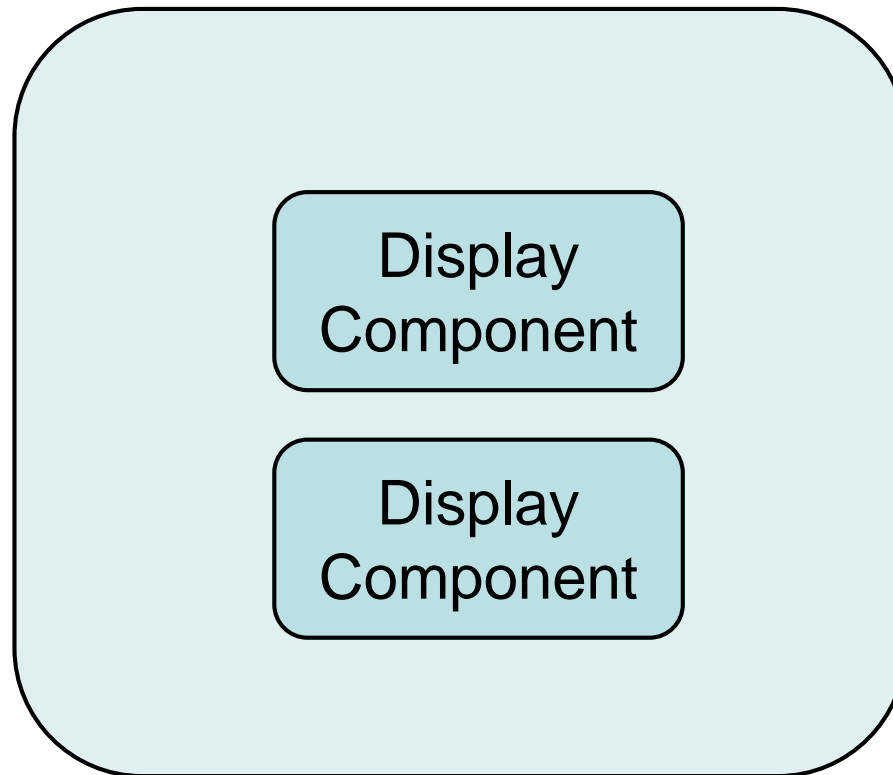
Motor: Flex and AS3 Highlights

- **URLLoader/URLRequest**: lightest-weight vehicle for loading XML markup from server
- **E4X**: perfect for parsing a simple XML dialect
- **Programmatic instantiation** of Flex components
- **Stateful Continuation** approach to asynchronous work

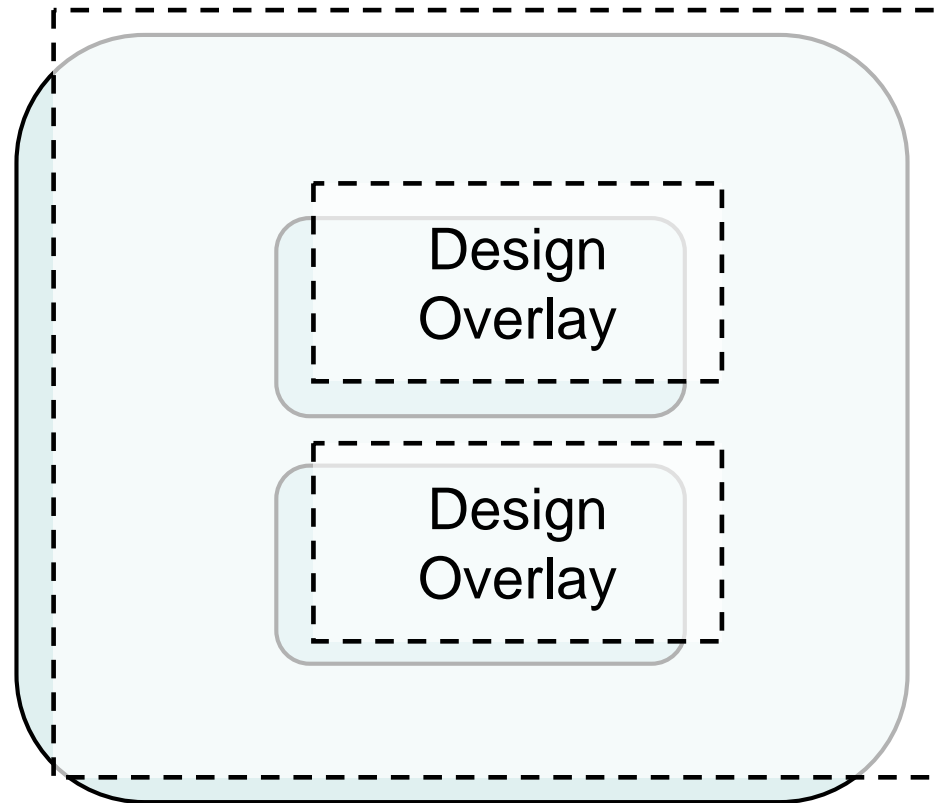
The Visual Merchandiser

- Authoring environment for Allurent Display
- A visual ASML editor
- 100% faithful to what the consumer sees
- A simple tool -- not Flash, not Photoshop
- Uses actual merchant content, not placeholders
- NOT FlexBuilder Design View or Thermo --
business audience

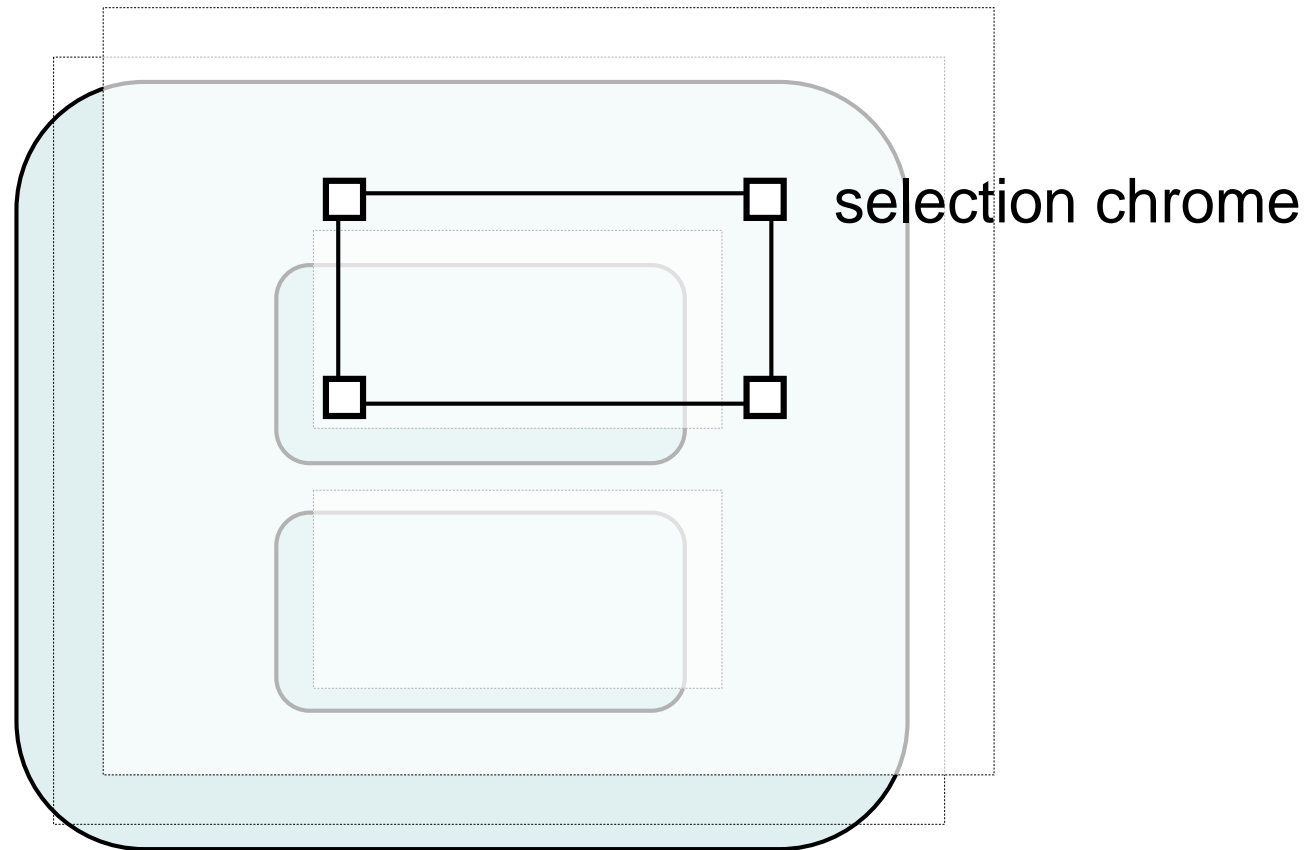
Design View Layers: WYSYWIG rendering by Motor



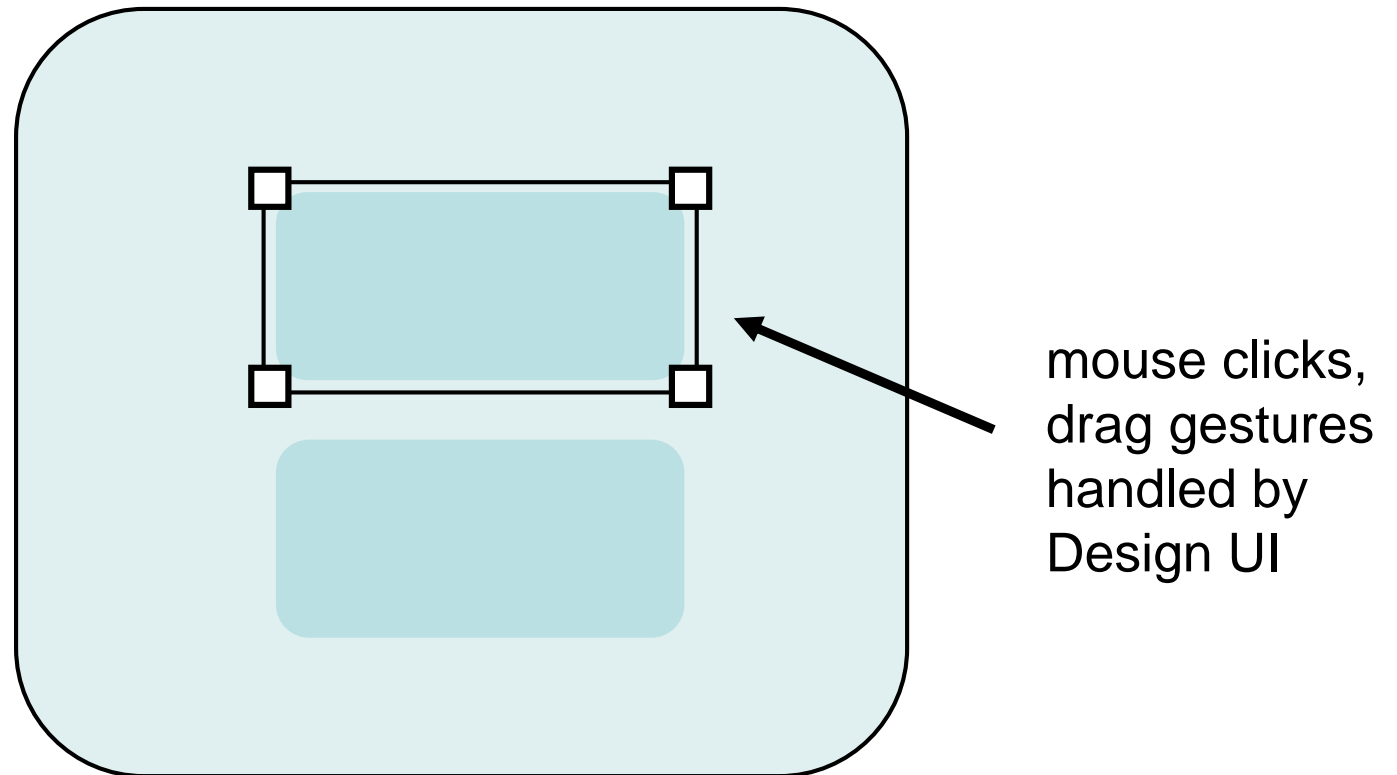
Design View Layers: Transparent Design Overlays



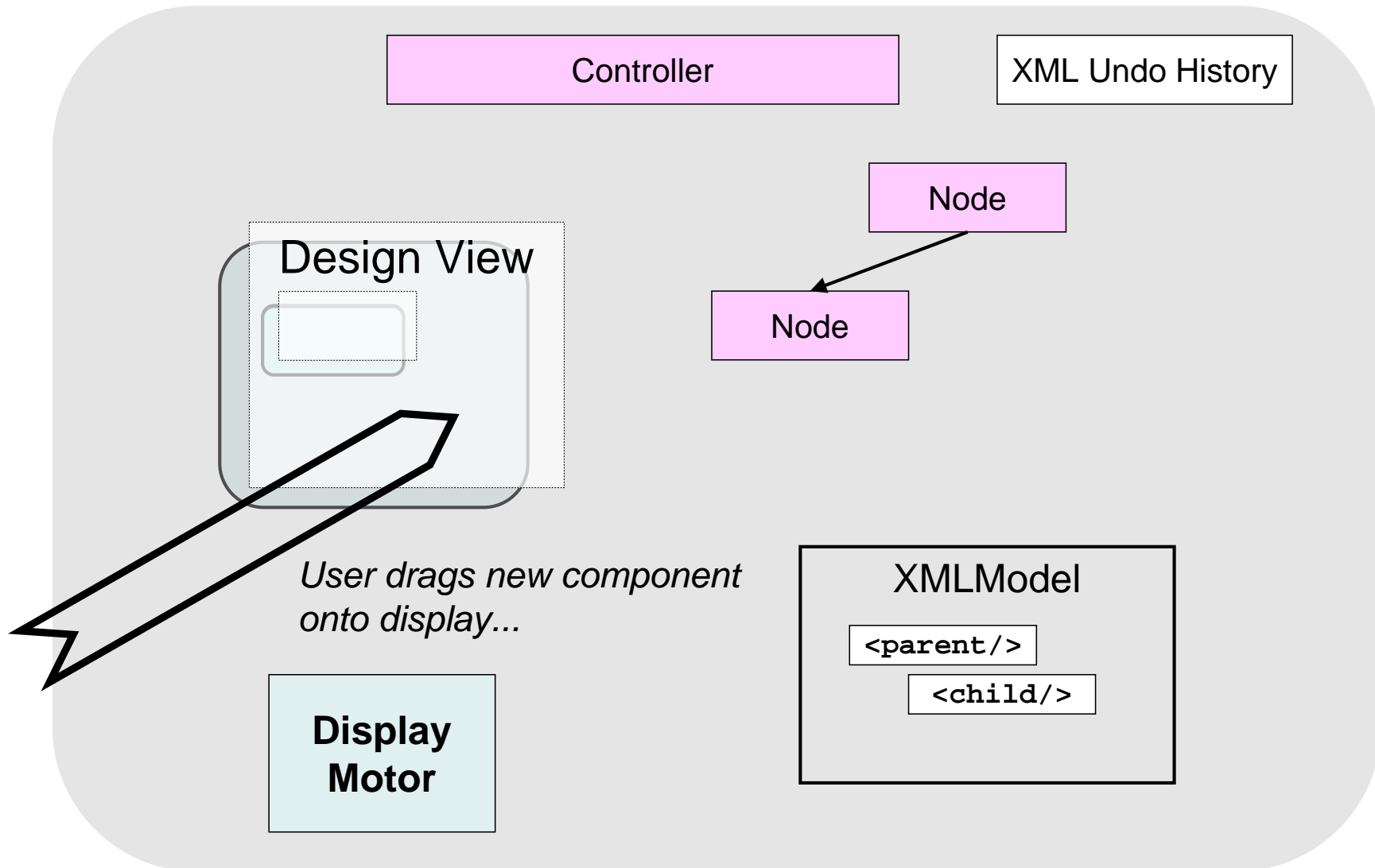
Design View Layers: Selection Chrome



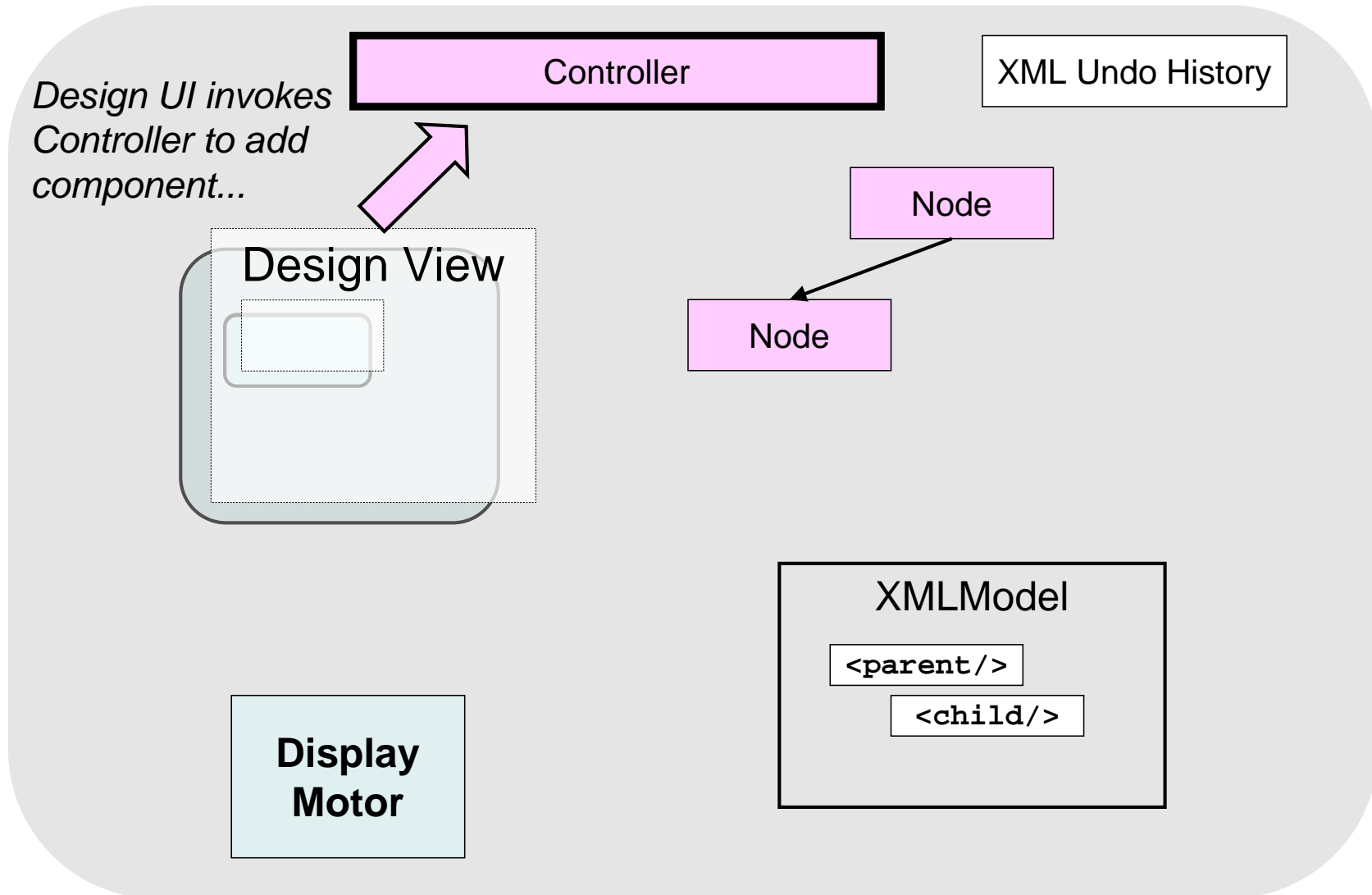
Design View Layers: What the User Sees



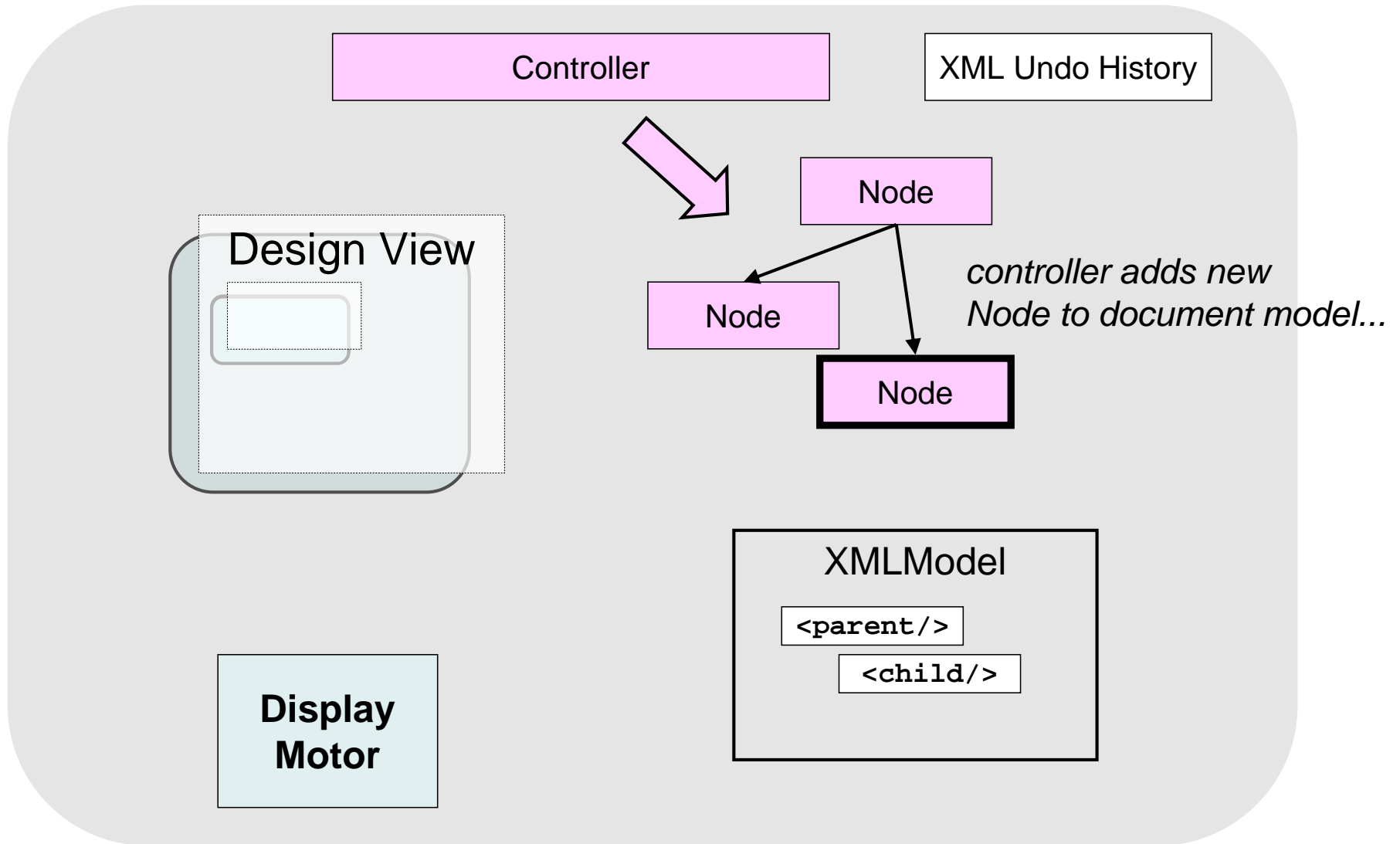
Visual Merchandiser Interaction



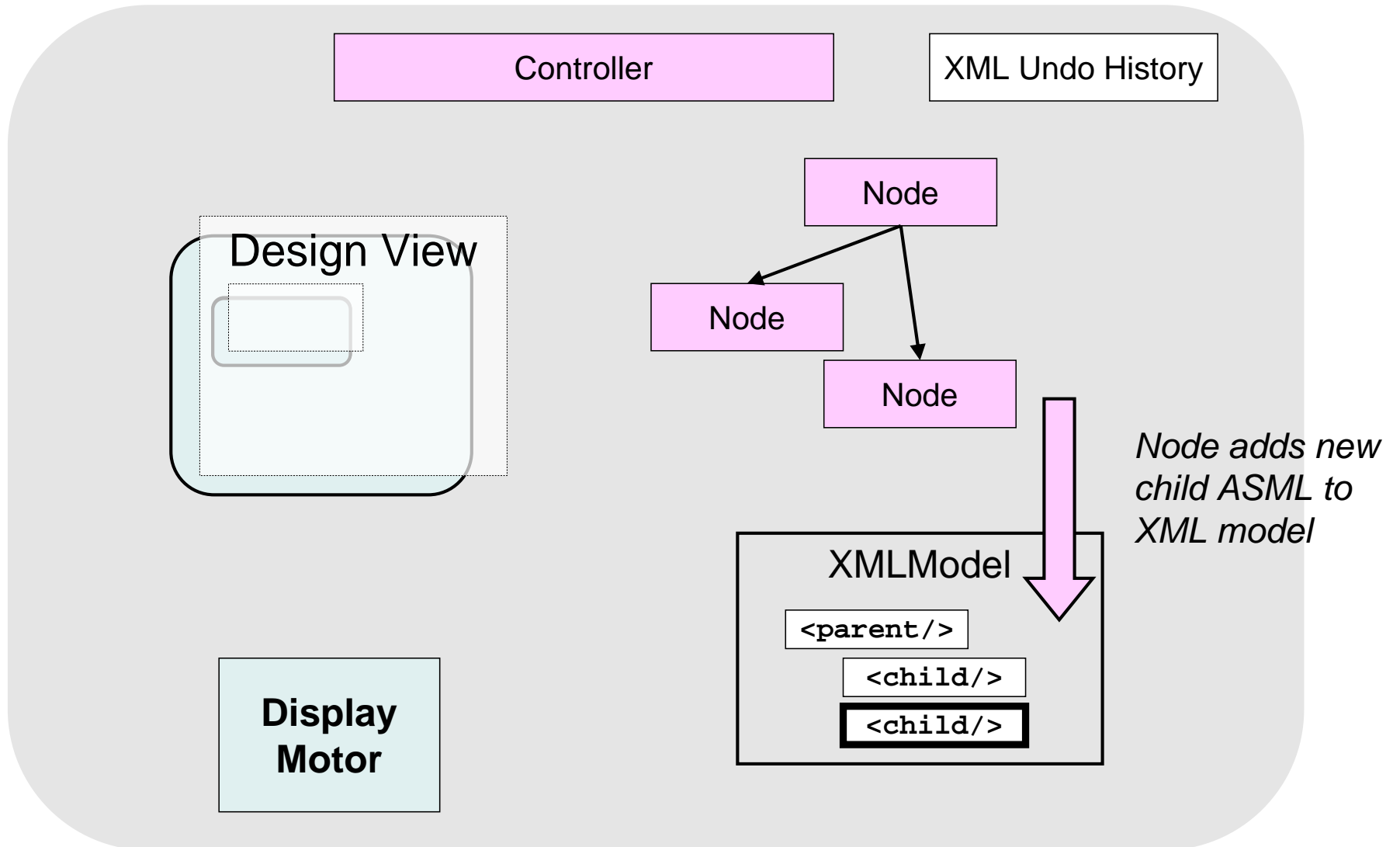
Visual Merchandiser Interaction



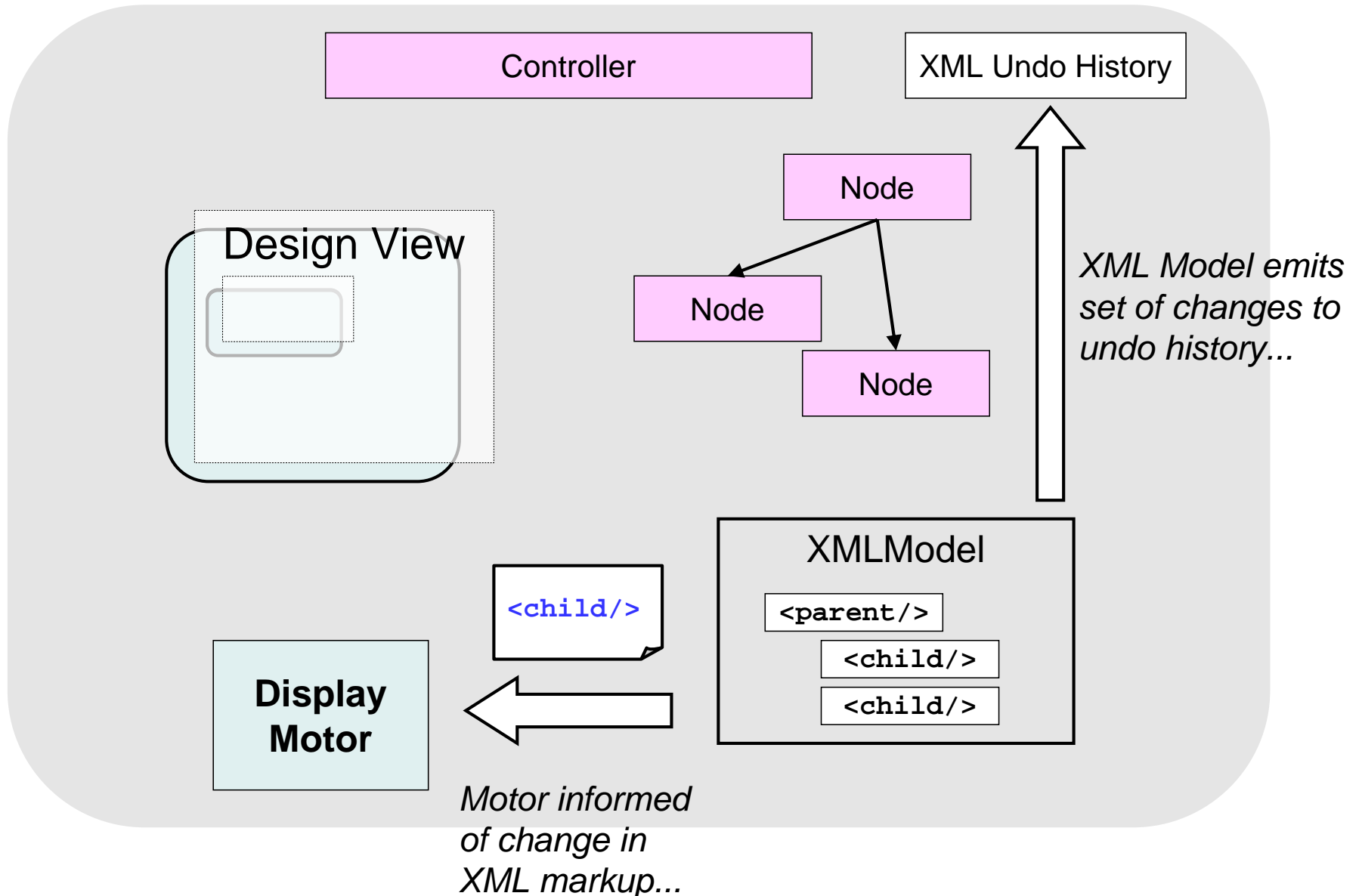
Visual Merchandiser Interaction



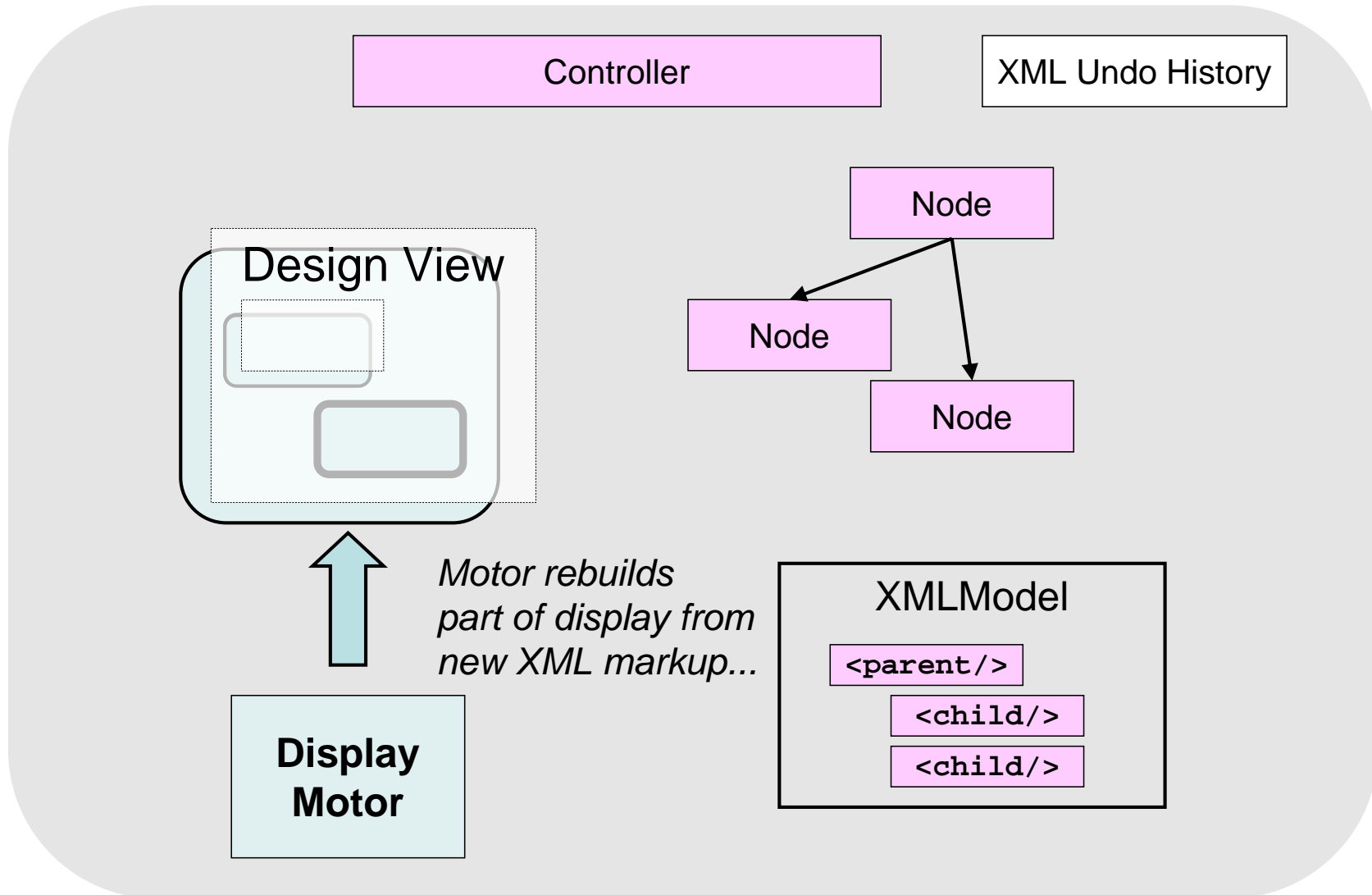
Visual Merchandiser Interaction



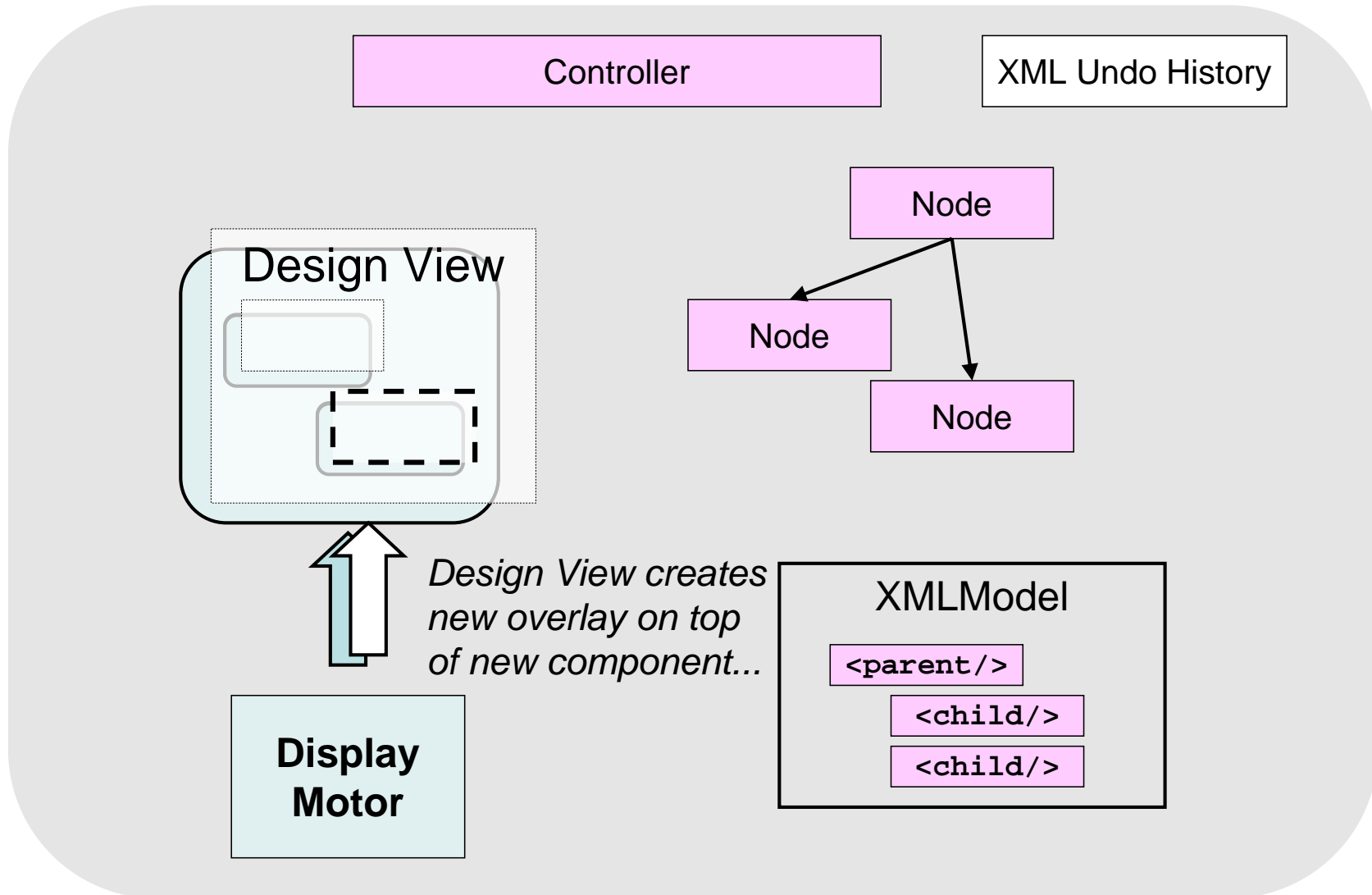
Visual Merchandiser Interaction



Visual Merchandiser Interaction



Visual Merchandiser Interaction



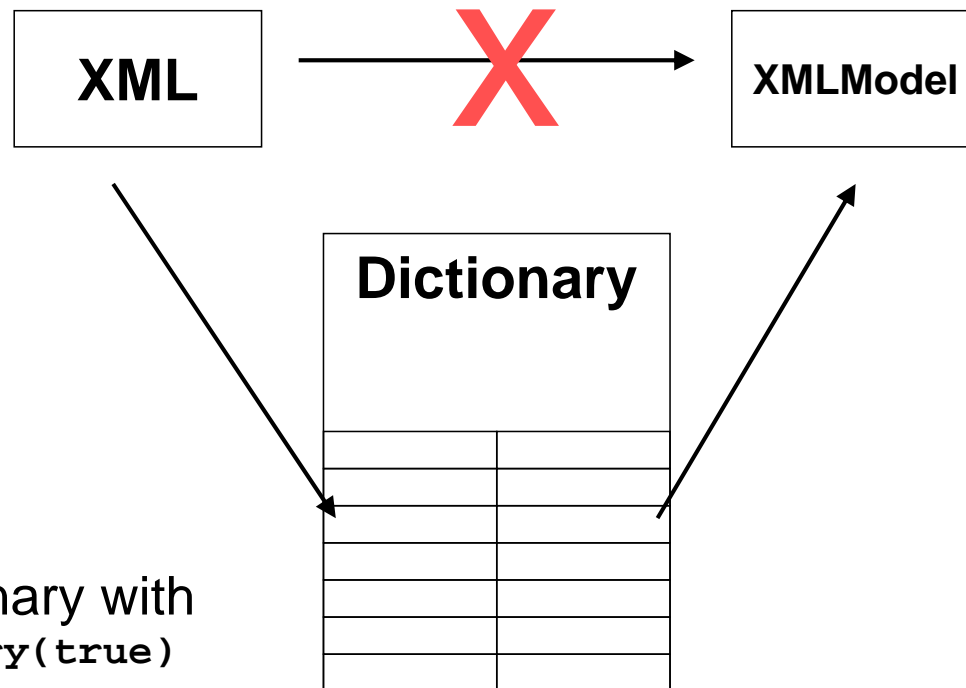
Visual Merchandiser: Framework Highlights

Weak Dictionaries:

- The runtime model (owned by the Motor) is self-contained and can know nothing about the design-time model
- Need 1-1 association between XML and XMLModel objects, Display components and their Design View components
- Cannot subclass XML or Display components to add a property!
- Solution: Use Weak Dictionary to provide a "shadow reference" from runtime model.

Weak Associations via Dictionaries

can't subclass XML to add property!



create Dictionary with
`new Dictionary(true)`
for weak refs, so XML
objects are not pinned
in memory.

Visual Merchandiser: More Framework Highlights

- DragManager: many drag-and-drop gestures in the interface
- Custom ITreeDataDescriptor implementation for Content Browser

Display Component Goals

- Components are "100% presentation" with no knowledge of Controllers, Services, data sources
- May be Flex Components or Flash MovieClips
- Passively populated with models by Motor
- Components are dynamically loadable on demand when a display is rendered

Display Component Code Walkthrough

Display Components: Framework Highlights

- Modules provide our vehicle for dynamically loading components on demand
- XML namespace of a Display component does not identify a package or library, but a loadable Module
- Modules are loaded directly into parent applicationDomain, permitting their classes to become visible in the parent application without requiring the Module to instantiate them itself

Display Components: Framework Highlights

- Flash/Flex synergy is getting much better, but still tricky
- Flash components are revealed through the *<name of Adobe Flash/Flex interop kit here>*
- Flex APIs must be revealed to Flash through simple, narrow interfaces that have no Flex framework dependencies

Display Components: The Metadata Problem

- We needed a way to embed metadata in component code for the Motor that would be uniform for Flash and Flex
- We did not want to use the [MetadataTag] format because it's Flex-specific and requires fussy compiler options that must change as the tag system evolves
- We like XML!

Display Component Metadata

```
public class SomeComponent {  
  
    public static const ALLURENT_INFO:XML =  
        <allurentInfo>  
            <mappings>  
                <contentMapping property="product"  
class="com.allurent.arc.model.Product"/>  
                <priceMapping property="price"/>  
                <imageMapping property="source"/>  
            </mappings>  
        </allurentInfo>;  
  
    //...  
}  
  
var metadata:XML =  
    getDefinitionByName(componentClass) [ "ALLURENT_INFO" ] ;
```

Thank You!

Resources:

- <http://www.allurent.com>
- <http://www.joeberkovitz.com>