



MAX

Performance-Tuning Mobile Flex Applications

Evtim Georgiev

Computer Scientist, Flex SDK
<http://evtimmy.com>

Steve Shongrunden

Computer Scientist, Flex SDK
<http://flexponential.com>



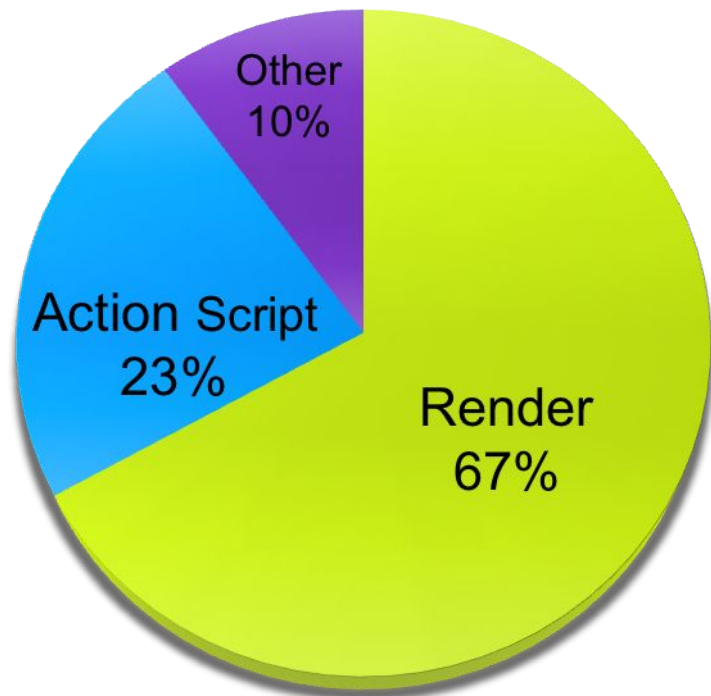
- Performance Metrics
- General Tips
- Item Renderers
- Views
- Performance Optimizations in Flex 4.6
- Q & A

Performance Metrics

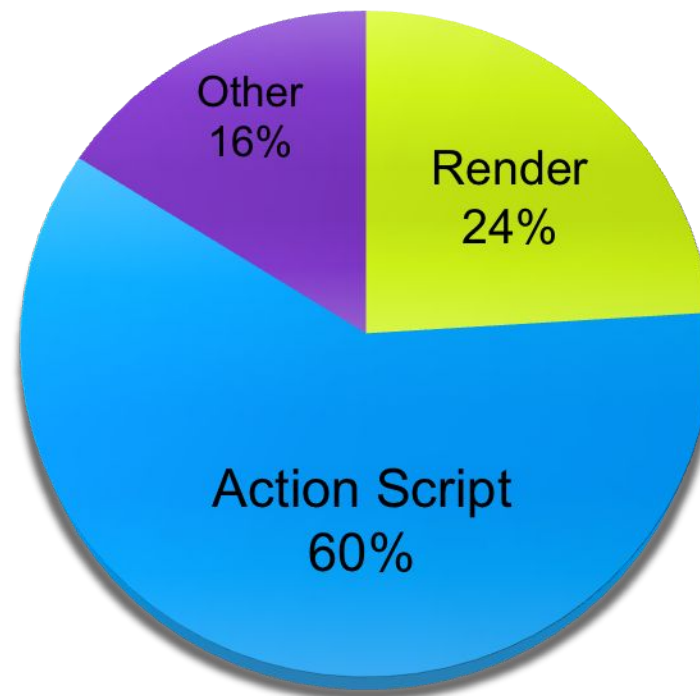
- Metrics
 - Types of Execution Time
 - Frame rate (fps)
 - Startup / validation time
 - Memory
 - SWF Size



Critical Areas: Object Creation, Measurement/Layout, Render



**Effects,
Scrolling,
Transitions**



**Startup,
Navigation,
Data processing**

General Tips

- Use the best component for the job
- Cache and queue external content
- Set `cacheAsBitmap` on graphics that change infrequently but redraw often
- Minimize nested containers

- BitmapImage vs Image
- Caching and Queuing (New in Flex 4.5)
 - ContentCache class
 - Cache on by default
 - Queue off by default
 - contentLoader property on Spark Image, BitmapImage
 - IContentLoader interface
- Use PNG instead of GIF/JPEG
- Avoid large images for small icons

- **Label - light**
 - Single-styled
 - Recommended for static text (mobile & desktop)
 - Used by DefaultItemRenderer (desktop)

- **RichText - heavier**
 - Multi-styled

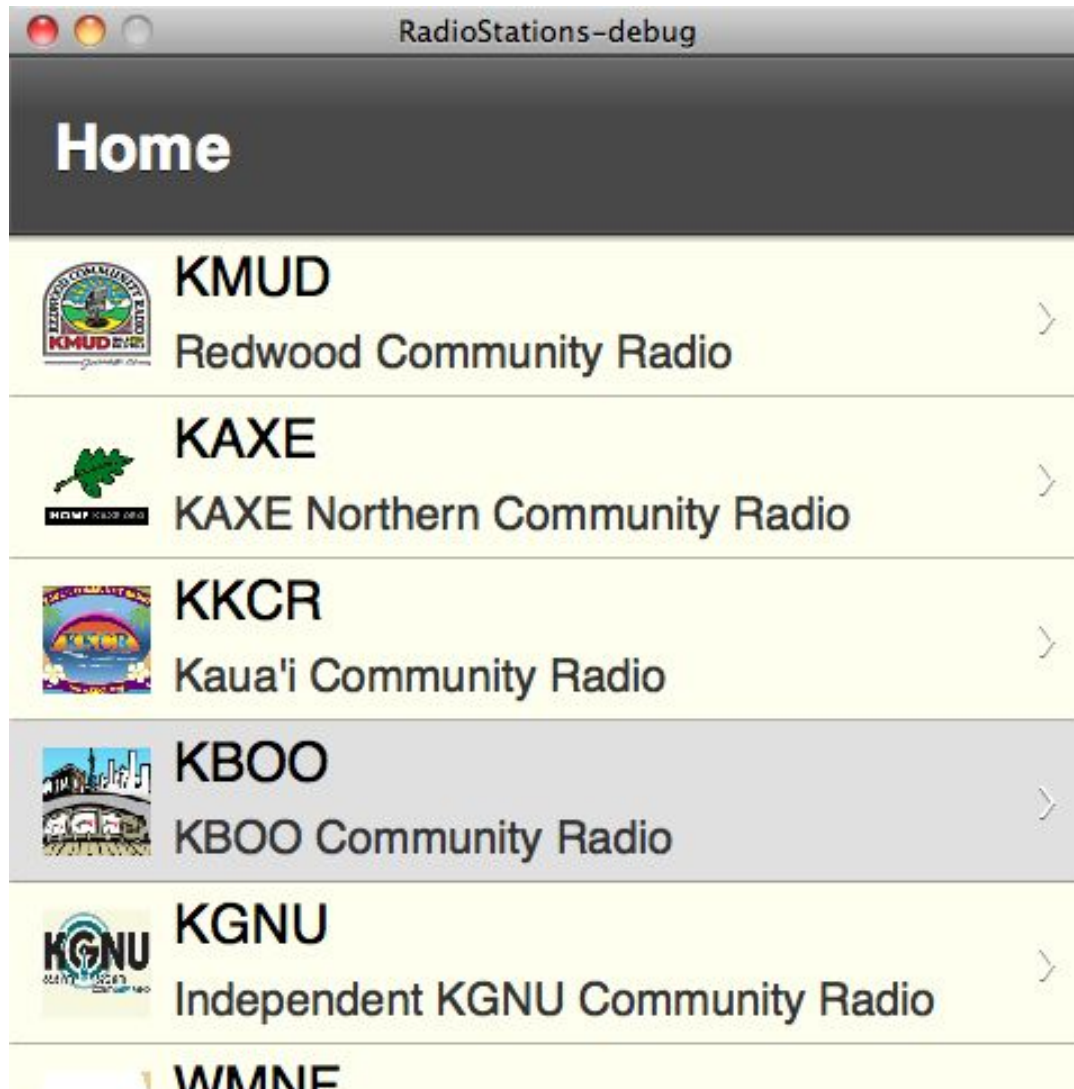
- **RichEditableText - heaviest**
 - Selection, edit
 - Used by TextInput and TextArea (desktop)

- **StyleableTextField (New in Flex 4.5)**
 - Mobile support for edit and selection (turn off if not needed!)
 - Used by LabelItemRenderer & IconItemRenderer (mobile)
 - Can't use directly in MXML

- **StyleableStageText (New in Flex 4.6)**
 - Native OS text control
 - Responsive editing
 - Really fast scrolling
 - Used by TextInput and TextArea (mobile)
 - Can't use directly in MXML

ItemRenderers

- Creating ItemRenderers in MXML is quick and simple
- Avoid creating heavy ItemRenderers
 - Don't use heavy (text) components
 - Cache and queue external content requests
 - Use cacheAsBitmap (carefully!)
 - Turn off “autoDrawBackground” if not needed
 - Avoid Filters / drop shadows
 - Avoid complex binding expressions
 - Reduce number of nested Groups
- Use the mobile-optimized IconItemRenderer and LabelItemRenderer

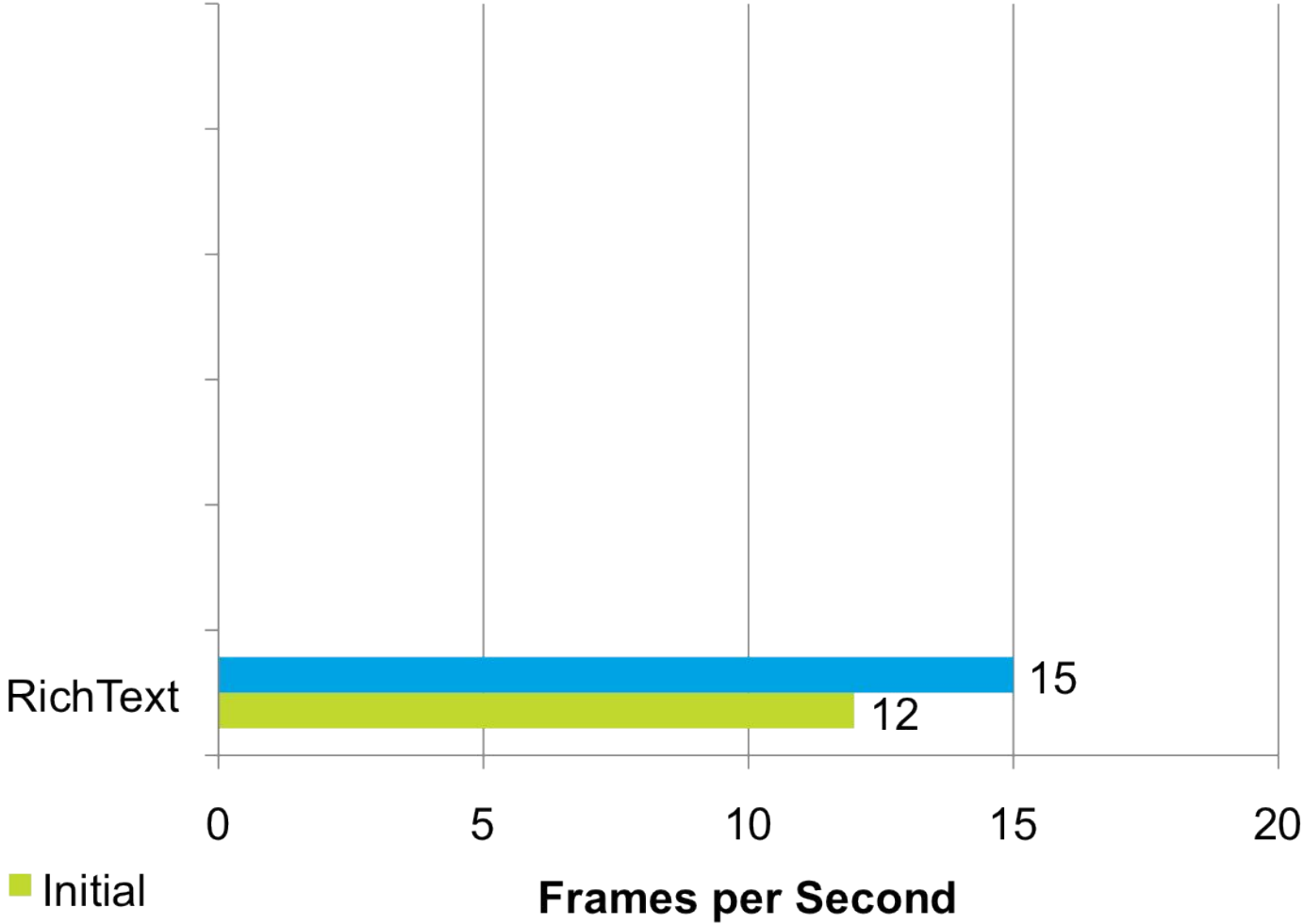


Optimizing MXML ItemRenderer



```
<s:ItemRenderer ...>  
  <s:Rect left="0" right="0" top="0" bottom="0">  
    <s:fill> <s:SolidColor color="0" alpha="0"/></s:fill>  
  </s:Rect>  
  
  <s:Line left="0" right="0" bottom="0" height="0">  
    <s:stroke><s:SolidColorStroke .../></s:stroke>  
  </s:Line>  
  
  <s:HGroup verticalAlign="middle" left="15" right="15"  
    top="0" bottom="0" gap="10">  
    <s:BitmapImage id="icon" source="{data.graphic}" ... />  
    <s:VGroup width="100%" height="100%" gap="12" ...>  
      <s:RichText width="100%" text="{data.callLetters}"/>  
      <s:RichText width="100%" text="{data.name}" .../>  
    </s:Vgroup>  
    <assets:Chevron/>  
  </s:Hgroup>  
</s:ItemRenderer>
```

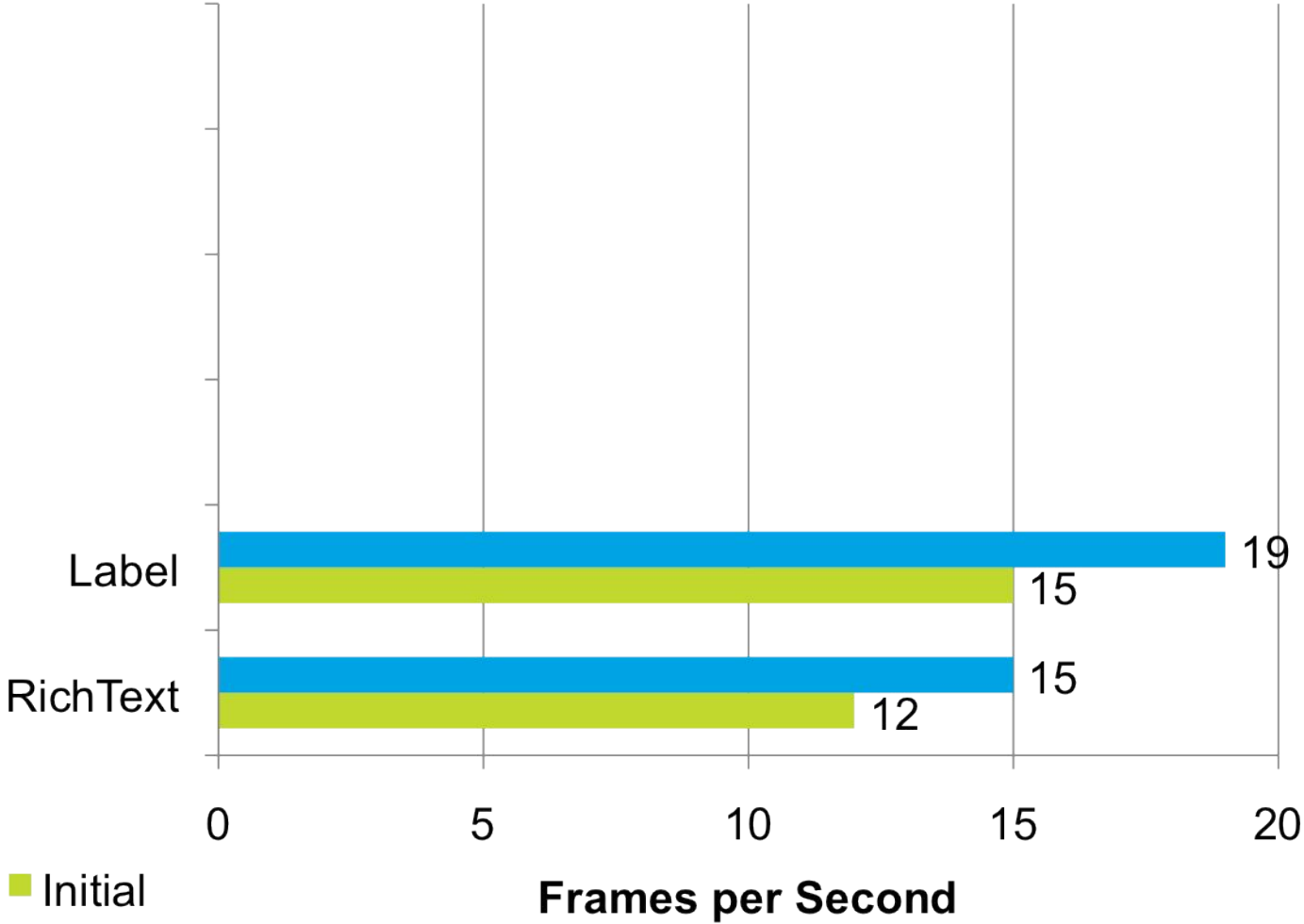
Scrolling



Replacing RichText with Label

```
<s:ItemRenderer ...>  
  <!-- background fill -->  
  <s:Rect ... />  
  
  <!-- bottom separator -->  
  <s:Line ... />  
  
  <s:HGroup ...>  
  
    <s:BitmapImage .../>  
  
    <s:VGroup ...>  
      <s:Label width="100%" text="{data.callLetters}"/>  
      <s:Label width="100%" text="{data.name}" .../>  
    </s:VGroup>  
  
    <assets:Chevron/>  
  </s:Hgroup>  
</s:ItemRenderer>
```


Scrolling

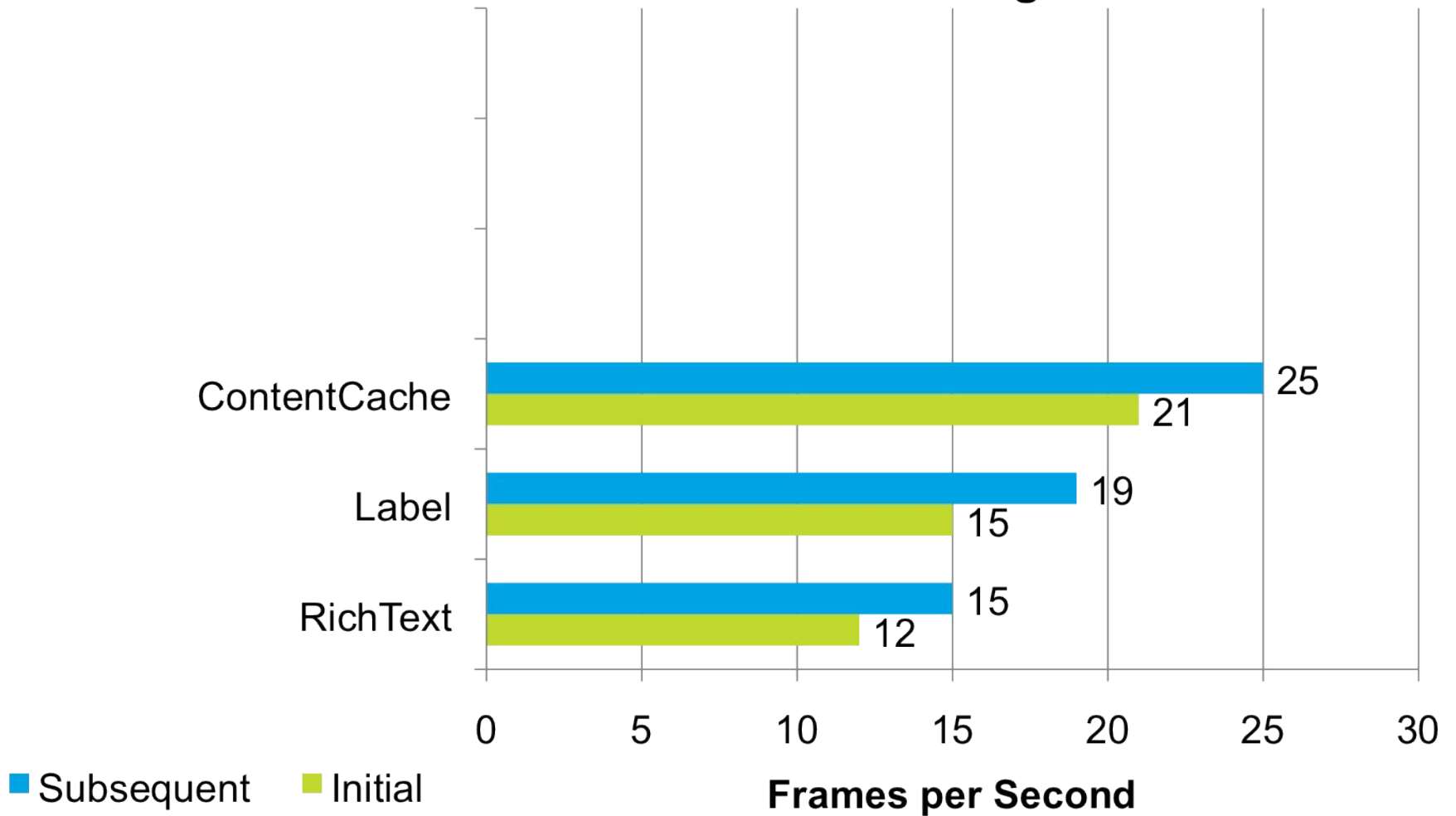


```
<s:ItemRenderer ...>
  <fx:Script>
    <![CDATA[
      import spark.core.ContentCache;
      static public const s_cache:ContentCache = new ContentCache();
    ]]>
  </fx:Script>

  <s:Rect ...
  <s:Line ...

  <s:HGroup ...>
    <s:BitmapImage id="icon" source="{data.graphic}"
      contentLoader="{s_cache}"/>
    <s:VGroup ...>
      <s:Label .../>
      <s:Label ... />
    </s:Vgroup>
    <assets:Chevron/>
  </s:Hgroup>
</s:ItemRenderer>
```

Scrolling



Set “cacheAsBitmap” on the Decorator

```
<s:ItemRenderer ...>

    <fx:Script>
    <![CDATA[
        ...
    ]]>
</fx:Script>

<s:Rect ...
<s:Line ...

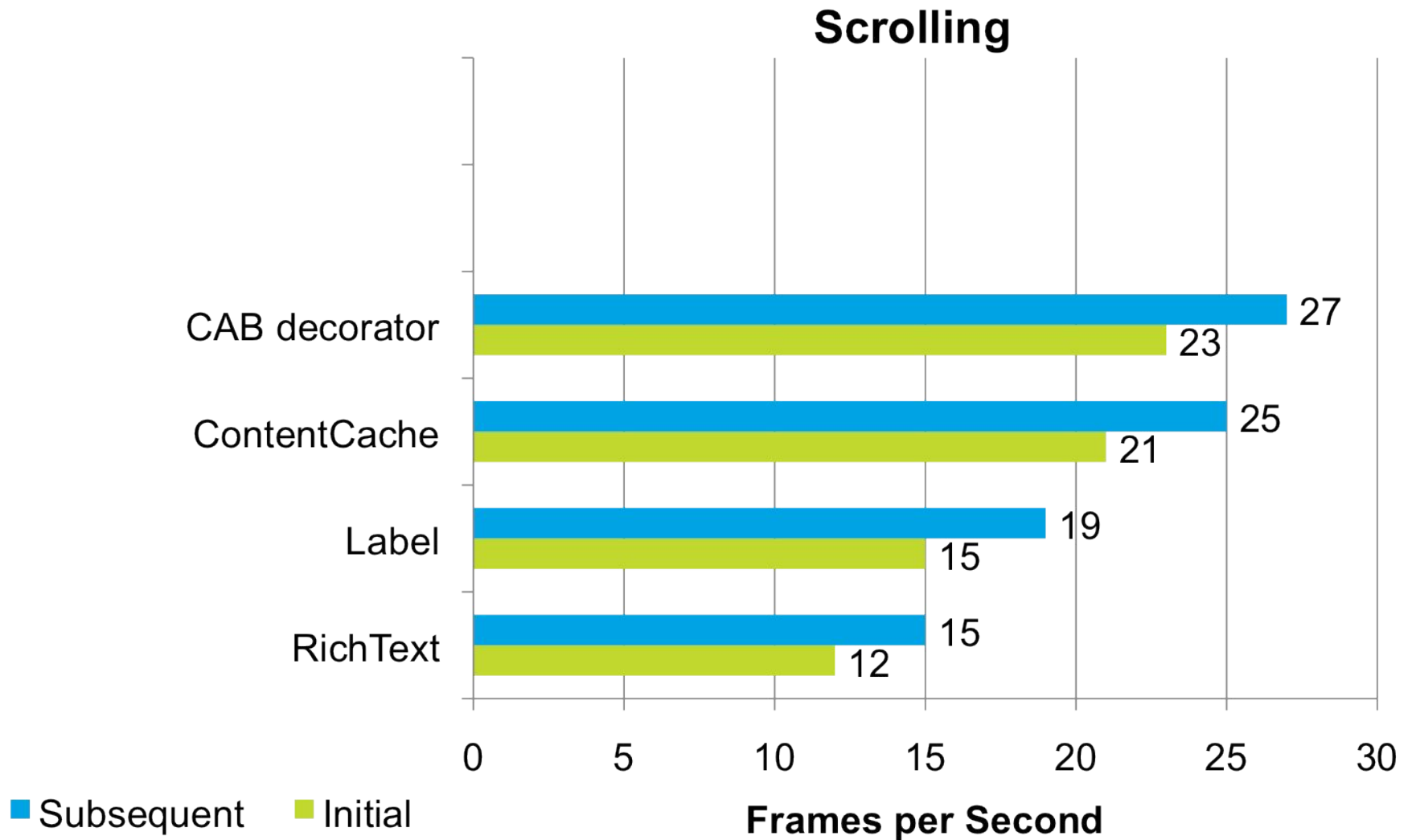
<s:HGroup ...>
    <s:BitmapImage .../>

    <s:VGroup ...>
        <s:Label .../>
        <s:Label ... />
    </s:VGroup>

    <assets:Chevron cacheAsBitmap="true"/>
</s:HGroup>

</s:ItemRenderer>
```

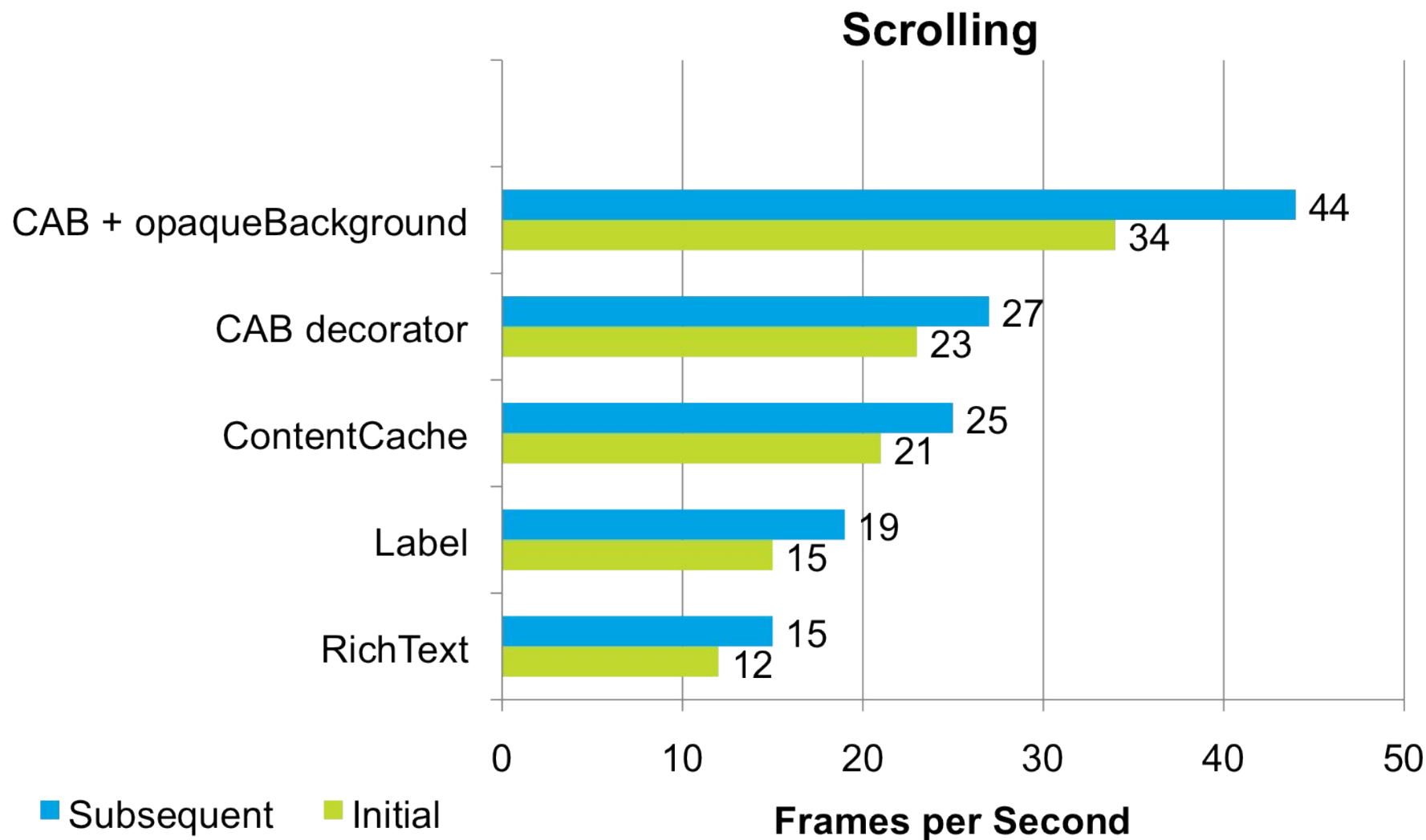
Set “cacheAsBitmap” on the Decorator



cacheAsBitmap + opaqueBackground on the ItemRenderer

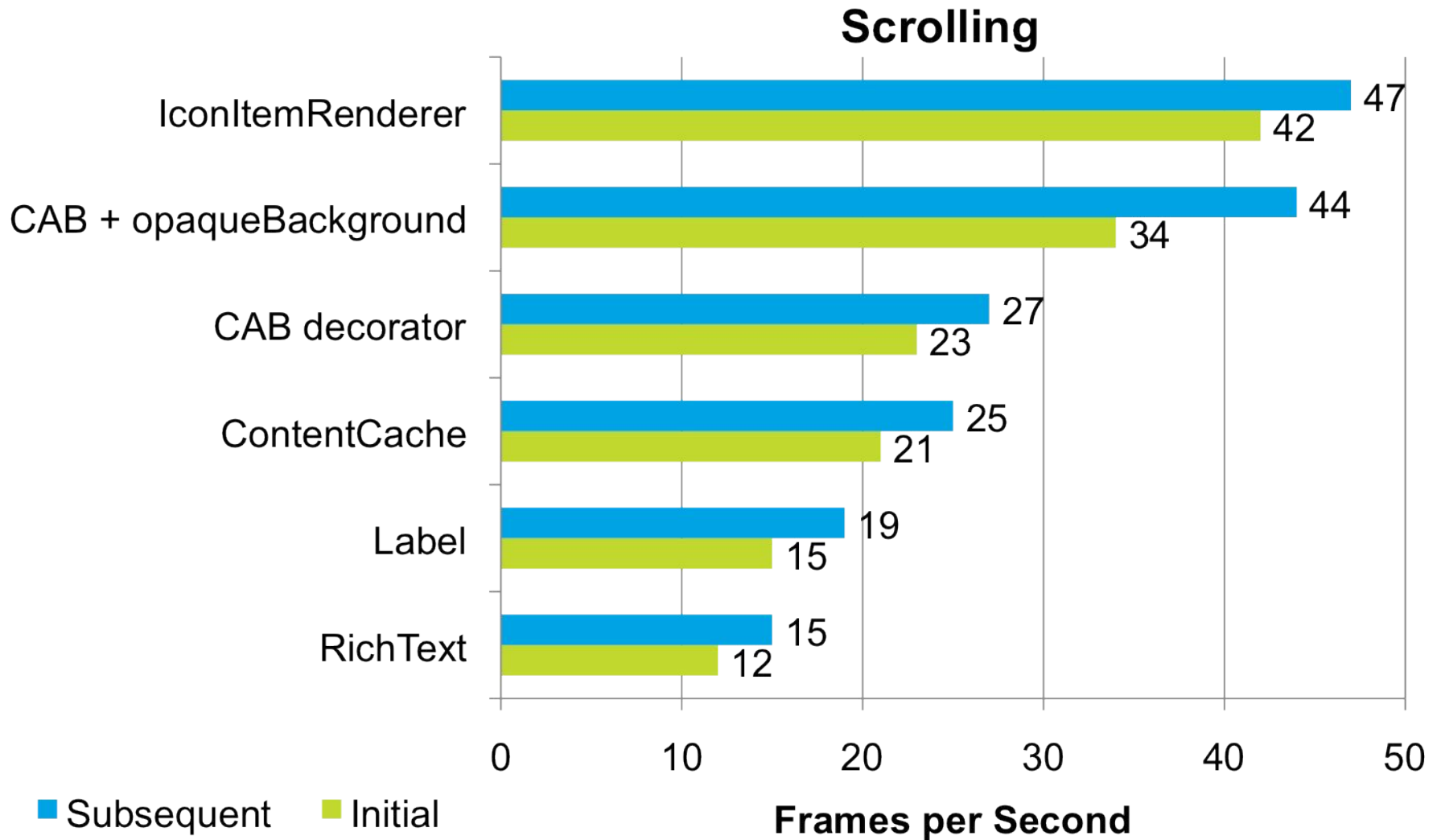
```
<s:ItemRenderer ... opaqueBackground="0xFFFFFFFF" cacheAsBitmap="true">
```

```
    <fx:Script>  
    <![CDATA[  
        ...  
    ]]>  
</fx:Script>  
  
<s:Rect ...  
<s:Line ...  
  
<s:HGroup ...>  
    <s:BitmapImage .../>  
  
    <s:VGroup ...>  
        <s:Label .../>  
        <s:Label ... />  
    </s:VGroup>  
  
    <assets:Chevron .../>  
</s:HGroup>  
  
</s:ItemRenderer>
```



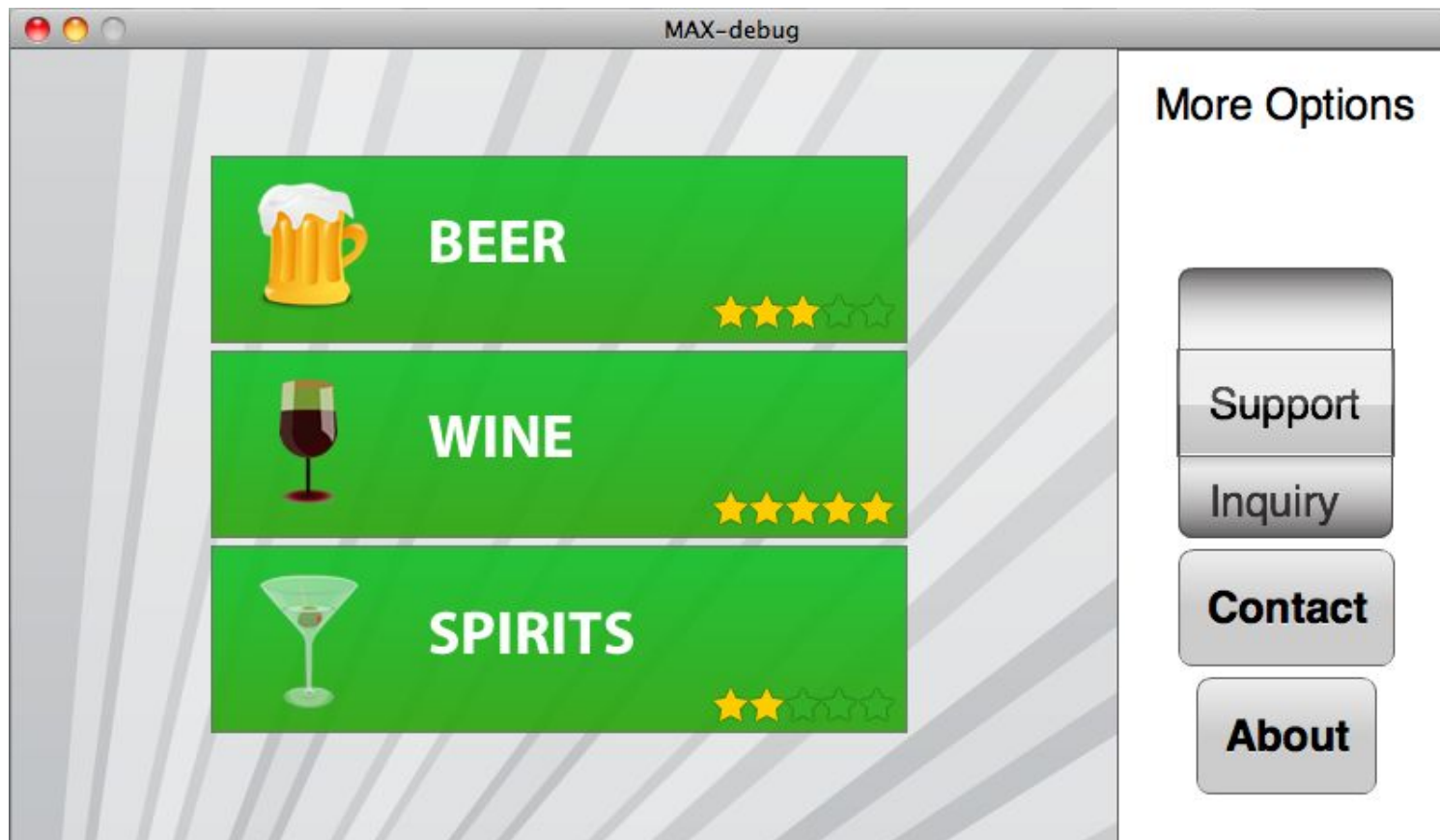
- Optimized for Mobile
 - Use StylableTextField
 - Lightweight layout
 - Add more sophisticated ContentCache management
- Configurable
 - Use styles, properties to control the layout, text, etc.
- Extensible
 - Subclass to tweak layout, parts, etc.
 - Tip: Create parts on demand


```
<s:List ...>
  <s:itemRenderer>
    <fx:Component>
      <s:IconItemRenderer labelField="callLetters"
        messageField="name"
        iconField="graphic"
        iconWidth="48"
        iconHeight="48"
        decorator="{assets.Chevron}">
        <fx:Script>
          <![CDATA[
            import assets.Chevron;
          ]]>
        </fx:Script>
      </s:IconItemRenderer>
    </fx:Component>
  </s:itemRenderer>
</s:List>
```

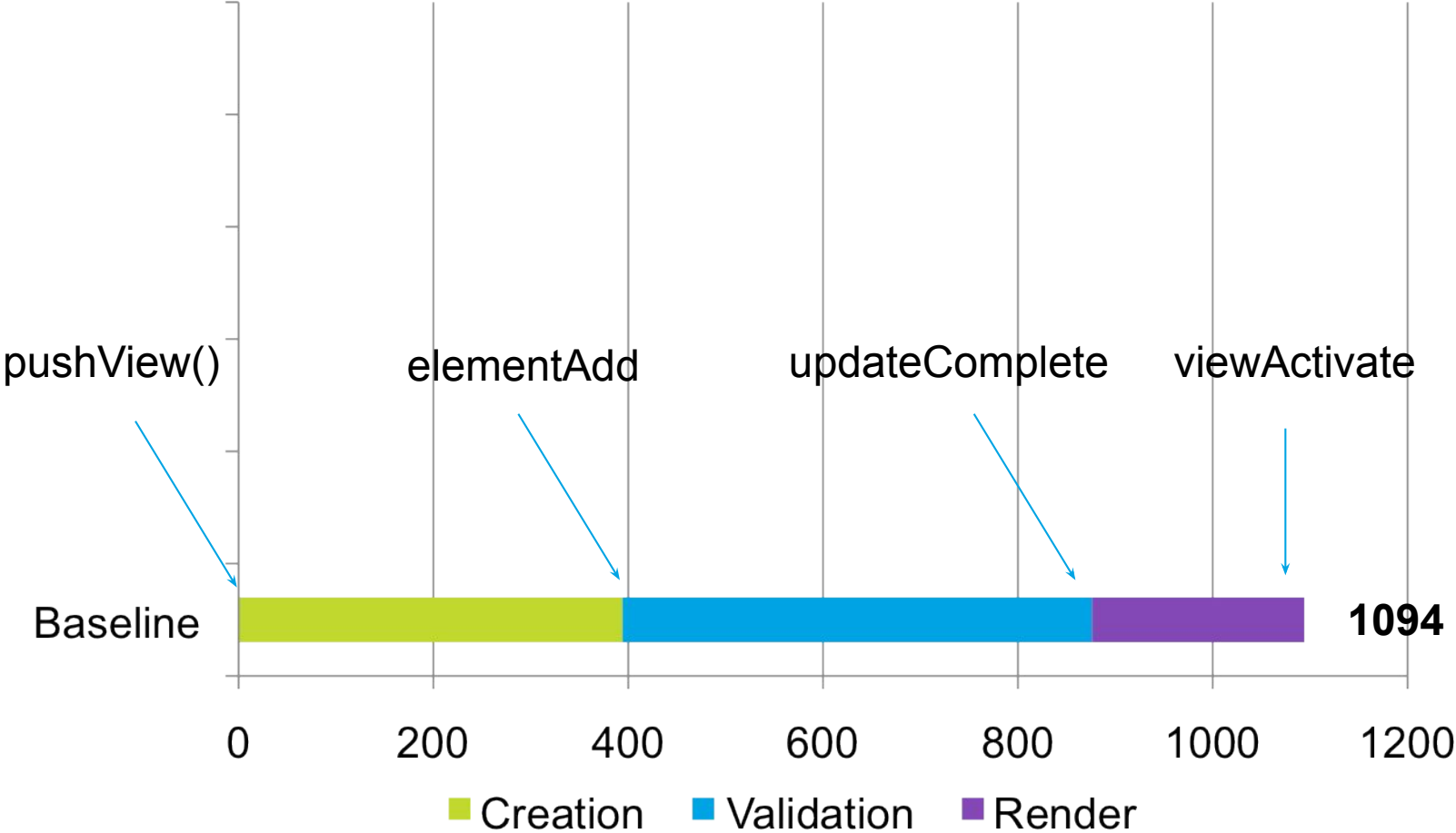


Views

- Creating Views in MXML is quick and simple
- Avoid creating heavy Views
 - Don't use unnecessarily heavy (text) components
 - Defer object creation
 - Use BitmapImage instead of Image
 - Cache and queue external content requests
 - Use Group instead of BorderContainer
 - Reduce nested containers
 - Use mobile optimized component skins



Activation Time Breakdown (ms)



- Don't create objects until needed

```
<s:Group id="helpView" height="100%"  
    visible.portrait="false"  
    includeInLayout.portrait="false">
```

```
...
```

```
<s:Group>
```

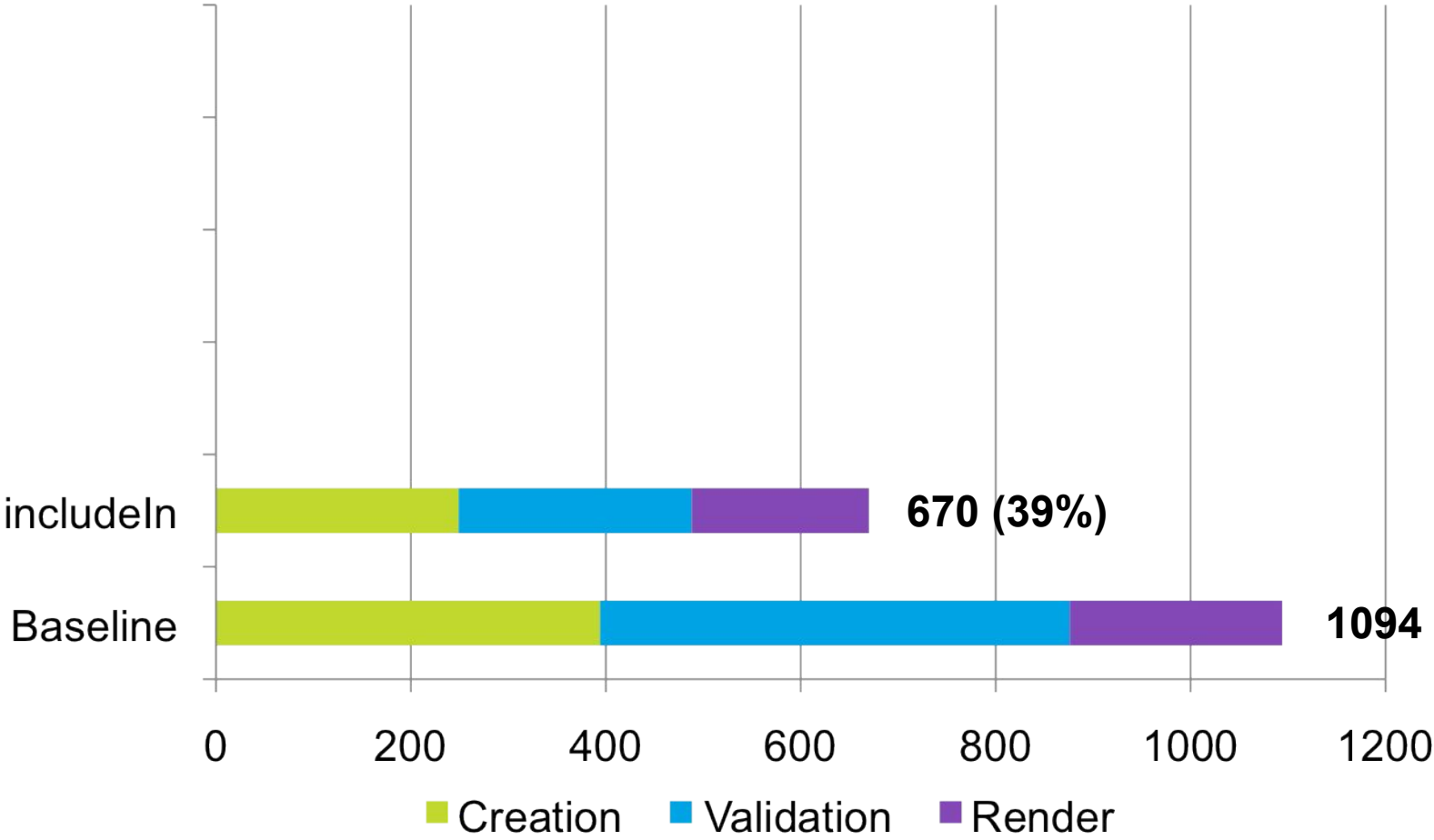


```
<s:Group id="helpView" height="100%"  
    includeIn="landscape">
```

```
...
```

```
<s:Group>
```

Activation Time (ms)



- Spark Image
 - SkinnableComponent
 - Customizable loading state
 - Customizable “error” (broken image) state
- BitmapImage
 - Lightweight GraphicElement
- Cache images that are used frequently



```
<s:View ...>
```

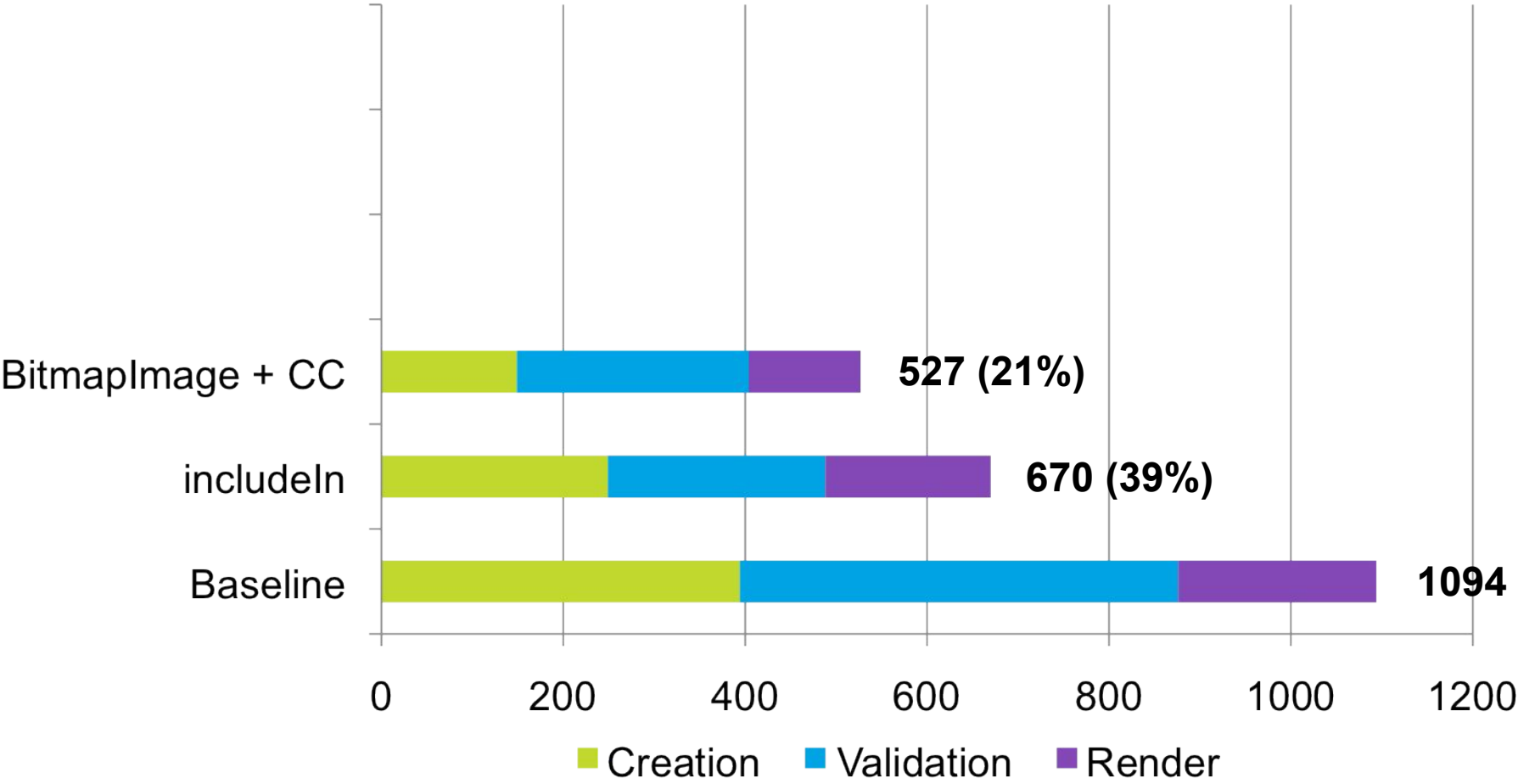
```
<fx:Script>  
  <![CDATA[  
    import spark.core.ContentCache;  
  
    [Bindable]  
    static protected var s_c:ContentCache = new ContentCache();  
  ]]>  
</fx:Script>
```

...

```
<s:BitmapImage source="star.jpg" contentLoader="{s_c}"/>
```

```
</s:View>
```

Activation Time (ms)



- BorderContainer is not optimized for mobile
- Instead use a Group with a Rect

```
<s:BorderContainer borderColor="red" width="100" height="100">  
  <s:Button label="Play" />  
</s:BorderContainer>
```



```
<s:Group width="100" height="100">  
  <s:Rect top="0" left="0" right="0" bottom="0">  
    <s:stroke>  
      <s:SolidColorStroke color="red" />  
    </s:stroke>  
  </s:Rect>  
  
  <s:Button label="Play" />  
</s:Group>
```

- Sometimes unnecessary nesting is easy to remove

```
<s:Group width="50" height="50">  
  <s:Group width="100%" height="100%">  
    <s:Button />  
  </s:Group>  
</s:Group>
```



```
<s:Group width="50" height="50">  
  <s:Button />  
</s:Group>
```

- Consider using ConstraintLayout instead of nested VGroup and HGroup



Using ConstraintLayout Instead of Nested Groups



Using ConstraintLayout Instead of Nested Groups

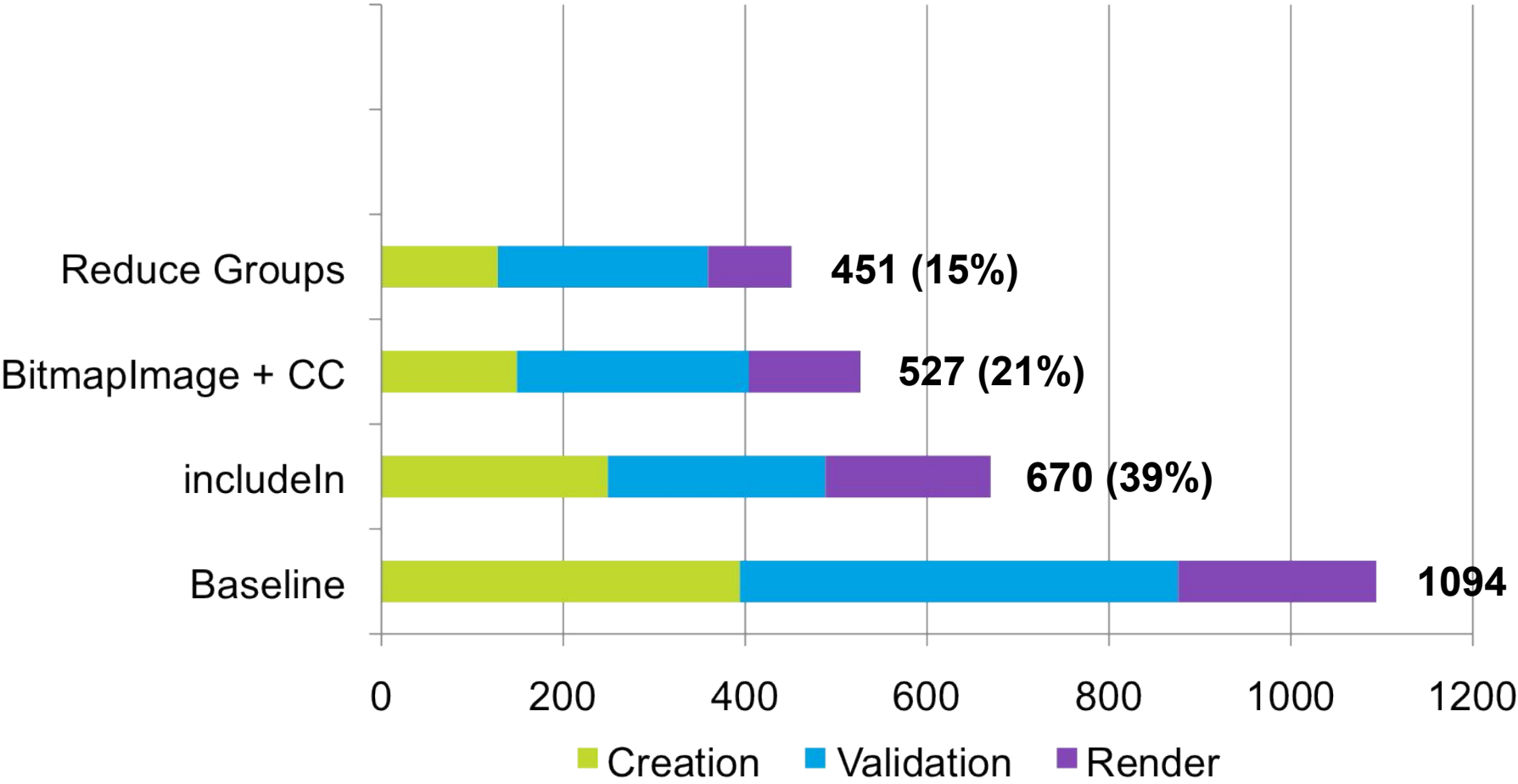
```
<s:Group width="100%" height="100%">
  <s:layout>
    <s:ConstraintLayout>
      <s:constraintColumns>
        <s:ConstraintColumn id="bmpColumn"/>
        <s:ConstraintColumn width="100%" id="contentColumn"/>
      </s:constraintColumns>
    </s:ConstraintLayout>
  </s:layout>

  <s:BitmapImage ...
    left="bmpColumn:1" right="bmpColumn:1"
    top="1" bottom="1"/>

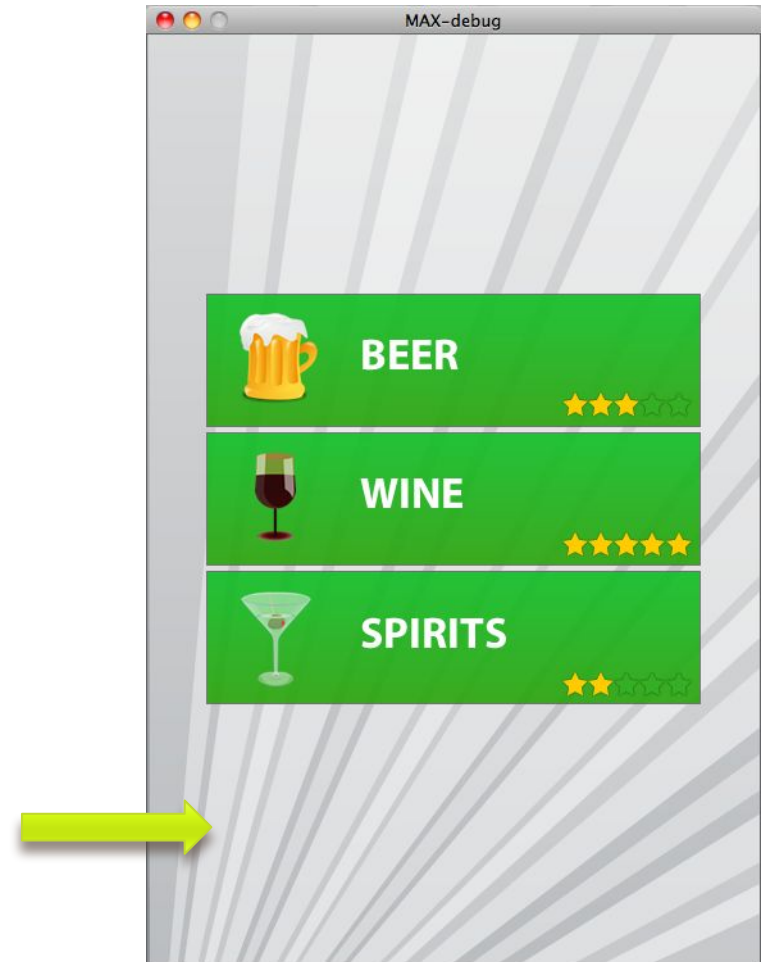
  <s:Label text="Beer" top="21" left="contentColumn:25" />

</s:Group>
```


Activation Time (ms)



- GraphicElements
 - Lightweight graphic primitives
- FXG
 - Static compile-time optimized graphics



Example of MXML GraphicElements

```
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"  
  xmlns:s="library://ns.adobe.com/flex/spark">
```

```
<s:Button text="Hello World" x="150"/>
```

```
<s:Rect id="myRect1" width="100" height="100">  
  <s:fill><s:SolidColor color="#FF0000" /></s:fill>  
</s:Rect>
```

```
<s:Rect id="myRect2" width="100" height="100" x="20" y="20">  
  <s:fill><s:SolidColor color="#00FF00" /></s:fill>  
</s:Rect>
```

```
<s:Rect id="myRect3" width="100" height="100" x="40" y="40">  
  <s:fill><s:SolidColor color="#0000FF" /></s:fill>  
</s:Rect>
```

```
</s:Application>
```

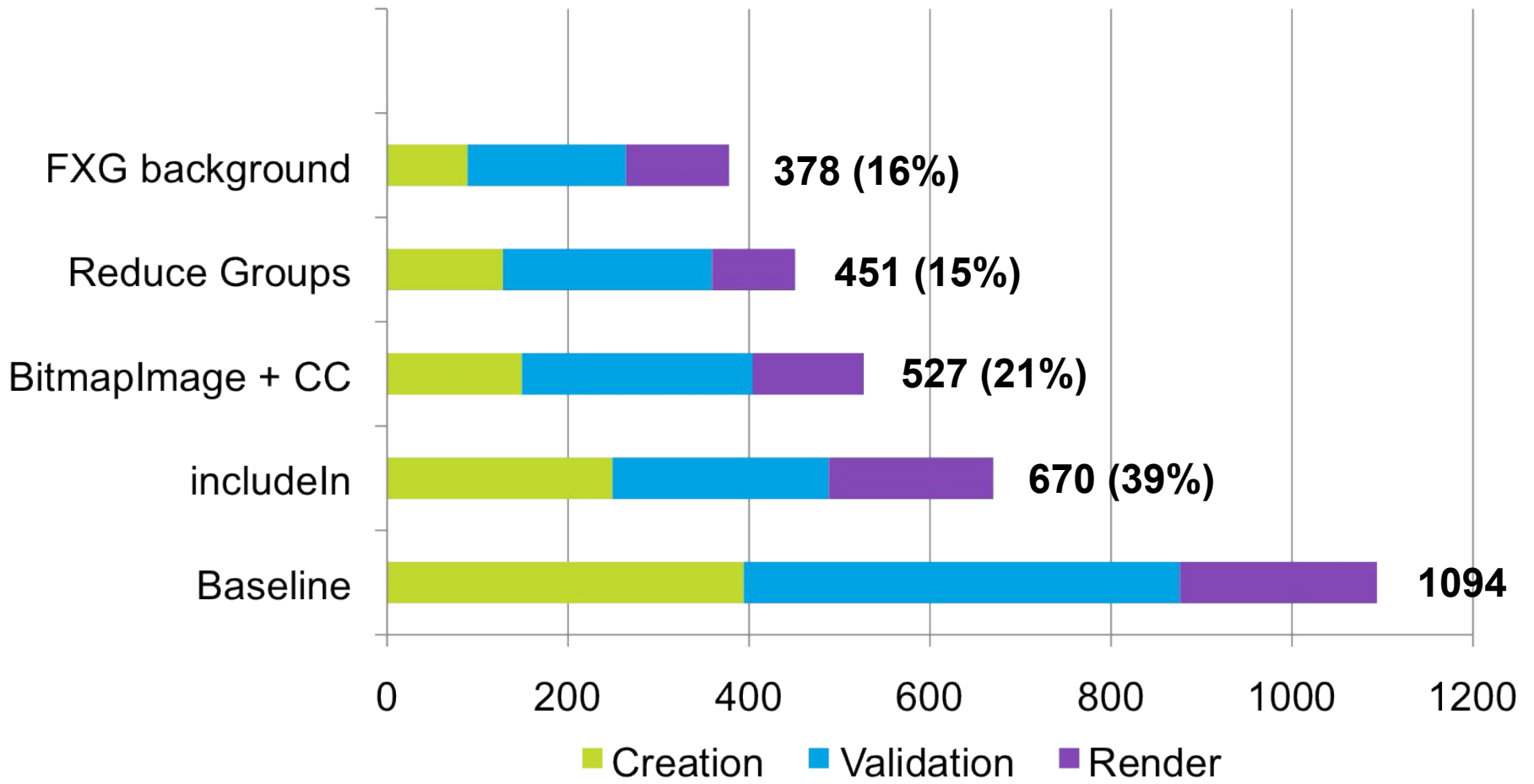
Example of Compiler Optimized FXG

```
<s:Application ...  
  xmlns:assets="*">  
  
  <s:Button  
    text="Hello World"  
    x="150" />  
  
  <assets:MyGraphic />  
  
</s:Application>
```

MyGraphic.fxg:

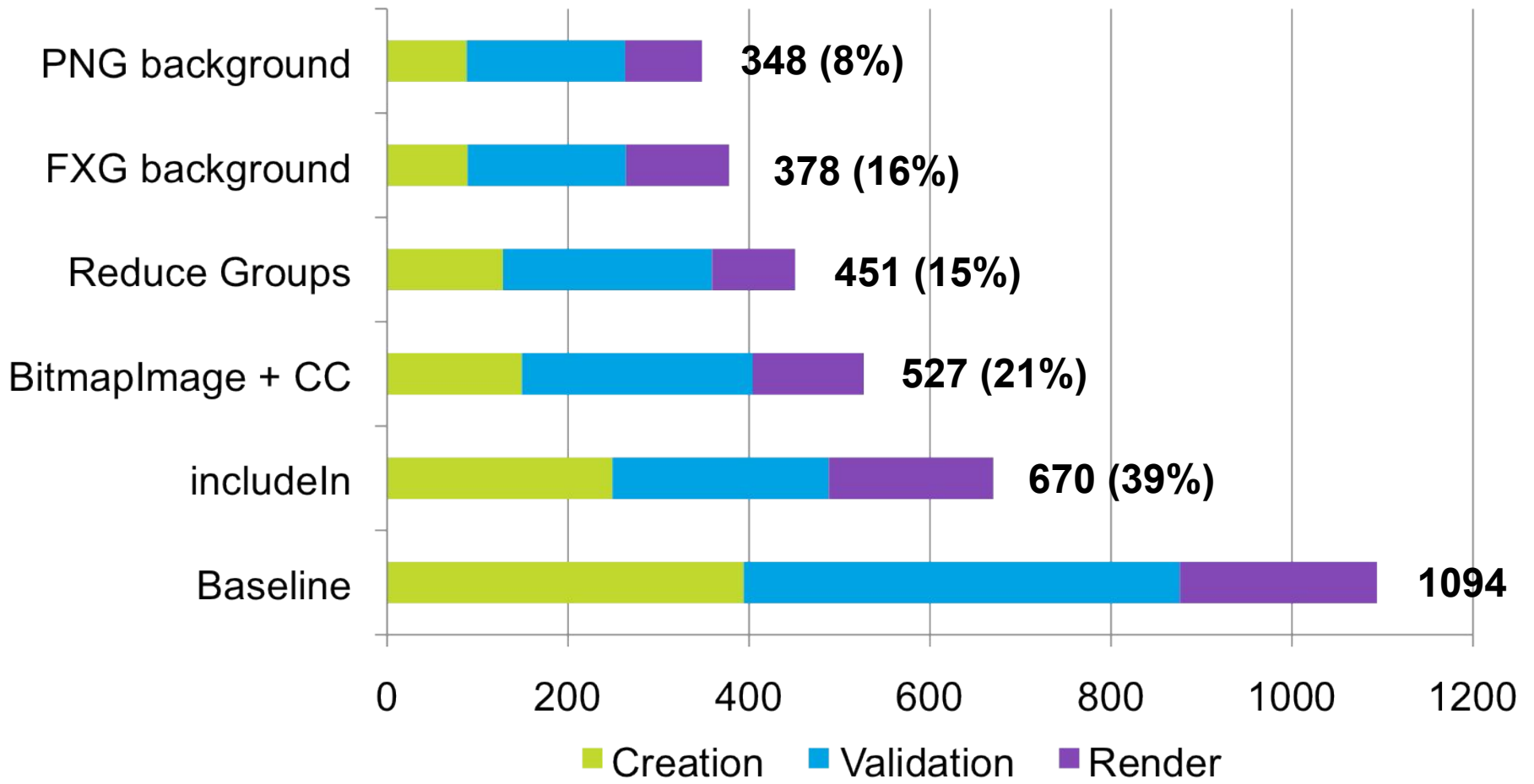
```
<Graphic xmlns="http://ns.adobe.com/fxg/2008">  
  <Rect width="100" height="100">  
    <fill>  
      <SolidColor color="#FF0000" />  
    </fill>  
  </Rect>  
  
  <Rect width="100" ... x="20" y="20">  
    <fill>  
      <SolidColor color="#00FF00" />  
    </fill>  
  </Rect>  
  
  <Rect width="100" ... x="20" y="20">  
    <fill>  
      <SolidColor color="#0000FF" />  
    </fill>  
  </Rect>  
</Graphic>
```

Activation Time (ms)

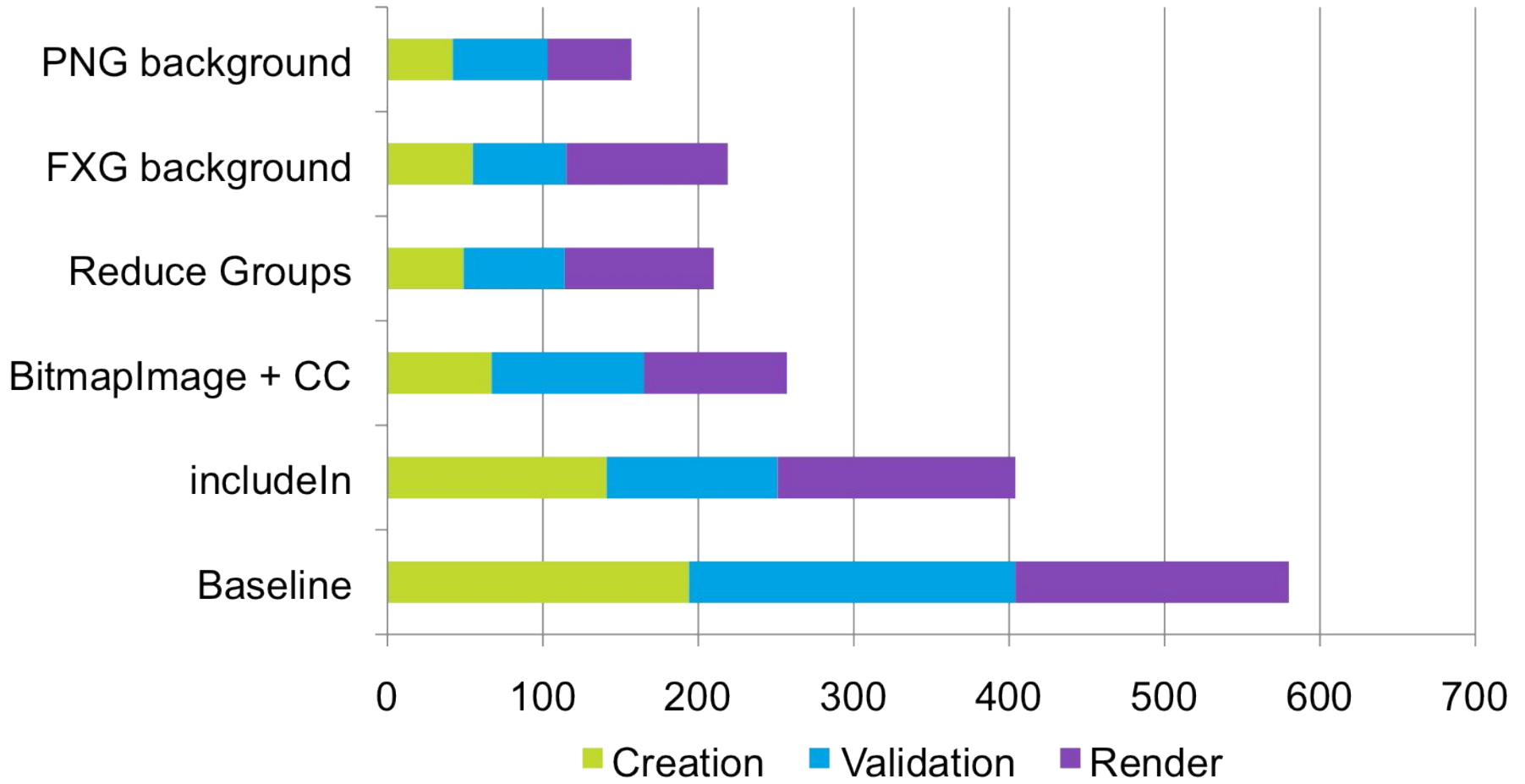


- Consider converting complicated FXG shapes to bitmaps
- Reduce rendering time
- Lose scaling fidelity

Activation Time (ms)



Activation Time (ms)



Performance Optimizations in Flex 4.6

- On demand scrollbars
 - Scrollbar skin parts are now factory parts
 - Created when touch interaction starts
 - Up to 15% faster for simple List views
 - Tip: Update custom Scroller skins
 - Tip: Use Scroller.viewport instead of Scroller.verticalScrollBar

```
trace(myScroller.verticalScrollBar.value);
```



```
trace(myScroller.viewport.verticalScrollPosition);
```

itemRendererFunction Recycles!

- Tip: Only create one ClassFactory per item renderer class

```
private function badIRFunction(item:Object):ClassFactory {  
    if (Number(item) % 2 == 0)  
        return new ClassFactory(RedRenderer);  
    else  
        return new ClassFactory(BlueRenderer);  
}
```



```
private var red:ClassFactory = new ClassFactory(RedRenderer);  
private var blue:ClassFactory = new ClassFactory(BlueRenderer);
```

```
private function goodIRFunction(item:Object):ClassFactory {  
    if (Number(item) % 2 == 0)  
        return red;  
    else  
        return blue;  
}
```

- 32-bit rendering enables better color rendering, Stage3D, StageVideo
- 16-bit rendering has better scrolling performance
- Flash Builder 4.6 will automatically set this to 16-bit in new projects
- Existing projects should be updated

```
<android>  
  <manifestAdditions>  
    <![CDATA[  
      <manifest android:installLocation="auto">  
        <uses-permission android:name="android.permission.INTERNET"/>  
        ...  
      </manifest>  
    ]]>  
  </manifestAdditions>  
  
  <colorDepth>16bit</colorDepth>  
</android>
```

- Blogs
 - Evtim – <http://www.evtimmy.com>
 - Steve – <http://www.flexponential.com>
- Best Practices for Building Flex Tablet Applications – Glenn Ruehle, Flex SDK
- Flex Performance Tips & Tricks from 360Flex Denver 2011
 - <http://flexponential.com/2011/04/20/flex-performance-tips-tricks/>
- General Performance Tips from Adobe MAX 2010
 - http://2010.max.adobe.com/online/2010/MAX232_1288211035765KTXV
- IconItemRenderer and LabelItemRenderer
 - <http://flexponential.com/tag/iconitemrenderer/>