Google™ 10 I O

GWT Testing Best Practices

Daniel Danilatos
May 10, 2010

View live notes and ask questions about this session on Google Wave: http://bit.ly/io2010-gwt7

Good Testing

- Tests run fast
- High code coverage
- Both granular (unit) and integration testing

View live notes and ask questions about this session on Google Wave: **http://bit.ly/io2010-gwt7**

GWT Testing - Common difficulties

GWT Testing - Common difficulties

- Dependence on JSNI code
    - Most of the Widget library
    - All low-level browser interaction code
    - GWTTestCase is slooooow

GWT Testing - Common difficulties

- Dependence on JSNI code
- UI event driven execution flow
    - Need to simulate user interactions
    - Cross-browser event behavior varies

GWT Testing - Common difficulties

- Dependence on JSNI code
- UI event driven execution flow
- Logic that depends on real browser properties
  - E.g. Something the user has typed
  - E.g. Rendered DOM size queries (height, width etc)

GWT Testing - Common difficulties

- Dependence on JSNI code
- UI event driven execution flow
- Logic that depends on real browser properties
- Need for web- or browser- specific optimizations
  - E.g. Widget reuse
  - E.g. Browser optimized data structures

GWT Testing - Common difficulties

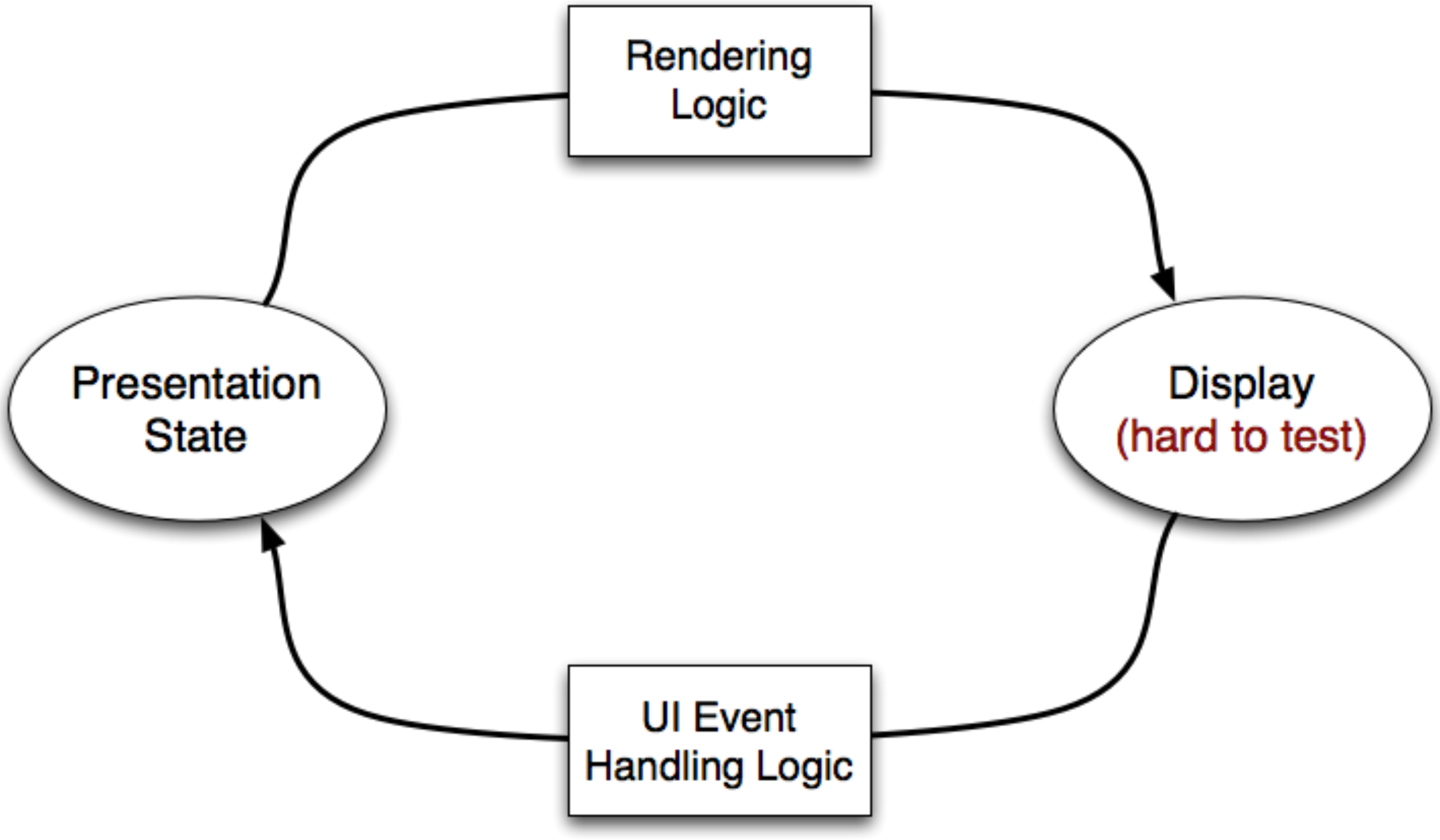
- Dependence on JSNI code
- UI event driven execution flow
- Logic that depends on real browser properties
- Need for web- or browser- specific optimizations
- Asynchronous execution flow
  - E.g. DeferredCommand /Timer
  - Allow layout
  - Split long running tasks to avoid blocking UI
  - etc.

GWT Testing - Common difficulties

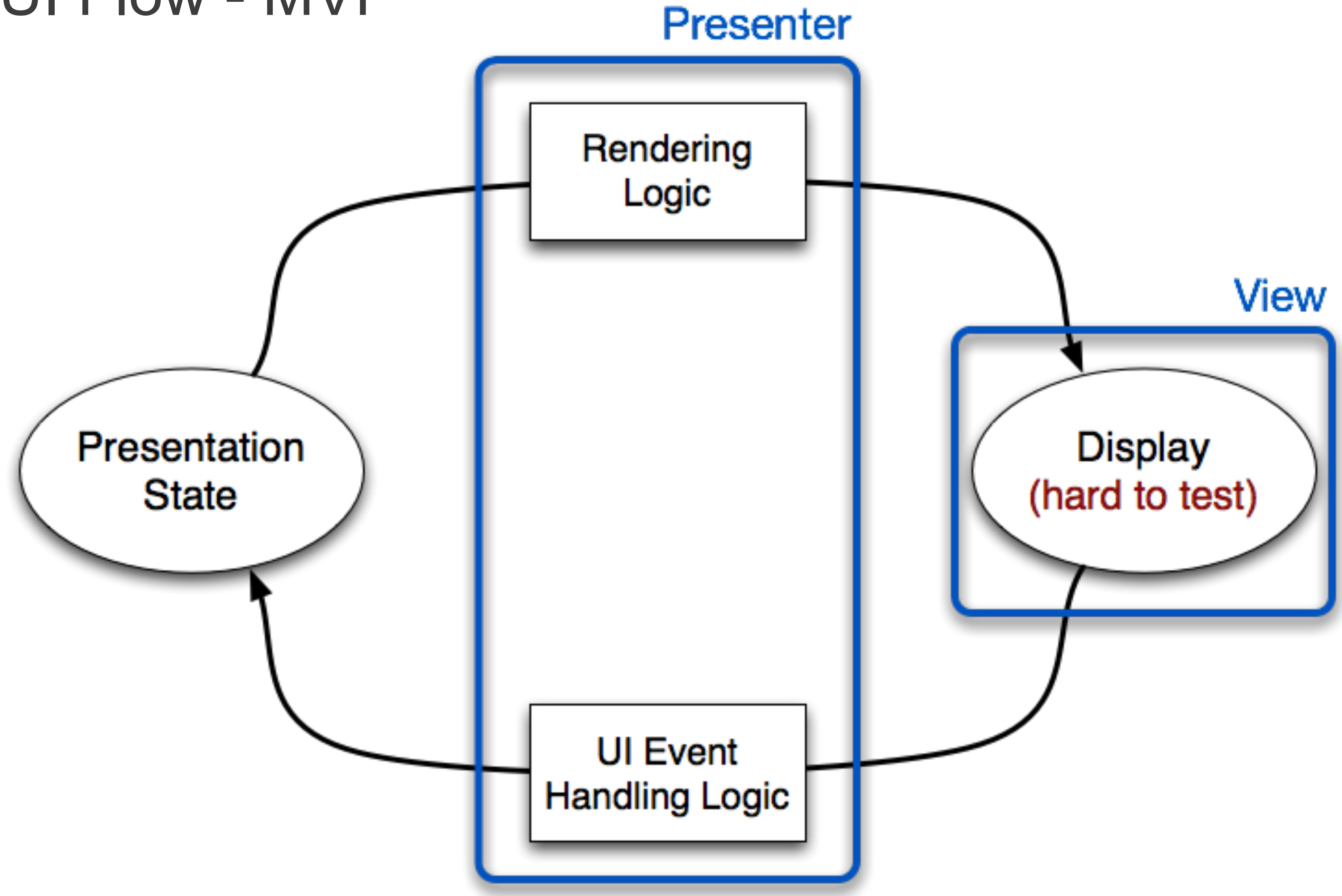
- Dependence on JSNI code
- UI event driven execution flow
- Logic that depends on real browser properties
- Need for web- or browser- specific optimizations
- Asynchronous execution flow

Model View Presenter (MVP)

UI Flow


Rendering Logic
Display (hard to test)
UI Event Handling Logic
Presentation State

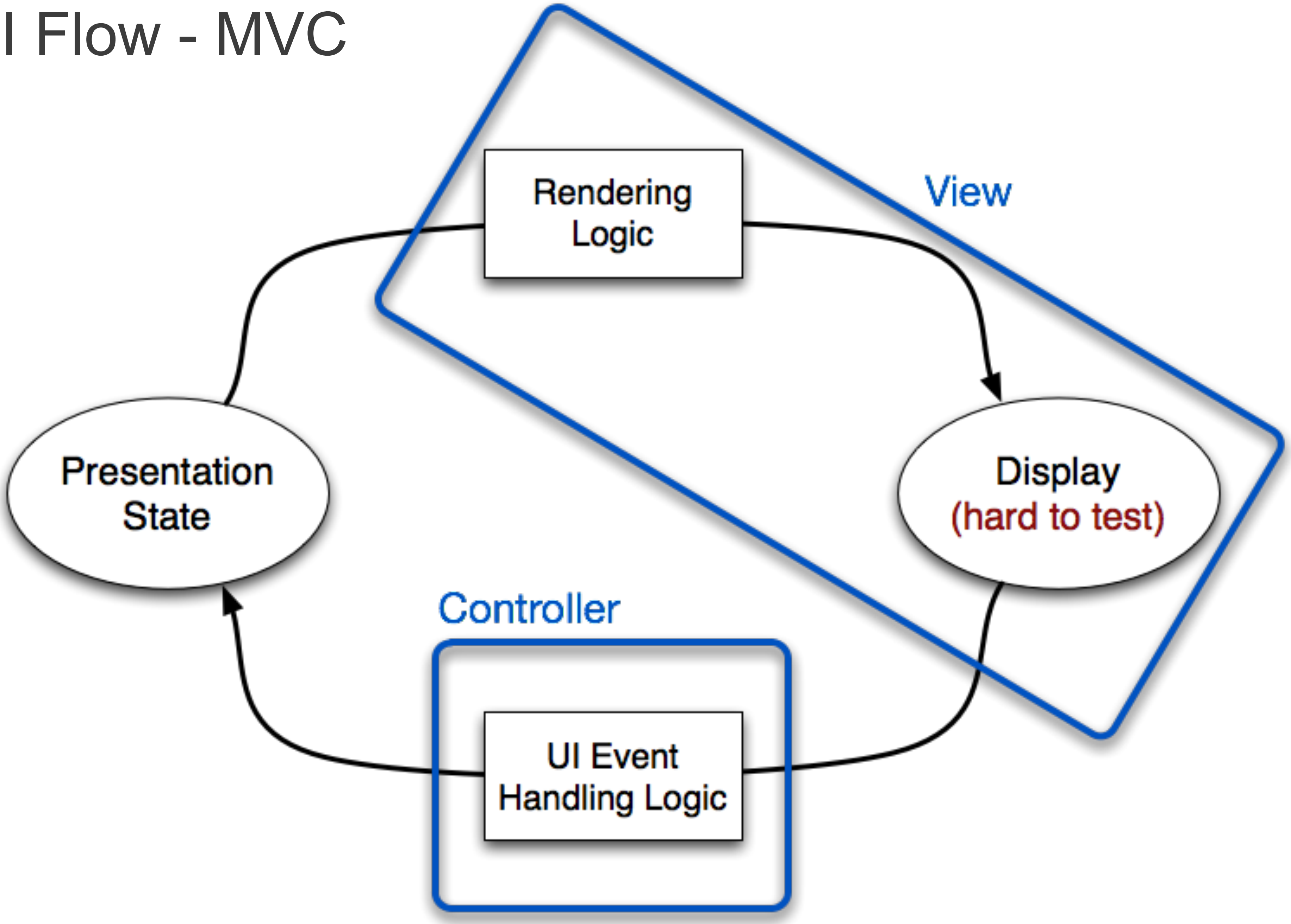
UI Flow - MVP


Presenter
Rendering Logic
View
Presentation State
Display (hard to test)
UI Event Handling Logic

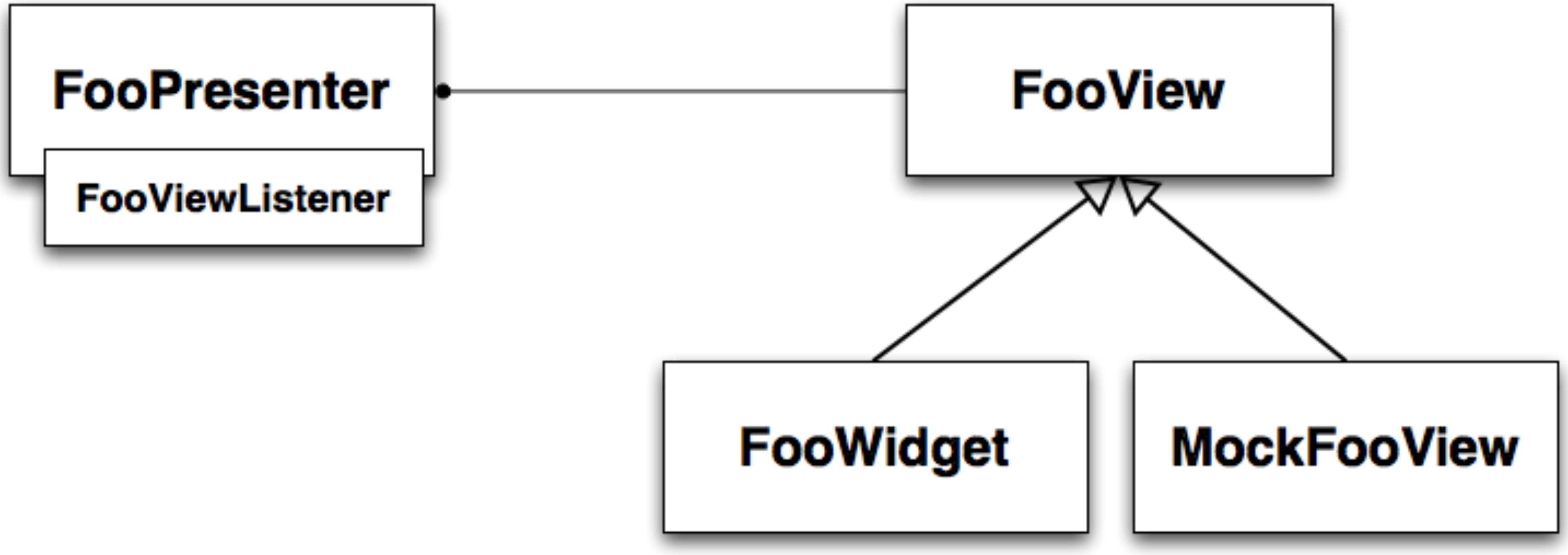
Goals of using MVP with GWT

- JSNI-dependent "view" code kept minimal and trivial
- Logic we want to test goes into "presenters"
- Use DI to hook up presenters with their views

UI Flow - MVC


Rendering Logic
View
Presentation State
Display
(hard to test)
Controller
UI Event Handling Logic

MVP - Example type hierarchy



Example - image thumbnail widget

```
interface ThumbnailView {

  interface Listener {
    void onClick();
  }
  void setListener(
               Listener);

  void setUrl(String);
  void setCaption(String);
}
```

A Thumbnail

Keeping in mind...

- Don't be prescriptive
- The goal is to write testable code, not to follow some rigid pattern or other
- MVP happens to fit well in most situations

Designing Good Presenters

- Have no transitive dependencies on JSNI
- Maintain full presentation state
- Don't have to be recyclable
  - Small "POJO" objects are relatively cheap - usually it's the widget instances we want to reuse
- May delegate handling to parent presenters, or use an event bus - whatever works

Designing Good Presenters

- Avoid the **new** keyword (except for value objects)
  - DI collaborators
  - Try Gin (http://code.google.com/p/google-gin/)

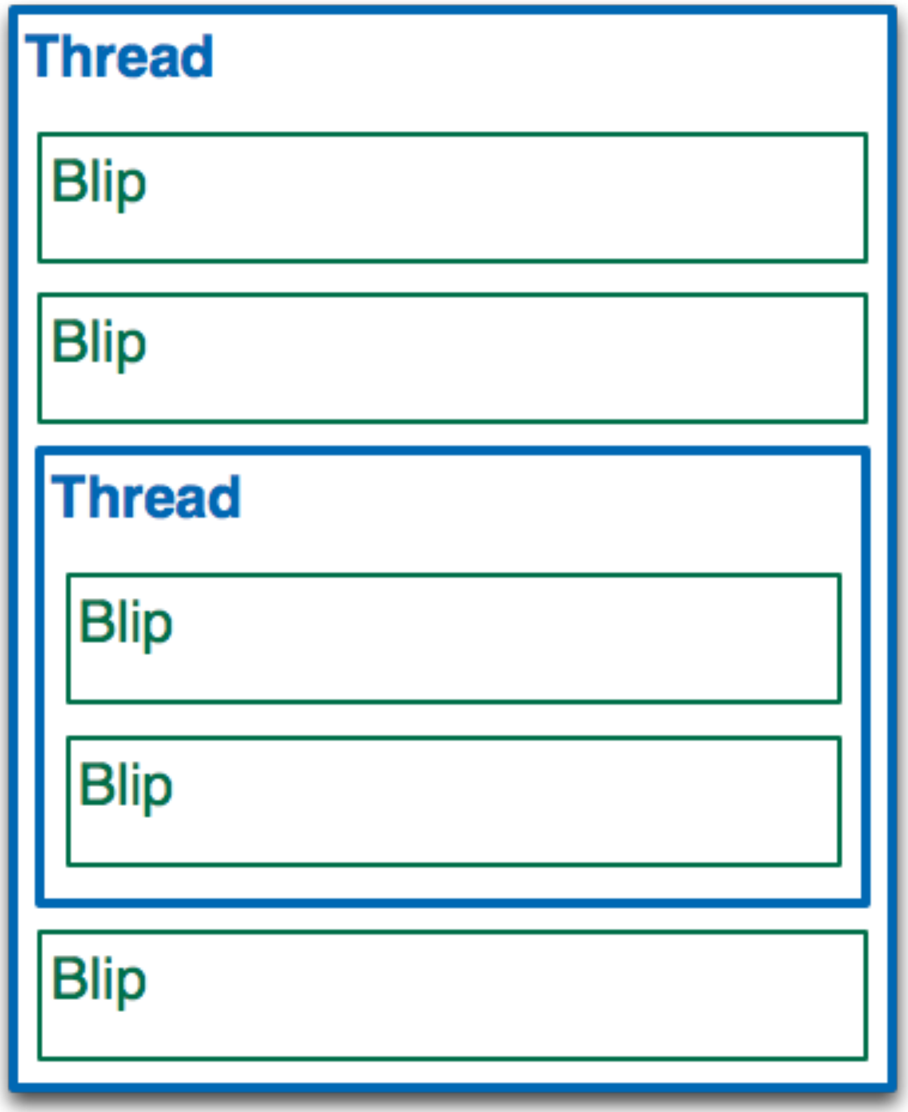
Designing Good View Interfaces

- Can be satisfied by an obvious, trivial implementation
- Avoid implying a particular layout or design
- Lack getters for view state (exceptions exist)

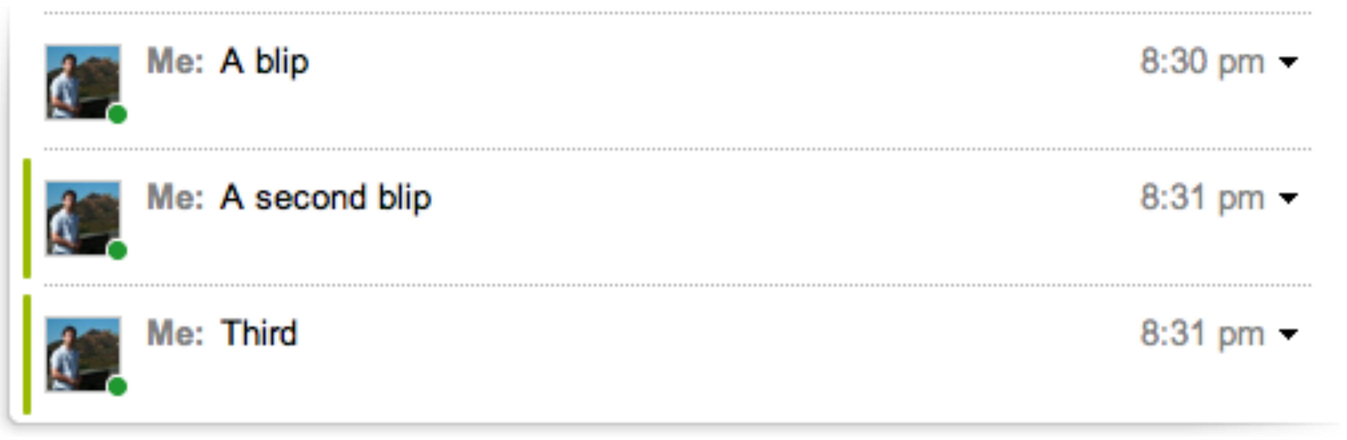
Designing Good View Interfaces - Events

- Views should generate events at their semantic level
- Prefer setListener to addListener
  (avoid > 1 listener)

Example - Wave Panel



Example - Wave Panel

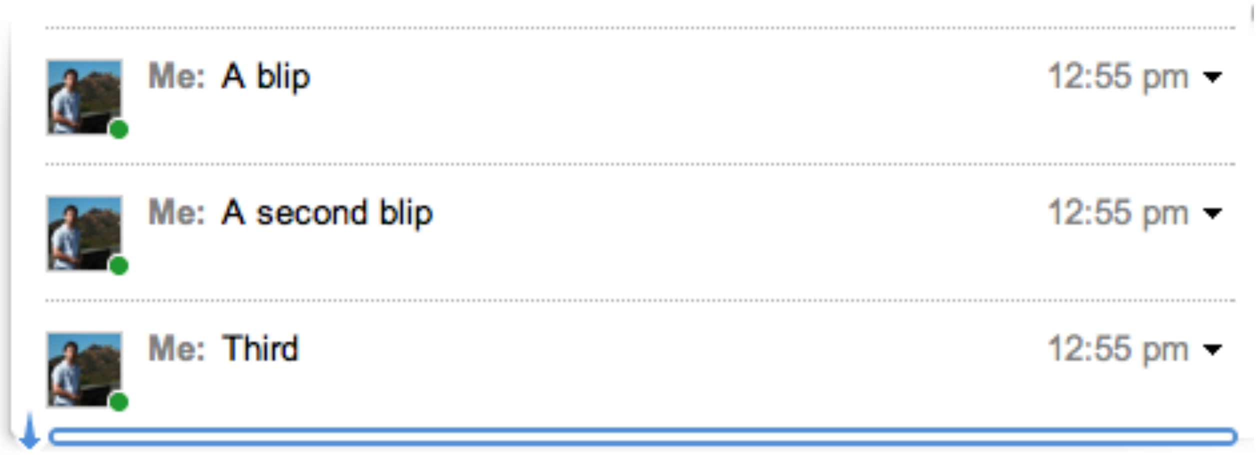


Me: A blip 8:30 pm

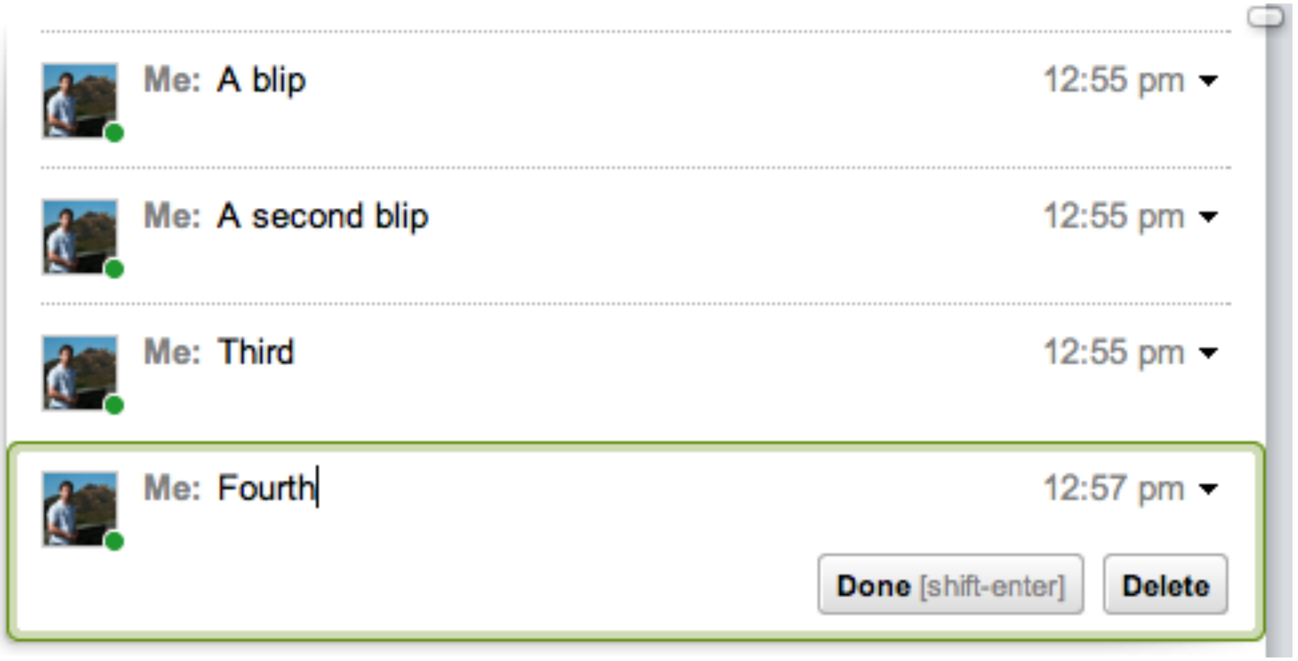
Me: A second blip 8:31 pm

Me: Third 8:31 pm

Example - Wave Panel


Me: A blip 12:55 pm
Me: A second blip 12:55 pm
Me: Third 12:55 pm

Example - Wave Panel



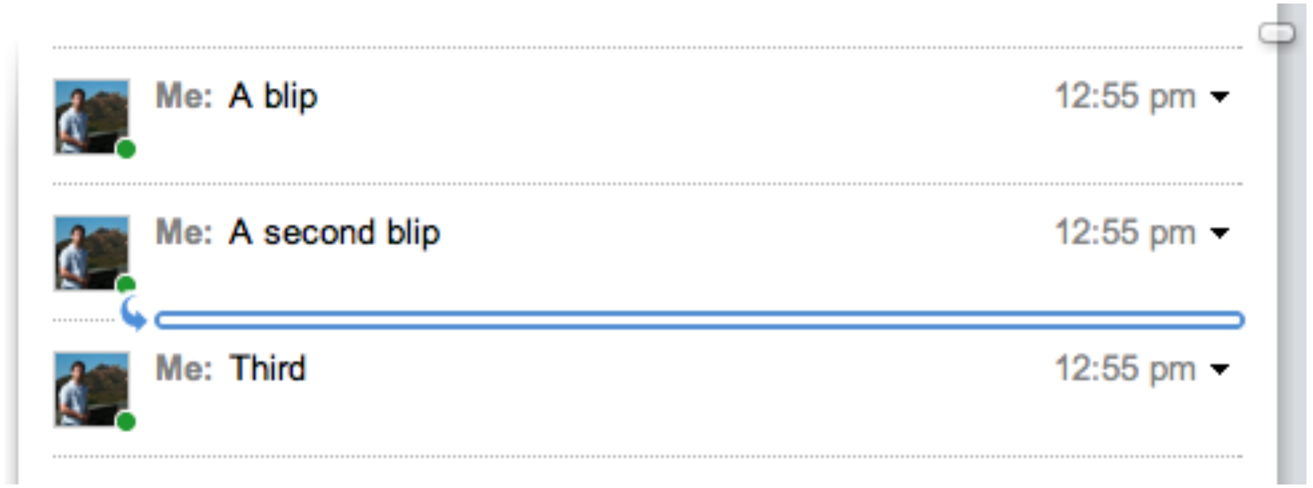
Me: A blip 12:55 pm

Me: A second blip 12:55 pm

Me: Third 12:55 pm

Me: Fourth 12:57 pm

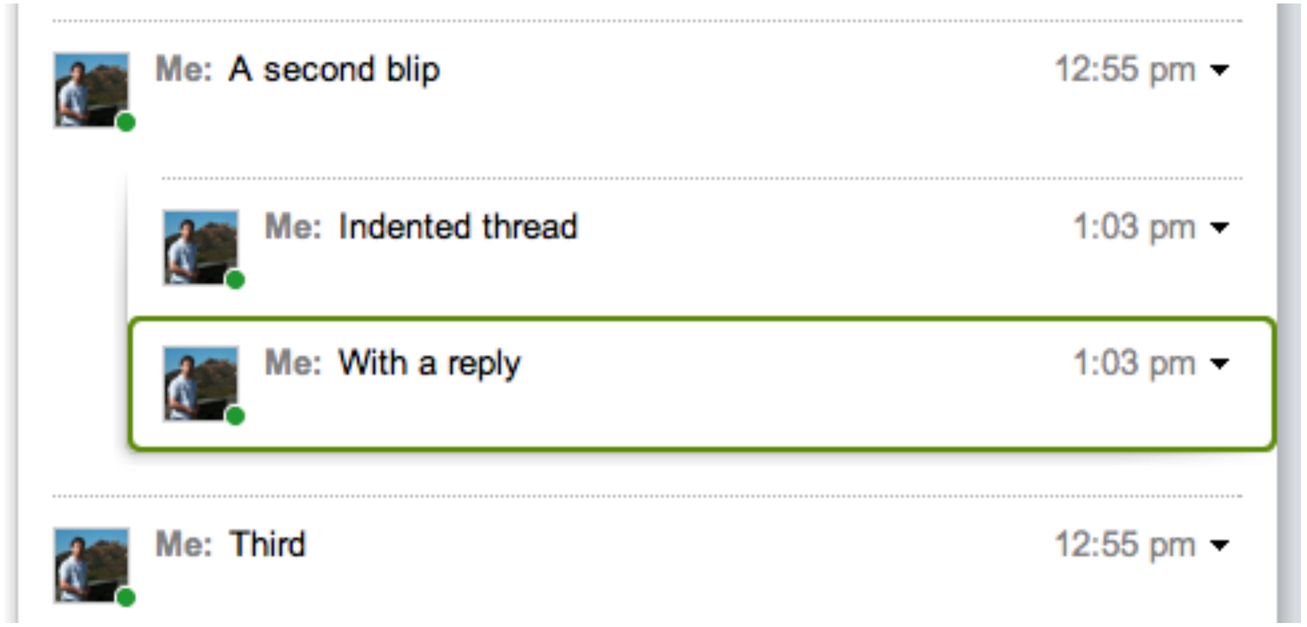
Example - Wave Panel


Me: A blip 12:55 pm

Me: A second blip 12:55 pm

Me: Third 12:55 pm

Example - Wave Panel


Me: A second blip 12:55 pm

Me: Indented thread 1:03 pm

Me: With a reply 1:03 pm

Me: Third 12:55 pm

Example - Wave Panel

```
interface ThreadView extends View {
  BlipView createBefore(View item);
  ThreadView createBranchBefore(View item);
}

interface BlipView extends View {
  IndicatorView getIndicator();
  void markUnread(boolean unread);
  void showDivider();
  void hideDivider();
}
```

Example - Wave Panel

```
interface ThreadView extends View {
  BlipView createBefore(View item);
  ThreadView createBranchBefore(View item);
}

interface BlipView extends View {
  IndicatorView getIndicator();
  void markUnread(boolean unread);
  void showDivider();
  void hideDivider();
}
```

Example - Wave Panel

```
interface ThreadView extends View {
  BlipView createBefore(View item);
  ThreadView createBranchBefore(View item);
}

interface BlipView extends View {
  IndicatorView getIndicator();
  void markUnread(boolean unread);
  void showDivider();
  void hideDivider();
}

interface IndicatorView { ... }
```

Example - Wave Panel

```
interface ThreadView extends View {
  BlipView createBefore(View item);
  ThreadView createBranchBefore(View item);
}

interface BlipView extends View {
  IndicatorView getIndicator();
  void markUnread(boolean unread);
  void showDivider();
  void hideDivider();
}
```

Example - Wave Panel

Creating a new dynamic sub component

```
// render the newly created model blip
private BlipPresenter renderNewModelBlip(Blip blip) {
  BlipView nextView = getView(blip.getNext());

  BlipView blipView = view.createBefore(nextView);
  BlipPresenter blipPresenter =
      blipPresenterFactory.create(blipView, blipPresenter);

  ... // update presentation state, mappings, etc

  return blipPresenter;
}
```

View Implementations - Testing

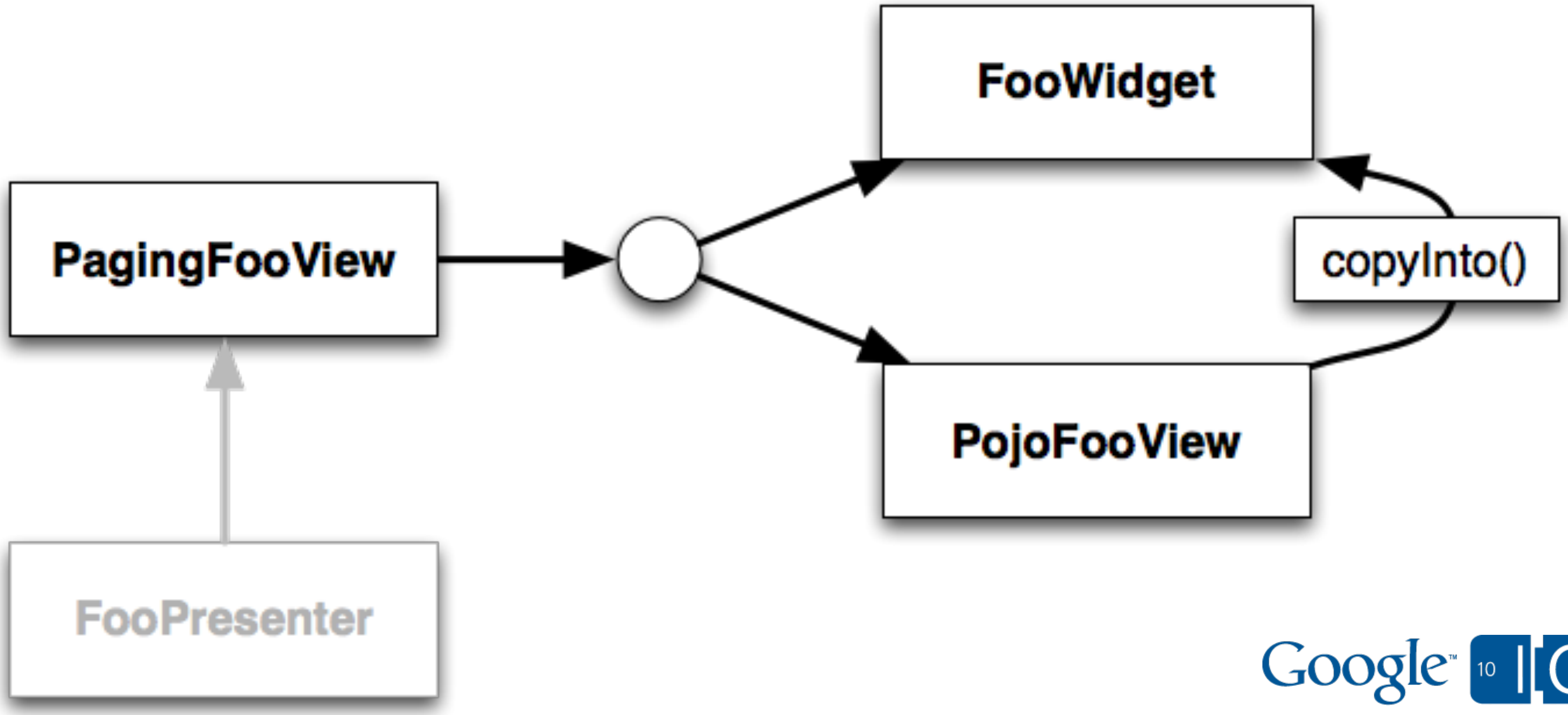
- Eclipse can generate most of a boilerplate implementation
- Or, in most cases should be trivial to mock without much boilerplate
  - e.g. mockito (http://mockito.org/)

View Implementations - Recyclable Widgets

- init() -> reset() recycling pattern
- Most of the expense of creating widgets is in the DOM, not the associated state
    - Makes sense to reuse widget views, not presenters (disposable presenters, reusable views)

View Implementations - Paging

- More efficient to render only the visible widgets
- Paging can be treated as a view concern
- Dumb view contract makes this possible.



Paging cont'd

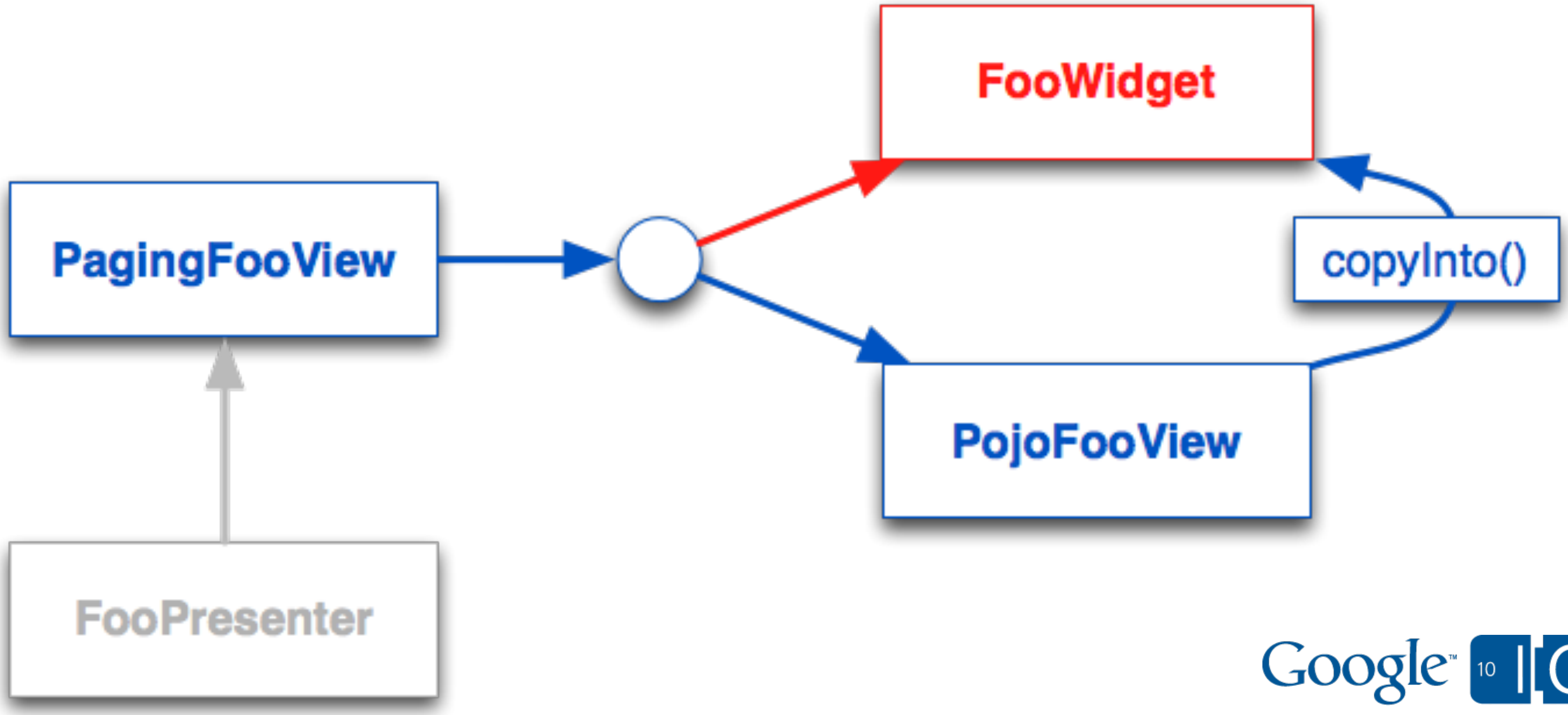
```
public class PagingBlipView implements BlipView {

  public void markUnread(boolean unread) {
    getImpl().markUnread(unread);
  }

  ...

}
```

Paging cont'd

```
public class PojoBlipView implements BlipView {
  boolean isUnread;

  public void markUnread(boolean unread) {
    isUnread = unread;
  }

  public void copyInto(BlipView other) {
    other.markUnread(isUnread);
    other...
    ...
  }

  ...
}
```

Paging cont'd

- The paging logic part of the view implementation is testable
- The Presenter code is unchanged and not conflated with paging logic



View Interfaces - Covariance
Rules of thumb

- Important to keep the View structure as a "closed universe"
    - Presenters can't decide the type of implementation
    - Different groups of collaborating view implementations can be injected in different contexts
- Type system covariance nice to have, but not strictly necessary.

MVP

Summary

- Dumb view contract
- Closed view universe
- Presenters maintain full presentation state

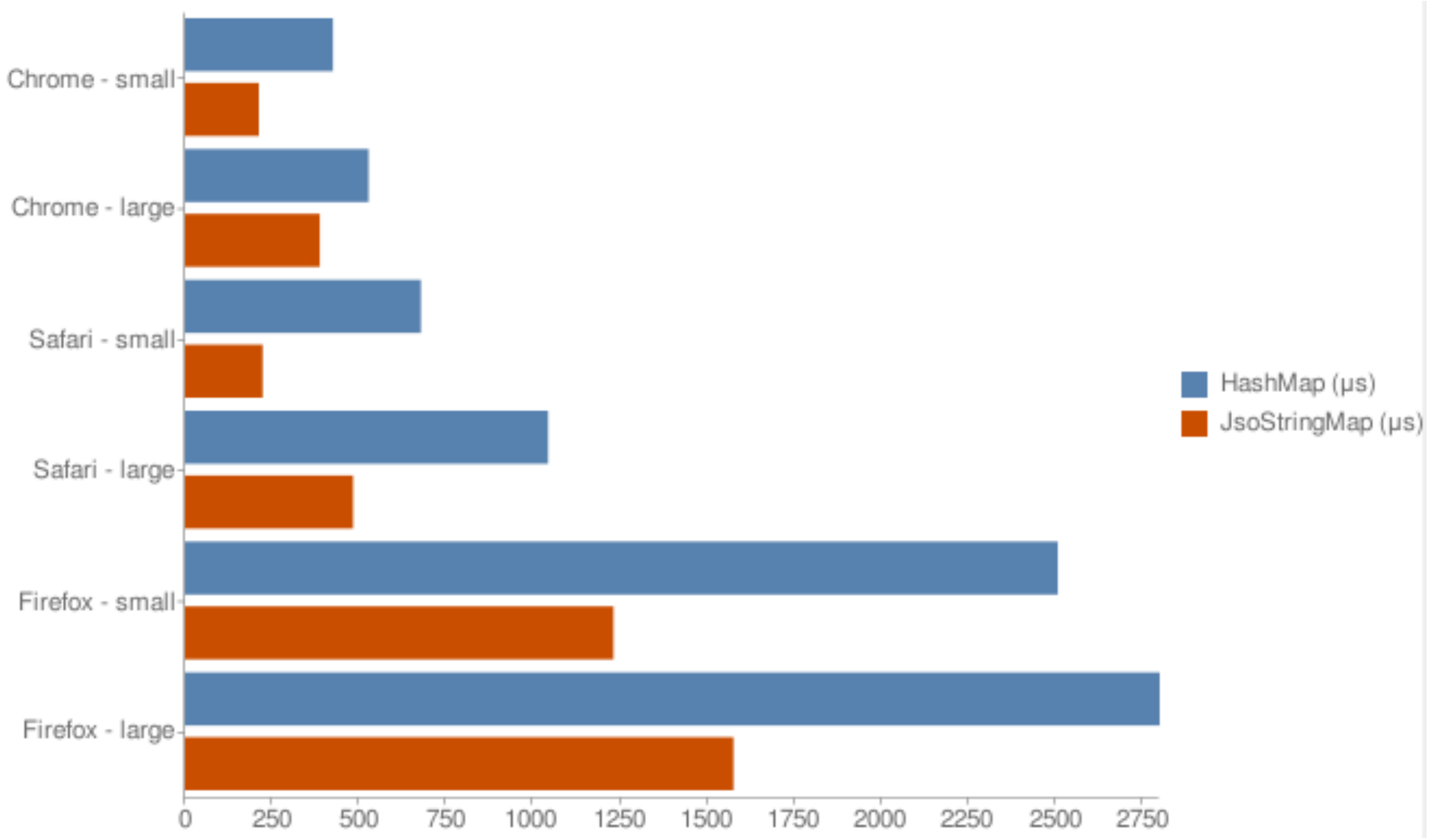
Javascript-specific optimizations

Need for web- or browser- specific optimizations

- DOM related - we can usually confine the solution inside our View implementations
- JS related - not so

Example: JSO data structures

- HashMap, HashSet etc. emulation is slow



Example: JSO data structures

- Efficient data structures
  StringMap, StringSet, NumberMap, IntMap,
  IdentityMap, IdentitySet, IntQueue
- Client: JSO based implementations (tuned per-browser)
- Testing/Server: java.util based implementations

Static Factories

- In practice we don't ever need to provide custom fake collections
- Static constructor methods delegating to a singleton factory are fine.
    - E.g. CollectionUtils.createStringMap()
- Use GWT.isScript() to switch

JsoCollectionFactory

```
public class JsoCollectionFactory
    implements CollectionFactory {

  public <V> StringMap<V> createStringMap() {
    return JsoStringMap.create();
  }

  public StringSet createStringSet() {
    return JsoStringSet.create()
  }

  // createIdentityMap, createIntMap, etc...
}
```

JavaCollectionFactory

```
public class JavaCollectionFactory
    implements CollectionFactory {

  public <V> StringMap<V> createStringMap() {
    return new StringMapAdapter<V>(
        new HashMap<String, V>());
  }

  public StringSet createStringSet() {
    return new new StringSetAdapter(
        new HashSet<String>());
  }

  // createIdentityMap, createIntMap, etc...
}
```

CollectionUtils

```
public class CollectionUtils {

 private static final CollectionFactory FACTORY =
    GWT.isScript() ? new JsoCollectionFactory()
                   : new JavaCollectionFactory();

 public static <V> StringMap<V> createStringMap() {
   return FACTORY.createStringMap();
 }

 // ... etc ...
}
```

Platform.java

- Sometimes, we don't want to depend on GWT at all (let alone just JSNI)
    - E.g. Share model code (and tests) on server side
- Use supersource

Platform.java - default

```
public class Platform {

  public static void initCollectionsFactory() {
    CollectionUtils.setFactory(
        JavaCollectionFactory.INSTANCE);
  }

}
```

Platform.java - default

```
<!-- Code in client gwt.xml file -->
<super-source path=""/>

// Code in Platform.java
public class Platform {

  public static void initCollectionsFactory() {
     if (GWT.isScript()) {
      CollectionUtils.setFactory(
         JsoCollectionFactory.INSTANCE);
    } else {
      CollectionUtils.setFactory(
         JavaCollectionFactory.INSTANCE);
    }
  }
}
```

Asynchronous logic

Asynchronous Logic

- GWTTestCase's delayTestFinish is evil (for small unit tests)
- Dependency inject a timer interface
    - Backed by a real timer in the application
    - Versatile fake for tests

Asynchronous Logic

Example interface

```
public interface TimerService {
  void schedule(Command task);
  void schedule(IncrementalCommand process);
  void scheduleDelayed(Command task, int minimumTime);
  void scheduleDelayed(
    IncrementalCommand process, int minimumTime);
  void scheduleRepeating(
    IncrementalCommand process, int minimumTime, int interval);

  void cancel(Schedulable job);

  boolean isScheduled(Schedulable job);

  double currentTimeMillis();
}
```

Events

Cross-Browser event normalizing

- Browser events are inconsistent
- E.g.
    - "delete" and "." have the same key code - different ways to distinguish them in FF vs Webkit
    - Key repeat behavior varies between browser and key

"Signal" - Event-like interface

```
interface Signal {
  Type getType();
  int getKeyCode();
  ...
}
```

"Signal" events

```
public void onBrowserEvent(Event rawEvent) {
  Signal event = SignalImpl.create(rawEvent);

  // Ignore redundant events
  if (event == null) {
    return;
  }

  if (event.isKey(DELETE)) {
    listener.onDelete();
  } else if (event.isInput()) {
    listener.onUserEditing();
  }
}
```

Test strategy

- Record event data
    - Use VNC + webdriver
    - For each browser/OS/input method combination
- Factor non-trivial logic in SignalImpl.create() into a testable method
- Tests

"Signal" events

```
public void testBasics() {
  for (Environment env : Environments.ALL) {
    checkSignals(env, "TAB", 0, INPUT, 9);
    checkSignals(env, "TAB", SHIFT, INPUT, 9);
    checkSignals(env, "DEL", 0, DELETE, 46);
    checkSignals(env, "LEFT", 0, NAVIGATION, 37);
    checkSignals(env, "ESC", 0, NOEFFECT, 27);
  }
}

private void checkSignals(Environment env,
    String key, int modes, int repeat,
    KeySignalType type, int keyCode) {

  // Pass in inputs, check outputs
}
```

"Signal" events

```
public void testAltGr() {
  for (Environment env : Environments.ALL) {
    if (env.layout != KeyboardLayout.DE ||
        env.os == OperatingSystem.MAC) {
      continue;
    }

    checkSignals(env, "2", SHIFT, INPUT, "\"");
    checkSignals(env, "2", ALTGR |
        NO_ALTGR_OUTPUT, INPUT, 178);
    checkSignals(env, "Q", ALTGR |
        NO_ALTGR_OUTPUT, INPUT, '@');
  }
}
```

SingleJsoImpl

JavaScriptObject (Jso)

- Used as an interface to raw browser javascript objects
    - E.g. DOM objects, or regular objects from a js library
- All methods in a JSO subtype must be effectively final
    - They are essentially syntactic sugar for static methods
- Cannot be constructed by Java code
    - Can only be instantiated as return values from native methods
    - Must have an empty, no-args, protected constructor

SingleJsoImpl

- Any interface may be implemented by a JavaScriptObject subtype
  - An interface method may have at most one implementation defined within a JSO subtype
  - This has nothing to do with whether or not the method implementation itself is native
  - Compiler knows to substitute interface method invocations with direct calls to the implementation
- The interface may still be implemented by any number of methods declared in non-JSO subtypes
  - If such methods exist, there will be a runtime dispatch penalty

Using Single Jso Impl - Collections

```
public final class JsoStringMap<V>
    extends JavaScriptObject
    implements StringMap<V> {

  public final void put(String key, V value) {
    JsoView.as(this).set(escape(key), value);
  }

  private static String escape(String key) {
    return (funky optimized escaping code)
  }

  ...
}
```

Using Single Jso Impl

```
public final class SignalImpl
    extends JavaScriptObject
    implements Signal {

  public getKeyCode() {
    return Event.as(this).getKeyCode();
  }
  ...
}
```

Test Harnesses

Test Harnesses

- Useful for fast feature development
- Help isolate performance problems
- Fight against dependency creep
- Can be easily packed with debugging hooks for Webdriver/Selenium "unit" tests
    - And build faster, so the test runs faster

Summary

Avoid non-trivial logic in hard-to-test code

View live notes and ask questions about this session on Google Wave: http://bit.ly/io2010-gwt7