



# Google Web Toolkit Best Practices

Chris Ramsdale  
Developer Relations, Google



# Quick Overview

---

- Development toolkit, not a framework
- Code in Java, run as Javascript
- One codebase, any browser
- Makes Ajax a piece of cake...and faster
- Used within many Google products, including Google Wave and Ad Words



# New in GWT 2.0

- Development Mode
- Speed Tracer
- Developer Guided Code Splitting
- Compiler Optimizations
- Draft Compile
- Declarative User Interfaces
- Layout Panels
- Bundled Resources via ClientBundle



# New in GWT 2.0

---

- Development Mode
- Speed Tracer
- Developer Guided Code Splitting
- Compiler Optimizations
- Draft Compile
- Declarative User Interfaces
- Layout Panels
- Bundled Resources via ClientBundle



# New in GWT 2.0

---

- Development Mode
- Speed Tracer
- Developer Guided Code Splitting
- Compiler Optimizations
- Draft Compile
- Declarative User Interfaces
- Layout Panels
- Bundled Resources via ClientBundle



# GWT Best Practices

---

- Model View Presenter
- Declarative UI
- Bundled Resources
- Code Splitting
- Prefetching RPCs

# The “direct” approach

Write a bunch of widgets with self-contained logic

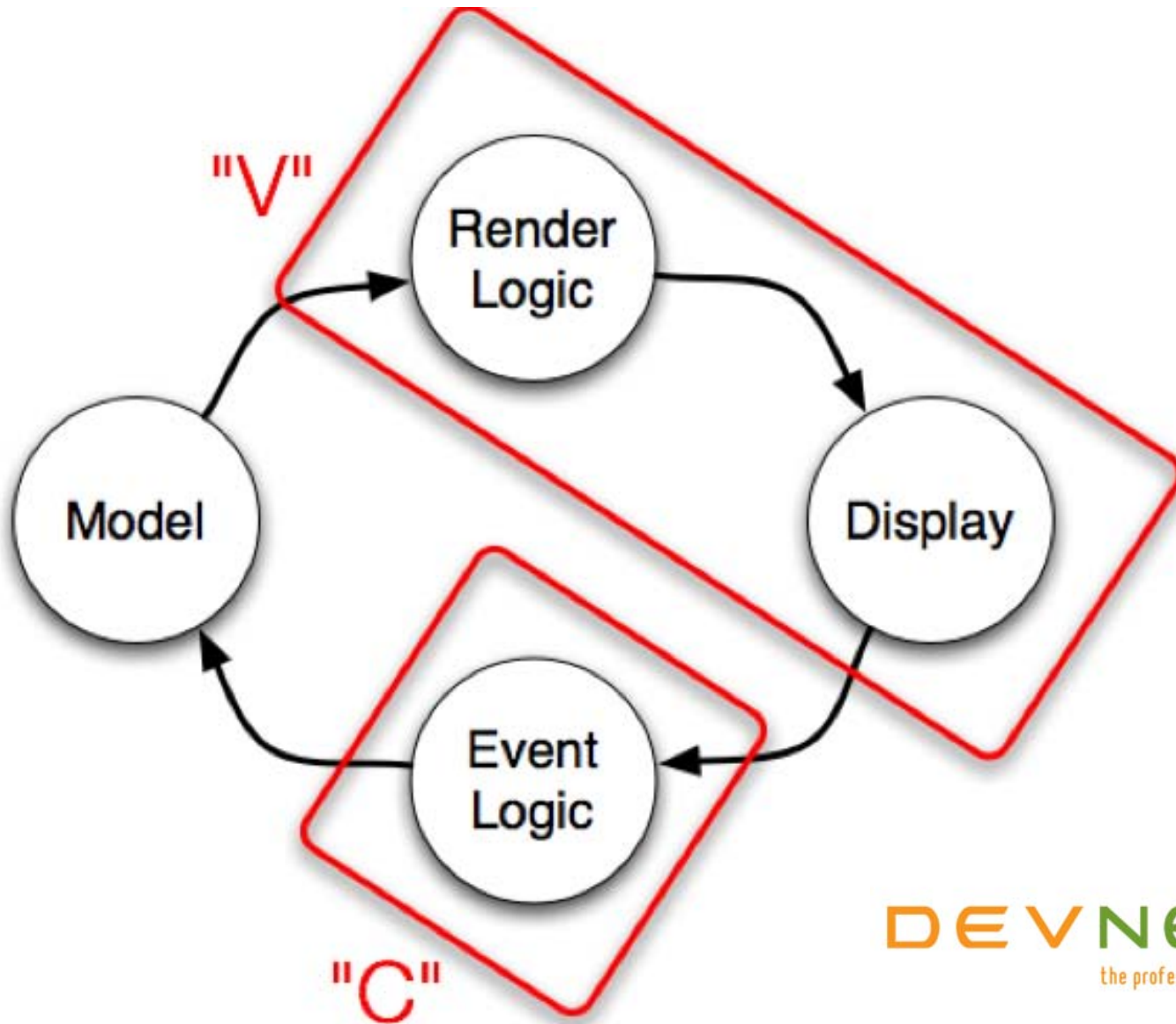
## Problems:

- Hard to test - need GWTTestCase
- Mocks not encouraged - harder to write smaller tests
- Platform specific UI code - limits code reuse
- Too many dependencies - difficult to optimize
- Too many dependencies - difficult to optimize

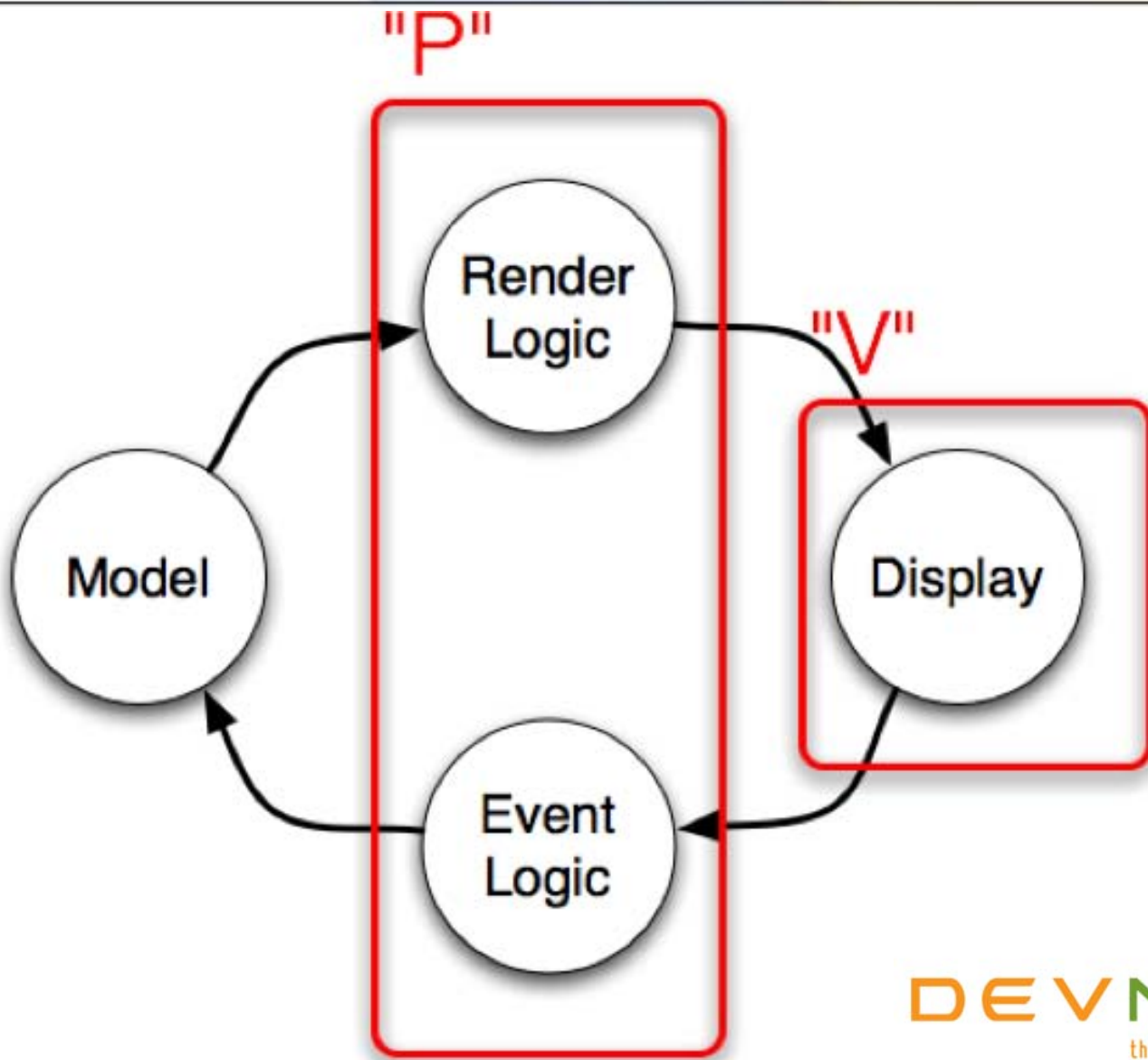
# MVP (Model-View-Presenter)

- Be practical
- The goal is not to follow some rigid pattern
- Put the complex logic in your Presenters
- Try to keep widget code contained within Views
- Make the Views as dumb as possible

# MVC



# MVP



# Example - Contacts

Features Display a list of contacts within a table

- Clicking displays contact information
- Selecting enables multi-delete
- Selecting enables multi-delete

MVP Components ContactDetails - Model

- ContactsPresenter - Presenter
- ContactsView - View
- ContactsView - View

# Example - Contacts

Features Display a list of contacts within a table

- Clicking displays contact information
- Selecting enables multi-delete
- Selecting enables multi-delete

MVP Components ContactDetails - Model

- ContactsPresenter - Presenter
- ContactsView - View
- ContactsView - View



# Example - Contacts

## Contact List UI

- Abigail Louis
- Bell Snedden
- Brigitte Cobb
- Bulrush Bouchard
- Candice Carson
- Chad Andrews
- Christina Blake

# ContactsView

```
public interface ContactsView<T> {  
  
    public interface Presenter<T> {  
        void onAddButtonClicked();  
        void onDeleteButtonClicked();  
        void onItemClick(T clickedItem);  
        void onItemClickSelected(T selectedItem);  
    }  
  
    void setPresenter(Presenter<T> presenter);  
    void setColumnDefinitions(  
        List<ColumnDefinition<T>> columnDefinitions);  
    void setRowData(List<T> rowData);  
}
```

# ContactsView Layout

```
<ui:UiBinder
  xmlns:ui="urn:ui:com.google.gwt.uibinder"
  xmlns:g="urn:import:com.google.gwt.user.client.ui">

  <ui:style>
    .contactsViewButtonHPanel {
      margin: 5px 0px 0x 5px;
    }
    .contactsViewContactsFlexTable {
      margin: 5px 0px 5px 0px;
    }
  </ui:style>

  <g:DecoratorPanel>
    <g:VerticalPanel>
      <g:HorizontalPanel addStyleNames="{style.contactsViewButtonHPanel}">
        <g:Button ui:field="addButton">Add</g:Button>
        <g:Button ui:field="deleteButton">Delete</g:Button>
      </g:HorizontalPanel>
      <g:FlexTable ui:field="contactsTable"
        addStyleNames="{style.contactsViewContactsFlexTable}"/>
    </g:VerticalPanel>
  </g:DecoratorPanel>
</ui:UiBinder>
```

# Before UiBinder

```
public ContactsView() {
    DecoratorPanel contentTableDecorator = new DecoratorPanel();
    initWidget(contentTableDecorator);
    contentTableDecorator.setWidth("100%");
    contentTableDecorator.setWidth("18em");

    contentTable = new FlexTable();
    contentTable.setWidth("100%");
    contentTable.getCellFormatter().addStyleName(0, 0, "contacts-ListContainer");
    contentTable.getCellFormatter().setWidth(0, 0, "100%");
    contentTable.getFlexCellFormatter().setVerticalAlignment(0, 0, DockPanel.ALIGN_TOP);

    // Create the menu
    //
    HorizontalPanel hPanel = new HorizontalPanel();
    hPanel.setBorderWidth(0);
    hPanel.setSpacing(0);
    hPanel.setHorizontalAlignment(HorizontalPanel.ALIGN_LEFT);
    addButton = new Button("Add");
    hPanel.add(addButton);
    deleteButton = new Button("Delete");
    hPanel.add(deleteButton);
    contentTable.getCellFormatter().addStyleName(0, 0, "contacts-ListMenu");
    contentTable.setWidget(0, 0, hPanel);

    // Create the contacts list
    //
    contactsTable = new FlexTable();
    contactsTable.setCellSpacing(0);
    contactsTable.setCellPadding(0);
    ....
}
```

# ContactsView Layout

```
<ui:UiBinder
  xmlns:ui="urn:ui:com.google.gwt.uibinder"
  xmlns:g="urn:import:com.google.gwt.user.client.ui">

  <ui:style>
    .contactsViewButtonHPanel {
      margin: 5px 0px 0x 5px;
    }
    .contactsViewContactsFlexTable {
      margin: 5px 0px 5px 0px;
    }
  </ui:style>

  <g:DecoratorPanel>
    <g:VerticalPanel>
      <g:HorizontalPanel addStyleNames="{style.contactsViewButtonHPanel}">
        <g:Button ui:field="addButton">Add</g:Button>
        <g:Button ui:field="deleteButton">Delete</g:Button>
      </g:HorizontalPanel>
      <g:FlexTable ui:field="contactsTable"
        addStyleNames="{style.contactsViewContactsFlexTable}"/>
    </g:VerticalPanel>
  </g:DecoratorPanel>
</ui:UiBinder>
```

# UI Events

- Minimize View code

```
@UiHandler("deleteButton")
void onDeleteButtonClicked(ClickEvent event) {
    if (presenter != null) {
        presenter.onDeleteButtonClicked();
    }
}
```

- Keep the complex logic in the Presenter

```
public void onDeleteButtonClicked() {
    /* Retrieve selected items
    * Make RPC
    * Put results on EventBus
    */
}
```

# Complex UIs

## Goals

- We need dumb Views, not dumb UIs
- Avoid state within Views
- Swap out Views for different platforms

# Invite a third party

---

## **Enter ColumnDefinition(s)**

- Column data is generally homogenous
- Generics abstract specific knowledge of DTOs/model
- Set within the View - unknown to the Presenter
- Will need to be tested with GWTTTestCase

# ColumnDefinition

```
public abstract class ColumnDefinition<T> {  
    public abstract Widget render(T t);  
  
    public boolean isClickable() {  
        return false;  
    }  
  
    public boolean isSelectable() {  
        return false;  
    }  
}
```

# ColumnDefinition

```
columnDefinitions.add(new ColumnDefinition<ContactDetails>() {  
    public Widget render(ContactDetails c) {  
        return new CheckBox();  
    }  
  
    public boolean isSelectable() {  
        return true;  
    }  
});
```

```
columnDefinitions.add(new ColumnDefinition<ContactDetails>() {  
    public Widget render(ContactDetails c) {  
        return new HTML(c.getDisplayName());  
    }  
  
    public boolean isClickable() {  
        return true;  
    }  
});
```

# Meaningful data as params

## Pass DTOs straight into Views

ContactsPresenter.java

```
rpcService.getContactDetails(new AsyncCallback<ArrayList<ContactDetails>>() {  
    public void onSuccess(ArrayList<ContactDetails> result) {  
        view.setRowData(result);  
    }  
    ...  
});
```

# Meaningful data as params

## Let ColumnDefinitions sort out the details

Contacts ViewImpl.java

```
public void setRowData(List<T> rowData) {
    this.rowData = rowData;

    for (int i = 0; i < rowData.size(); ++i) {
        T t = rowData.get(i);
        for (int j = 0; j < columnDefinitions.size(); ++j) {
            ColumnDefinition<T> columnDefinition = columnDefinitions.get(j);
            contactsTable.setWidget(i, j, columnDefinition.render(t));
        }
    }
}
```

# Meaningful data as params

- Differentiate between clicks and selects

```
private boolean shouldFireClickEvent(HTMLTable.Cell cell) {  
    return columnDefinitions.get(cell.getCellIndex()).isClickable();  
}
```

```
private boolean shouldFireSelectEvent(HTMLTable.Cell cell) {  
    return columnDefinitions.get(cell.getCellIndex()).isSelectable();  
}
```

- Pass model objects

```
if (shouldFireClickEvent(cell)) {  
    presenter.onItemClicked(rowData.get(cell.getRowIndex()));  
}  
if (shouldFireSelectEvent(cell)) {  
    presenter.onItemSelected(rowData.get(cell.getRowIndex()));  
}
```

# SelectionMode

## Nothing more than a list of DTOs

```
public class SelectionModel<T> {  
    List<T> selectedItems = new ArrayList<T>();  
  
    public List<T> getSelectedItems() {  
        return selectedItems;  
    }  
  
    public void addSelection(T item) {  
        selectedItems.add(item);  
    }  
  
    public void removeSelection(T item) {  
        selectedItems.remove(item);  
    }  
  
    public boolean isSelected(T item) {  
        return selectedItems.contains(item);  
    }  
}
```

# SelectionMode

## Maintain state within the Presenter

```
private void deleteSelectedContacts() {  
    List<ContactDetails> selectedContacts = selectionModel.getSelectedItems();  
    ArrayList<String> ids = new ArrayList<String>();  
  
    for (int i = 0; i < selectedContacts.size(); ++i) {  
        ids.add(selectedContacts.get(i).getId());  
    }  
  
    rpcService.deleteContacts(ids, new AsyncCallback<ArrayList<ContactDetails>>() {  
  
    });  
}
```

# Testing

## Goals

- High code coverage
- Run fast (@SmallTest JUnit)
- GWTTTestCase/Selenium/Webdriver for integration testing only
- Minimize use of GWTTTestCase

# Testing Example

Delete button -> correct RPC -> correct action

```
public void testDeleteButton() {
    contactDetails = new ArrayList<ContactDetails>();
    contactDetails.add(new ContactDetails("0", "a_contact"));
    contactsPresenter.setContactDetails(contactDetails);

    mockRpcService.deleteContacts(isA(ArrayList.class),
        isA(AsyncCallback.class));

    expectLastCall().andAnswer(new IAnswer() {
        public Object answer() throws Throwable {
            final Object[] arguments = getCurrentArguments();
            AsyncCallback callback =
                (AsyncCallback) arguments[arguments.length - 1];
            callback.onSuccess(new ArrayList<ContactDetails>());
            return null;
        }
    });
    ...
}
```

# Testing Example

Delete button -> correct RPC -> correct action

```
public void testDeleteButton() {  
    ...  
  
    replay(mockRpcService);  
    contactsPresenter.onDeleteButtonClicked();  
    verify(mockRpcService);  
  
    List<ContactDetails> updatedContactDetails =  
        contactsPresenter.getContactDetails();  
  
    assertEquals(0, updatedContactDetails.size());  
}  
}
```



# Optimizations

---

- Resource bundling
- Code splitting
- Prefetching RPCs

# Resource Bundling

- Example - associating icons with a contact





# Resource Bundling

## One at a time

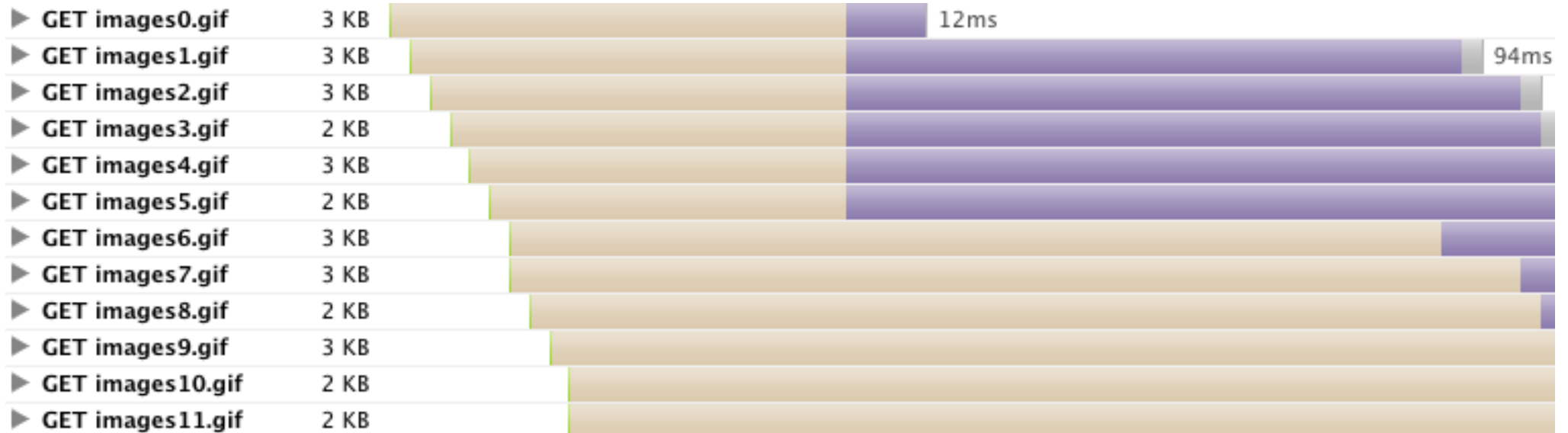
```
Image image = new Image("images/images" + i + ".gif");  
image.setHeight("50px");  
image.setWidth("50px");  
imagesPanel.add(image);
```

# Resource Bundling

## Initial download



## Call to display images



# Resource Bundling

## All at once

```
public interface Resources extends ClientBundle {  
    public static final Resources INSTANCE = GWT.create(Resources.class);  
  
    @Source("Contacts.css")  
    public ContactsCss contactsCss();  
  
    @Source("images0.gif")  
    public ImageResource image0();  
  
    @Source("images1.gif")  
    public ImageResource image1();  
  
    ...  
}
```



# Resource Bundling

## Initial download



## Call to display images



# Code splitting

## Initial download



Not needed on startup

# Code splitting

## Split points - runAsync()

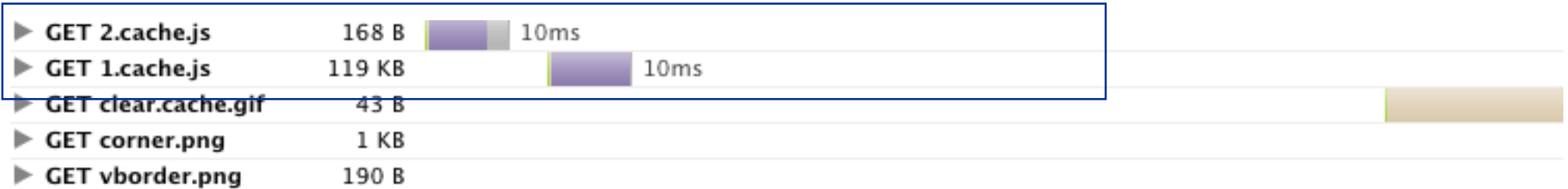
```
@UiHandler("showImagesButton")
void onOkClicked(ClickEvent event) {
    GWT.runAsync(new RunAsyncCallback() {
        public void onSuccess() {
            showImagesDialog();
        }
    });
}
```

# Code splitting

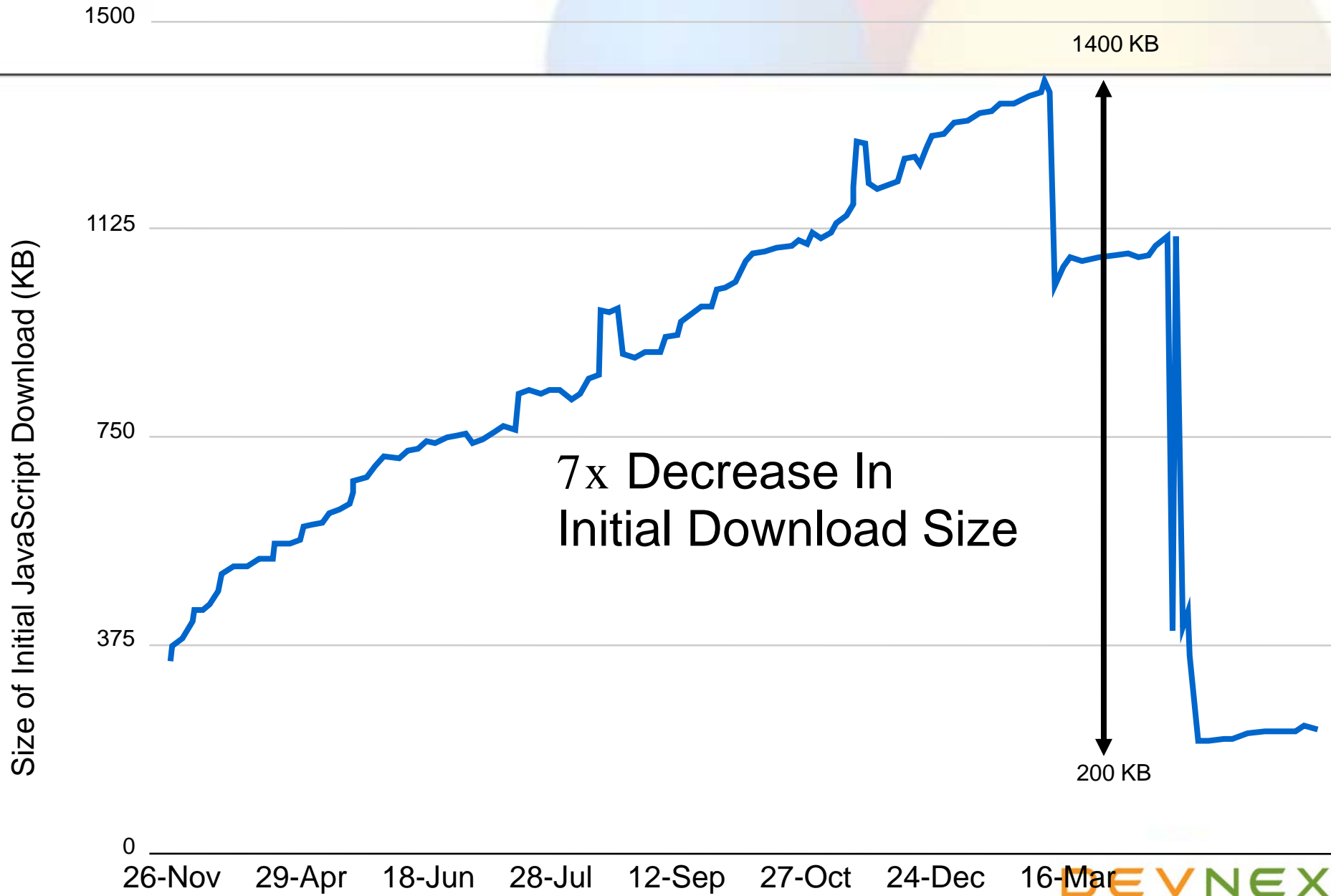
## Initial download



## Call to display images

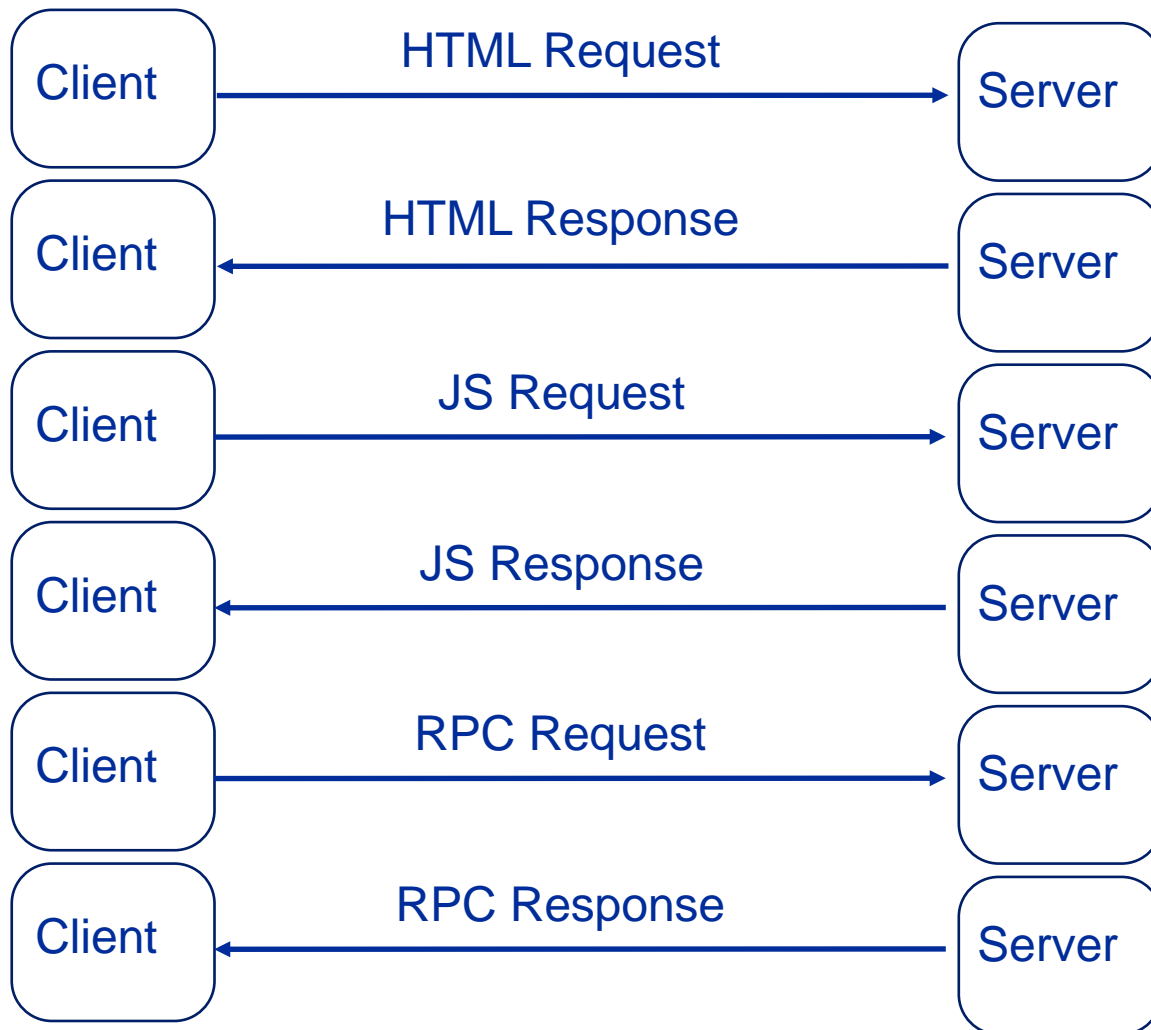


# Real world results - Google Wave



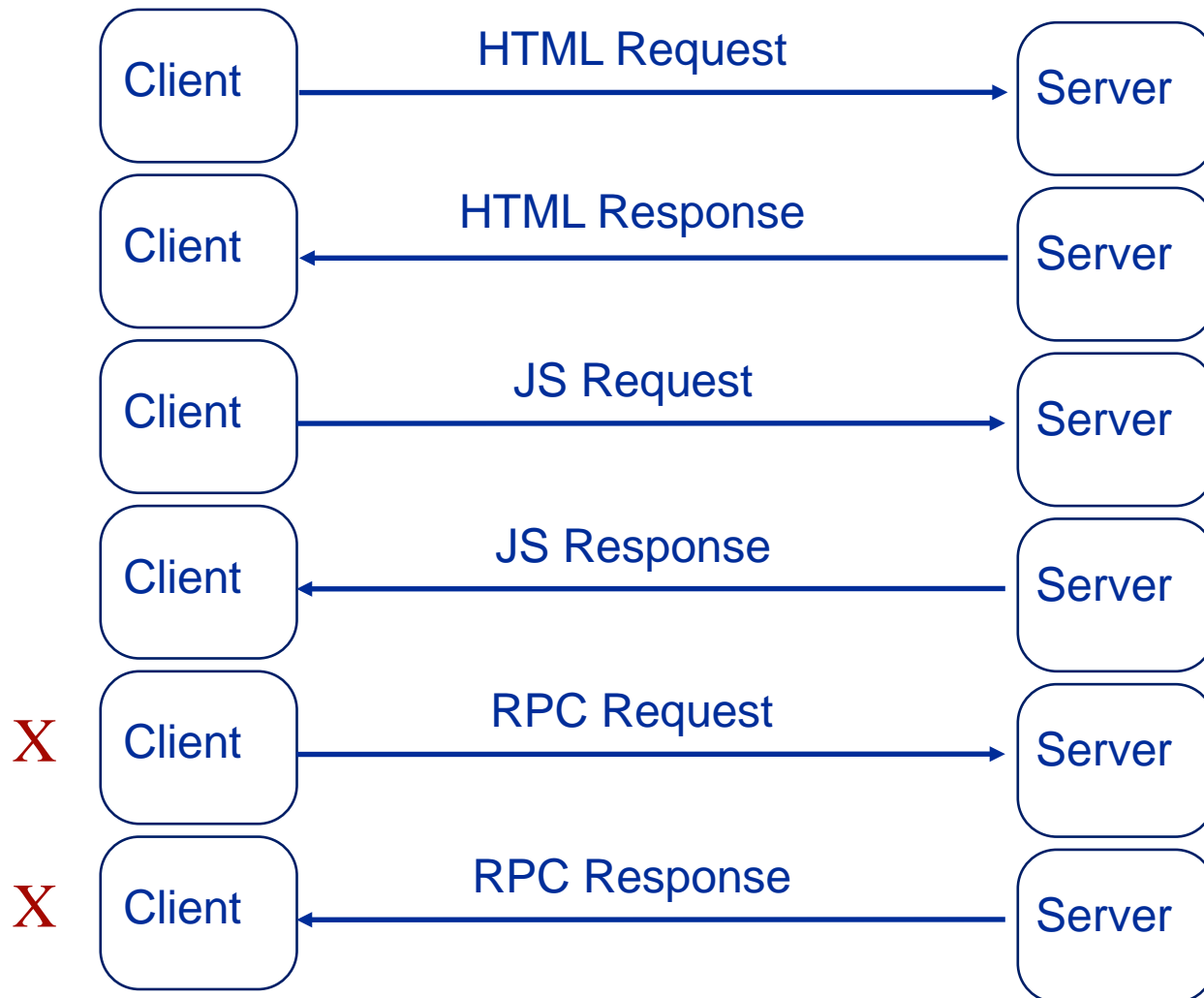
# Prefetching RPCs

## Typical app initialization



# Prefetching RPCs

## Typical app initialization



# Prefetching RPCs

## Embed the RPC payload in your HTML

```
private String getEncodedData() {  
    Method serviceMethod =  
        ContactsService.class.getMethod("getContactDetails");  
  
    ArrayList<ContactDetails> contactDetails =  
        new ContactsServiceImpl().getContactDetails();  
  
    SerializationPolicy serializationPolicy = getSerializationPolicy();  
  
    encodedResponse = RPC.encodeResponseForSuccess(serviceMethod,  
        contactDetails, serializationPolicy);  
}
```

Server

# Prefetching RPCs

## Decode it on the client

```
private static native String getPrefectedData() /*-{  
    return $wnd._prefetchedData;  
}-*/;
```

```
private void initContactDetails() {  
    String data = getPrefectedData();
```

```
    SerializationStreamFactory streamFactory =  
        (SerializationStreamFactory)rpcService;  
    try {  
        ArrayList<ContactDetails> contactDetails =  
            streamFactory.createStreamReader(data).readObject();
```

```
    } catch (SerializationException e) {
```

```
        ...  
    }  
}
```

Client

# Recap

- GWT 2.0 has features to help with large scale development
- Vanilla JRE tests are king
- The MVP pattern can help
- Make your Views dumb, but don't skimp on the UI
- Once testing is in place, focus on optimizations



</presentation>

Chris Ramsdale

[cramsdale@google.com](mailto:cramsdale@google.com)

<http://googlewebtoolkit.blogspot.com/>