Google™ 09 I/O


Google™ 09 I/O

The Story of Your Compile:

Reading the Tea Leaves of the GWT Compiler for an Optimized Future

Lex Spoon and Bruce Johnson
May 28, 2009

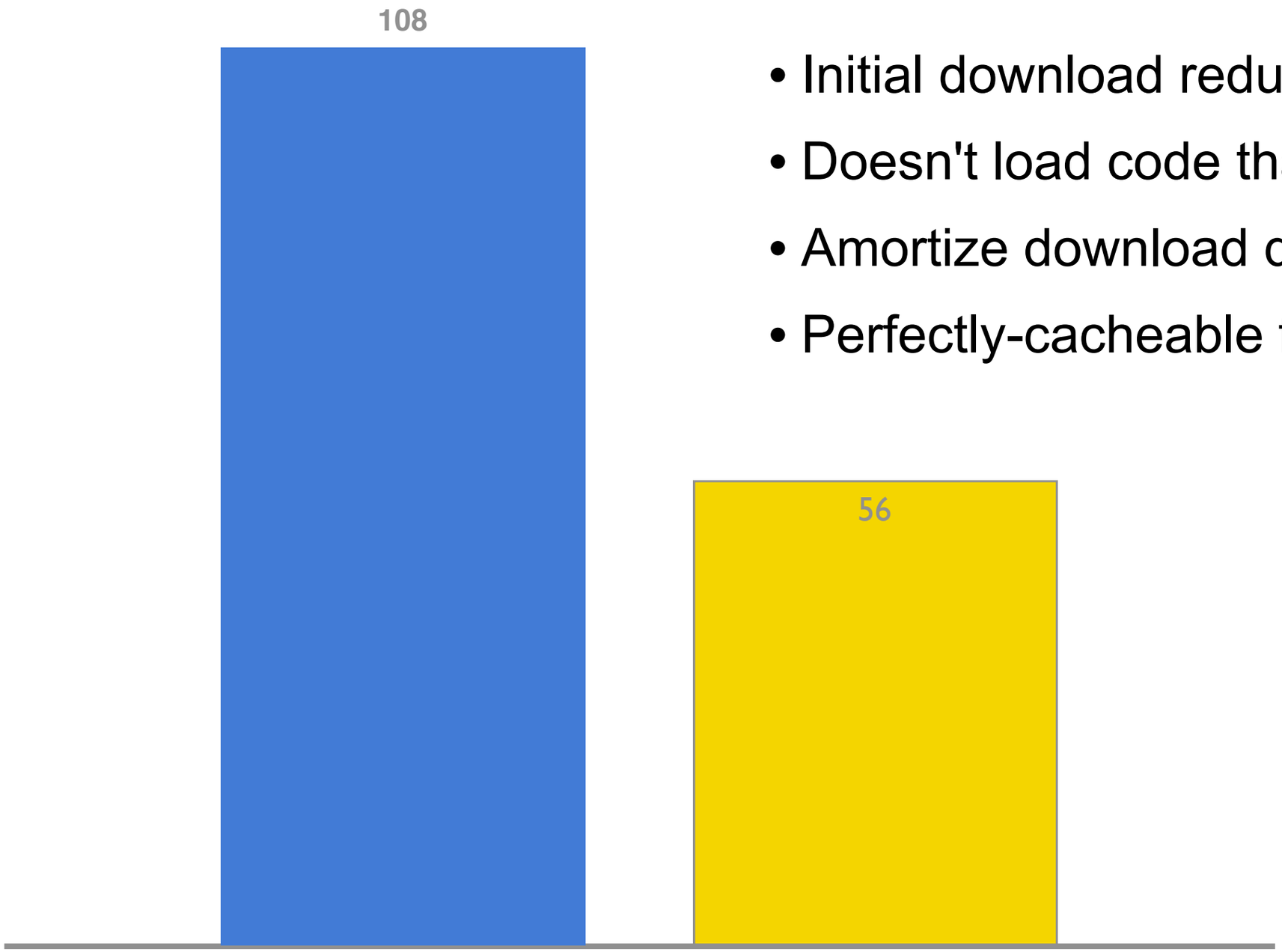
Google™ 09 I/O

Would You Take This Deal?

Tweak your app for 1 hour

Reduce your startup script size by 50%

Leverage, I Say!



- Initial download reduced by half
- Doesn't load code that doesn't run
- Amortize download delay
- Perfectly-cacheable fragments

Compiled with GWT trunk, r5406

Topics

- Should we really care about optimization?
- Stuff we should've said last year
- Code splitting in GWT 2.0
- Code splitting walkthrough
- The story of your compile
- Async package pattern

Should We Really Care About Optimization?

YES

Stuff We Should've Said Last Year

Deciphering RPC

- As of GWT 1.6, pass `-extra` flag to the compiler for extras
- *`module`*`.rpc.log` explains the RPC code generator logic
- See `dynatable.rpc.log`

Watching the Compiler Optimize

-Dgwt.jjs.traceMethods = "ShapeExample.onModuleLoad"

```
public class ShapeExample implements EntryPoint {
  private static final double SIDE_LEN_SMALL = 2;
  private final Shape shape = new SmallSquare();

  public static abstract class Shape {
    public abstract double getArea();
  }

  public static abstract class Square extends Shape {
    public double getArea() { return getSideLength() * getSideLength(); }
    public abstract double getSideLength();
  }

  public static class SmallSquare extends Square {
    public double getSideLength() { return SIDE_LEN_SMALL; }
  }

  public void onModuleLoad() {
    Shape shape = getShape();
    Window.alert("Area is " + shape.getArea());
  }

  private Shape getShape() { return shape; }
}
```

Better, eh?

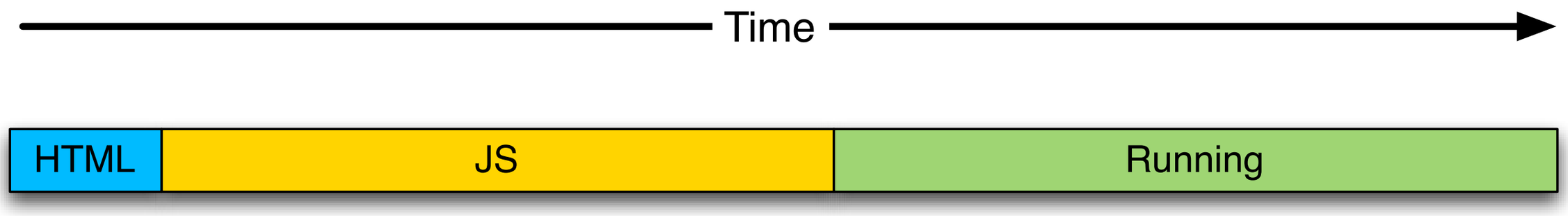
```
public class ShapeExample implements EntryPoint {
  public void onModuleLoad() {
    Window.alert("Area is 4.0");
  }
}
```

Code Splitting in GWT 2.0

“We strongly prioritize features that can make the biggest differences to end users. Obviously, we want to make developers' lives easier, too, but never at the expense of the user experience.”

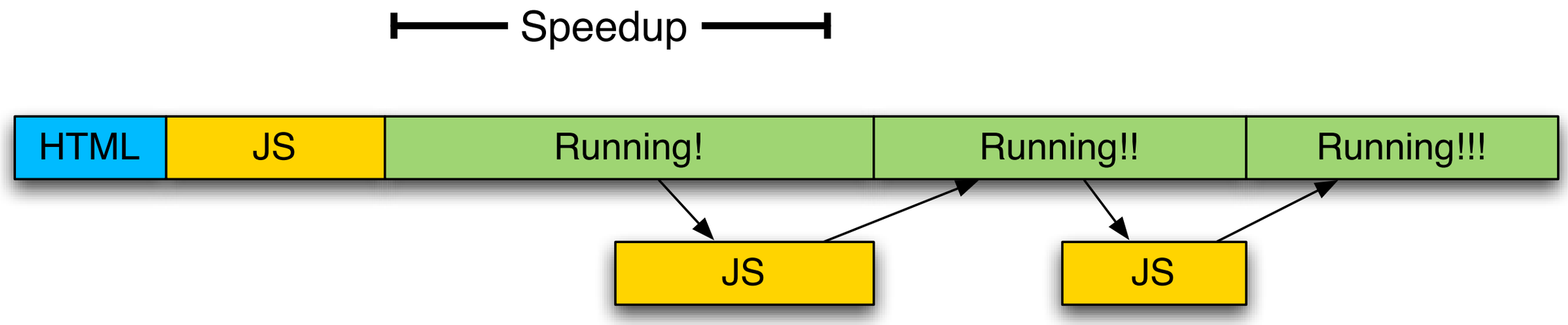
"Making GWT Better"

Why Code Splitting?



Without splitting

With splitting



Partially Loaded as the Normal State

- Accessing unloaded code might need a *delay*
- It can also *fail*

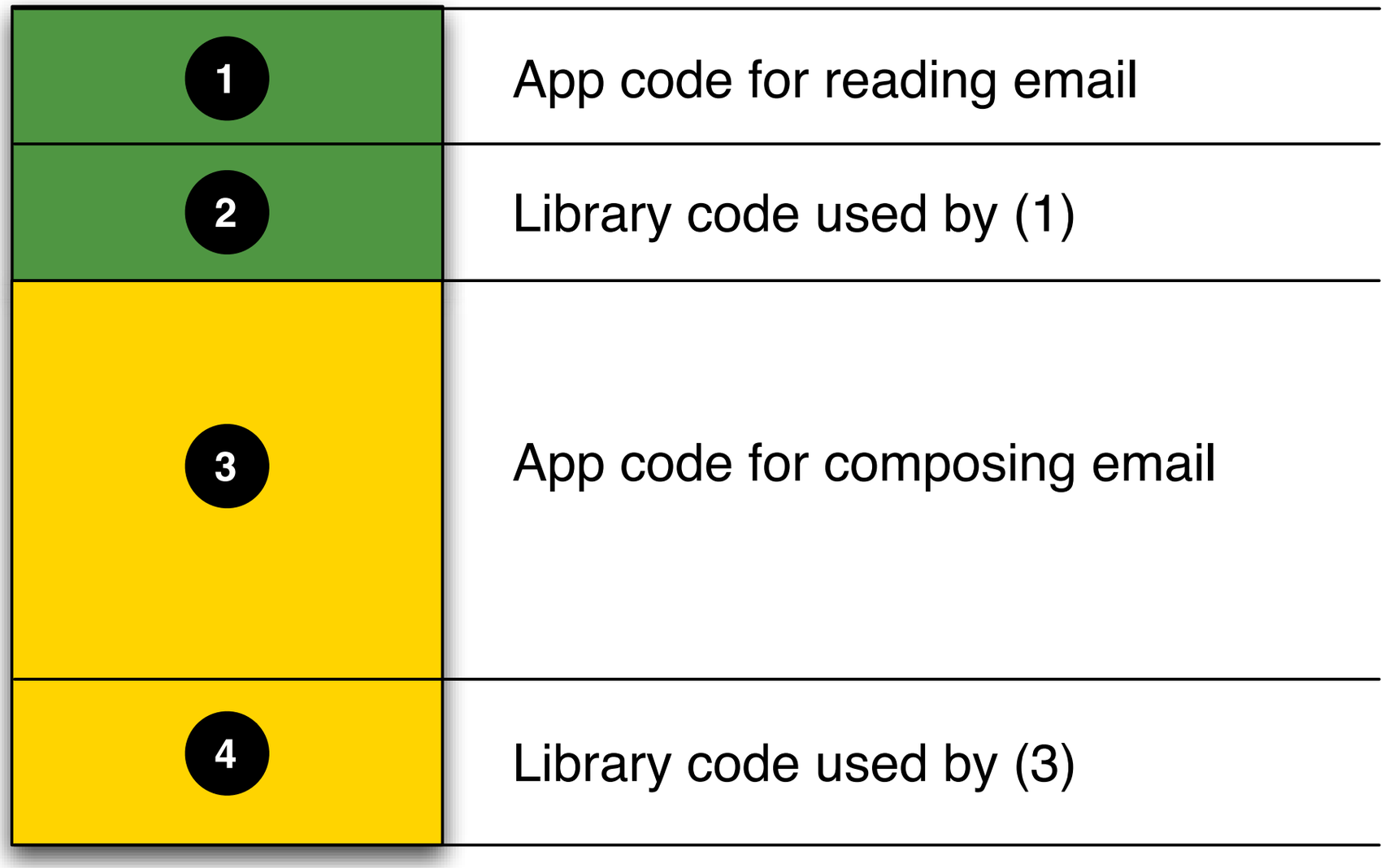
```
GWT.runAsync(new RunAsyncCallback() {
  public void onSuccess() {
    // runs when code loads
  }
  public void onFailure(Throwable reason) {
    // runs on failure
  }
}
```

Code Splitting Walkthrough

Simple Example: Composing Email

```
public static void onComposeEmailButtonClicked() {
  EmailCompositionView.show();
}
```

Not All Functionality is Created Equal

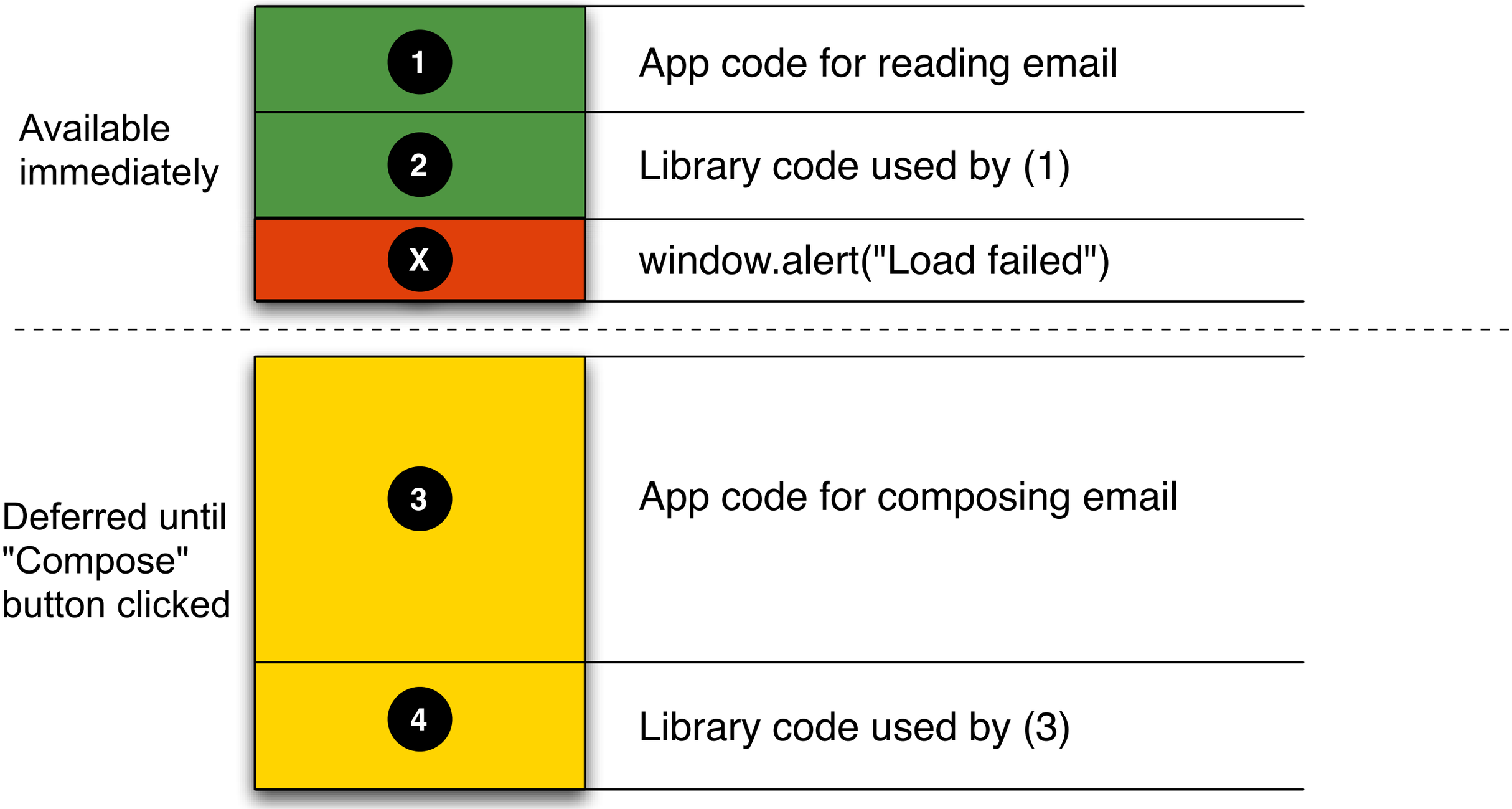


Compiled JavaScript

Splitting Out "Compose" With runAsync

```
public void onComposeEmailButtonClicked() {
  GWT.runAsync(new RunAsyncCallback() {
    public void onSuccess() {
      // This will run once the necessary fragment is loaded
      EmailCompositionView.show();
    }
    public void onFailure(Throwable reason) {
      // This will run if the necessary fragment cannot be loaded
      Window.alert("failed to load");
    }});
}
```

w00t!



A Week Later, un-w00t :-(

Recall splitting out "Compose" code like this

```
public void onComposeEmailButtonClicked() {
  GWT.runAsync(new runAsyncCallback() {
    public void onSuccess() {
      EmailCompositionView.show();
    }
    public void onFailure(Throwable e) {
      Window.alert("failed to load");
    }});
}
```

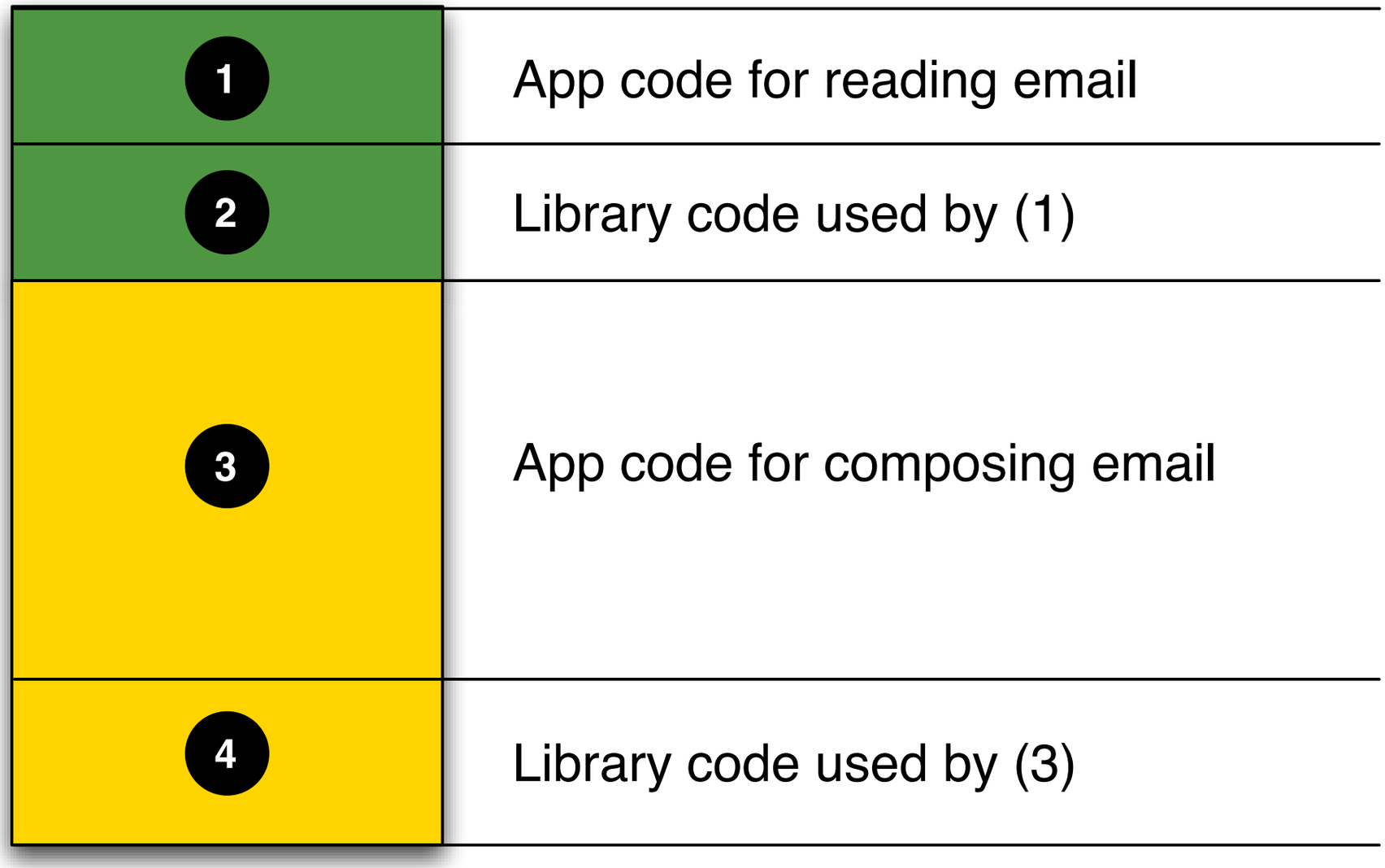
Split point...good!

But then the new guy on the team does this

Direct reference...bad!

```
// This is reachable directly
// from onModuleLoad()
public void onComposeKeystrokeClicked() {
  EmailCompositionView.show();
}
```

Back to Where We Started



Compiled JavaScript

The Story of Your Compile

Story of Your Compile (SOYC)

- What code downloads when?
- How big is each part?
- Why can't a particular method be deferrred?
- Let's take a look...

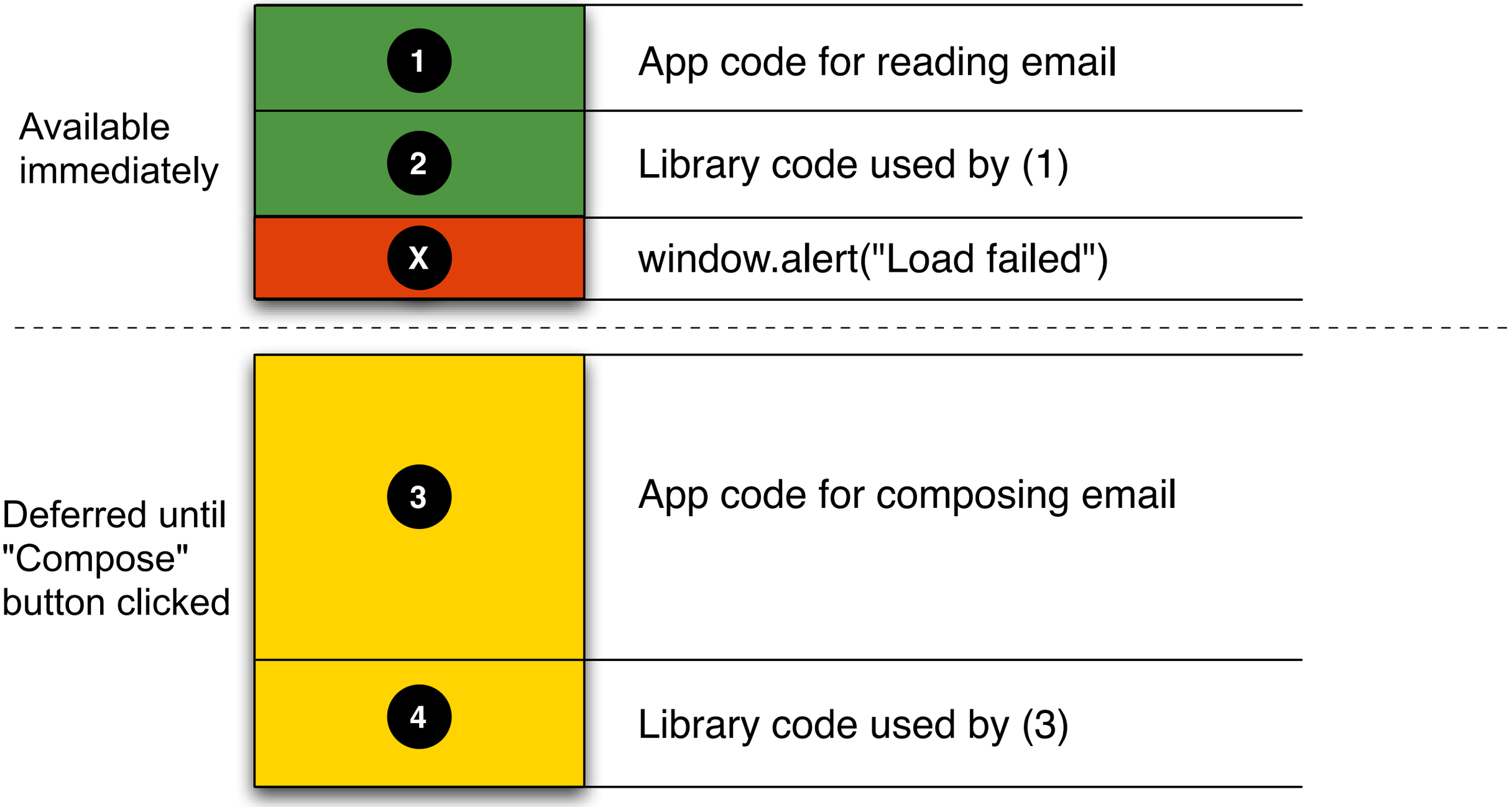
The Whack-a-Mole Solution

```
public void onComposeEmailButtonClicked() {
  GWT.runAsync(new runAsyncCallback() {
    public void onSuccess() {
      EmailCompositionView.show();
    }
    public void onFailure(Throwable e) {
      Window.alert("failed to load");
    }
  });
}
```

```
public void onComposeKeystrokeClicked(){
  GWT.runAsync(new runAsyncCallback() {
    public void onSuccess() {
     EmailCompositionView.show();
    }
    public void onFailure(Throwable e) {
     Window.alert("failed to load");
    }
  });
}
```

...repeat everywhere this recurs...

re-w00t! (+ Uneasiness About Maintainability)



The Async Package Pattern

Async Package Pattern

- Only the library developer needs to be careful, not the client
- Carefully isolate classes within a package
- Have exactly one "gateway class"
- Make its constructor private
- Remove static methods
- Instantiate the gateway class only within a runAsync call

Async Package Pattern: Gateway Class

```
public class EmailCompositionView {
  private EmailCompositionView() { }

  public interface Callback {
    void onCreated(EmailCompositionView view);
    void onCreateFailed();
  }

  // Only callable once a client gets an instance
  public show() {
    // Use lots of other code that has restricted visibility
    ...
  }

  public static void createAsync(final Callback callback) {
    GWT.runAsync(new RunAsyncCallback() {
      public void onSuccess() {
        callback.onCreated(new EmailCompositionView());
      }
      public void onFailure(Throwable e) {
        callback.onCreateFailed();
      }
    });
  }
}
```

Callers Can't Get it Wrong

```
public void onComposeKeystrokeClicked(){

  EmailCompositionView.createAsync(
    new EmailCompositionView.Callback() {
      public void onCreated(EmailCompositionView view) {
        // "show" is only callable once client has an instance
        view.show();
      }
      ...
    }
  );
  ...
```

Keep the Module Instance

```
public class MyModule {
  private final Expensive expensive;
  public MyModule(Expensive expensive) {
    this.expensive = expensive;
  }
  public void randomMethod() {
    expensive.doSomethingExpensive();
  }
}
```

Wrap-up

Summary of Suggestions

- Think about performance early, and track it as your app grows
- Use .rpc.log to understand size costs of RPC
- Identify less-used functionality and use runAsync() to split it out
- Use SOYC to understand code size and to debug splitting
- Find a splitting pattern to keep split points working reliably
- Think about what to do while fragments are being loaded and what to do if fragment loading fails
- In other words, put yourself in the end-user's shoes

Q & A

Google™ 09 I/O