

Google Inbox

Multi Platform Native Apps with GWT and J2ObjC

Ray Cromwell

Inbox



The future of email

Google

Email is old


The way we use email today is different from 10 years ago when Gmail launched



Gmail's creator, Paul Buchheit, at his desk at Google in 1999

Gmail is really powerful

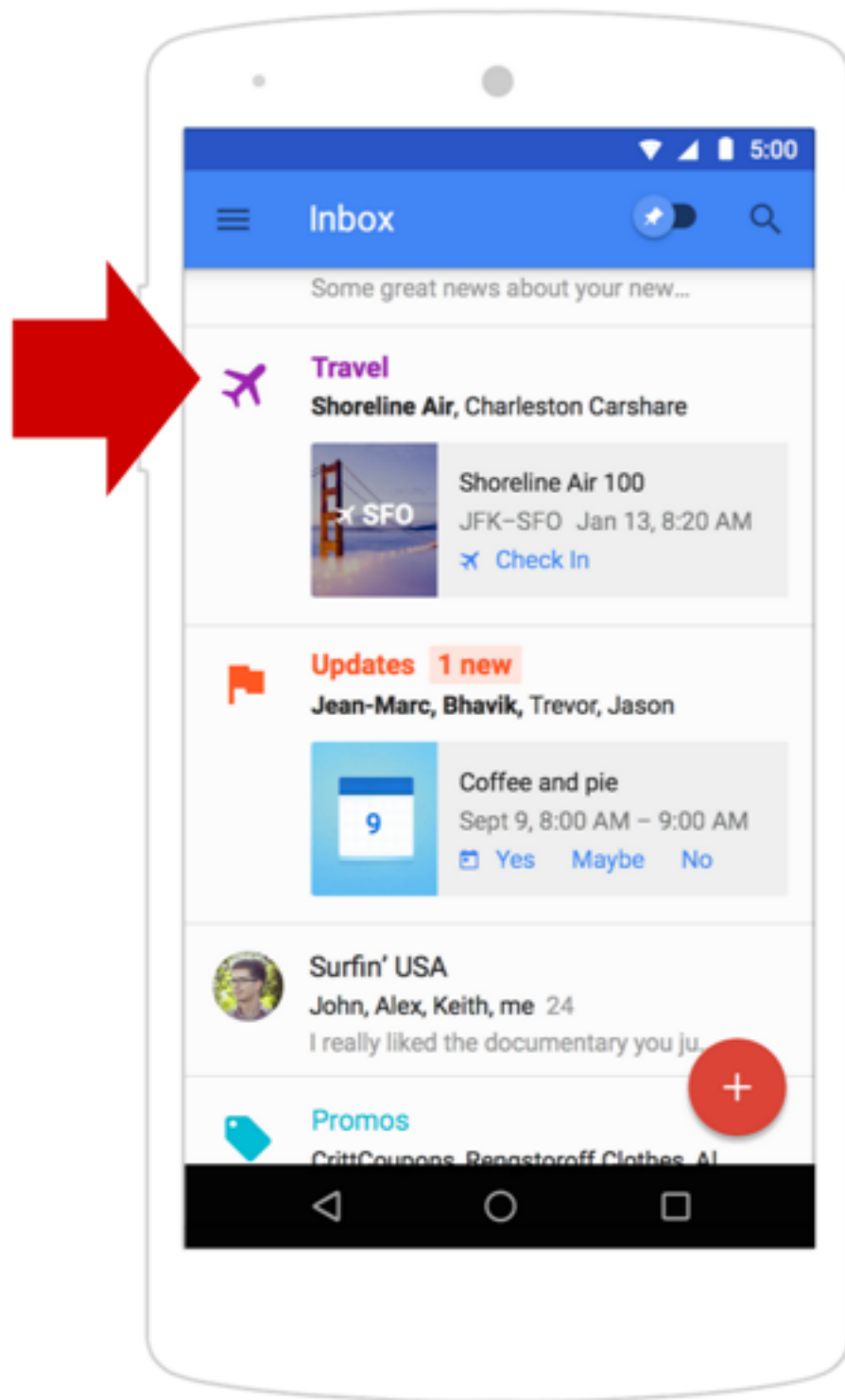
But many people don't know how to take advantage of the features



- Labels
- Filters
- Spam
- Signatures
- Labs
- Offline mode
- Tasks
- Chat
- Canned responses

Key findings

1. Email often works as a to-do list
2. Mobile usage is eclipsing desktop usage
3. Email feels like a lot of work

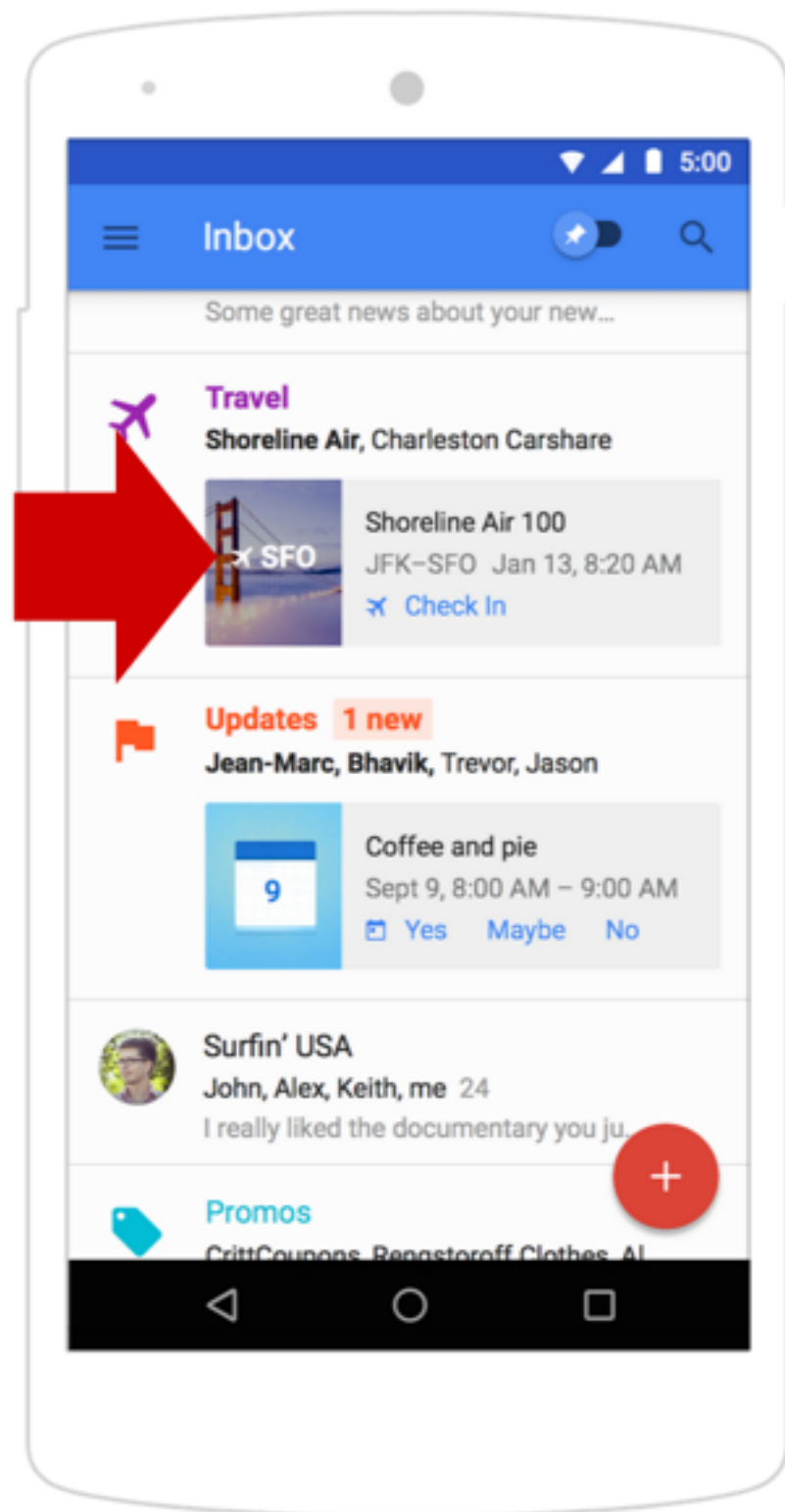


Bundles

Collapse all messages belonging to the same label as the one in your inbox

Use sweep to mark all unpinned items Done with a single click

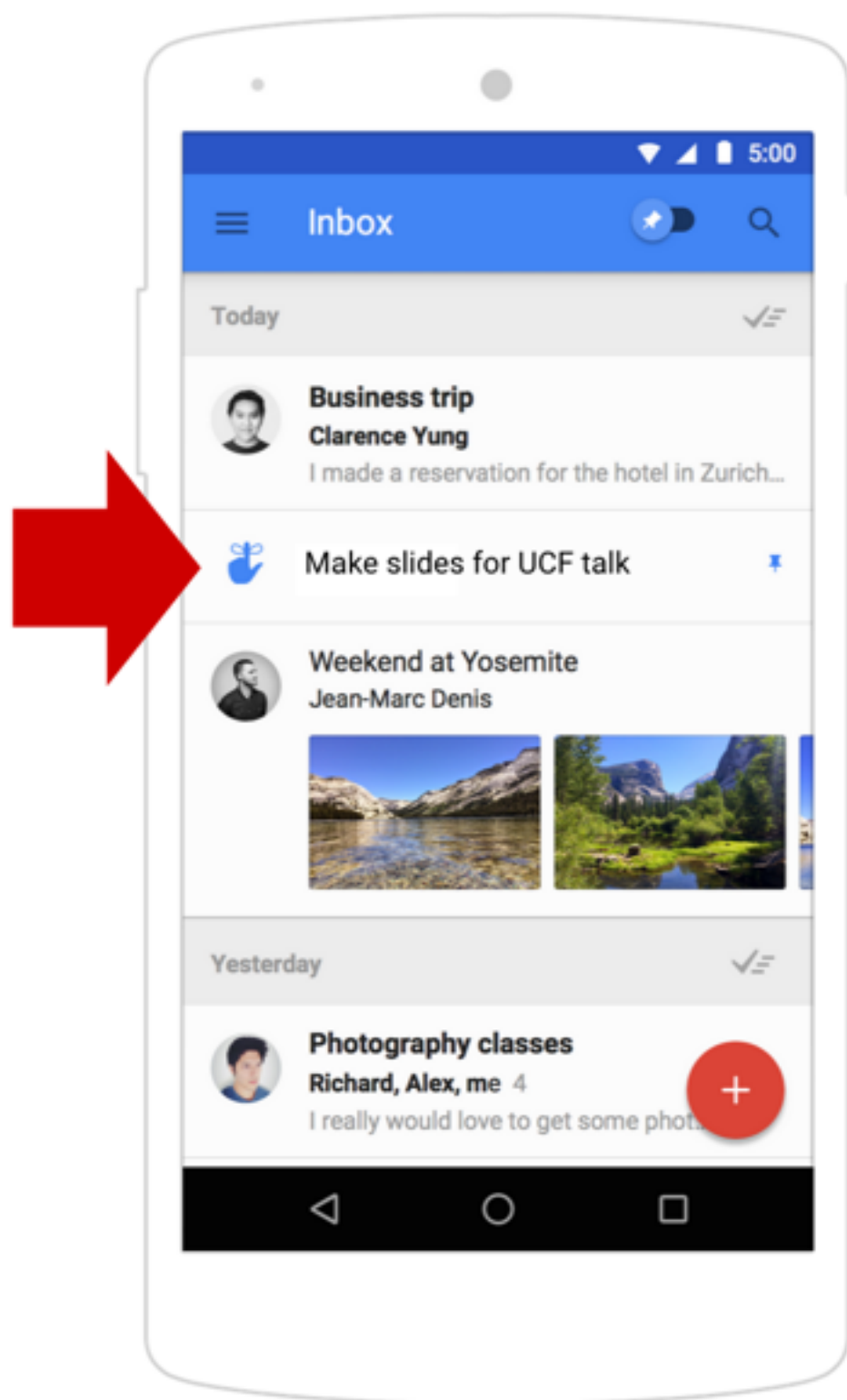
Throttle bundles to appear once a day or once a week



Highlights

See all of the important information at a glance without even opening your email

Real-time flight information and package tracking

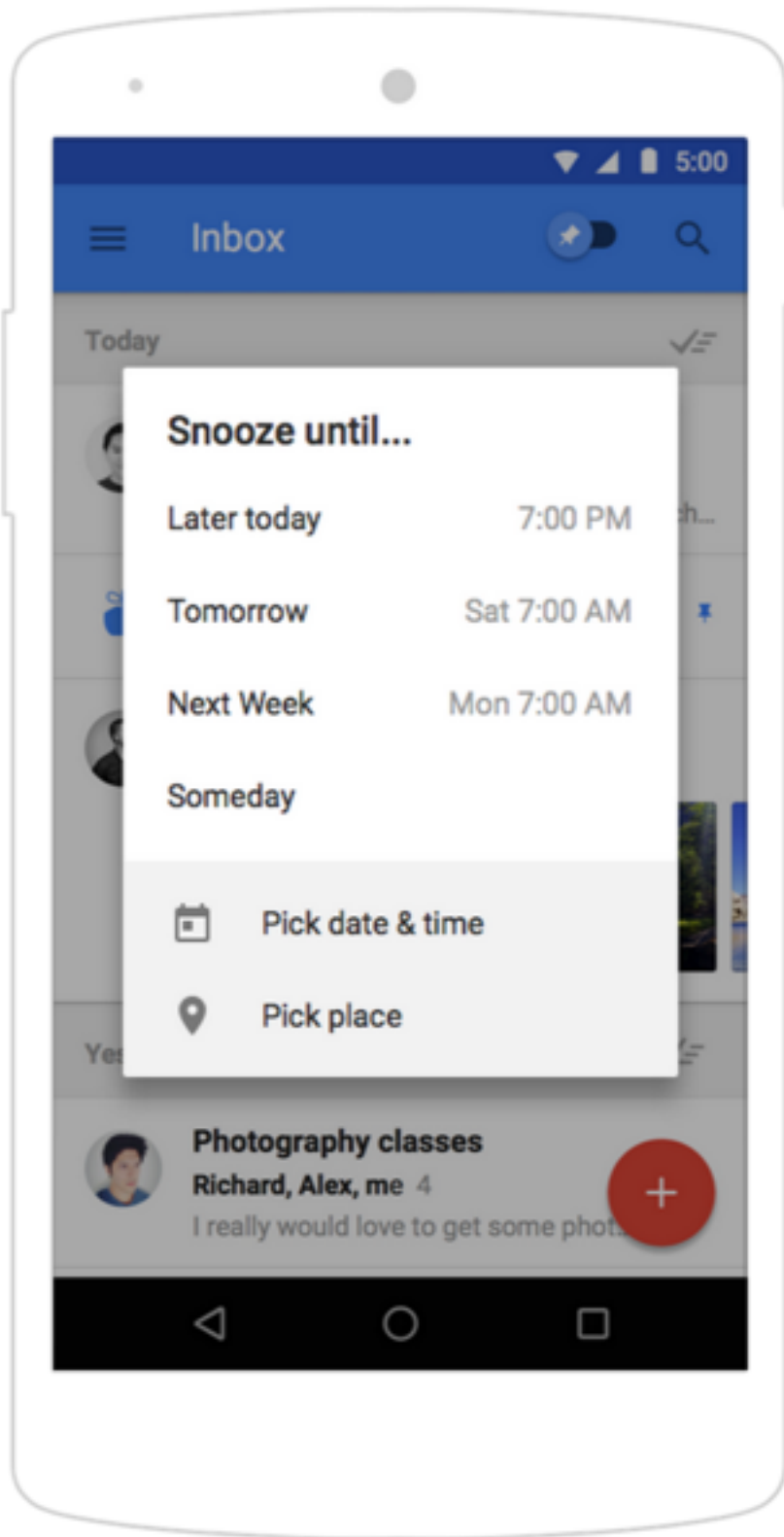


Reminders

All of your to-dos are in one place you'll remember to check (your email)

Assists to help you act on your reminders

Synced with Google Now



Snooze

Move things out of your way until you want to get back to them

Snooze to a time or place

How did we build
Inbox?

One App, Many Platforms



The Big Problem



**Web
JS**



**Android
Java**



**iOS
ObjC**

Technical Goals

- Lightning fast
- Feature parity across clients
- Quick prototyping and iteration
- Share code as much as possible

One Possible Solution

- Write the app 3 times from scratch
 - Once in Objective-C
 - Once in Java
 - Once in Javascript/HTML

Problems For You

- 3x the code
- 3x the tests
- 3x the bugs

Problems for Users

Why are features missing?!



Divergence

- Increased difficulty maintaining feature parity across platforms
- Example: Google Drive iOS SpreadSheets used to be read-only.

Solution #2?

- Write it all in one language
- Arrange to run that language everywhere

Web Everywhere?

- Mobile Web Performance Not There Yet
- Distribution in App Store needs wrapper
 - iOS doesn't let you leverage JIT
 - Performance on some apps not good

The Uncanny Valley

- HTML5 emulations of native OS UIs don't match up exactly
- Subtle differences annoy and confuse users
- Didn't we learn out lesson with Swing PLAFs?

C/C++ everywhere?

- Compile to NaCL/PNaCL for Chrome
- asm.js/pepper.js for Mozilla with Emscripten
- NDK/JNI on Android
- Link in C/C++ code wit ObjC using Xcode

Might work, but...

- C/C++ code not as easy to debug/test
- You may have more JS/Java engineers
- Unclear in large app if Web experience would be good

Write everything in Java then

- Run natively on Dalvik/ART
- Compile to Web with GWT
- Use Java->iOS compiler for iOS

Uncanny Valley Returns

- One UI API to render on multiple platforms still an 'impedance mismatch'
- Won't be able to leverage UI design toolchains native to each platform
- End users will still suffer

The Middle Ground

- Clearly we want to share some code
- Try to reduce the 3x penalty
- Best choice is non-UI dependent code
- But how?

Use the Cloud?

- Move non-essential business logic to cloud
- Clients share code via RPC

Pros

- Need to read/write to server anyway sometimes
- Enhanced security for some apps
- For some platforms, server is faster

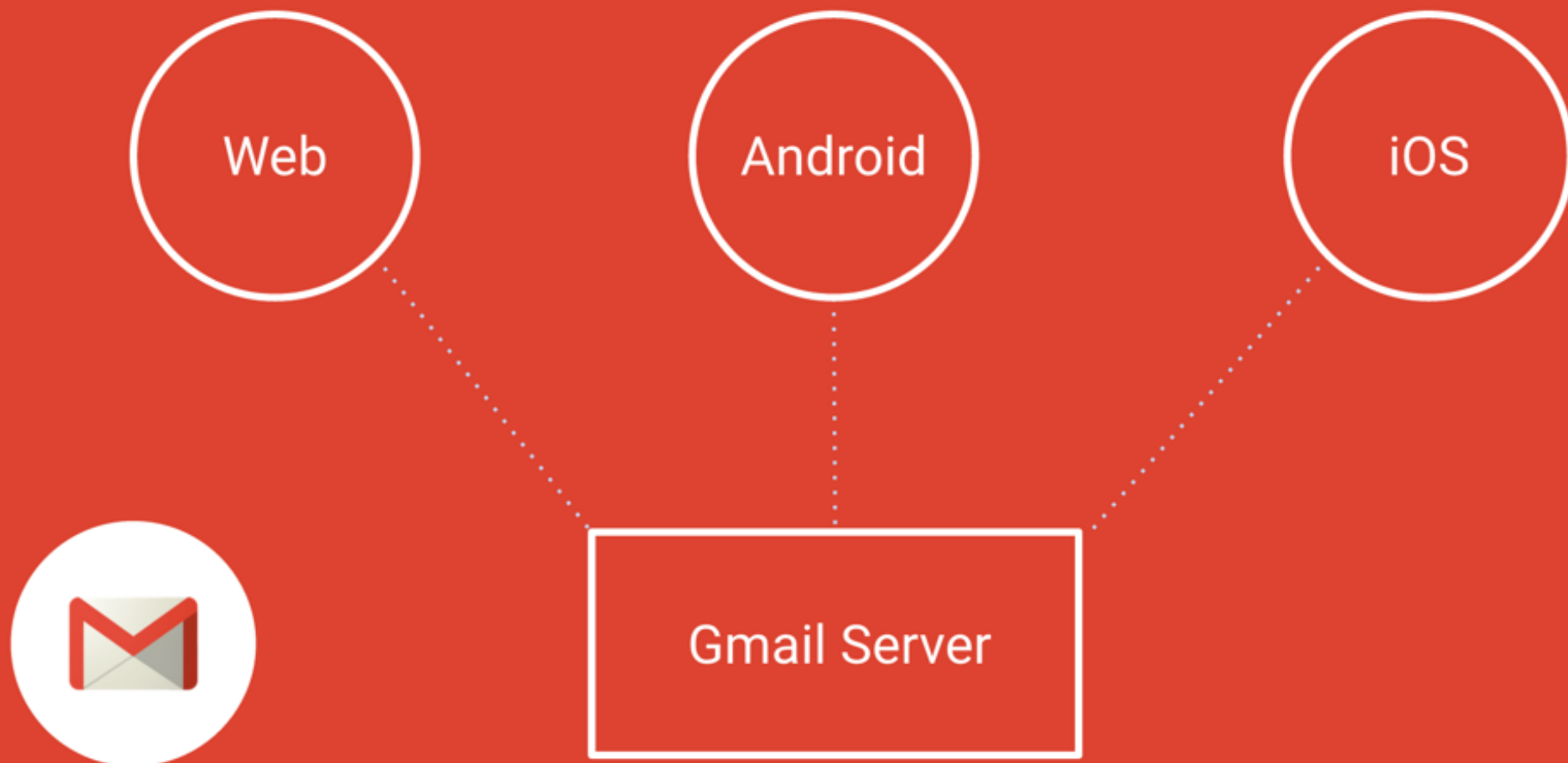
Cons

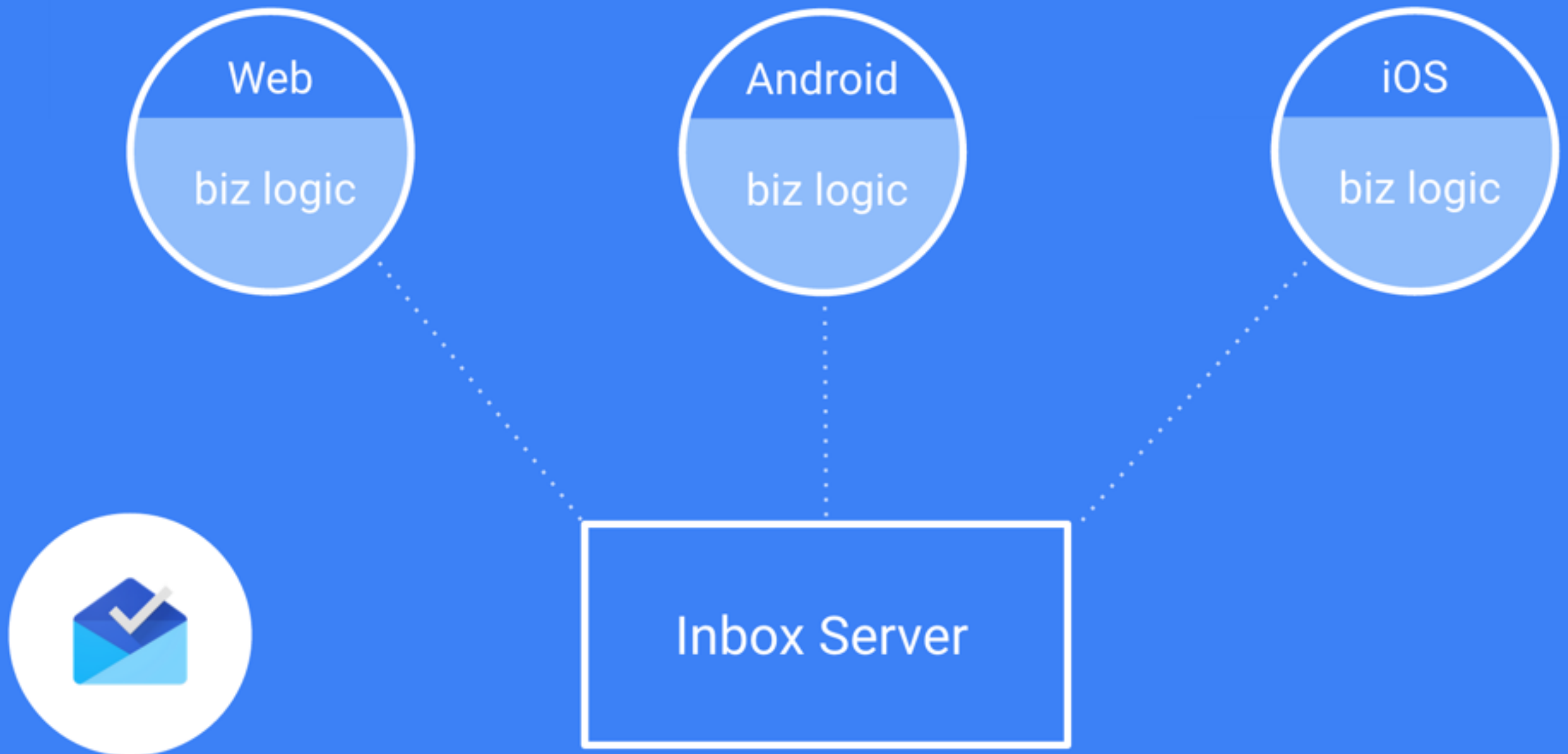
- Higher latency
- Doesn't work offline
- Not optimal customer experience

Hybrid Model

- UI Frontend
 - Use native UI/toolchain
 - Language with 'impedance match' to platform
- Shared Client Code
 - Written in Java

60-70% of code (with tests)
shared





How does this work?

Make JS Ninjas Happy

- GWT compilation of Java code to JS
- JsInterop annotations mark which classes visible to JS
- Specialized linker generates Closure JsDoc header
- Closure Compiler type-checks and optimizes JS and transpiled JS together

Benefits

- JS developers see hard errors
- Unused code from Java may be pruned
- Permits Single JS binary

Deeper Integrations

- `Java.util.logging` integrated with Closure Logging — single log window
- Java exception handling merges with Closure error reporting — unified deobfuscated stack traces
- SourceMap merging allows simultaneous debugging of JS and Java

Deeper Integration 2

- Closure Compiler 'FileHandler' + SuperDevMode
- Edit-and-Refresh of Java code

Item.java

```
package com.google.inbox.items;
@JsType
public interface Item {

    public Item makeItem(
        ItemType type);
    void pin();
    void markDone();
    void snooze();
    ...
}
```

package-info.java

```
@JsNamespace("inbox.api");
package com.google.inbox.items;
```

ItemHandler.js

```
inbox.ItemHandler.prototype.onPin = function(item) {
    item.pin();
};

inbox.ItemHandler.prototype.createItem = function(
    type) {
    return inbox.api.makeItem(type);
};
```

The Objective-C Process

- Translate Java to Objective-C via j2objc
- Import generated .h/.m files into Xcode project
- <hack hack hack>
- Prophet!

Not So Fast: Memory!

- j2obj supports Ref Counting, GC, and ARC
- Realistically, GC not an option on iOS
- RefCounting and ARC handled automatically

What about Cycles?

- Supports `@Weak` and `@WeakOuter` annotations
 - Mark a field weakly reference, or outer class weakly referenced
- CycleFinder - Analyzes Java code and reports potential cycles

Example

```
package com.sfeir.calc;  
  
public interface CalcEngine {  
    int evaluate(String expr);  
}
```

Example Usage

```
#import <Foundation/Foundation.h>
#include "com/sfeir/calc/CalcEngine.h"
#include "com/sfeir/calc/CalcEngineImpl.h"

int main (int argc, const char * argv[]) {

    @autoreleasepool {

        id<ComSfeirCalcCalcEngine> calcEngine = [[ComSfeirCalcCalcEngineImpl alloc] init];

        printf ("insert expression...\n");

        char str[50] = {0};

        scanf("%s", str);

        NSString *expr = [NSString stringWithUTF8String:str];

        int result = [calcEngine evaluateWithNSString:expr];

        printf("%s = %i\n", str, result);
    }

    return 0;
}
```

Taken from jhybrid

Android

- It's all Java folks.
- But sometimes you have too many methods
 - Use Proguard to chop out unneeded reps

Other Gotchas

- ProtoBufs
 - Cross compiling Java impls isn't optimal
 - Instead, use protoc compiler to generate one for each platform
 - And then wrap the native representation with a Java facade

Debugging

- Good old printf always works
- SourceMaps in Chrome permit source-level Java debugging
 - And server-side stack trace deobfuscation
- Android just works
- Xcode works

General Use Cases

- Hard to write/rewrite compute-intensive libraries
 - Operational Transforms
 - Formula/Statistical Calculation Engines
 - Syncing layers
 - Cryptography Libraries, Codecs, et al

Summary

- Many Ways to handle multiple platforms
- For a certain class of apps, Hybrid apps work well
- Google has built several of these and appear to be getting high ROI on productivity

World Peace

- Lives no longer need be lost in language wars
- Can't JS Ninjas, Java Experts, and Objective-C hipsters get along?

Thanks

- <https://github.com/Sfeir/jhybrid> (proof of concept example)