
Google™
11
I/O

Building Game Development Tools with App Engine, GWT, and WebGL

Lilli Thompson
5/10/2011

FEEDBACK: Please provide feedback on this session at http://goo.gl/lj56w

HASHTAGS: #io2011, #DevTools

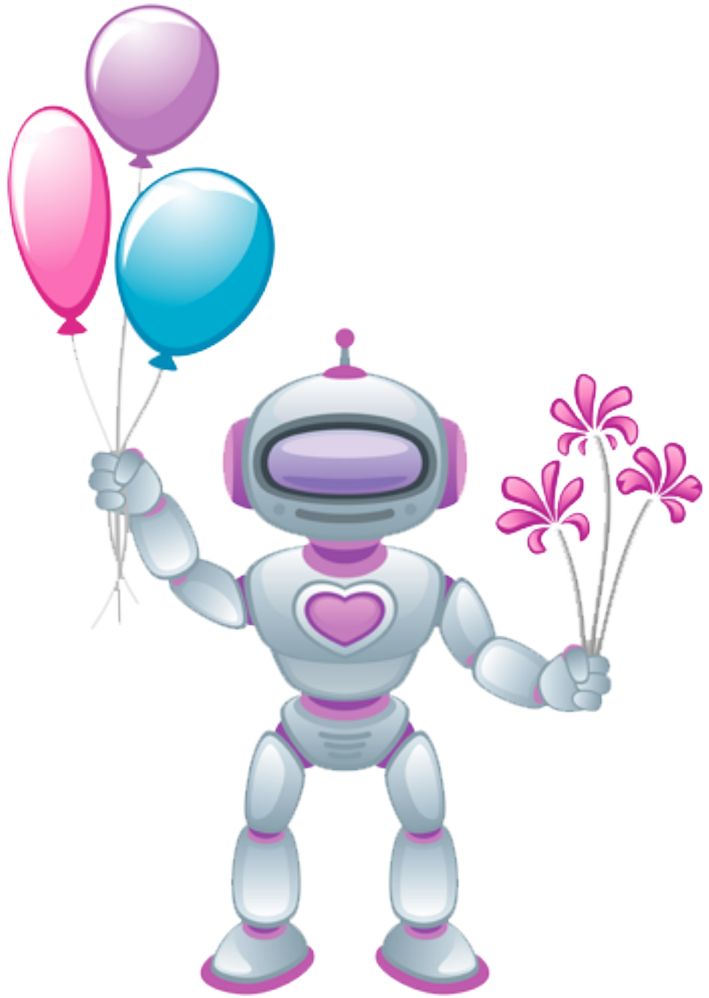
Overview

- Who is this talk for?
- What's a "Game Development Tool"?
- Why use App Engine / GWT / WebGL?
- The story of my game tools project

Who is this talk for?

Nice to meet you!

- Who am I?
- Who are you?



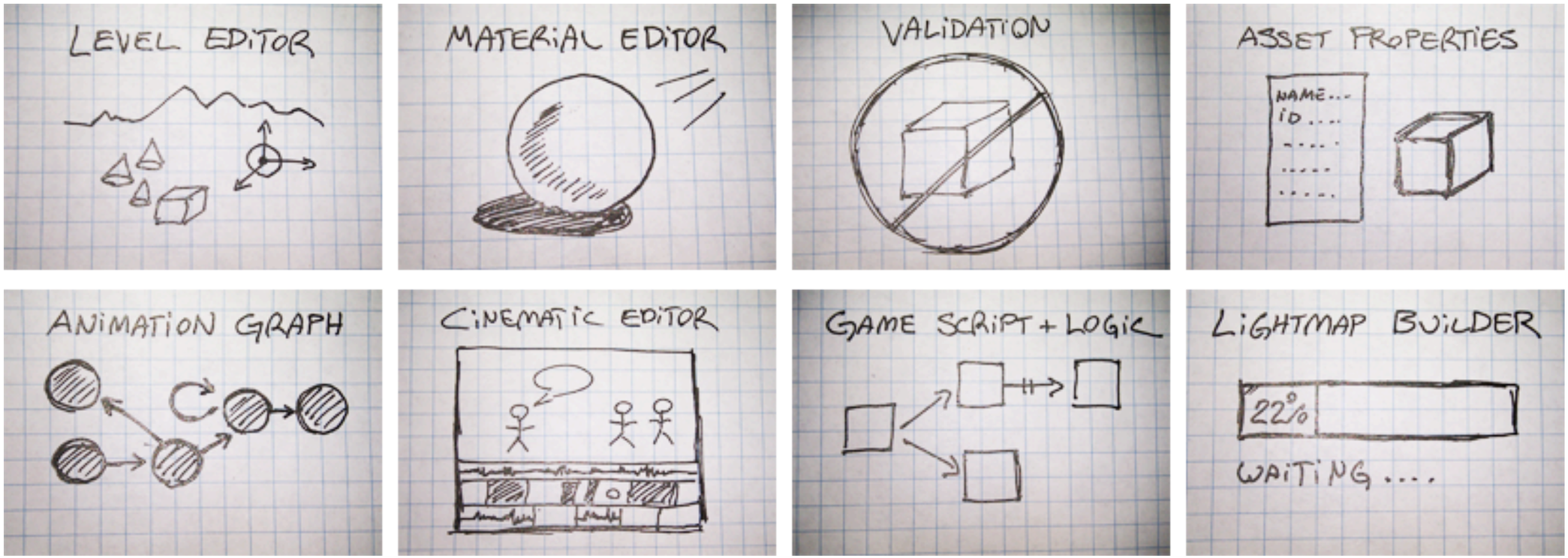
What's a "Game Development Tool"?

Game assets are Complicated!

- Demanding
- Binary
- Custom Visualization
- Interconnected

You’re going to need a lot of tools

So many custom data formats, so little time...



What’s special about tools programming?

- Process, not product
- Fast iteration time
- Art / tools / engine glue always evolving
- Complicated content = complicated bugs

“But my game is small and my data is simple! Why should I care?”

Joe Game Developer

You should care.

- Web and mobile games may be small now, but...
- Content expectations rising rapidly
- Learn from the console game industry's mistakes
- Familiarize dev team with new technologies

Why GWT, App Engine, and WebGL?

Considerations for a tools platform

UI Options

Dev Skills

Setup Ease

Debugging Tools

Why App Engine?

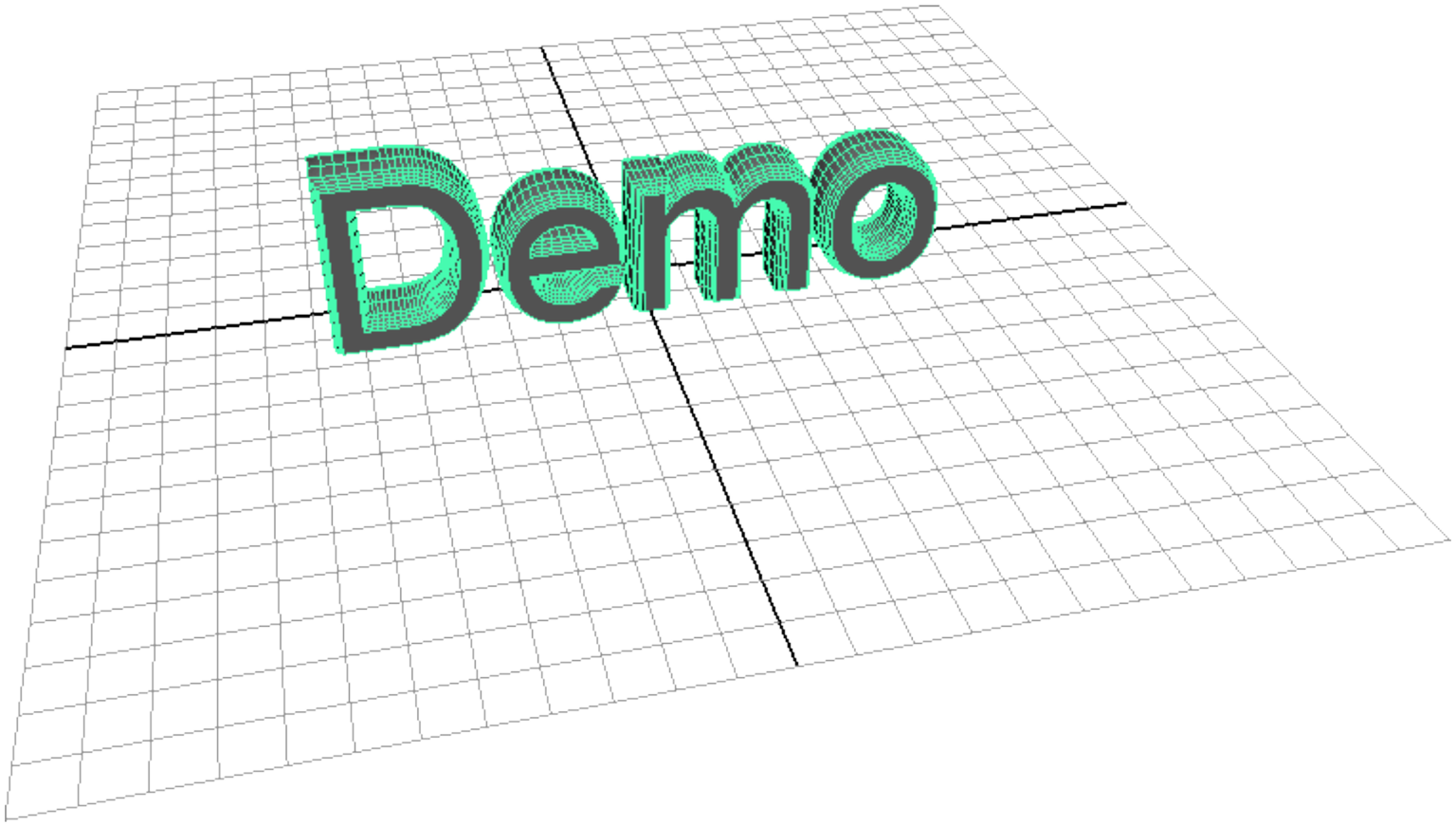
- Build, host, manage cloud apps
- Collaboration
- Cross-platform, easy setup
- No hardware management
- Easy data storage
- Scalable
- Logging, versioning, longevity

Why GWT?

- Framework for browser apps
- Compile Java to JS
- Easier handling of browser quirks
- Scalable coding environment
- Debugging functionality
- UI widgets
- Open source

Why WebGL?

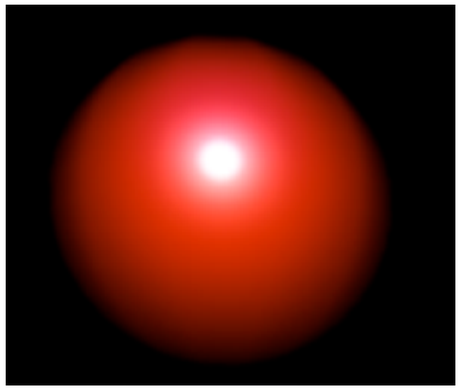
- Render 3D directly in browser
- No plugins, easy setup
- Debugging tools
- Familiar to OpenGL devs
- Fast



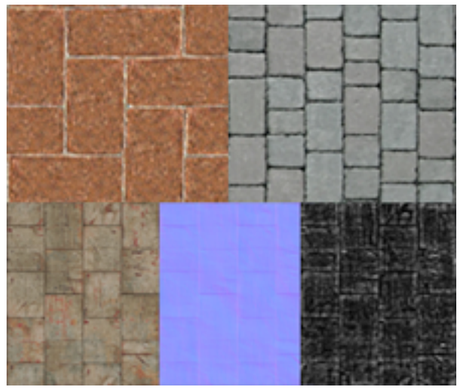
The Story of My Project

The Building Blocks

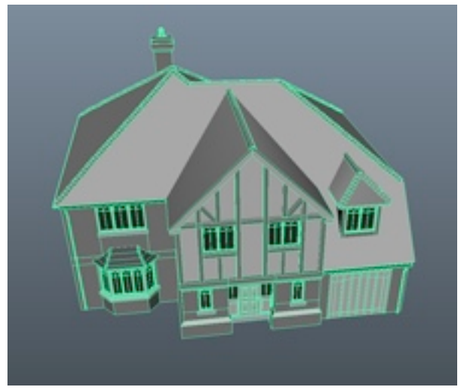
Shaders



Textures



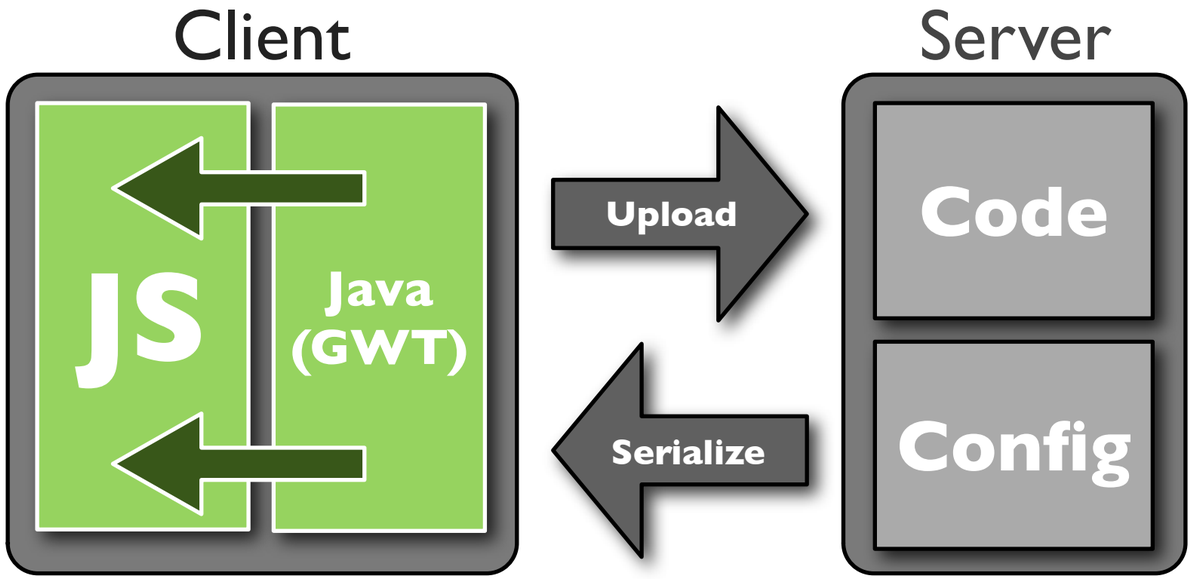
Meshes



How do I use both GWT and JS APIs?

GWT JSNI

- Looks like a comment
- Call from Java into Javascript
- Pass data to WebGL or any JS API

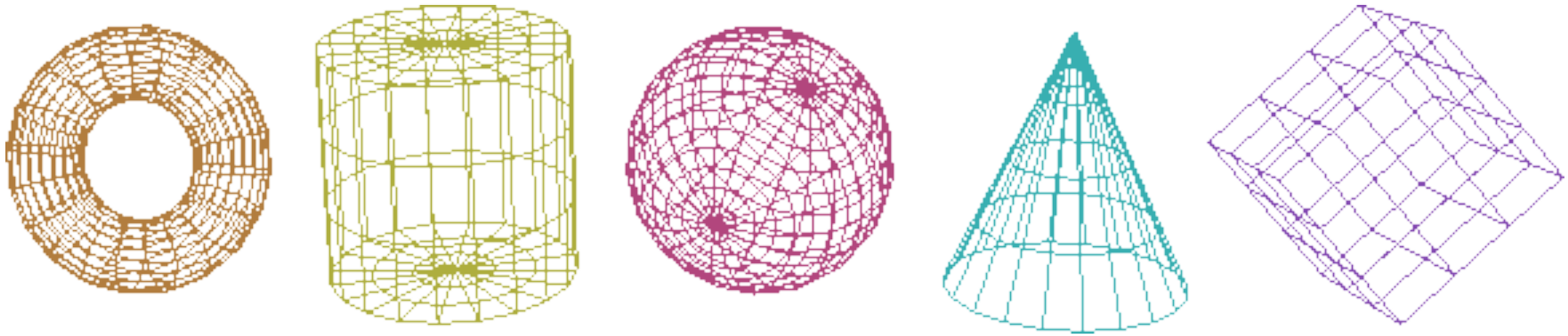



```
// A very simple JSNI Method
private native void helloJSNI(String name) /*-{
    alert("Hello " + name + "!");
}-*/;
```

Where do I put my data?

Static Data

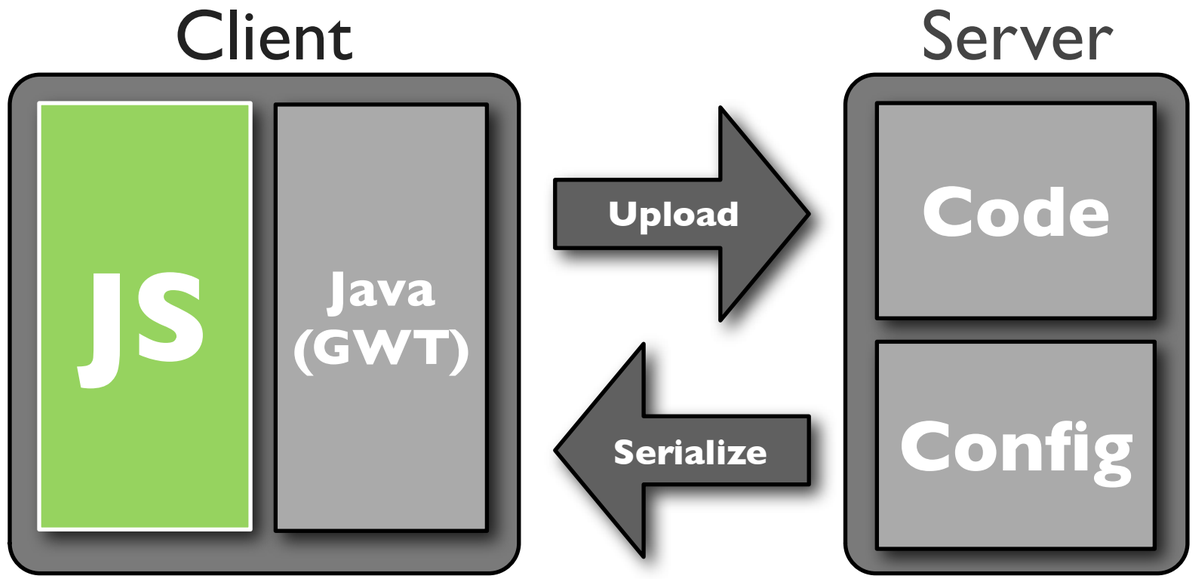
- Easiest and most familiar
- Individual assets or bundles
- Deployed with your app
- Served Statically



How do I handle static resource bundles?

The HTML5 FileSystem API

- Manipulate and expand data locally
- Chrome 9+, with flags
  - `unlimited-quota-for-files`
  - `allow-file-access-from-files`
- Sandboxed file system
  - Reading: File, Blob, FileReader, FileList
  - Writing: BlobBuilder, FileWriter, FileSaver
  - DirectoryReader, FileEntry, LocalFileSystem



File System API Sample Code: Initialization

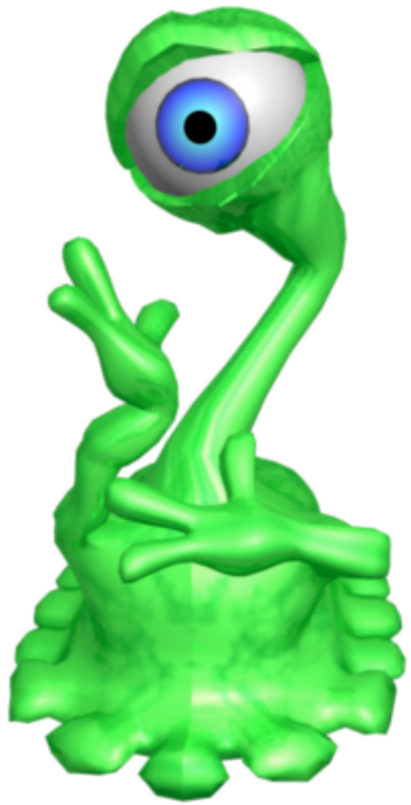
```
window.requestFileSystem(
  window.PERSISTENT, 5*1024*1024 /*5MB*/, onInitFs,
  errorHandler);

function onInitFs(fs) {
  console.log('Opened file system: ' + fs.name);

  // See if the resources have been cached locally.
  // If not, go get them.
  checkResourcesUpToDate();
}
```

File System API Sample Code: Writing

```
function writeFile(fs, fileName, contents) {
  fs.root.getFile(fileName, {create: true, exclusive: true},
    function(fileEntry) {
      fileEntry.createWriter(function(fileWriter) {

        fileWriter.onwriteend = function(e) {
          console.log('Write completed.');
        };

        fileWriter.onerror = function(e) {
          console.log('Write failed.');
        };
        var builder = new BlobBuilder();
        builder.append(contents);
        fileWriter.write(bb.getBlob('text/plain'));
      }, errorHandler);
    }, errorHandler);
}
```

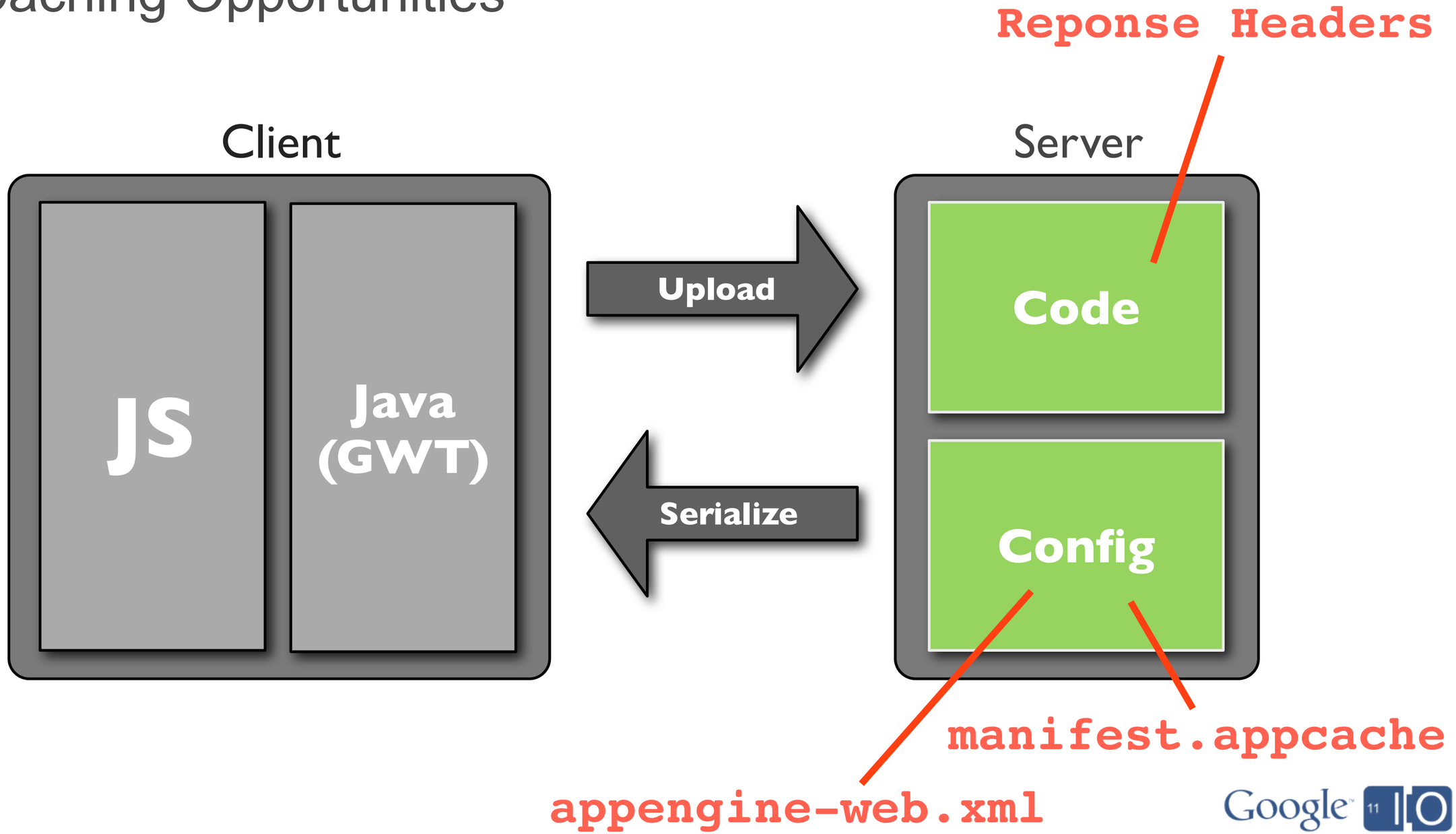
File System API Sample Code: Reading

```
function readFile(fs, fileName) {
  fs.root.getFile(fileName, {},
    function(fileEntry) {
      fileEntry.file(function(file) {

        var reader = new FileReader();

        reader.onloadend = function(e) {
          console.log('file contents: ' + this.result);
        };

        reader.readAsText(file);
      }, errorHandler);
    }, errorHandler);
}
```



How do I make it load even faster?

Many Caching Opportunities



Resource Caching

- Static Resources
  - App Engine Config sets browser cache expiration
- HTTP Responses
  - Max-age
  - Cache-control: public
- Content fingerprinting

appengine-web.xml

```
<static-files>
    <include path="**.nocache.*" expiration="0s" />
    <include path="/**.obj" expiration="7d 6h" />
</static-files>
```

HTML5 Application Cache

- Available offline, fast
- Easy to use
- 5MB limit
- Configured with a manifest file
- Need custom MIME type

index.html

```
<html manifest="example.appcache">
    <head>...</head>
    <body>...</body>
</html>
```

Configuring the MIME Type

web.xml

```
<mime-mapping>
    <extension>appcache</extension>
    <mime-type>text/cache-manifest</mime-type>
</mime-mapping>
```

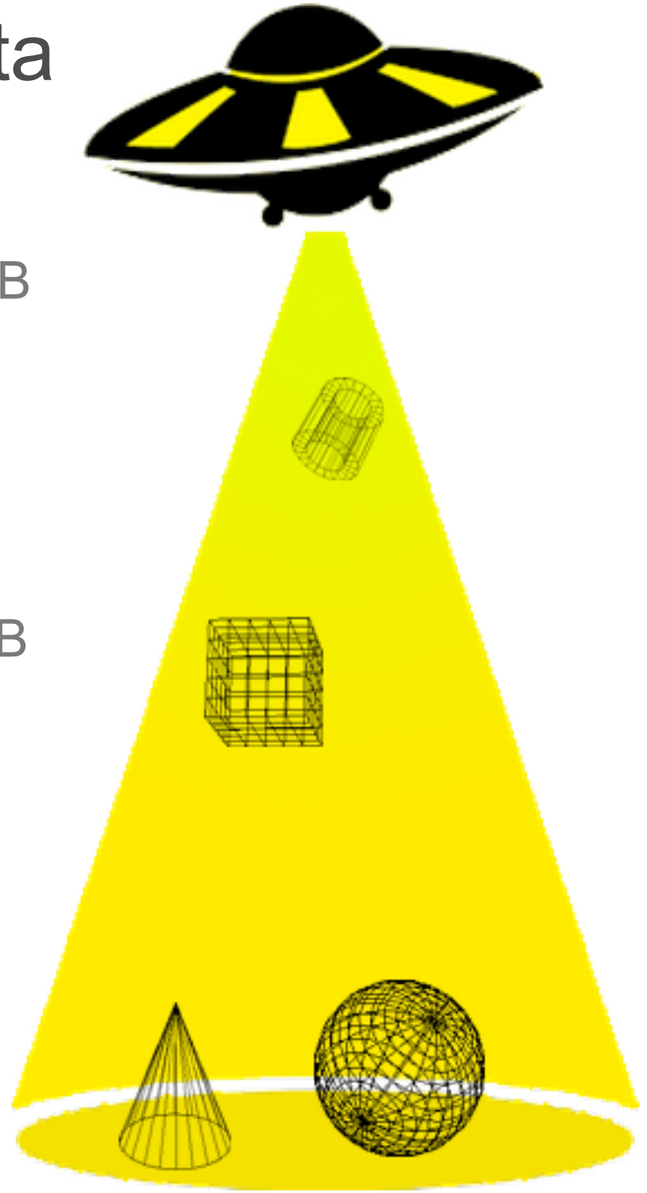
Application Cache Manifest Sample

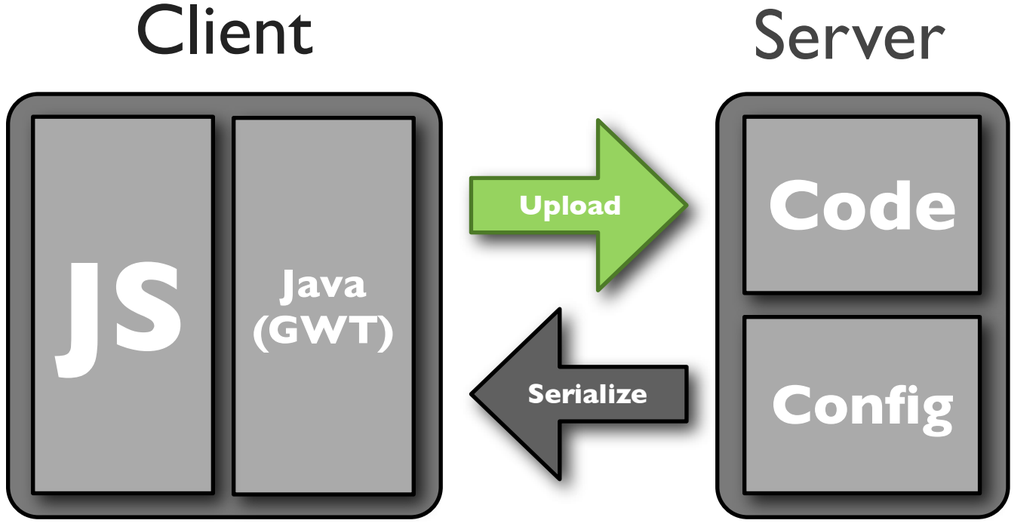
```
CACHE MANIFEST
# 2011-05-12:v1

# Explicitly cached after first download
CACHE:
/favicon.ico
myPage.html
images/logo.png

# Resources that require the user to be online.
NETWORK:
/myServlet

# offline.jpg will be served if the app is offline
FALLBACK:
images/* images/offline.jpg
```

What about dynamic data?

Uploading Data

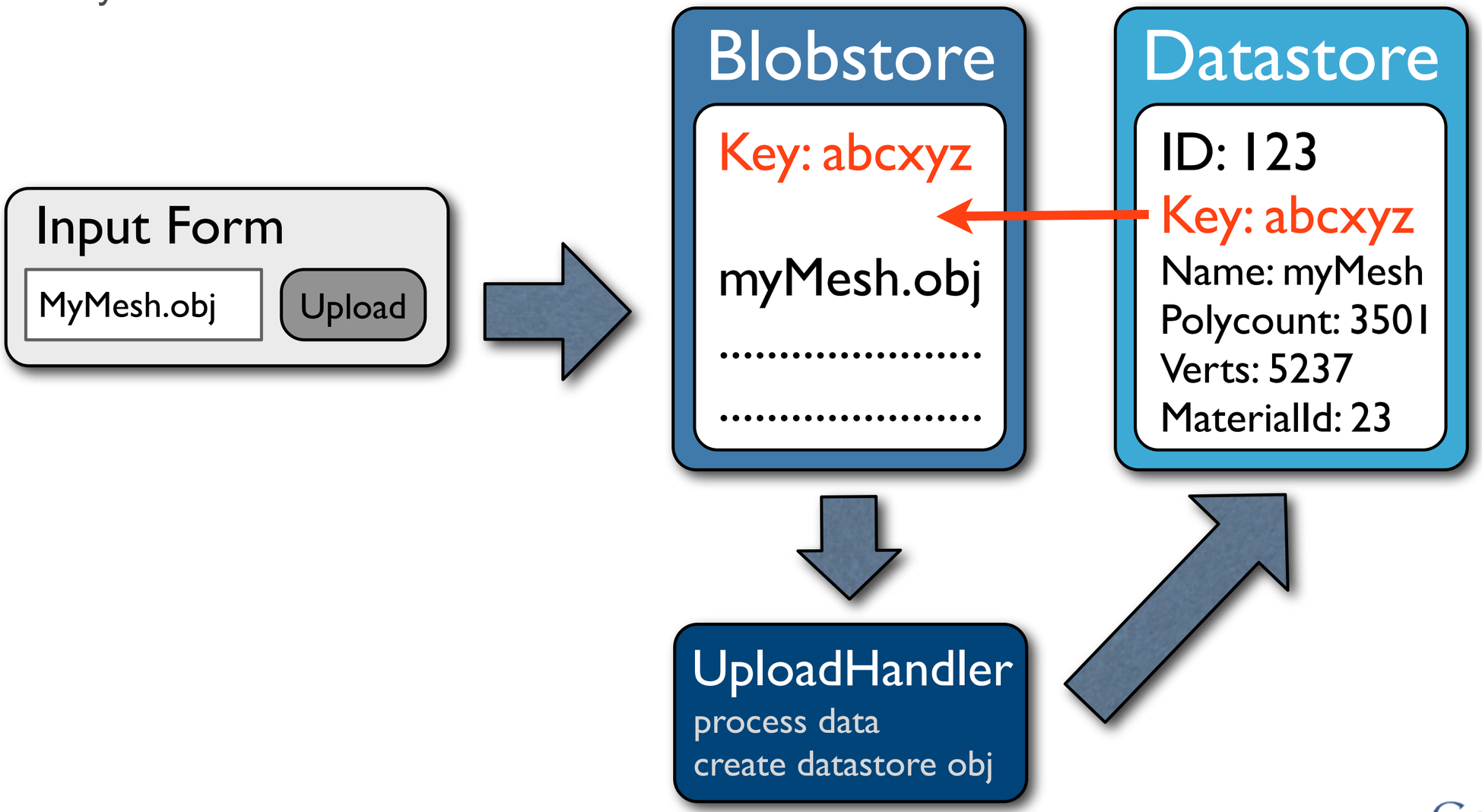
- Datastore
  - Entities up to 1MB
  - Easy to use
  - Objectify, JDO
- Blobstore
  - Entities up to 2GB
  - 1MB per API call
  - Opaque





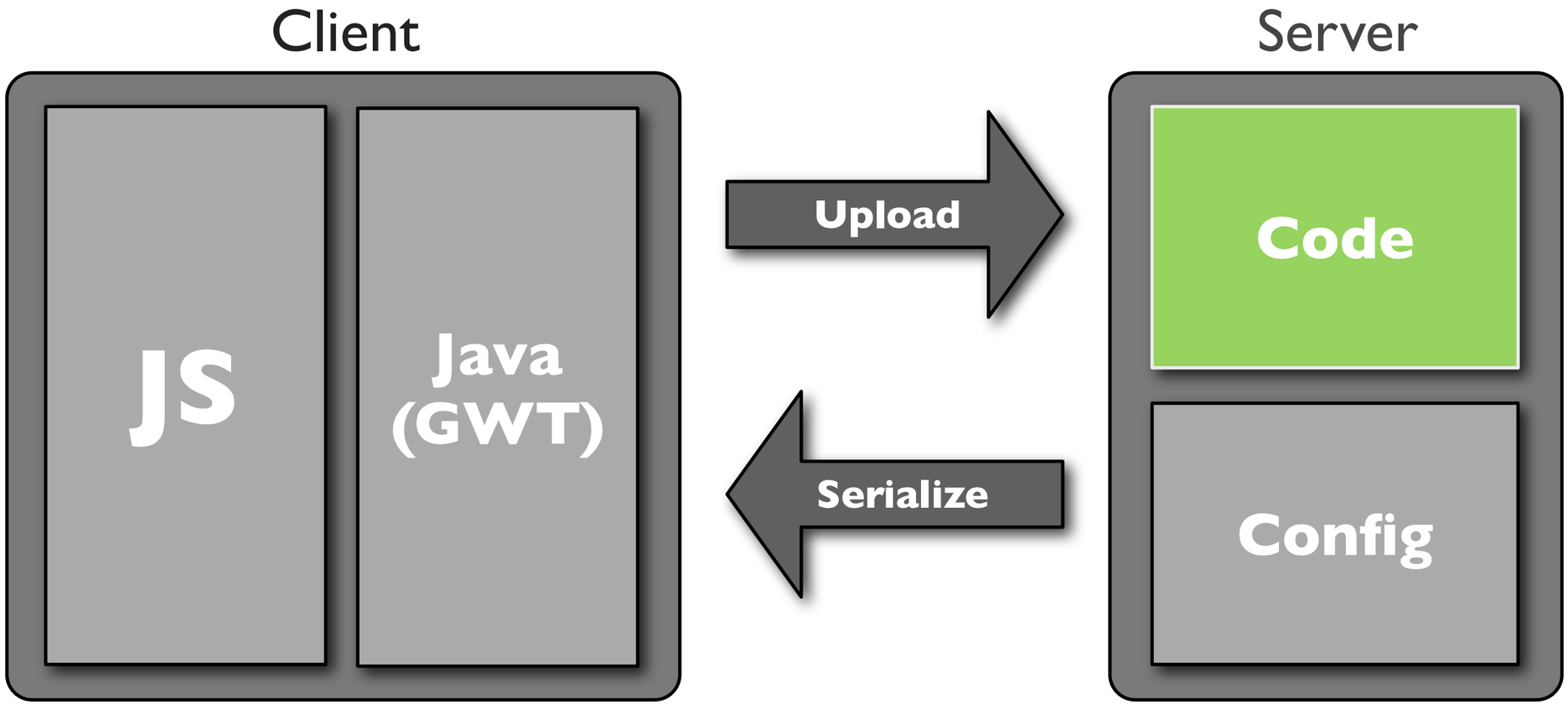
Datastore and Blobstore

You'll probably use both



Writing to Blobstore Programatically

- File API to create blobs
- New in 1.4.3

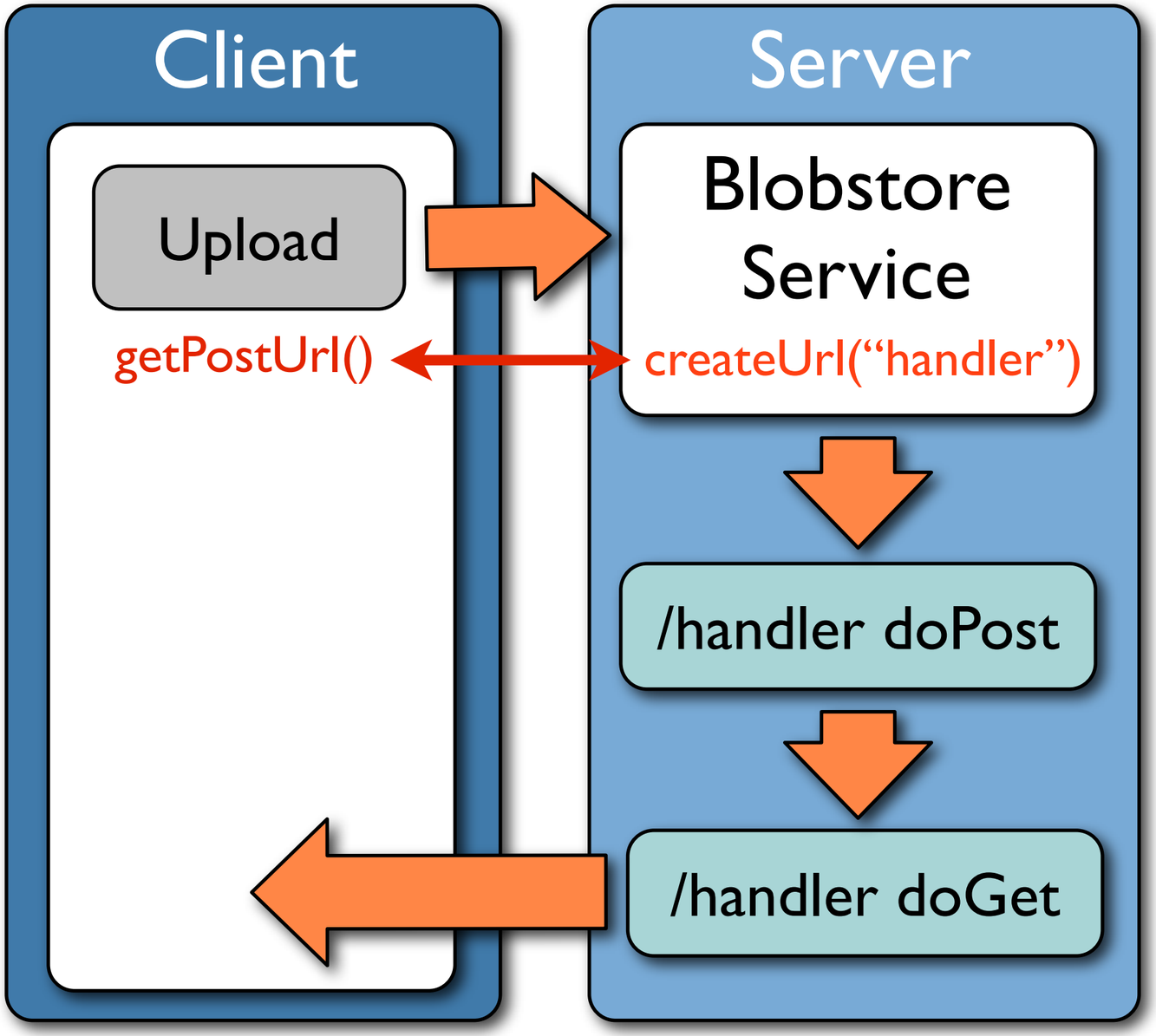


Blobstore Write API Sample Code

```
// Create the data for the blob to write.
byte[] byteArray = createAssetData(fileName);

FileService service = FileServiceFactory.getFileService();

AppEngineFile file =
    service.createNewBlobFile("application/octet-stream");

FileWriteChannel writeChannel =
    service.openWriteChannel(file, true);

writeChannel.write(ByteBuffer.wrap(byteArray));
writeChannel.closeFinally();

// Get the blobKey of the newly created blob.
String key = service.getBlobKey(file).getKeyString();
```

Uploading to Blobstore

- Request upload URL
- Form posts to URL
- Handler called with new keys
- Returns headers-only redirect
- Second handler writes response



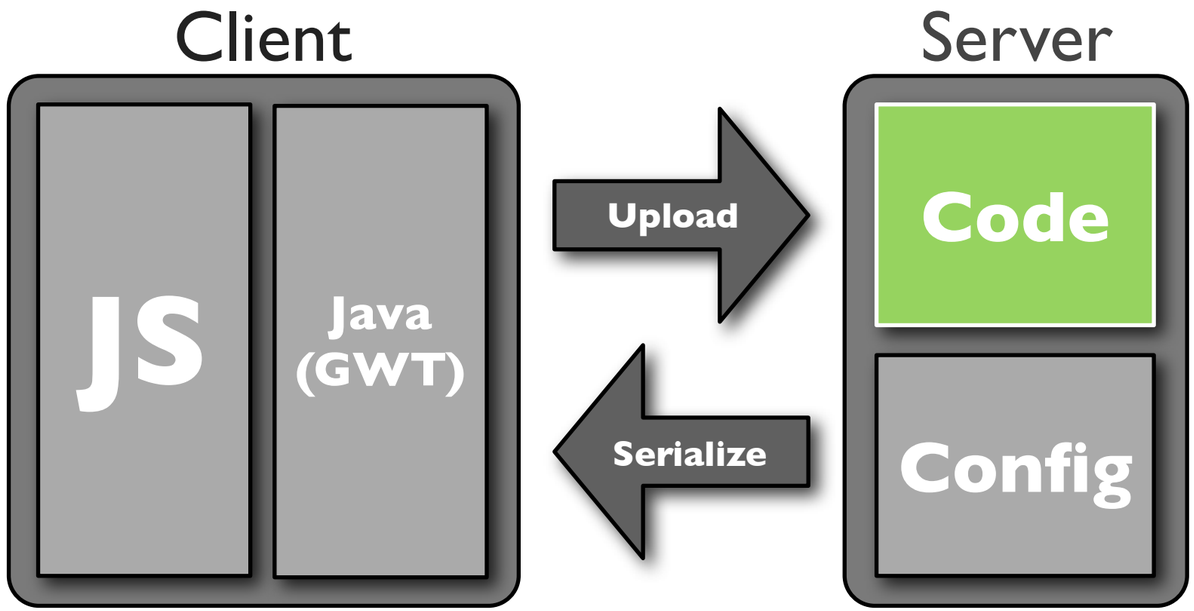
Blobstore Upload Sample Code

```
// Upload form posts to the URL created by this.
blobstoreService.createUploadUrl("/blobUploadHandler");

void doPost(HttpRequest req, HttpResponse resp) {
   Map<String, BlobKey> uploadedBlobs = blobstoreService.getUploadedBlobs(req);

   // Create a datastore object to wrap the blobkey.
   DatastoreObj newAsset = createDatastoreObj(blobKey);

   resp.sendRedirect("/blobUploadHandler?id=" +
       newAsset.getId());
}

void doGet(HttpRequest req, HttpResponse resp) {
   String id = req.getParameter("id");

   // Write the id of the datastore obj to the client.
   resp.getWriter().print(newAsset.getId());
}
```

What if I want to process my data?

Several Options

- ImageService for 2D operations
- Task queue for parallel processing
- App Engine Backends for demanding work



2D Processing with ImageService

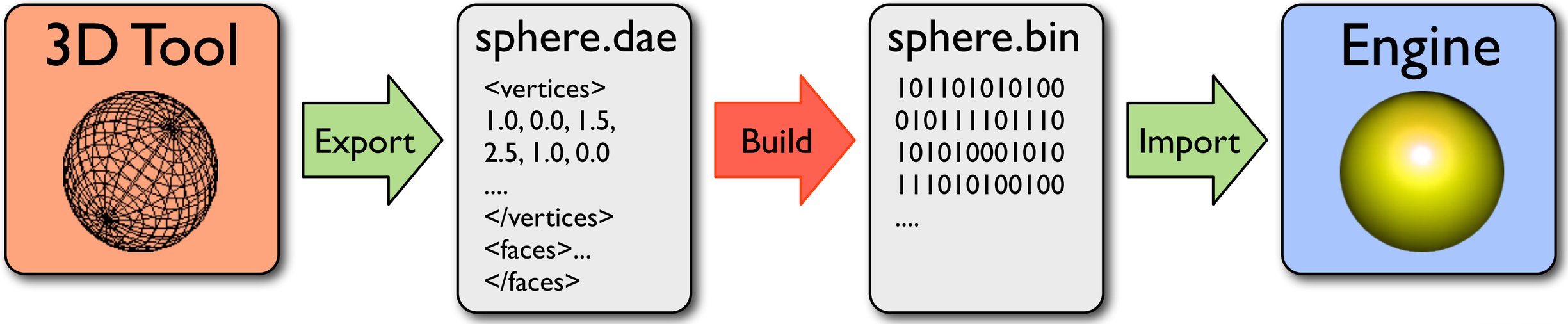
- Lightweight image processing
- Data must be in Blobstore
- High performance, but URL takes time to create
- Use for
  - Thumbnails
  - Cropping

ImageService Sample Code

```
ImagesService service =
    ImagesServiceFactory.getImagesService();

// getServingUrl can take several hundered ms.
String thumbnailUrl = service.getServingUrl(blobKey);

// Scale the image down and crop to 100x100.
thumbnailUrl = thumbnailUrl + "=s100-c";
```

The Task Queue

- Parallel background processing
- Looks like an HTTP request
- 10 minute Timeout
- 100/s



Configuring a task queue

queue.xml

```
<queue-entries>
    <queue>
        <name>processAssets</name>
        <rate>100/s</rate>
    </queue>
</queue-entries>
```

Task Queue Sample Code

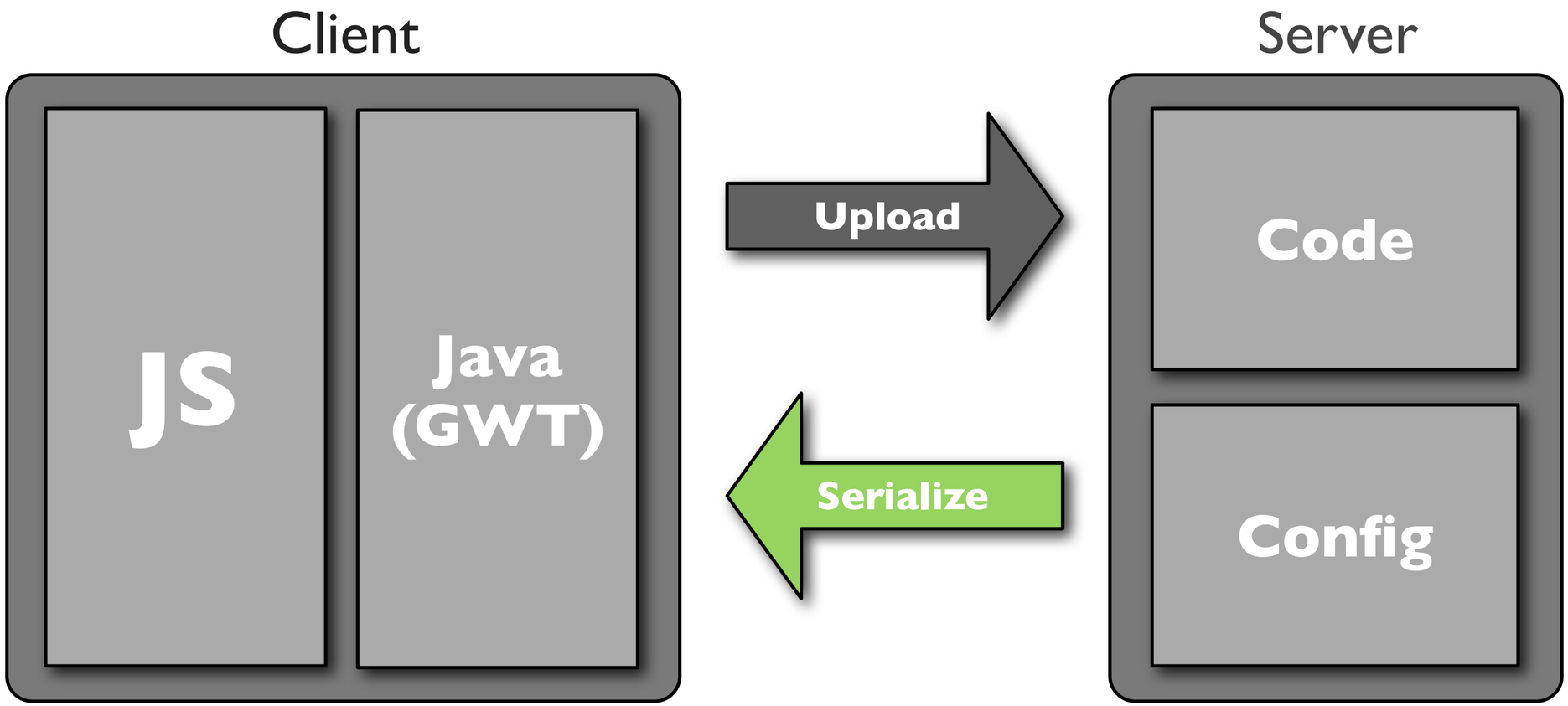
```
Queue queue =
    QueueFactory.getQueue("processAssets");

for (String fileName : assetFileNames) {
   queue.add(TaskOptions.Builder
      .withUrl("/processAssetData")
      .method(Method.GET)
      .param("fileName", fileName));
}

// Get method of /processAssetData
void doGet(HttpRequest req, HttpResponse resp) {
   String fileName = req.getParameter("fileName");

   // Process the specified asset into a new blob.
   Long id = processFileAndCreateBlob(fileName);
}
```

App Engine Backends

- Standard App Engine
  - 30s request timeout
  - Instances startup and shutdown dynamically
  - Heavyweight processing may be difficult
- App Engine Backends
  - No request deadlines
  - Increased memory usage allowance
  - Addressable instances
  - Configurable to remain in memory

What format is most efficient for serialization?

Data Formats and Serialization



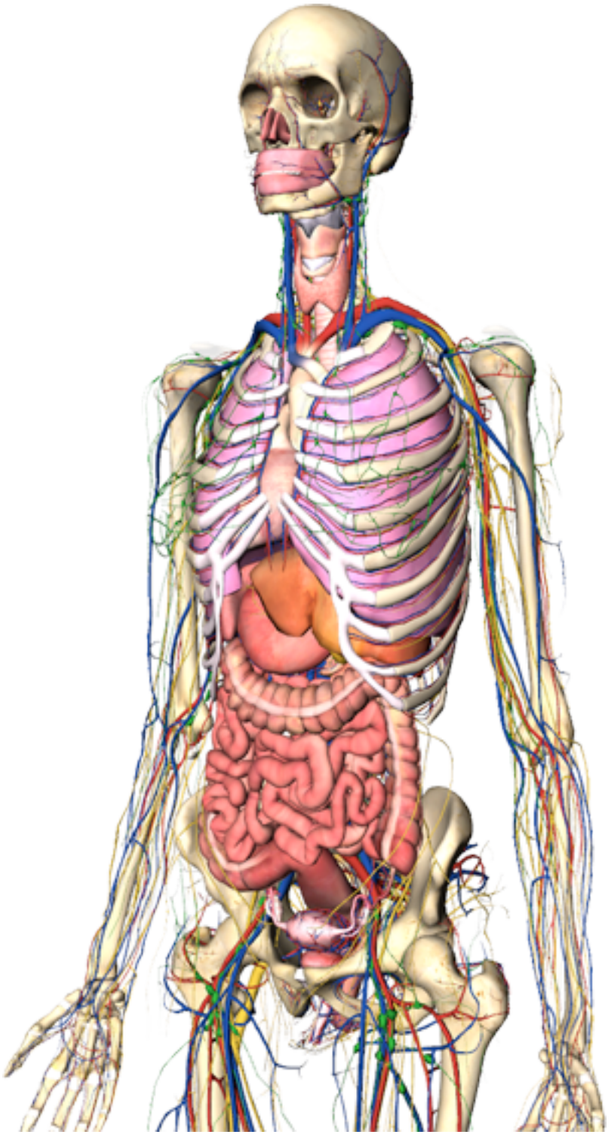
Binary Data

- XMLHttpRequest Level 2
- ArrayBuffer
  - Defined in the TypedArray spec
  - Generic, fixed-length binary data buffer
  - Manipulate with typed array or ArrayBufferView
- Blob
  - Defined in the File API spec
  - Raw data not necessarily in JS-native format

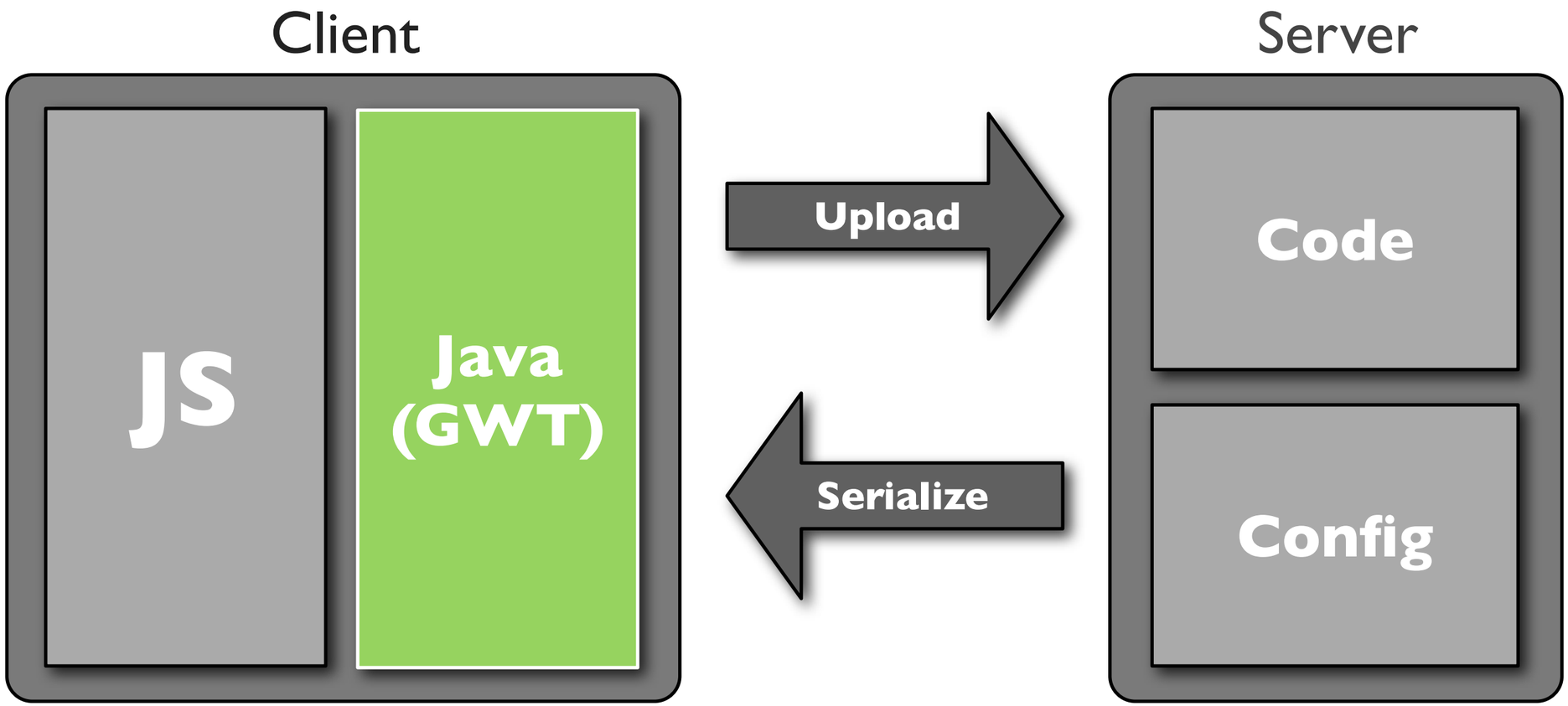
Efficient UTF-8 Serialization

- From the Google Body team
  - 1.4M triangles
  - 8 attribute vertices
- Use delta compression on geometry
- Send HTTP using GZip
- Set UTF-8 variable length encoding
  - Unpacked as a string
  - Sent back as responseText from XMLHttpRequest
  - Use charCodeAt(i) to extract data

- **Compression ratio over binary: ~3.5:1!**



How do I request data from GWT?



New GWT binding to use XHR Level 2

```
public class BinaryXMLHttpRequest extends XMLHttpRequest{

    public native JavaScriptObject getResponse() /*-{
        return this.response;
    }-*/;

      // Set to "arraybuffer" or "blob" for binary response.
    public native void setResponseType(String value) /*-{
        this.responseType = value;
    }-*/;

  …
}
```

Using the new GWT Binding

```
BinaryHttpRequest xhr = BinaryHttpRequest.create();
xhr.open("GET", "/mesh?id=" + id);
xhr.setResponseType("arraybuffer");

xhr.setOnReadyStateChange(new ReadyStateChangeHandler() {
    public void onReadyStateChange(BinaryHttpRequest xhr) {
        if (xhr.getReadyState() == XMLHttpRequest.DONE) {
            xhr.clearOnReadyStateChange();
            loadData(xhr.getResponse()));
        }
    }
});
xhr.send();

// Pass data to WebGL through JSNI.
private native void loadData(JavaScriptObject data) /*-{
    $wnd.loadMeshData(data);
}-*/;
```

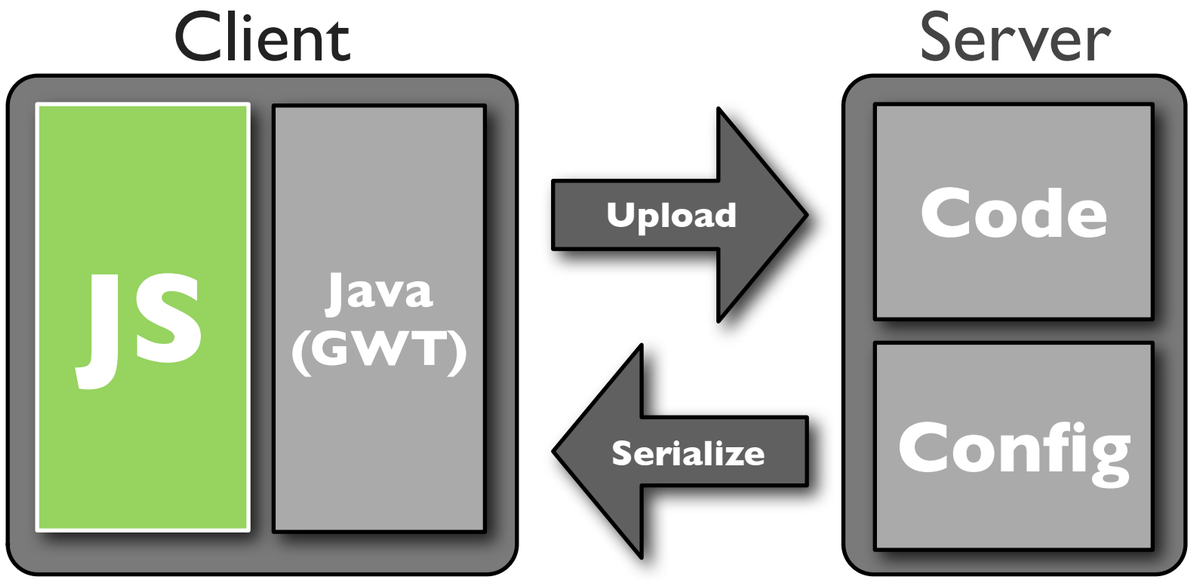
Meanwhile, in Javascript

```
function loadMeshData(binaryMeshData) {
    var floatMeshData = new Float32Array(binaryMeshData);

    //Process into WebGL-ready buffers.
    ...
}
```

How do I make my animation efficient?

Request Animation Frame

- Javascript setInterval()
  - Might be calling way too fast
  - Might be in a background tab!
- RequestAnimationFrame
  - Render batches efficiently
  - Browser adapts to available resources
- API still stabilizing, use a shim



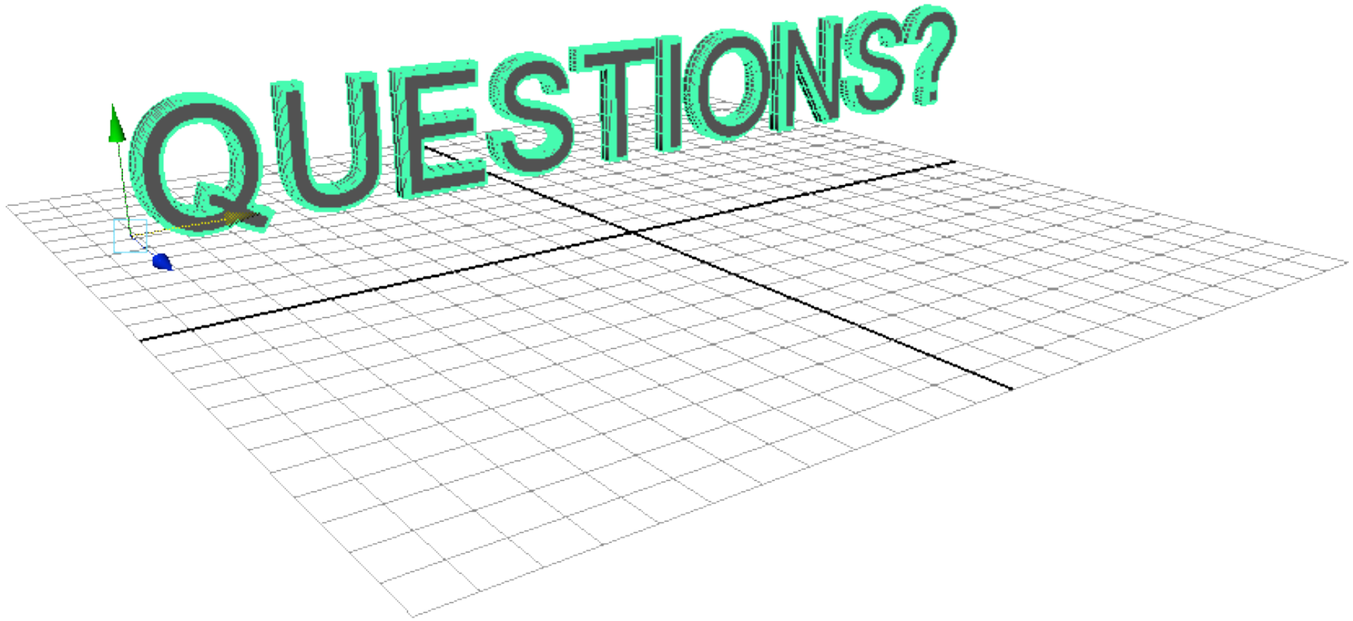
RequestAnimationFrame Code

```
// shim layer with setTimeout fallback -- Paul Irish
window.requestAnimFrame = (function() {
  return window.requestAnimationFrame       ||
         window.webkitAnimationFrame        ||
         window.mozRequestAnimationFrame    ||
         window.oRequestAnimationFrame      ||
         window.msRequestAnimationFrame     ||
         function(/*function*/ callback,
             /*DOMElement*/ element) {
             window.setTimeout(callback, 1000/60);
  };
})();
```

That’s it!

Recap

- Static Data
  - Cache with AppCache and HTTP cache
  - Use resource bundles and HTML5 File System API
- Dynamic Data
  - Datastore and Blobstore
  - Blobstore objects in Datastore wrappers
- Asset Processing
  - ImageService for simple 2D
  - Task Queues for parallel processing
  - App Engine Backends for heavy work
- Serialization
  - Binary data with XHR Level 2
  - UTF-8, delta compression, HTTP with GZip



FEEDBACK: Please provide feedback on this session at http://goo.gl/lj56w

HASHTAGS: #io2011, #DevTools