Effective Use of

Volley

Scott Weber

What is(n't) Volley?

Volley is a library intended to take care of many of the commonly used, but more advanced networking issues for you.

- Extensible ✓
- Battle-tested ✓
- Good for small transactions ✓
- Good for large downloads X
- Well documented X Coming Soon?

Why should I use Volley?

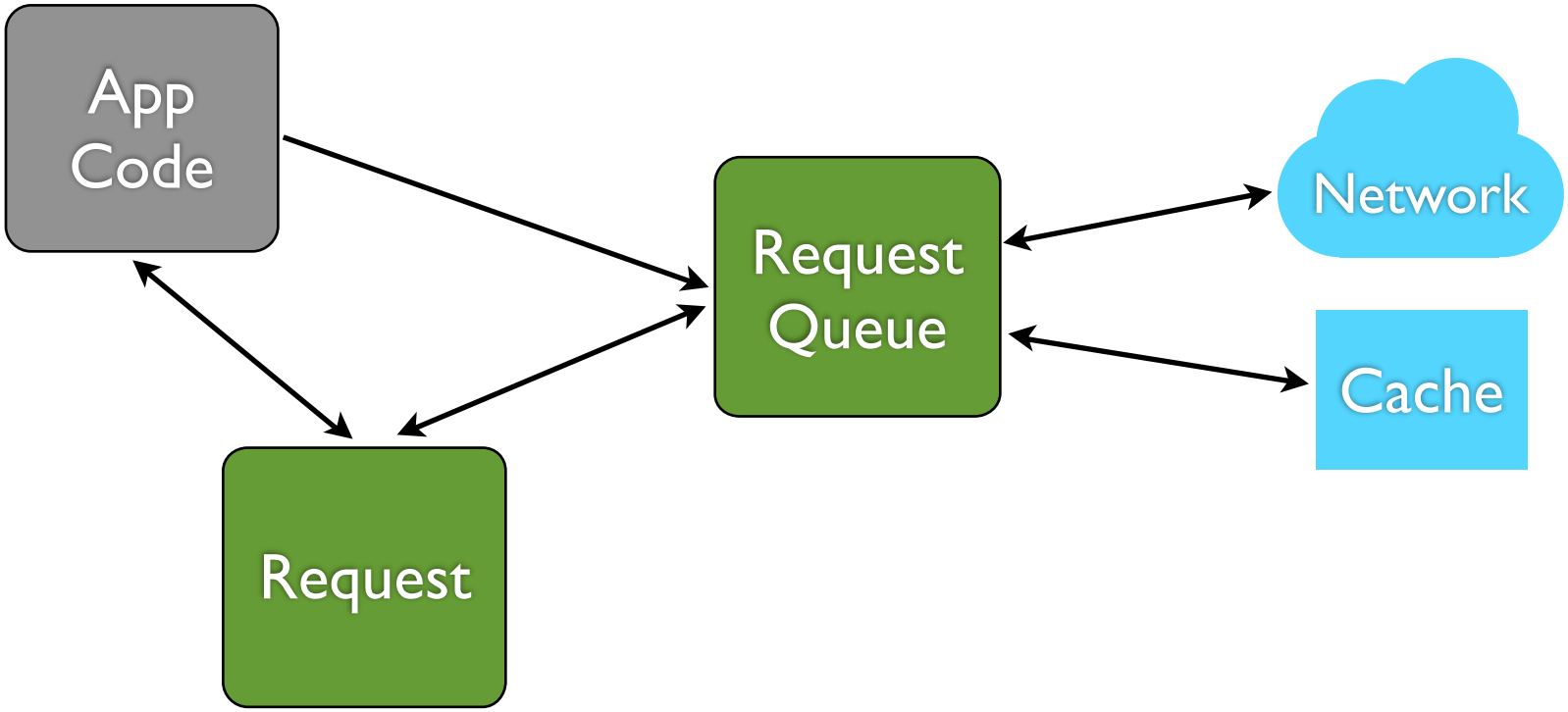
	Volley
• ApacheHttpClient vs. HttpUrlConnection	✓
• Asynchronous Downloading	✓
• Caching	✓
• Loading images into ListViews	✓

Is Volley My Only Choice?

- Picasso+Retrofit+OkHTTP
- RoboSpice
- Ion
- and many more...

How Volley Works

The 10k Foot View



Getting Started

Get the code:

```
git clone https://android.googlesource.com/platform/frameworks/volley
```

Modify project settings.gradle:

```
include ':lib:volley'
```

Modify app build.gradle:

```
dependencies {
    compile project(':lib:volley')
}
```

Getting Started

Establish a RequestQueue:

```
mRequestQueue = Volley.newRequestQueue(getApplicationContext());
```

Executing a Request

Build a Request:

```
String URL_BASE = "http://maps.googleapis.com/maps/api/elevation/json";
String url = URL_BASE + "?locations=42.282136,-83.749807&sensor=false";

Request elevationRequest = new JsonObjectRequest(url, null,
    new Listener<JSONObject>() {
        public void onResponse(JSONObject response) {
            // TODO: show the result to the user
        }
    },
    new ErrorListener() {
        public void onErrorResponse(VolleyError error) {
            // TODO: gracefully show the user the failure
        }
    }
);
```

Executing a Request

Run the Request:

```
elevationRequest.setTag(this);
mRequestQueue.add(elevationRequest);
```

Cancel the Request (if necessary):

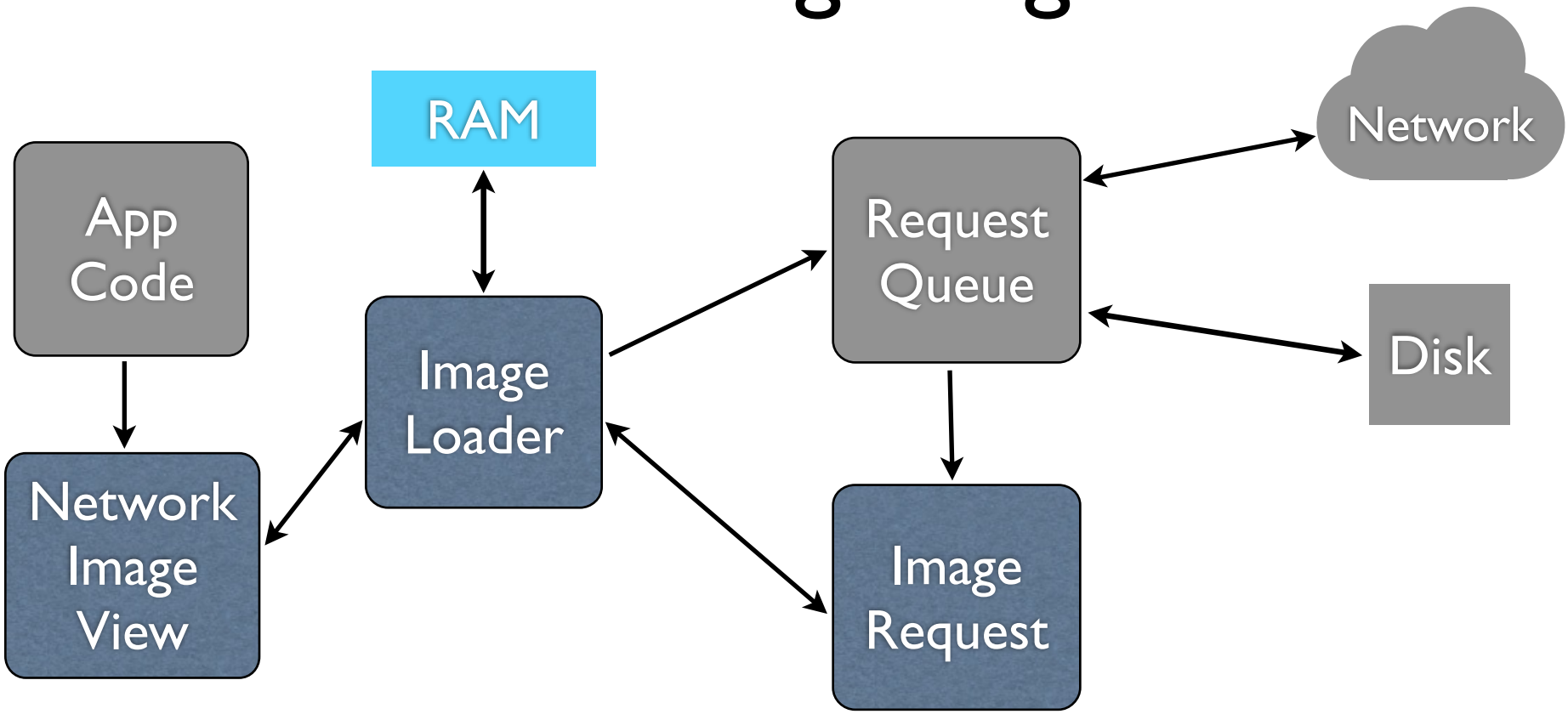
```
@Override
public void onStop() {
    super.onStop();
    mRequestQueue.cancelAll(this);
}
```

Getting Images

Additional Challenges:

- Set a placeholder while image is loading
- Set a placeholder when image load fails
- Don’t set images on recycled views
- Decoding bitmaps is time consuming

Getting Images



Getting Images

```
ImageLoader mImageLoader = new ImageLoader(mRequestQueue, mBitmapCache);

...

String BASE_URL = "http://maps.googleapis.com/maps/api/staticmap?zoom=17&size=300x300&sensor=false";
NetworkImageView imageView = (NetworkImageView) findViewById(R.id.image);
imageView.setImageUrl(BASE_URL + "&center=123+N+Ashley,+Ann+Arbor,MI", mImageLoader);
```

Caching

- Disk caching built-in (DiskBasedCache)
- Leverages caching at network layer when available (HttpUrlConnection)
- In-memory caching used for images only, must provide your own cache

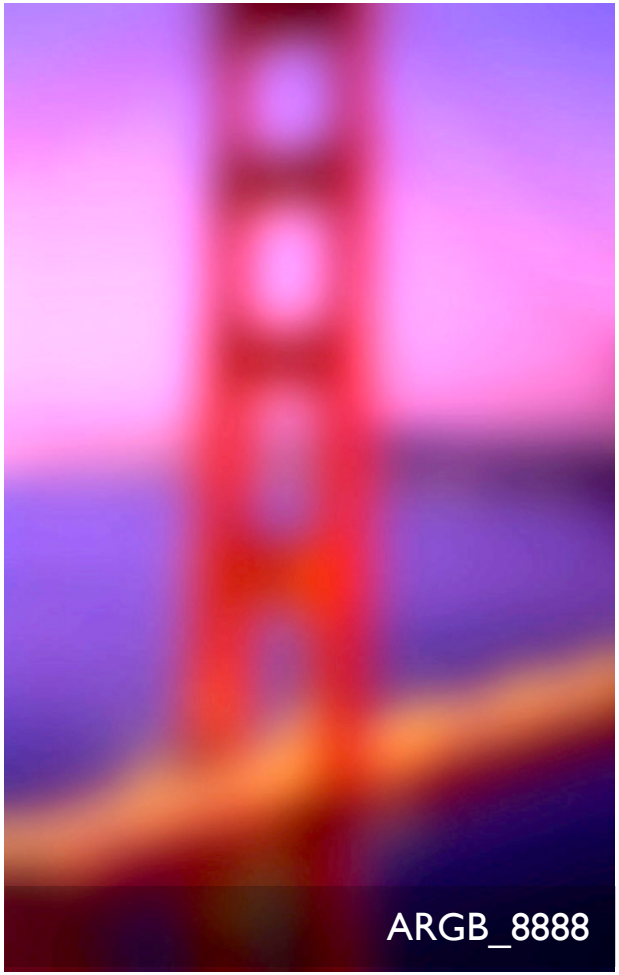
Resources:

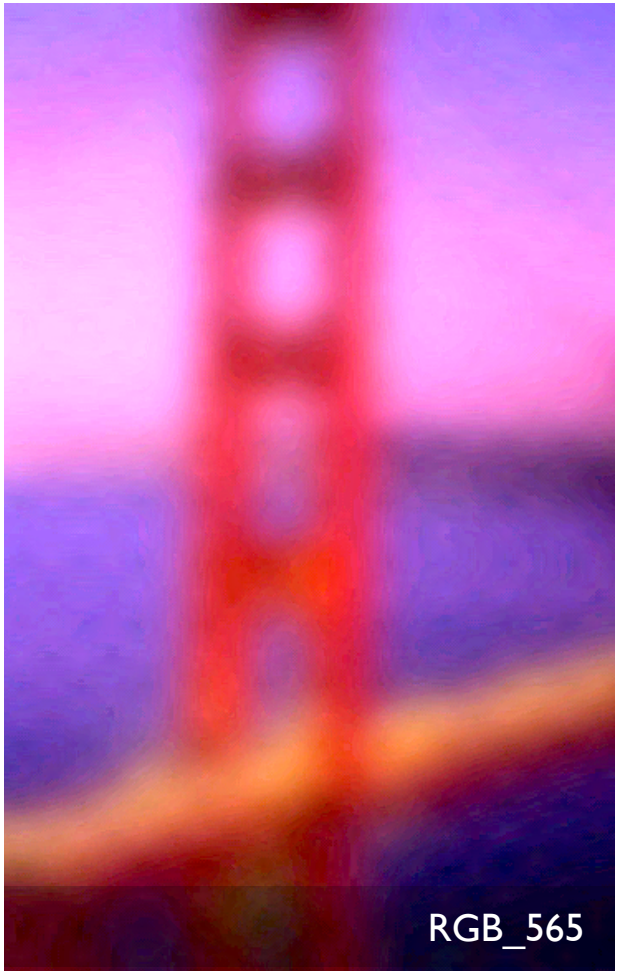
http://developer.android.com/training/displaying-bitmaps/cache-bitmap.html
https://developer.android.com/reference/android/support/v4/util/LruCache.html
http://stackoverflow.com/a/16684652/3032

Extending Volley - ImageLoader

- ImageLoader will only load images in RGB_565 mode
- Code that sets RGB_565 mode is not in a place that can be extended or overriden

Extending Volley - ImageLoader





Extending Volley - ImageLoader

So, let's fix the ImageLoader!

```
Request<?> newRequest = new ImageRequest(..., Config.RGB_565, ...);
Request<?> newRequest = getImageRequest(...);

protected ImageRequest getImageRequest(String requestUrl, Listener<Bitmap> listener, int maxWidth,
        int maxHeight, ErrorListener errorListener) {
    return new ImageRequest(requestUrl, listener, maxWidth, maxHeight, Config.RGB_565, errorListener);
}
```

Now we can do this:

```
public class HighQualityImageLoader extends ImageLoader {
    public HighQualityImageLoader(RequestQueue queue, ImageCache imageCache) {
        super(queue, imageCache);
    }

    @Override
    protected ImageRequest getImageRequest(String requestUrl, Listener<Bitmap> listener, int maxWidth,
            int maxHeight, ErrorListener errListener) {
        return new ImageRequest(requestUrl, listener, maxWidth, maxHeight, Config.ARGB_8888, errListener);
    }
}
```

https://gist.github.com/scottdweber/9799142

Extending Volley - ImageRequest

An excellent opportunity for pre-processing images:

```
public class BlurredImageRequest extends ImageRequest {
    @Override
    protected Response<Bitmap> parseNetworkResponse(NetworkResponse response) {
        Response<Bitmap> bitmapResponse = super.parseNetworkResponse(response);

        // TODO -- 1. prevent double-processing image returned from cache
        // TODO -- 2. synchronize to avoid OOM when processing lots of images
        if (bitmapResponse.isSuccess()) {
            Bitmap bmp = blurBitmap(bitmapResponse.result);
            if (bmp != null) {
                bitmapResponse.cacheEntry.data = compressBitmap(bmp);
                bitmapResponse = Response.success(bmp, bitmapResponse.cacheEntry);
            }
        }

        return bitmapResponse;
    }
}
```

Extending Volley - JSON Handlers

- Write a little “glue” to use your existing response handlers with Volley

Imagine a set of JSON Response Handlers inheriting from:

```
public abstract class JSONResponseHandler<T> {
    public abstract T handleJSON(JSONObject response);
}
```

Extending Volley - JSON Handlers

A simple "glue" implementation:

```
public class JSONResponseHandlerRequest<T> extends JsonRequest<T> {
    private JSONResponseHandler<T> mResponseHandler;

    public JSONResponseHandlerRequest(String url, JSONResponseHandler<T> responseHandler,
            Listener<T> listener, ErrorListener errorListener) {
        super(Method.GET, url, null, listener, errorListener);
        mResponseHandler = responseHandler;
    }

    // TODO: error handling
    protected Response<T> parseNetworkResponse(NetworkResponse response) {
        String jsonString = new String(response.data, HttpHeaderParser.parseCharset(response.headers));
        JSONObject json = new JSONObject(jsonString);
        T result = mResponseHandler.handleJson(json);

        Cache.Entry cacheEntry = HttpHeaderParser.parseCacheHeaders(response);
        return Response.success(result, cacheEntry);
    }
}
```

Extending Volley - JSON Handlers

Use the glue:

```
Request weatherRequest = new JSONResponseHandlerRequest<WeatherForecast>(forecastUrl,
    new WeatherForecastResponseHandler(),
    new Listener<WeatherForecast>() {
        public void onResponse(WeatherForecast response) {
            // TODO: show the result to the user
        }
    },
    new ErrorListener() {
        public void onErrorReponse(VolleyError error) {
            // TODO: gracefully show the user the failure
        }
    }
);
```

Extending Volley - Cache

- Volley provides two implementations: DiskBasedCache, NoCache
- Intended for disk-based caching
- Can provide cache implementation that is specific to your business rules
  - Keep critical data in the cache, even if it would have been evicted by an LRU algorithm
  - Maybe frequent use indicates a greater need to refresh from data source

Extending Volley - Network

Think Outside the Network

- Content Providers (e.g. Contacts) are supposed to be queried off the main thread
- Volley already provides all the mechanisms for asynchronous queries, but only if you are accessing the network
- Custom Network implementation to the rescue!

Extending Volley - Network

Think Outside the Network

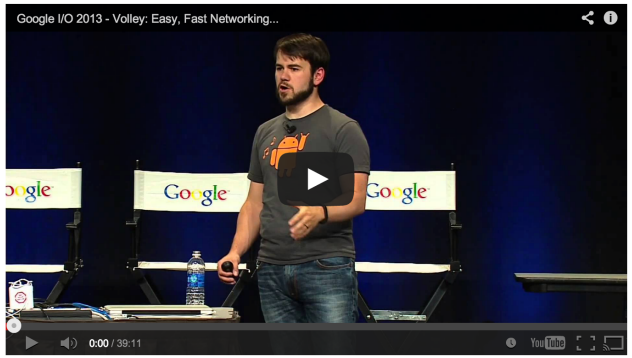
```
public class ContactNetwork implements Network {

    public NetworkResponse performRequest(Request<?> request) {
        Uri uri = Uri.parse(request.getUrl());
        byte[] data = ContactLookupUtils.getPhotoBytes(uri);

        if (data != null && data.length > 0) {
            return new NetworkResponse(data);
        }
        else {
            throw new NetworkError();
        }
    }
}
```

Resources

- I/O 2013 Presentation
  - Video: http://goo.gl/F3Km5Y
  - Slides: http://goo.gl/NjeACV



-  +FicusKirkpatrick
- Volley Users Email List: http://goo.gl/r82FzD
- Android Developers Backstage, Episode 8: http://goo.gl/lw9csA

Questions

Scott Weber

scottweber.com

+ScottWeberD

@ScottDWeber