

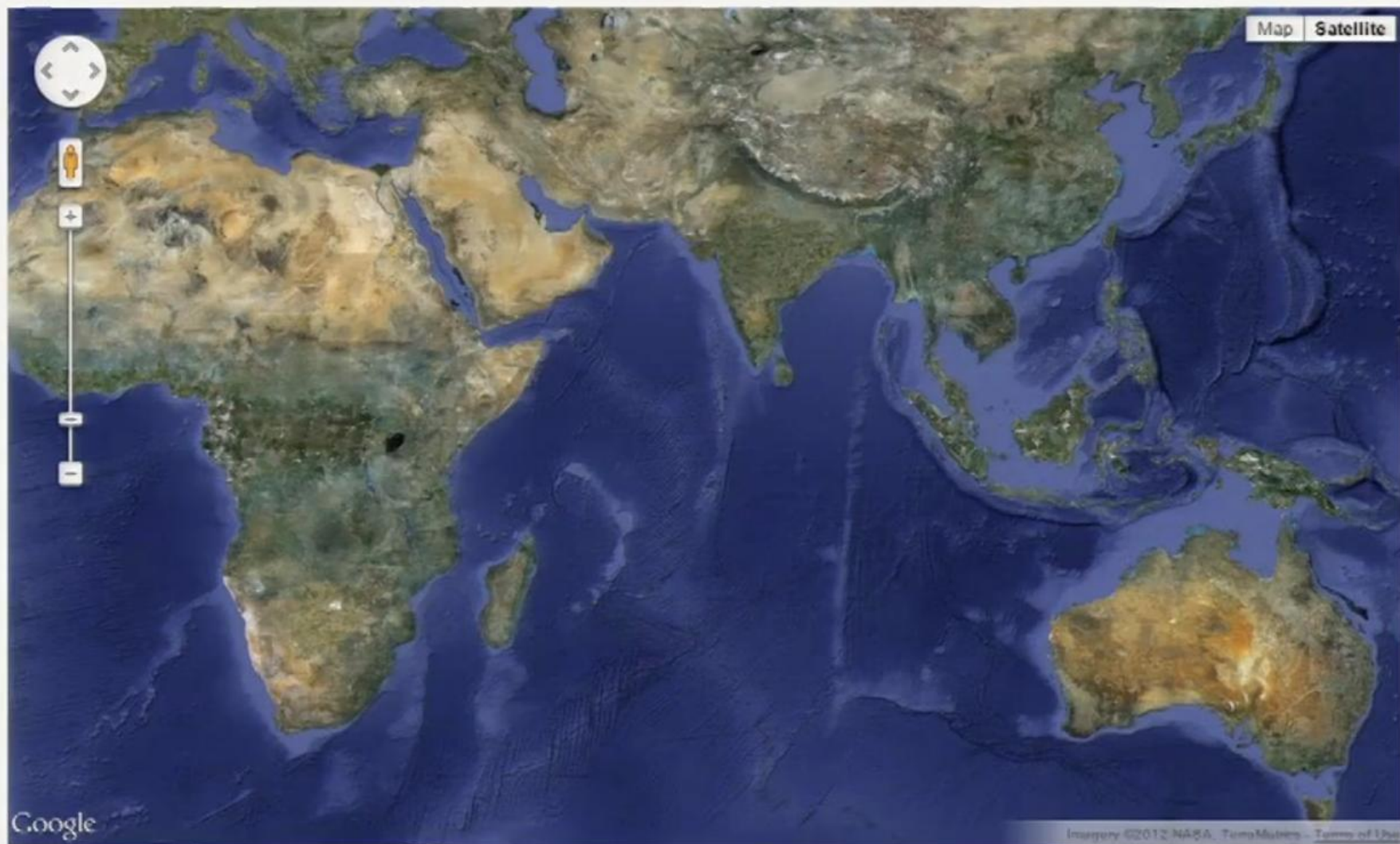


Computing Map Tiles

With Go on App Engine

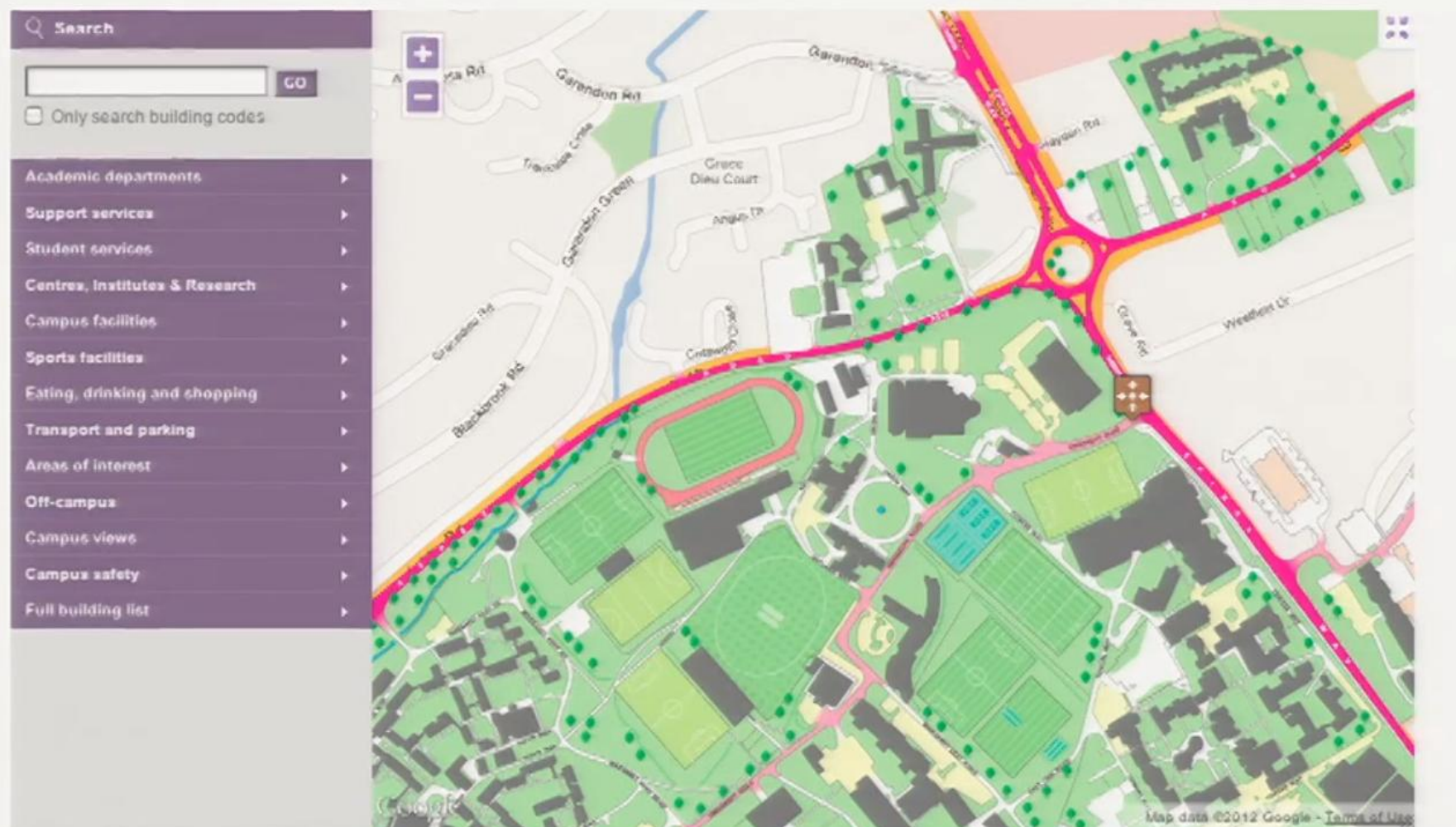
Chris Broadfoot - Google Sydney
Andrew Gerrand - Google Sydney











Loughborough University – maps.lboro.ac.uk

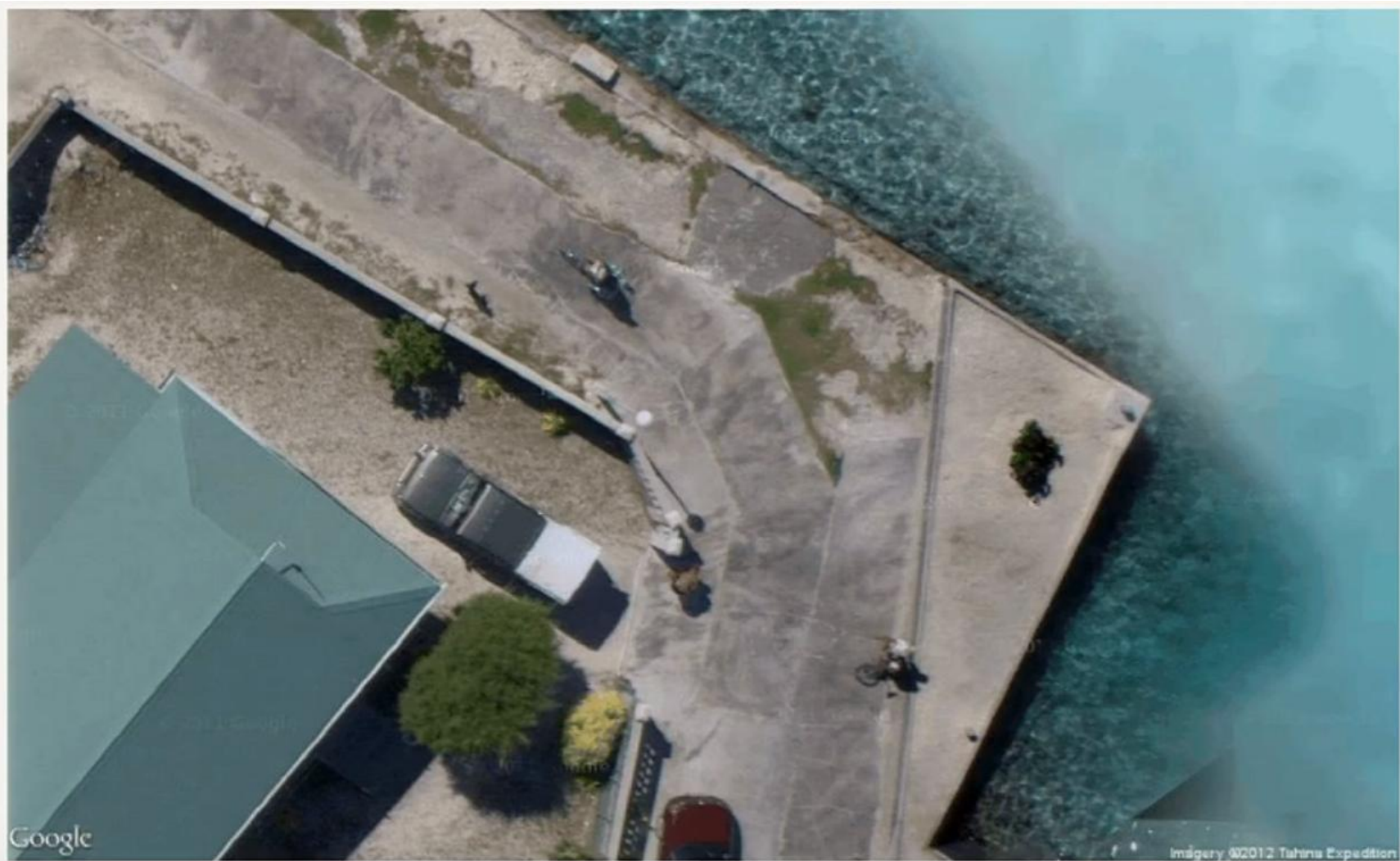


Google

Imagery ©2012 DigitalGlobe, GeoEye, TerraVision, and others

#io12

7/57



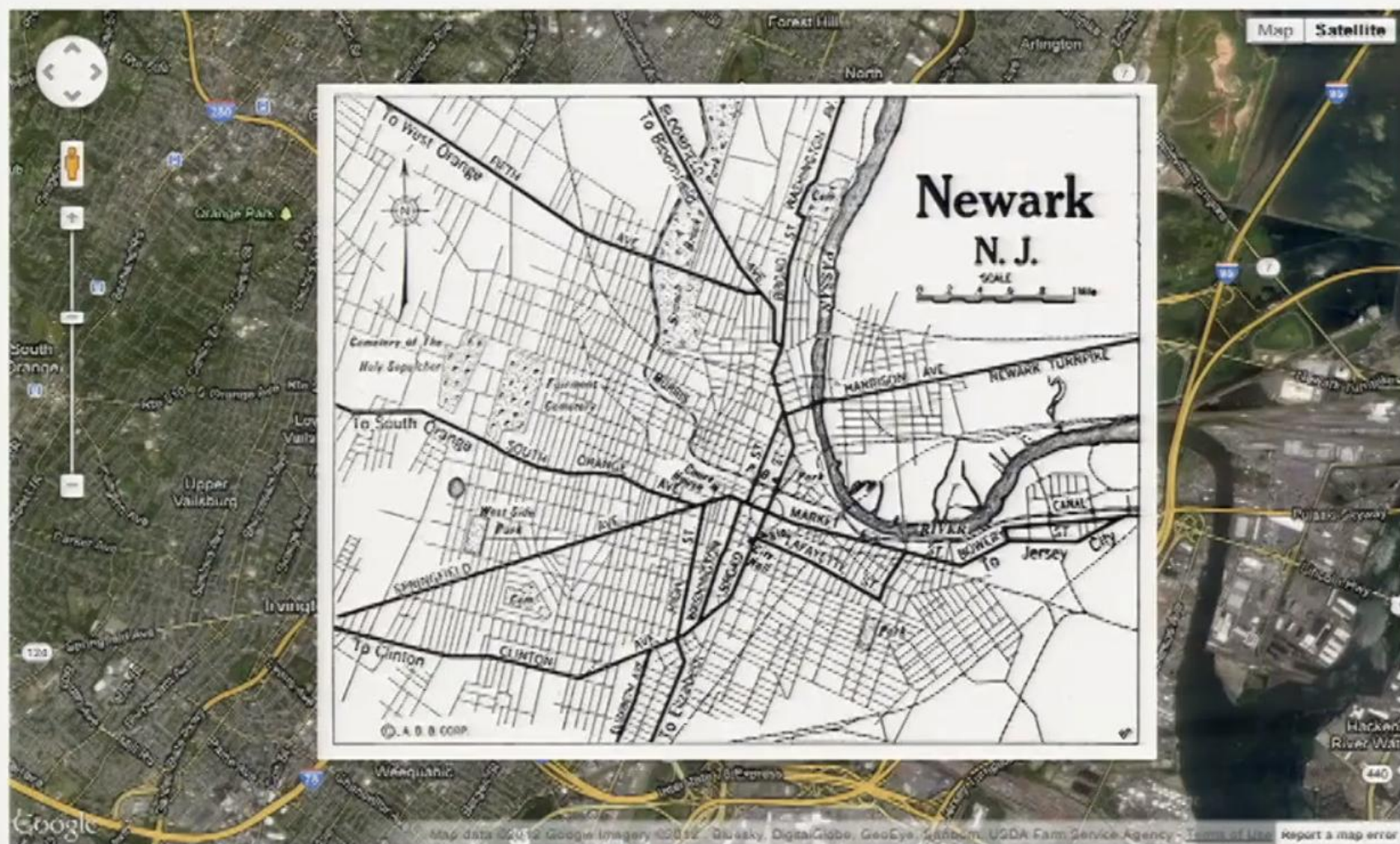
Google

Imagery ©2012 Tahira Expedition

Ground Overlay

JAVASCRIPT

```
var url = "https://www.lib.utexas.edu/maps/historical/newark_nj_1922.jpg";  
var imageBounds = new google.maps.LatLngBounds(  
    new google.maps.LatLng(40.712216, -74.22655),  
    new google.maps.LatLng(40.773941, -74.12544));  
  
var groundOverlay = new google.maps.GroundOverlay(url, imageBounds);  
groundOverlay.setMap(map);
```

Why tiles?



Why tiles?



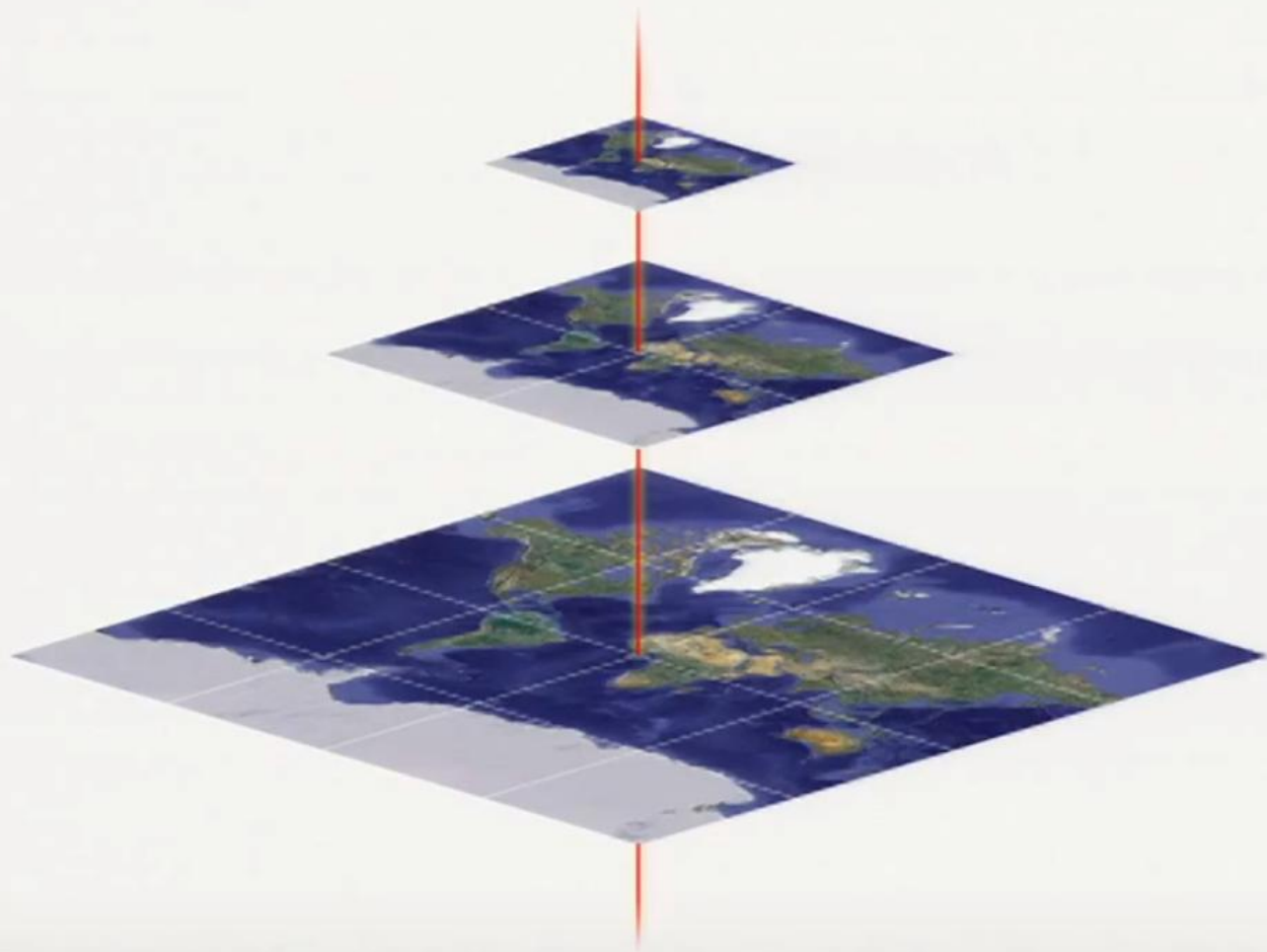
Why tiles?

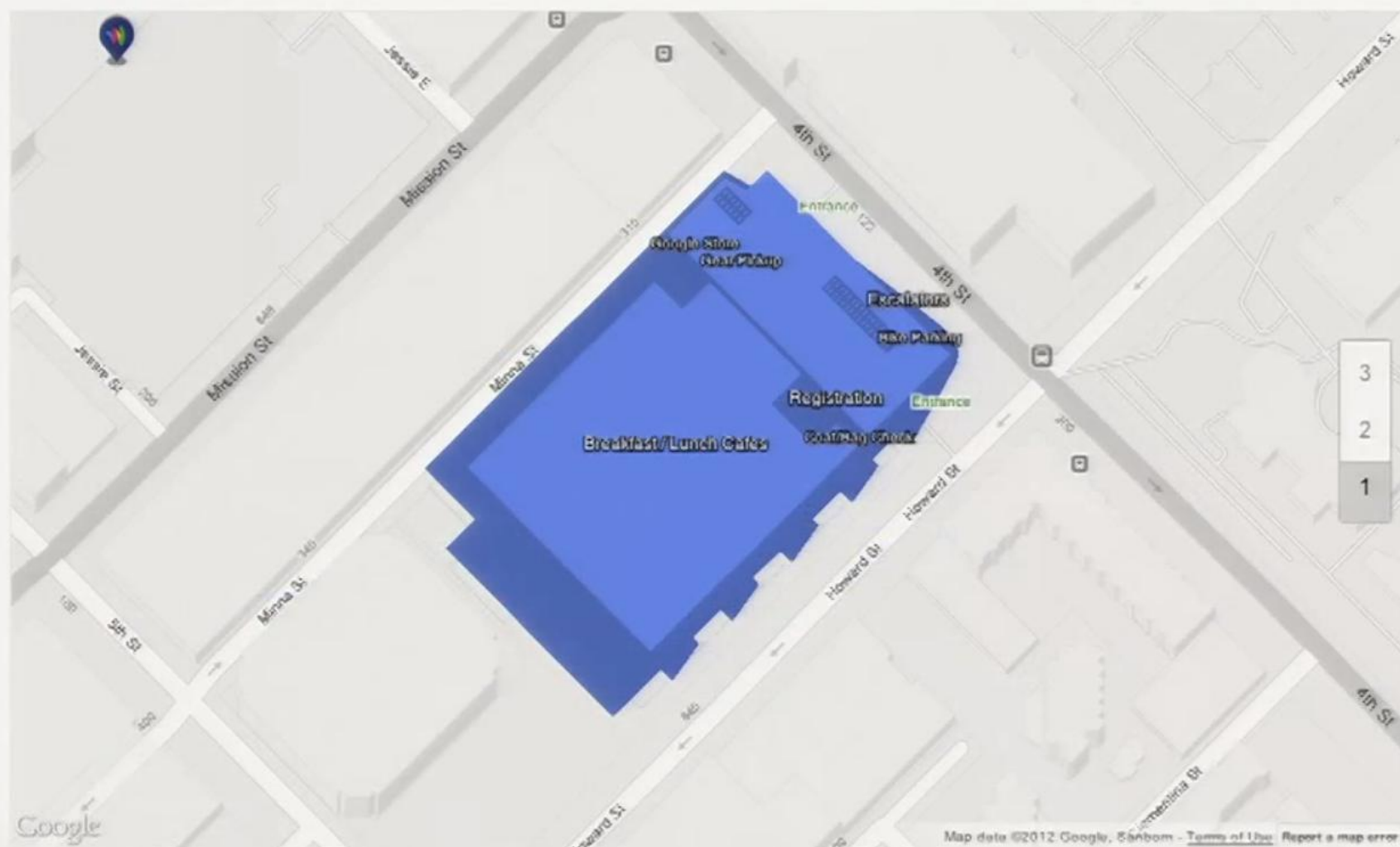


Why tiles?



Why tiles?







Tile Overlay

JAVASCRIPT

```
var overlay = new google.maps.ImageMapType({  
  getTileUrl: function(coord, zoom) {  
    return tileBase + '/' + zoom + '/' + coord.x + '/' + coord.y + '.png';  
  },  
  tileSize: new google.maps.Size(256, 256)  
});  
map.overlayMapTypes.push(overlay);
```


Generating custom tile sets

gdal.org

```
$ gdal_translate \  
-gcp 0 0 -122.56756591796875 37.69663646445409 \  
-gcp 800 0 -122.40689086914062 37.86378845789171 \  
-gcp 800 600 -122.35195922851562 37.80630478370753 \  
"in.png" "out.png"
```

SH

```
$ gdal2tiles.py \  
-s "EPSG:4326" \  
-z 16-19 \  
"out.png" "out"
```

SH

maptiler.org

Today's approach: Overlay Tiler

code.google.com/p/overlay-tiler



Introducing Overlay Tiler

Demo



Overlay Tiler

Enter a location



adg@google.com

Log out

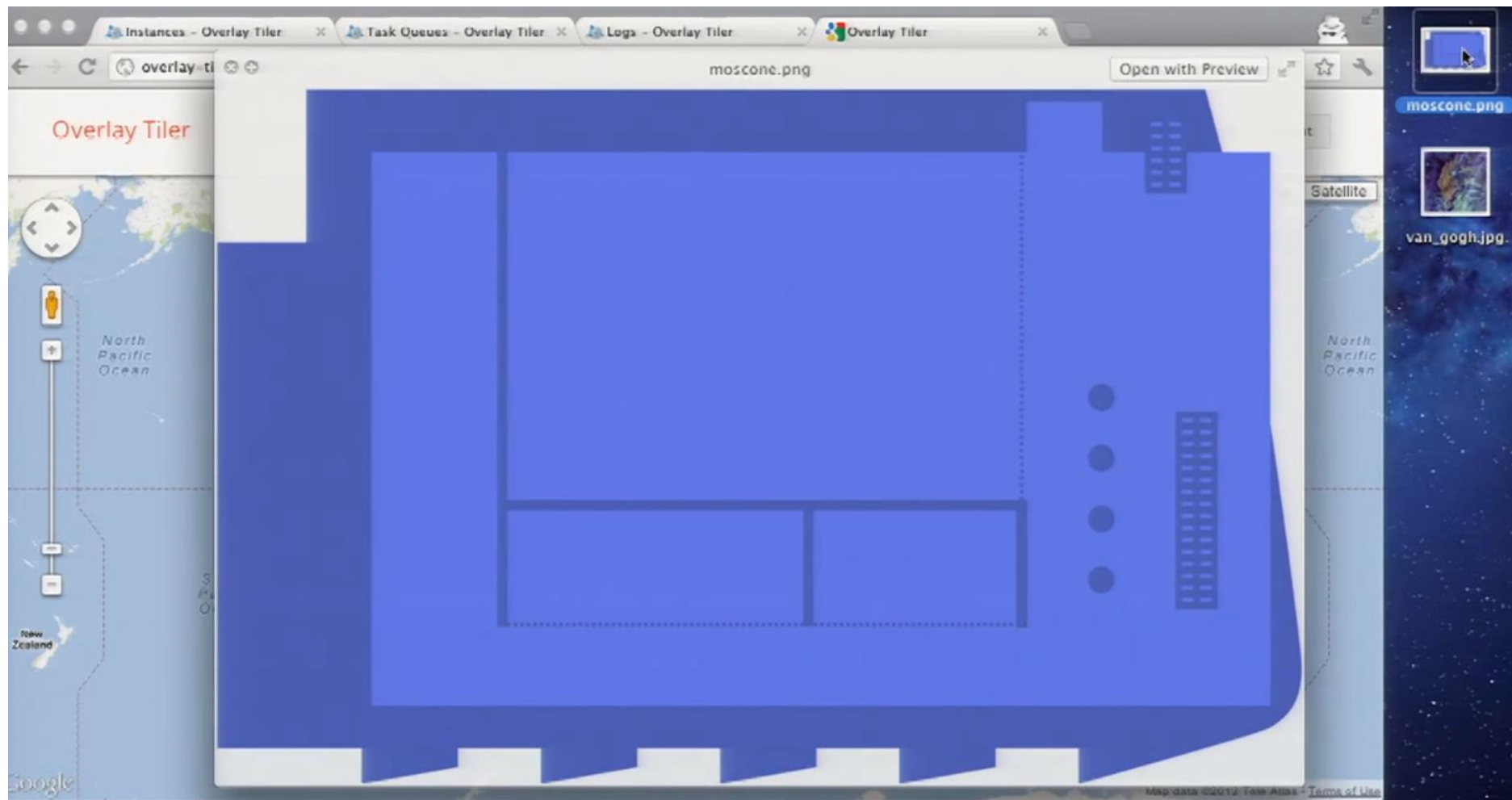


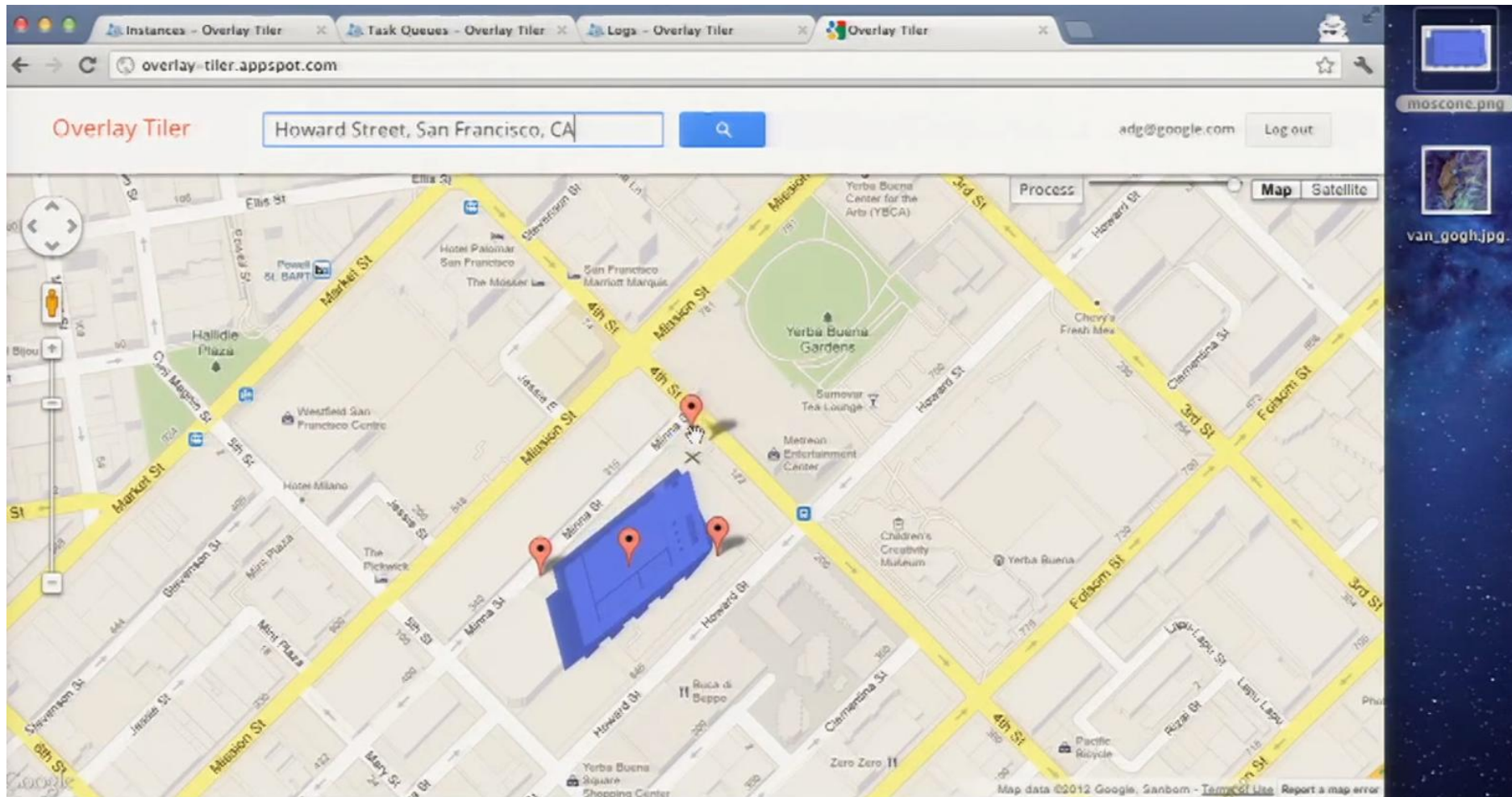
Map

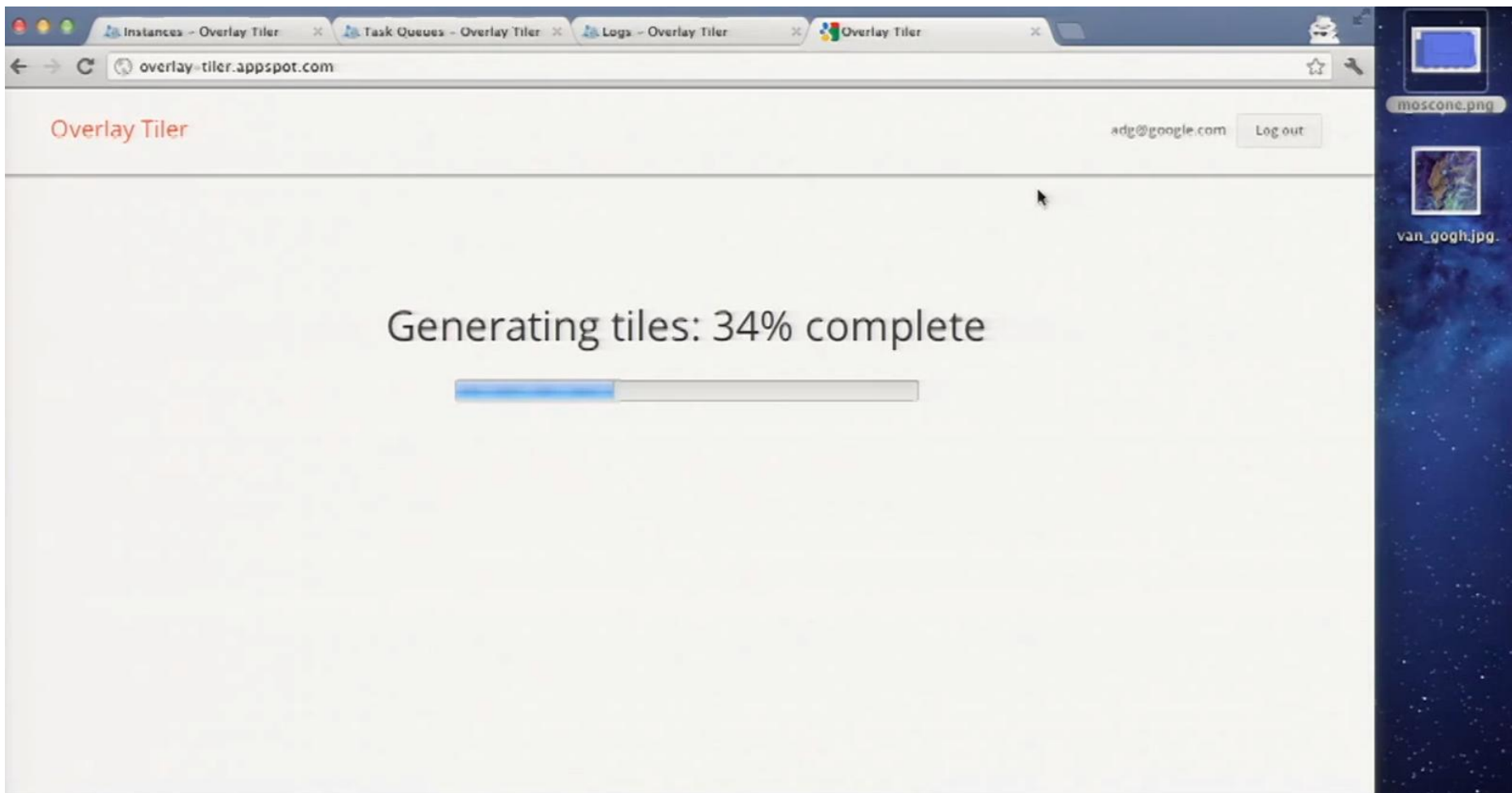
Satellite

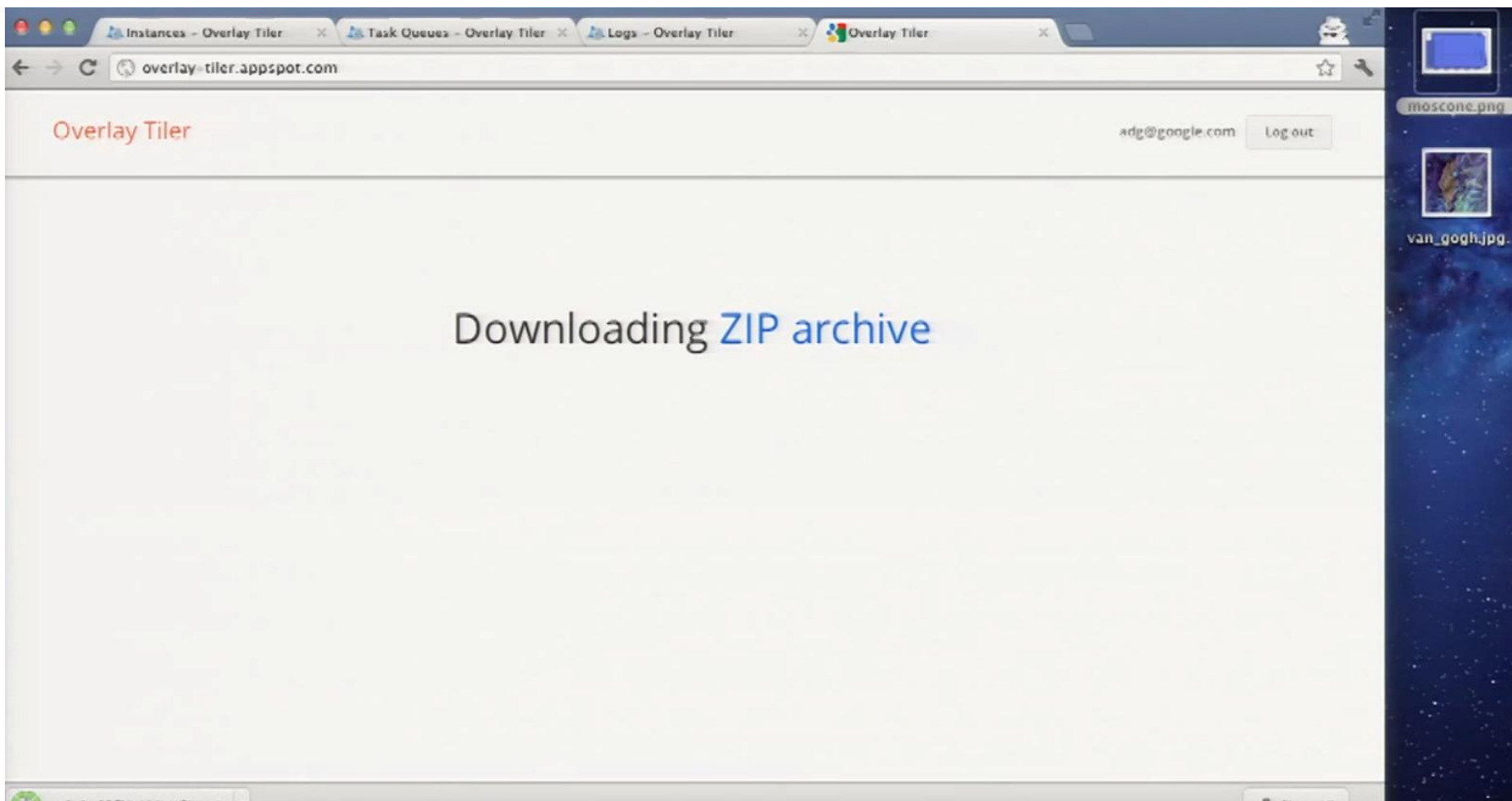
Google

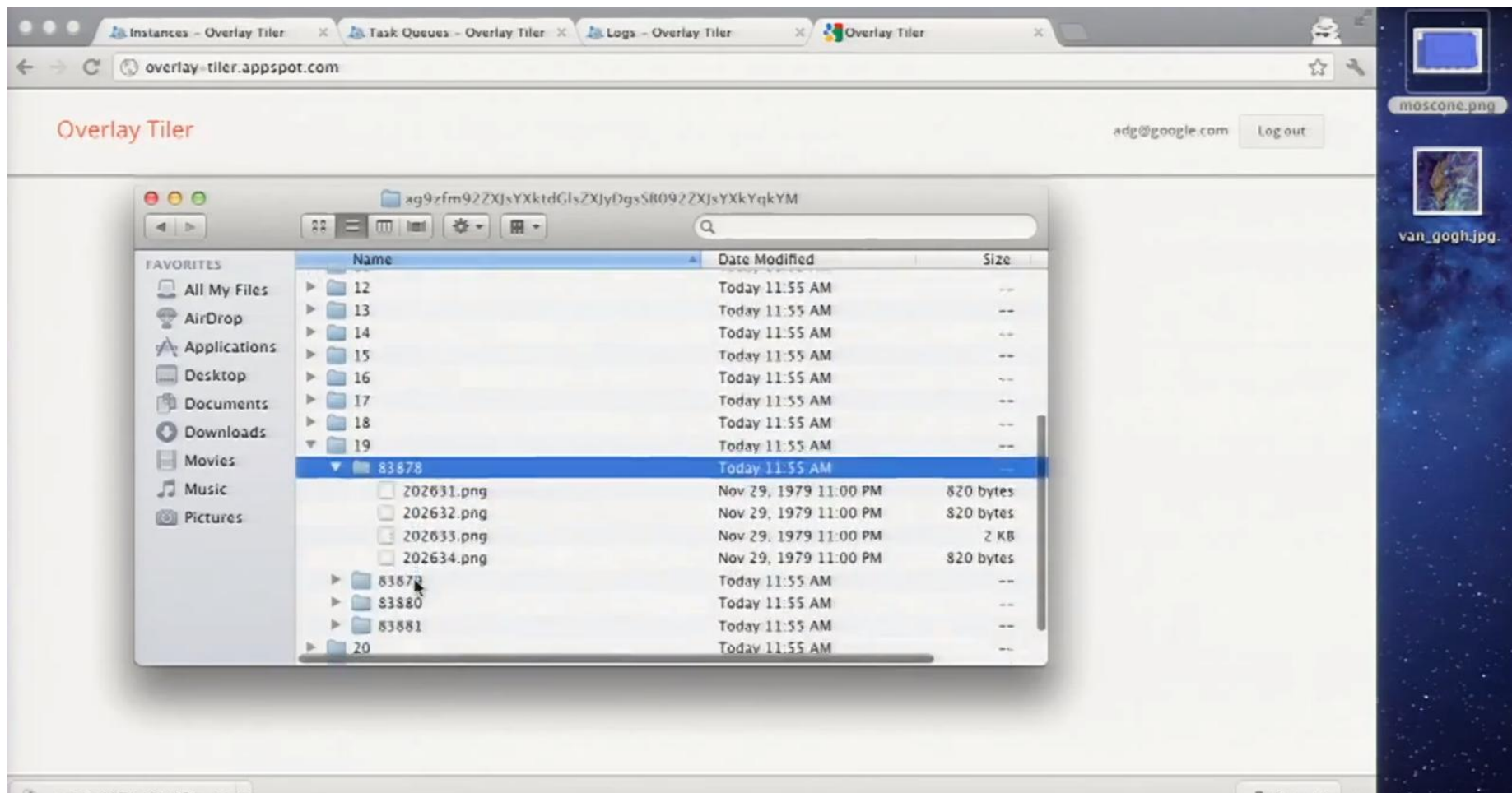
Map data ©2012 Tele Atlas - Terms of Use

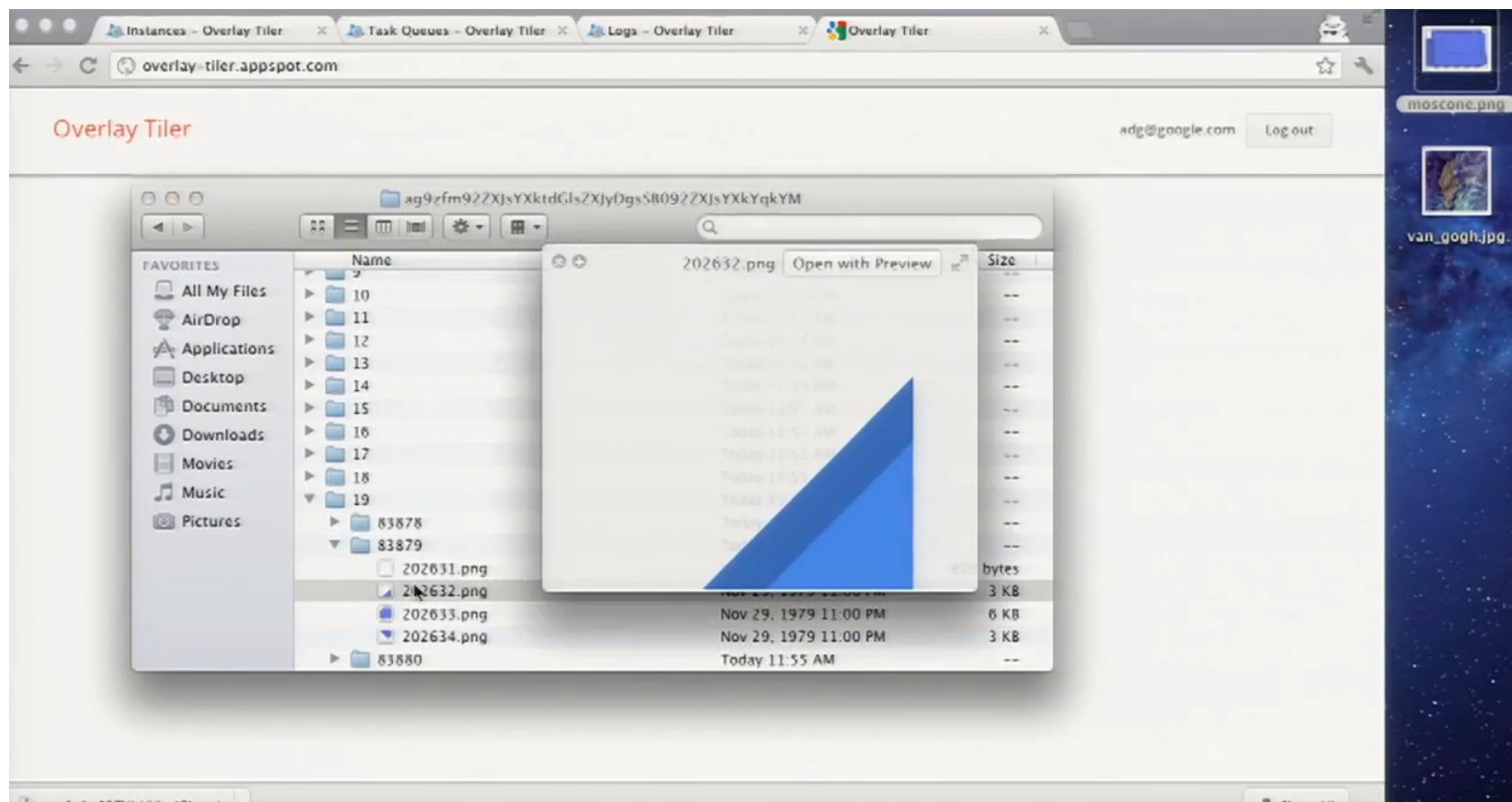


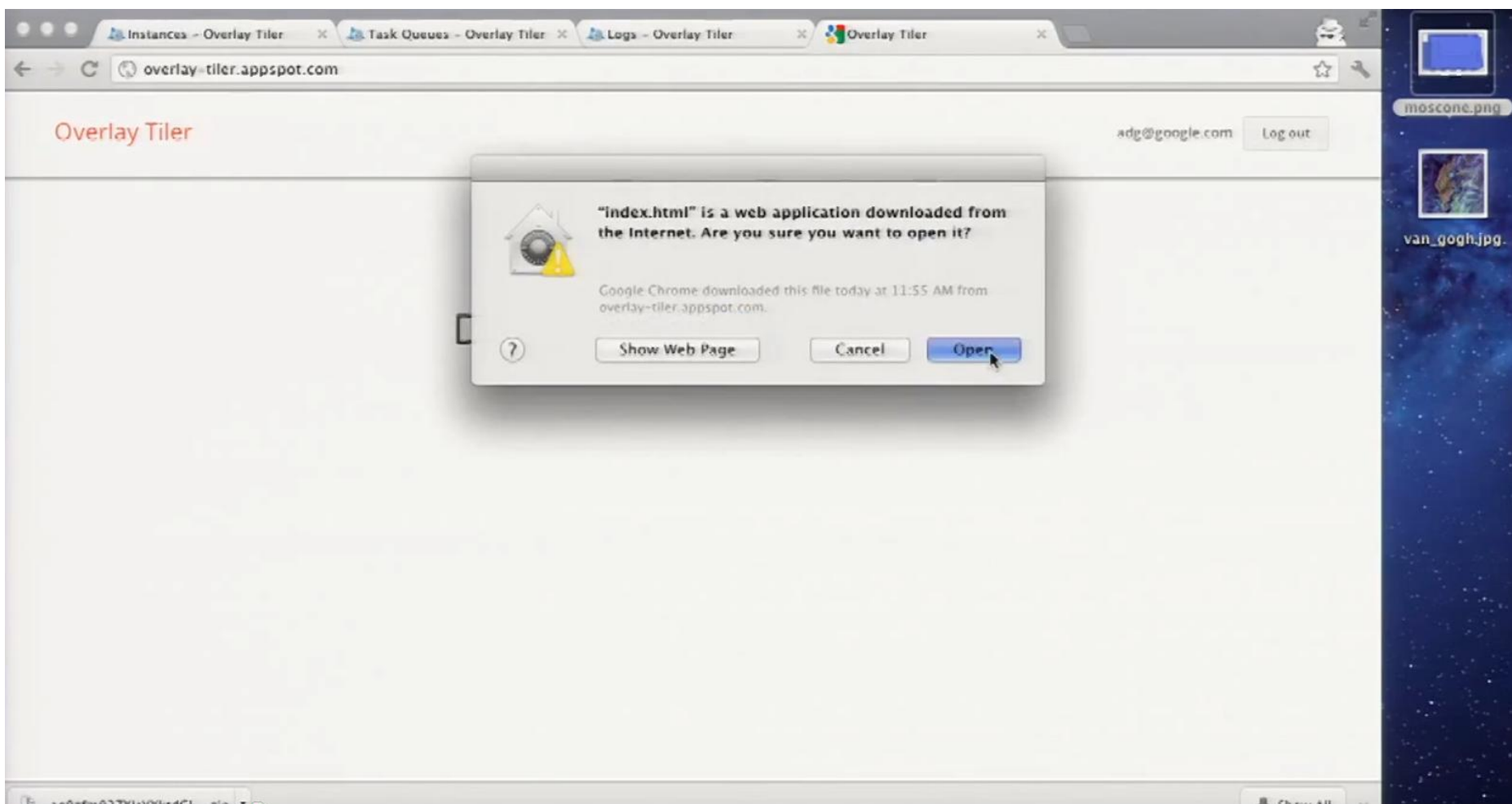


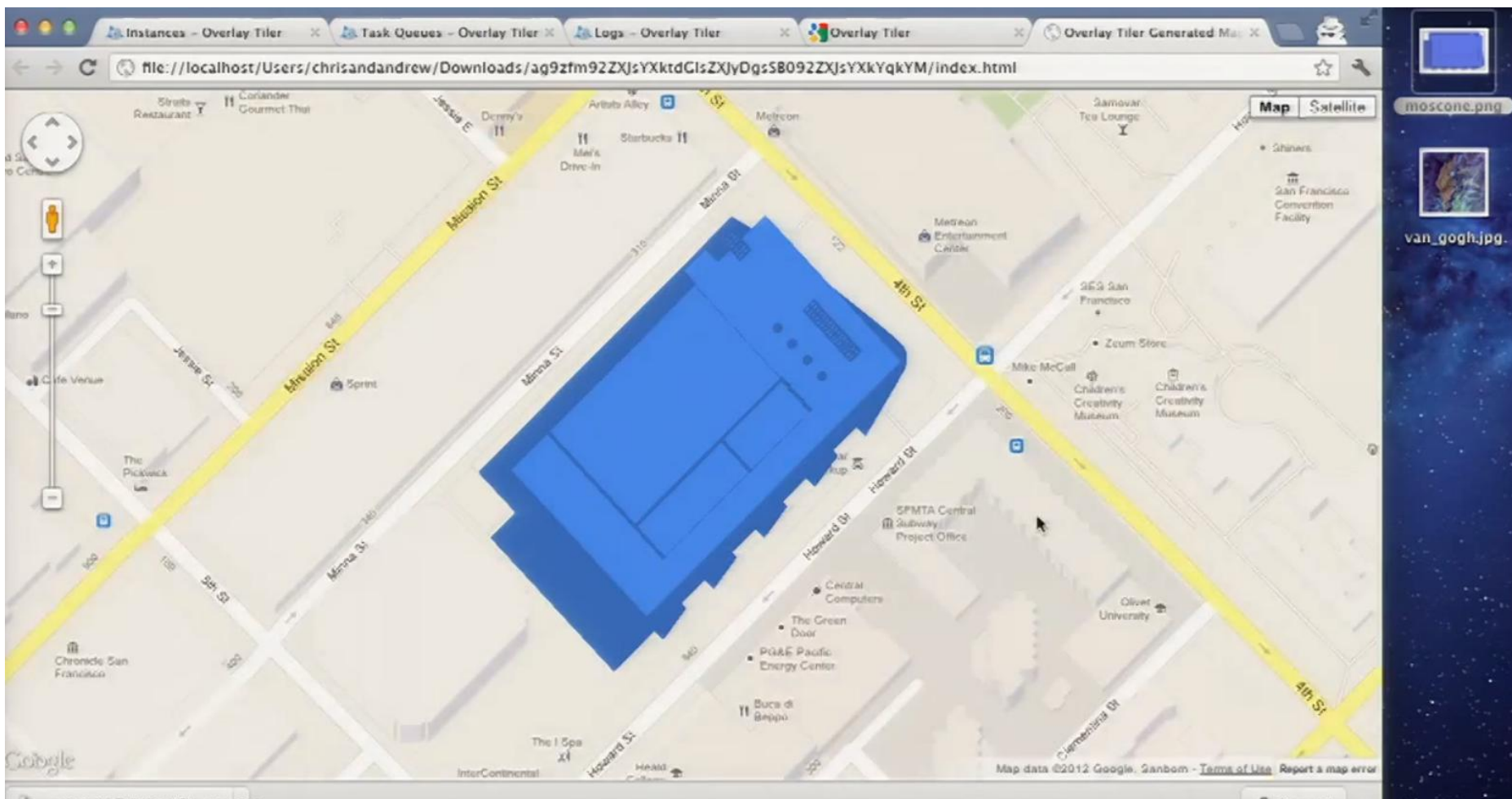












The technology stack

- Google Maps API
- Google Places API
- HTML5
 - CSS3 transform
 - Drag and drop
 - XHR2
- Go
- Google App Engine
 - Datastore
 - Blobstore
 - Backends
 - Task Queues
 - Channel API

Drag and Drop

JAVASCRIPT

```
var drop = document.querySelector('#map');
drop.addEventListener('drop', function(e) {
  stopEvent(e);
  var files = e.dataTransfer.files;
  if (!files.length) {
    window.alert('No file uploaded');
    return;
  }
  var imageURL = (window.URL || window.webkitURL).createObjectURL(files[0]);
  var rect = map.getDiv().getBoundingClientRect();
  var x = e.pageX - rect.left;
  var y = e.pageY - rect.top;
  var overlay = new Overlay(imageURL, x, y);
  ...
}, false);
```

Drag and Drop

JAVASCRIPT

```
var drop = document.querySelector('#map');
drop.addEventListener('drop', function(e) {
  stopEvent(e);
  var files = e.dataTransfer.files;
  if (!files.length) {
    window.alert('No file uploaded');
    return;
  }
  var imageURL = (window.URL || window.webkitURL).createObjectURL(files[0]);
  var rect = map.getDiv().getBoundingClientRect();
  var x = e.pageX - rect.left;
  var y = e.pageY - rect.top;
  var overlay = new Overlay(imageURL, x, y);
  ...
}, false);
```

Adding the overlay

`google.maps.OverlayView`

Implement:

- `onAdd()`
- `draw()`
- `onRemove()`

Methods:

- `getMap()`
- `getPanels()`
- `getProjection()`
- `setMap(map)`

Adding the overlay

JAVASCRIPT

```
function Overlay(src, x, y) {  
  var el = this.el_ = document.createElement('img');  
  el.style.width = el.style.height = '1px';  
  el.onload = this.setup_.bind(this);  
  el.src = src;  
  el.style.position = 'absolute';  
  el.style.transformOrigin = el.style.webkitTransformOrigin = '0 0';  
  this.start_ = new google.maps.Point(x, y);  
}  
Overlay.prototype = new google.maps.OverlayView;  
Overlay.prototype.setup_ = function() {  
  this.aspectRatio_ = this.el_.naturalWidth / this.el_.naturalHeight;  
  ...  
  this.getPanes().overlayImage.appendChild(this.el_);  
  this.setInitialGCP_(this.start_);  
};
```

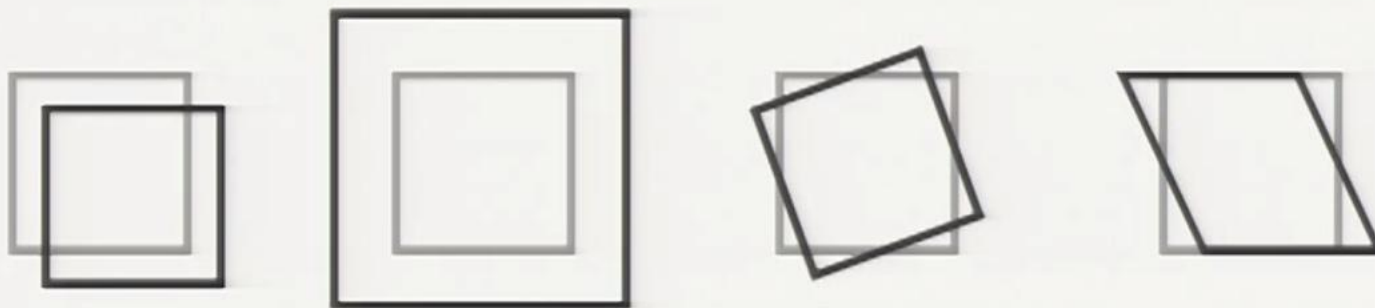
Editing the overlay

```
function OverlayEditor(overlay) {  
  this.set('overlay', overlay);  
  this.bindTo('map', overlay);  
  this.markers_ = [  
    this.addGCPControl_('topLeft'),  
    this.addGCPControl_('topRight'),  
    this.addGCPControl_('bottomRight')  
  ];  
  ...  
}  
OverlayEditor.prototype.addGCPControl_ = function(anchor) {  
  var marker = new google.maps.Marker({  
    draggable: true  
  });  
  marker.bindTo('map', this);  
  this.get('overlay').bindTo(anchor, marker, 'position');  
  this.set(anchor, marker);  
  return marker;  
};
```

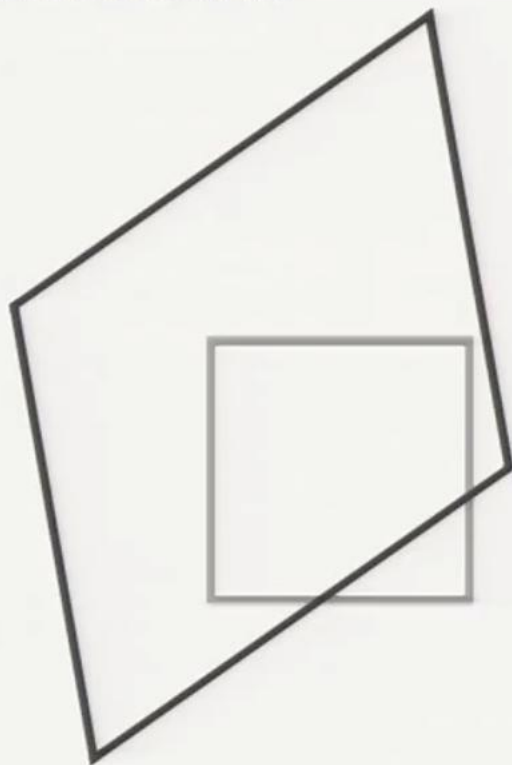
Editing the overlay

```
function OverlayEditor(overlay) {  
  this.set('overlay', overlay);  
  this.bindTo('map', overlay);  
  this.markers_ = [  
    this.addGCPControl_('topLeft'),  
    this.addGCPControl_('topRight'),  
    this.addGCPControl_('bottomRight')  
  ];  
  ...  
}  
OverlayEditor.prototype.addGCPControl_ = function(anchor) {  
  var marker = new google.maps.Marker({  
    draggable: true  
  });  
  marker.bindTo('map', this);  
  this.get('overlay').bindTo(anchor, marker, 'position');  
  this.set(anchor, marker);  
  return marker;  
};
```

Affine Transformations



Affine Transformations



Rendering the Overlay

```
Overlay.prototype.draw = function() {  
  if (!this.added_) return; // not ready yet  
  requestAnimationFrame(this.draw_.bind(this), this.el_);  
};  
Overlay.prototype.draw_ = function() {  
  this.el_.style.transform = this.el_.style.webkitTransform = this.computeTransform_();  
};  
Overlay.prototype.computeTransform_ = function() {  
  var proj = this.getProjection();  
  var tl = proj.fromLatLngToDivPixel(this.get('topLeft'));  
  var tr = proj.fromLatLngToDivPixel(this.get('topRight'));  
  var br = proj.fromLatLngToDivPixel(this.get('bottomRight'));  
  return 'matrix(' +  
    [tr.x - tl.x, tr.y - tl.y, br.x - tr.x, br.y - tr.y, tl.x, tl.y] + ')';  
};
```

Rendering the Overlay

```
Overlay.prototype.draw = function() {  
  if (!this.added_) return; // not ready yet  
  requestAnimationFrame(this.draw_.bind(this), this.el_);  
};  
Overlay.prototype.draw_ = function() {  
  this.el_.style.transform = this.el_.style.webkitTransform = this.computeTransform_();  
};  
Overlay.prototype.computeTransform_ = function() {  
  var proj = this.getProjection();  
  var tl = proj.fromLatLngToDivPixel(this.get('topLeft'));  
  var tr = proj.fromLatLngToDivPixel(this.get('topRight'));  
  var br = proj.fromLatLngToDivPixel(this.get('bottomRight'));  
  return 'matrix(' +  
    [tr.x - tl.x, tr.y - tl.y, br.x - tr.x, br.y - tr.y, tl.x, tl.y] + ')';  
};
```

Handling large images

```
Overlay.MAX_DIMENSION_ = 800;
```

JAVASCRIPT

```
if (img.naturalWidth * img.naturalHeight > maxPixels) {  
  this.resize_(Overlay.MAX_DIMENSION_);  
}
```

JAVASCRIPT

```
Overlay.prototype.resize_ = function(maxDimension) {  
  var el = this.el_;  
  var canvas = document.createElement('canvas');  
  canvas.width = maxDimension;  
  canvas.height = canvas.width / this.aspectRatio_;  
  var ctx = canvas.getContext('2d');  
  ctx.drawImage(el, 0, 0, canvas.width, canvas.height);  
  el.src = canvas.toDataURL();  
};
```

JAVASCRIPT



Uploading using XHR

```
drop.addEventListener('drop', function(e) {  
  var files = e.dataTransfer.files;  
  ...  
  var xhr = new XMLHttpRequest;  
  xhr.open('POST', document.querySelector('#upload-url').value, true);  
  xhr.onload = function(e) {  
    ...  
  }  
  ...  
  var form = new FormData;  
  form.append('overlay', files[0]);  
  xhr.send(form);  
}
```



The back end



What is App Engine?

A web application stack built on Google's infrastructure.

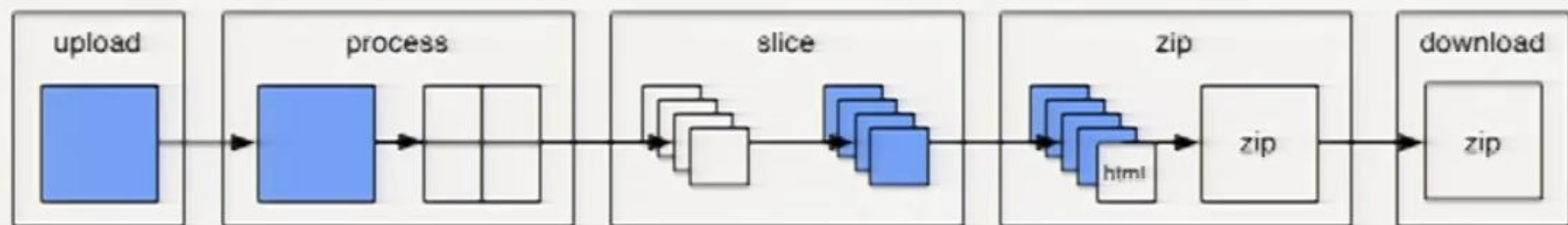
"We carry the pager so you don't have to."

Write code in Python, Java, or Go. (We're using Go.)

Parts of the stack used by this app:

- Datastore
- Blobstore
- Task Queues
- Backends
- Channels

App Overview



- Store the overlay image in Blobstore.
- Store the overlay metadata in Datastore.
- Slice the image into tiles, using
 - the `image`, `image/png`, and `image/jpeg` packages to read and write images,
 - the `graphics-go` package to do affine transforms, and
 - the Datastore to store tiles.
- Generate an `index.html` file. (`html/template` package)
- Store the tiles in a zip file and serve it to the user.
 - Write the directly to Blobstore with the `archive/zip` package.

App Overview: data structures

```
type Overlay struct {  
    Image appengine.BlobKey // Overlay image location.  
    Zip   appengine.BlobKey // Zip file location.  
  
    TopLeft    []float64 // Position of the overlay in world coordinates.  
    TopRight   []float64  
    BottomRight []float64  
  
    Tiles    int // Total number of Tiles to generate.  
}
```

GO

```
type Tile struct {  
    Image []byte  
    X, Y, Zoom int64  
}
```

GO



Design problems

(And solutions)



Problem: make it reliable

- This is a large task.
- Computer programs crash.
- It is slow and wasteful to get halfway through, crash, and then start again.

Make it reliable: Task Queues

The Task Queue API lets us schedule and run work with fault-tolerance.

Queues come in two flavors:

- Push queues start workers.
- Pull queues are consumed by workers.

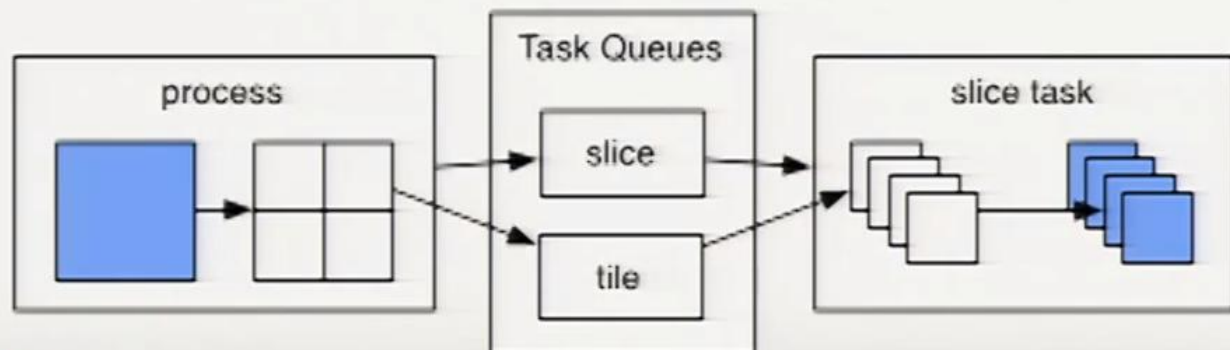
In Overlay Tiler:

- Push queues start the `slice` tasks on a backend.
- Pull queues track the `tile` tasks to be processed.

Make it reliable: Backends

Backends are high powered app instances that have no request deadlines.

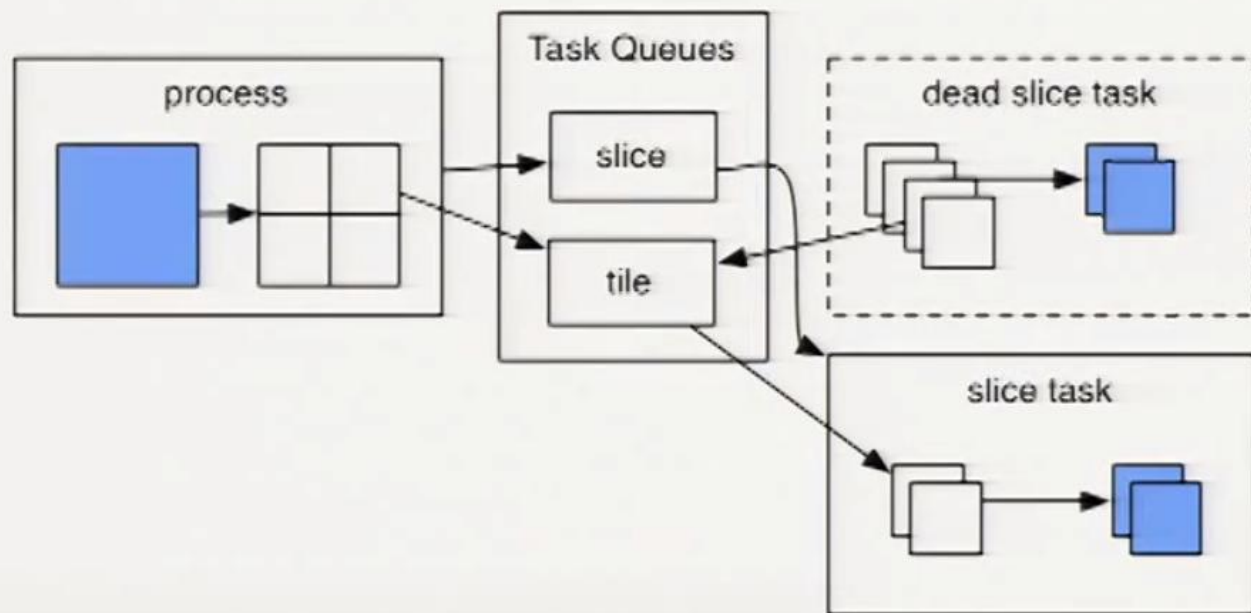
- The slicer backend runs `slice` tasks.
- `slice` tasks are triggered by a push queue.
- The `slice` task:
 - leases `tile` tasks from a pull queue,
 - stores each rendered `Tile` in the datastore,
 - removes the `tile` task from the queue.



Make it reliable: in case of failure

If a slice task fails:

- the tile tasks are returned to the tile queue,
- the slice task is restarted by the slice queue.



Problem: make it fast

- This is a large task.
- Computing each tile in series is unnecessary.

Make it fast: parallelism!

Get more throughput by running multiple `slice` tasks on multiple backend instances.

Our Task Queue-based design makes this trivial.

Increase the number of backends:

```
- name: slicer  
  class: B8  
  options: dynamic  
  instances: 4
```

BACKENDS.YAML

Increase the queue's concurrent request limit:

```
- name: slice  
  rate: 1/s  
  max_concurrent_requests: 4
```

QUEUE.YAML

Make it fast: parallelism!

And create one `slice` task for each backend:

```
task := taskqueue.NewPOSTTask("/slice", url.Values{"key": {k.Encode()}})
for i := 0; i < sliceBackends; i++ {
    host := appengine.BackendHostname(c, sliceBackend, i)
    task.Header.Set("Host", host)
    if _, err := taskqueue.Add(c, task, sliceQueue); err != nil {
        return err
    }
}
```

GO

Make it fast: concurrency!

The two expensive parts of the slice process are the tile computation and the datastore put.

Why not do them simultaneously?

Go's concurrency primitives make this trivial.

Make it fast: without concurrency

```
for {  
    // Lease tasks  
    // Render tiles  
    // Store tiles in datastore  
    // Delete completed tasks  
}
```

GO

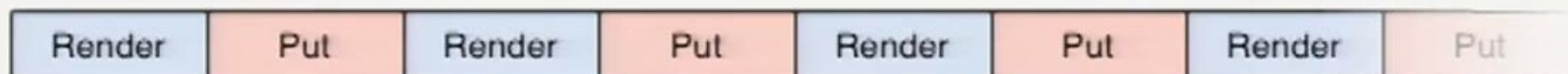
Make it fast: with concurrency

```
errc := make(chan error)
for {
    // Lease tasks
    // Render tiles
    go func() {
        // Store tiles in datastore
        // Delete completed tasks
        errc <- err
    }()
    errs++
}
for i := 0; i < errs; i++ {
    if err := <-errc; err != nil {
        return err
    }
}
```

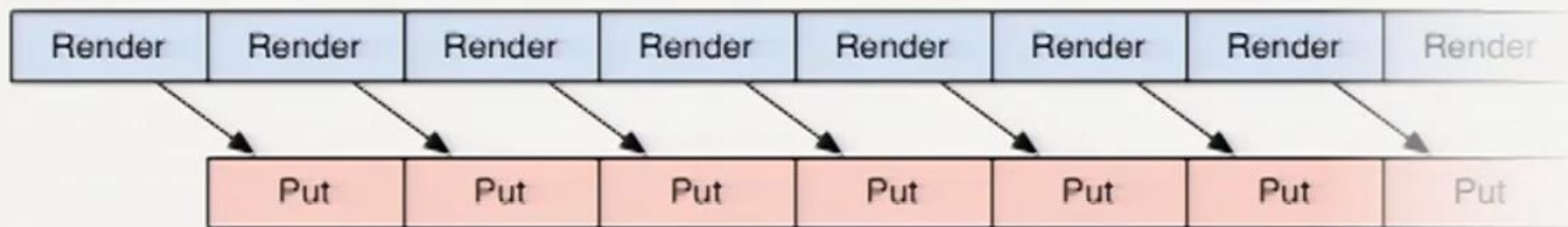
GO

Making it fast: concurrency

Without concurrency



With concurrency



Time

Problem: knowing when we're done

When all the tiles are generated we must create the zip file.

But with multiple tasks generating tiles, how can we

- know when the tiles are done, and
- create the zip only once?

Knowing when we're done: state

Bad places to keep state on App Engine:

- App instances (may die)
- Memcache (may expire)
- Client (may be broken, malicious, or go away at any time)

A good place to keep state:

- Datastore (consistent, structured, reliable)

And we're already using it!

Knowing when we're done: the algorithm

- Count tiles in datastore, compare against total number of tiles.
 - If not all tiles have been generated, keep slicing.
- Check the ZipRunning value in the Overlay record.
 - If true, shut down.
- Add a zip task to the zip push queue.
- Set ZipRunning to true the Overlay record.

Problem: talk to the user

- Must tell the user when slice and zip tasks are done.
- Can't just return an HTTP response from a backend.
- This is a large task; must show progress.

Talk to the user: Channels

The Channel API provides a push channel from App instances to the user's browser.

Creating a channel:

```
token, err := channel.Create(c, clientID)
// Send token to client.
```

GO

Sending a message:

```
err := channel.Send(c, clientID, "Hello!")
```

GO

Receiving a message:

```
var sock = new goog.appengine.Channel(token).open();
sock.onmessage = function(message) { console.log(message); }
```

JS

Talk to the user: sending messages

```
type Message struct {  
    Total    int    // Number of tiles in overlay  
    IDs      []string // Recently processed tiles  
    TilesDone bool  
    ZipDone  bool  
}
```

GO

```
ids := make([]string, len(tiles))  
for i, t := range tiles {  
    ids[i] = t.String()  
}  
channel.SendJSON(c, clientID, Message{Total: o.Tiles, IDs: ids})
```

GO

```
channel.SendJSON(c, clientID, Message{TilesDone: true})
```

GO

```
channel.SendJSON(c, clientID, Message{ZipDone: true})
```

GO

Talk to the user: receiving messages

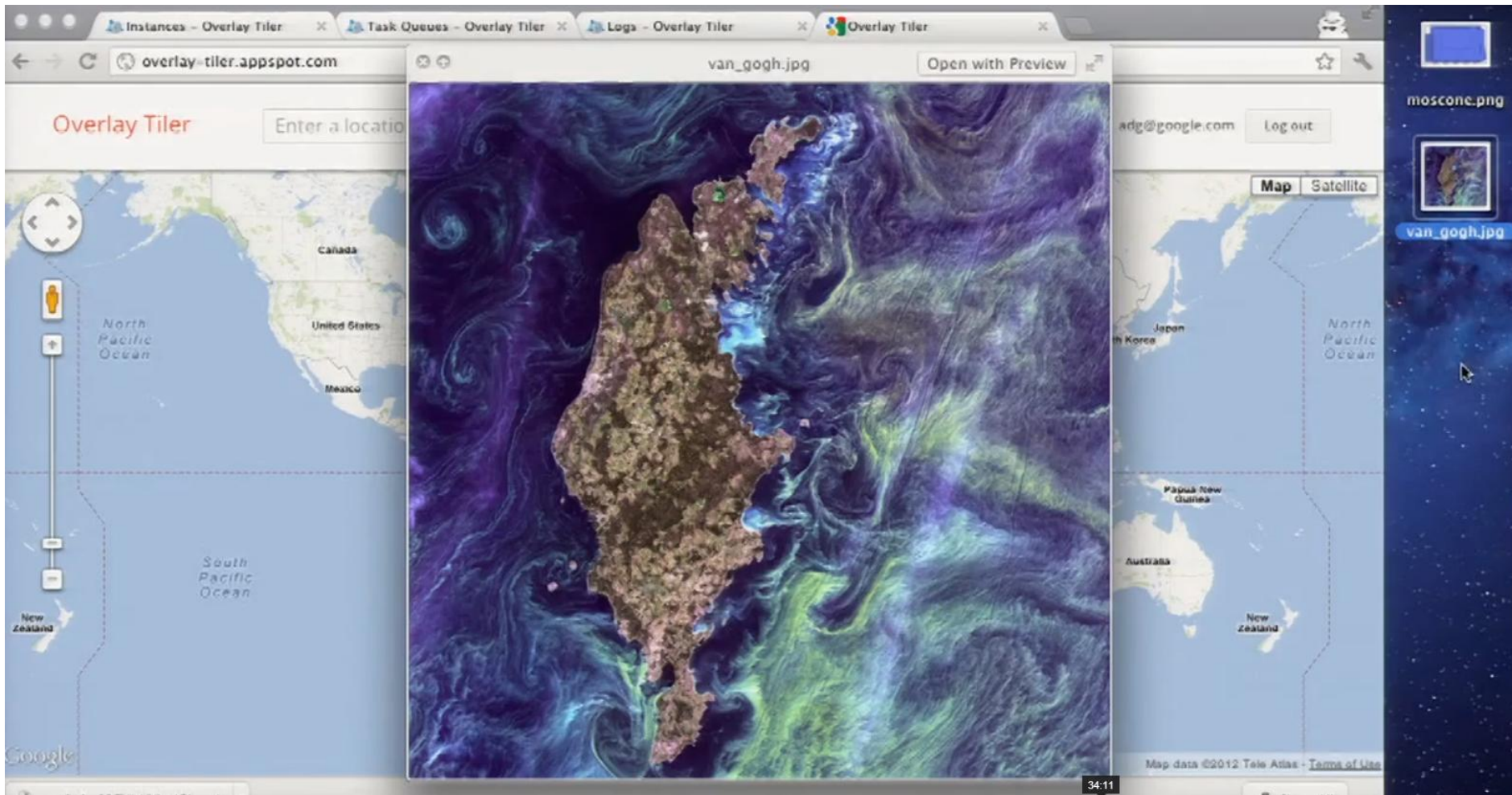
JS

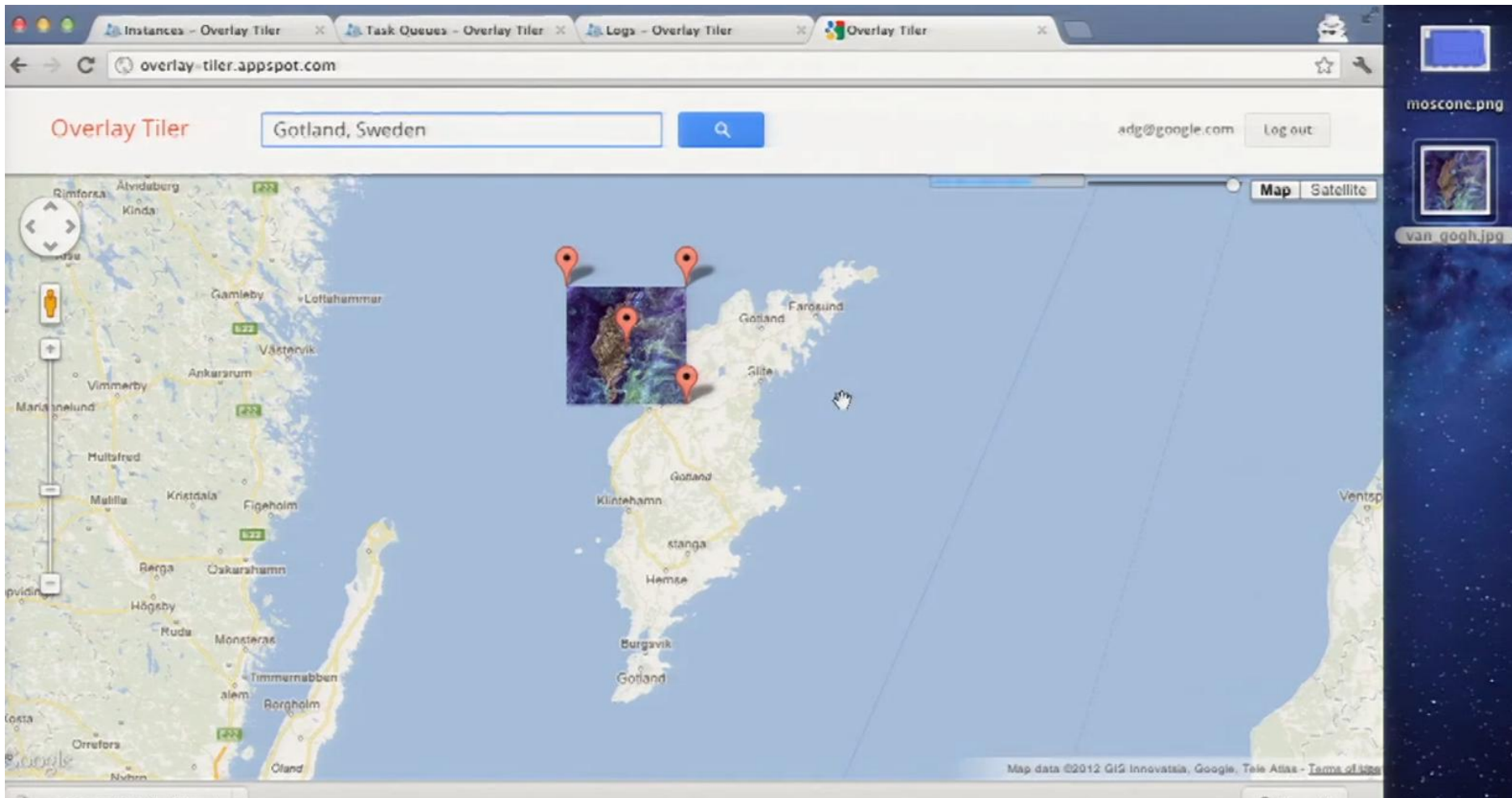
```
var tiles = {};  
var sock = new goog.appengine.Channel(token).open();  
sock.onmessage = function(msg) {  
  var d = JSON.parse(msg.data);  
  if (d.ZipDone) {  
    setStatus('Downloading ZIP archive');  
    window.location = '/download?key=' + key;  
  } else if (d.TilesDone) {  
    setStatus('Creating ZIP archive');  
  } else if (d.IDs) {  
    for (var i = 0, id; id = d.IDs[i]; i++) tiles[id] = true;  
    var count = 0;  
    for (var i in tiles) count++;  
    var pc = Math.floor(count / d.Total * 100);  
    setStatus('Generating tiles: ' + pc + '% complete');  
  }  
};
```

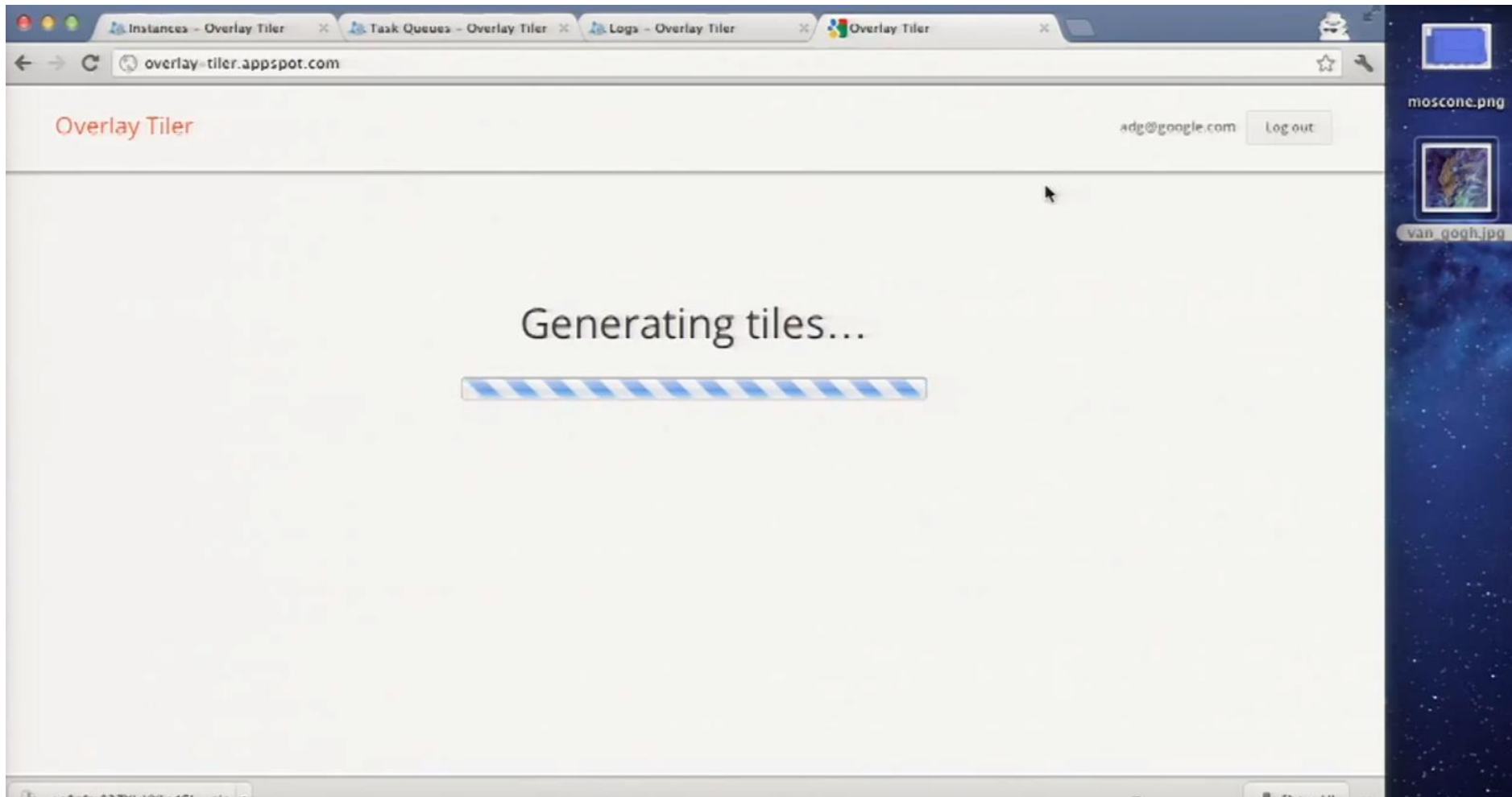


Final demo









Instances - Overlay Tiler | Task Queues - Overlay Tiler | Logs - Overlay Tiler | Overlay Tiler

https://appengine.google.com/instances?app_id=s~overlay-tiler&version_id=slicer.359933465706688569

Google app engine [adg@google.com](#) | [My Account](#) | [Help](#) | [Sign out](#)

Application: **overlay-tiler [High Replication]** | [« All google.com Applications](#) | [Report Production Issue](#) | [« My Applications](#)

Version: **slicer**

Main

- [Dashboard](#)
- [Instances](#)**
- [Logs](#)
- [Versions](#)
- [Backends](#)
- [Cron Jobs](#)
- [Task Queues](#)
- [Quota Details](#)
- [Endpoints](#)

Data

- [Datastore Indexes](#)
- [Datastore Viewer](#)
- [Datastore Statistics](#)
- [Blob Viewer](#)
- [Prospective Search](#)
- [Text Search](#)
- [Datastore Admin](#)
- [Memcache Viewer](#)

Total number of instances		Average QPS*		Average Latency*		Average Memory	
4 total		0.058		1861.1 ms		24.8 MBytes	

Instances [▼](#)

Index	QPS*	Latency*	Requests	Errors	Age	CPU Rate	Memory	Logs	Availability	Shutdown
0	0.050	788.0 ms	2	0	0:00:06	1.87 (0%)	23.5 MBytes	View Logs	Dynamic	Shutdown
1	0.050	354.0 ms	2	0	0:00:07	1.40 (0%)	27.9 MBytes	View Logs	Dynamic	Shutdown
2	0.067	2575.5 ms	1	1	0:00:07	114.33 (2%)	22.8 MBytes	View Logs	Dynamic	Shutdown
3	0.067	3082.0 ms	1	1	0:00:07	188.53 (4%)	24.9 MBytes	View Logs	Dynamic	Shutdown

* QPS and latency values are an average over the last minute.

Waiting for appengine.google.com...

ag9zfm92ZXJsYXktdGl...zip

Show All

moscone.png

van gogh.jpg

Instances - Overlay Tiler Task Queues - Overlay Tiler Logs - Overlay Tiler Overlay Tiler

https://appengine.google.com/queues?app_id=s~overlay-tiler&version_id=2~dev.359946617424040081

Google app engine adg@google.com | My Account | Help | Sign out

Application: overlay-tiler [High Replication] « All google.com Applications | Report Production Issue « My Applications

Main

- [Dashboard](#)
- [Instances](#)
- [Logs](#)
- [Versions](#)
- [Backends](#)
- [Cron Jobs](#)
- [Task Queues](#)
- [Quota Details](#)
- [Endpoints](#)

Data

- [Datastore Indexes](#)
- [Datastore Viewer](#)
- [Datastore Statistics](#)
- [Blob Viewer](#)
- [Prospective Search](#)
- [Text Search](#)
- [Datastore Admin](#)
- [Memcache Viewer](#)

Administration

Task queues are defined in a `queue.yaml` (Python and Go) or `queue.xml` (Java). [Learn more about task queues.](#)

Tasks Daily Quota

Task Queue API Calls	0%	488 of 1,199,900,000
----------------------	----	----------------------

Tasks Storage Quota

Task Queue Stored Task Count	0%	1,291 of 10,999,900,000
Task Queue Stored Task Bytes		0.00 of Unlimited GBytes

Note: If your application exceeds the task queue storage quotas, consider increasing Stored Task Bytes Quota, increasing execution rates of queues, or reducing the rate at which tasks are added to queues. Purging a queue does not immediately reclaim its quota. [Learn more.](#)

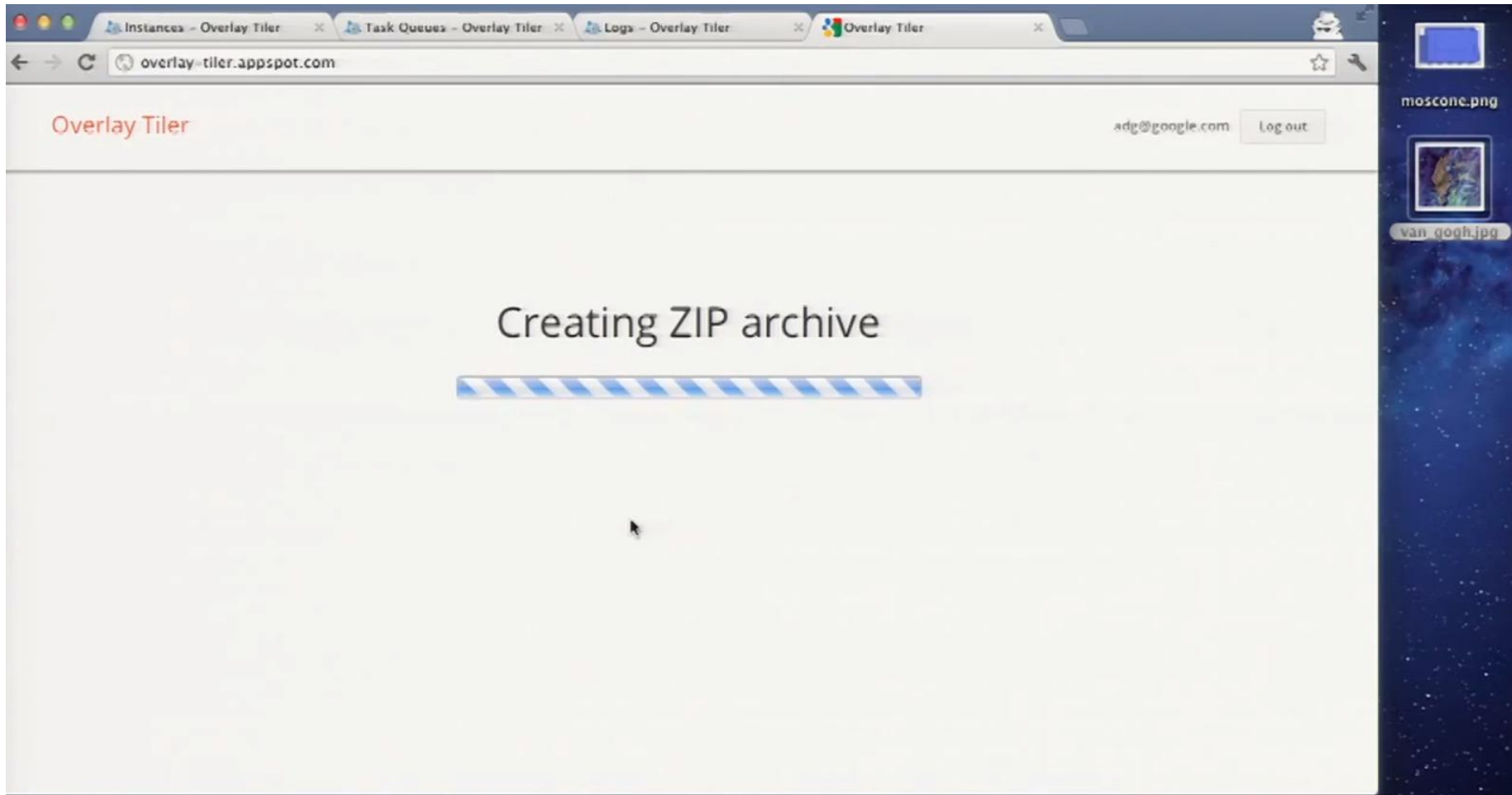
Push Queues

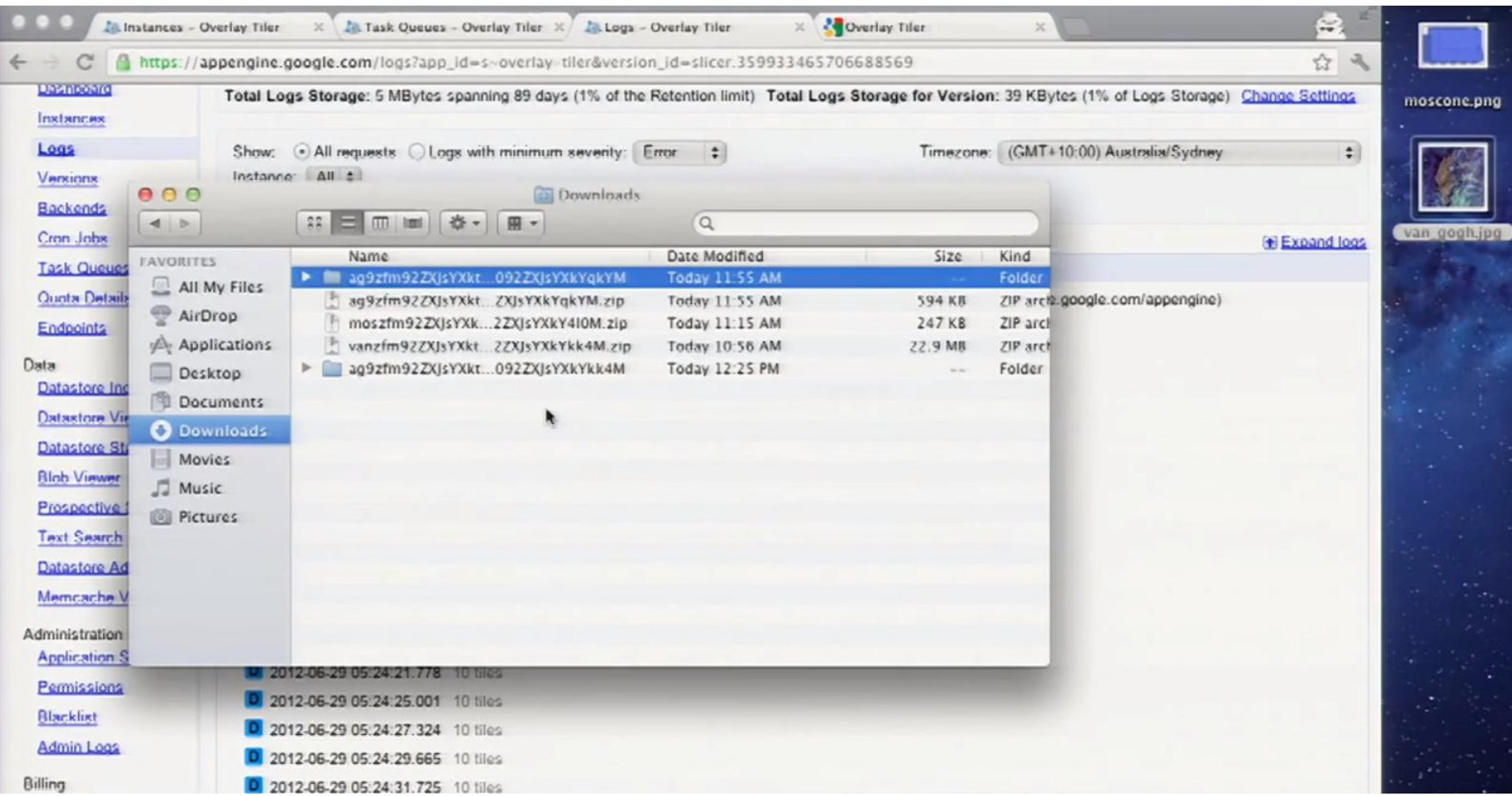
Queue Name	Maximum Rate	Enforced Rate	Bucket Size	Maximum Concurrent	Oldest Task	Tasks in Queue	Run in Last Minute	Running
default	50/s	50.00/s	5.0			0	0	0
zend	100/s	100.00/s	5.0			0	14	0
alice	1/s	1.00/s	5.0	4	2012-06-28 12:23:31 (0:00:14 ago)	4	2	2
zip	1/s	1.00/s	5.0	1		0	0	0

Pull Queues

Queue Name	Oldest Task	Tasks in Queue	Leased in Last Minute
tile	2012-06-28 10:54:03 (1:29:42 ago)	1,108	180







Instances - Overlay Tiler

Task Queues - Overlay Tiler

Logs - Overlay Tiler

Overlay Tiler

https://appengine.google.com/logs?app_id=s-overlay-tiler&version_id=slicer.359933465706688569

Instances

Logs

Versions

Backends

Cron Jobs

Task Queues

Queue Details

Endpoints

Data

Datastore Indexes

Datastore Viewer

Datastore Statistics

Index Viewer

Prospective Search

Text Search

Datastore Admin

Memcache Viewer

Administration

Application Settings

Permissions

Blacklist

Admin Logs

Billing

Total Logs Storage: 5 MBytes spanning 89 days (1% of the Retention limit) Total Logs Storage for Version: 39 KBytes (1% of Logs Storage) [Change Settings](#)

Show:

All requests

 Logs Instance:

All

[Options](#)

Tip: Click a log line to show or hide

Prev 20

1-20

Next 20

(Top: 0:0)

2012-06-29 05:24:33.319 /slice

2012-06-29 05:23:59.412 10

2012-06-29 05:24:01.419 10

2012-06-29 05:24:02.395 10 tiles

2012-06-29 05:24:02.780 10 tiles

2012-06-29 05:24:03.200 10 tiles

2012-06-29 05:24:05.078 10 tiles

2012-06-29 05:24:06.564 10 tiles

2012-06-29 05:24:09.015 10 tiles

2012-06-29 05:24:11.247 10 tiles

2012-06-29 05:24:13.528 10 tiles

2012-06-29 05:24:17.788 10 tiles

2012-06-29 05:24:19.913 10 tiles

2012-06-29 05:24:21.778 10 tiles

2012-06-29 05:24:25.001 10 tiles

2012-06-29 05:24:27.324 10 tiles

2012-06-29 05:24:29.665 10 tiles

2012-06-29 05:24:31.725 10 tiles

Expand logs

"Index.html" is a web application downloaded from the Internet. Are you sure you want to open it?

Google Chrome downloaded this file today at 10:56 AM from overlay-tiler.appspot.com.

?

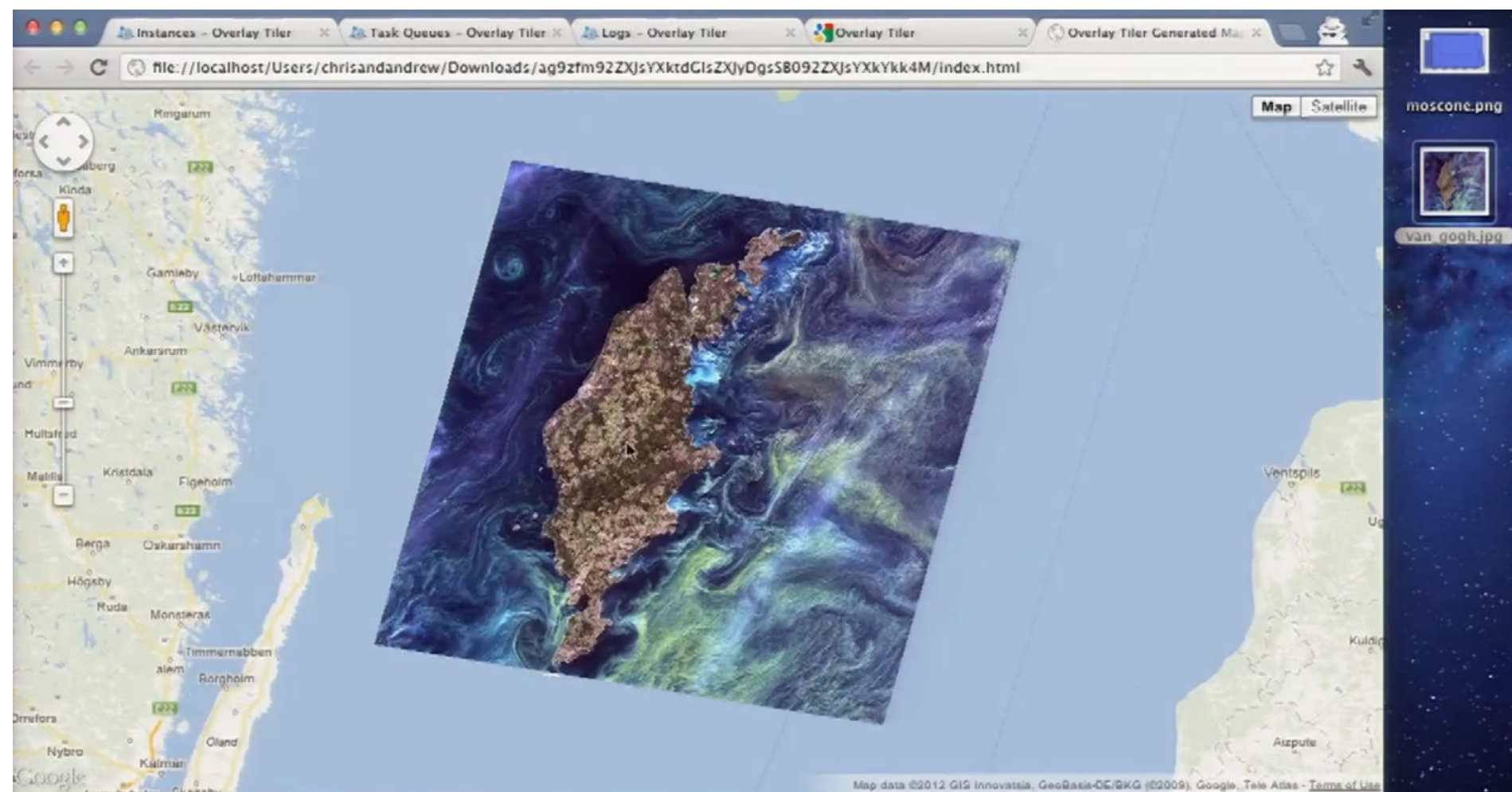
Show Web Page

Cancel

Open

moscone.png

van_gogh.jpg





Final demo



Go's strengths

- Great standard packages
 - `archive/zip`, `encoding/json`, `html/template`, `image`
- Great external packages
 - `code.google.com/p/graphics-go`
 - See go.pkgdoc.org for more
- High performance
 - Only App Engine runtime that compiles user code to machine code
- Great language
 - Simple, consistent, comprehensible
 - Useful concurrency primitives

Learn More

Overlay Tiler is available under the Apache 2.0 License:

code.google.com/p/overlay-tiler

Maps API:

developers.google.com/maps

App Engine:

developers.google.com/appengine

Go: (including an interactive tutorial)

golang.org

Thank You!



Chris Broadfoot - Google Sydney
Andrew Gerrand - Google Sydney