





# A scalable system for processing the historical Landsat satellite imagery record

Mike Hiley  
Earthling Interactive  
Big Data Wisconsin 2017

# Outline

- Introduce myself
- Intro to Landsat
- Land remote sensing 101
- Motivation for building this system
- Intro to geospatial tools
- Design of scalable cloud-based processing system
- Unexpected challenges
- Conclusion

# Hi everyone!



- My academic background is in satellite meteorology
- For masters research, I studied snowfall using a satellite-based cloud radar (Cloudsat)
- Worked at UW Space Science & Engineering for several years with a group doing satellite-based cloud climatologies
- Started at Earthling in April 2016. Several things were new to me:
  - AWS and cloud processing
  - Land remote sensing (clouds are now annoying artifacts as opposed to object to study)
  - Proprietary algorithms (have to be careful what I say)

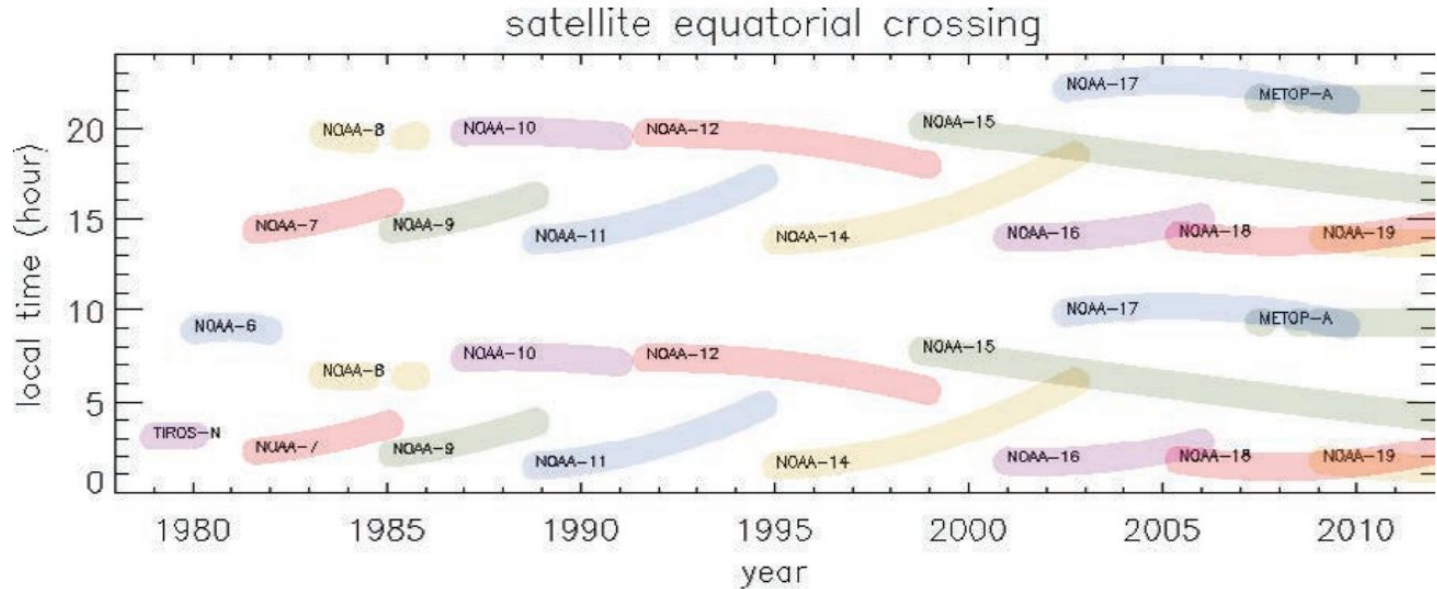
# Previous work: cloud climatology using on-site compute cluster

- For a couple years prior to previous work, I worked for Andy Heidinger at the UW-Madison Cooperative Institute for Meteorological Satellite Studies (CIMSS) on a satellite-derived cloud\* climatology (PATMOS-x) dating back to 1979
- All processing was done with on-site hardware! No cloud\*\* involved

\*meteorological clouds

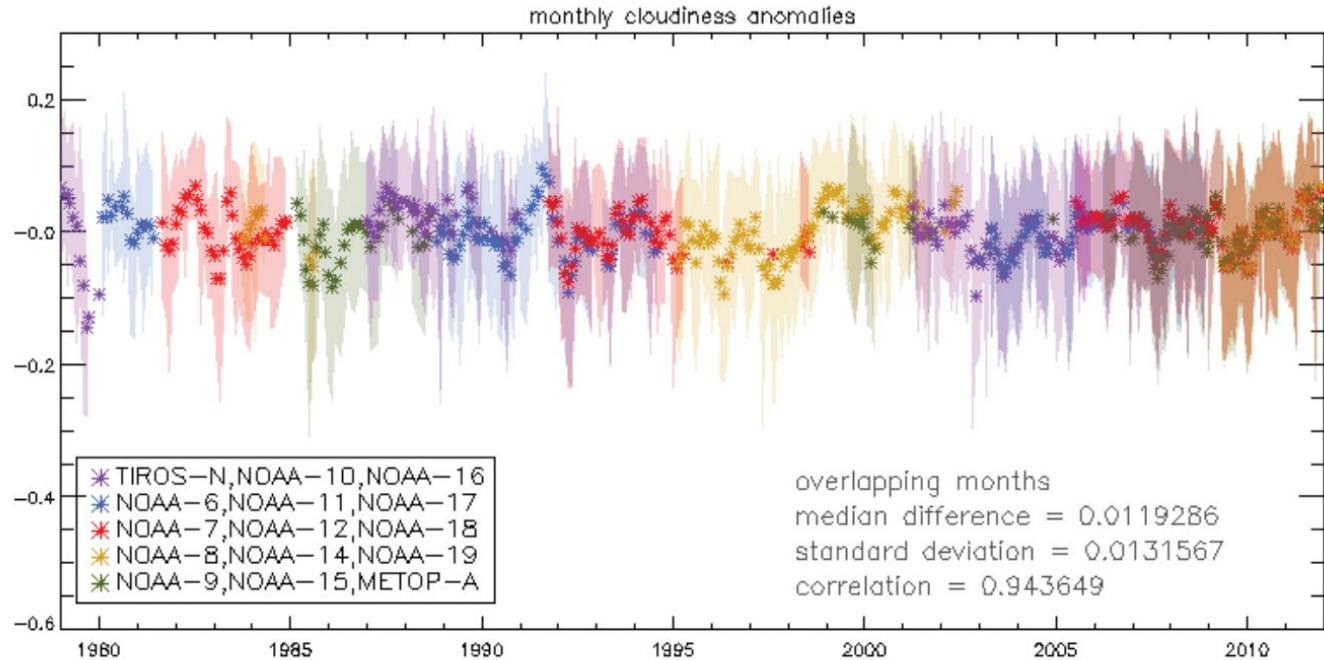
\*\*cloud computing

# PATMOS-x: how to get coherent time series of global cloudiness from 15 poorly calibrated satellites



**FIG. 1. Equatorial crossing time of the NOAA and MetOp polar-orbiting satellite series spanning 1979–2012.**

# They make it work!



**FIG. 8. Time series of AVHRR PATMOS-x monthly cloudiness anomalies over the North Pacific. Shading represents uncertainty estimates, calculated using**

# All processing on-site! No Amazon involved!

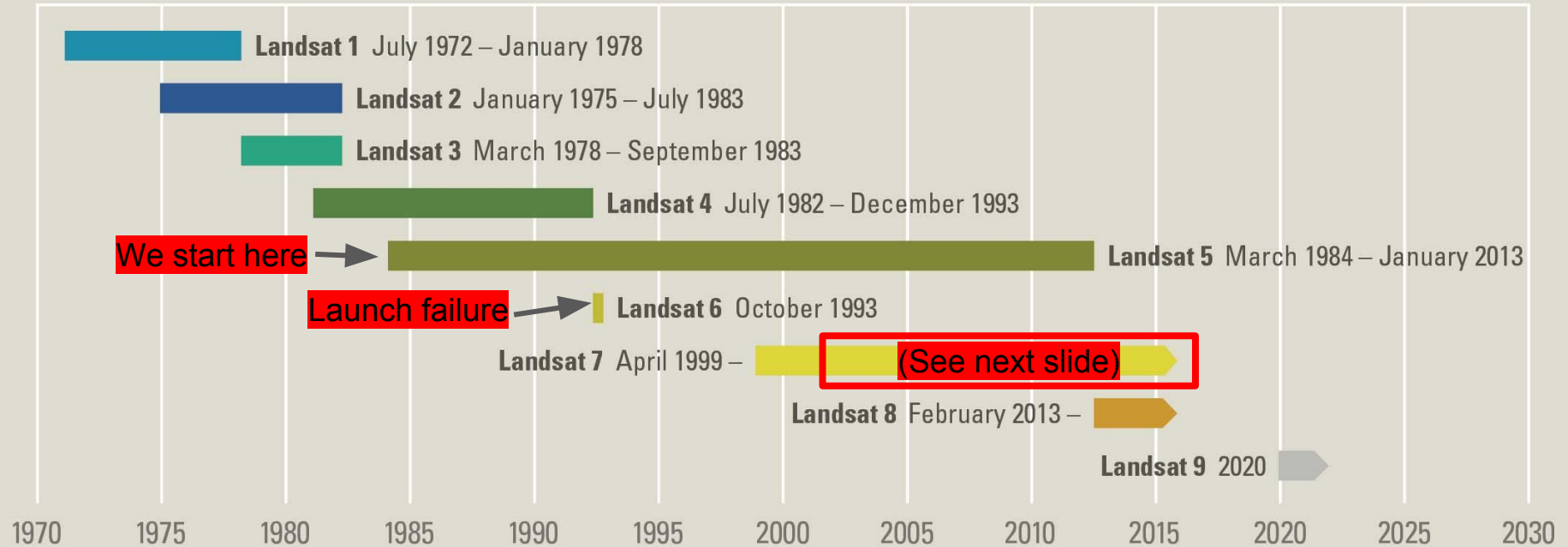
- The “Slurm” scheduler was used for managing scheduling of compute jobs in cluster with tens of compute nodes and hundreds of cores
- “Lustre” was used to manage a **multi-petabyte** storage array
- All resources located in the Atmospheric Oceanic & Space Sciences building on UW campus
- Creating entire dataset (almost 40 years of data) was a multi-week project involving significant manual effort (and lots of Perl scripts)
- My job was to manage the processing and write Python analysis code to perform quality control on the final product



# Fast forward to April 2016

- I get hired by Earthling, initially to work on a rewrite of an existing system for processing Landsat imagery into vegetation products for use in precision agriculture
- The experience was a boot camp in:
  - Land/agriculture remote sensing
  - Amazon Web Services

# Intro to Landsat - program history

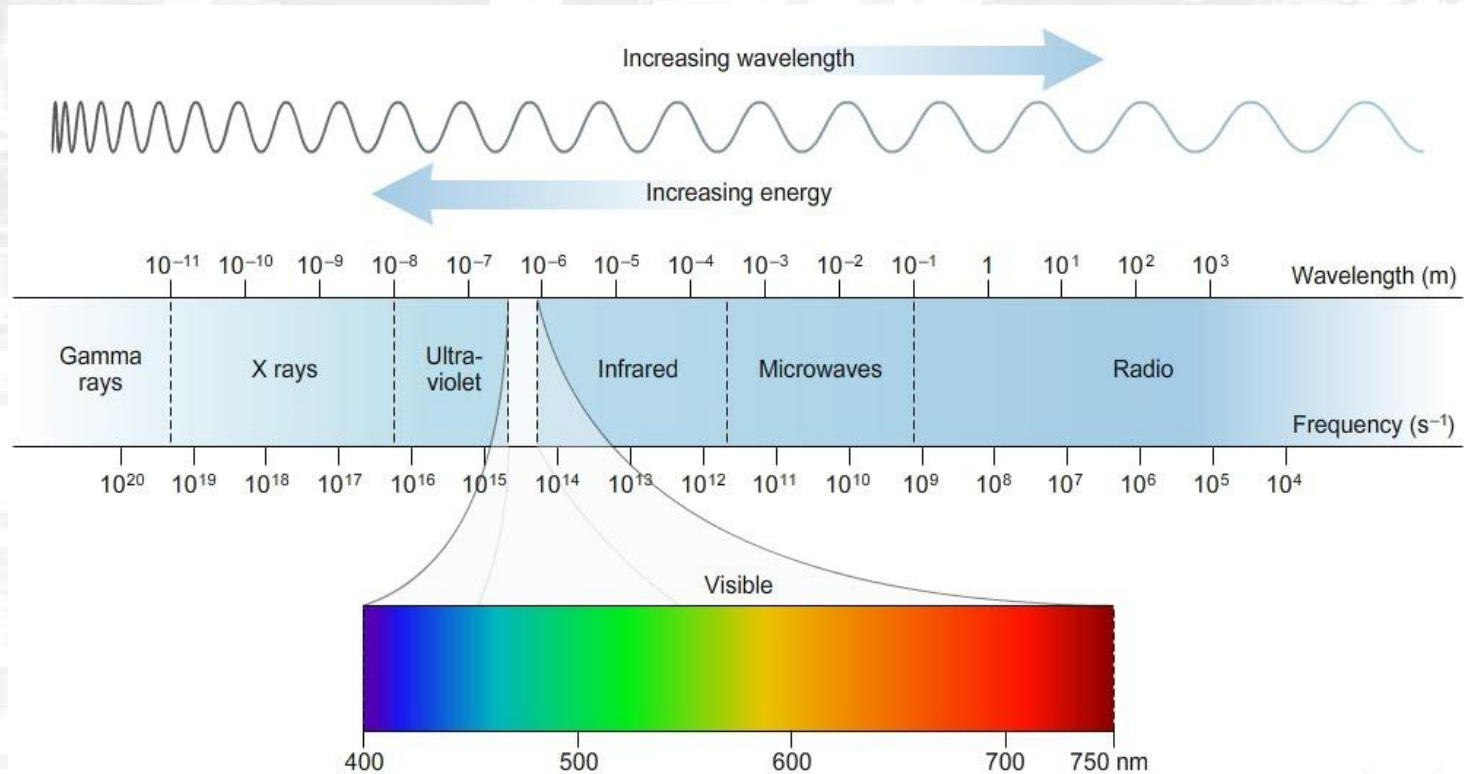


# Intro to Landsat - Landsat 7 failure

- In May 2003, the Landsat 7 Scan Line Corrector (SLC) failed permanently, resulting in these large gaps through much of the imagery
- Makes data largely unusable if interested in spatial patterns within a field



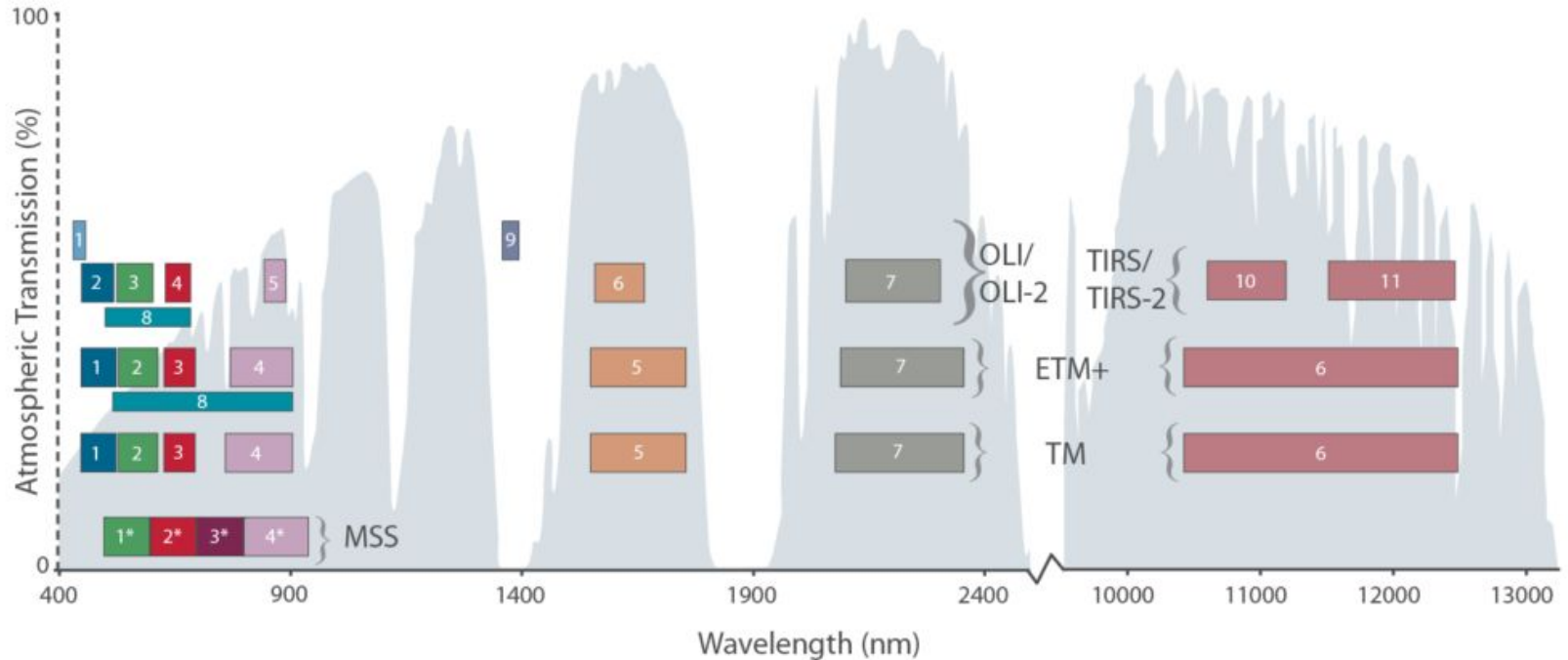
# Physics reminder: the electromagnetic spectrum



# Intro to Landsat - what does it observe?

<b>Landsat 8 Operational Land Imager (OLI) and Thermal Infrared Sensor (TIRS)</b>	<b>Bands</b>	<b>Wavelength (micrometers)</b>	<b>Resolution (meters)</b>
	Band 1 - Ultra Blue (coastal/aerosol)	0.435 - 0.451	30
	Band 2 - Blue	0.452 - 0.512	30
	Band 3 - Green	0.533 - 0.590	30
	Band 4 - Red	0.636 - 0.673	30
	Band 5 - Near Infrared (NIR)	0.851 - 0.879	30
	Band 6 - Shortwave Infrared (SWIR) 1	1.566 - 1.651	30
	Band 7 - Shortwave Infrared (SWIR) 2	2.107 - 2.294	30
	Band 8 - Panchromatic	0.503 - 0.676	15
	Band 9 - Cirrus	1.363 - 1.384	30
	Band 10 - Thermal Infrared (TIRS) 1	10.60 - 11.19	100 * (30)
	Band 11 - Thermal Infrared (TIRS) 2	11.50 - 12.51	100 * (30)

# Intro to Landsat - what does it observe?





Intro to Landsat - can make really nice pictures

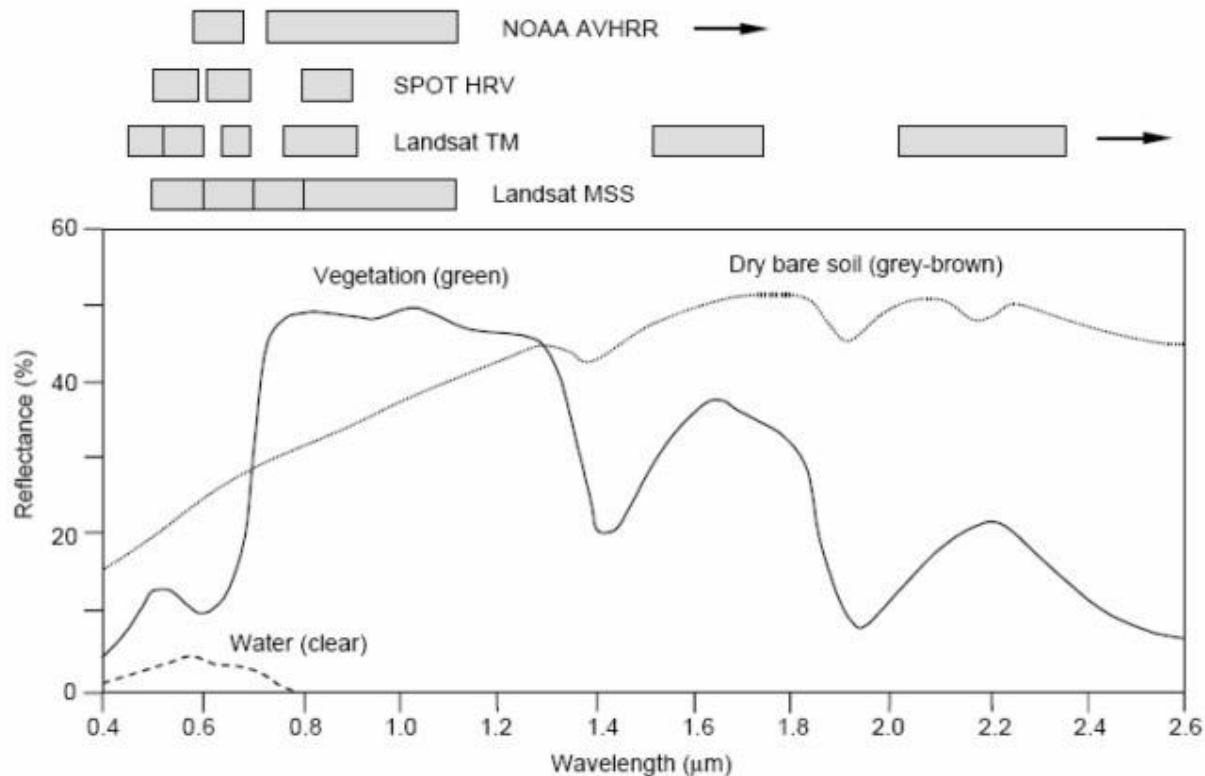


# Intro to Landsat - but, how to turn raw observations into useful products?

Landsat observations tell us how much energy the earth is reflecting across a wide variety of wavelengths. How is that useful for agriculture?



# Intro to land remote sensing - theoretical basis for vegetation indices



# Intro to land remote sensing - Normalized Difference Vegetation Index (NDVI)

Using what we know about spectral properties of vegetation, we can combine bands to learn whether a pixel contains vegetation (and how much).

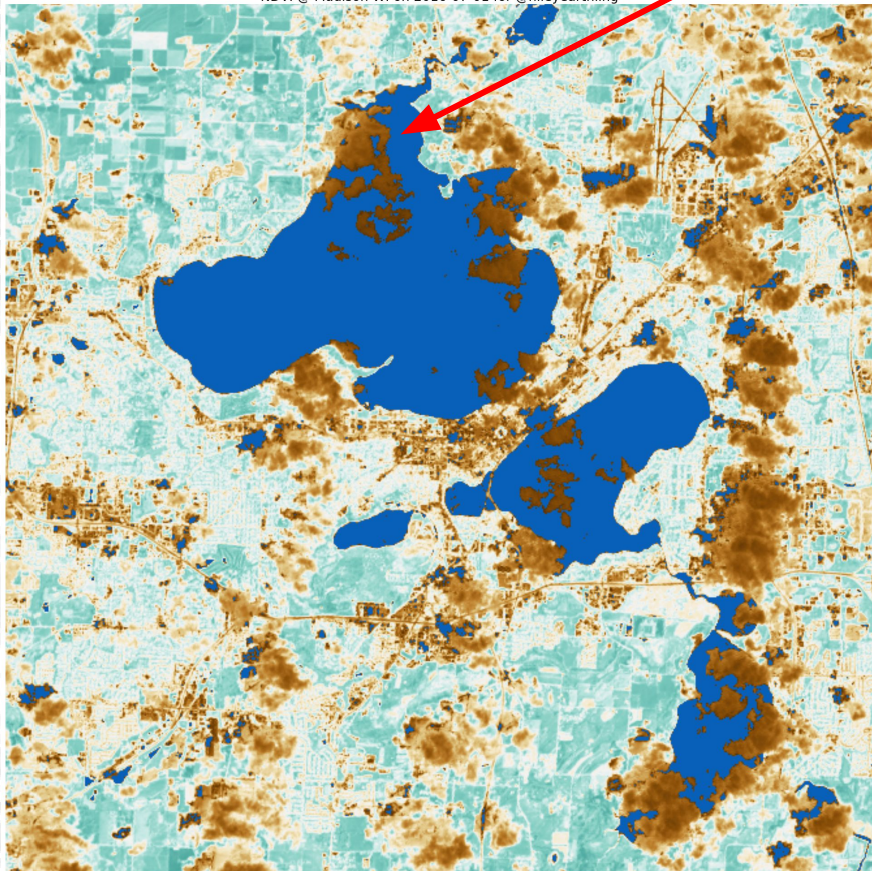
$$\text{NDVI} = \frac{(\text{NIR} - \text{Red})}{(\text{NIR} + \text{Red})}$$

# Intro to land remote sensing - NDVI properties

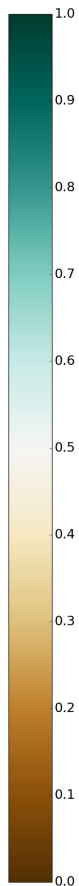
- Always within range -1 to 1
  - Water: close to zero or slightly negative
  - Clouds and snow: less than 0
  - Green vegetation: 0.3 to 1 (dense vegetation will be close to 1)
- As simple as it gets. More complex algorithms can measure specific properties like:
  - Leaf Area Index (spatial coverage of vegetation)
  - Chlorophyll concentration (greenness of vegetation)

# Intro to land remote sensing - NDVI example

NDVI @ Madison WI on 2016-07-01 for @hileyearthling



Cloud contamination!



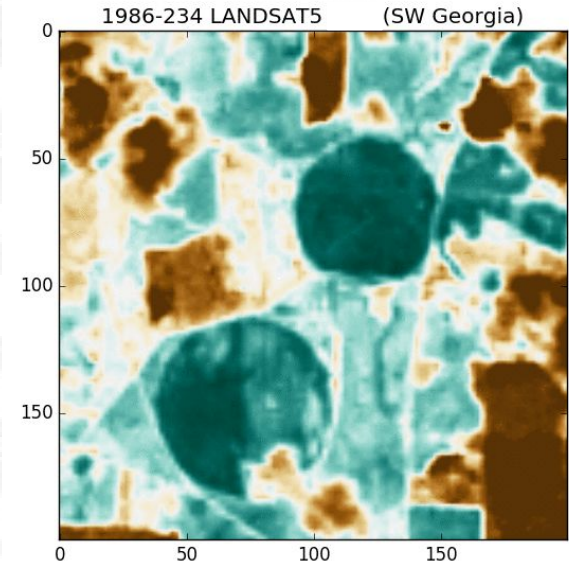
Tweet us:

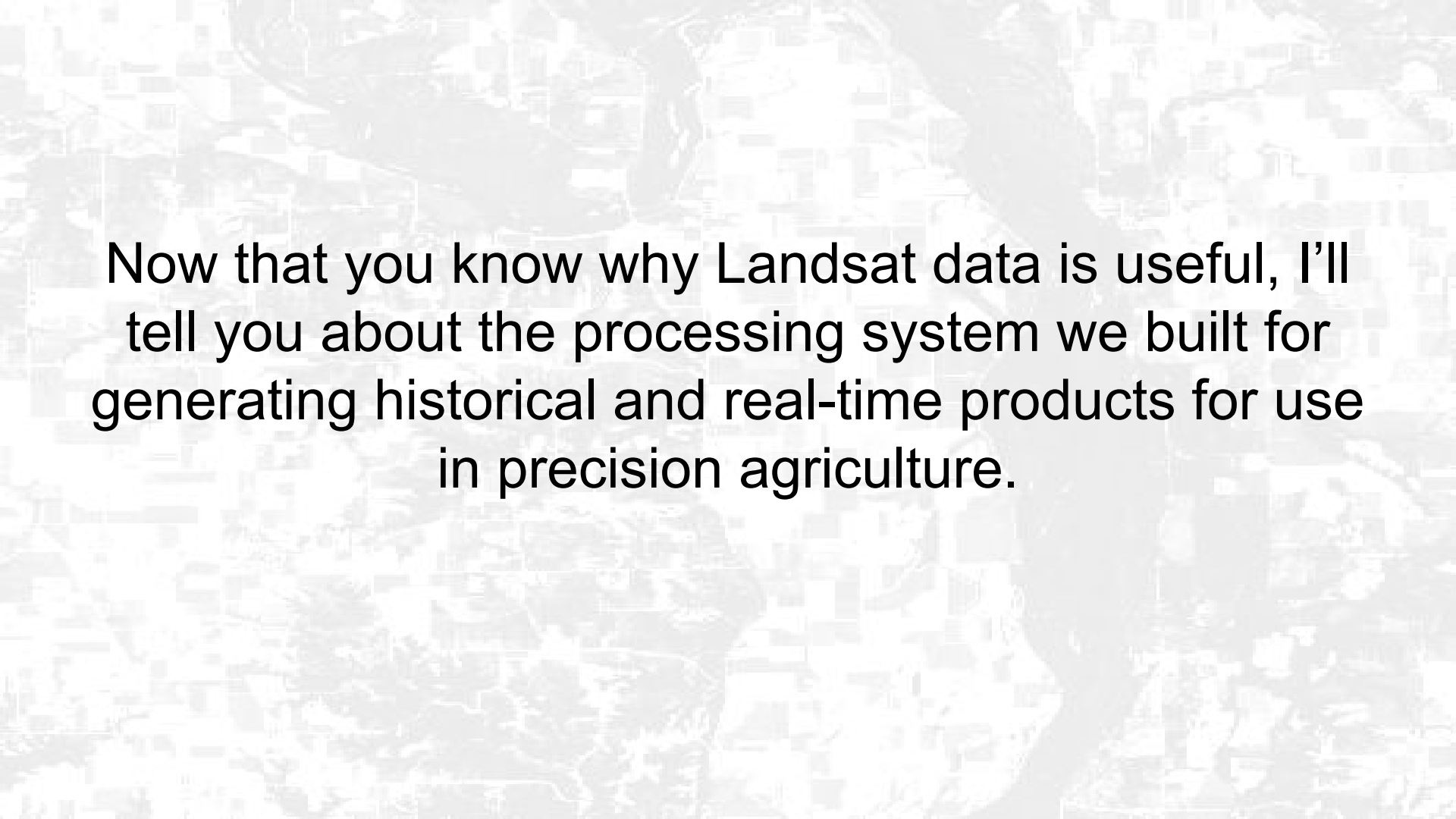
**@WeAreEarthling #agmap**  
**[any address or city**  
**and state]**

To get your own NDVI image!

# Intro to land remote sensing - utility in precision agriculture

- Historical performance: By observing vegetation products over a long period of time, a farmer can learn about the historical productivity (yield) of each of his/her fields
- Spatial variation within a field: Due to the high resolution of Landsat (30 meters), we can learn about spatial variation of performance within a single field, allowing the farmer to apply advanced management techniques like optimizing fertilizer inputs across the field, to achieve the greatest yield at the lowest cost



The background of the slide is a grayscale aerial photograph of a rural landscape. It shows a patchwork of agricultural fields, some of which are divided into smaller plots by thin lines, possibly roads or irrigation canals. The terrain appears to be flat, and the overall image has a high-contrast, slightly grainy quality typical of satellite or aerial photography.

Now that you know why Landsat data is useful, I'll tell you about the processing system we built for generating historical and real-time products for use in precision agriculture.

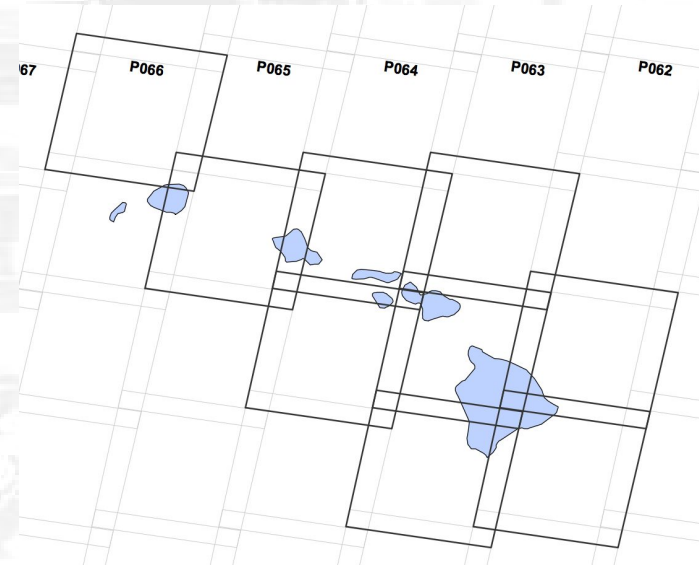
# Landsat is “big data”

Global coverage (every 16 days) at 30 meter resolution!

Imagery is split into “scenes”

Scene count to date:

- Landsat 5: 2,285,693
- Landsat 7: 2,143,313
- Landsat 8: 1,111,446
- Total: 5,540,452



Scene coverage over Hawaii  
to show scale of each scene  
footprint



# Landsat is “big data”

If you wanted to process all historical Landsat data globally:

$(5,540,452 \text{ scenes}) * (3 \text{ bands per scene}) * (60\text{MB per band}) =$

**951 terabytes !**

How to cope?

- Obvious first step: only process data over agricultural areas of interest during growing seasons
- Build a **scalable** processing system in the cloud



# Landsat data sources

- Amazon has all Landsat 8 data in a publicly accessible S3 bucket!
  - <https://aws.amazon.com/public-datasets/landsat/>
- Landsat 5 and Landsat 7 is available through a USGS API
  - Much slower than grabbing Landsat 8 via Amazon ... several minutes per scene

# Why scalable?

- Clearly have more processing to do than a single machine can accomplish in a reasonable amount of time, so need to scale out across multiple machines
- Resource requirements inherently change over time. Don't want to pay for more resources than are currently being utilized.
  - Initial processing of all historical data of interest needs to be accomplished in a short enough period of time to meet business requirements
  - Ongoing resource requirements will be much lower
    - Process new imagery as it is acquired by satellites
    - Process imagery for new agricultural areas as new customers are acquired

# Toolkit - Algorithm implementation

- Python
  - NumPy (fast vectorized array math)
  - Matplotlib (to visualize output during development)
  - Fiona and Shapely for geospatial geometry
  - Boto (Python wrapper of AWS API)
- GDAL (Geospatial Data Abstraction Library)
  - Resample bands with differing resolution
  - Crop imagery to some geometry of interest

```
# read data arrays from GeoTIFF files:  
near_IR = read_raster(near_IR_file)  
vis = read_raster(vis_file)  
# array operations are vectorized:  
ndvi = (near_IR - vis) / (near_IR + vis)  
water_mask = ndvi < 0
```



# Original implementation...

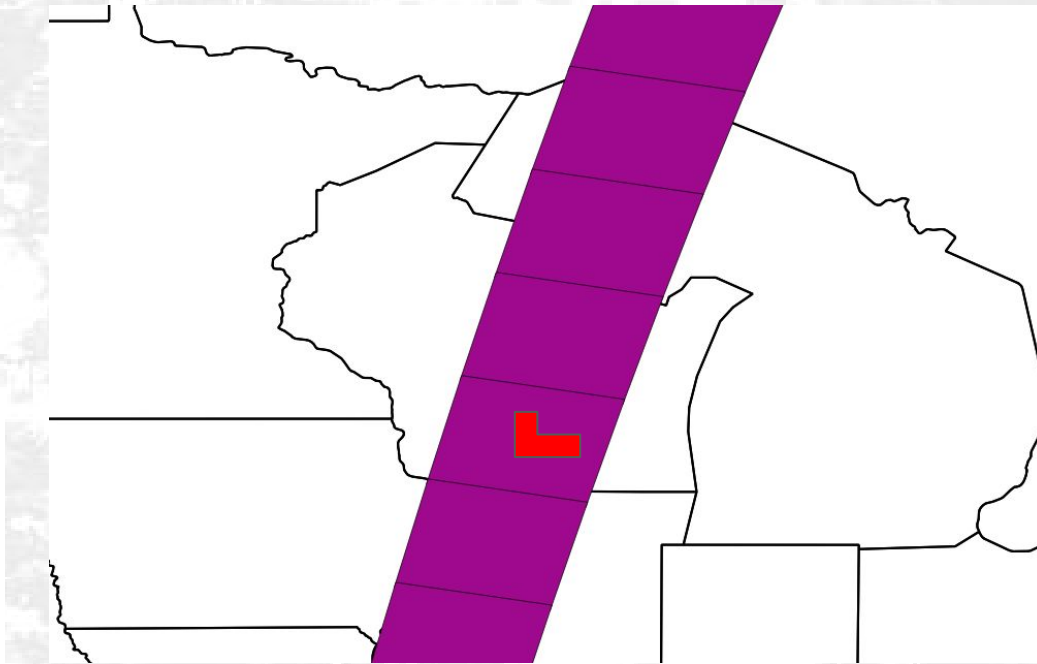
- C Sharp / .NET for data input/output and connections to AWS
- Proprietary object-based image analysis software for implementing algorithms themselves

## Why did we rewrite in Python?

- Windows instances are expensive! New system can run on cheaper Linux instances
- Python has become a top choice for all kinds of scientific analysis; great selection of open source libraries for geospatial processing
- NumPy allows fast pixel-based manipulation of raster data without additional image processing software

```
# read data arrays from GeoTIFF files:
near_IR = read_raster(near_IR_file)
vis = read_raster(vis_file)
# array operations are vectorized:
ndvi = (near_IR - vis) / (near_IR + vis)
water_mask = ndvi < 0
```

PostGIS operator example - given some coordinates  
how do you know what imagery to process?



# PostGIS operator example - database design

Given database tables in a PostGIS enabled Postgres database:

What we have →

<u>fields</u>
field_geometry (polygons)

<u>scene_footprints</u>
footprint_geometry (polygons)
path
row

What we want →

<u>landsat_scenes</u>
scene_id
path
row



# PostGIS operator example - the actual query

**ST\_Contains**: check if some geometry lies completely within another geometry.

```
SELECT
  scenes.scene_id
FROM landsat_scenes scenes
INNER JOIN scene_footprints footprints
  ON scenes.path = footprints.path
  AND scenes.row = footprints.row
WHERE ST_Contains(footprints.geometry, field_geometry);
```

PostgreSQL



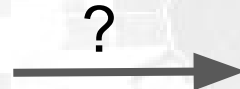
PostGIS



Spatial PostgreSQL



GDAL example: how do you clip an image to some arbitrary polygon?





# Solution: use “gdalwarp”

```
gdalwarp
```

```
-cutline polygon.geojson
```

```
-crop_to_cutline
```

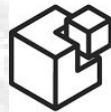
```
LC80410262013181LGN00_B2.TIF
```

```
cropped_result.TIF
```



# Toolkit - AWS/cloud

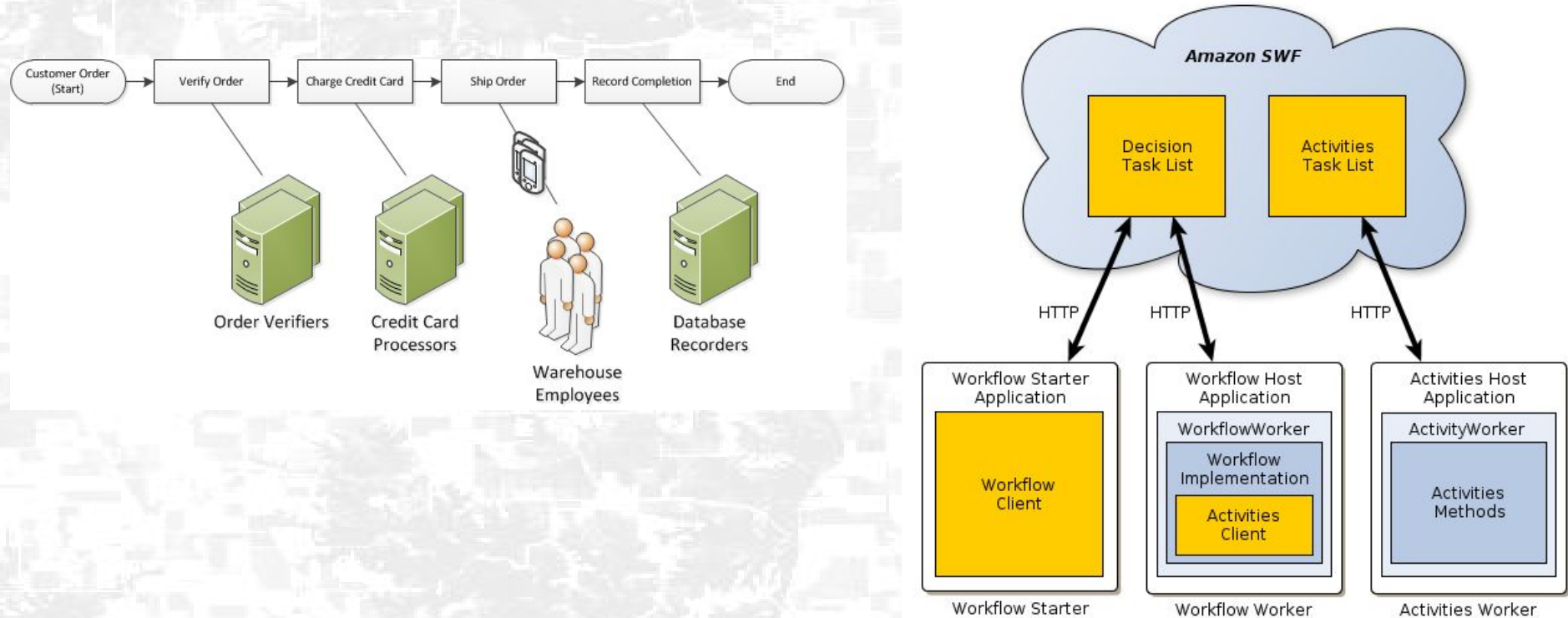
- EC2 compute instances running Ubuntu 14.04, using spot pricing to minimize cost
- SaltStack for provisioning new instances as they come online
- Simple Workflow Service (SWF) for managing tasks across multiple compute instances
- Simple Storage Service (S3) for storage of algorithm results
- Postgres database for tracking state of processing jobs
  - with PostGIS for spatial queries



**SALTSTACK**

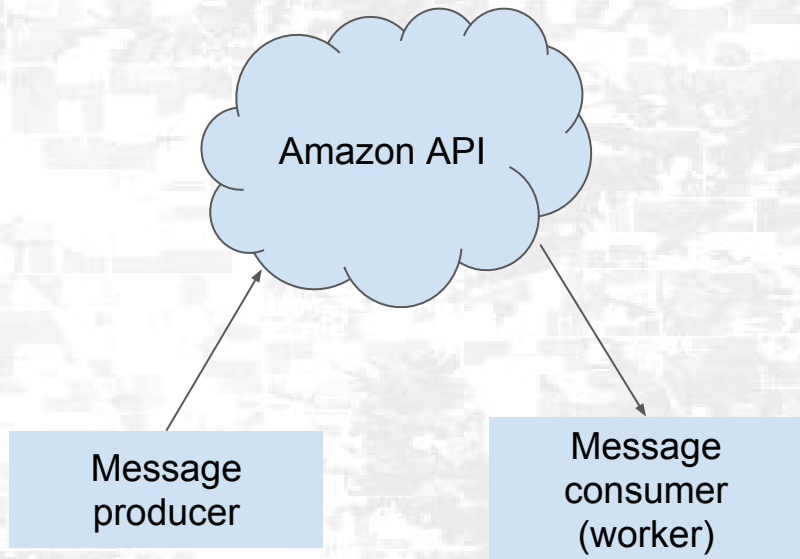


# Simple Workflow Service (SWF) overview

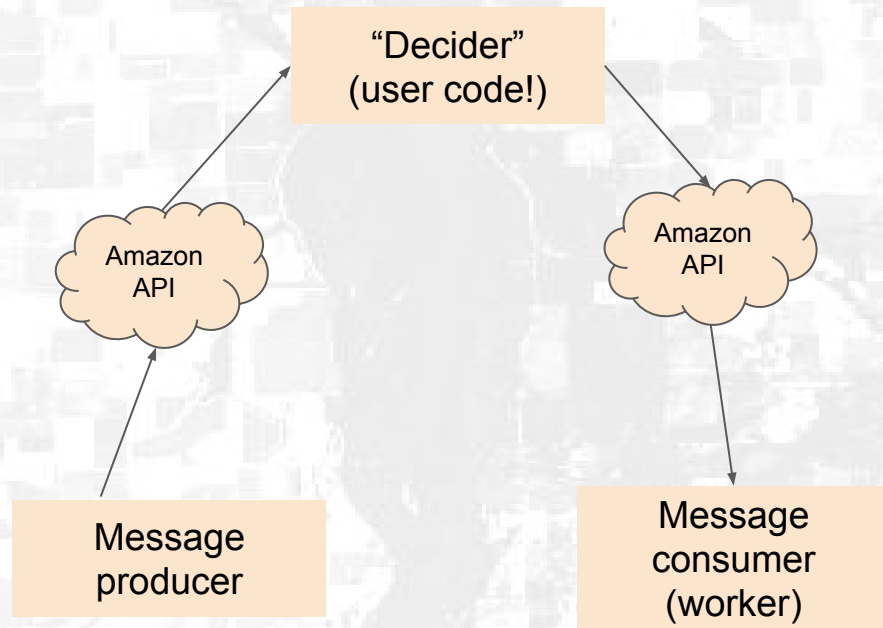


# Simple Workflow Service (SWF) vs. Simple Queue Service (SQS)

## SQS



## SWF



# How we use SWF

We use the relational database to do all our state tracking as opposed to putting much logic into deciders

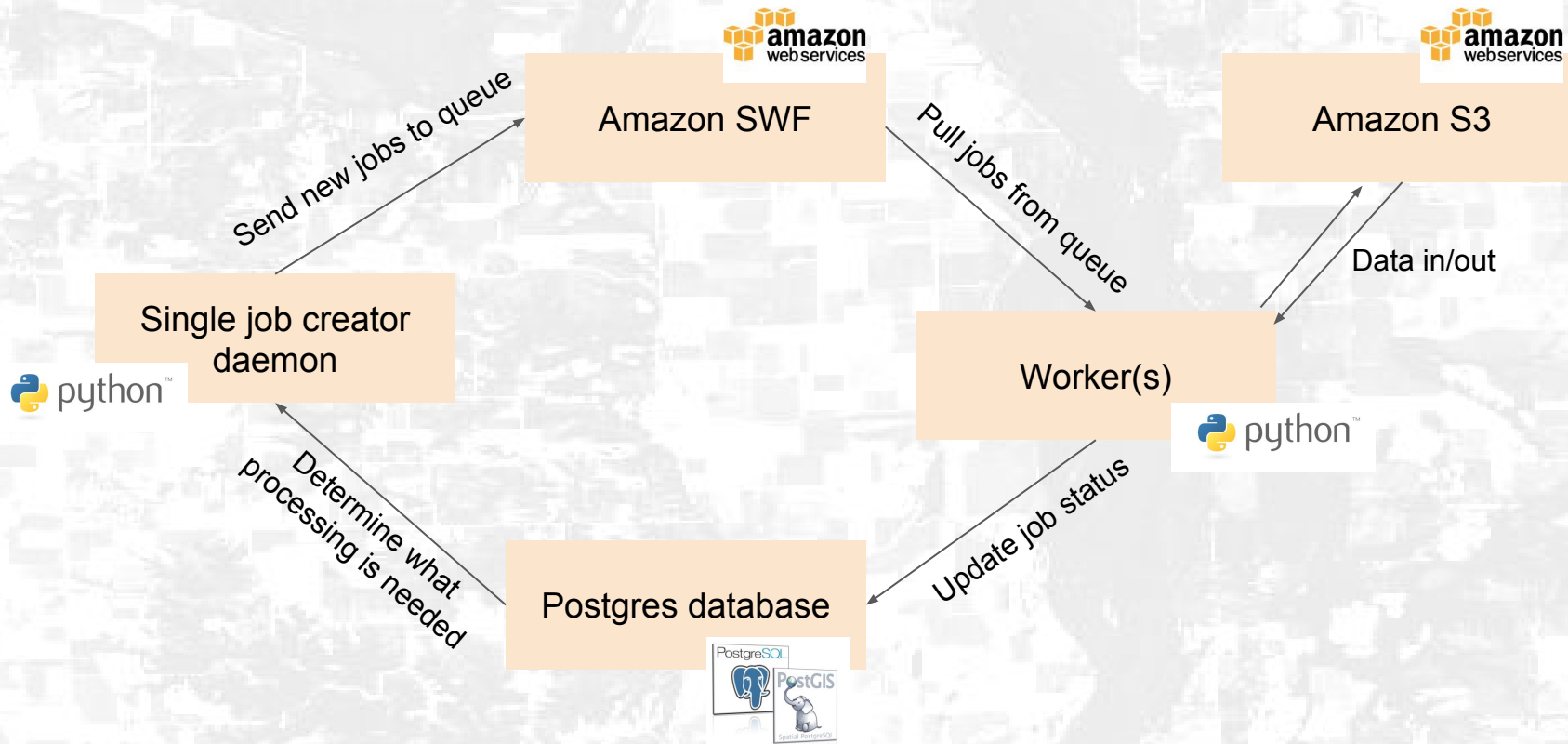
So, SWF currently doesn't provide much benefit over SQS for our purposes

SWF has been cumbersome to work with at times:

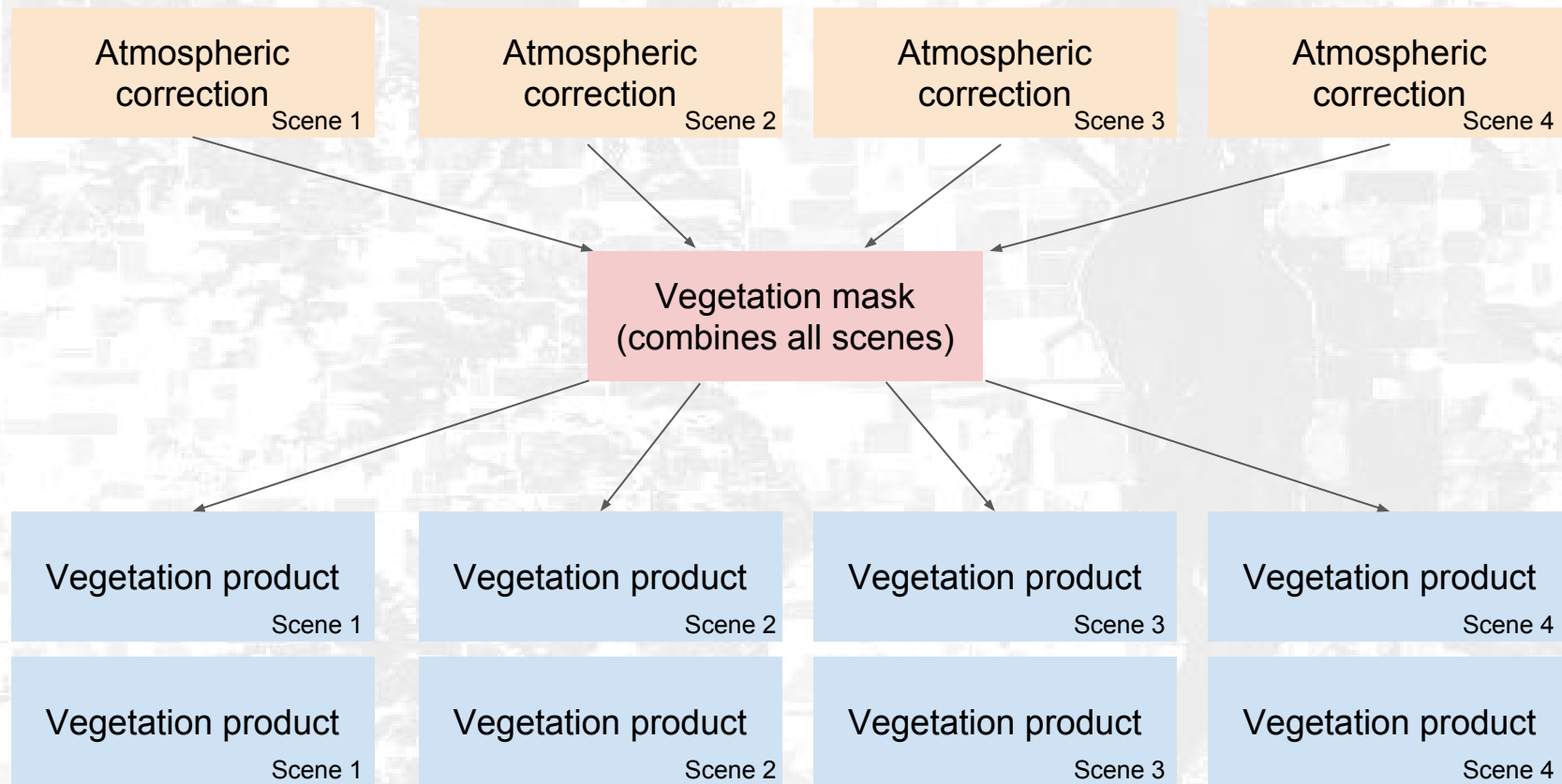
- Decider adds extra piece to deploy and debug
- Terminology is complex and can be confusing
- Too generalized for our use case

Example where we do actually use custom decider logic: retry failed jobs at increasing interval

# Overall processing flow



# Job order dependency



# General flow of each job

Read message from SWF defining job type, scene of interest, other details



Download input data from S3



Manipulate data as needed (atmospheric correction, vegetation algorithm)



Upload result to S3



Inform SWF of task completion

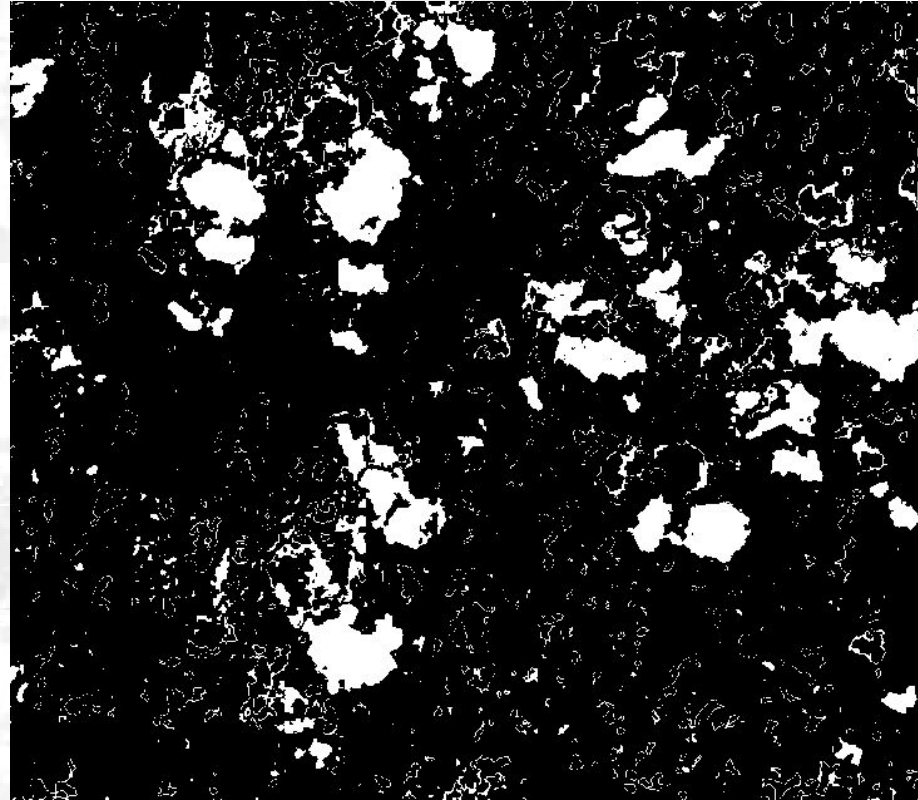
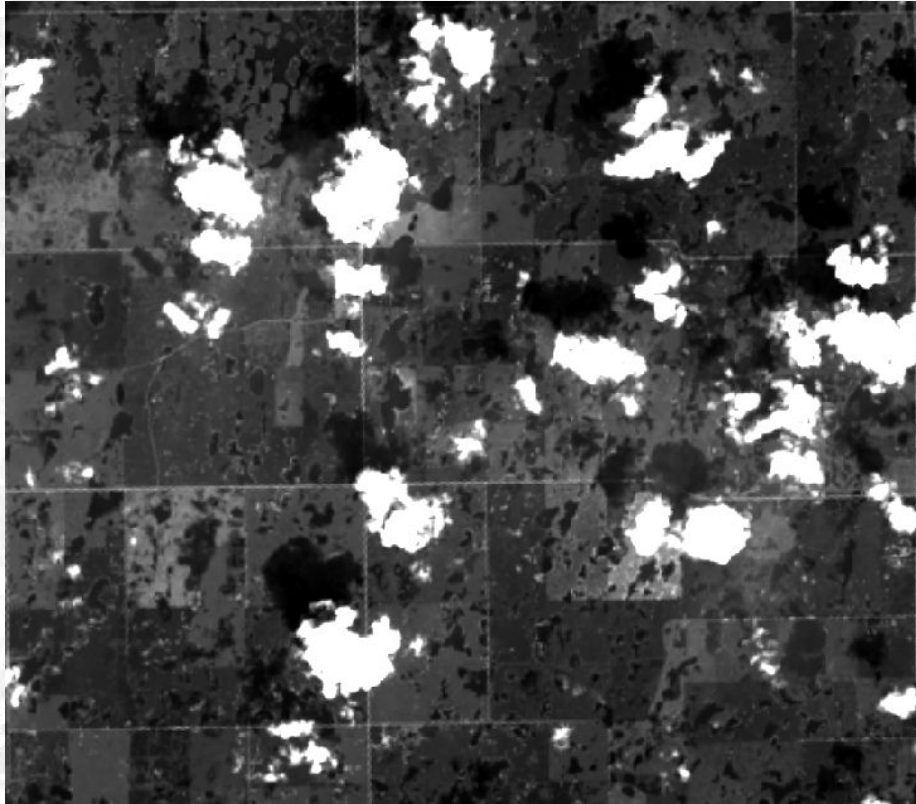
**At every step, the database is updated as needed to indicate success or failure. The daemon process uses this information to determine what further processing tasks need to be kicked off.**



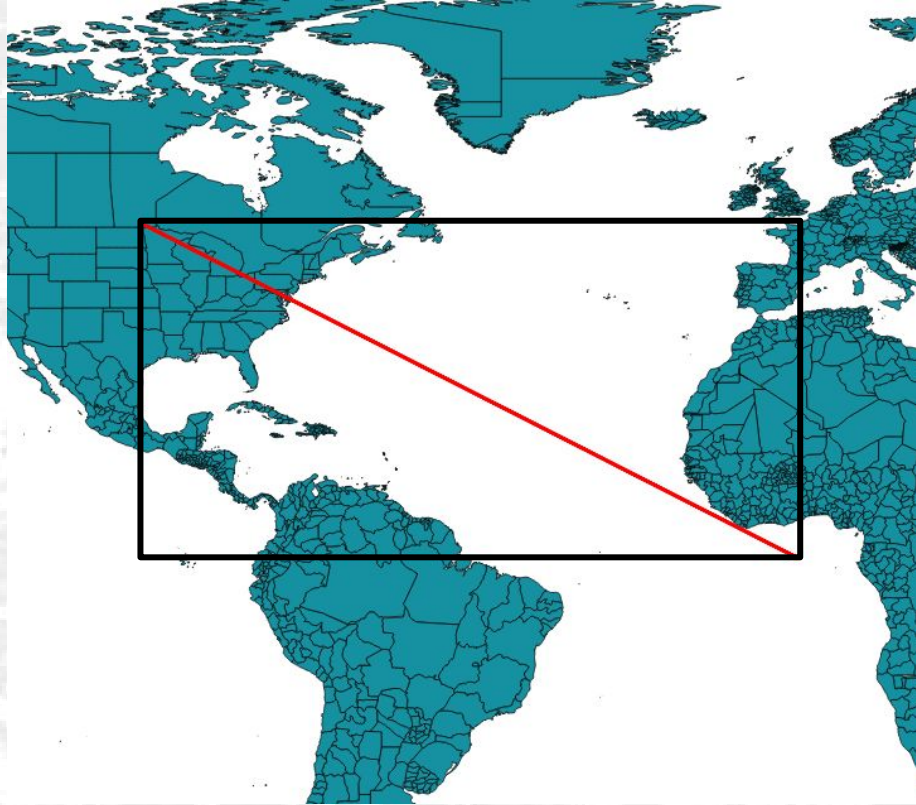
# Scaling strategy

- One scale group for each type of task:
  - Atmospheric correction (longest: large download, then lots of CPU usage by GDAL)
  - Vegetation mask creation
  - Vegetation product algorithm (shortest: smaller download, algorithm itself is fast)
- Multiple scale groups allow us to ensure each task type is completed quickly without wasting resources
- Auto-scaling is a work in progress. Can scale based on:
  - CPU usage
  - I/O
  - Time to complete task
- But, determining proper thresholds for each of those is difficult.
  - Currently we are still scaling each group manually as needed.
  - However, scaling up is “push button”: increase size of a scale group and new instances are provisioned automatically with SaltStack

# Challenge: clouds and cloud shadows



Challenge: bad field boundary causes us to process most of North America



# Challenge: unexpected SWF limits

Can be either helpful or annoying depending on the situation

## Limits on Workflow Executions

- **Maximum open workflow executions** – 100,000 per domain
- This count includes child workflow executions.
- **Maximum workflow execution time** – 1 year
- **Maximum workflow execution history size** – 25,000 events
- **Maximum child workflow executions** – 1,000 per workflow execution.
- **Workflow execution idle time limit** – 1 year (constrained by workflow execution time limit)
- You can configure [workflow timeouts](#) to cause a timeout event to occur if a particular stage of your workflow takes too long.
- **Workflow retention time limit** – 90 days
- After this time, the workflow history can no longer be retrieved or viewed. There is no further limit to the number of closed workflow executions that are retained by Amazon SWF.

## Conclusion

We built a scalable cloud processing system to turn 30+ years of high resolution satellite data into a useful business product that covers large swaths of agricultural North America!

# Acknowledgements

Earthling Interactive for letting me spend time on this talk

Everyone at Earthling for all their help at every step of this work



Thanks!