My name is Dan Hammer, and I am a PhD student at Berkeley studying environmental economics and information theory. My presentation today is, broadly speaking, about the intersection of the two fields. And specifically, the FORMA algorithm to monitor forests (not random forests but leafy, non-stochastic forests) in near real time from satellite imagery using Clojure. And why Clojure was the natural tool for this application.

First, I'll make a pitch for why we should care about monitoring deforestation, especially in an open way. Deforestation accounts for 12% of annual greenhouse gas emissions, nearly as much as the global transportation sector (planes, trains, and automobiles). Any viable effort to mitigate climate change will have to address deforestation. This poses an ever increasing problem, though, since tropical deforestation is accelerating. This time series indicates the number of pixels at 500-meter resolution in the tropicals affected by deforestation for each 16-day period in the past five years.

A measure more worrisome from a policy perspective is the increasing dispersion of deforestation. More and more countries are clearing significant levels of standing forest. This time series indicates the level of dispersion or entropy of deforestation across tropical countries. As the measure increases, the number of countries that must be at the negotiating table must also increase — and the edges between between the countries (or bilateral negotiations) increases exponentially. Whereas in 2008, Brazil and Indonesia accounted for the majority of tropical deforestation — or about 60% — the two countries only account for about 40% today. The complexity of any comprehensive conservation program increases at an order commensurate with the number of bilateral negotiations. And large, cumbersome bureaucracies like the United Nations don't deal well with complexity.

We believe that any viable conservation effort will, in turn, rely on open information on deforestation, so that – among other things – negotiations focus on substantive issues rather than arguments about deforestation measurement.

For this, we created the FORMA algorithm that produces 16-day updates of deforestation at 500-meter resolution. These data are the basis for the Global Forest Watch intiative at the World Resources Institute, which will be launched on May 8.

Everything about this initiative is free and open source — from the raw satellite imagery to the front-end

design. All of our mistakes are on display — in the code or the algorithm design. You can even see if we were having a bad day, given the snarky comments on our pull requests. This project is totally, embarrassingly open.

So, enter Clojure. Or the basic tool we use to process and interpret the satellite data. Our project relies on NASA satellite imagery, which reports the daily spectral signatures of each 500-meter pixel, worldwide. We organize these images into a stack or cube, where each layer is a two-dimensional image. We extract features from this cube along different dimensions. And once we have the features, we apply basic machine learning techniques to interpret the signals toward a normalized measure of forest clearing intensity. It turns out that the feature extraction takes on a common form, where we map a function across partitions. This is where the talk is heading: we map across partitions of a one-dimensional array, a two-dimensional array, a three-dimensional array – and then, basically by induction, an n-dimensional array.

But let me start here. This time series indicates the Normalized Difference Vegetation Index, which is a transformation on the near-infrared and red spectral bands — it is highly correlated with vegetation density. A time series in the context of the data cube is a column at a certain pixel coordinate, given the stacked and sequential images. This line represents the values along the column for a forested pixel in Indonesia that was converted to a palm oil plantation. One of the simplest time series features we extract is the magnitude of the largest short-term drop in the NDVI series. As is common in econometrics, we move along partitions of the series, collecting the slope of the moving blocks. Here are a few. Here are all of them. We are looking for the most negative slope. Even this simple feature is powerful in identifying clearing activity; and as you'll see, this boils down to just a few lines of Clojure code.

In writing this presentation, I have realized the value of restraint in posting code. I think it's valuable to slowly step through this one-dimensional example; but I will rush through the rest, just highlighting the important results.

We depend heavily on the incanter library for matrix manipulation. The first step in collecting the linear trend of a sub-block is building a covariate matrix that is composed of a column of ones, and a running index. Suppose that the block is 24 periods long, or about 1 year. The dimension of this matrix is 24 by 2, with the column of ones and a running index from 0 to 23. The linear trend is derived, then, from premultiplying the the values of the sub-time series by the pseudoinverse of the trend matrix – and then grabbing the second dimension of the output vector which is a 2x1 column vector. We create the constant pseudomatrix, and

then map grab-trend across the partitions of the full time series. Finally, we find the minimum value, the most negative value, within the output vector. As you can see, we have gone from a vector of length 199 to a singleton feature that has real explanatory power in identifying clearing activity. The way we will interpret this number, along with other features, depends on the learning algorithms we define later.

This same framwork is extensible to other, more complicated time series operations – where we map across partitions and reduce along the appropriate dimension.

Suppose that the trajectory or relative sequencing in one time series has some explanatory power in determining the trajectory of sub-blocks in another time series. This lies at the heart of permutation or transfer entropy in information theory. If you Google the code for transfer entropy in R, you will find an absolute mess. There are so many nested loops that the code is incomprehensible. This is standard in the statistics department at Berkeley, which uses R almost exclusively. This same framework of mapping across partitions yields the basis for permutation entropy in just a few lines of code. This function yields the number of subsequences within a larger series. For example, a sequence like 8,5,7 where the highest number comes first, the lowest comes second, and the middle comes last occurs just once. We have found that these reasonably difficult concepts in information theory, time series processing, and even game theory, can be expressed so clearly and compactly in Clojure. We spend less time wrangling the code and more time concentrating on that which *should* be hard – the mathematical properties of the estimators. In fact, the simplicity of the code sometimes reveals concepts that we handn't thought about before. It's very cool. It's a very fun language.

Suppose these are pixels where a fire has occurred. We can find hotspots or clusters of fire activity using precisely the same framework of mapping across partitions. However, here, the partitions are in two dimensions. Note that, within the data cube, this represents a single layer rather than a column. Instead of extracting features by mapping across partitions of a column, we map across partitions of a single layer. I won't go into details, but this function, walk-matrix, effectively acts like partition does over a sequence. We can find all of the hits in each window, or partition, and return a new matrix that acts like a moving sum in two dimensions.

This is what this algorithm yields in production. Each 500-meter pixel indicates the intensity of clearing signal for a particular area in Brazil. The black outlines indicate verified – or as good as verified – deforestation in the same area over the same period. You can see that we do a pretty good job as an alerting system. Much of the intense clearing falls within the boundaries of ex-post verified deforestation. But there is a lot of noise. The two-dimensional moving sum, or average, given the rigid structure of the grid, yields a much

smoother image, where hotspots are highlighted.

In three dimensions things get exciting. Within the data cube, the partitions are not a column, nor a plane, but instead sub-cubes. We apply algorithms to examine the spatiotemporal development of deforestation clusters. We look at the directionality of clearing activity in order to predict where it might be headed next. This is exceedingly relevant, especially in the market for forest carbon credits, where investors want to track the risk-adjusted value of their forest carbon assets. I don't have any compact code to present for this dimension; but you can see that it is merely a generalization of the two simpler cases.

Likewise, mapping across partitions of an n-dimensional array is also a generalization, which we use for ... well, I haven't found a reasonable, real-life use case. But it's possible to compute, I guess. And I love that the constraint isn't the code but rather my own lack of creativity.

So, now to the actual interpretation of the extracted features. We use a semi-trained learning algorithm, derived from a logistic classifier. We match the features against historical, training data on deforestation in order to interpret the feature vectors after the training period has ended. The features are collected from streaming NASA data, whereas the training data reports cumulative deforestation between 2000 and 2005. We align the 0-1 labels on whether the pixel was deforested during the training period with the feature vectors representing data during the same period, 2000-2005. We then estimate the parameter vector required to convert the feature vector into a normalized (between zero and 1) measure of deforestation.

The data are big. Way bigger than anything I had ever dealt with.

But using cascalog solved our problems.

It is noteworthy, here, that I am not a computer scientist. Far from it, I study economics, which means I am skilled at ruining a perfectly good code base and the economy. I don't know Java and I know very little about Cascading. And yet, because I figured out Cascalog, I process and intensively analyze terabytes of satellite imagery on Hadoop. This is also cool. I feel like a rockstar in a department where professors still have little rolly balls in their mouses.

This query produces a beta vector for each ecoregion. We train the model separately for each ecoregion; and thus create a separate vector for each ecoregion, based only on pixels within that ecoregion. Each pixel is

associated with a label of historical deforestation or not (one or zero) an ecoregion and an NDVI time series. This is mostly just a dummy example; but it shows the basic framework. We call a clojure function called create-features that accepts an NDVI series and returns a vector of features. We then call estimate-beta which is a buffer operation that collects the labels and features all pixels within each ecoregion and returns the beta vector.

This constitutes another tap, which we can call in a subsequent query. We join on ecoregion in order to apply the beta vector to calculate the probability for each pixel, based on the join on ecoregion. Using the cascalog testing library, this tap produces tuples with a unique, global pixel identifier and the intensity of clearing activity associated with it.

Now, all we have to do is map and visualize the output. We work closely with a group called Vizzuality, which makes truly beautiful maps based on their back-end CartoDB, which is a wrapper for PostGres. In lieu of a direct cascalog tap to CartoDB, we lazily upload the pixels and probabilities into CartoDB.

Again everything is open source. And this is what we've got: An easy, easy way to visualize the data. A way that anyone can view, interpret, and analyze.