



# Masquerading Malicious DNS Traffic

Bayesian Inference, Rainier, Spark

David Rodriguez

March 28, 2019

# The Outline



Masquerading  
DNS  
Traffic



Time Series  
Modeling



Rainier  
+  
Spark



Anomaly  
Detection

# The Outline



Masquerading  
DNS  
Traffic



Time Series  
Modeling



Rainier  
+  
Spark



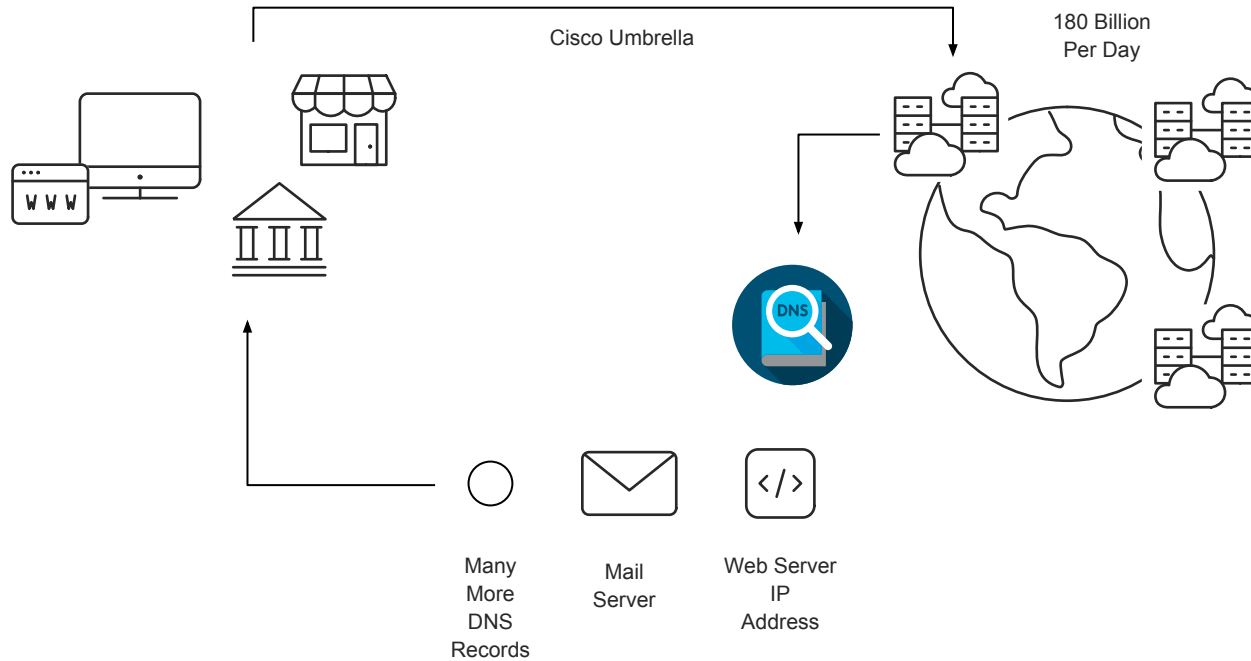
Anomaly  
Detection

# You've been chosen for a top-secret mission: Threat Busters.

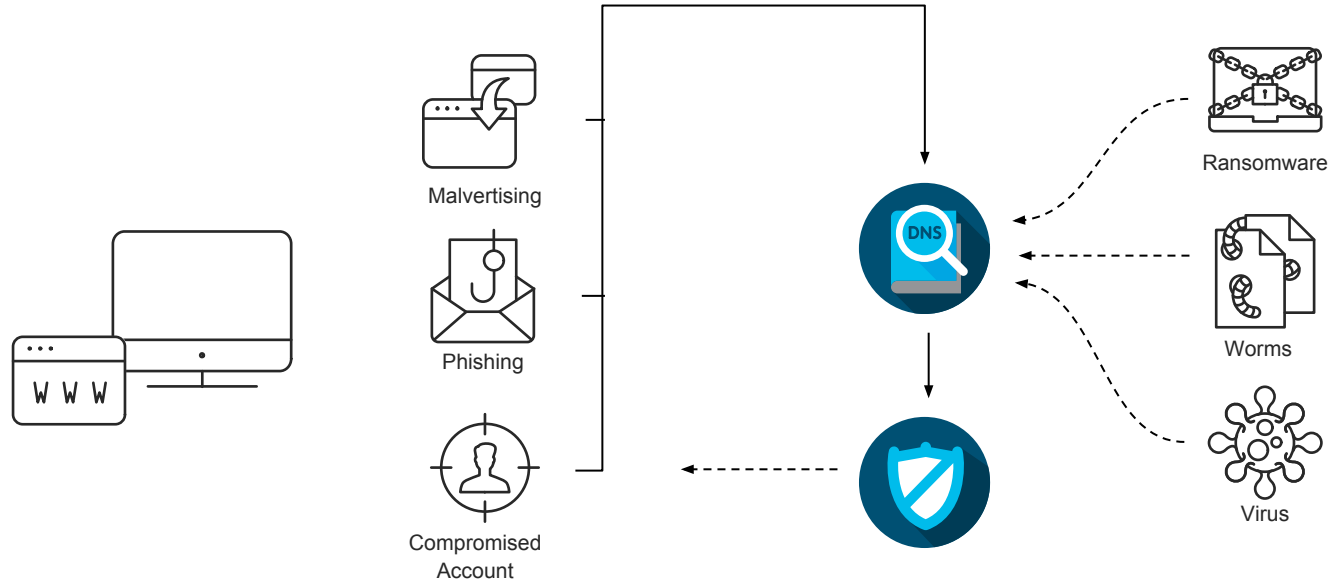
Gather threat intel to defend your network.

[Accept the mission](#)

# Part 1 DNS Resolution



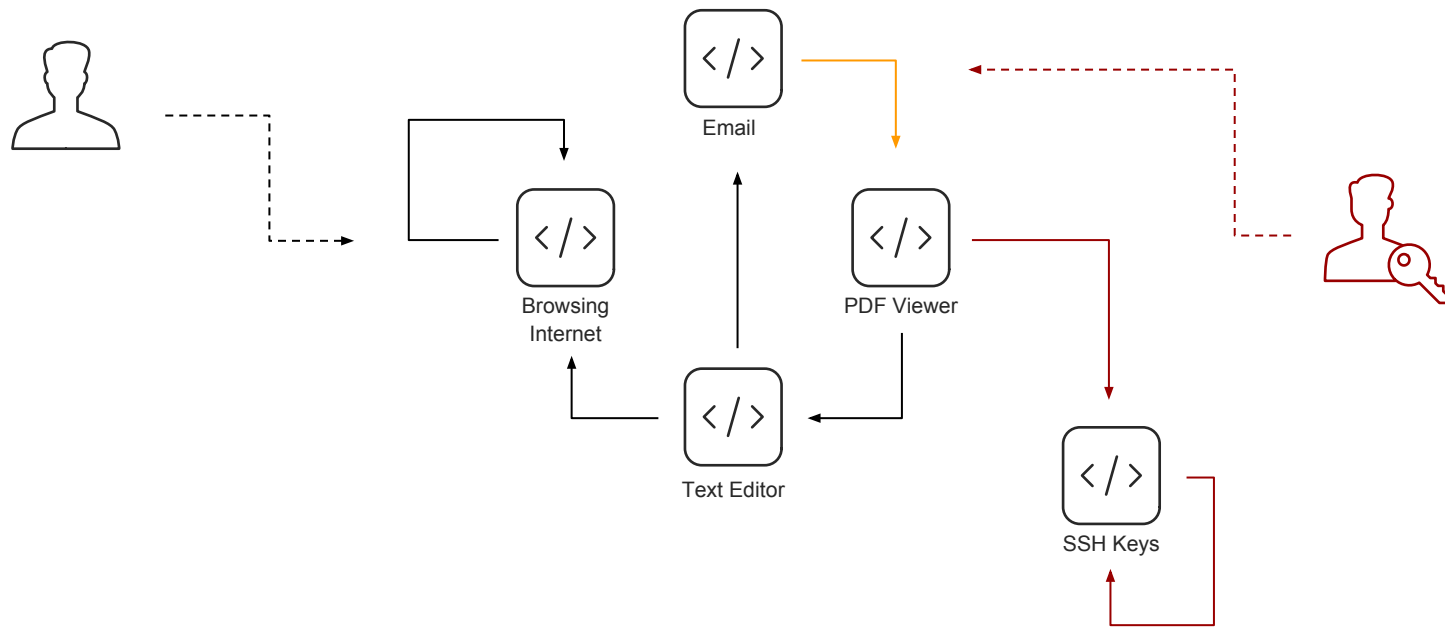
# Part 1 Protection 101



## Part 1 Definition

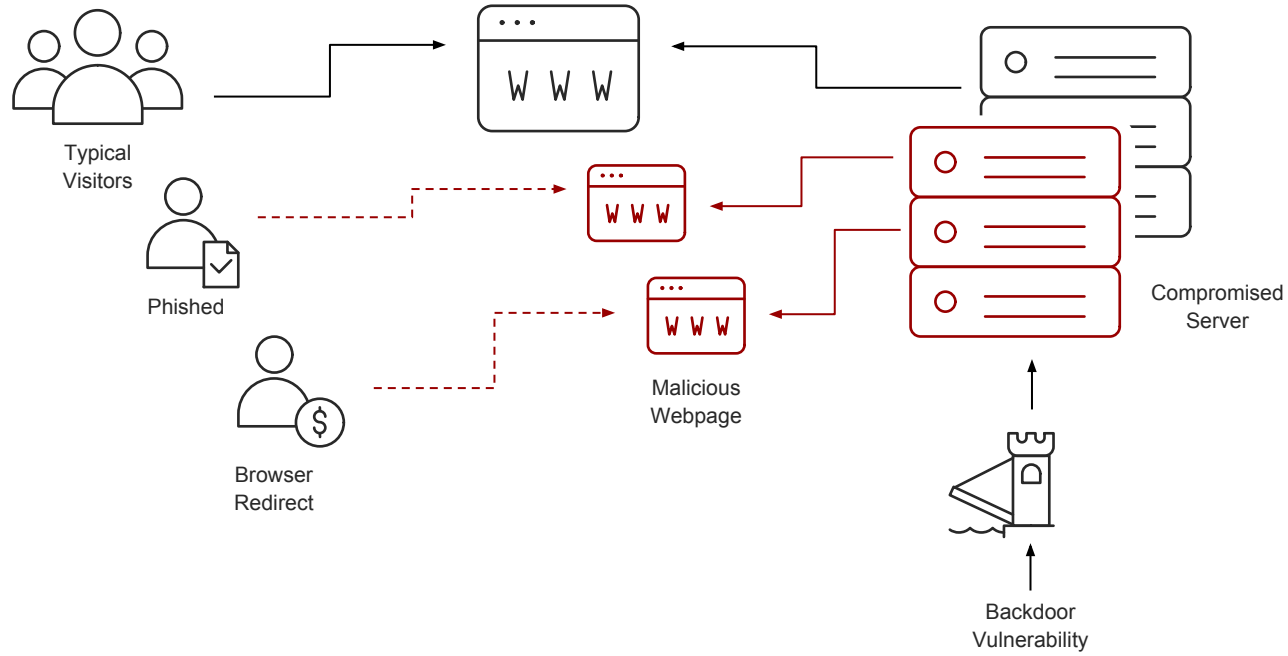
Masquerading Traffic = Masquerading Users +  
Compromised Websites

# Part 1 Masquerading Users

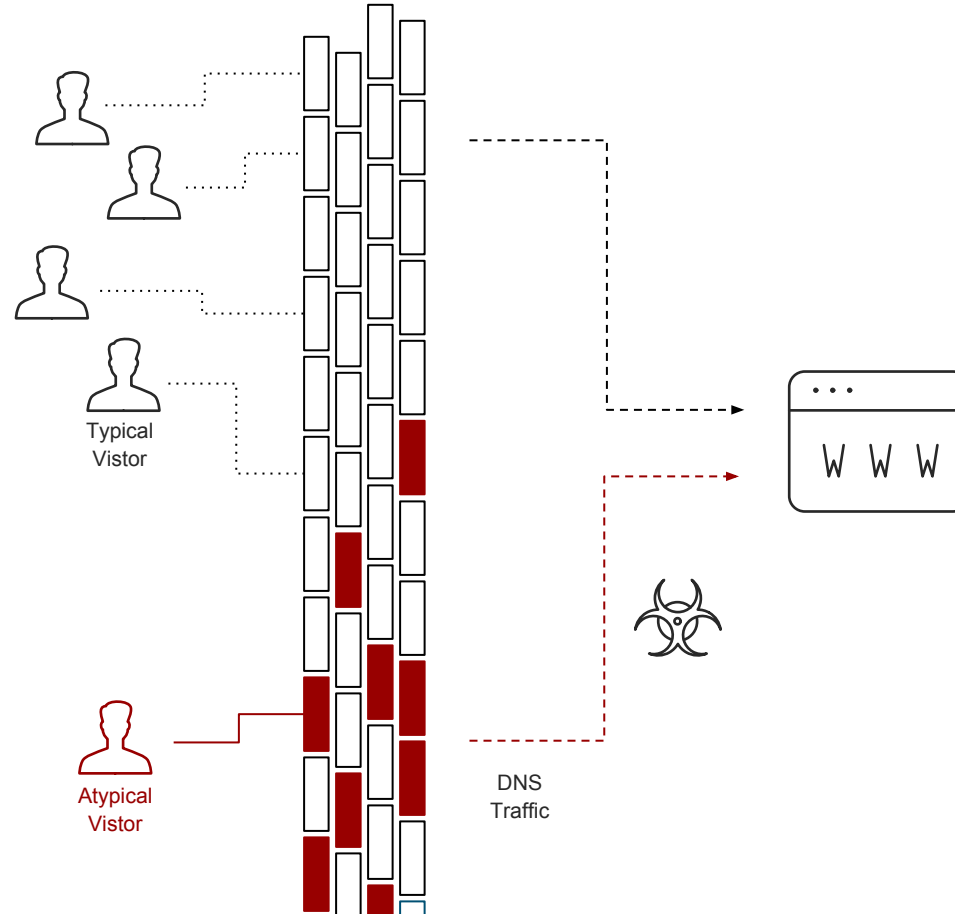




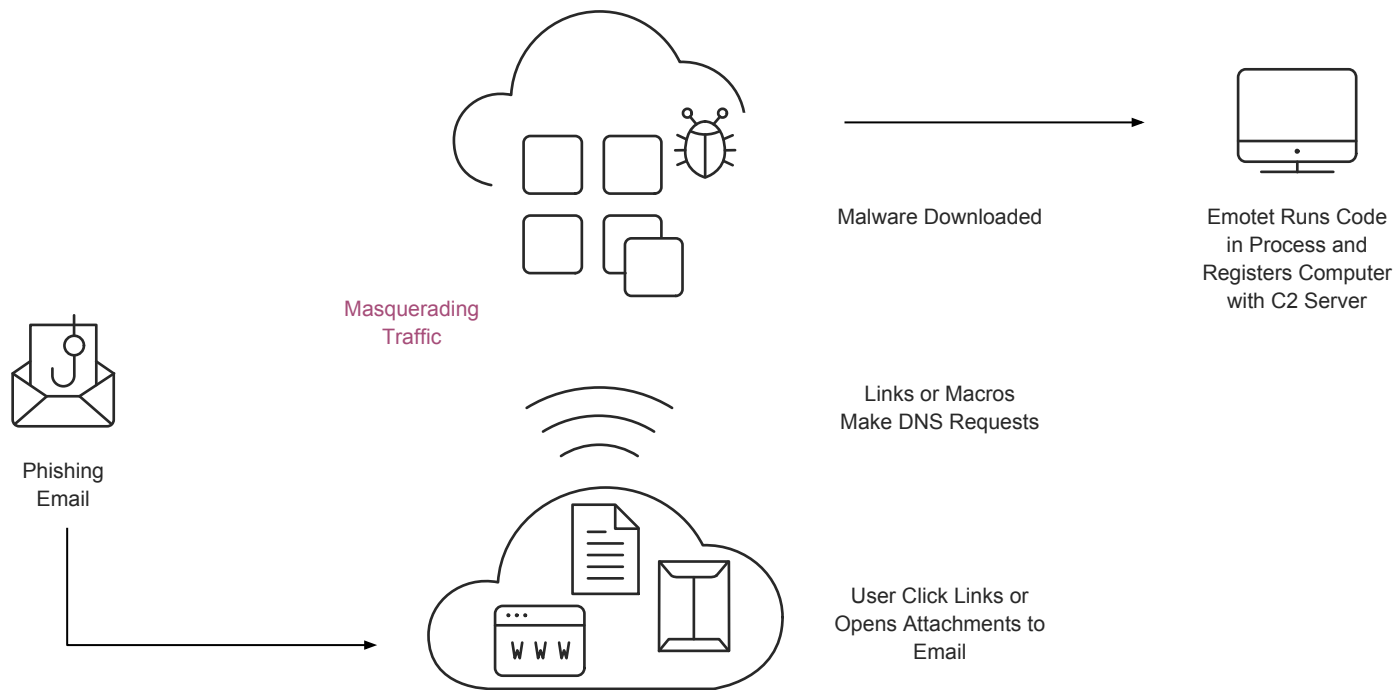
# Part 1 Compromised Websites



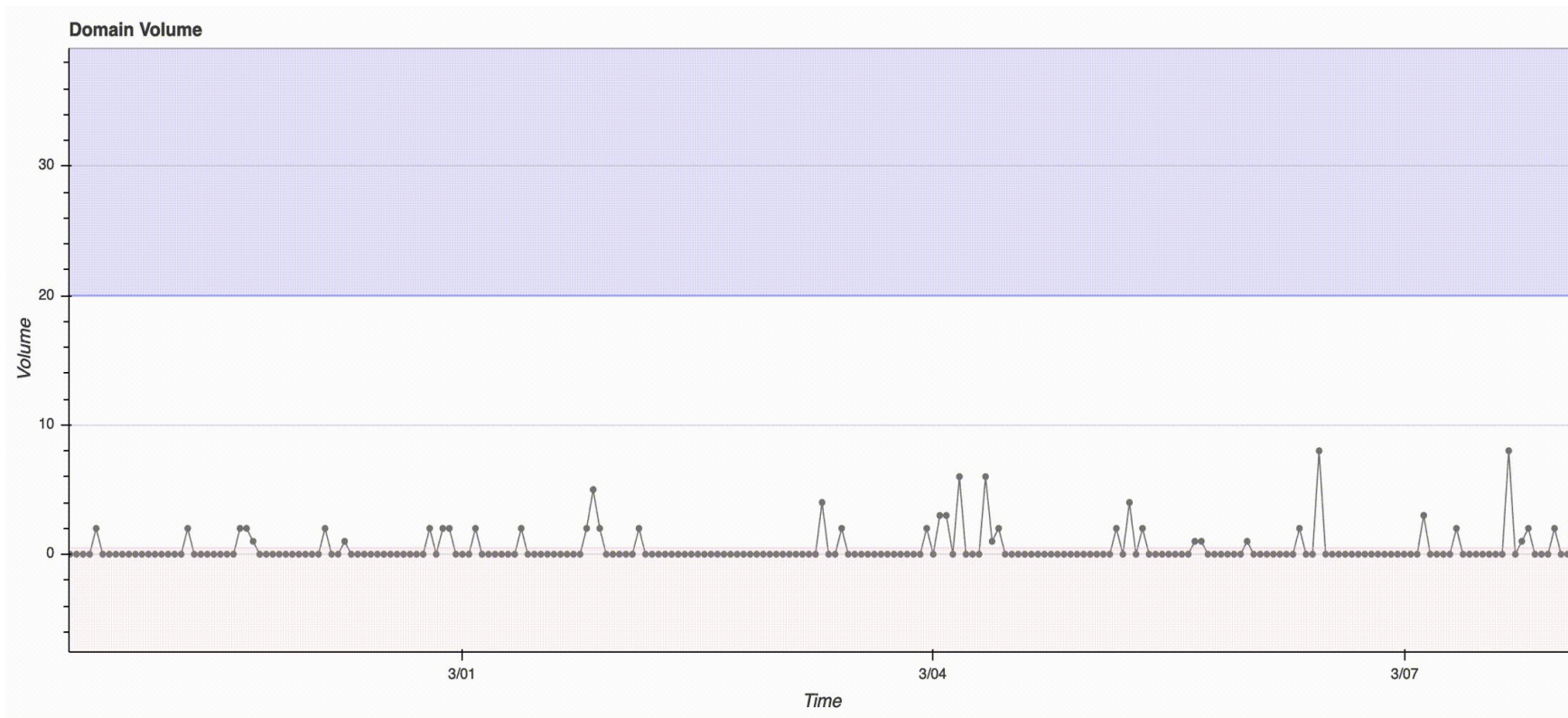
# Part 1 Masquerading DNS Traffic



# Part 1 Emotet Campaign



# Part 1 Emotet Campaign



# The Outline



Masquerading  
DNS  
Traffic



Time Series  
Modeling

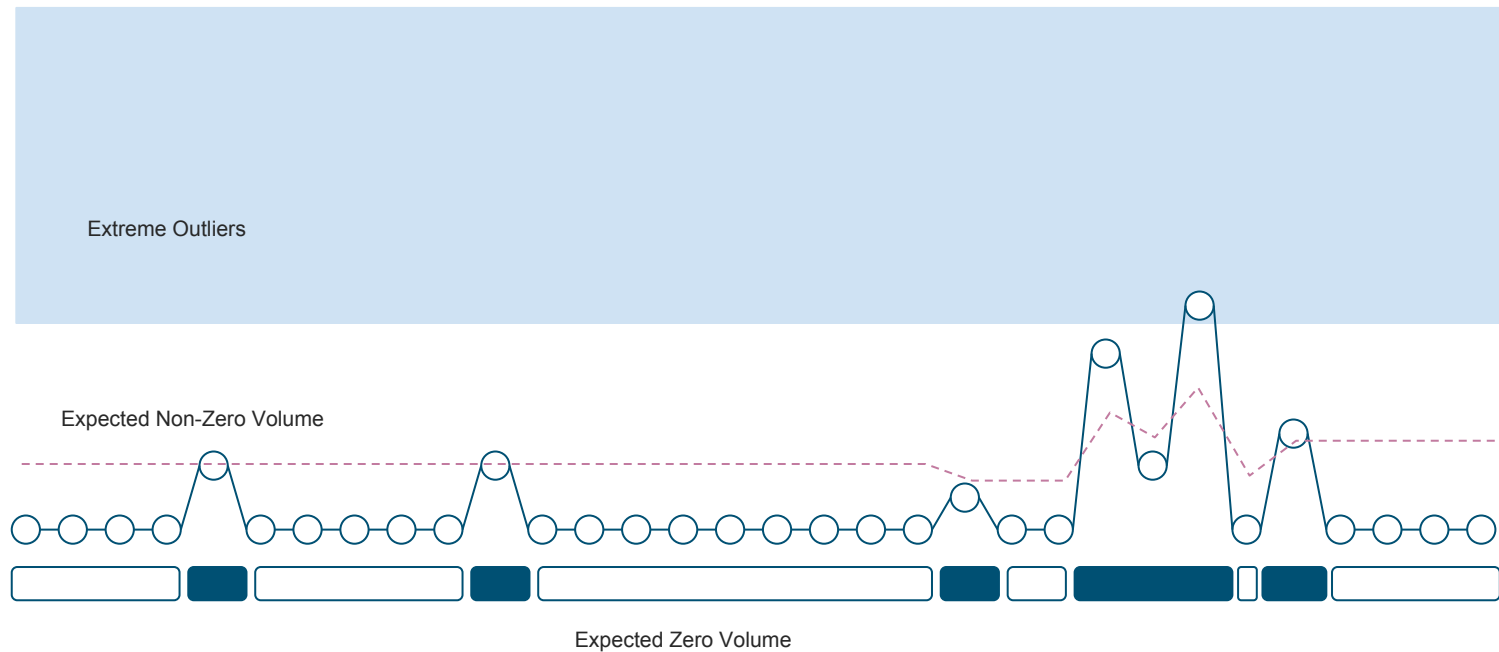


Rainier  
+  
Spark

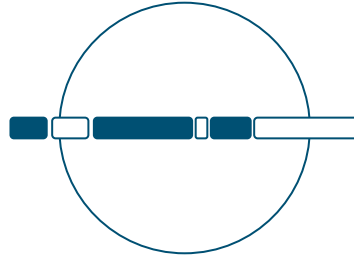


Anomaly  
Detection

# Part 2 Time-Series Analysis

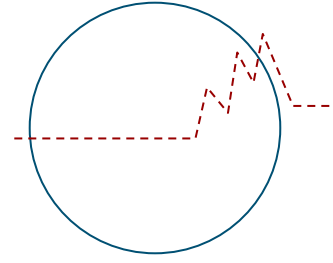


## Part 2 Time-Series Analysis



$\psi$

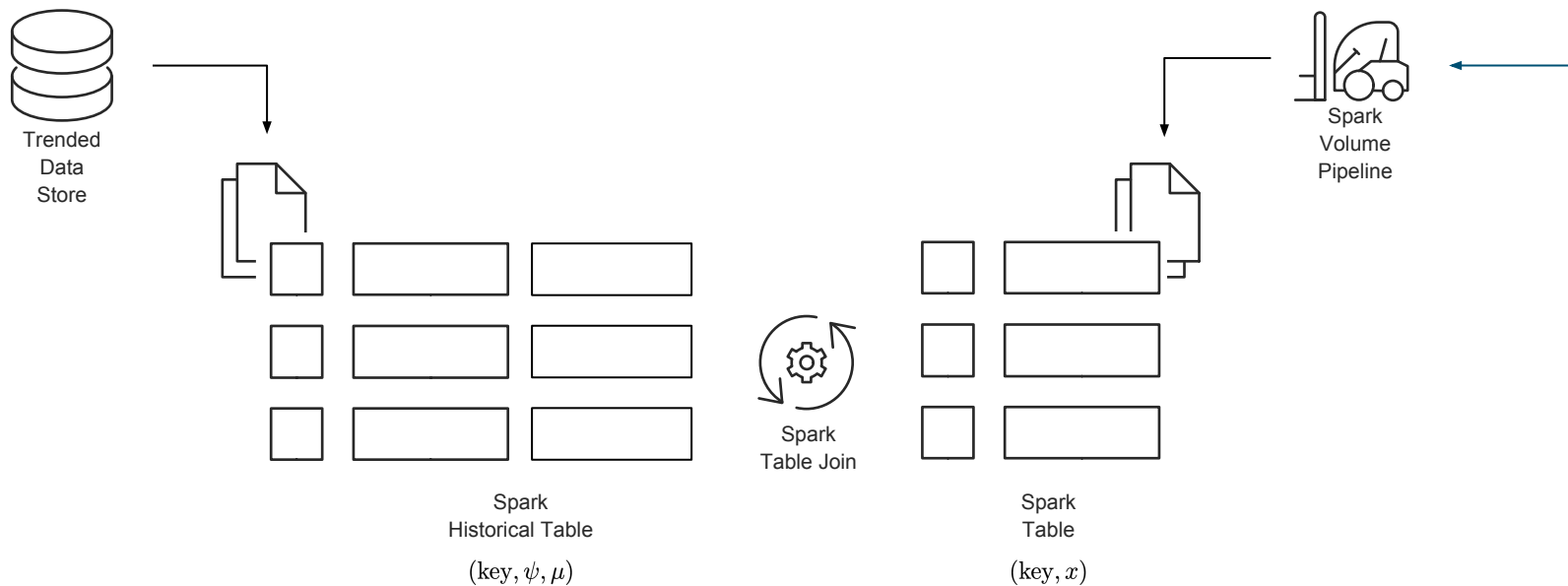
Probability of Demand



$\mu$

Expected Demand  
when non-zero

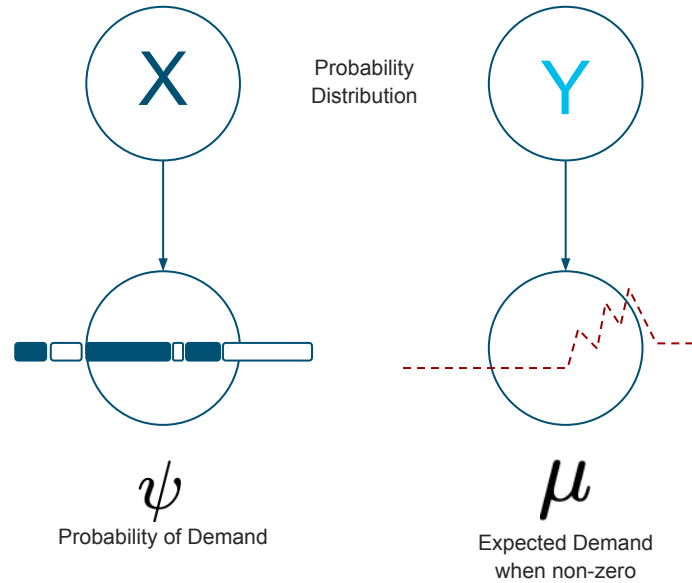
## Part 2 Croston's Method



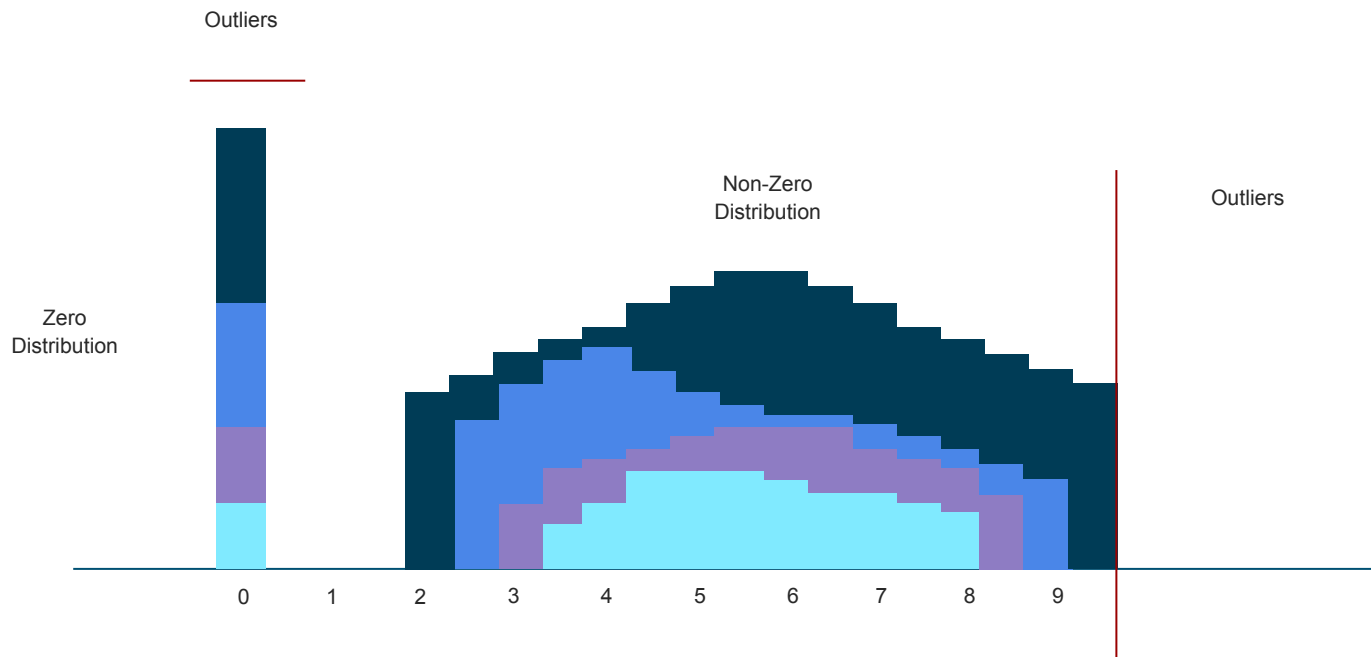
Note :  $\mu_{new} = \mu \cdot w + (1 - w) \cdot x$      $\psi_{new} = \psi \cdot w + (1 - w) \cdot \chi_{x>0}$



# Part 2 Bayesian Approach



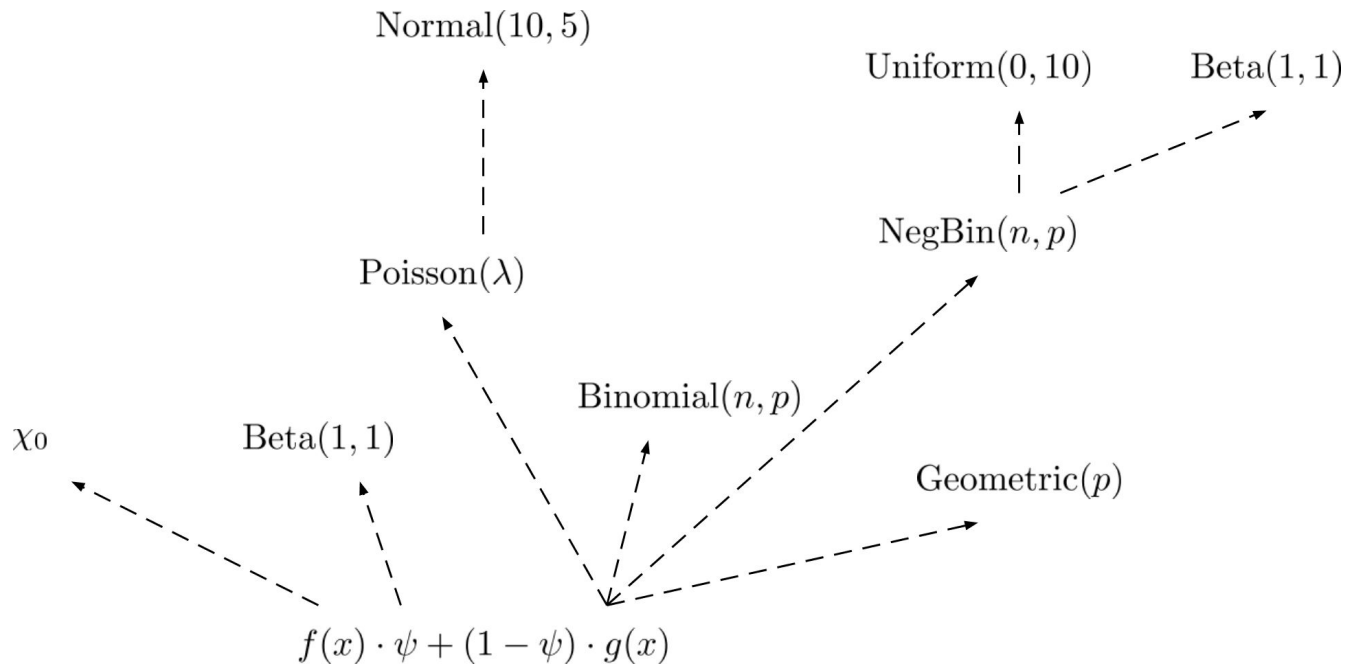
# Part 2 Bayesian Approach



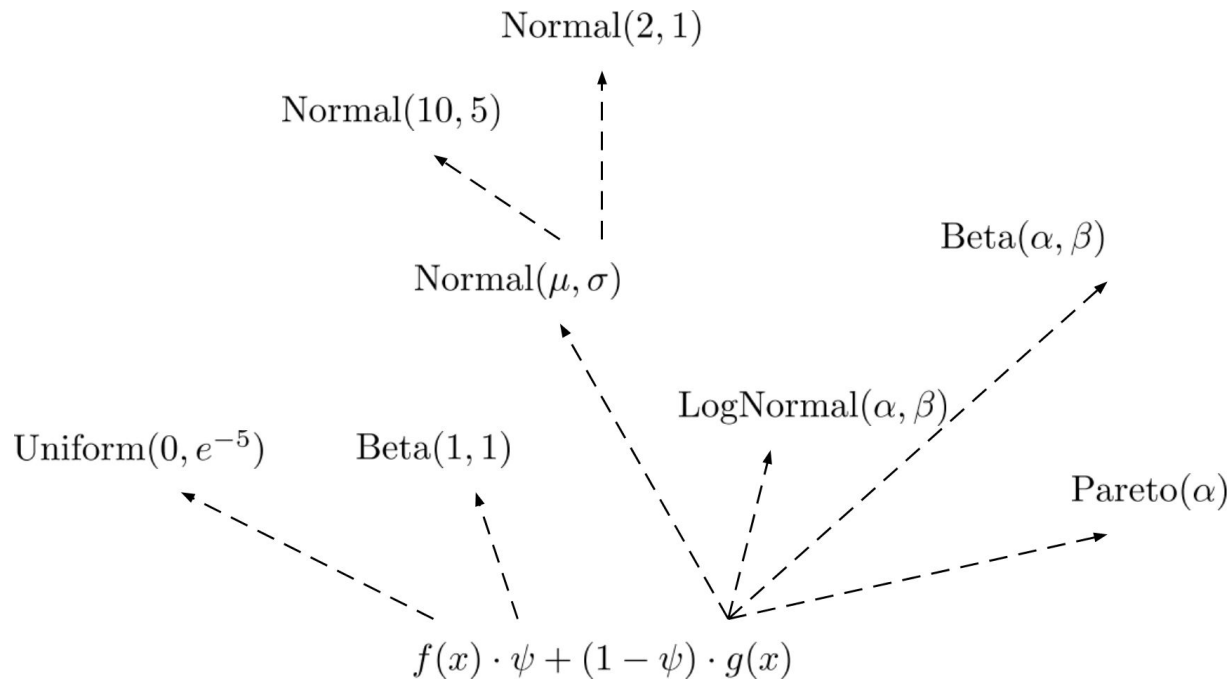
## Part 2 Mixture Models

$$f(x) \cdot \psi + (1 - \psi) \cdot g(x)$$

## Part 2 Discrete Models



## Part 2 Continuous Models



# The Outline



Masquerading  
DNS  
Traffic



Time Series  
Modeling

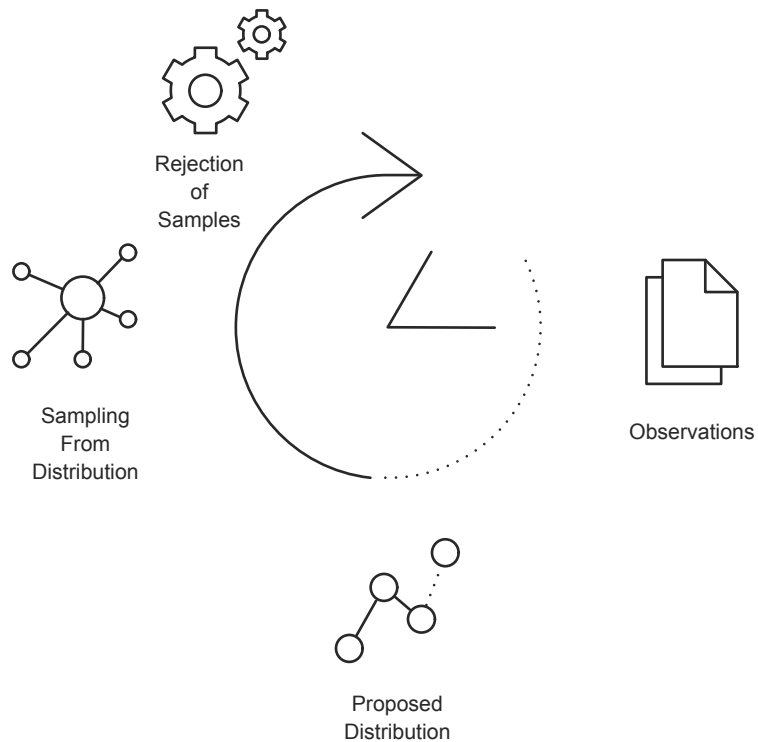


Rainier  
+  
Spark





Anomaly  
Detection

# Part 3 MCMC Methods



# Part 3 MCMC Methods

 [Tutorials](#) [Examples](#) [Books + Videos](#) [API](#) [Developer Guide](#) [About PyMC3](#)



# PyMC3

Probabilistic Programming in Python

[Quickstart →](#)

launch **render** build **passing** coverage **89%** powered by **sumFOCUS**

### Friendly modelling API

PyMC3 allows you to write down models using an intuitive syntax to describe a data generating process.

### Cutting edge algorithms and model building blocks

Fit your model using gradient-based MCMC algorithms like NUTS, using ADVI for fast approximate inference – including minibatch-ADVI for scaling to large datasets – or using Gaussian processes to build Bayesian nonparametric models.

```
import pymc3 as pm

X, y = linear_training_data()
with pm.Model() as linear_model:
    weights = pm.Normal('weights', mu=0, sd=1)
    noise = pm.Gamma('noise', alpha=0, beta=1)
    y_observed = pm.Normal('y_observed',
                           mu=X.dot(weights),
                           sd=noise,
                           observed=y)

prior = pm.sample_prior_predictive()
posterior = pm.sample()
posterior_pred = pm.sample_posterior_predictive(posterior)
```

### Installation

Via conda-forge:

```
conda install -c conda-forge pymc3
```

 <a href="#">WORKSPACE</a>	Generated new Bazel BUILD files and hand-corrected core's BUILD (#322)	2 months ago
 <a href="#">build.sbt</a>	jupyter support for evilplot (#334)	22 days ago
 <a href="#">rainier.jpg</a>	add rainier image	10 months ago
 <a href="#">version.sbt</a>	0.2.2 prep (#335)	22 days ago

## README.md



## Rainier

build **passing** coverage **89%** maven-central **v0.2.2** chat [ask](#) [github](#)

Rainier provides an idiomatic, high-performance functional Scala API for bayesian inference via Markov Chain Monte Carlo.

Rainier allows you to describe a complex prior distribution by composing primitive distributions using familiar combinators like `map`, `flatMap`, and `zip`; condition that prior on your observed data; and, after an inference step, sample from the resulting posterior distribution.

Underlying this is a static scalar compute graph with auto-differentiation and very fast CPU-based execution.

It is implemented in pure Scala, with minimal external dependencies and no JNI libs, and as such is convenient to deploy, including to Spark or Hadoop clusters.

Rainier currently provides two samplers: `affine-invariant MCMC`, an ensemble method popularized by the [Emcee](#) package in Python, and `Hamiltonian Monte Carlo`, a gradient-based method used in [Stan](#) and [PyMC3](#).



## Part 3 Rainier

Depending on your background, you might think of Rainier as aspiring to be either:

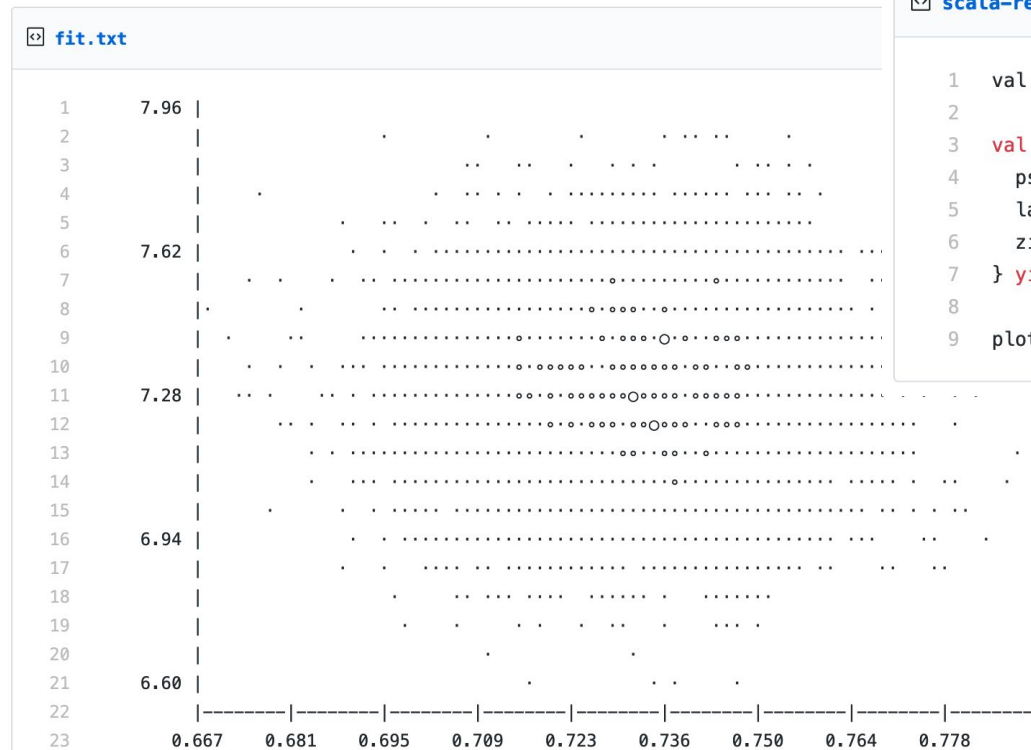
**“Stan, but on the JVM”**

or

**“Tensorflow, but for small data”.**

~ README

## Part 3 Rainier Methods

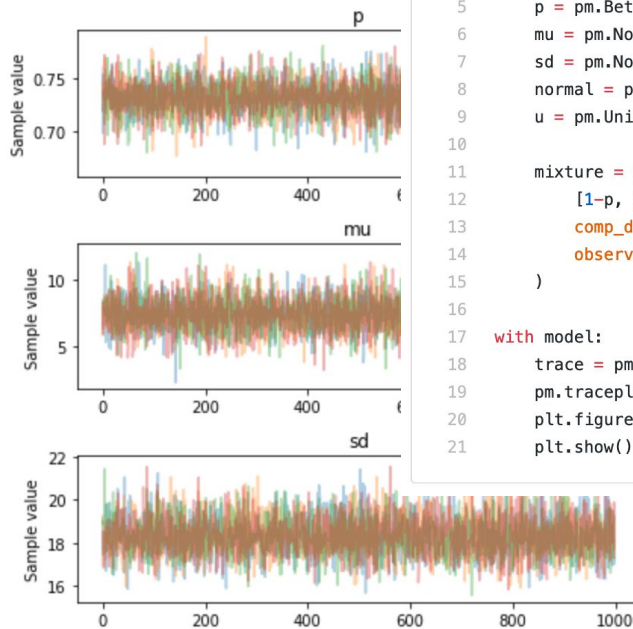
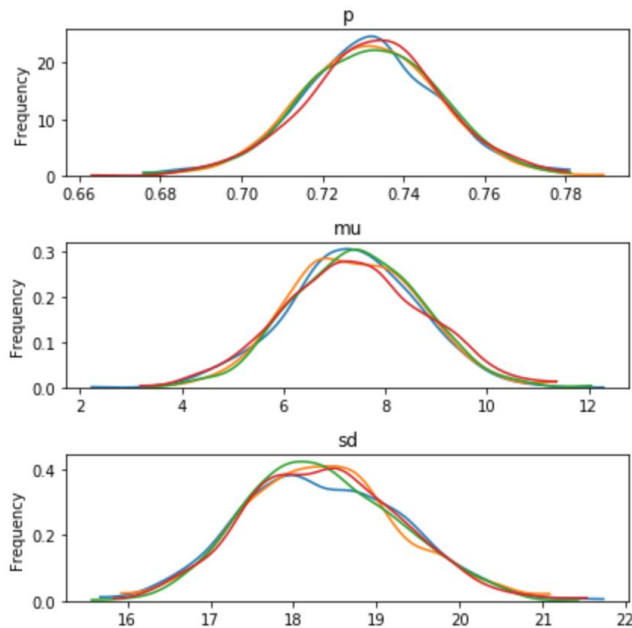


```
<> scala-repl.scala
```

```
1 val data = List(2,0,0,3,3,3,0,3,5,0,0,0,0,0,0,0,0,0,0,2,0)
2
3 val zipModel = for {
4   psi <- Beta(1,1).param
5   lambda <- Normal(5, 5).param
6   zip <- Poisson(lambda).zeroInflated(psi).fit(data)
7 } yield (psi, lambda)
8
9 plot2D(zipModel.sample())
```

# Part 3 PyMC Methods

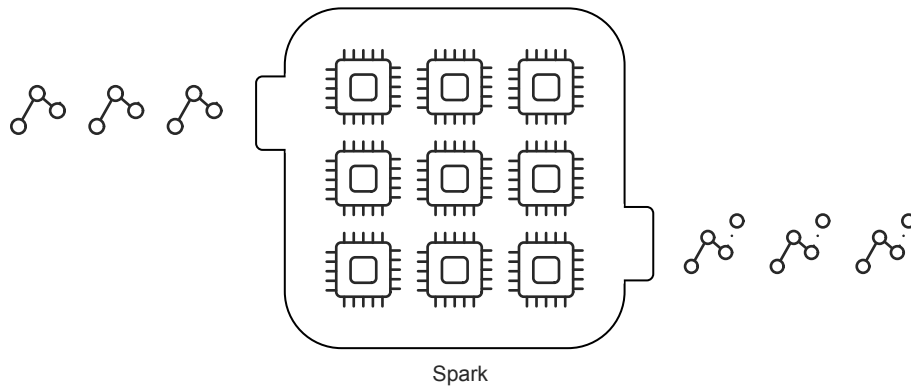
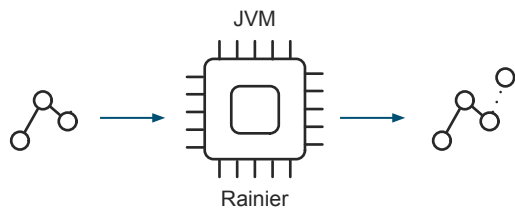
zero\_inflated\_normal.png



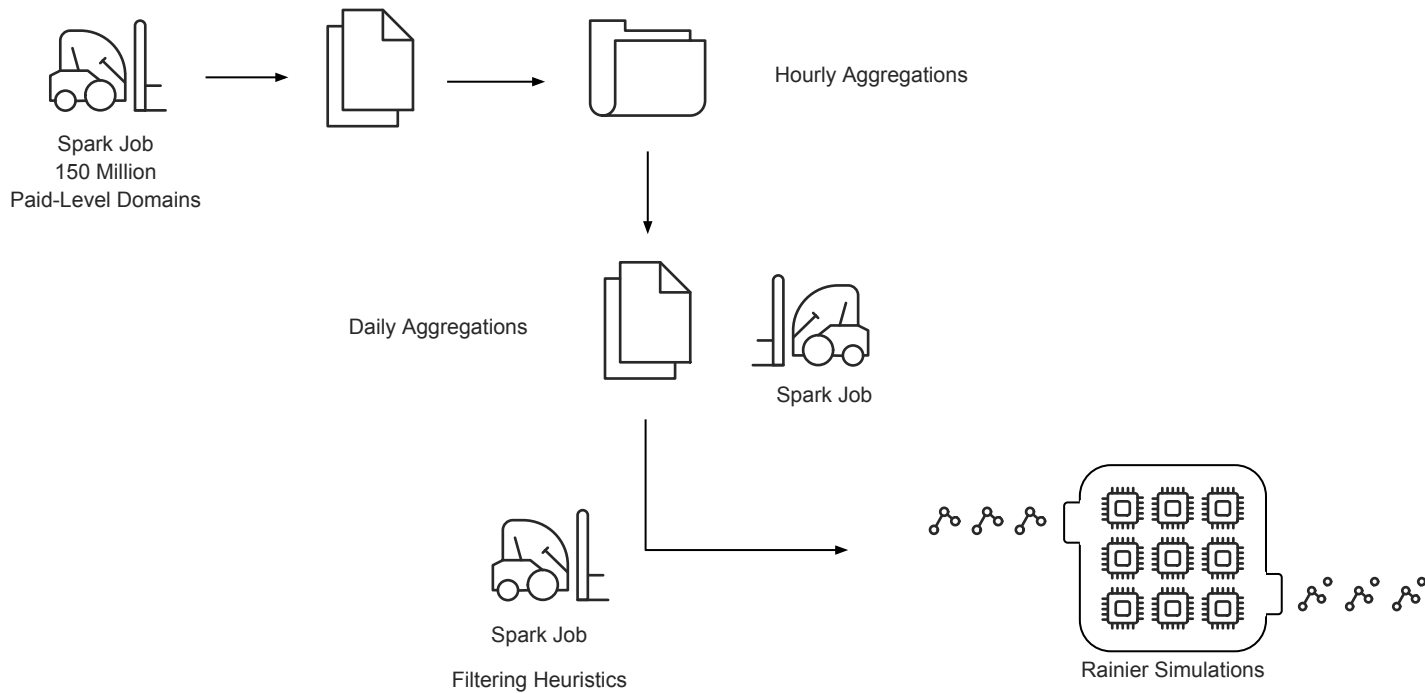
pymc-zero-inflated-normal.py

```
1 data = [7.650241800665554e-05, 3.923801883327712e-05, 3.0, 3.0, 3.0]
2
3 model = pm.Model()
4 with model:
5     p = pm.Beta('p', 1, 1)
6     mu = pm.Normal('mu', 10, 10)
7     sd = pm.Normal('sd', 10, 10)
8     normal = pm.Normal('normal', mu, sd)
9     u = pm.Uniform('u', 0, 1e-5)
10
11     mixture = pm.Mixture('mixture',
12                           [1-p, p],
13                           comp_dists=[normal.distribution, u.distribution],
14                           observed=data)
15
16
17 with model:
18     trace = pm.sample(1000, tune=1000)
19     pm.traceplot(trace, varnames=['p', 'mu', 'sd'])
20     plt.figure()
21     plt.show()
```

# Part 3 Rainier + Spark



# Part 3 Rainier + Spark



## [Rainier] Massive Bayesian Inference in Spark using Rainier

 **SparkRainer.scala**

Raw

```
1  import com.stripe.rainier.core.{Normal, Poisson}
2  import com.stripe.rainier.sampler.{RNG, ScalaRNG}
3  import org.apache.spark.{SparkConf, SparkContext}
4
5  object Driver {
6    implicit val rng: RNG = ScalaRNG(1527608515939L)
7    val DROP_BURN_IN = 100
8
9    /*
10     Refer to StackOverflow Q, about serializing methods/objects:
11     https://stackoverflow.com/questions/22592811/task-not-serializable-java-io-notserializableexception-when-calling-funct
12     */
13     def genMapper[A, B](f: A => B): A => B = {
14       val locker = com.twitter.chill.MeatLocker(f)
15       x => locker.get.apply(x)
16     }
17
18     def average(l: List[Double]): Double =
19       l.size.toDouble / l.sum
20
21     def dropBurnIn(dropBurn: Int)(v: List[Double]): List[Double] =
22       v.drop(dropBurn)
23
24
25     def fitPoisson(y: List[Int]): List[Double] = {
26       val rate = for {
```

# The Outline



Masquerading  
DNS  
Traffic



Time Series  
Modeling

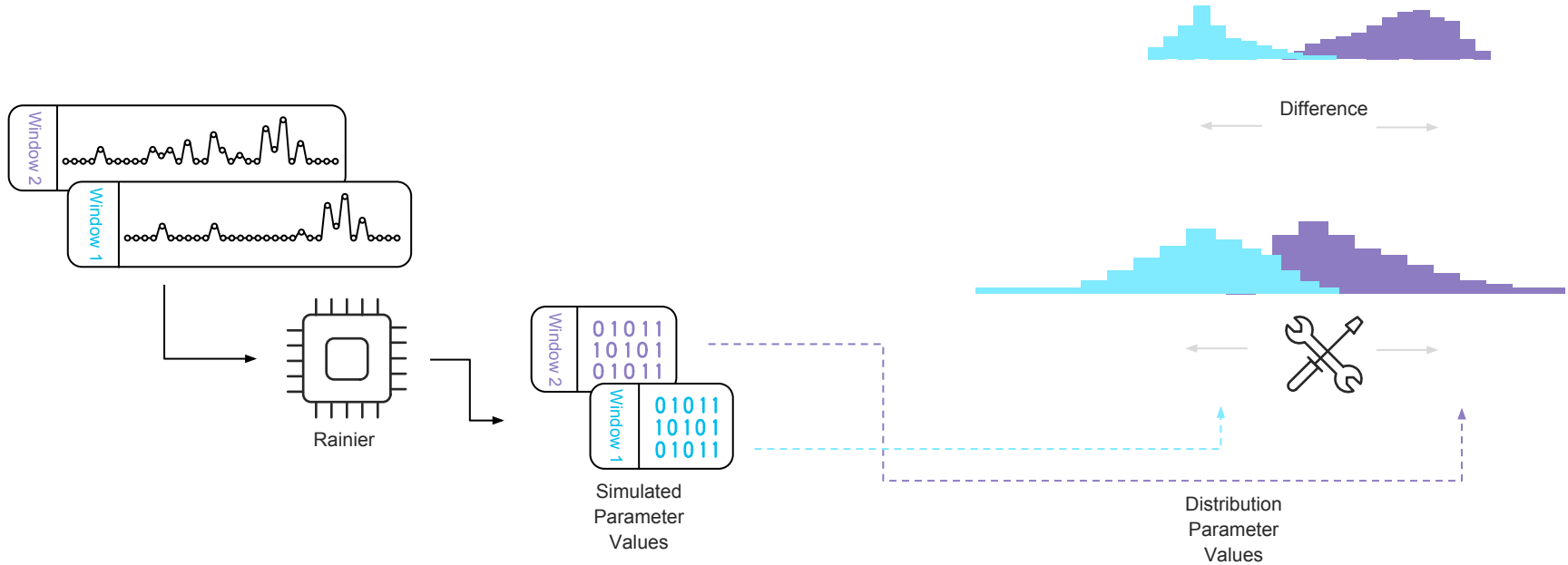


Rainier  
+  
Spark



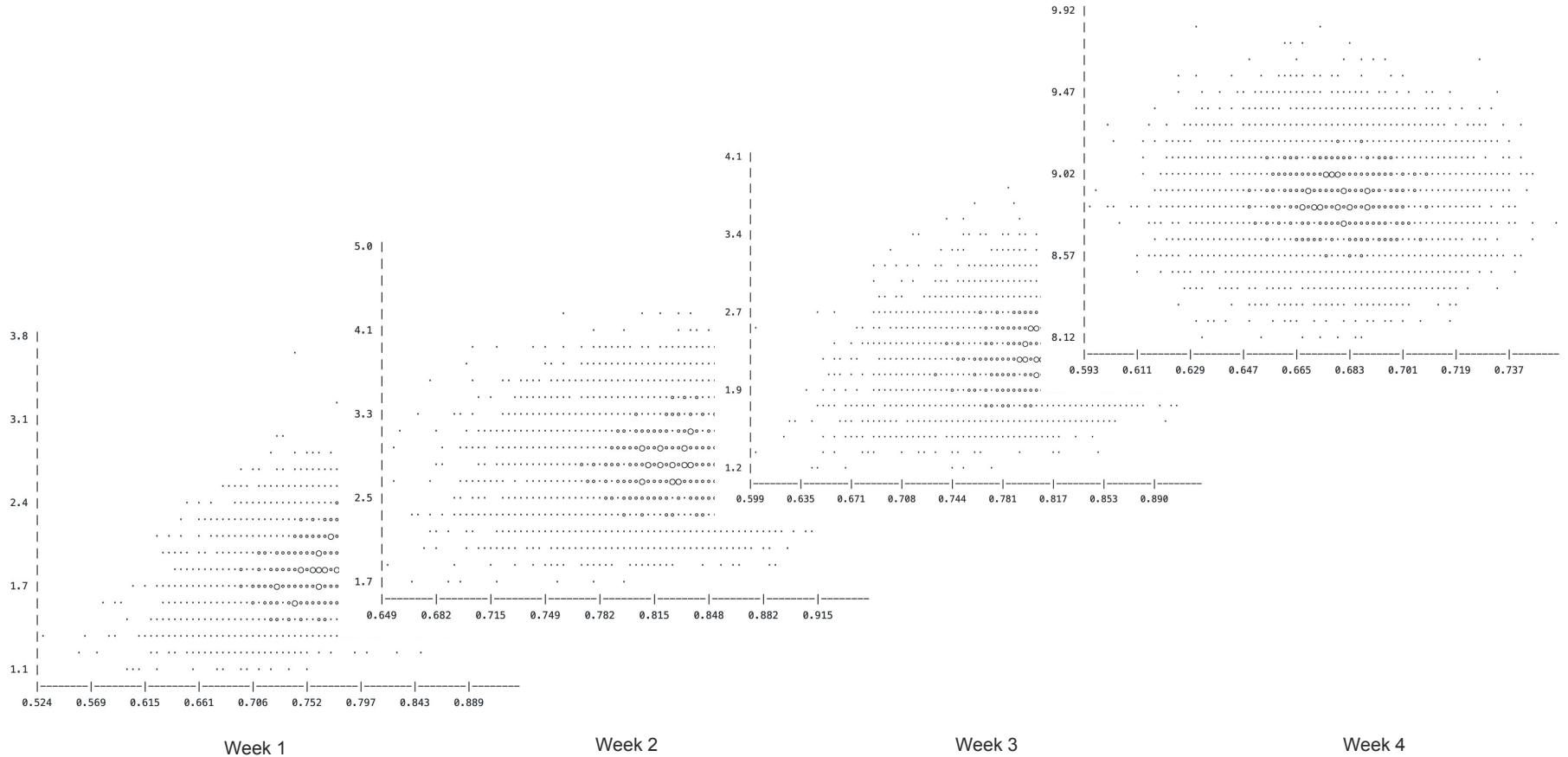
Anomaly  
Detection

# Part 4 Window Based

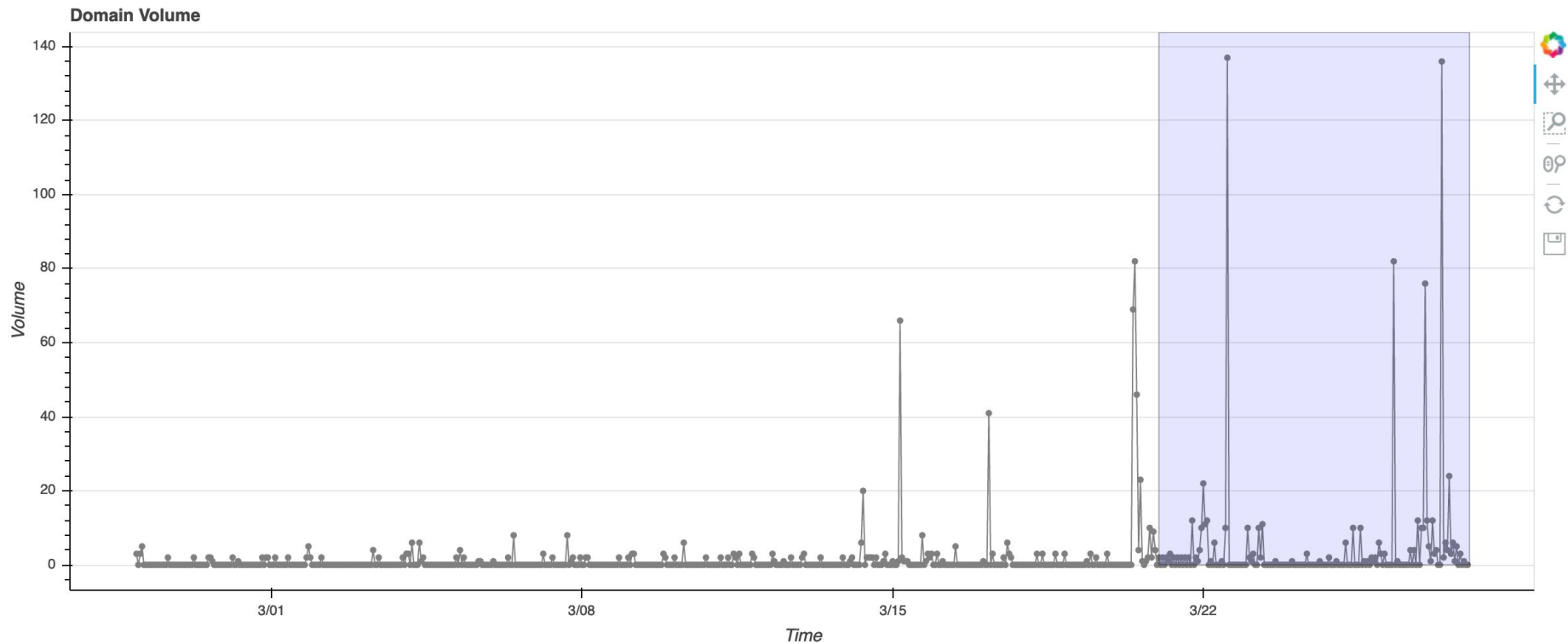




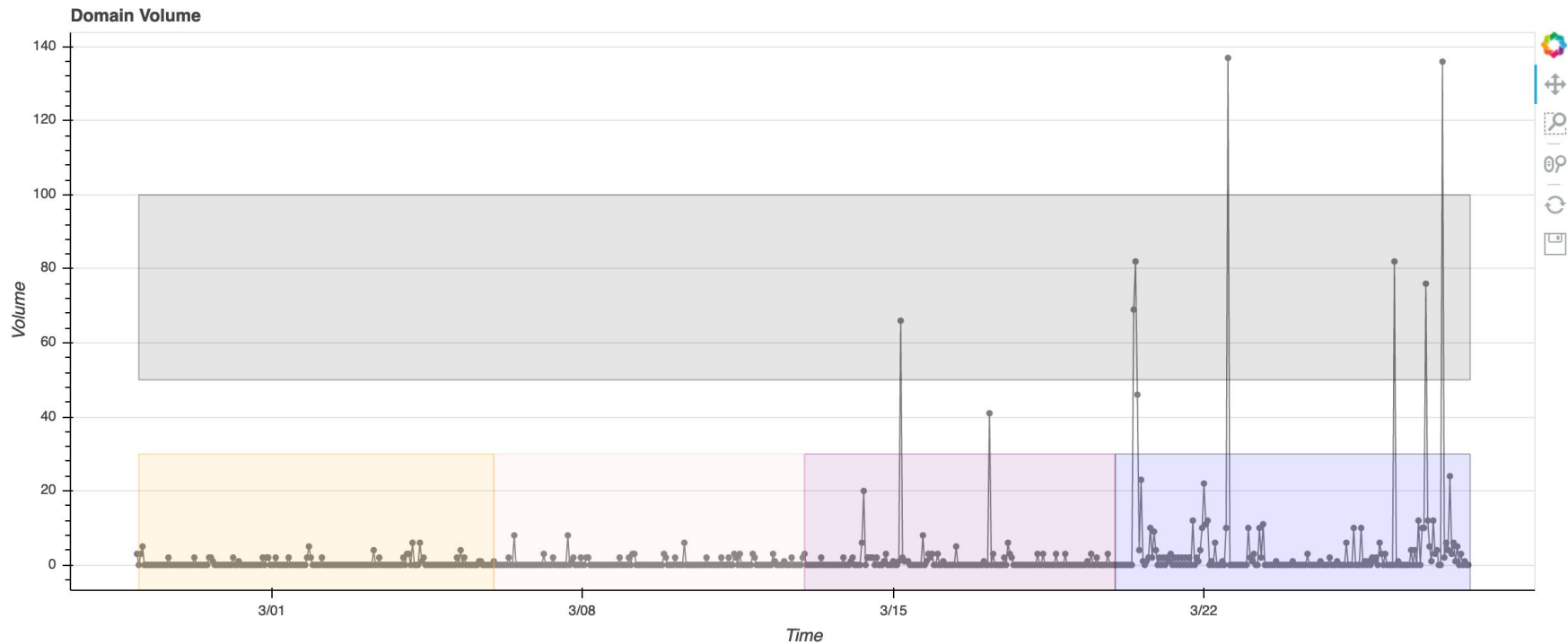
# Part 4 Window Simulations



# Part 4 Outlier Window



## Part 4 Local Outlier to Global



# Closing Recap



Masquerading  
DNS  
Traffic



Time Series  
Modeling



Rainier  
+  
Spark



Anomaly  
Detection

# Closing Glossed Over Details



Outliers



Goodness  
of  
Fit

# Closing References

A Review of Croston's method for intermittent demand forecasting

[https://www.researchgate.net/publication/254044245\\_A\\_Review\\_of\\_Croston's\\_method\\_for\\_intermittent\\_demand\\_forecasting](https://www.researchgate.net/publication/254044245_A_Review_of_Croston's_method_for_intermittent_demand_forecasting)

Rainier

<https://github.com/stripe/rainier>

PyMC3

<https://docs.pymc.io/>

Emotet

<https://www.us-cert.gov/ncas/alerts/TA18-201A>

Bokeh Plots

<https://bokeh.pydata.org/en/latest/>

Twitter Chill

<https://github.com/twitter/chill>

# Closing Contact

## Website

[davidrdgz.github.io](https://davidrdgz.github.io)

## Github

[@davidrdgz](https://github.com/davidrdgz)

## Twitter

[@davidrdgz](https://twitter.com/davidrdgz)

## Email

davrodr3 at cisco.com