TU/e Technische Universiteit Eindhoven University of Technology

# *ForneyLab.jl*

# a Julia Toolbox for Factor Graph-based Probabilistic Programming

JuliaCon 2018

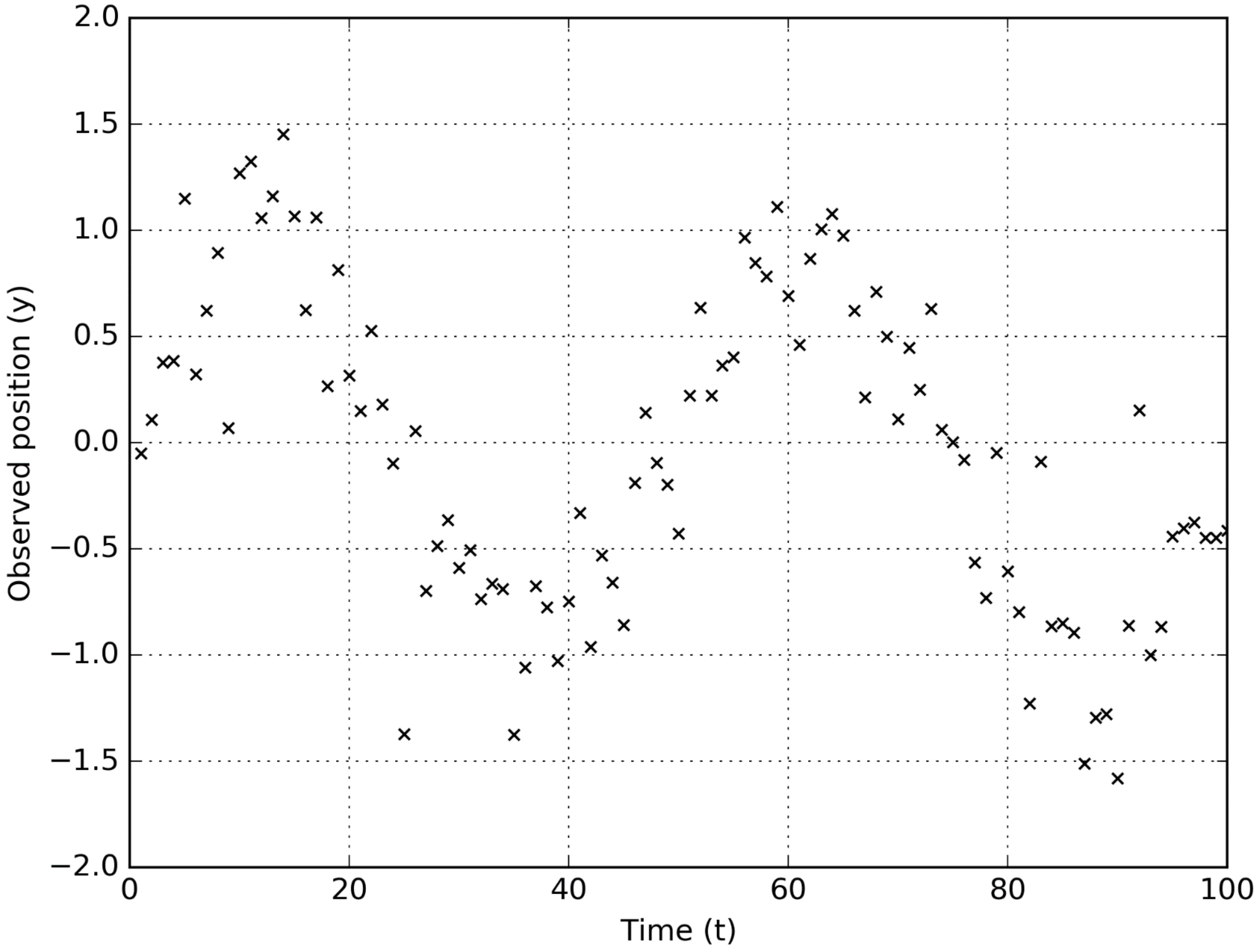
Thijs van de Laar
TU/e

Marco Cox
TU/e

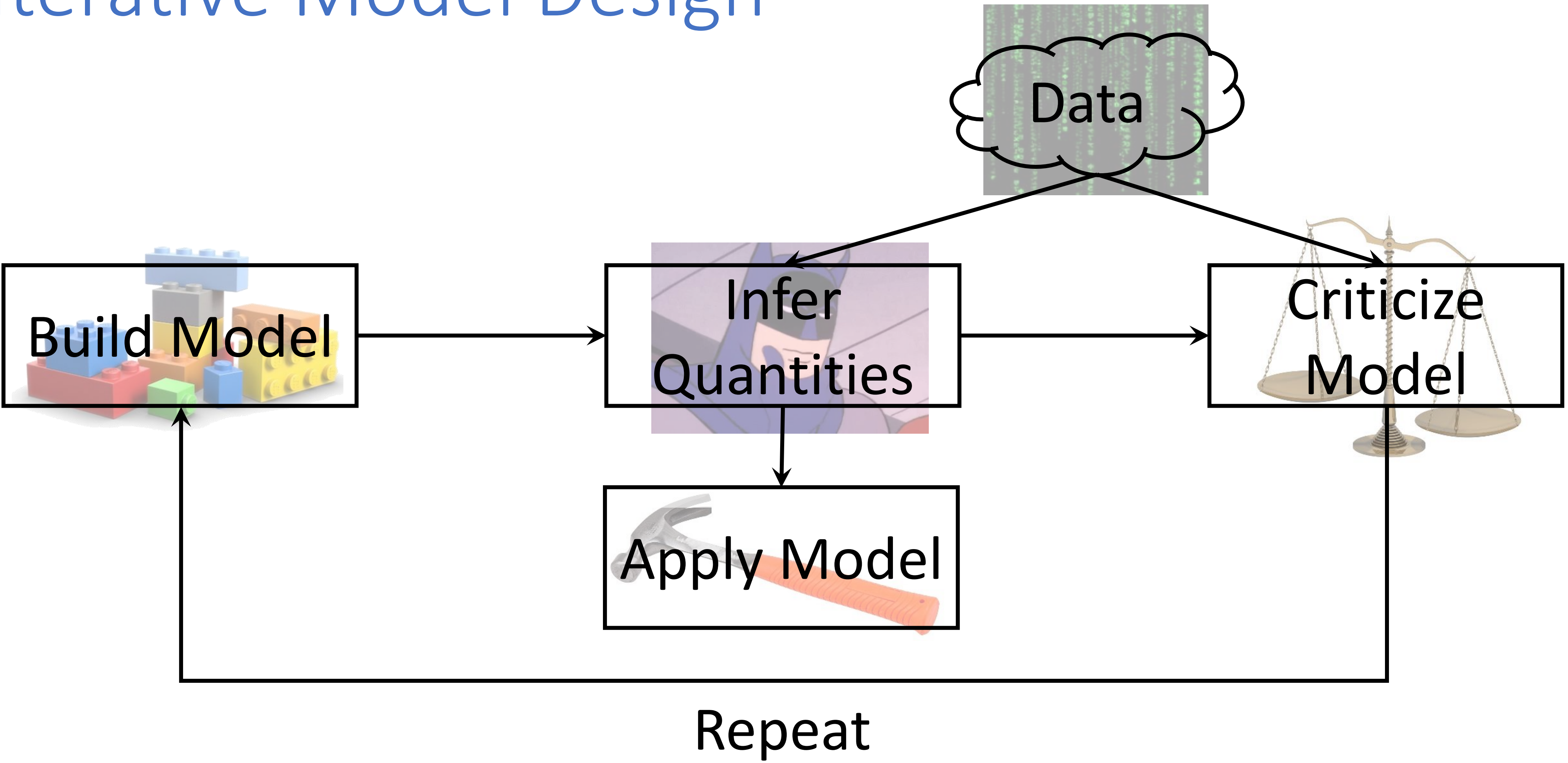
Bert de Vries
TU/e GN Hearing

BIASLAB

# Example Dataset

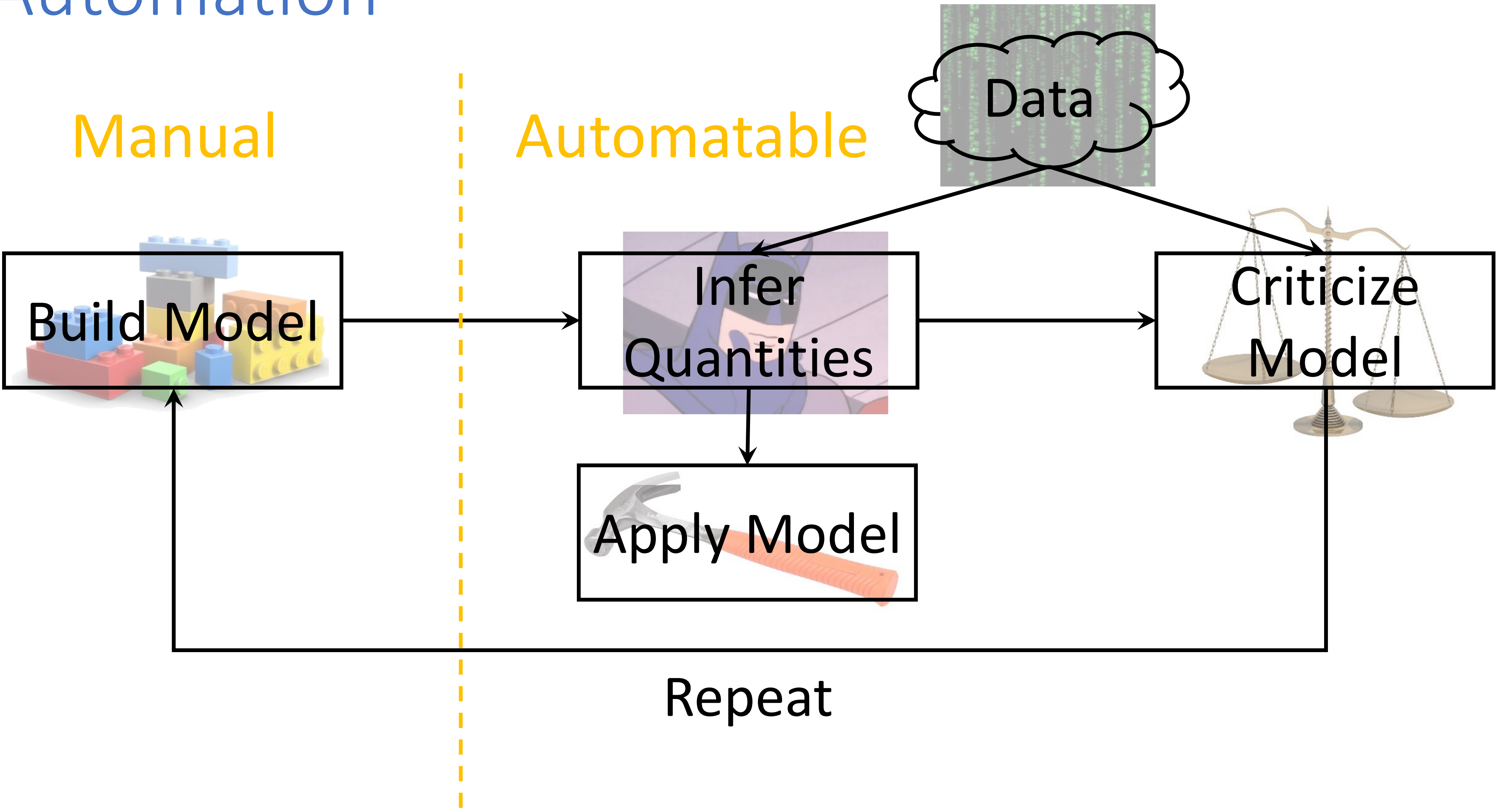


# Iterative Model Design

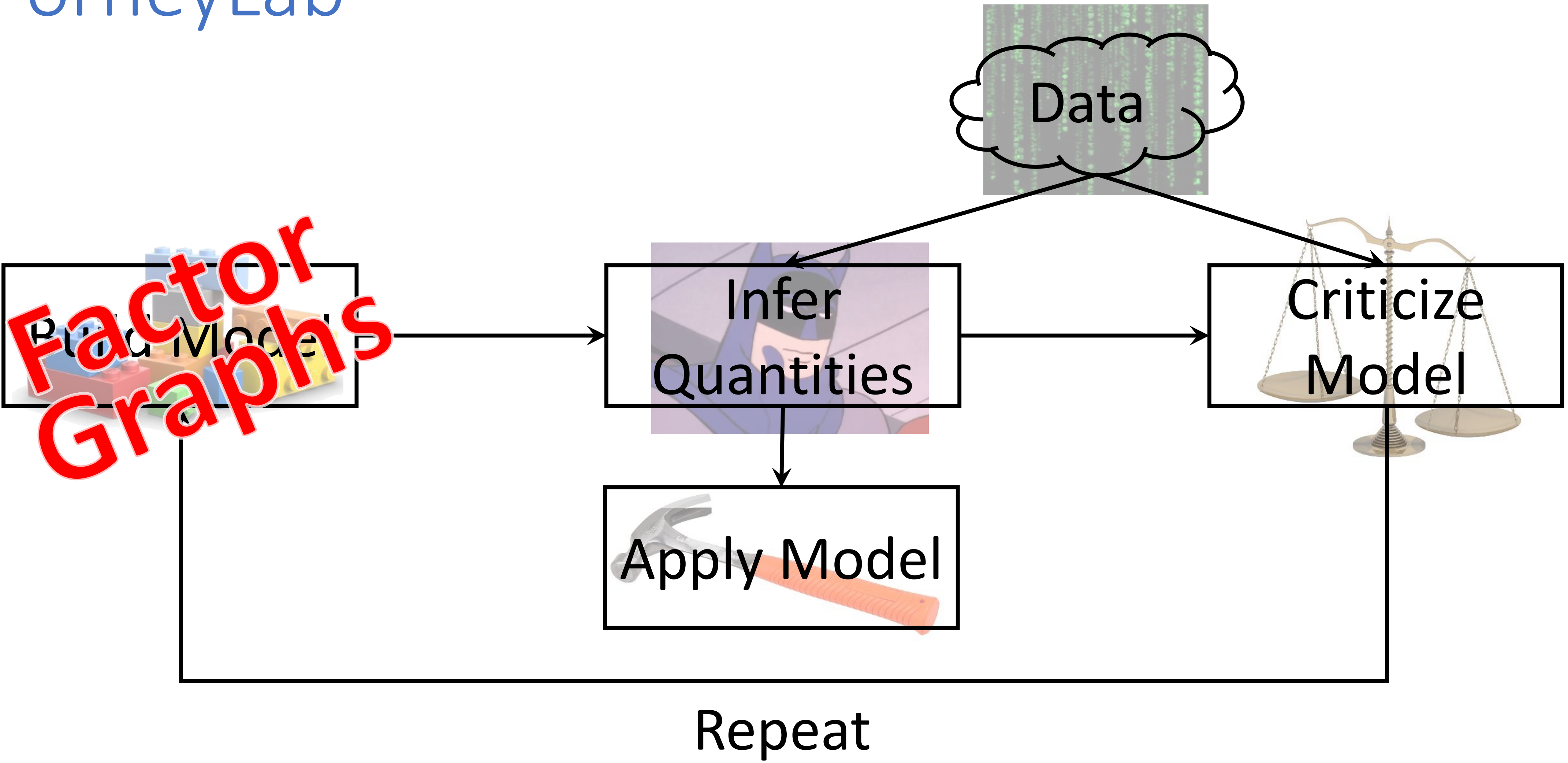


Blei, D. M. (2014). Build, compute, critique, repeat: Data analysis with latent variable models. *Annual Review of Statistics and Its Application*, *1*, 203-232.

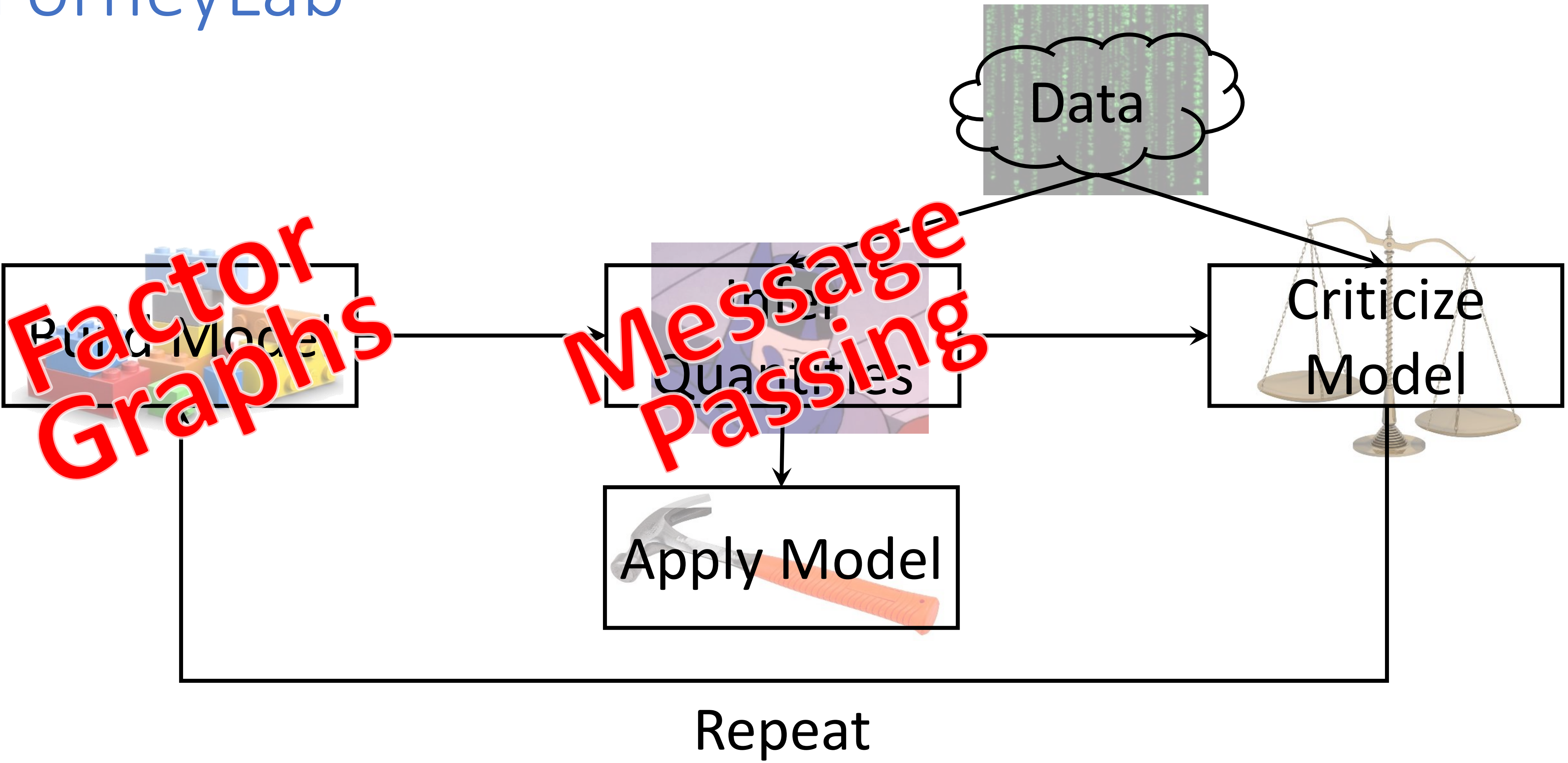
# Automation


Manual
Automatable
Data
Build Model
Infer Quantities
Criticize Model
Apply Model
Repeat

# ForneyLab


Data
Build Model
Factor Graphs
Infer Quantities
Criticize Model
Apply Model
Repeat

# ForneyLab



Data

Build Model

Infer Quantities

Criticize Model

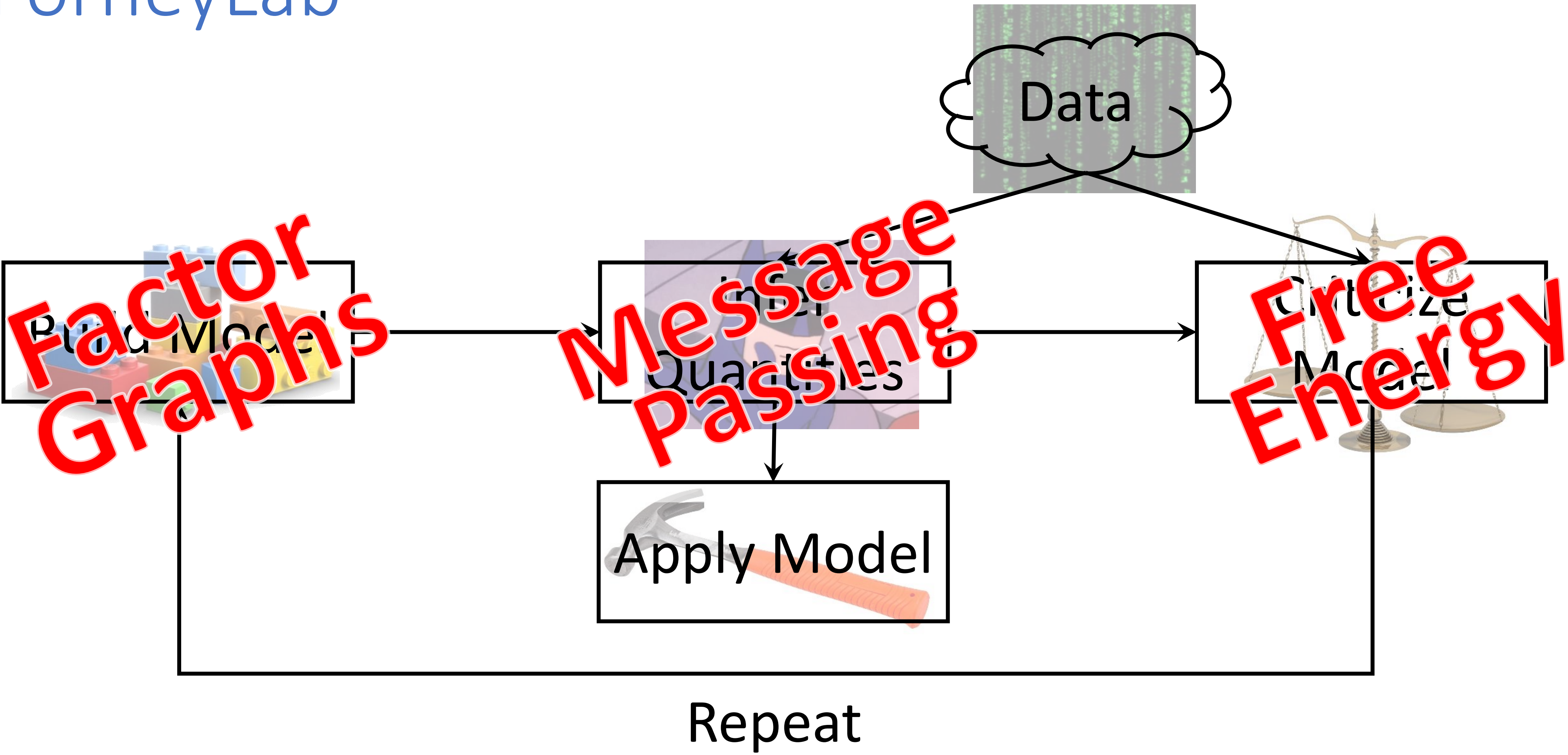
Apply Model

Repeat

Factor Graphs

Message Passing

# ForneyLab



Data

Build Model

Infer Quantities

Criticize Model

Apply Model

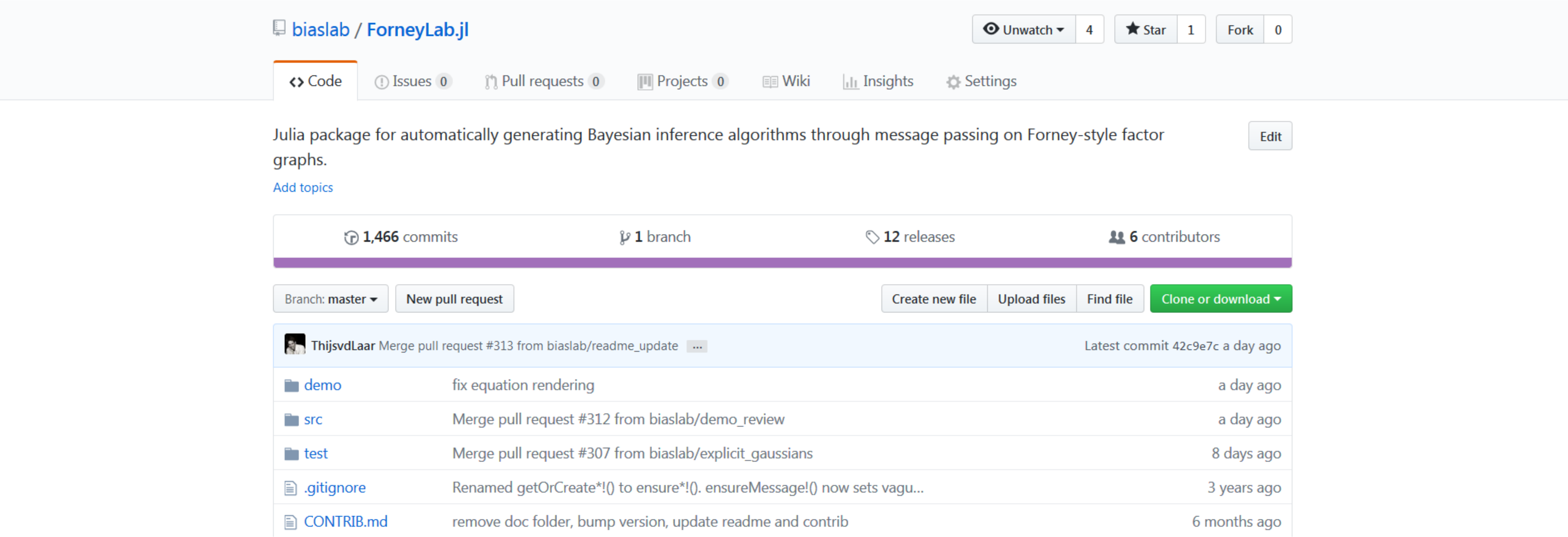
Repeat

Factor Graphs

Message Passing

Free Energy

# ForneyLab



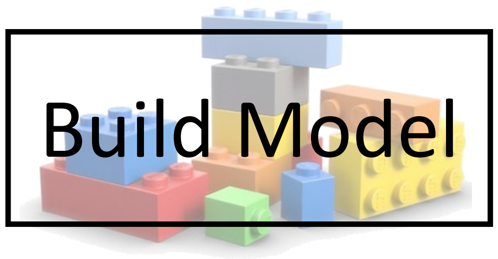
Julia package for automatically generating Bayesian inference algorithms through message passing on Forney-style factor graphs.

ThijsvdLaar Merge pull request #313 from biaslab/readme_update … Latest commit 42c9e7c a day ago

| | | |
|---|---|---|
| demo | fix equation rendering | a day ago |
| src | Merge pull request #312 from biaslab/demo_review | a day ago |
| test | Merge pull request #307 from biaslab/explicit_gaussians | 8 days ago |
| .gitignore | Renamed getOrCreate*!() to ensure*!(). ensureMessage!() now sets vagu… | 3 years ago |
| CONTRIB.md | remove doc folder, bump version, update readme and contrib | 6 months ago |

*https://github.com/biaslab/ForneyLab.jl*

# Model Specification



Build Model

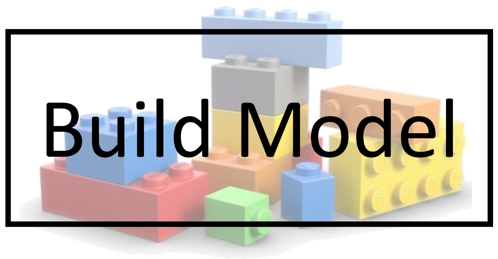
*Prior:* $x_0 \sim \mathcal{N}_p(0, 0.04)$

*State transition model:* $x_t \sim \mathcal{N}_p(x_{t-1}, 100)$

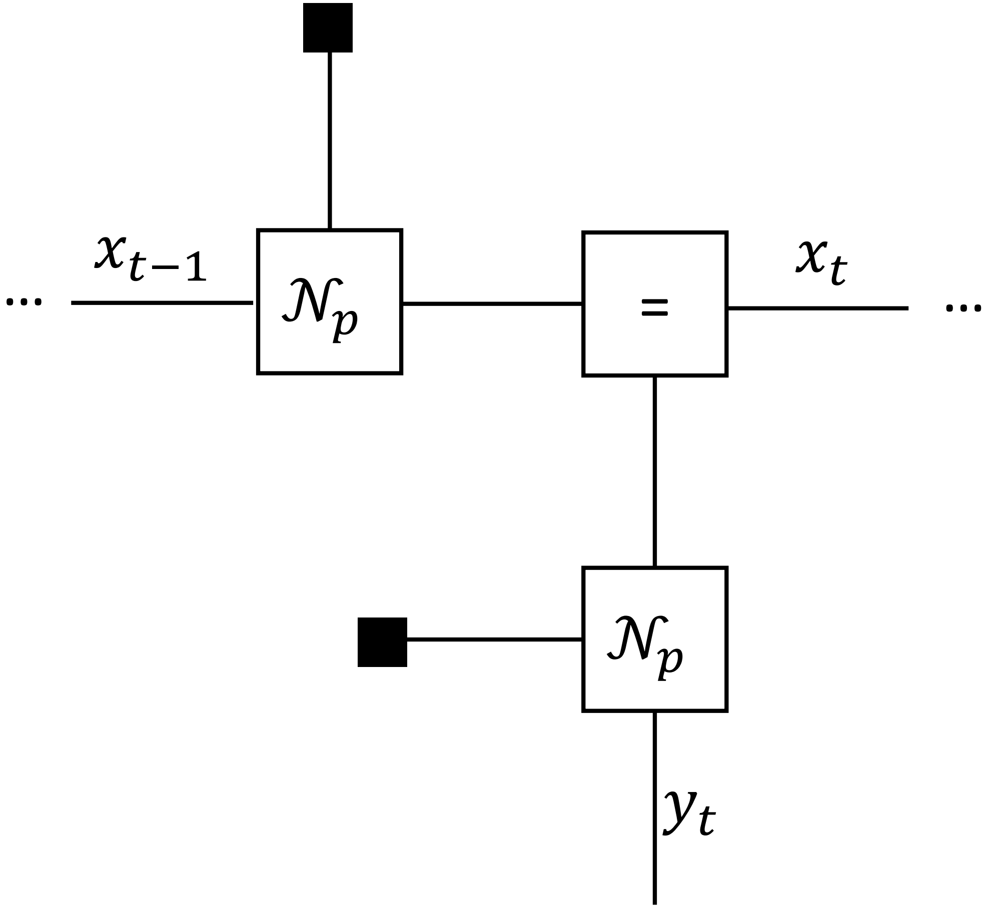
*Observation model:* $y_t \sim \mathcal{N}_p(x_t, 10)$

```
@RV x_0 ~ GaussianMeanPrecision(0.0, 0.04) # State prior

x_t_min = x_0
for t=1:T
    @RV x[t] ~ GaussianMeanPrecision(x_t_min, 100.0) # State transition model
    @RV y[t] ~ GaussianMeanPrecision(x[t], 10.0) # Observation model
    placeholder(y[t], :y, index=t) # Placeholder for data

    x_t_min = x[t] # Reset state for next section
end
```

# Factor Graph Representation



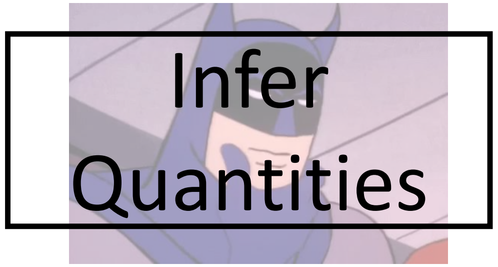
Build Model



Forney, G. D. (2001). Codes on graphs: Normal realizations. *IEEE Transactions on Information Theory*, *47*(2), 520-548.

Loeliger, H. A. (2004). An introduction to factor graphs. *IEEE Signal Processing Magazine*, *21*(1), 28-41.

# Inference Specification

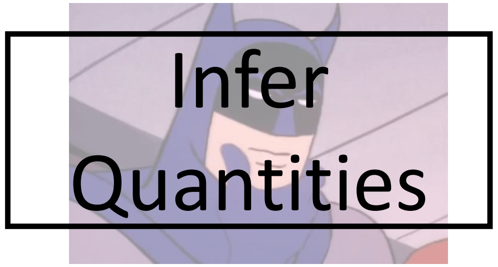


Infer Quantities

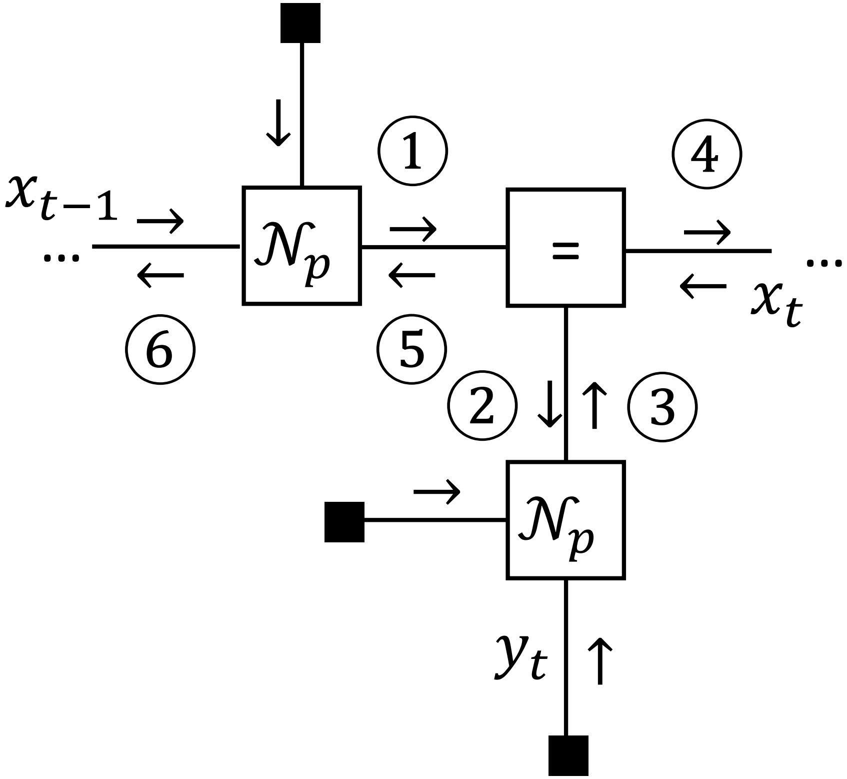
```
q = RecognitionFactorization([x_0; x], ...) # Specify a recognition distribution
algo = variationalAlgorithm(q) # Construct the inference algorithm
```

Dauwels, J. (2007, June). On variational message passing on factor graphs. In *Information Theory, 2007. ISIT 2007. IEEE International Symposium on* (pp. 2546-2550). IEEE.

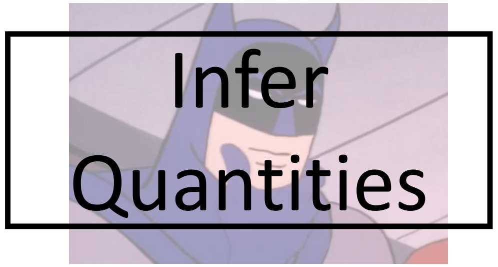
# Variational Message Passing



```
q = RecognitionFactorization([x_0; x], ...) # Specify a recognition distribution
algo = variationalAlgorithm(q) # Construct the inference algorithm
```



# Automated Algorithm Generation

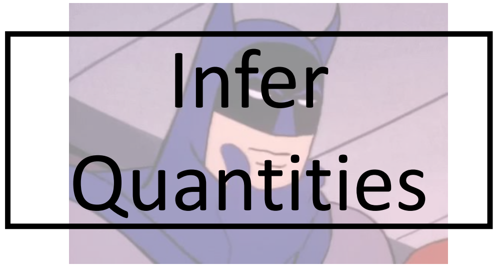


Infer Quantities

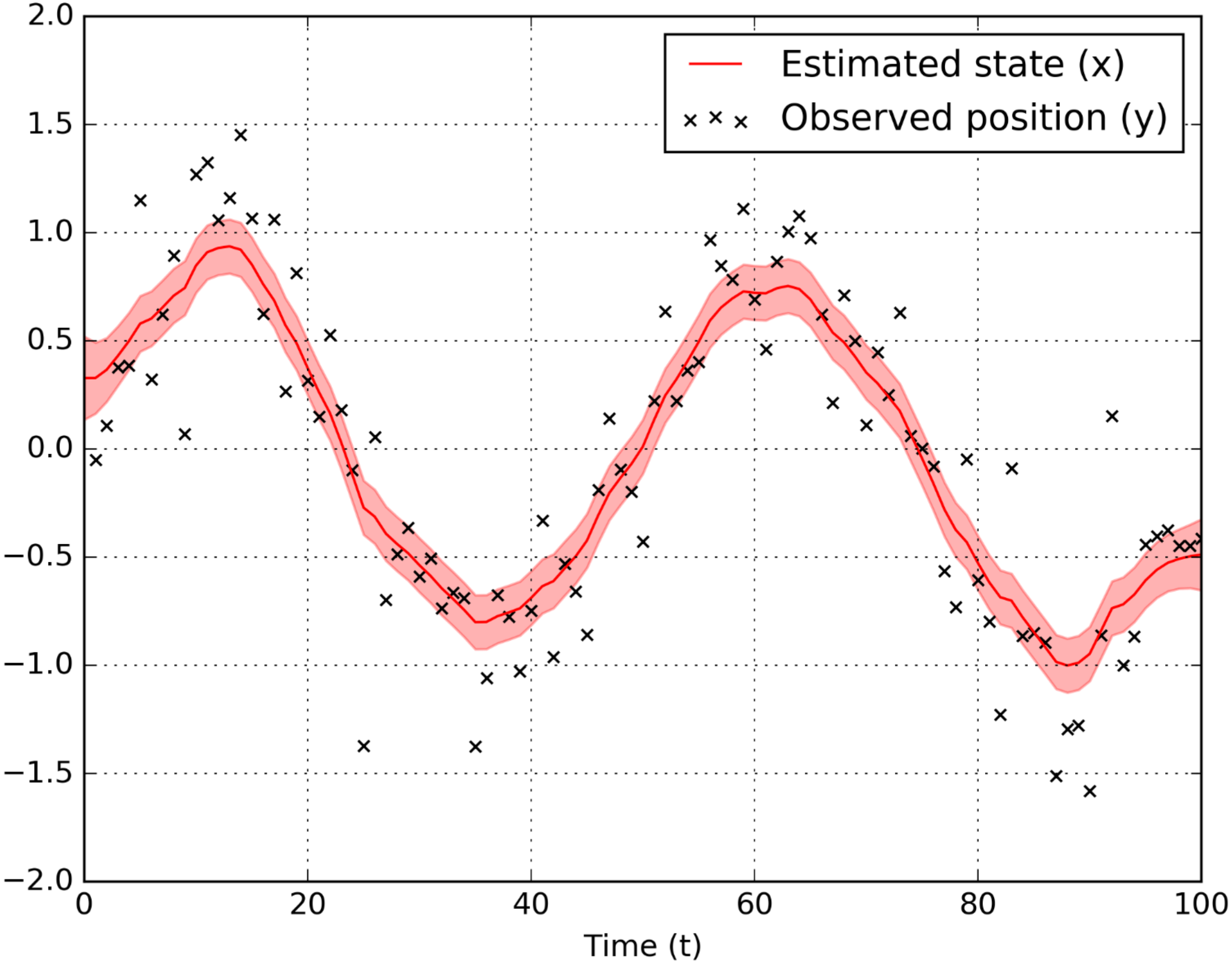
```
q = RecognitionFactorization([x_0; x], ...) # Specify a recognition distribution
algo = variationalAlgorithm(q) # Construct the inference algorithm
```

```
function step!(data::Dict, marginals::Dict=Dict(),
messages::Vector{Message}=Array{Message}(499))

  messages[1] = ruleVBGaussianMeanPrecisionM(ProbabilityDistribution(Univariate,
  PointMass, m=data[:y][50]), nothing, ProbabilityDistribution(Univariate,
  PointMass, m=10.0))
  ...
  messages[499] = ruleSVBGaussianMeanPrecisionMGVD(messages[498], nothing,
  ProbabilityDistribution(Univariate, PointMass, m=100.0))

  marginals[:x_0] = messages[3].dist * messages[499].dist
  ...
  return marginals
end
```

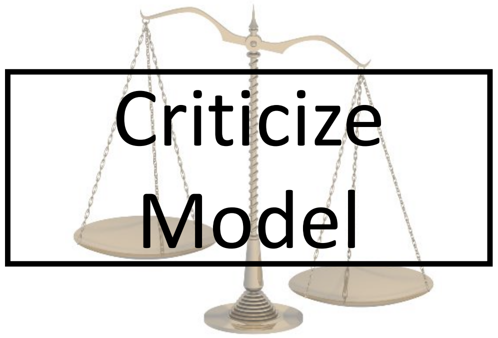
# Inference Results



Infer Quantities



# Model Performance

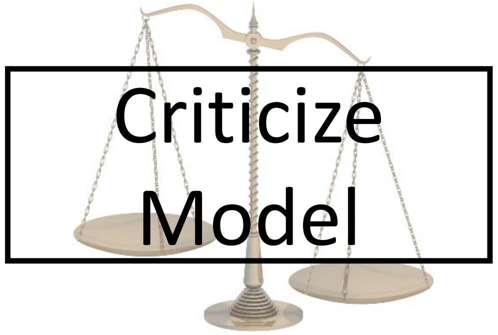


Criticize Model

```
algo_F = freeEnergyAlgorithm(q) # Construct a performance evaluation metric
```

Attias, H. (2000). A variational bayesian framework for graphical models. In *Advances in neural information processing systems* (pp. 209-215).

# Automated Performance Evaluation

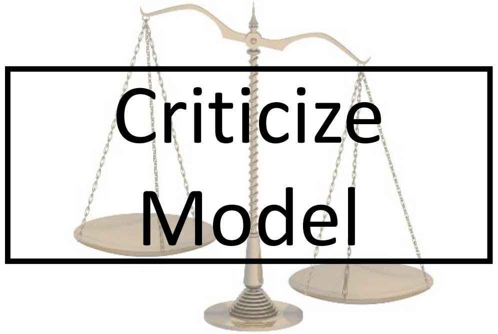


Criticize Model

```
algo_F = freeEnergyAlgorithm(q) # Construct a performance evaluation metric
```

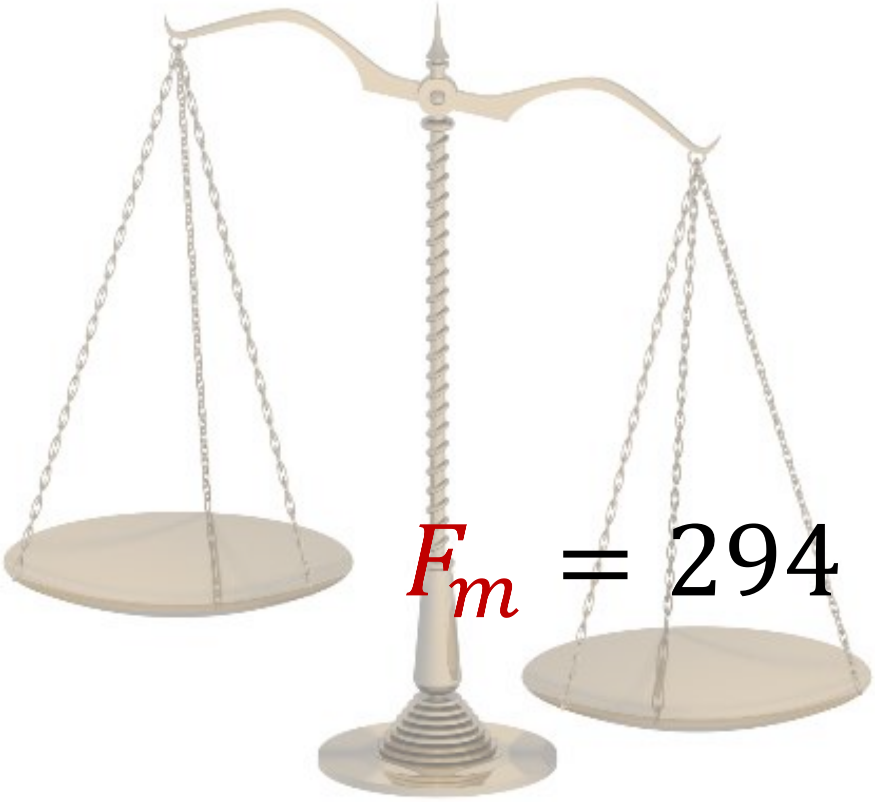
```
function freeEnergy(data::Dict, marginals::Dict)
  F = 0.0

  F += averageEnergy(GaussianMeanPrecision, marginals[:x_1_x_0],
  ProbabilityDistribution(Univariate, PointMass, m=100.0))
  F += averageEnergy(GaussianMeanPrecision, ProbabilityDistribution(Univariate,
  PointMass, m=data[:y][44]), marginals[:x_44],
  ProbabilityDistribution(Univariate, PointMass, m=10.0))
  ...
  F -= differentialEntropy(marginals[:x_0])

  return F
end
```

# Model Comparison



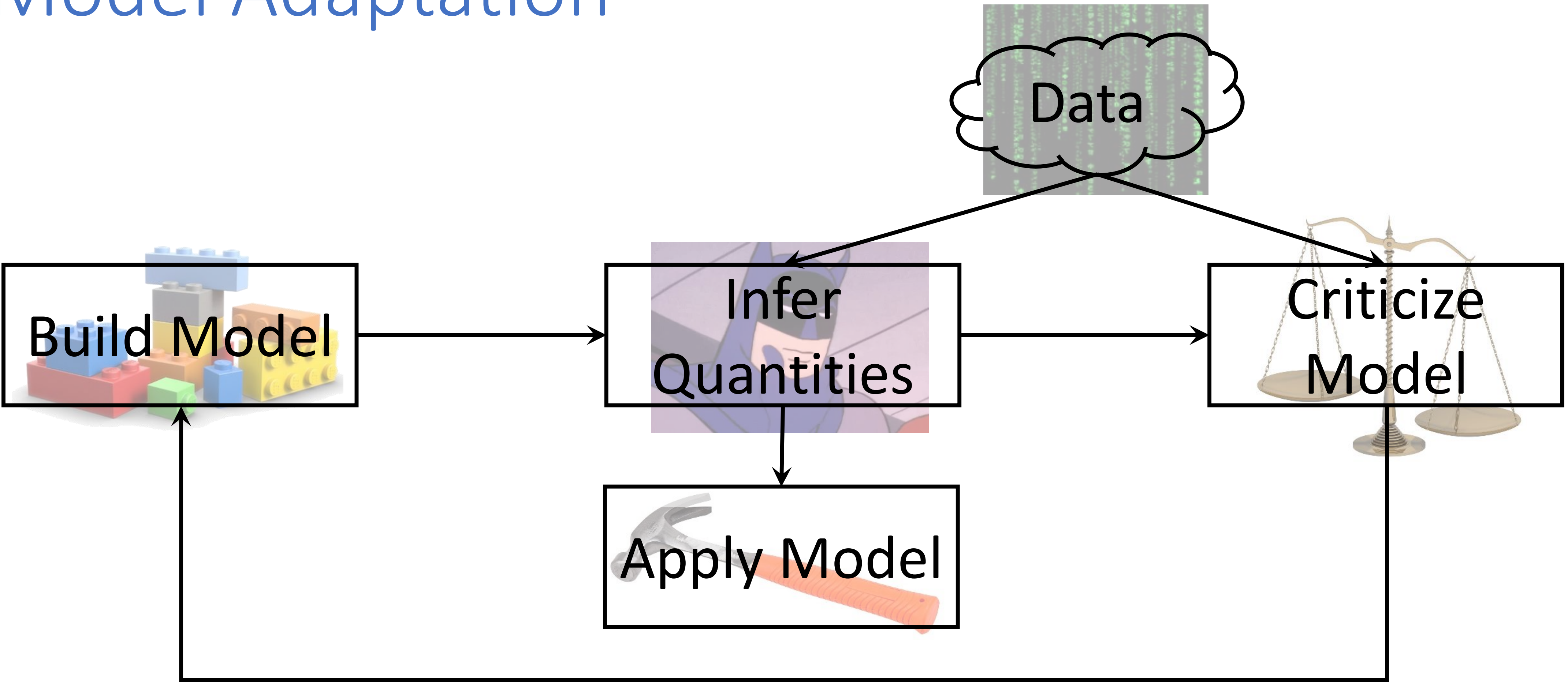
Criticize Model

Evaluate free energy (less is better)



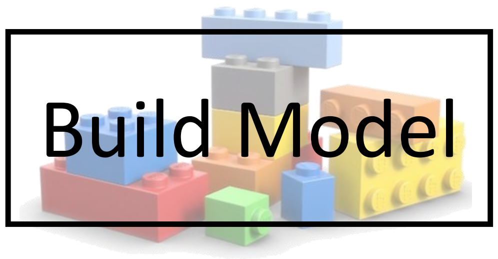
$$F_m = 294\ [dB]$$

# Model Adaptation



Repeat

# Model Adaptation



Build Model

*Prior:* $x_0 \sim \mathcal{N}_p(0, 0.04)$

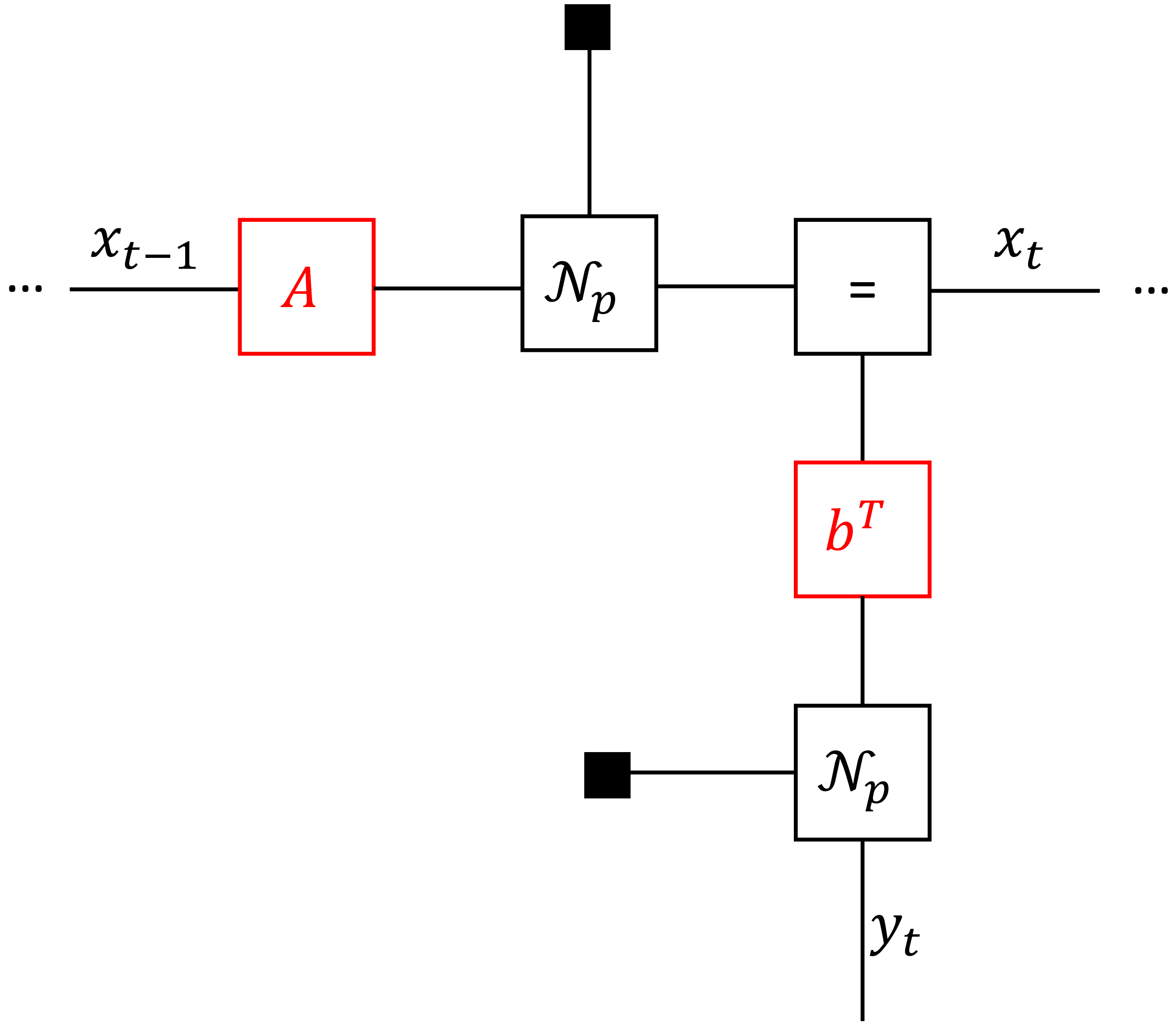
*State transition model:* $x_t \sim \mathcal{N}_p(Ax_{t-1}, 100)$

*Observation model:* $y_t \sim \mathcal{N}_p(b^T x_t, 10)$

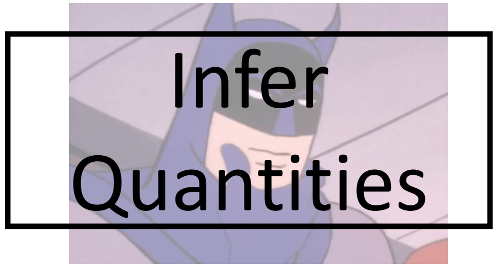
```
@RV x_0 ~ GaussianMeanPrecision(zeros(2), 0.04*eye(2)) # State prior

x_t_min = x_0
for t=1:T
    @RV x[t] ~ GaussianMeanPrecision(A*x_t_min, 100.0*eye(2)) # Transition model
    @RV y[t] ~ GaussianMeanPrecision(dot(b, x[t]), 10.0*eye(2)) # Obs. model
    placeholder(y[t], :y, index=t) # Placeholder for data

    x_t_min = x[t] # Reset state for next section
end
```

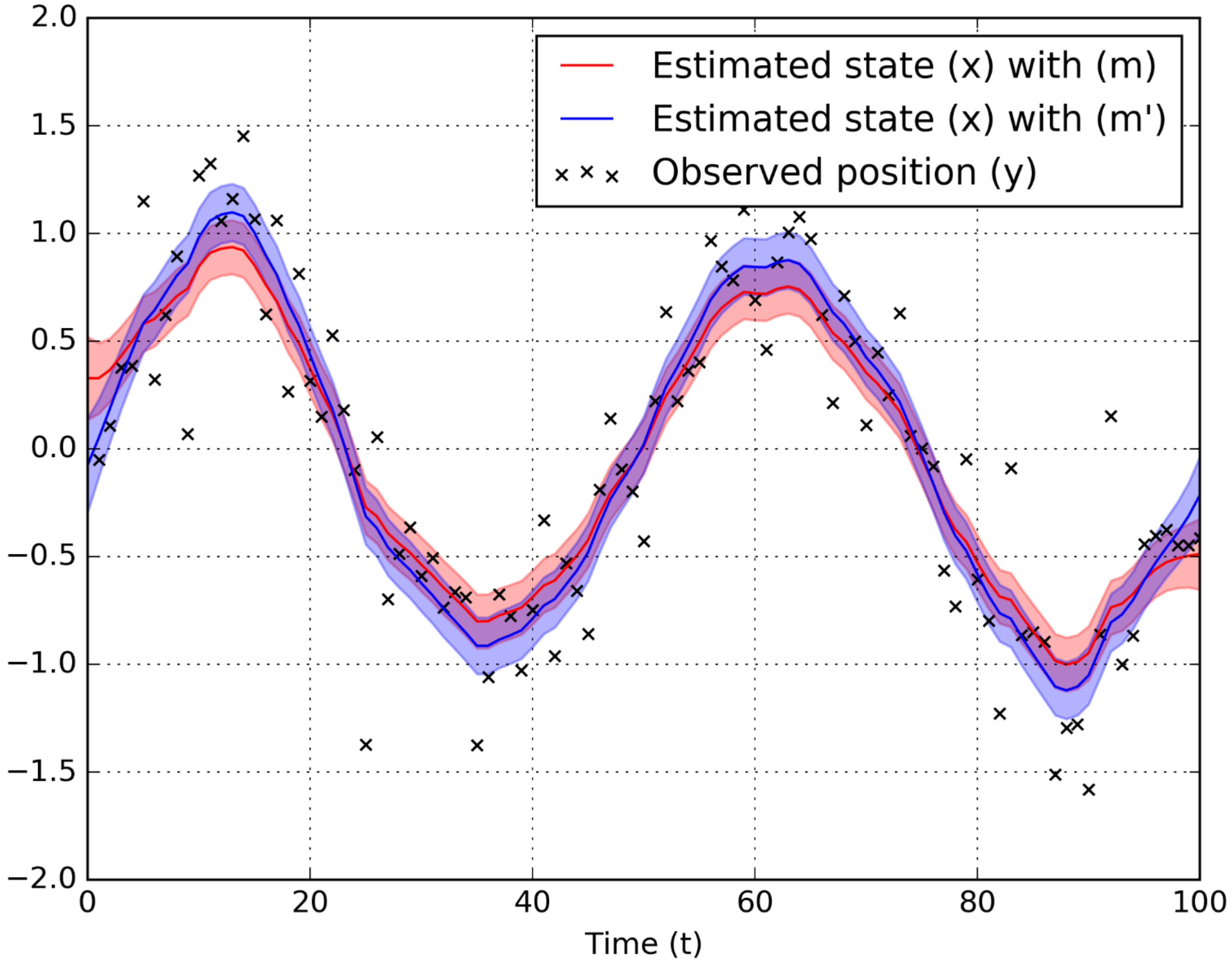
# Model Adaptation

Build Model

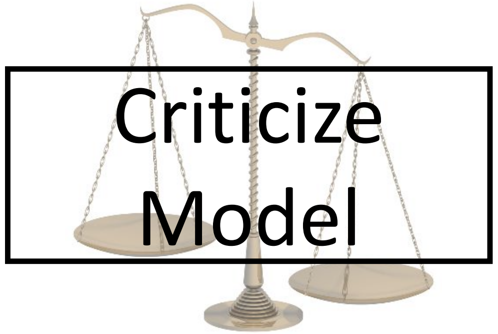


# Inference Results



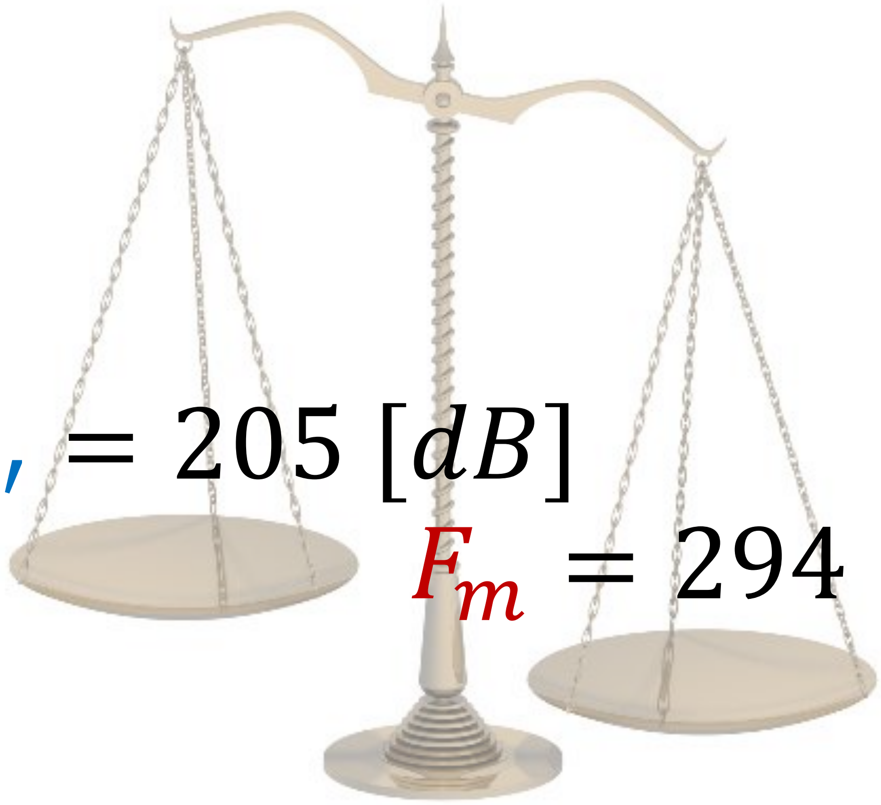


# Model Comparison



Criticize Model

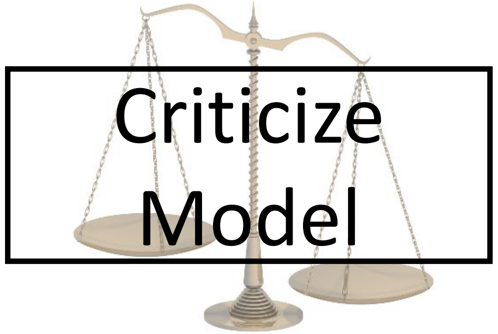
Evaluate free energy (less is better)



$$F_{m'} = 205\ [dB]$$

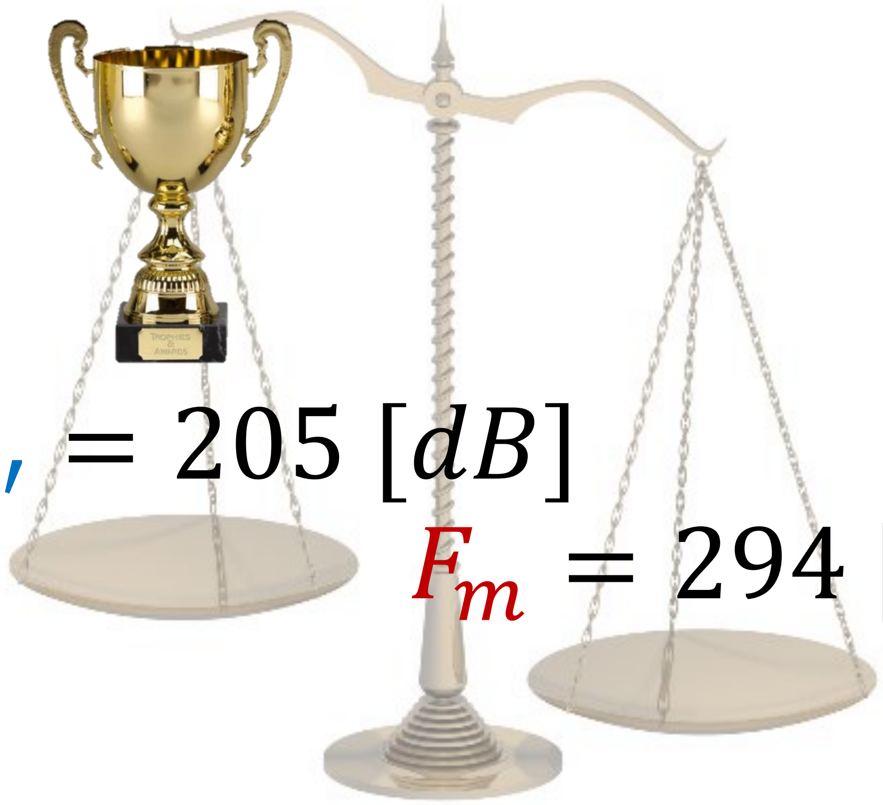
$$F_m = 294\ [dB]$$

# Model Comparison



Criticize Model

Evaluate free energy (less is better)



$$F_{m'} = 205\ [dB]$$

$$F_m = 294\ [dB]$$

# ForneyLab

- Enhances the probabilistic model design cycle
- Is a Julia program that writes Julia programs
- Is available on GitHub

# Thanks

*Ivan Bocharov*

*Anouk van Diepen*

*Joris Kraak*

*Ismail Senoz*

*Tjalling Tjalkens*

*Wouter van Roosmalen*