CS423: Probabilistic Programming
Introduction

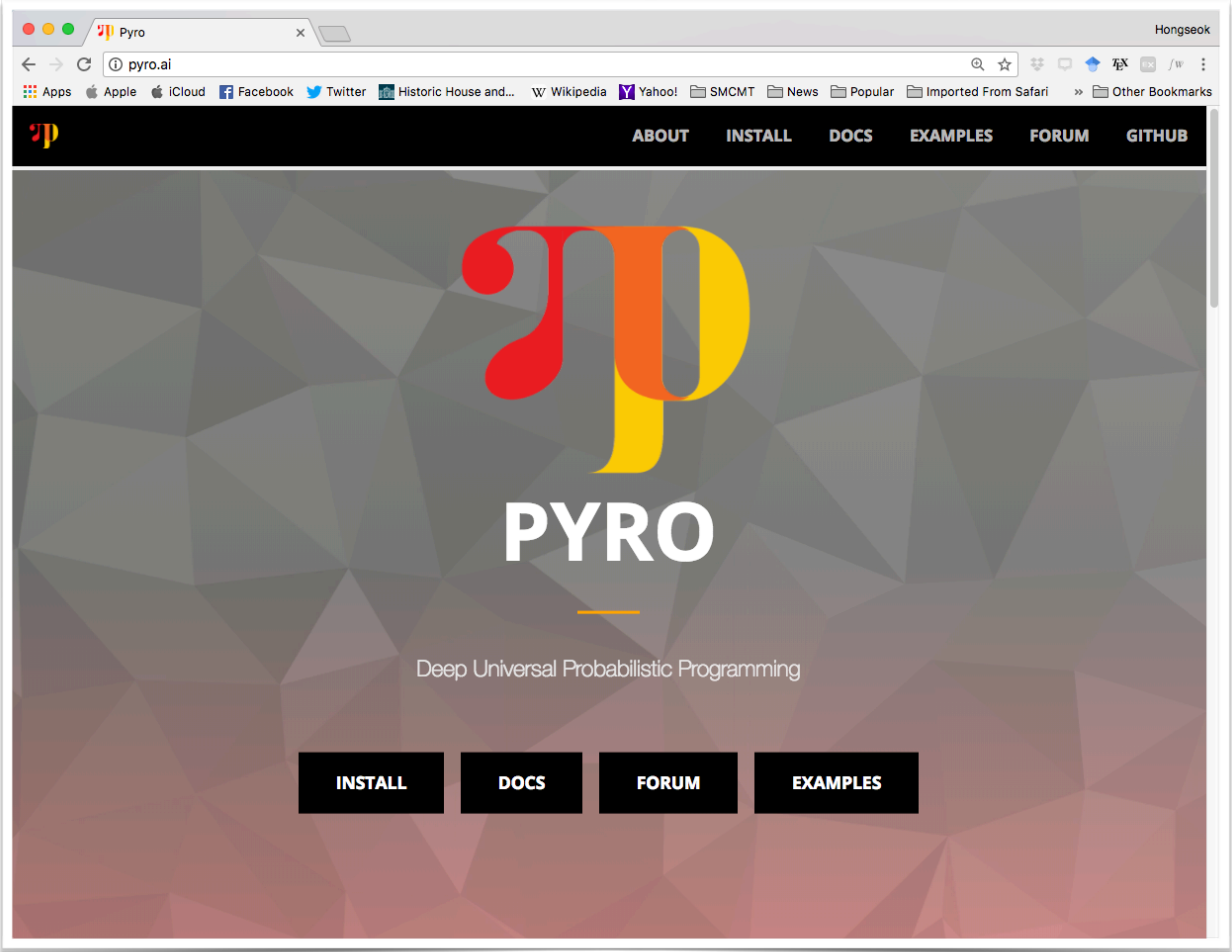
Hongseok Yang
KAIST

Can we capture the origins of human common sense … in engineering terms? … New tools from probabilistic programming, game engines and program learning … are beginning to let us answer these questions.

Josh Tenenbaum
Invited talk at ICML'18 & IJCAI'18

This Review … discusses some of the state-of-the-art advances in the field, namely, probabilistic programming, Bayesian optimization, data compression and automatic model discovery.

Zoubin Gharahmani
2015 Nature Review


Pyro
Hongseok
pyro.ai
Apps Apple iCloud Facebook Twitter Historic House and... Wikipedia Yahoo! SMCMT News Popular Imported From Safari » Other Bookmarks
ABOUT INSTALL DOCS EXAMPLES FORUM GITHUB
PYRO
Deep Universal Probabilistic Programming
INSTALL DOCS FORUM EXAMPLES

What is probabilistic programming?

(Bayesian) probabilistic modelling of data

1. Develop a new probabilistic (generative) model.
2. Design an inference algorithm for the model.
3. Using the algo., fit the model to the data.

(Bayesian) probabilistic modelling of data

in a prob. prog. language

1. Develop a new probabilistic (generative) model.
2. Design an inference algorithm for the model.
3. Using the algo., fit the model to the data.

(Bayesian) probabilistic modelling of data

in a prob. prog. language

as a program

1. Develop a new probabilistic (generative) model.
2. Design an inference algorithm for the model.
3. Using the algo., fit the model to the data.

(Bayesian) probabilistic modelling of data

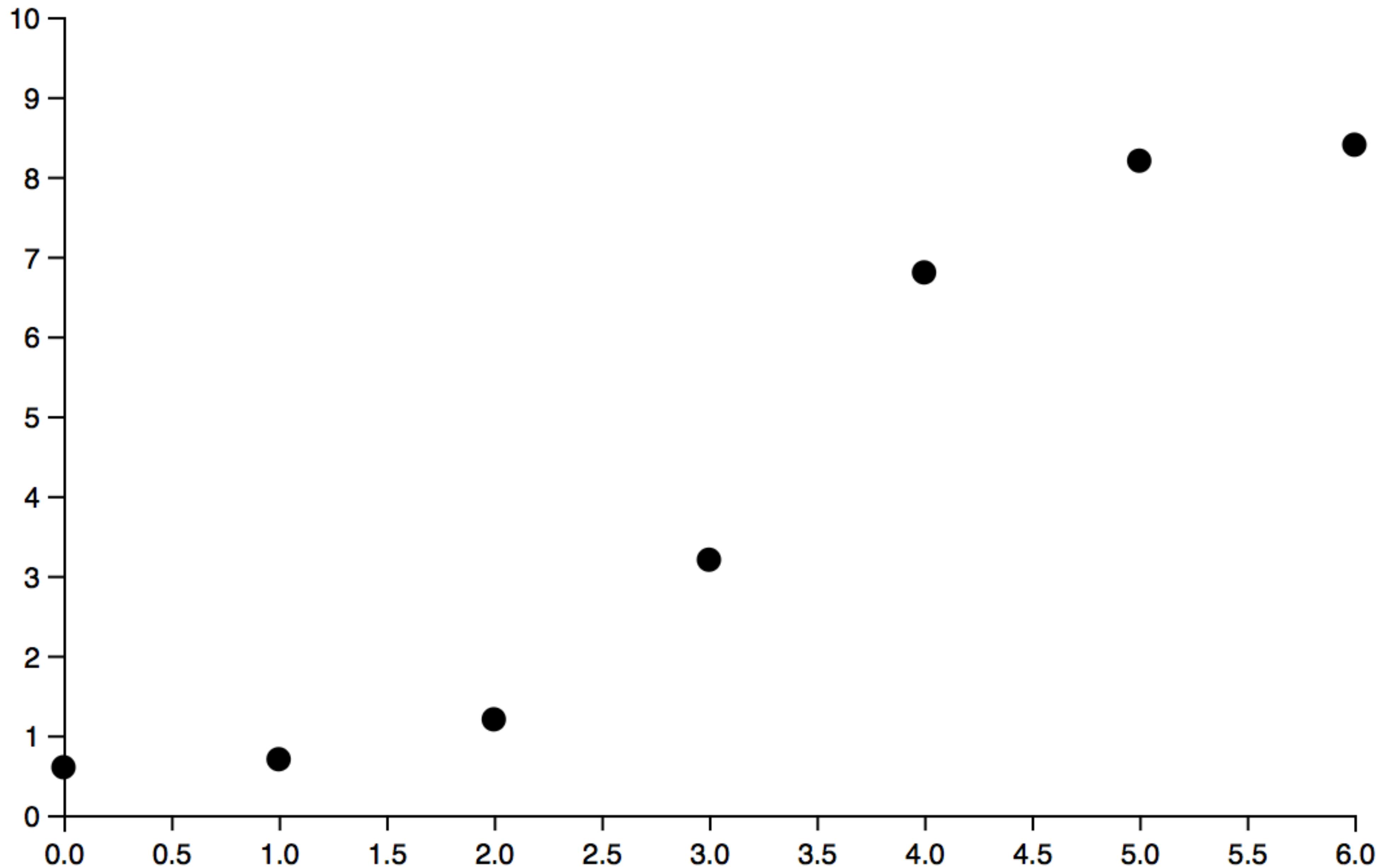
in a prob. prog. language

as a program

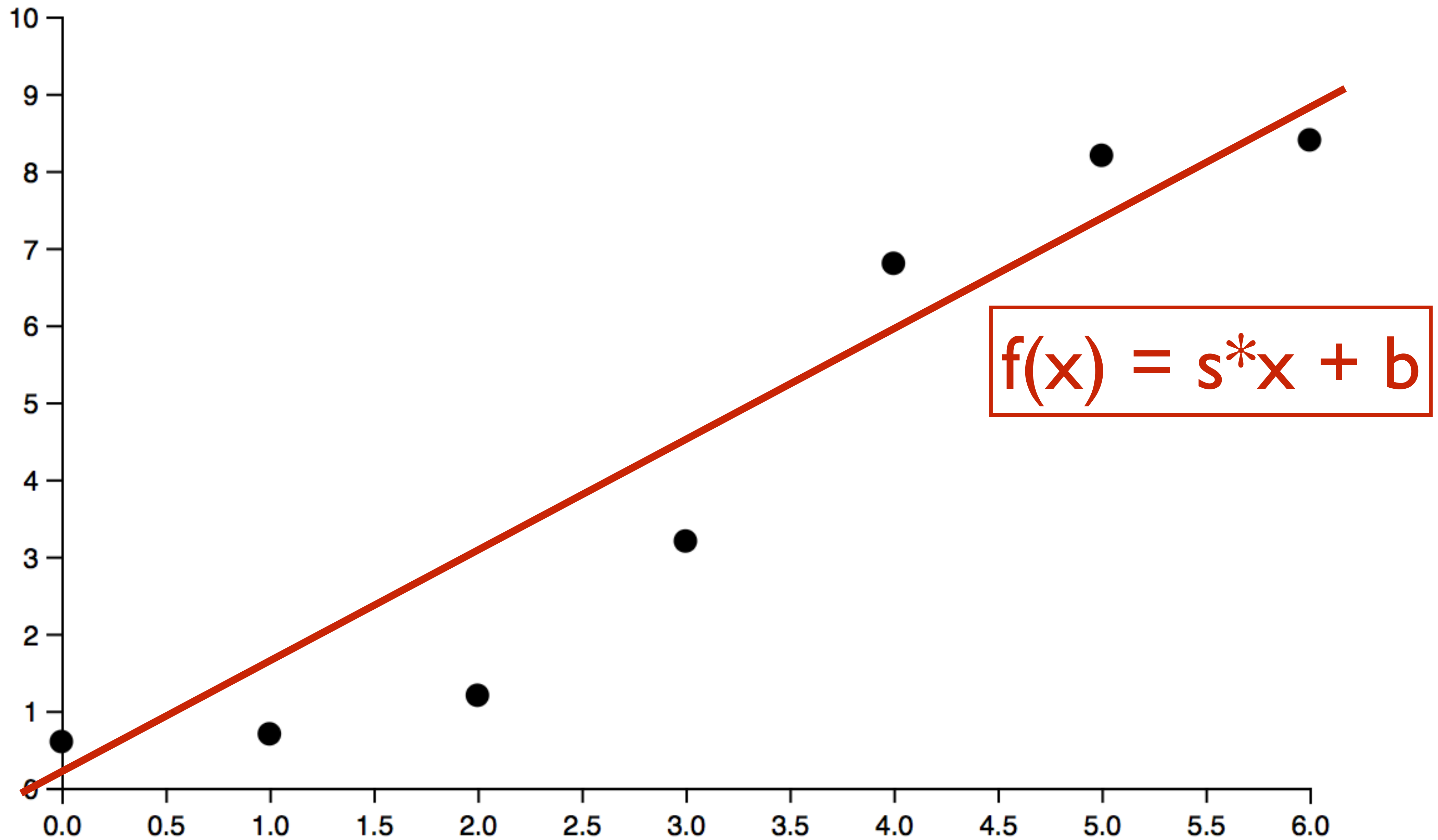
1. Develop a new probabilistic (generative) model.
2. ~~Design an inference algorithm for the model.~~
3. Using ~~the algo.~~, fit the model to the data.

a generic inference algo.
of the language

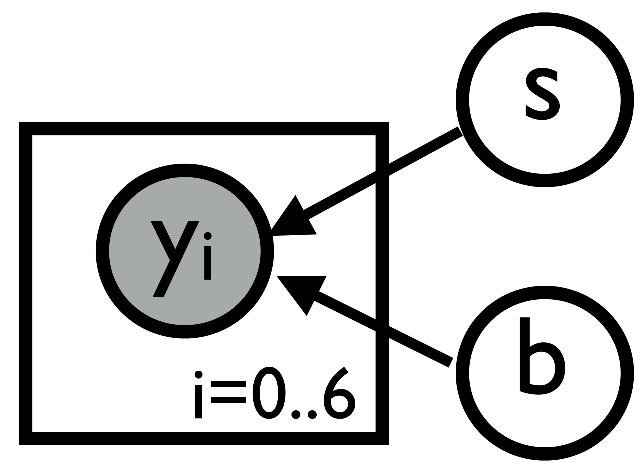
Line fitting



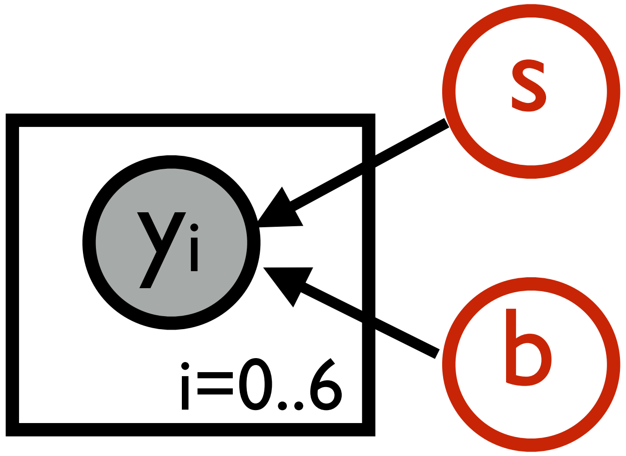
Line fitting



Bayesian generative model

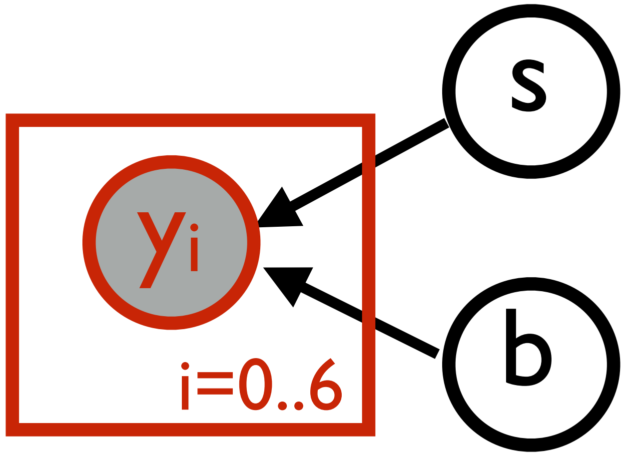


Bayesian generative model



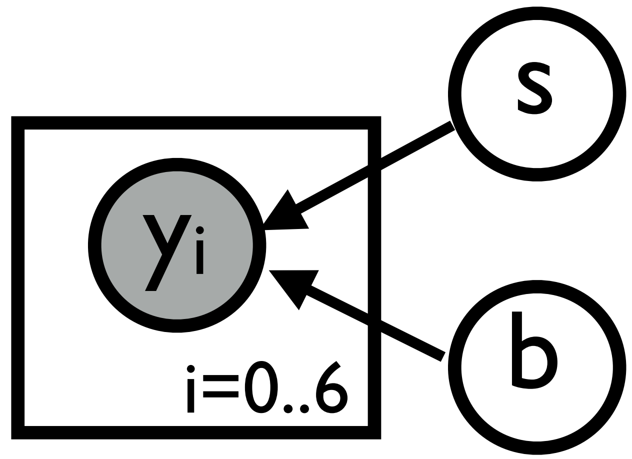

```
s      ~ normal(0, 2)
b      ~ normal(0, 6)
```

Bayesian generative model




```
s     ~ normal(0, 2)
b     ~ normal(0, 6)
f(x)  = s*x + b
yi     ~ normal(f(i), 0.5)
             where i = 0 .. 6
```

Bayesian generative model




```
s     ~ normal(0, 2)
b     ~ normal(0, 6)
f(x)  = s*x + b
yi     ~ normal(f(i), 0.5)
            where i = 0 .. 6
```

Q: posterior of (s,b) given y_0=0.6, …, y_6=8.4?

Posterior of s and b given y_i's

$$p(s, b \mid y_0, .., y_6) = \frac{p(y_0, .., y_6 \mid s,b) \times p(s,b)}{p(y_0, .., y_6)}$$

Posterior of s and b given y_i's

$$p(s, b \mid y_0, .., y_6) = \frac{p(y_0, .., y_6 \mid s,b) \times p(s,b)}{p(y_0, .., y_6)}$$

Posterior of s and b given y_i's

$$p(s, b \mid y_0, .., y_6) = \frac{p(y_0, .., y_6 \mid s,b) \times p(s,b)}{p(y_0, .., y_6)}$$

Posterior of s and b given y_i's

$$p(s, b \mid y_0, .., y_6) = \frac{p(y_0, .., y_6 \mid s,b) \times p(s,b)}{p(y_0, .., y_6)}$$

Posterior of s and b given y_i's

$$p(s, b \mid y_0, .., y_6) = \frac{p(y_0, .., y_6 \mid s,b) \times p(s,b)}{p(y_0, .., y_6)}$$

Anglican program

```
(let [s (sample (normal 0 2))
      b (sample (normal 0 6))
      f (fn [x] (+ (* s x) b))]
```

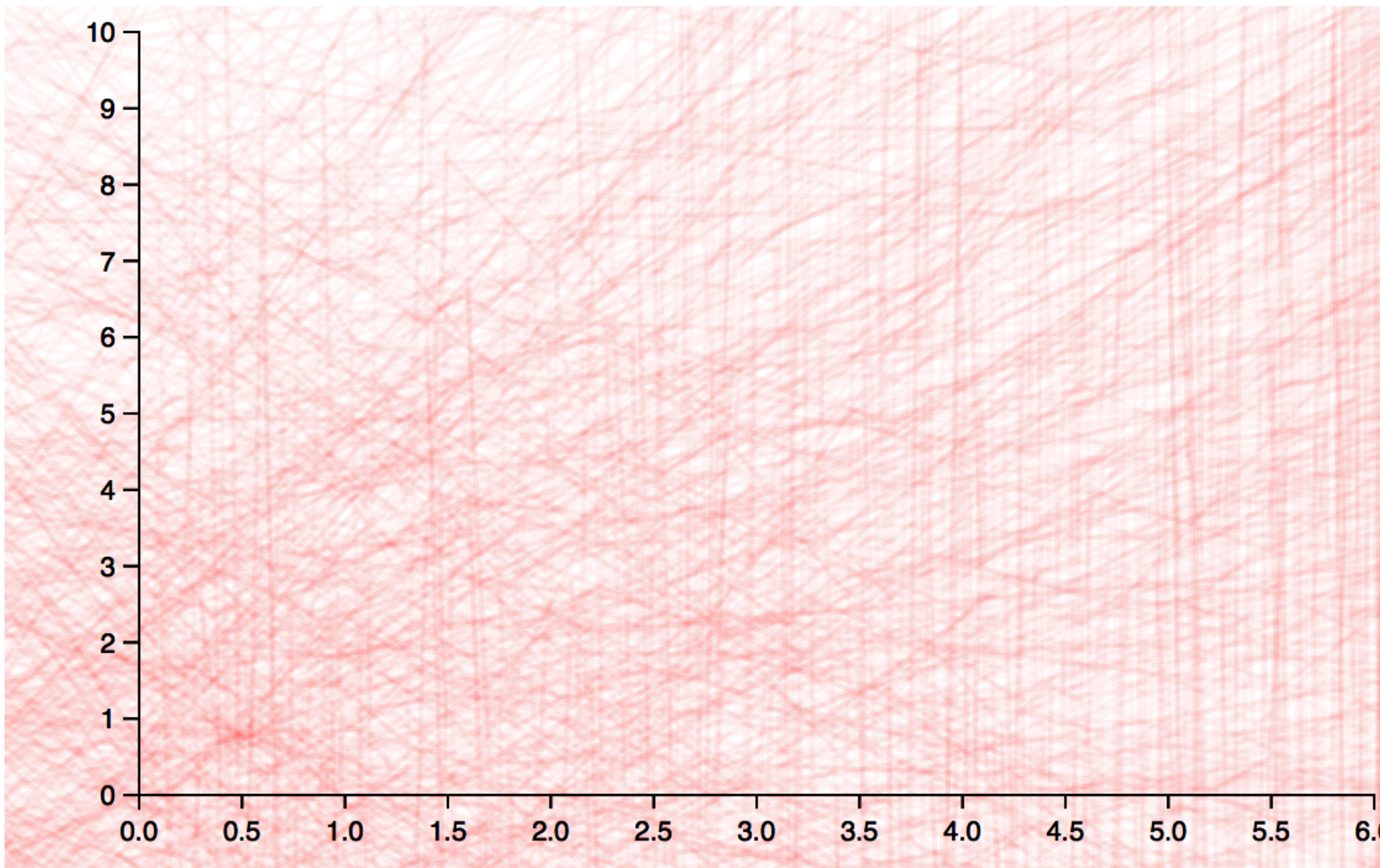
Anglican program

```
(let [s (sample (normal 0 2))
      b (sample (normal 0 6))
      f (fn [x] (+ (* s x) b))]

    (observe (normal (f 0) 0.5) 0.6)
    (observe (normal (f 1) 0.5) 0.7)
    (observe (normal (f 2) 0.5) 1.2)
    (observe (normal (f 3) 0.5) 3.2)
    (observe (normal (f 4) 0.5) 6.8)
    (observe (normal (f 5) 0.5) 8.2)
    (observe (normal (f 6) 0.5) 8.4)
```

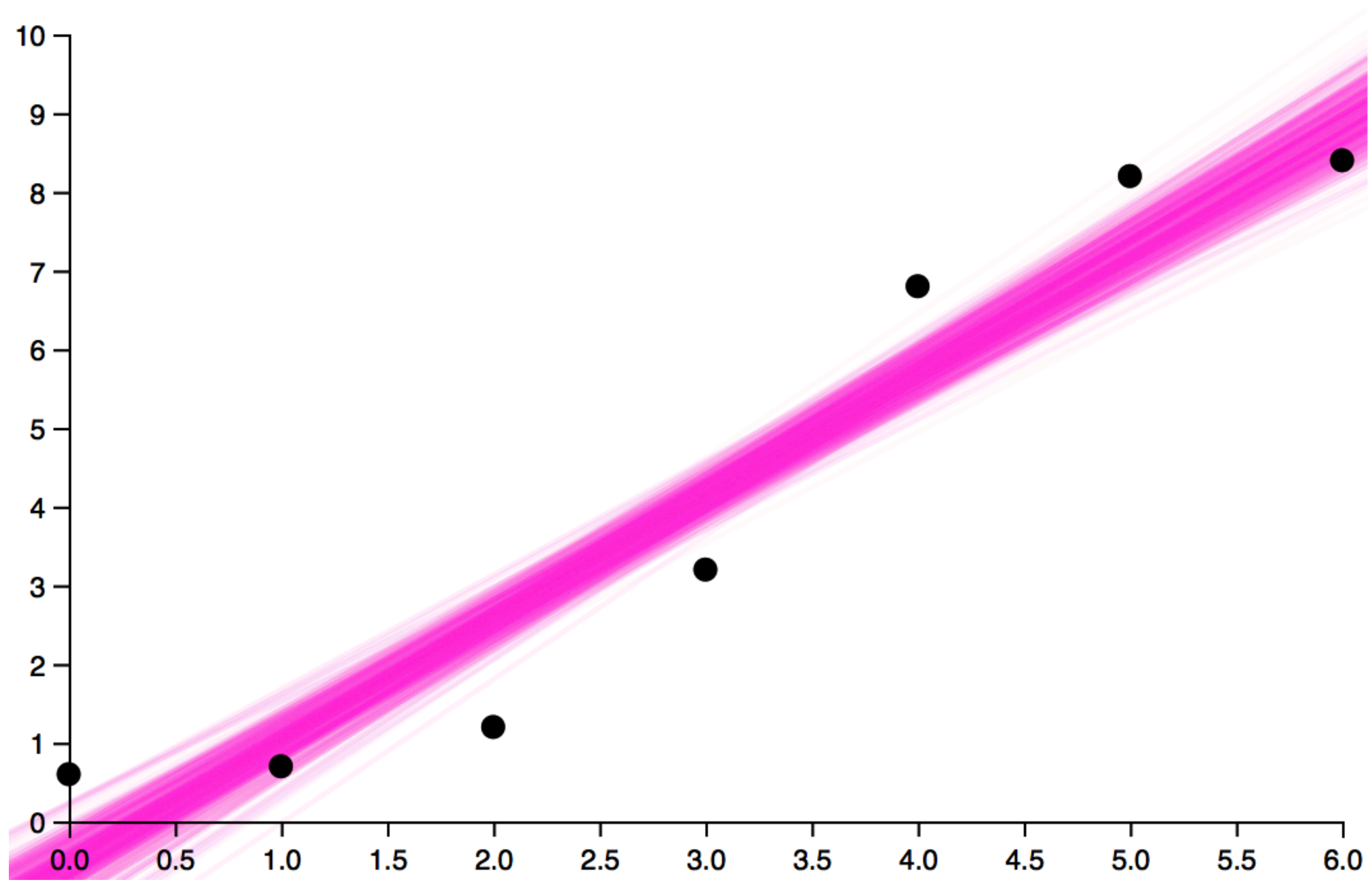
Anglican program

```
(let [s (sample (normal 0 2))
      b (sample (normal 0 6))
      f (fn [x] (+ (* s x) b))]

    (observe (normal (f 0) 0.5) 0.6)
    (observe (normal (f 1) 0.5) 0.7)
    (observe (normal (f 2) 0.5) 1.2)
    (observe (normal (f 3) 0.5) 3.2)
    (observe (normal (f 4) 0.5) 6.8)
    (observe (normal (f 5) 0.5) 8.2)
    (observe (normal (f 6) 0.5) 8.4)

  [s b])
```

Samples from prior



Samples from posterior

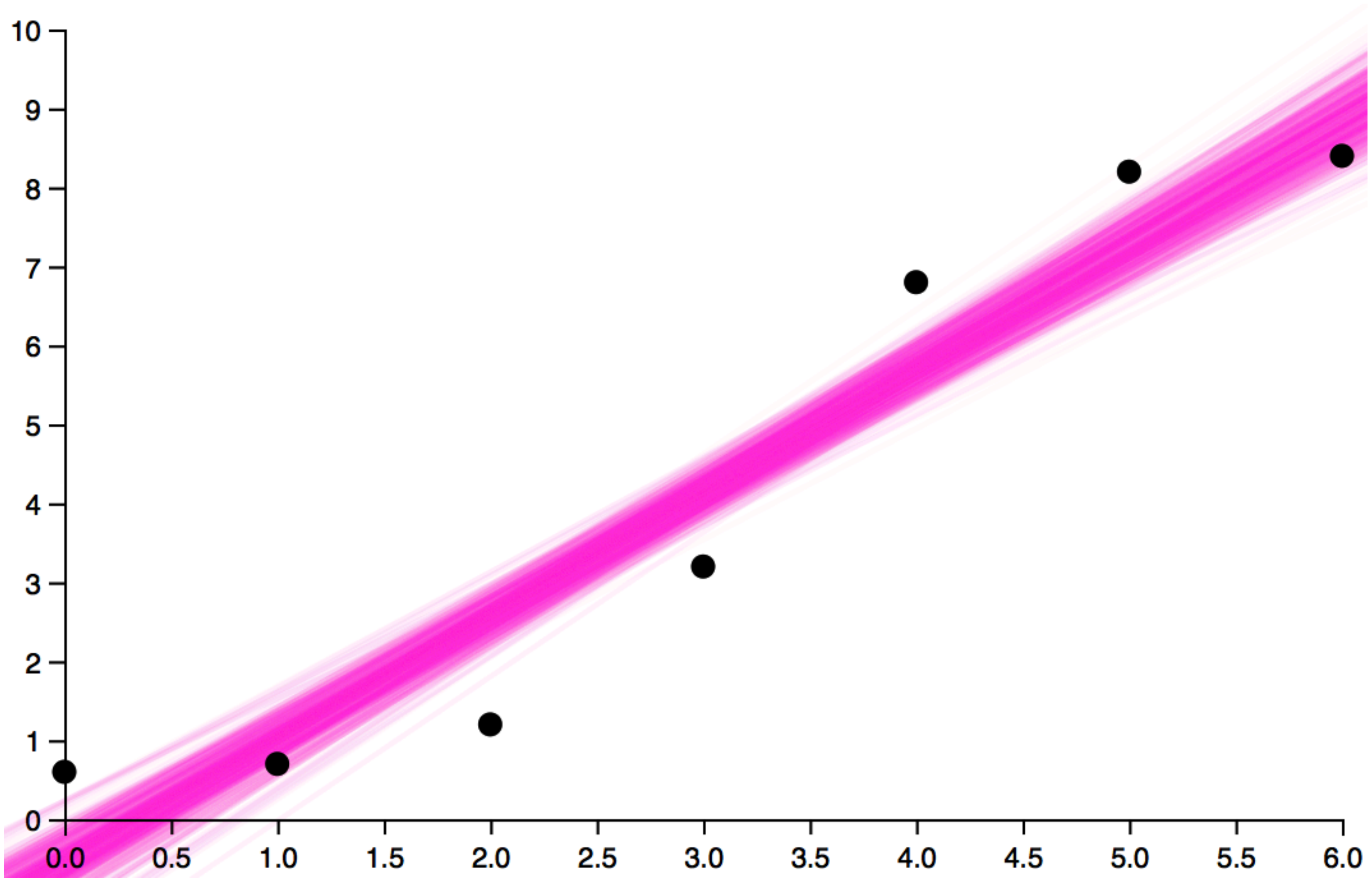


Why should one care about prob. programming?

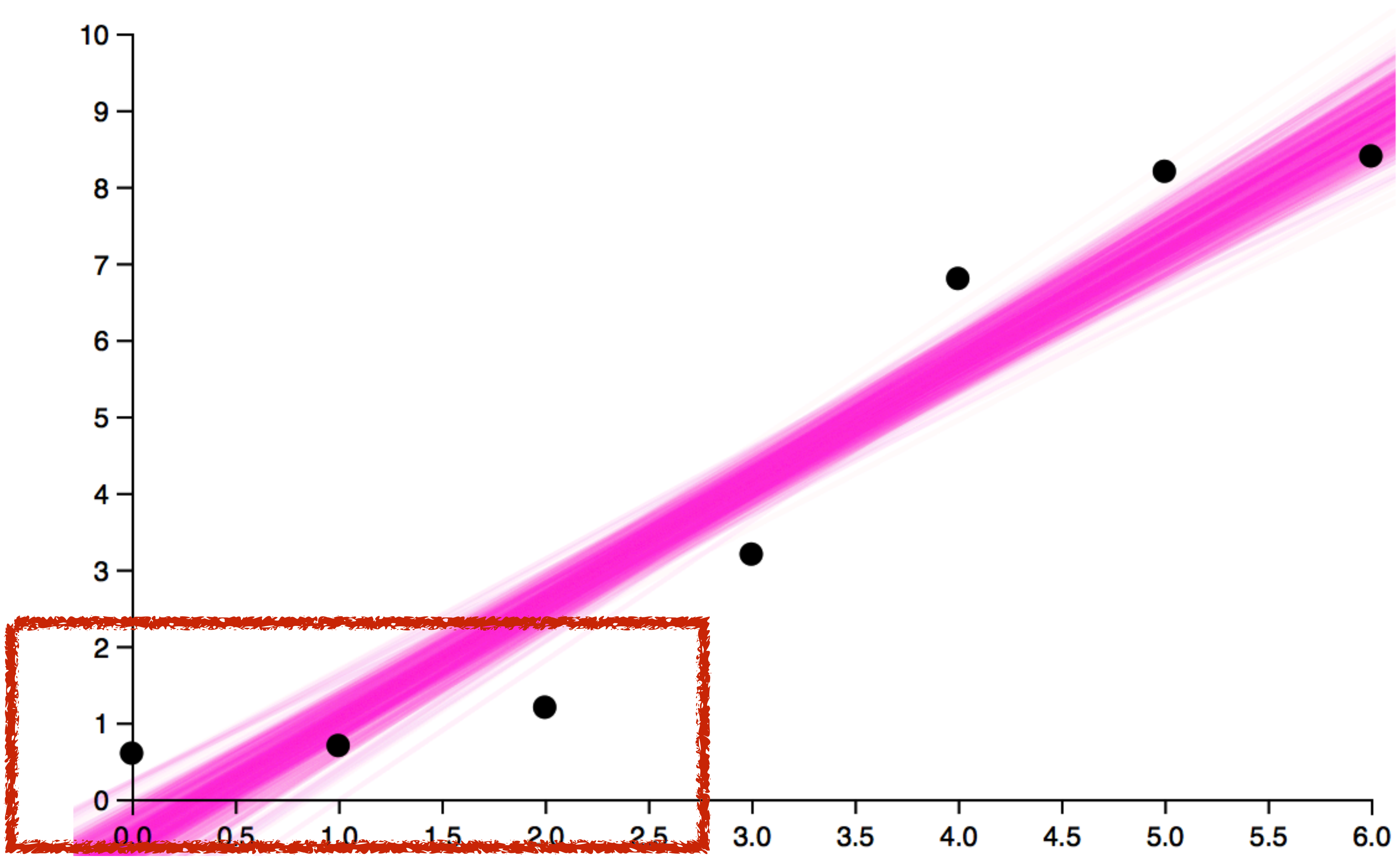
"Because probabilistic programming is a good way to build an AI." (My ML colleague)

Prob. programming languages enable one to build and explore highly complex models.

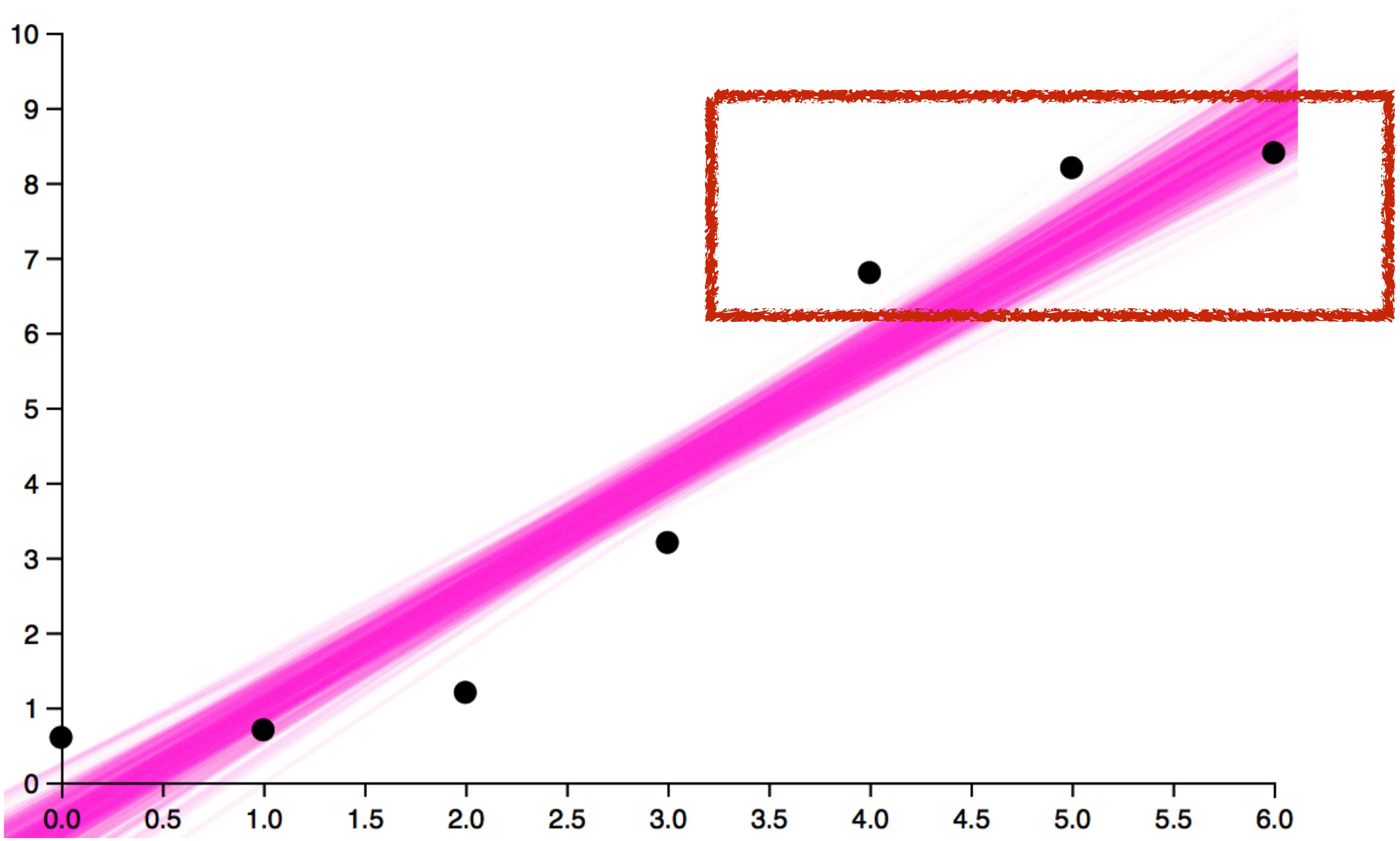
Underfit?



Underfit?



Underfit?



```
(let [s (sample (normal 0 2))
      b (sample (normal 0 6))
      f (fn [x] (+ (* s x) b))]

    (observe (normal (f 0) 0.5) 0.6)
    (observe (normal (f 1) 0.5) 0.7)
    (observe (normal (f 2) 0.5) 1.2)
    (observe (normal (f 3) 0.5) 3.2)
    (observe (normal (f 4) 0.5) 6.8)
    (observe (normal (f 5) 0.5) 8.2)
    (observe (normal (f 6) 0.5) 8.4)

  [s b])
```

```
(let [s (sample (normal 0 2))
      b (sample (normal 0 6))
      f (fn [x] (+ (* s x) b))]

    (observe (normal (f 0) 0.5) 0.6)
    (observe (normal (f 1) 0.5) 0.7)
    (observe (normal (f 2) 0.5) 1.2)
    (observe (normal (f 3) 0.5) 3.2)
    (observe (normal (f 4) 0.5) 6.8)
    (observe (normal (f 5) 0.5) 8.2)
    (observe (normal (f 6) 0.5) 8.4)

  [s b])
```

Functions as first-class citizen.

```
(let [s (sample (normal 0 2))
      b (sample (normal 0 6))
      f (fn [x] (+ (* s x) b))]

    (observe (normal (f 0) 0.5) 0.6)
    (observe (normal (f 1) 0.5) 0.7)
    (observe (normal (f 2) 0.5) 1.2)
    (observe (normal (f 3) 0.5) 3.2)
    (observe (normal (f 4) 0.5) 6.8)
    (observe (normal (f 5) 0.5) 8.2)
    (observe (normal (f 6) 0.5) 8.4)

  [s b])
  f)
```

Functions as first-class citizen.

```
(let [F (fn []
          (let [s (sample (normal 0 2))
                b (sample (normal 0 6))]
            (fn [x] (+ (* s x) b))))
      f (F)]
    (observe (normal (f 0) 0.5) 0.6)
    (observe (normal (f 1) 0.5) 0.7)
    (observe (normal (f 2) 0.5) 1.2)
    (observe (normal (f 3) 0.5) 3.2)
    (observe (normal (f 4) 0.5) 6.8)
    (observe (normal (f 5) 0.5) 8.2)
    (observe (normal (f 6) 0.5) 8.4)

  ~~[s b])~~
   f)
```

Functions as first-class citizen.

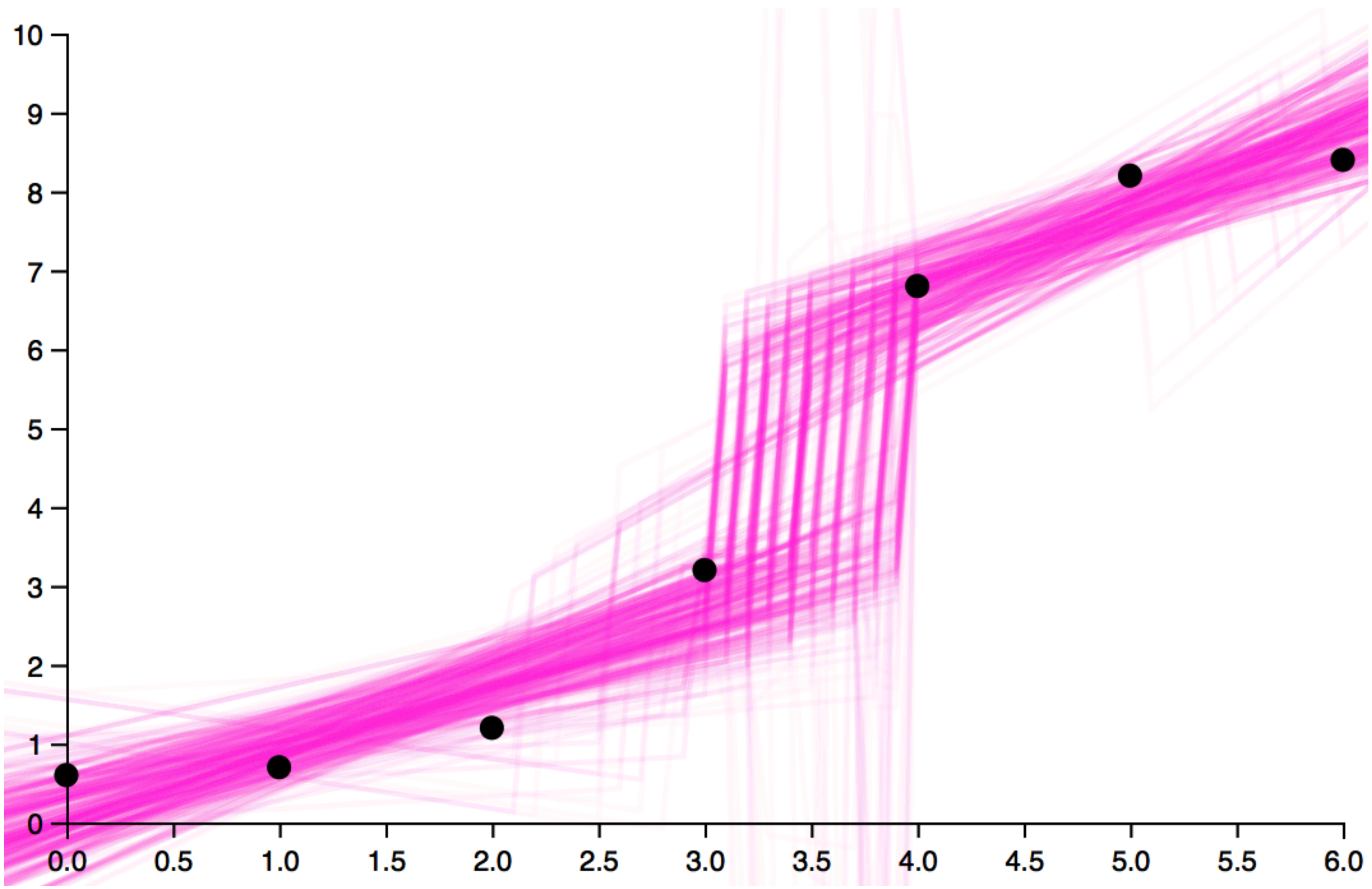
```
(let [F (fn []
          (let [s (sample (normal 0 2))
                b (sample (normal 0 6))]
            (fn [x] (+ (* s x) b))))
      f (add-change-points F 0 6)]
    (observe (normal (f 0) 0.5) 0.6)
    (observe (normal (f 1) 0.5) 0.7)
    (observe (normal (f 2) 0.5) 1.2)
    (observe (normal (f 3) 0.5) 3.2)
    (observe (normal (f 4) 0.5) 6.8)
    (observe (normal (f 5) 0.5) 8.2)
    (observe (normal (f 6) 0.5) 8.4)

  [s b])
  f)
```

Functions as first-class citizen.

```
(defm add-change-points [F l u]
  (if (sample (flip 0.5)) (F)
    (if (sample (flip 0.5))
       (let [cp1 (sample
                   (uniform-continuous l u))
             f1 (F)
             f2 (F)]
         (fn [x] (if (< x cp1) (f1 x) (f2 x))))
       (let [cp1 (sample
                   (uniform-continuous l u))
             cp2 (sample
                   (uniform-continuous cp1 u))
             f1 (F)
             f2 (F)
             f3 (F)]
         (fn [x] (if (< x cp1) (f1 x)
                    (if (< x cp2) (f2 x)
                       (f3 x)))))))))
```

```
(defm add-change-points [F l u]
  (if (sample (flip 0.5)) (F)
    (if (sample (flip 0.5))
      (let [cp1 (sample
                  (uniform-continuous l u))
            f1 (F)
            f2 (F)]
        (fn [x] (if (< x cp1) (f1 x) (f2 x))))
      (let [cp1 (sample
                  (uniform-continuous l u))
            cp2 (sample
                  (uniform-continuous cp1 u))
            f1 (F)
            f2 (F)
            f3 (F)]
        (fn [x] (if (< x cp1) (f1 x)
                   (if (< x cp2) (f2 x)
                      (f3 x))))))))
```

Samples from posterior



```
(let [F (fn []
          (let [s (sample (normal 0 2))
                b (sample (normal 0 6))]
            (fn [x] (+ (* s x) b))))
      f (add-change-points F 0 6)]
    (observe (normal (f 0) .5) .6)
    (observe (normal (f 1) .5) .7)
    (observe (normal (f 2) .5) 1.2)
    (observe (normal (f 3) .5) 3.2)
    (observe (normal (f 4) .5) 6.8)
    (observe (normal (f 5) .5) 8.2)
    (observe (normal (f 6) .5) 8.4)
```

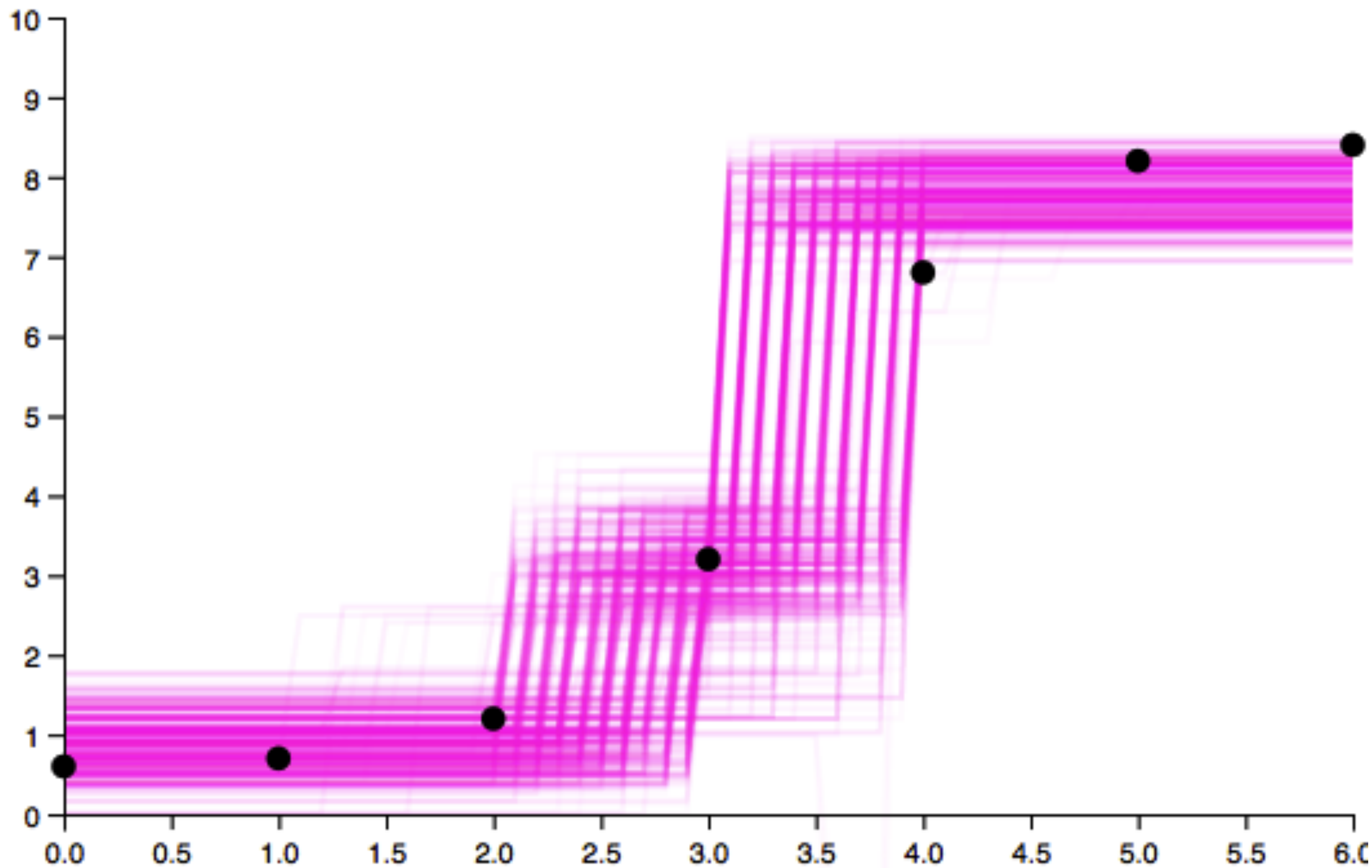
~~[s b])~~
`f)`

[Q] Change the model so that it finds constant functions with change points.

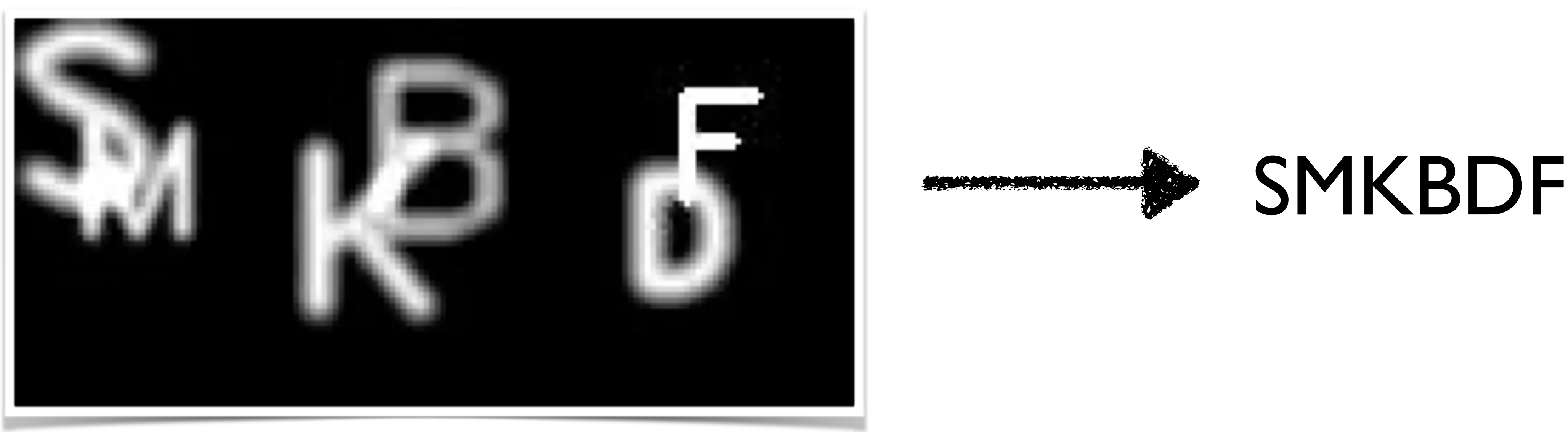
```
(let [F (fn []
          (let [s (sample (normal 0 2))
                b (sample (normal 0 6))]
            (fn [x] (+ (* s x) b))))
      f (add-change-points F 0 6)]
    (observe (normal (f 0) .5) .6)
    (observe (normal (f 1) .5) .7)
    (observe (normal (f 2) .5) 1.2)
    (observe (normal (f 3) .5) 3.2)
    (observe (normal (f 4) .5) 6.8)
    (observe (normal (f 5) .5) 8.2)
    (observe (normal (f 6) .5) 8.4)

  [s b])
  f)
```

[Q] Change the model so that it finds constant functions with change points.

Samples from posterior



Three success stories: 1) captcha breaking



Le, Baydin, Wood [2016]

Three success stories: 1) captcha breaking

1. Sample a string.

Le, Baydin, Wood [2016]

Three success stories:
1) captcha breaking

SMKBDF

1. Sample a string.
2. Generate an image using complex JVM code.

Le, Baydin, Wood [2016]

Three success stories: 1) captcha breaking

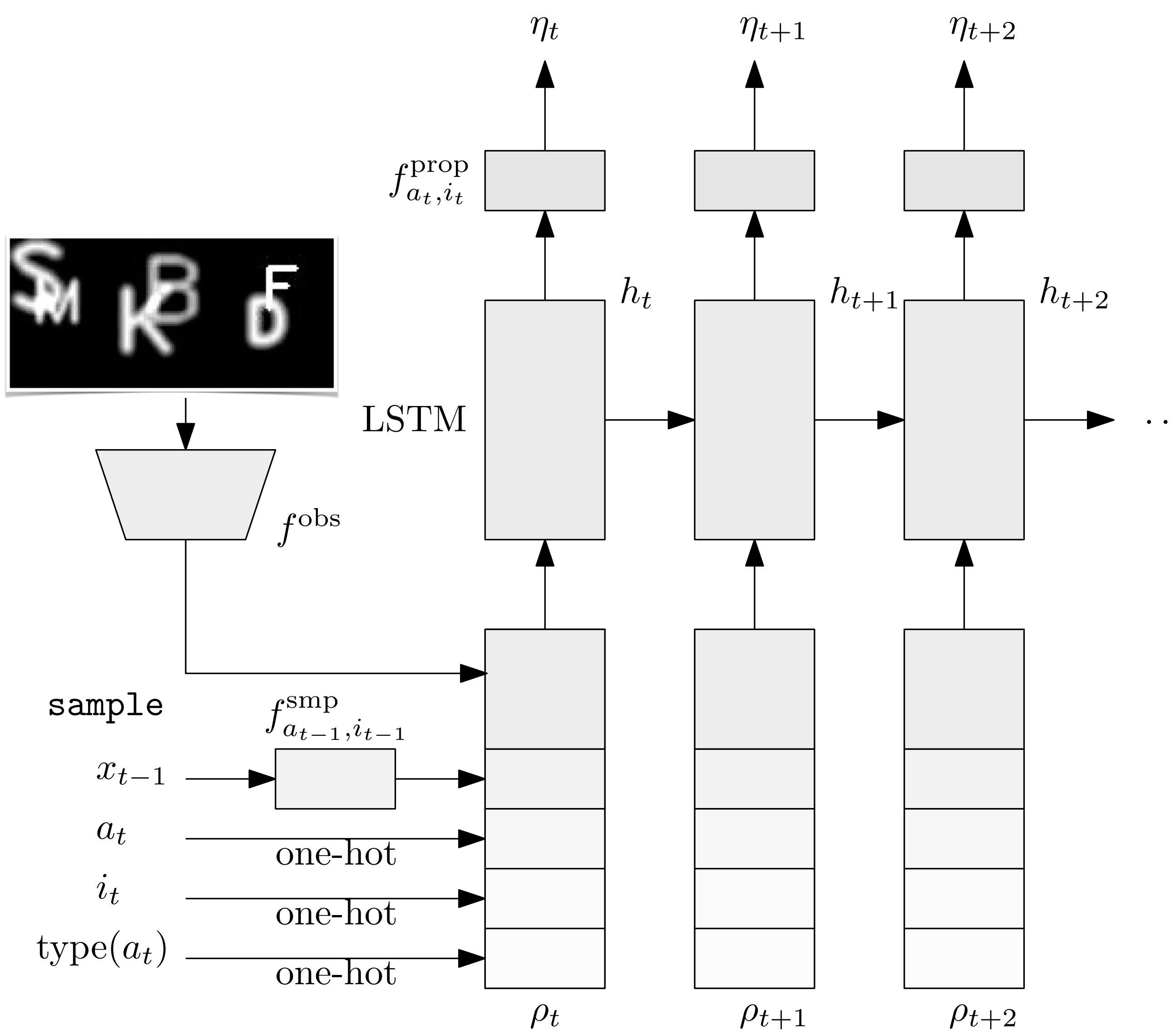
Anglican's inference engine based on neural nets

→ SMKBDF

1. Sample a string.
2. Generate an image using complex JVM code.

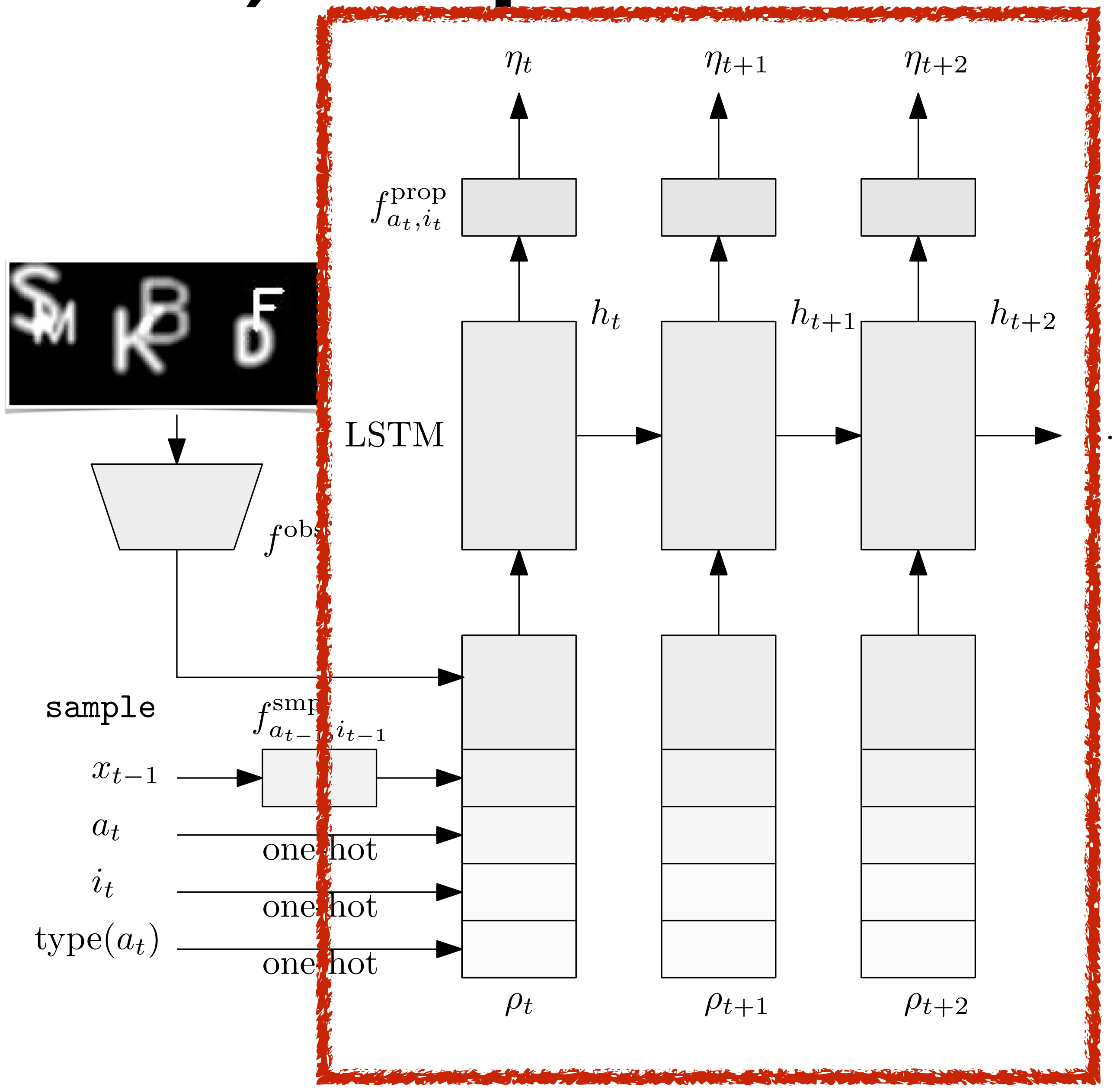
Le, Baydin, Wood [2016]

Three success stories: 1) captcha breaking



Le, Baydin, Wood [2016]

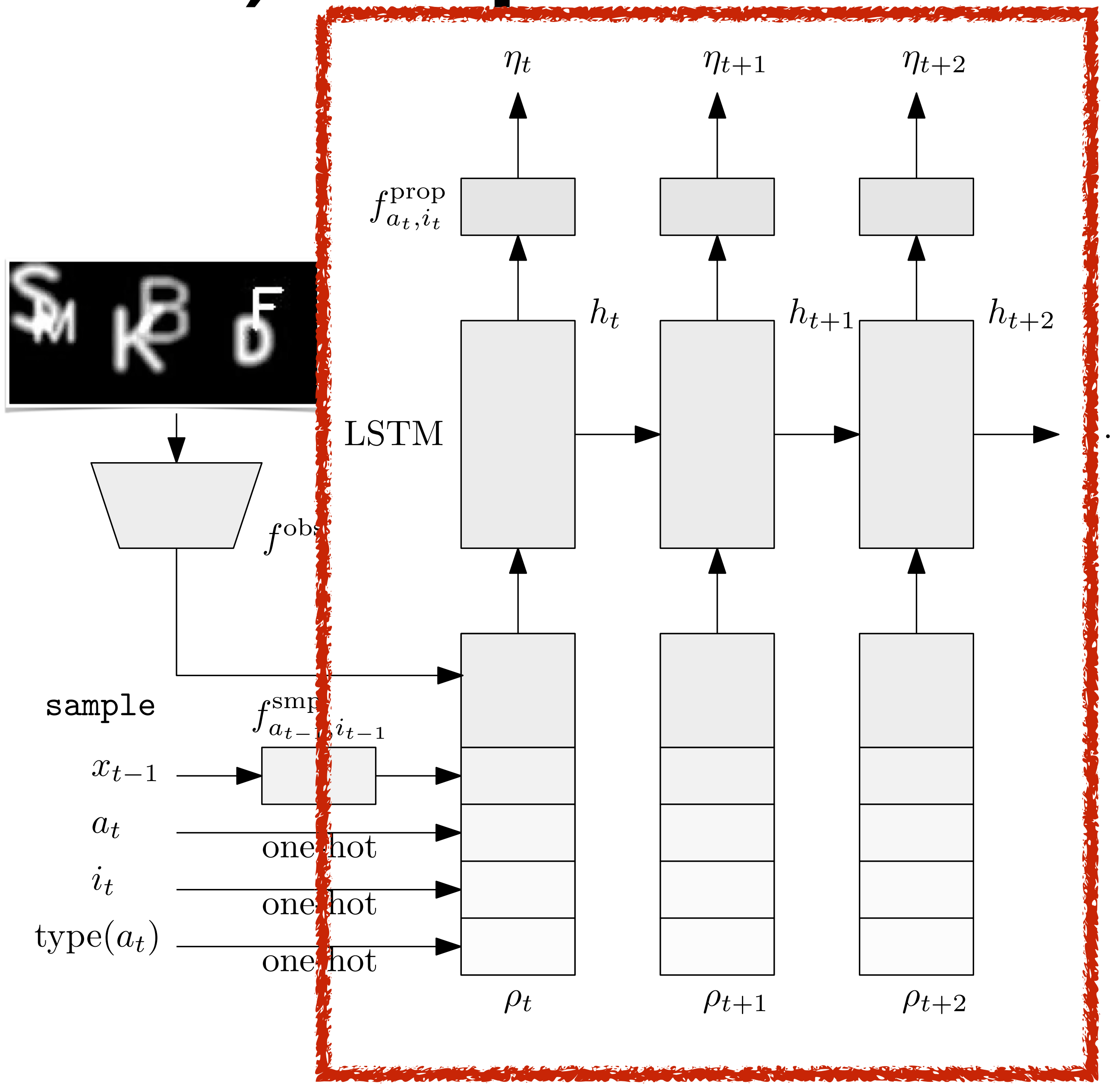
Three success stories: 1) captcha breaking



Neural net as a part of inference engine.

Le, Baydin, Wood [2016]

Three success stories:
1) captcha breaking

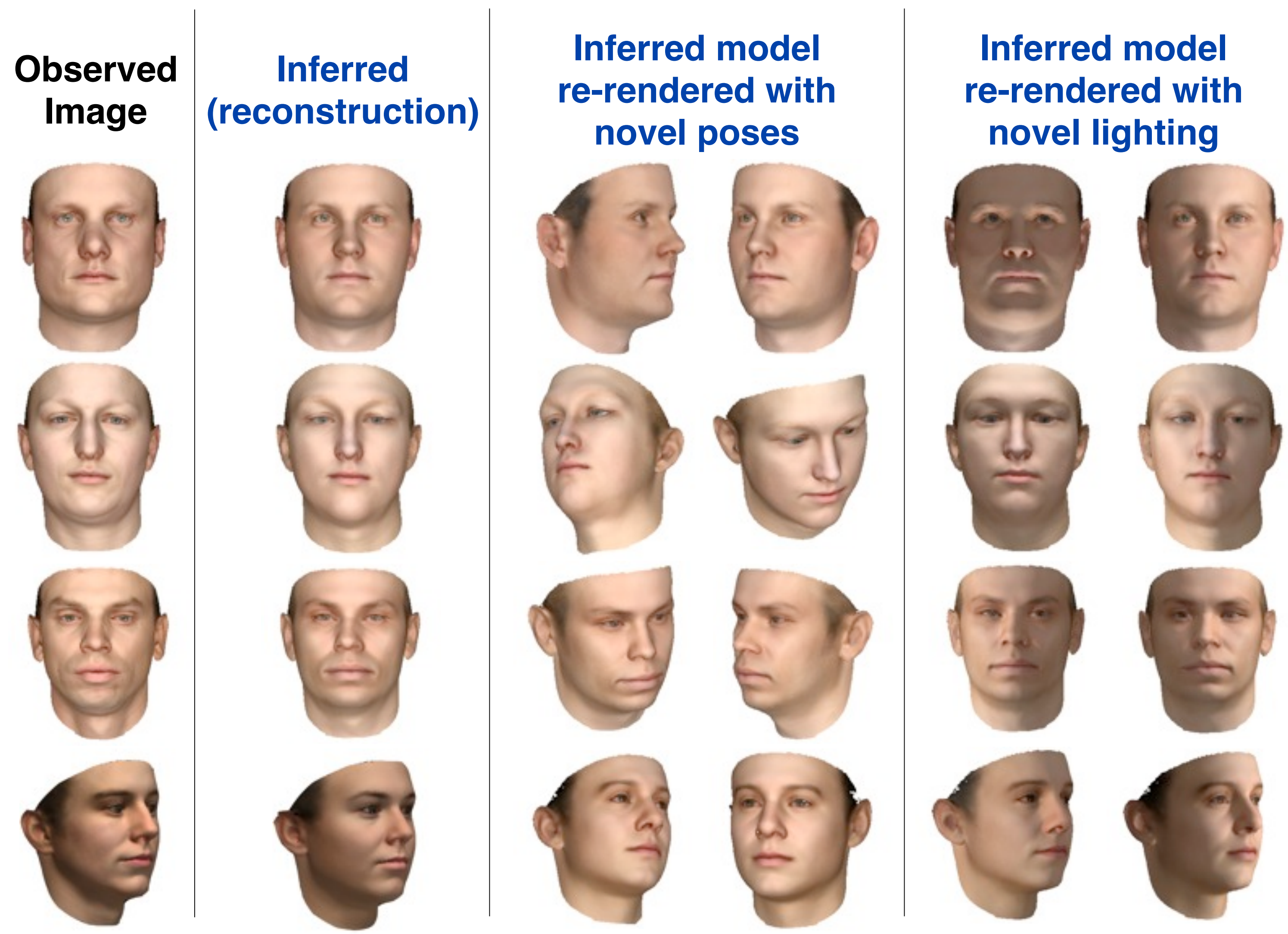


Neural net as a part of inference engine.

Approximates the inverse of the Captcha program.

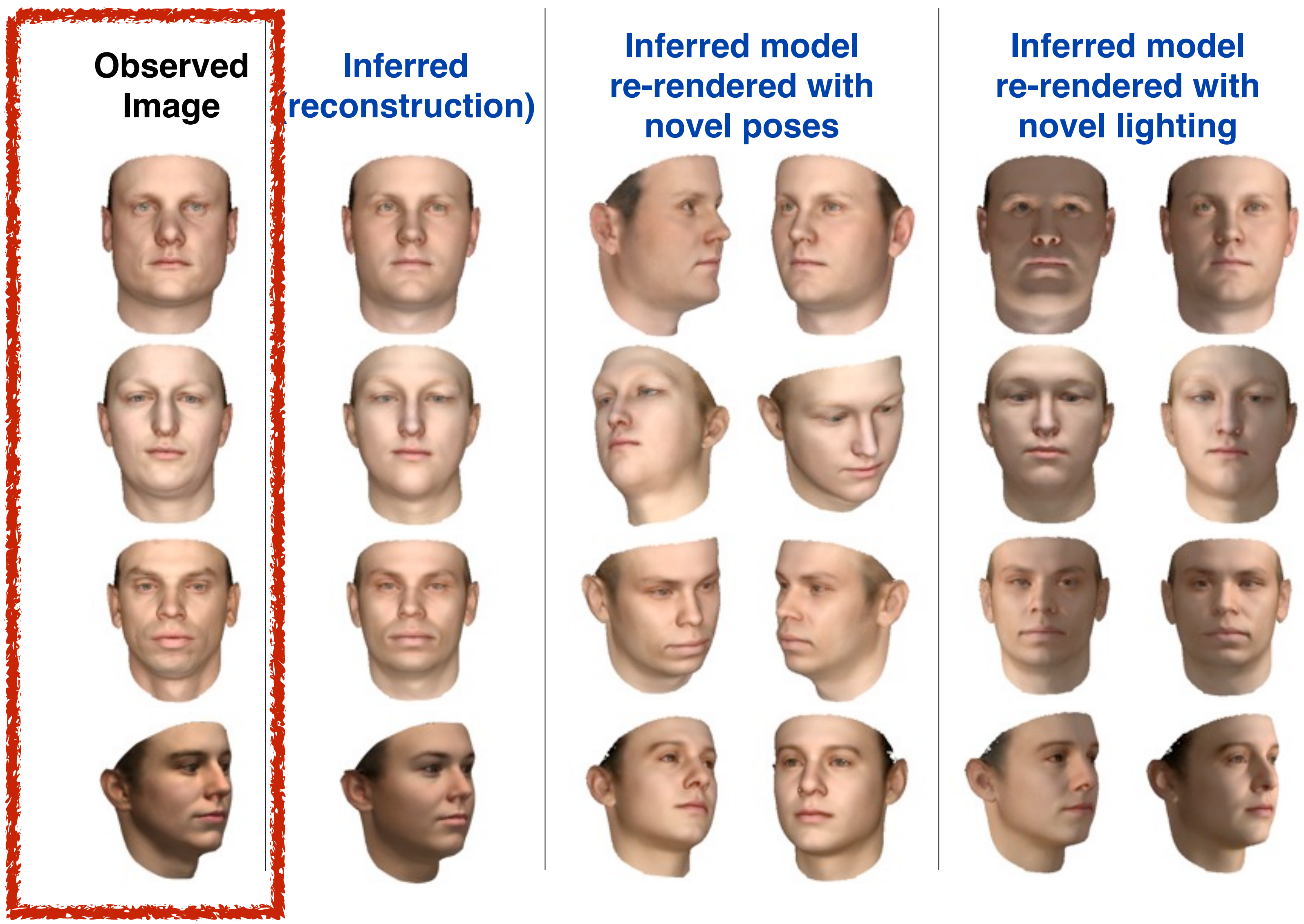
Le, Baydin, Wood [2016]

Three success stories:
2) inverse graphics



Kulkarni, Kohl, Tenenvaum, Mansinghka [CVPR'15]

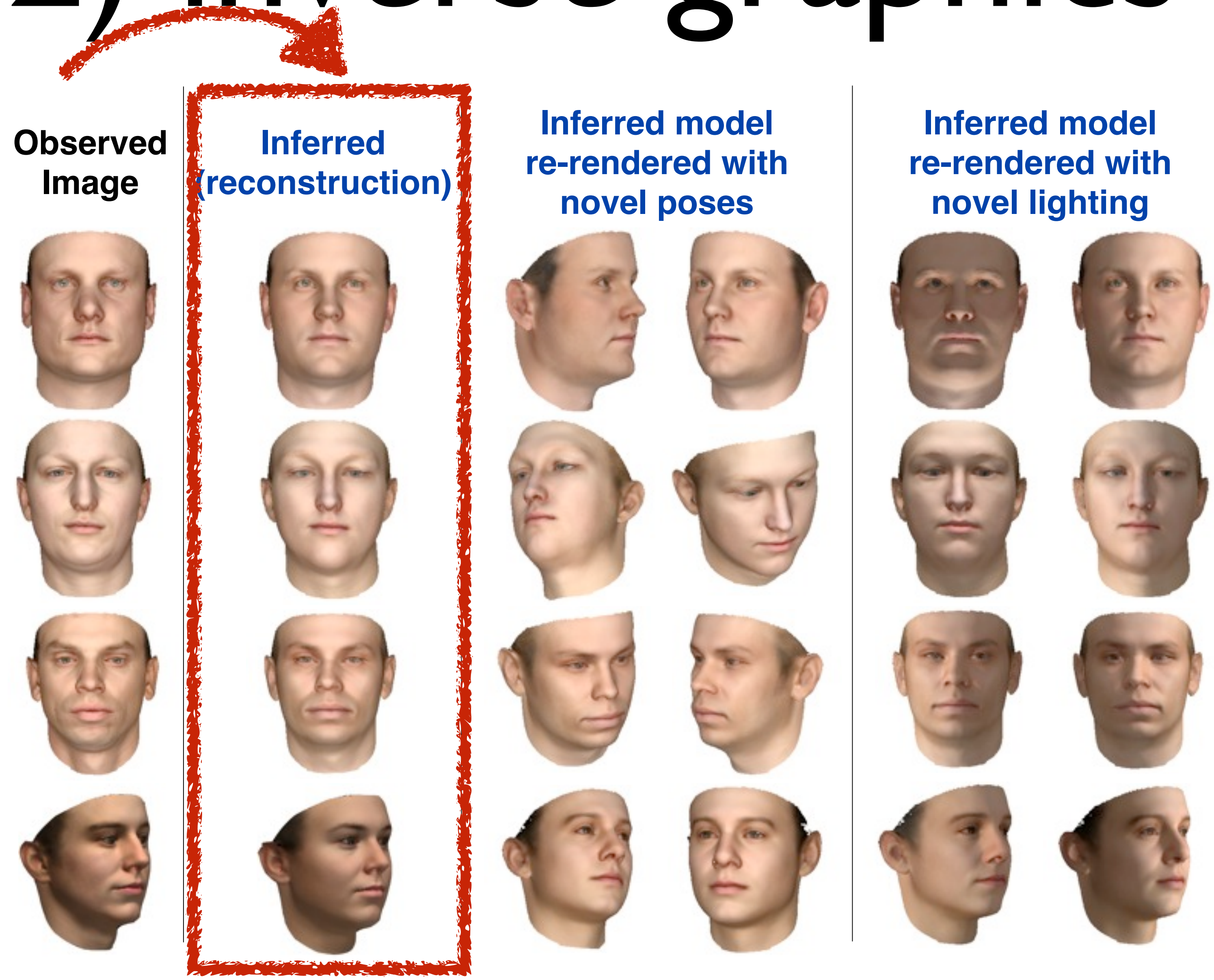
Three success stories:
2) inverse graphics



Image

Kulkarni, Kohl, Tenenvaum, Mansinghka [CVPR'15]

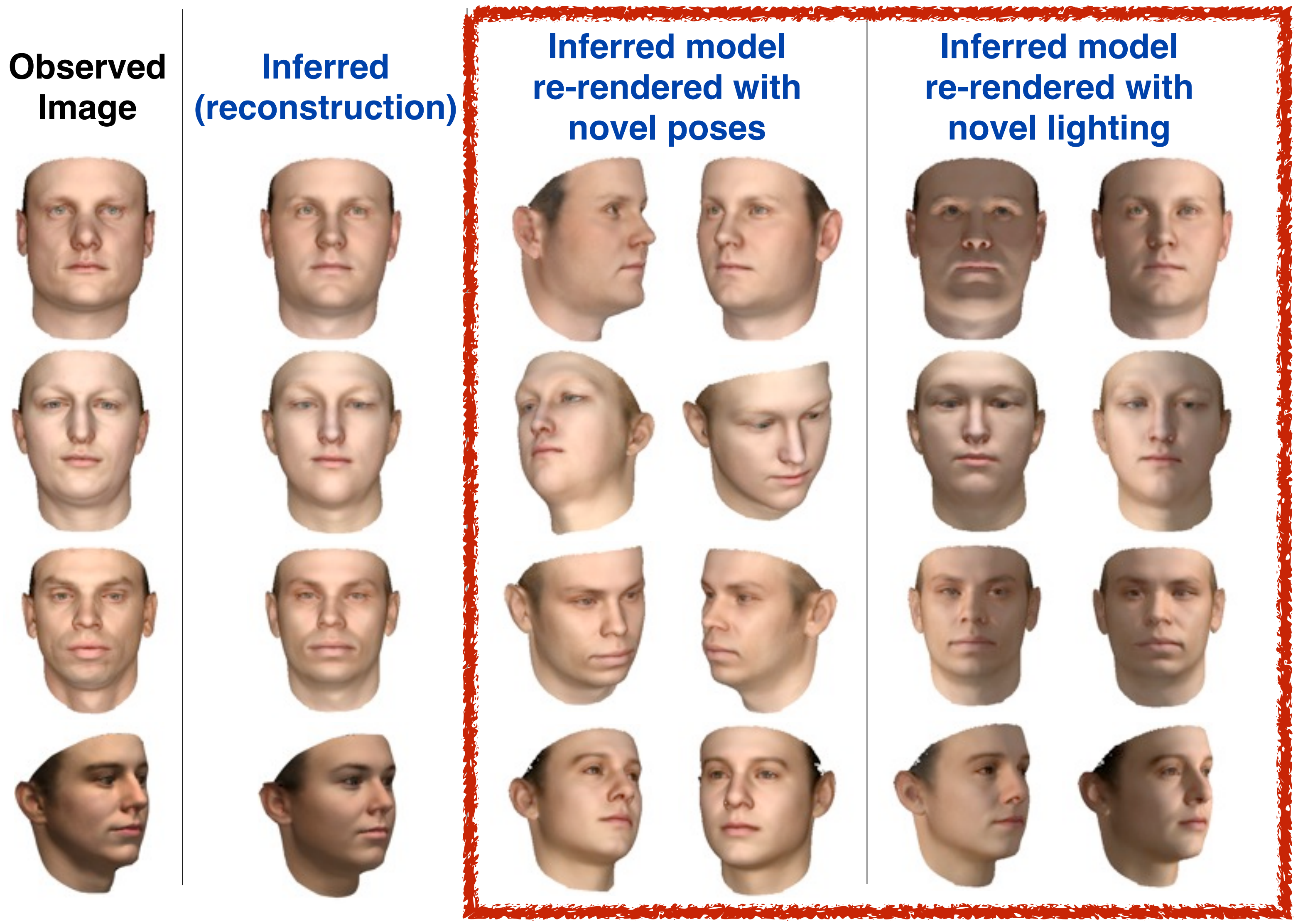
Three success stories:
2) inverse graphics



3D model

Kulkarni, Kohl, Tenenvaum, Mansinghka [CVPR'15]

Three success stories:
2) inverse graphics

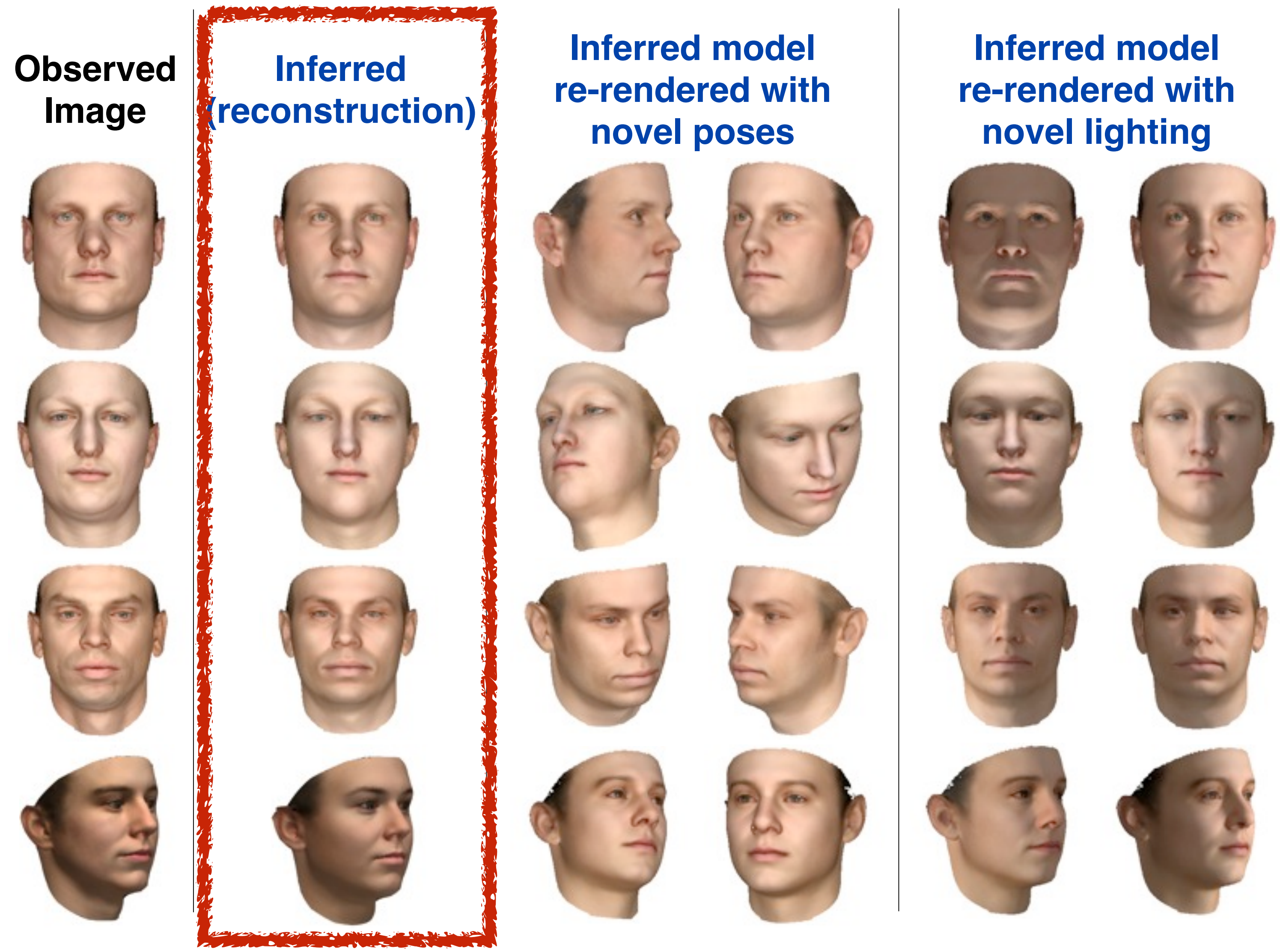


Transformed 3D models

Kulkarni, Kohl, Tenenvaum, Mansinghka [CVPR'15]

Th... ccess stories: 2) inverse graphics

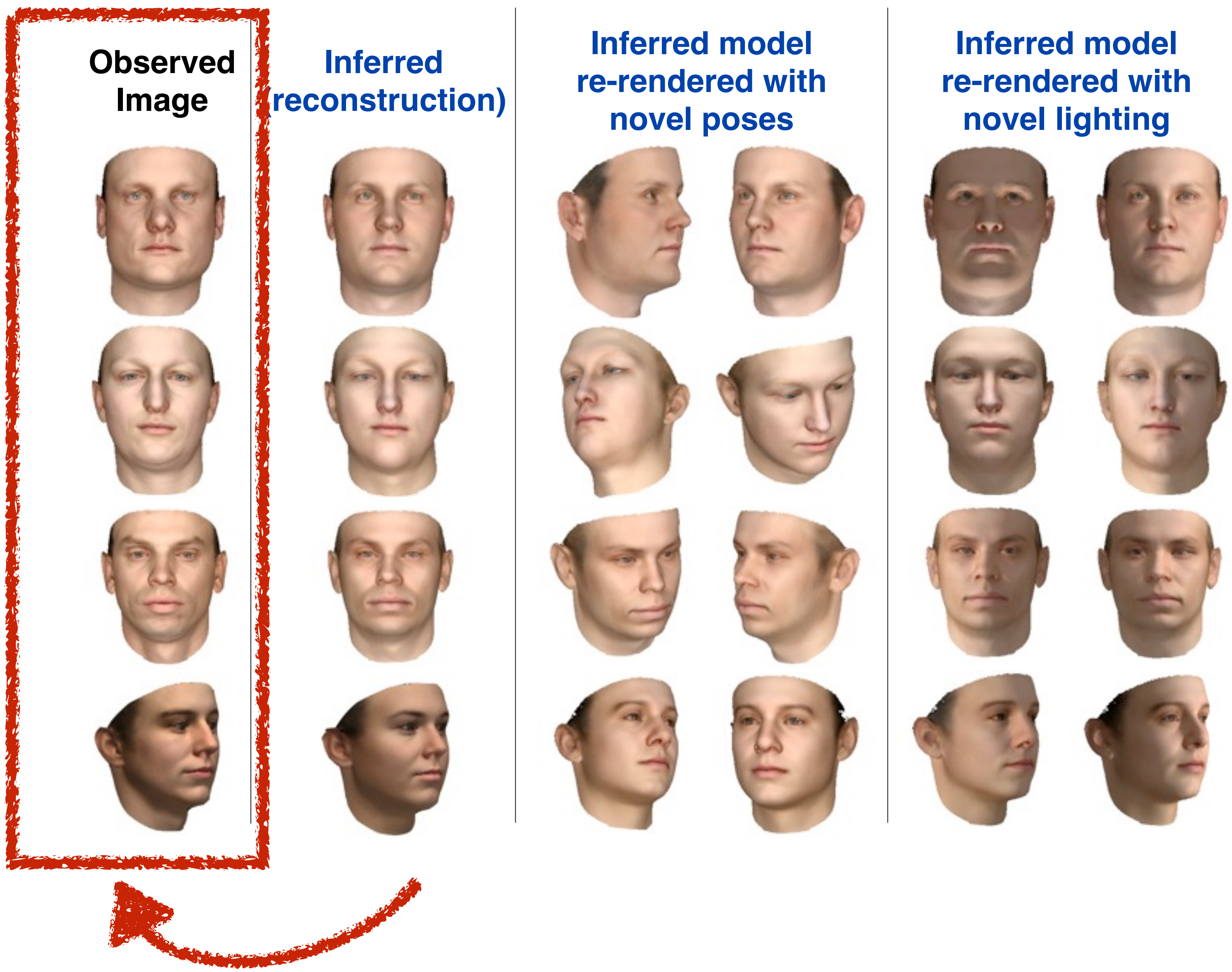
1. Sample a 3D model.



Kulkarni, Kohl, Tenenvaum, Mansinghka [CVPR'15]

ccess stories:
rse graphics

1. Sample a 3D model.
2. Generate an image.

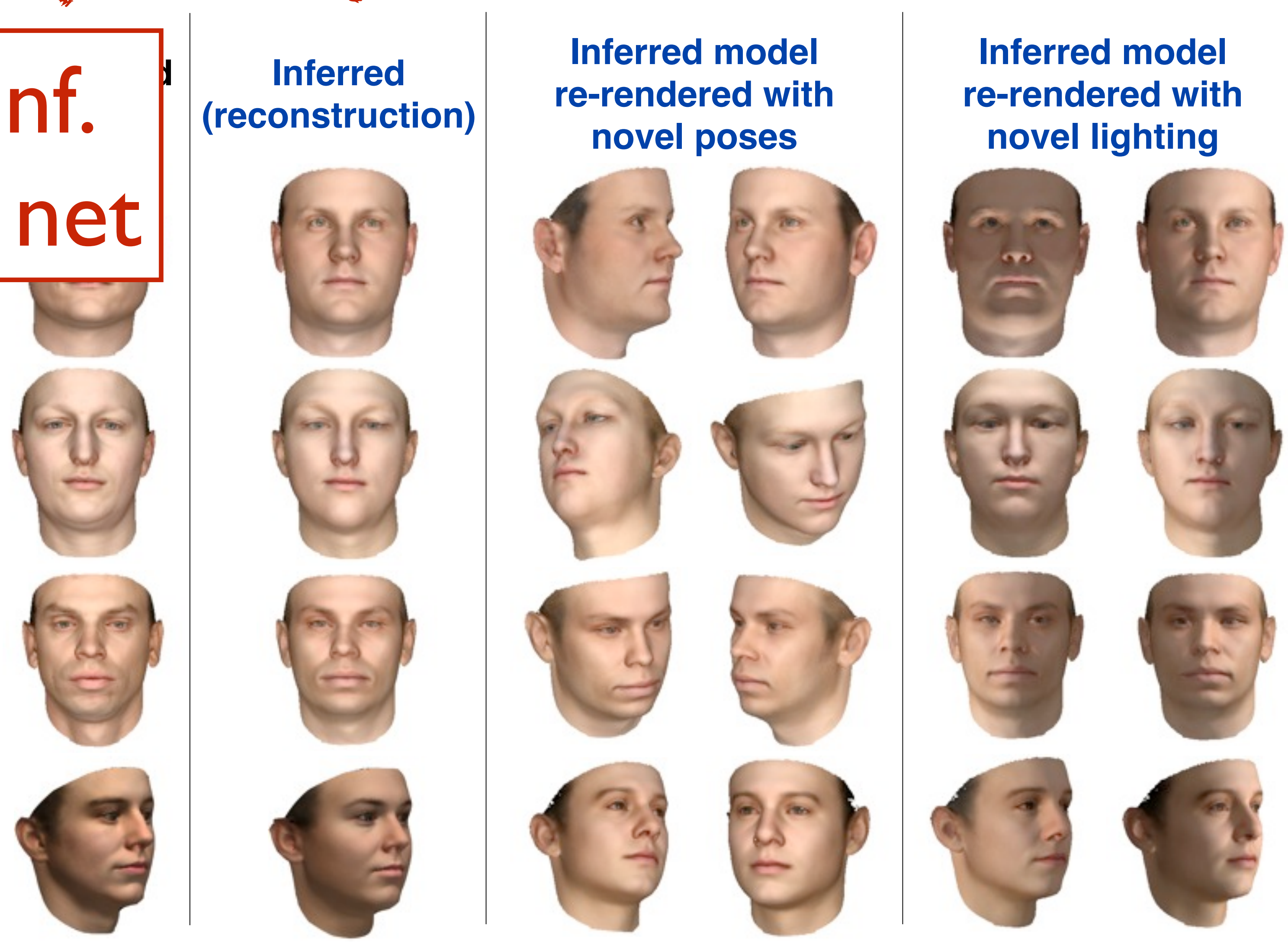


Kulkarni, Kohl, Tenenvaum, Mansinghka [CVPR'15]

ccess stories:
rse graphics

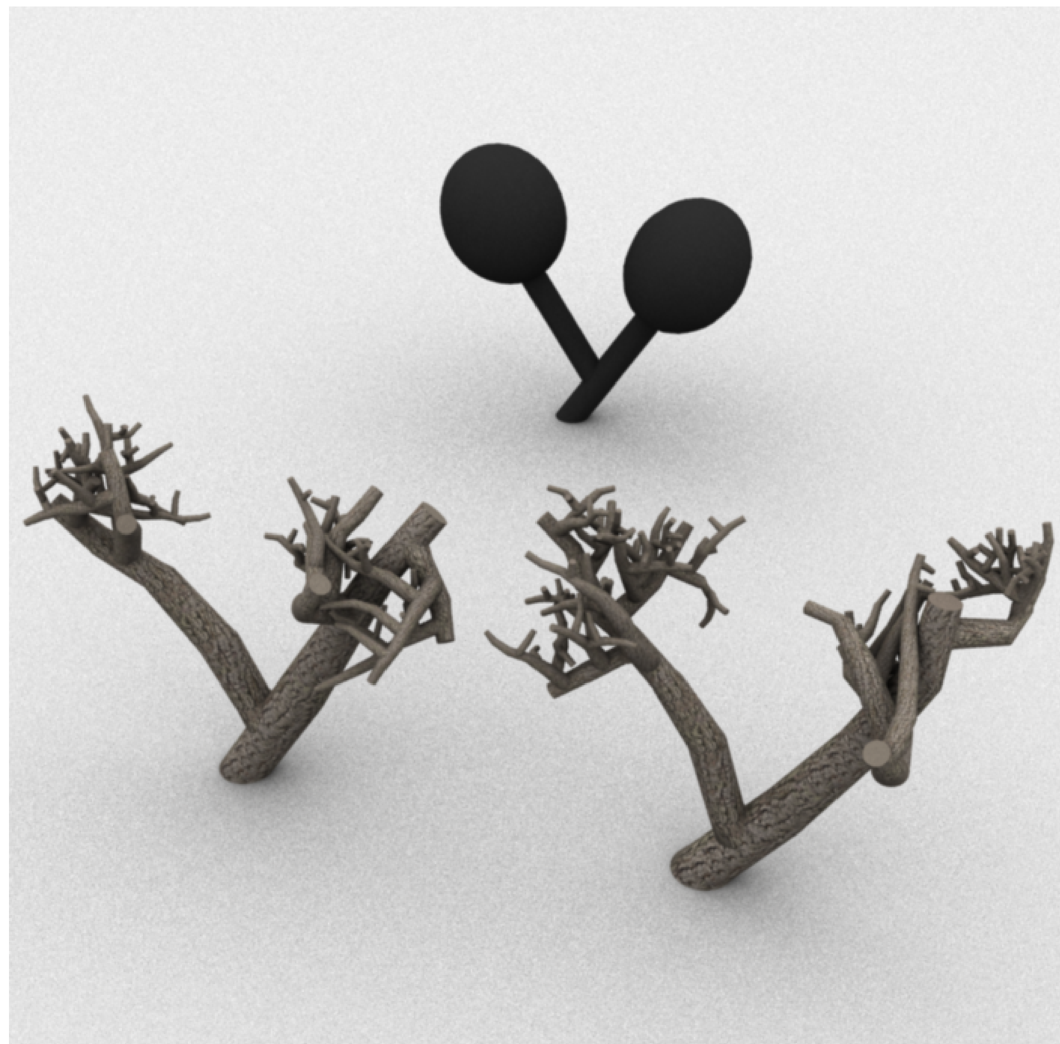
1. Sample a 3D model.
2. Generate an image.

Posterior inf. with neural net



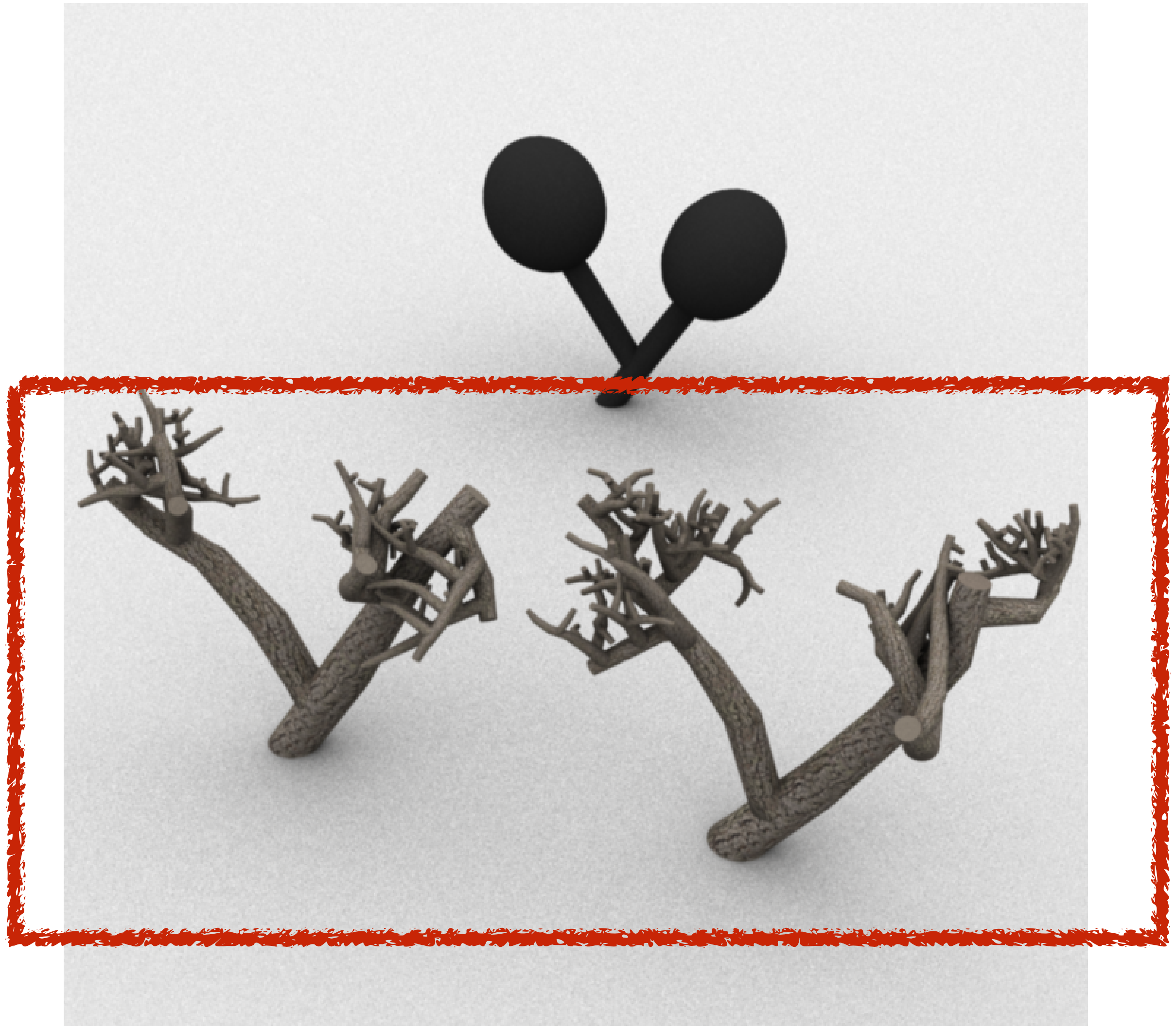
Kulkarni, Kohl, Tenenvaum, Mansinghka [CVPR'15]

Three success stories:
3) procedural modelling



Ritchie, Mildenhall, Goodman, Hanrahan [SIGGRAPH'15]

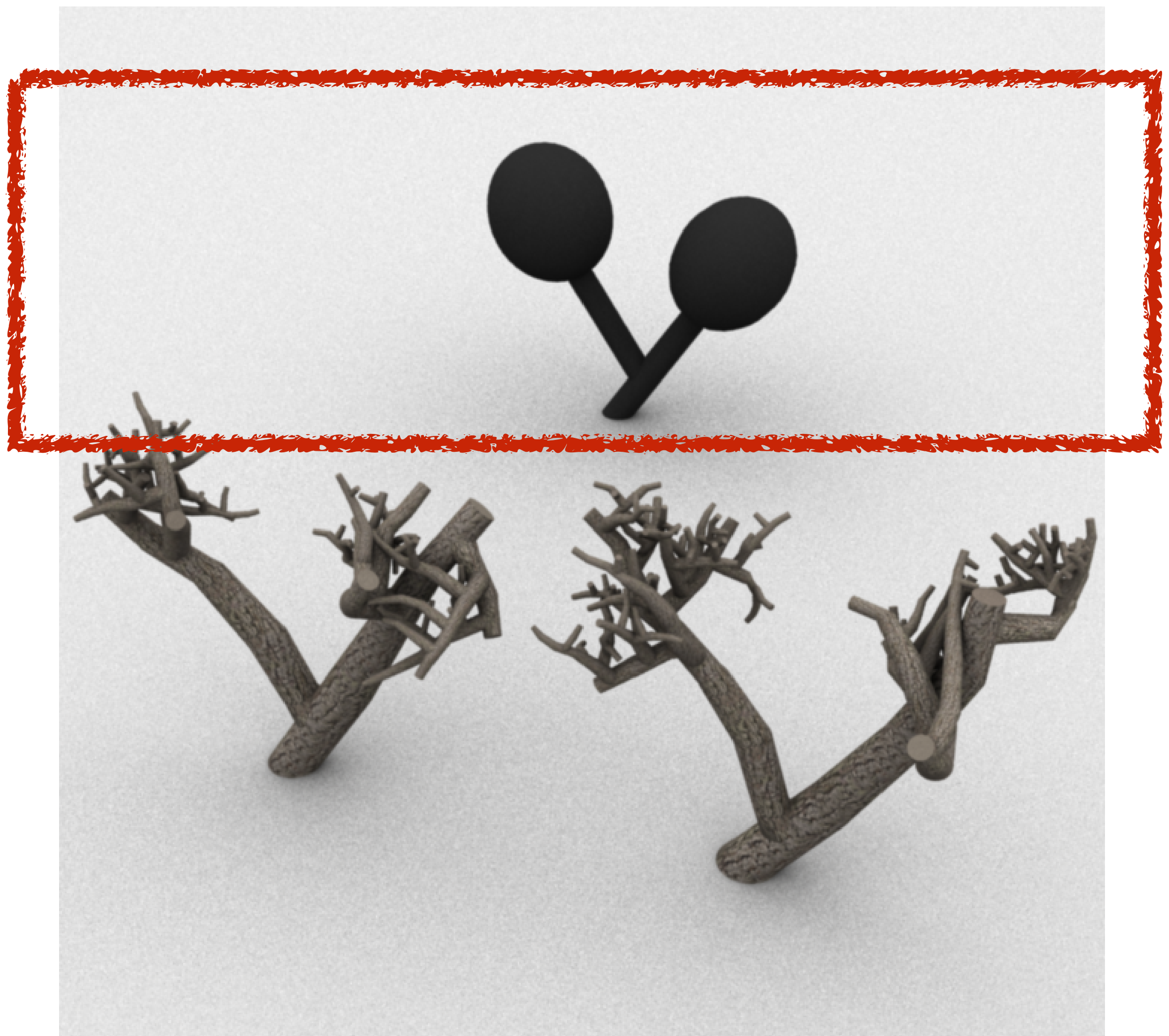
Three success stories:
3) procedural modelling



1. Sample a 3D object.

Ritchie, Mildenhall, Goodman, Hanrahan [SIGGRAPH'15]

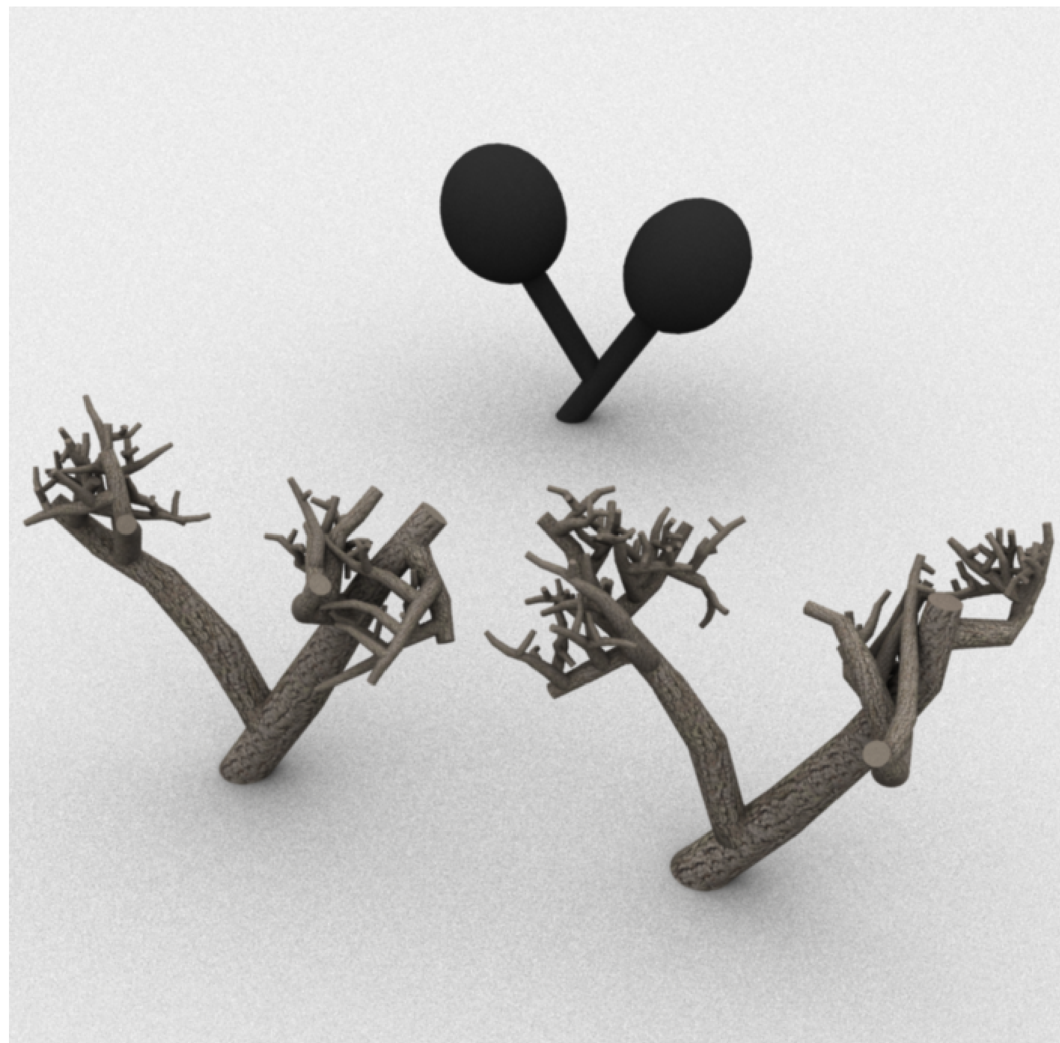
Three success stories: 3) procedural modelling



1. Sample a 3D object.
2. Score the object.

Ritchie, Mildenhall, Goodman, Hanrahan [SIGGRAPH'15]

Three success stories:
3) procedural modelling



1. Sample a 3D object.
2. Score the object.

Used stochastic future.

Ritchie, Mildenhall, Goodman, Hanrahan [SIGGRAPH'15]

Three suc

Asynchronous function call via future

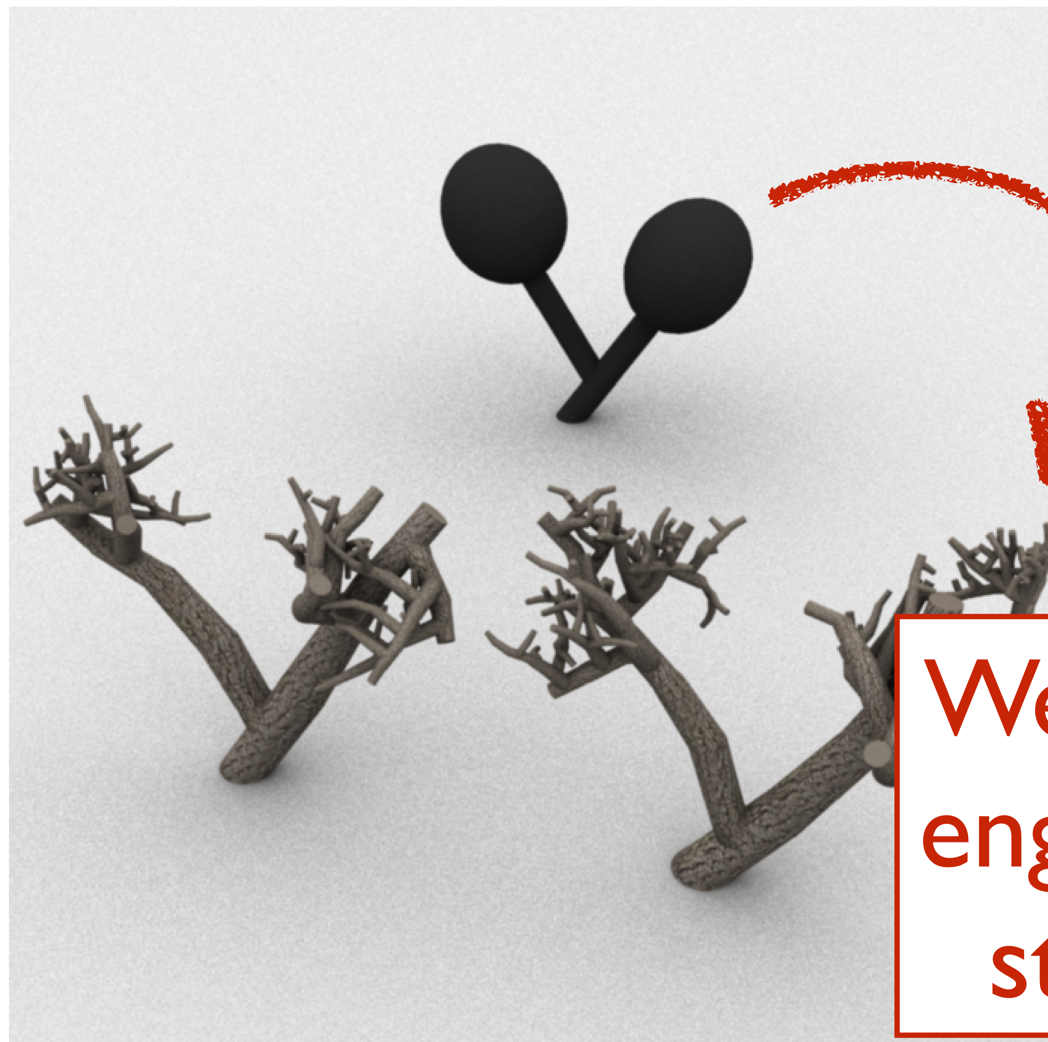
```
future.create(function(i, frame, prev)
        if flip(T.branchProb(depth, i)) then
                -- Theta mean/variance based on avg weighted by
                local theta_mu, theta_sigma = T.estimateThetaD
                local theta = gaussian(theta_mu, theta_sigma)
                local maxbranchradius = 0.5*(nextframe.center
                local branchradius = math.min(uniform(0.9, 1)
                local bframe, prev = T.branchFrame(splitFrame,
                branch(bframe, depth+1, prev)
        end
```

1. Sample a 3D object.
2. Score the object.

Used stochastic future.

Ritchie, Mildenhall, Goodman, Hanrahan [SIGGRAPH'15]

Three success stories:
3) procedural modelling



WebPPL's inference engine that exploits stochastic future

1. Sample a 3D object.
2. Score the object.

Used stochastic future.

Ritchie, Mildenhall, Goodman, Hanrahan [SIGGRAPH'15]

Techniques used

- Changepoints: CPS transformation and new foundation of probability theory.
- Captcha and inverse graphics: neural nets and inference amortisation.
- Procedural modelling: sequential Monte Carlo algorithms and stochastic future.

Techniques used

- Changepoints: CPS transformation and new foundation of probability theory.
- Captcha and inverse graphics: neural nets and inference amortisation.
- Procedural modelling: sequential Monte Carlo algorithms and stochastic future.

Prog. Languages,

Techniques used

- Changepoints: CPS transformation and new foundation of probability theory.
- Captcha and inverse graphics: neural nets and inference amortisation.
- Procedural modelling: sequential Monte Carlo algorithms and stochastic future.

Prog. Languages, Machine Learning,

Techniques used

- Changepoints: CPS transformation and new foundation of probability theory.
- Captcha and inverse graphics: neural nets and inference amortisation.
- Procedural modelling: sequential Monte Carlo algorithms and stochastic future.

Prog. Languages, Machine Learning, Probability Theory

Overview of the course

Objective

1. Learn how to write and reason about models in a prob. prog. language (PPL).
2. Learn results from ML/PL/prob. theory that are used for building effective PPLs.
3. Contribute to probabilistic programming.

Objective

1. Learn how to write and reason about models in a prob. prog. language (PPL).
2. Learn results from ML/PL/prob. theory that are used for building effective PPLs.
3. Contribute to probabilistic programming.

Group project by a group of 1-3 students.

Webpage

https://github.com/hongseok-yang/probprog20

All the important announcements will be made in this webpage.

Evaluation

- 3-5 homework exercises (20%).
- Group project (40%).
- Final exam (40%).

Evaluation

- 3-5 homework exercises (20%).
- Group project (40%).
- Final exam (40%).

1-3 students form a group.

New cool application of a PPL.

Tasks:

- Project.
- 2 (likely online) presentations (topic - 27 Apr, result - 17/22/24 June).
- Report. 2 pages not including bibliography.

Really important announcement

Form a group and inform me and TAs by the midnight of 30 March (Monday).

Somewhat important announcement

- Lecturer: Prof Hongseok Yang

  email: hongseok00@gmail.com

- TAs: Gwonsoo Che, Daeseok Lee

  email: Gwonsoo.che@gmail.com

  email: dsleemaths@gmail.com

- Install Anglican. Try the changepoint example. Get hints from the webpage.

Webpage

https://github.com/hongseok-yang/probprog20

All the important announcements will be made in this webpage.