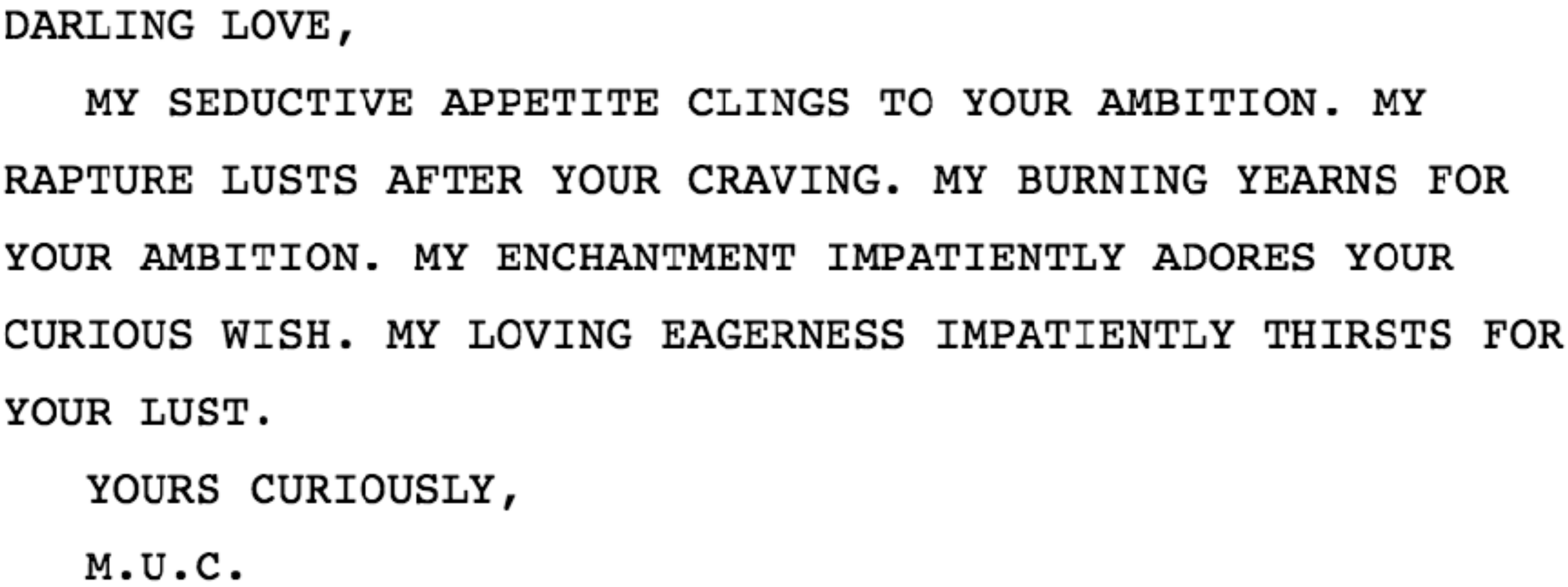
# Probabilistic Programming

Hongseok Yang
University of Oxford

# Manchester Univ. 1953



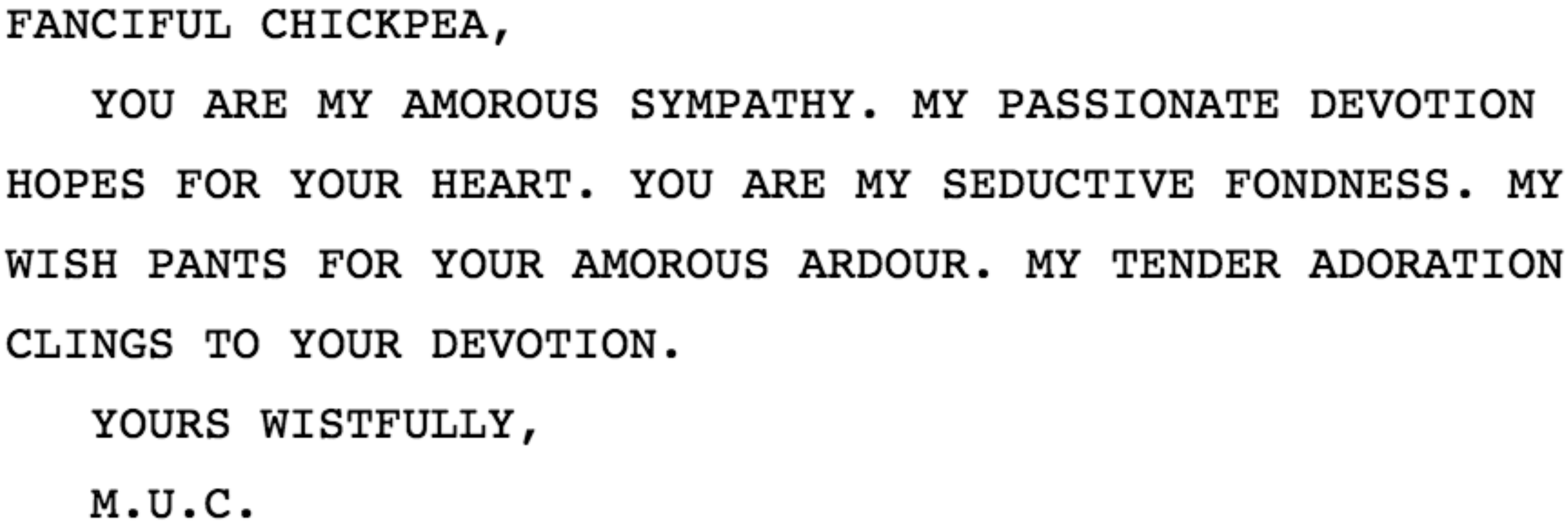
DARLING LOVE,

MY SEDUCTIVE APPETITE CLINGS TO YOUR AMBITION. MY RAPTURE LUSTS AFTER YOUR CRAVING. MY BURNING YEARNS FOR YOUR AMBITION. MY ENCHANTMENT IMPATIENTLY ADORES YOUR CURIOUS WISH. MY LOVING EAGERNESS IMPATIENTLY THIRSTS FOR YOUR LUST.

YOURS CURIOUSLY,

M.U.C.

# Manchester Univ. 1953

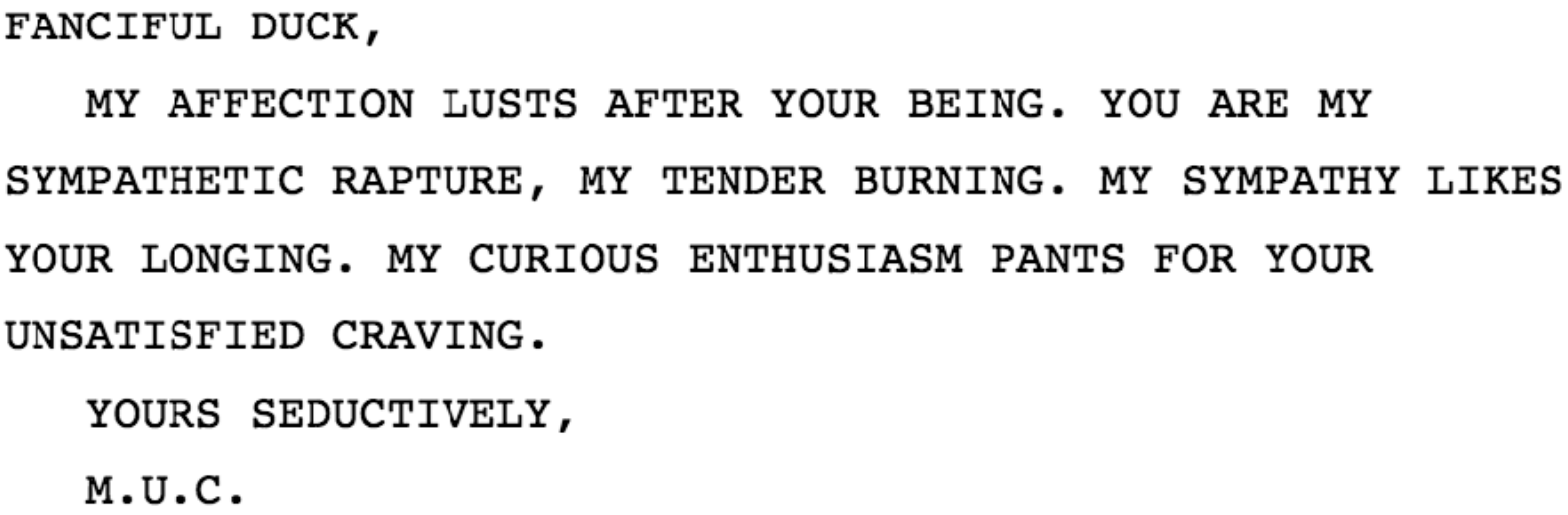


FANCIFUL CHICKPEA,

YOU ARE MY AMOROUS SYMPATHY. MY PASSIONATE DEVOTION HOPES FOR YOUR HEART. YOU ARE MY SEDUCTIVE FONDNESS. MY WISH PANTS FOR YOUR AMOROUS ARDOUR. MY TENDER ADORATION CLINGS TO YOUR DEVOTION.

YOURS WISTFULLY,
M.U.C.

# Manchester Univ. 1953



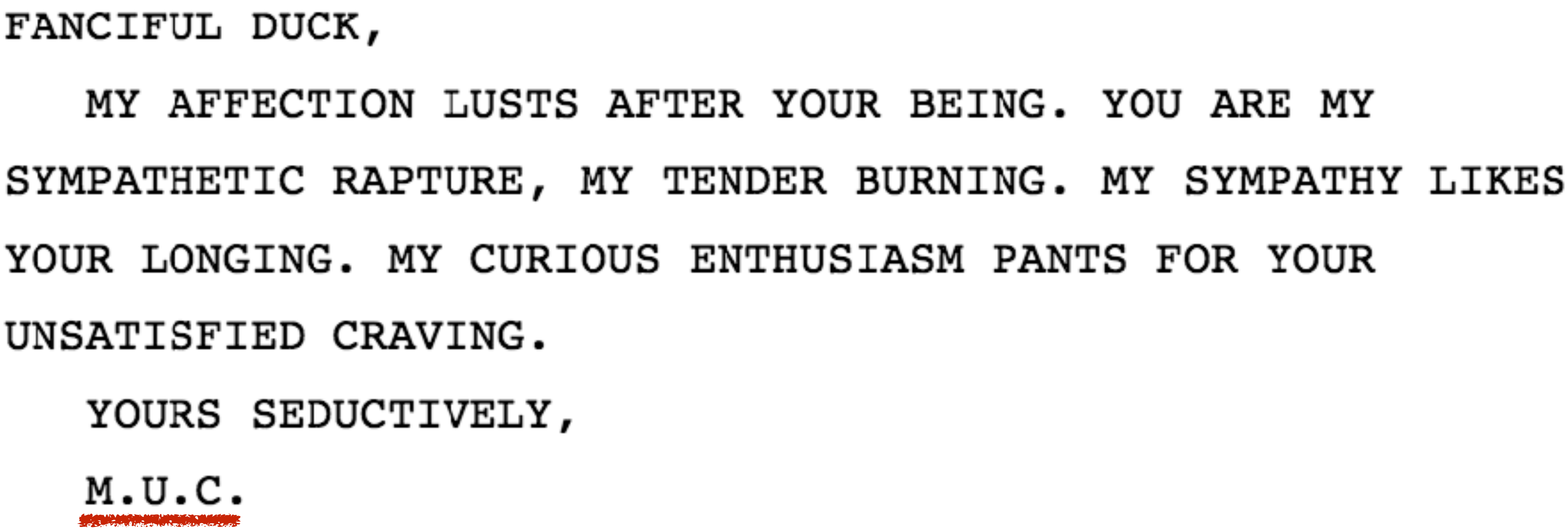
FANCIFUL DUCK,

MY AFFECTION LUSTS AFTER YOUR BEING. YOU ARE MY SYMPATHETIC RAPTURE, MY TENDER BURNING. MY SYMPATHY LIKES YOUR LONGING. MY CURIOUS ENTHUSIASM PANTS FOR YOUR UNSATISFIED CRAVING.

YOURS SEDUCTIVELY,

M.U.C.

# Manchester Univ. 1953



```
FANCIFUL DUCK,
   MY AFFECTION LUSTS AFTER YOUR BEING. YOU ARE MY
SYMPATHETIC RAPTURE, MY TENDER BURNING. MY SYMPATHY LIKES
YOUR LONGING. MY CURIOUS ENTHUSIASM PANTS FOR YOUR
UNSATISFIED CRAVING.
   YOURS SEDUCTIVELY,
   M.U.C.
```

Manchester Univ. Computer.

Produced by Strachey's "Love Letter" (1952)

Generated by the reimplementation in http://www.gingerbeardman.com/loveletter/

# Strachey's program

Implements a simple randomised algorithm:

1. Randomly pick two opening words.
2. Repeat the following five times:
    - Pick a sentence structure randomly.
    - Fill the structure with random words.
3. Randomly pick closing words.

# Strachey's

1. More randomness.

Implements a simple randomised algorithm:

1. Randomly pick two opening words.
2. Repeat the following ~~five times:~~ random N times
   - Pick a sentence structure randomly.
   - Fill the structure with random words.
3. Randomly pick closing words.

# Strachey's

1. More randomness.
2. Adjust randomness.
   Use data.

Implements a simple randomised algorithm:

1. Randomly pick two opening words.
2. Repeat the following ~~five times:~~ random N times
   - Pick a sentence structure randomly.
   - Fill the structure with random words.
3. Randomly pick closing words.

# 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

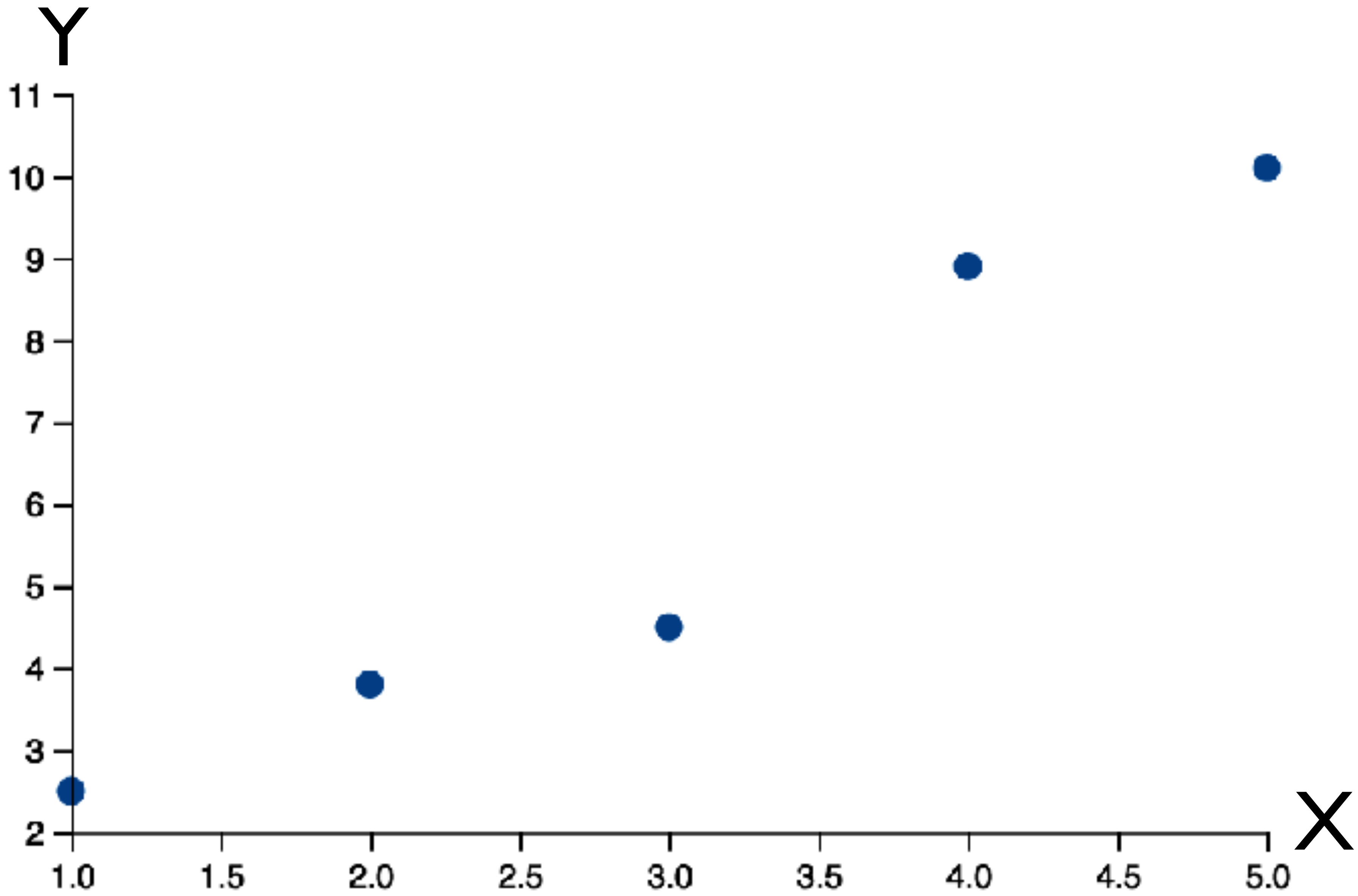
in a prob. prog. language

as a program

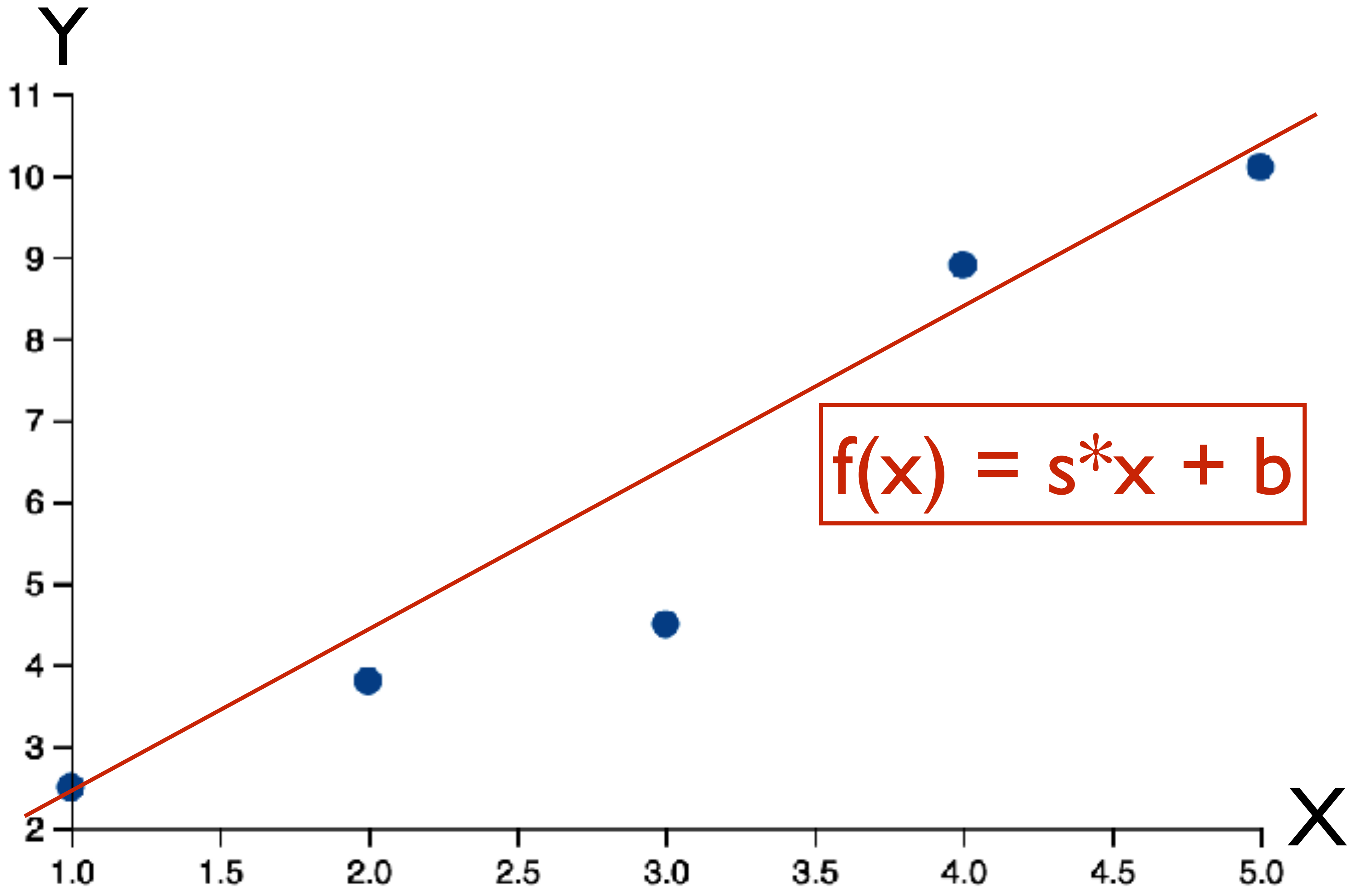
1. Develop a new probabilistic (generative) model.
2. ~~Design an inference algorithm for the model.~~
3. Using ~~the algo~~, fit the model to the data.

a generic inference algo.
of the language

# Line fitting

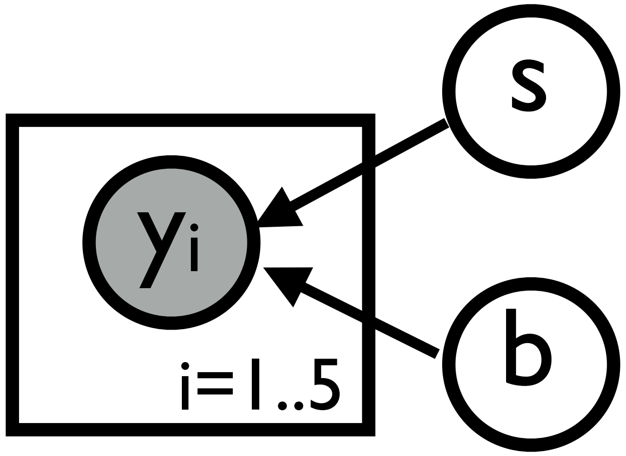


# Line fitting

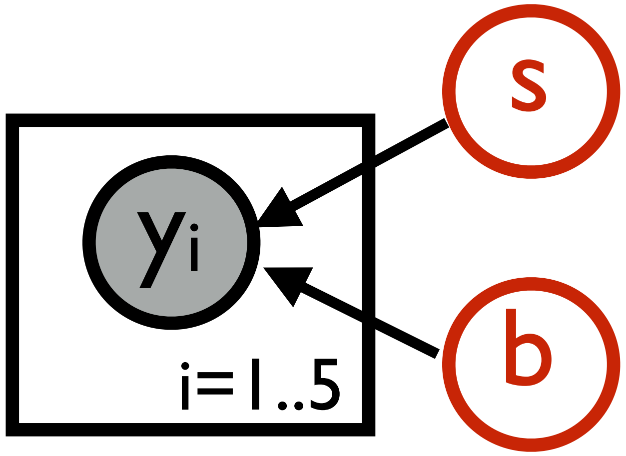


$f(x) = s*x + b$

# Bayesian generative model

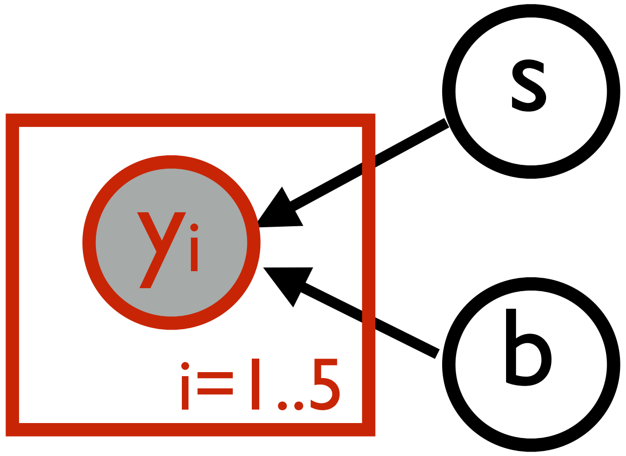


# Bayesian generative model



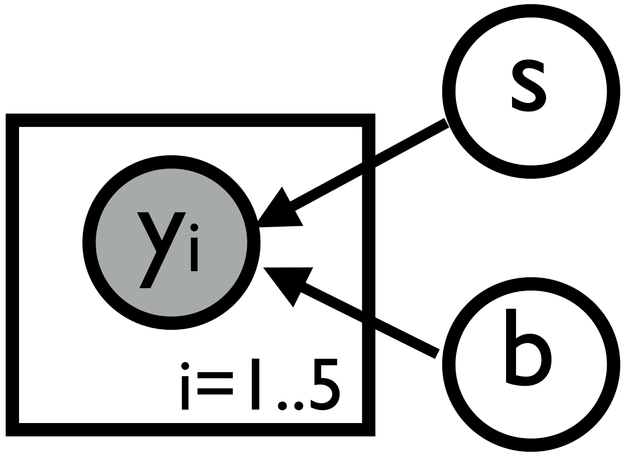
```
s     ~ normal(0, 10)
b     ~ normal(0, 10)
```

# Bayesian generative model



```
s    ~ normal(0, 10)
b    ~ normal(0, 10)
f(x) = s*x + b
yi    ~ normal(f(i), 1)
           where i = 1 .. 5
```

# Bayesian generative model



```
s      ~ normal(0, 10)
b      ~ normal(0, 10)
f(x)   = s*x + b
$y_i$     ~ normal(f(i), 1)
              where i = 1 .. 5
```

Q: posterior of (s,b) given $y_1$=2.5, …, $y_5$=10.1?

# Posterior of s and b given $y_i$'s

$$P(s, b \mid y_1, .., y_5) = \frac{P(y_1, .., y_5 \mid s,b) \times P(s,b)}{P(y_1, .., y_5)}$$

# Posterior of s and b given $y_i$'s

$$P(s, b \mid y_1, .., y_5) = \frac{P(y_1, .., y_5 \mid s,b) \times P(s,b)}{P(y_1, .., y_5)}$$

# Posterior of s and b given $y_i$'s

$$P(s, b \mid y_1, .., y_5) = \frac{P(y_1, .., y_5 \mid s,b) \times P(s,b)}{P(y_1, .., y_5)}$$

# Posterior of s and b given $y_i$'s

$$P(s, b \mid y_1, .., y_5) = \frac{P(y_1, .., y_5 \mid s,b) \times P(s,b)}{P(y_1, .., y_5)}$$

# Posterior of s and b given $y_i$'s

$$P(s, b \mid y_1, .., y_5) = \frac{P(y_1, .., y_5 \mid s,b) \times P(s,b)}{P(y_1, .., y_5)}$$

# Anglican program

```
(let [s (sample (normal 0 10))
      b (sample (normal 0 10))
      f (fn [x] (+ (* s x) b))]
```

# Anglican program

```
(let [s (sample (normal 0 10))
      b (sample (normal 0 10))
      f (fn [x] (+ (* s x) b))]

  (observe (normal (f 1) 1) 2.5)
  (observe (normal (f 2) 1) 3.8)
  (observe (normal (f 3) 1) 4.5)
  (observe (normal (f 4) 1) 8.9)
  (observe (normal (f 5) 1) 10.1)
```

# Anglican program

```
(let [s (sample (normal 0 10))
      b (sample (normal 0 10))
      f (fn [x] (+ (* s x) b))]

  (observe (normal (f 1) 1) 2.5)
  (observe (normal (f 2) 1) 3.8)
  (observe (normal (f 3) 1) 4.5)
  (observe (normal (f 4) 1) 8.9)
  (observe (normal (f 5) 1) 10.1)

  (predict :sb [s b]))
```

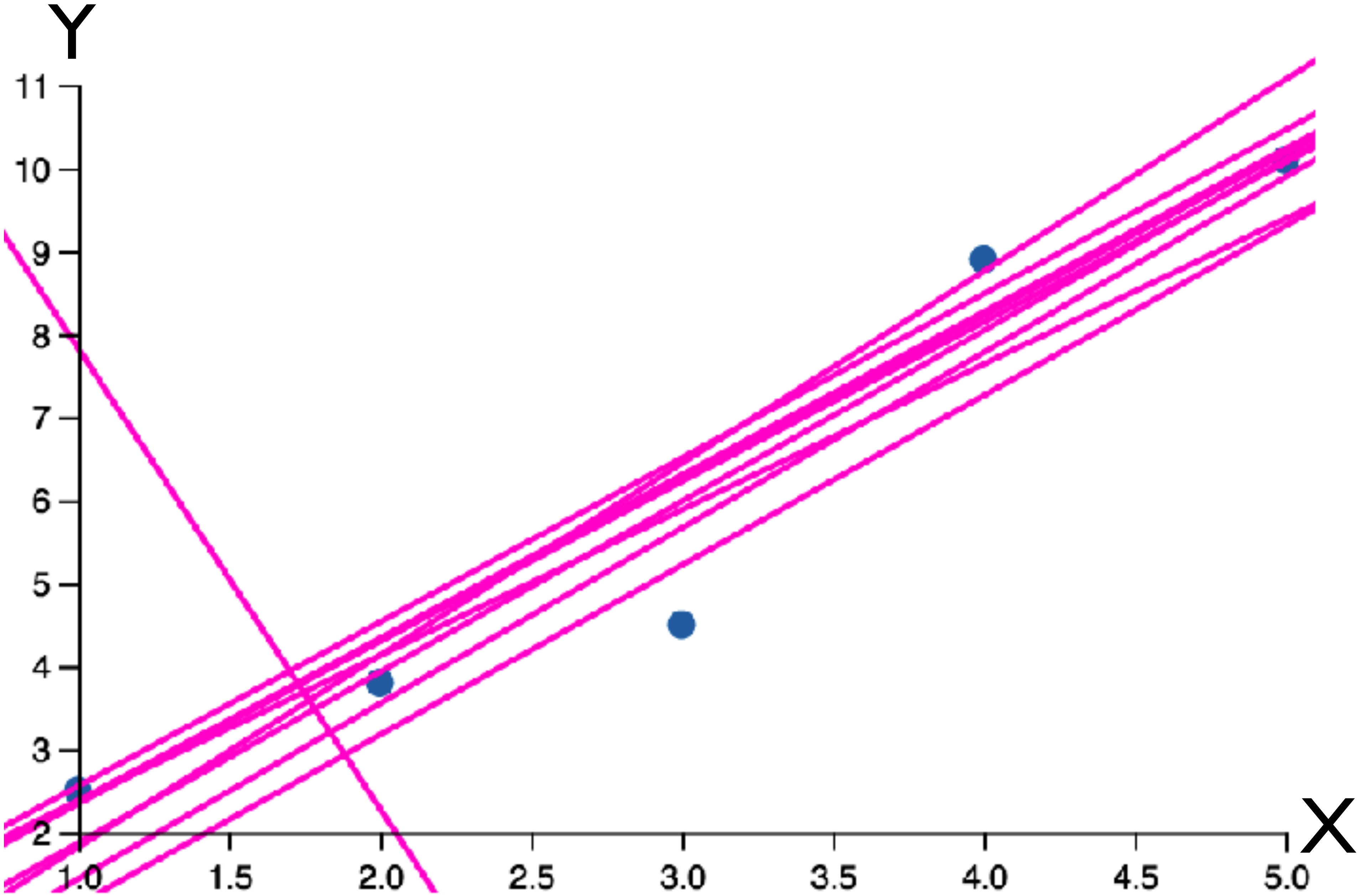
# Anglican program

```
(let [s (sample (normal 0 10))
      b (sample (normal 0 10))
      f (fn [x] (+ (* s x) b))]

  (observe (normal (f 1) 1) 2.5)
  (observe (normal (f 2) 1) 3.8)
  (observe (normal (f 3) 1) 4.5)
  (observe (normal (f 4) 1) 8.9)
  (observe (normal (f 5) 1) 10.1)

  (predict :sb [s b]))
```

# Samples from posterior

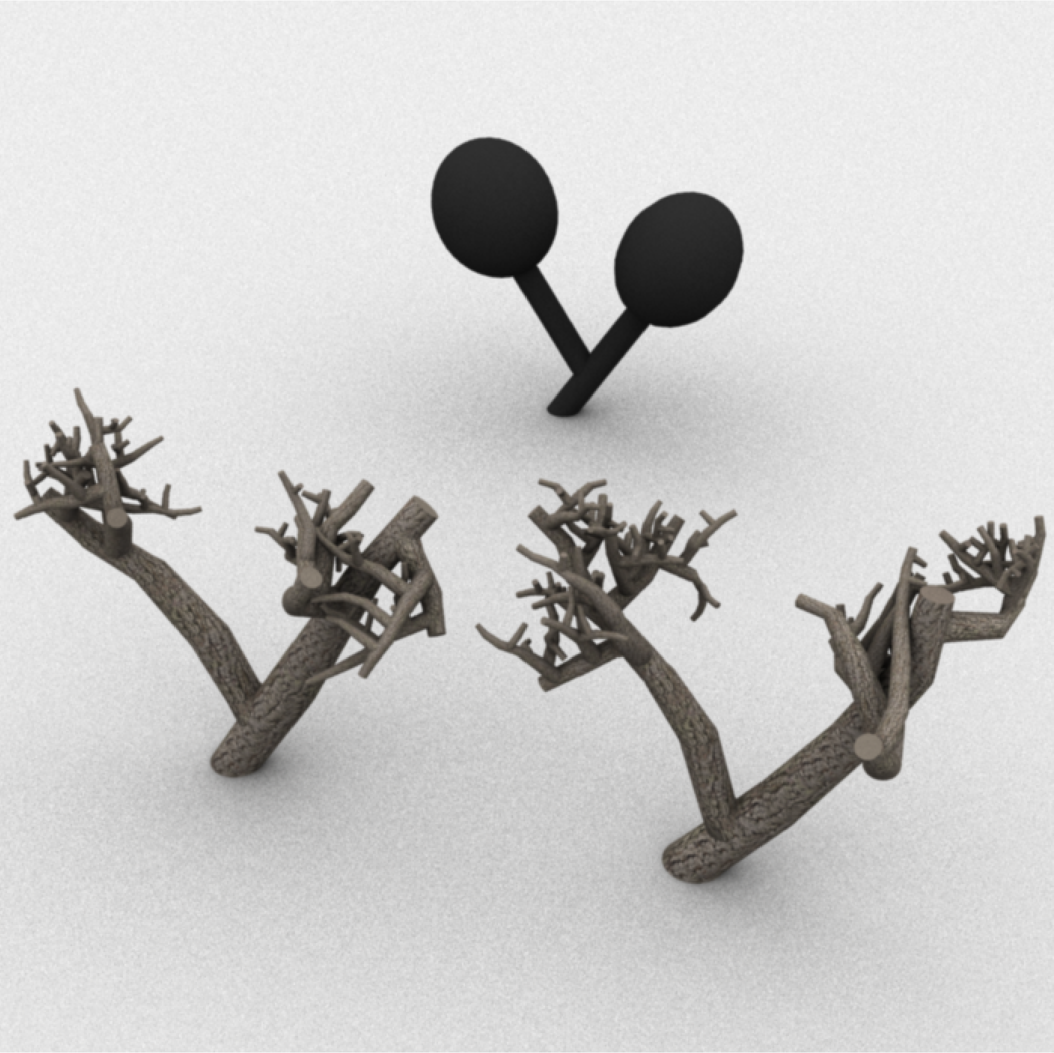


# Why should one care about prob. programming?

# My favourite answer

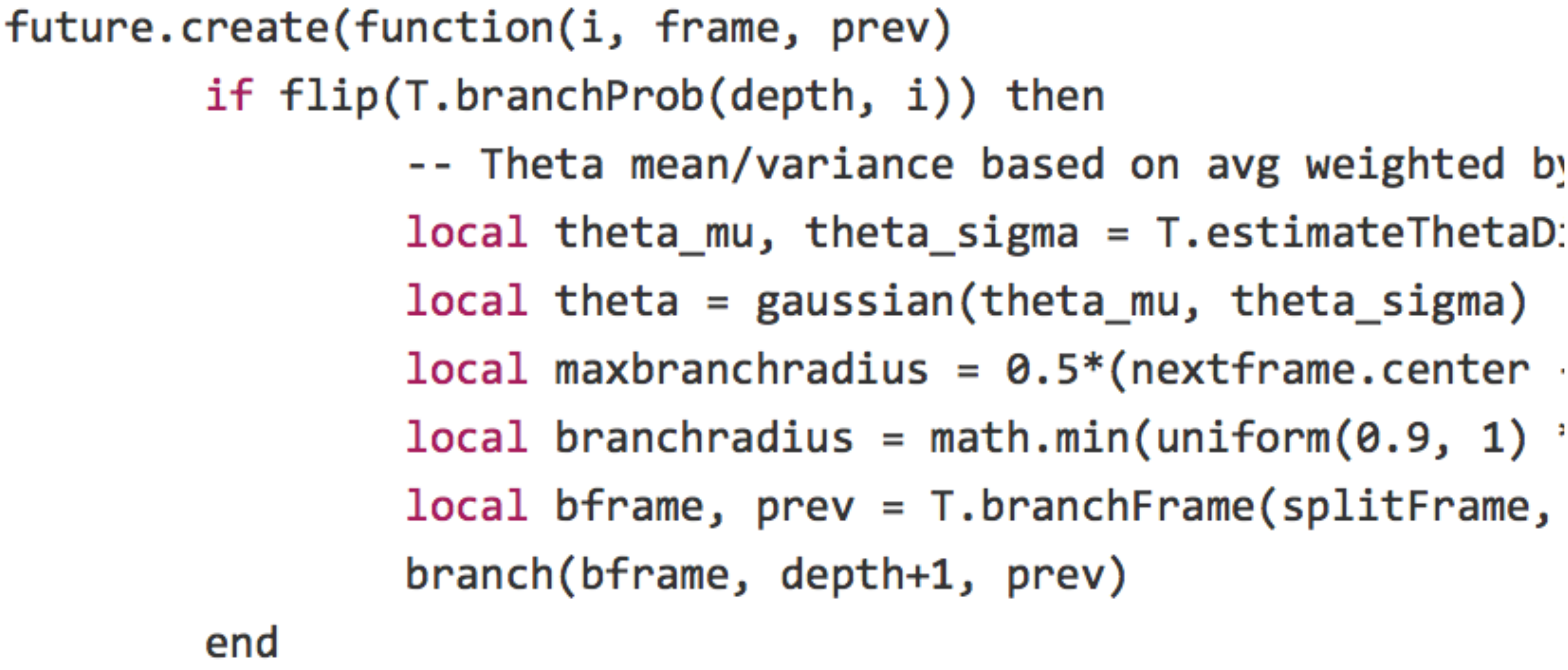
“Because probabilistic programming is a good way to build an AI.” (My ML colleague)

# Procedural modelling



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

# Procedural modelling

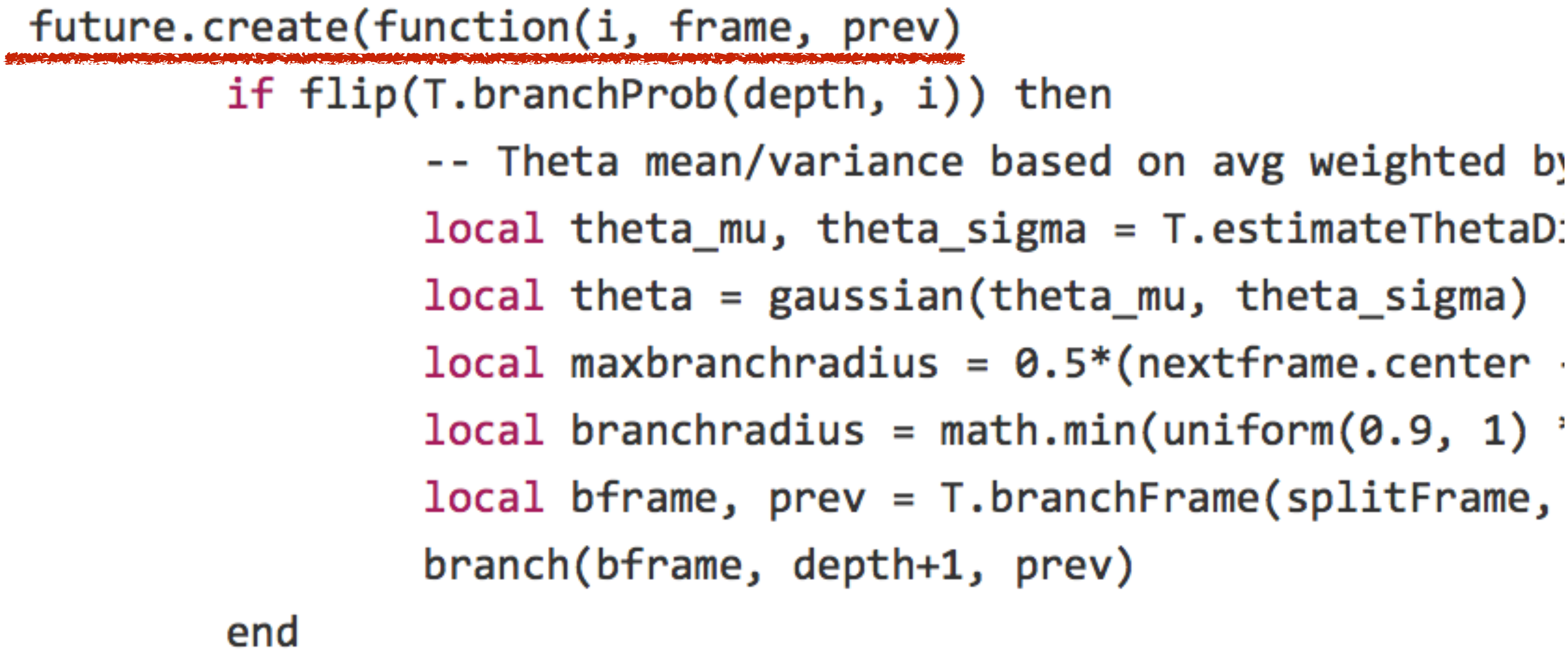


```
future.create(function(i, frame, prev)
        if flip(T.branchProb(depth, i)) then
                -- Theta mean/variance based on avg weighted b
                local theta_mu, theta_sigma = T.estimateThetaD
                local theta = gaussian(theta_mu, theta_sigma)
                local maxbranchradius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

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

# Procedural

Asynchronous function call via future

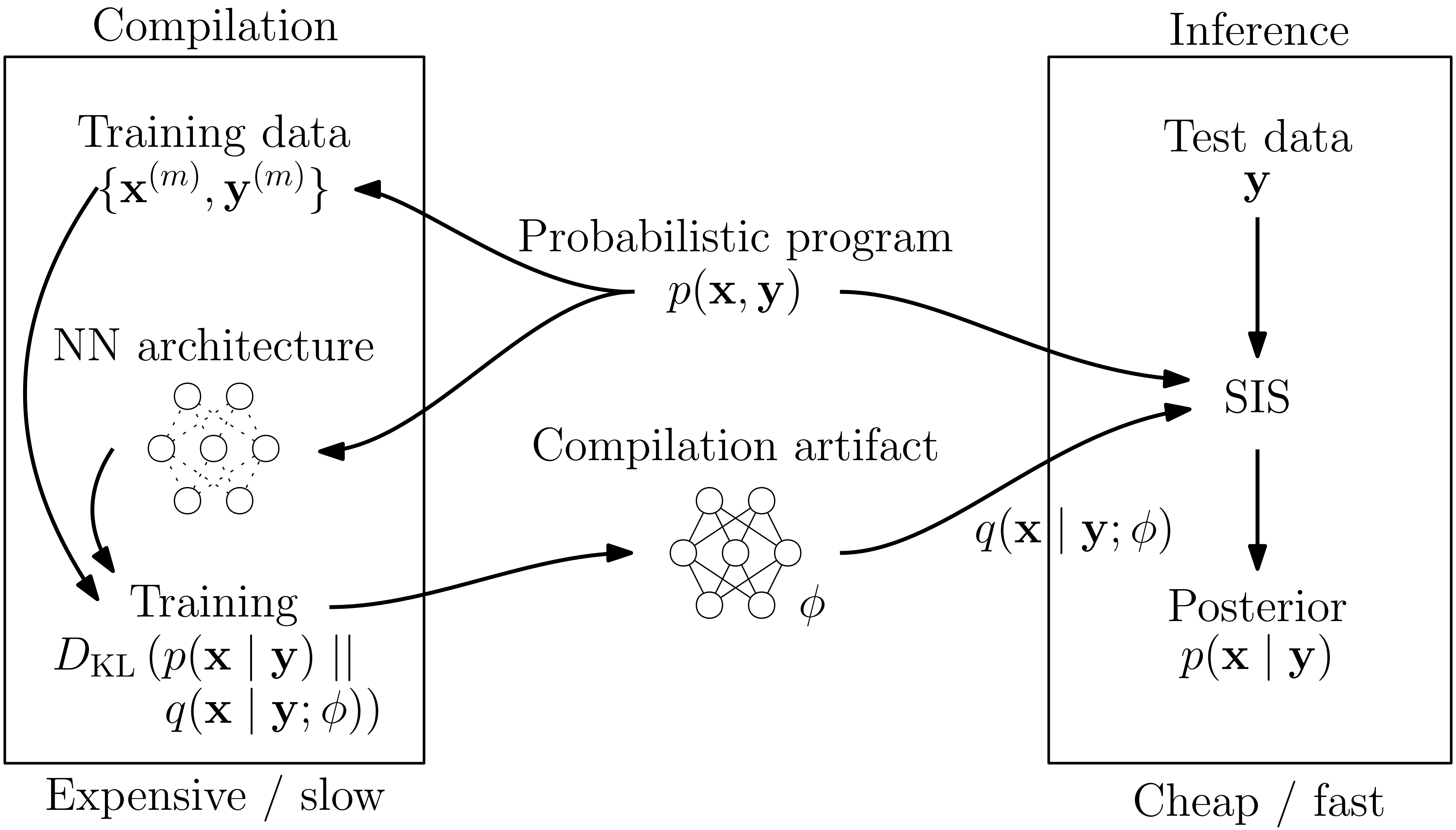


```
future.create(function(i, frame, prev)
        if flip(T.branchProb(depth, i)) then
                -- Theta mean/variance based on avg weighted b
                local theta_mu, theta_sigma = T.estimateThetaD
                local theta = gaussian(theta_mu, theta_sigma)
                local maxbranchradius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

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

# Captcha solving

Le, Baydin, Wood [2016]



Compilation

Training data

$\{\mathbf{x}^{(m)}, \mathbf{y}^{(m)}\}$

NN architecture

Training

$D_{\mathrm{KL}}\left(p(\mathbf{x} \mid \mathbf{y}) \mid\mid q(\mathbf{x} \mid \mathbf{y}; \phi)\right)$

Expensive / slow

Probabilistic program

$p(\mathbf{x}, \mathbf{y})$

Compilation artifact

$\phi$

$q(\mathbf{x} \mid \mathbf{y}; \phi)$

Inference

Test data

$\mathbf{y}$

SIS

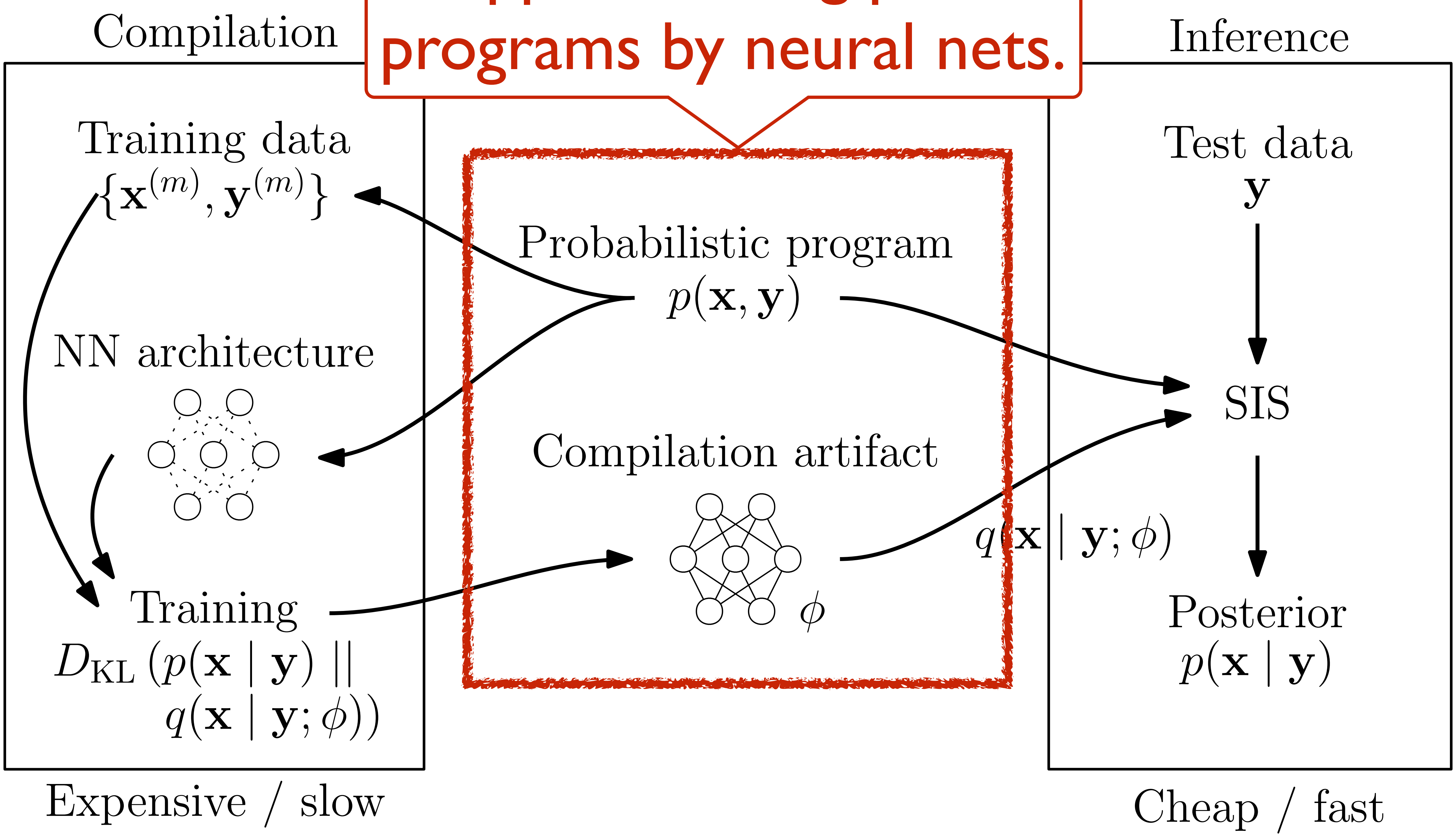
Posterior

$p(\mathbf{x} \mid \mathbf{y})$

Cheap / fast

Le, Baydin, Wood [2016]

Approximating prob. programs by neural nets.



Compilation

Training data
$\{\mathbf{x}^{(m)}, \mathbf{y}^{(m)}\}$

NN architecture

Training
$D_{\mathrm{KL}}\left(p(\mathbf{x} \mid \mathbf{y}) \mid\mid q(\mathbf{x} \mid \mathbf{y}; \phi)\right)$

Expensive / slow

Probabilistic program
$p(\mathbf{x}, \mathbf{y})$

Compilation artifact
$\phi$

$q(\mathbf{x} \mid \mathbf{y}; \phi)$

Inference

Test data
$\mathbf{y}$

SIS

Posterior
$p(\mathbf{x} \mid \mathbf{y})$

Cheap / fast

Le, Baydin, Wood [2016]

# Nonparametric Bayesian: Indian buffer process

```
(define (ibp-stick-breaking-process concentration base-measure)
  (let ((sticks (mem (lambda j (random-beta 1.0 concentration))))
        (atoms  (mem (lambda j (base-measure)))))
    (lambda ()
      (let loop ((j 1) (dualstick (sticks 1))}
        (append (if (flip dualstick)            ;; with prob. dualstick
                    (atoms j)                   ;;   add feature j
                    ’())                        ;; otherwise, next stick
                (loop (+ j 1) (* dualstick (sticks (+ j 1)))) ))))))
```

Roy et al. 2008

# Nonparametric Bayesian: Indian buffer process

```
(define (ibp-stick-breaking-process concentration base-measure)
  (let ((sticks (mem (lambda j (random-beta 1.0 concentration))))
        (atoms  (mem (lambda j (base-measure)))))
    (lambda ()
      (let loop ((j  ) (dualstick (sticks 1))}
        (append (if  flip dualstick)            ;; with prob. dualstick
                     atoms j)                   ;;    add feature j
                      ))                        ;; otherwise, next stick
                (l    (+ j 1) (* dualstick (sticks (+ j 1))) ))))))
```

Lazy infinite array

Roy et al. 2008

# Nonparametric B
# Indian buffer p

Higher-order parameter

```
(define (ibp-stick-breaking-process concentration base-measure)
  (let ((sticks (mem (lambda j (random-beta 1.0 concentration))))
        (atoms  (mem (lambda j (base-measure)))))
    (lambda ()
      (let loop ((j 1) (dualstick (sticks 1))}
        (append (if (flip dualstick)                    ;; with prob. dualstick
                    (atoms j)                           ;;    add feature j
                    '())                                ;; otherwise, next stick
                (loop (+ j 1) (* dualstick (sticks (+ j 1)))) ))))))
```

Roy et al.  2008

# My research : Denotational semantics

Joint work with Chris Heunen, Ohad Kammar, Sam Staton, Frank Wood
[LICS 2016]

```
(let [s (sample (normal 0 10))
      b (sample (normal 0 10))
      f (fn [x] (+ (* s x) b))]

  (observe (normal (f 1) 1) 2.5)
  (observe (normal (f 2) 1) 3.8)
  (observe (normal (f 3) 1) 4.5)
  (observe (normal (f 4) 1) 8.9)
  (observe (normal (f 5) 1) 10.1)

  (predict :sb [s b]))
```

```
(let [s (sample (normal 0 10))
      b (sample (normal 0 10))
      f (fn [x] (+ (* s x) b))]

  (observe (normal (f 1) 1) 2.5)
  (observe (normal (f 2) 1) 3.8)
  (observe (normal (f 3) 1) 4.5)
  (observe (normal (f 4) 1) 8.9)
  (observe (normal (f 5) 1) 10.1)

  ~~(predict :sb [s b])~~)
     (predict :f f)
```

```
(let [s (sample (normal 0 10))
      b (sample (normal 0 10))
      f (fn [x] (+ (* s x) b))]

 (observe (normal (f 1) 1) 2.5)
 (observe (normal (f 2) 1) 3.8)
 (observe (normal (f 3) 1) 4.5)
 (observe (normal (f 4) 1) 8.9)
 (observe (normal (f 5) 1) 10.1)

 ~~(predict :sb [s b])~~)
  (predict :f f)
```

Generates a random function of type R→R.
But its mathematical meaning is not clear.

# Measurability issue

- Measure theory is the foundation of probability theory that avoids paradoxes.
- Silent about high-order functions.
  - [Halmos] ev(f,a) = f(a) is not measurable.
  - The category of measurable sets is not CCC.
- But Anglican supports high-order functions.

# Use category theory to extend measure theory.

$$\begin{tikzcd}
\text{Meas} \arrow[d, "\text{Monad}"'] \\
\text{Meas}
\end{tikzcd}$$

# Use category theory to extend measure theory.

$$\begin{tikzcd}
\mathrm{Meas} \arrow[r, "\text{Yoneda Embedding}"] \arrow[d, "\text{Monad}"'] & {[\mathrm{Meas}^{op}, \mathrm{Set}]_{\Pi}} \\
\mathrm{Meas} \arrow[r, "\text{Yoneda Embedding}"'] & {[\mathrm{Meas}^{op}, \mathrm{Set}]_{\Pi}}
\end{tikzcd}$$

# Use category theory to extend measure theory.

$$\begin{tikzcd}
\mathrm{Meas} \arrow[r, "\text{Yoneda Embedding}"] \arrow[d, "\text{Monad}"'] & {[\mathrm{Meas}^{op}, \mathrm{Set}]_{\Pi}} \arrow[d, dashed, "\text{Left Kan Extension}"] \\
\mathrm{Meas} \arrow[r, "\text{Yoneda Embedding}"'] & {[\mathrm{Meas}^{op}, \mathrm{Set}]_{\Pi}}
\end{tikzcd}$$

# Use category theory to extend measure theory.

$$\begin{tikzcd}
\text{Meas} \arrow[r, "\text{Yoneda Embedding}"] \arrow[d, "\text{Monad}"'] & {[\text{Meas}^{op}, \text{Set}]_{\Pi}} \arrow[d, "\text{Left Kan Extension}"] \\
\text{Meas} \arrow[r, "\text{Yoneda Embedding}"'] & {[\text{Meas}^{op}, \text{Set}]_{\Pi}}
\end{tikzcd}$$

# Use category theory to extend measure theory.

Enough structure for function types

$$\begin{tikzcd}
\text{Meas} \arrow[r, "\text{Yoneda Embedding}"] \arrow[d, "\text{Monad}"'] & {[\text{Meas}^{op}, \text{Set}]_{\Pi}} \arrow[d, "\text{Left Kan Extension}"] \\
\text{Meas} \arrow[r, "\text{Yoneda Embedding}"'] & {[\text{Meas}^{op}, \text{Set}]_{\Pi}}
\end{tikzcd}$$

# Use category theory to extend measure theory.

$$\begin{tikzcd}
\mathrm{Meas} \arrow[r, "\text{Yoneda Embedding}"] \arrow[d, "\text{Monad}"'] & {[\mathrm{Meas}^{op}, \mathrm{Set}]_{\Pi}} \arrow[d, "\text{Left Kan Extension}"] \\
\mathrm{Meas} \arrow[r, "\text{Yoneda Embedding}"'] & {[\mathrm{Meas}^{op}, \mathrm{Set}]_{\Pi}}
\end{tikzcd}$$

Enough structure for function types

Preserves nearly all the structures

[Question] Are all definable functions from R to R in a high-order probabilistic PL measurable?

Our semantics says that the answer is yes for a core call-by-value language, such as Anglican.

The monad $\underline{M}([\![R \to R]\!])$ at $[\![R \to R]\!]$ consists of:

equivalence classes of measurable functions $f : \Omega \times R \to R$ for probability spaces $\Omega$.

The function f is what probabilists call a measurable stochastic process.

The extended monad $\underline{M}$ describes computations with dynamically allocated read-only variables.

$$\underline{M}(T)(w) =$$
$$\{ [(a, f)]_{\sim} \ | \ \exists v.\ a \in T(v) \wedge f : w \to_{m} \mathrm{Prob}(v) \}$$

The extended monad $\underline{M}$ describes computations with dynamically allocated read-only variables.

$$\underline{M}(T)(w) =$$

$$\{\ [(a, f)]_{\sim}\ \mid\ \exists v.\ a \in T(v)\ \wedge\ f : w \rightarrow_m \mathrm{Prob}(v)\ \}$$

T is the type of a value.

The extended monad $\underline{M}$ describes computations with dynamically allocated read-only variables.

$$\underline{M}(T)(w) =$$
$$\{\ [(a, f)]_{\sim}\ \mid\ \exists v.\ a \in T(v) \wedge f : w \rightarrow_m \mathrm{Prob}(v)\ \}$$

T is the type of a value.

w represents a space of all random vars so far.

The extended monad $\underline{M}$ describes computations with dynamically allocated read-only variables.

$$\underline{M}(T)(w) =$$
$$\{\ [(a, f)]_{\sim}\ \mid\ \exists v.\ a \in T(v) \wedge\ f : w \rightarrow_{m} \mathrm{Prob}(v)\ \}$$

T is the type of a value.

w represents a space of all random vars so far.

v extends w with new random variables according to f.

Try a probabilistic prog. language. It is fun.

- Anglican:
  http://www.robots.ox.ac.uk/~fwood/anglican/index.html

- WebPPL:
  http://webppl.org/