Introduction to probabilistic programming

Frank Wood

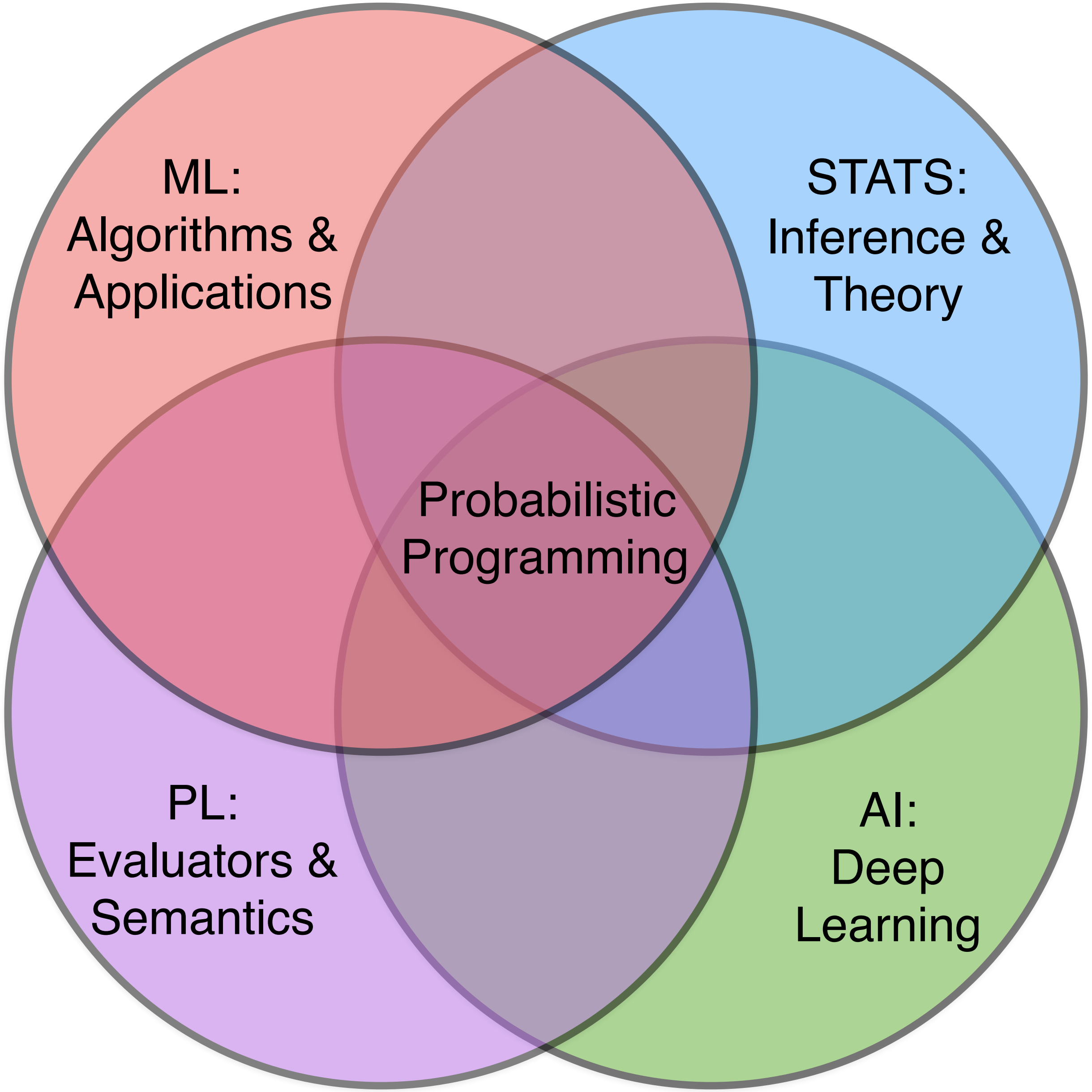
fwood@cs.ubc.ca

Objectives For Today

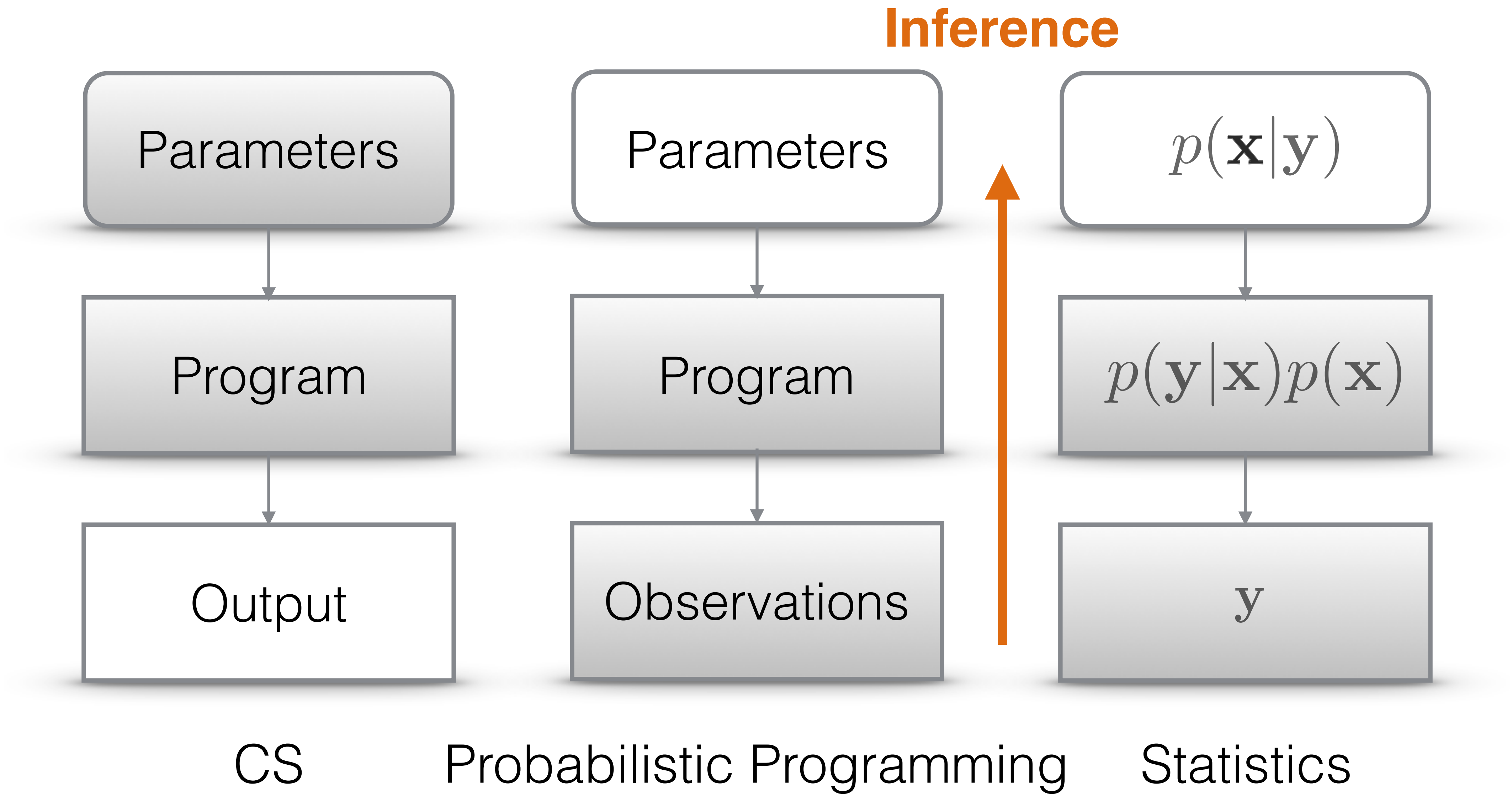
Get you to

- Understand what probabilistic programming is
- Think generatively
- Understand inference
  - Importance sampling
  - SMC
  - MCMC
- Understand something about how modern, performant higher-order probabilistic programming systems are implemented at a very high level

Probabilistic Programming



Intuition


Inference
Parameters
Program
Output
Parameters
Program
Observations
$p(\mathbf{x}|\mathbf{y})$
$p(\mathbf{y}|\mathbf{x})p(\mathbf{x})$
$\mathbf{y}$
CS
Probabilistic Programming
Statistics

Probabilistic *Programs*

"Probabilistic programs are usual functional or imperative programs with two added constructs:

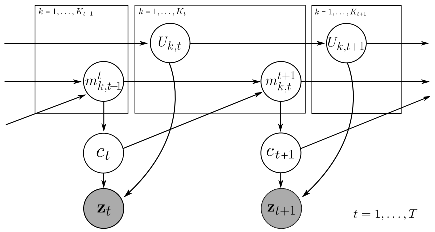
(1) the ability to draw values at random from distributions, and

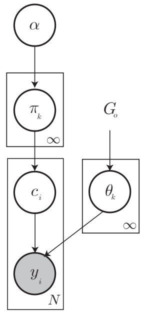
(2) the ability to **condition** values of variables in a program via observations."

Gordon, Henzinger, Nori, and Rajamani
"Probabilistic programming." In Proceedings of On The Future of Software Engineering (2014).

Key Ideas

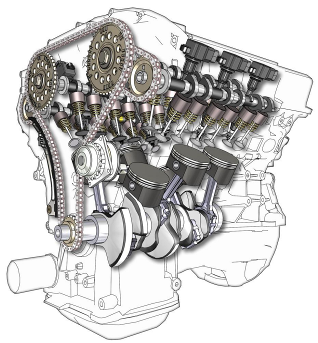
Models





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

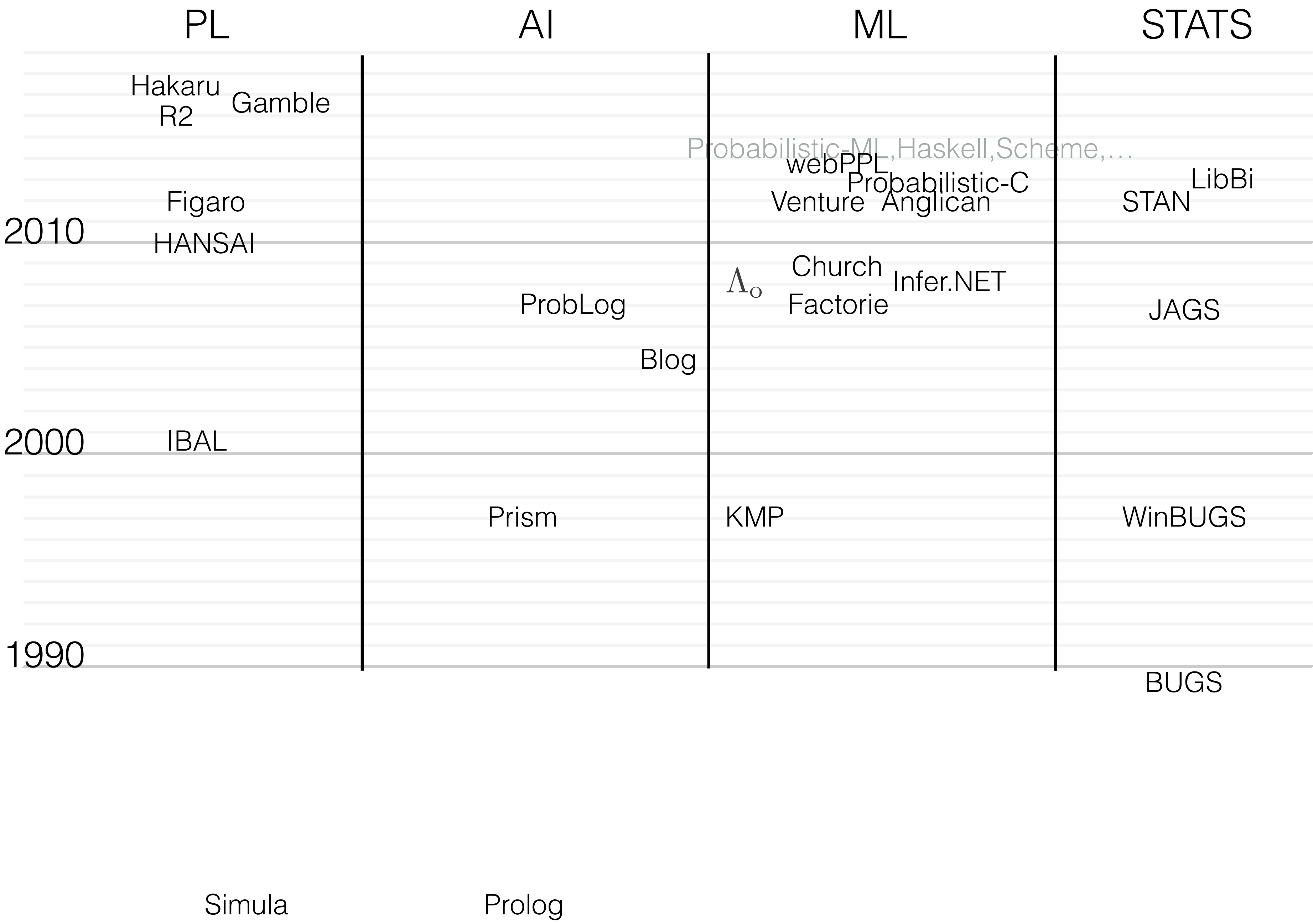
Programming Language Abstraction Layer



$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{y})}$$

Evaluators that automate Bayesian *inference*

Long History



Existing Languages

Graphical Models

BUGS

STAN

Factor Graphs

Factorie

Infer.NET

Infinite Dimensional Parameter Space Models

Anglican

WebPPL

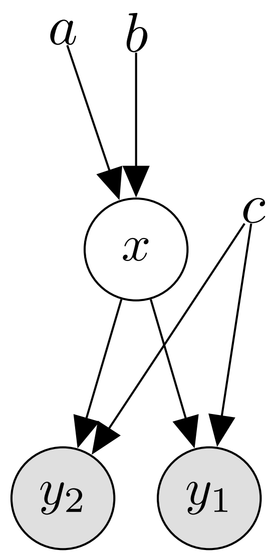
Unsupervised Deep Learning

PYRO

ProbTorch

BUGS

```
model
{
  x ~ dnorm(1, 1/5)
  for(i in 1:N) {
    y[i] ~ dnorm(x, 1/2)
  }
}
"N" <- 2
"y" <- c(9, 8)
```



- Language restrictions
  - Bounded loops
  - No branching
- Model class
  - Finite graphical models
- Inference - sampling
  - Gibbs

Spiegelhalter et al. "BUGS: Bayesian inference using Gibbs sampling, Version 0.50." Cambridge 1995.

STAN : Finite Dimensional Differentiable Distributions

```
parameters {
    real xs[T];
}
model {
    xs[1] ~ normal(0.0, 1.0);
    for (t in 2:T)
        xs[t] ~ normal(a * xs[t - 1], q);
    for (t in 1:T)
        ys[t] ~ normal(xs[t], 1.0);
}
```

$\rightarrow \nabla_{\mathbf{x}} \log p(\mathbf{x}, \mathbf{y})$

- Language restrictions
  - Bounded loops
  - No discrete random variables*
- Model class
  - Finite dimensional differentiable distributions
- Inference
  - Hamiltonian Monte Carlo
    - Reverse-mode automatic differentiation
  - Black box variational inference, etc.

Goal
$p(\mathbf{x}|\mathbf{y})$

STAN Development Team "Stan: A C++ Library for Probability and Sampling." 2014.

Modeling language desiderata

- Unrestricted language (C++, Python, Lisp, etc.)
  - “Open-universe” / infinite dim. parameter spaces
  - Mixed variable types
- Pros
  - Unfettered access to existing libraries
  - Easily extensible
- Cons
  - Inference is going to be harder
  - More ways to shoot yourself in the foot

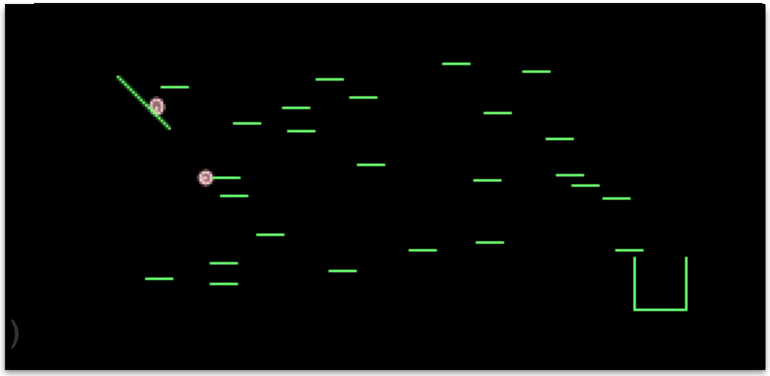
Deterministic Simulation and Other Libraries

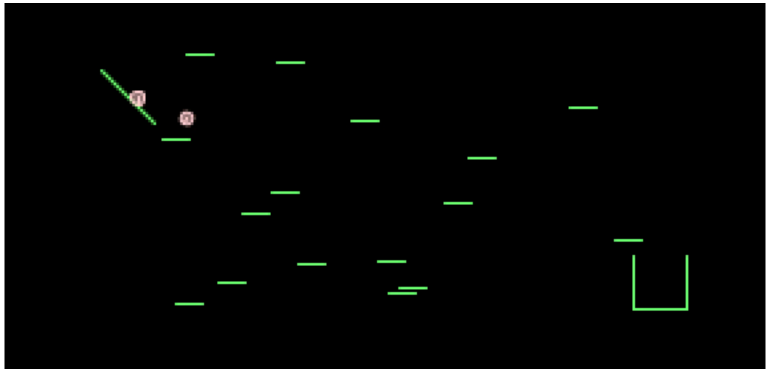
```
(defquery arrange-bumpers []
    (let [bumper-positions []

          ;; code to simulate the world
          world (create-world bumper-positions)
          end-world (simulate-world world)
          balls (:balls end-world)

          ;; how many balls entered the box?
          num-balls-in-box (balls-in-box end-world)]

      {:balls balls
       :num-balls-in-box num-balls-in-box
       :bumper-positions bumper-positions}))
```

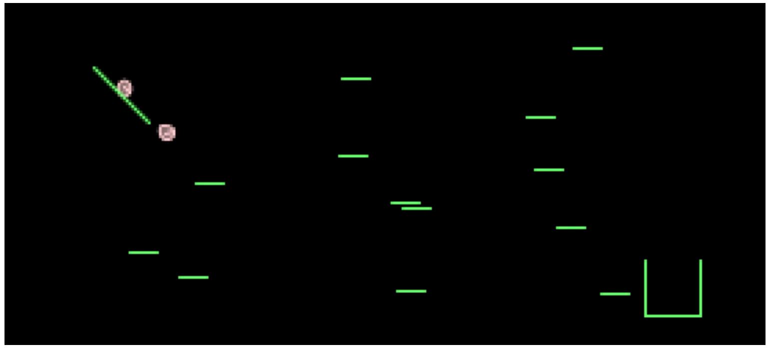
goal: “world” that puts ~20% of balls in box…

Open Universe Models and Nonparametrics

```
(defquery arrange-bumpers []
    (let [number-of-bumpers (sample (poisson 20))
          bumpydist (uniform-continuous 0 10)
          bumpxdist (uniform-continuous -5 14)
          bumper-positions (repeatedly
                             number-of-bumpers
                             #(vector (sample bumpxdist)
                                      (sample bumpydist)))

          ;; code to simulate the world
          world (create-world bumper-positions)
          end-world (simulate-world world)
          balls (:balls end-world)

          ;; how many balls entered the box?
          num-balls-in-box (balls-in-box end-world)]

      {:balls balls
       :num-balls-in-box num-balls-in-box
       :bumper-positions bumper-positions}))
```

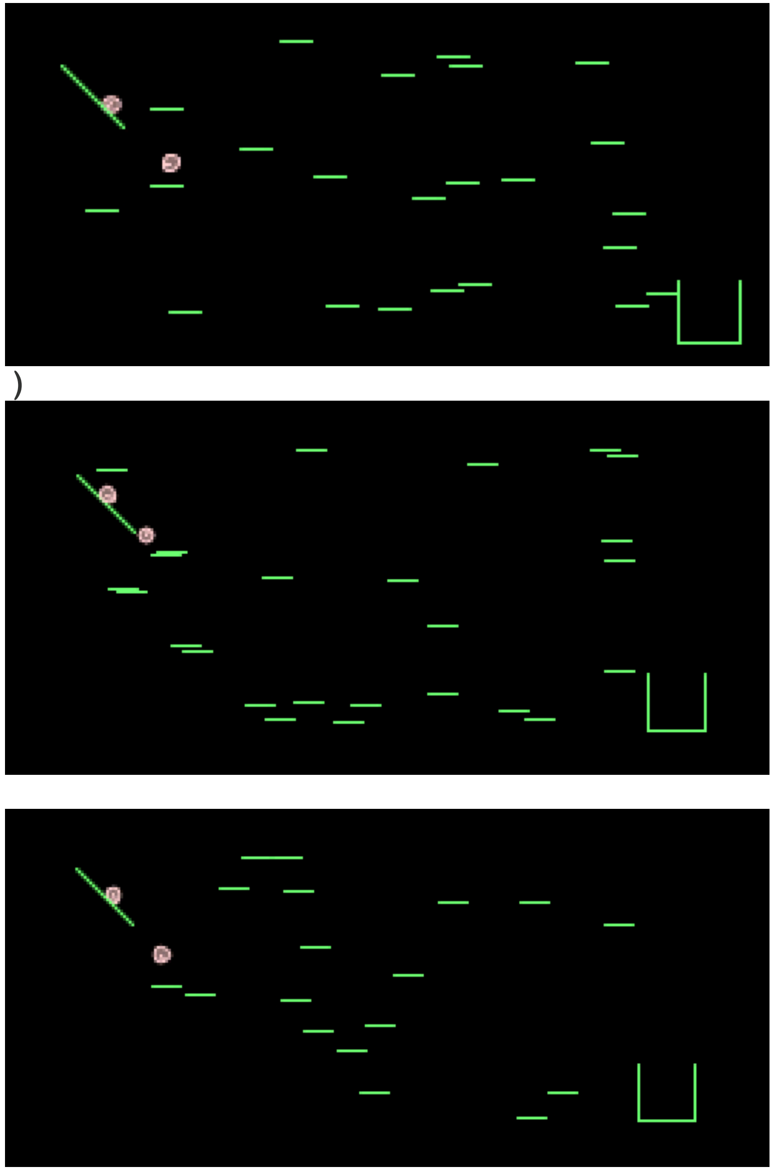






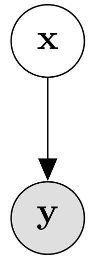
Conditional (Stochastic) Simulation

```
(defquery arrange-bumpers []
    (let [number-of-bumpers (sample (poisson 20))
          bumpydist (uniform-continuous 0 10)
          bumpxdist (uniform-continuous -5 14)
          bumper-positions (repeatedly
                             number-of-bumpers
                             #(vector (sample bumpxdist)
                                      (sample bumpydist)))

          ;; code to simulate the world
          world (create-world bumper-positions)
          end-world (simulate-world world)
          balls (:balls end-world)

          ;; how many balls entered the box?
          num-balls-in-box (balls-in-box end-world)

          obs-dist (normal 4 0.1)]

      (observe obs-dist num-balls-in-box)

      {:balls balls
       :num-balls-in-box num-balls-in-box
       :bumper-positions bumper-positions}))
```



New Kinds of Models

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{y})}$$



$\mathbf{x}$	$\mathbf{y}$
program source code	program return value
scene description	image
policy and world	rewards
cognitive process	observed behavior
simulation	simulator output

Thinking Generatively

CAPTCHA breaking

SMKBDF

Can you write a program to do this?

$\mathbf{x}$	$\mathbf{y}$
text	image

Mansinghka, Kulkarni, Perov, and Tenenbaum
"Approximate Bayesian image interpretation using generative probabilistic graphics programs." NIPS (2013).

Captcha Generative Model

```
(defm sample-char []
 {:symbol (sample (uniform ascii))
  :x-pos (sample (uniform-cont 0.0 1.0))
  :y-pos (sample (uniform-cont 0.0 1.0))
  :size (sample (beta 1 2))
  :style (sample (uniform-dis styles))
  …})
```

```
(defm sample-captcha []
 (let [n-chars (sample (poisson 4))
       chars (repeatedly n-chars
                         sample-char)
     noise (sample salt-pepper)
     …]
  gen-image))
```

Conditioning

```
(defquery captcha [true-image]
 (let [gen-image (sample-captcha)]
  (observe (similarity-kernel gen-image)
           true-image)
  gen-image))
```

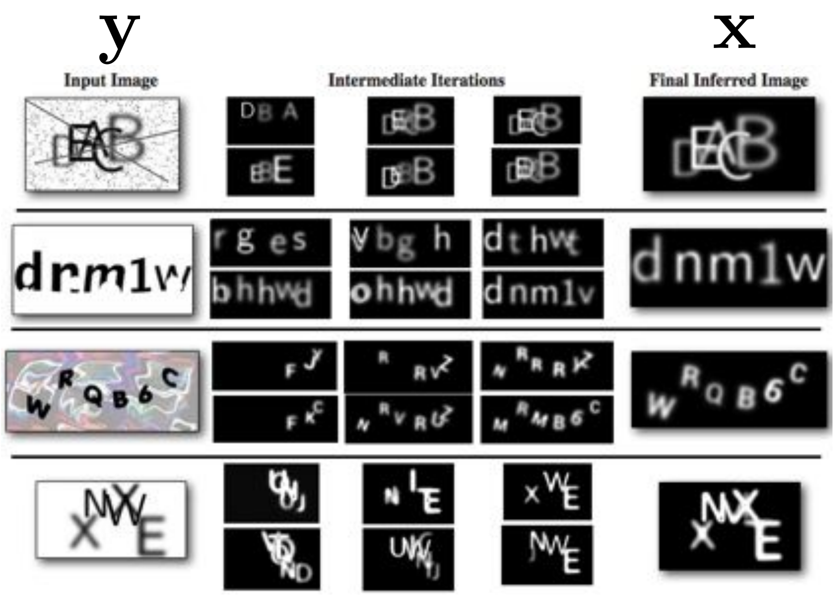
Generative Model

```
(doquery :ipmcmc captcha true-image)
```

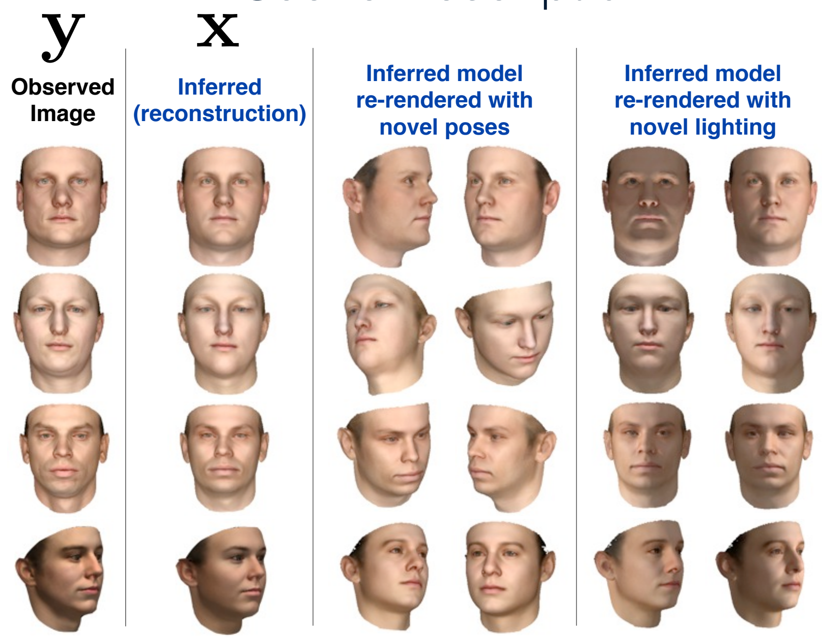
Inference

Perception / Inverse Graphics

Captcha Solving



Scene Description



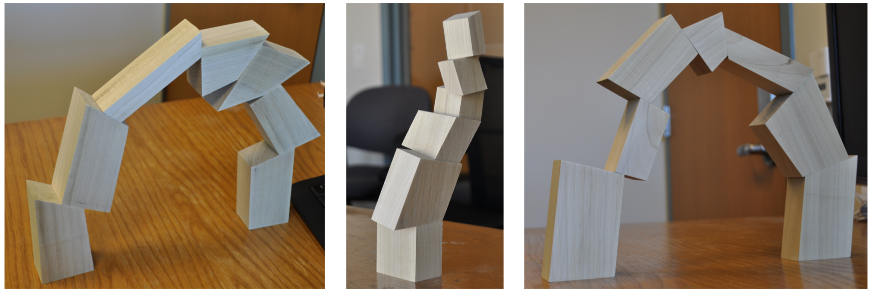
$\mathbf{x}$	$\mathbf{y}$
scene description	image

Mansinghka, Kulkarni, Perov, and Tenenbaum. "Approximate Bayesian image interpretation using generative probabilistic graphics programs." NIPS (2013).

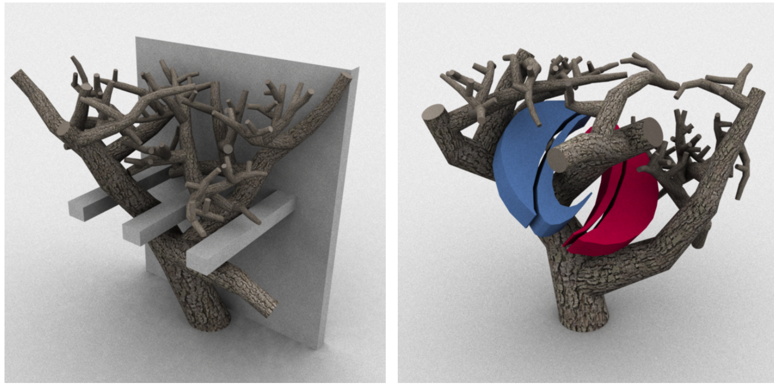
Kulkarni, Kohli, Tenenbaum, Mansinghka "Picture: a probabilistic programming language for scene perception." CVPR (2015).

Directed Procedural Graphics

Stable Static Structures



Procedural Graphics

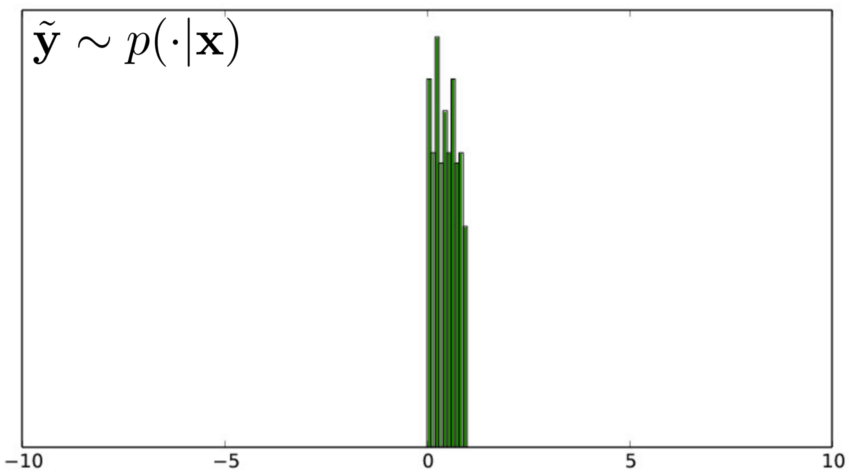


$\mathbf{x}$	$\mathbf{y}$
simulation	constraint

Ritchie, Lin, Goodman, & Hanrahan.
Generating Design Suggestions under Tight Constraints with Gradient-based Probabilistic Programming.
In Computer Graphics Forum, (2015)

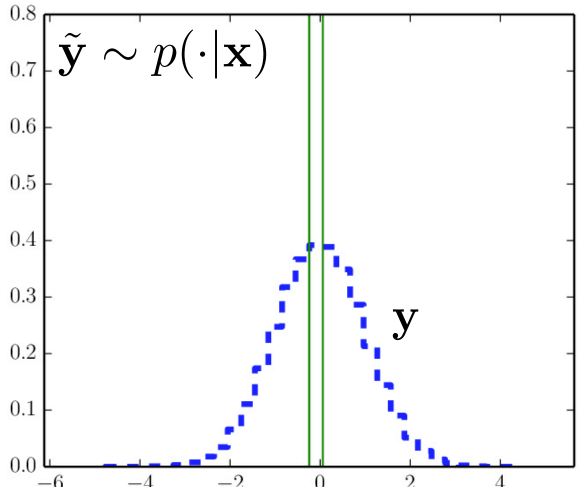
Ritchie, Mildenhall, Goodman, & Hanrahan.
"Controlling Procedural Modeling Programs with Stochastically-Ordered Sequential Monte Carlo."
SIGGRAPH (2015)

Program Induction




```
(lambda (stack-id) (safe-uc (* (if (< 0.0 (* (* (* -1.0 (begin (define
 G__1147 (safe-uc 1.0 1.0)) 0.0)) (* 0.0 (+ 0.0 (safe-uc (* (* (dec -2
.0) (safe-sqrt (begin (define G__1148 3.14159) (safe-log -1.0)))) 2.0)
 0.0)))) 1.0)) (+ (safe-div (begin (define G__1149 (* (+ 3.14159 -1.0)
 1.0)) 1.0) 0.0) (safe-log 1.0)) (safe-log -1.0)) (begin (define G__11
...
```

$\mathbf{x} \sim p(\mathbf{x})$



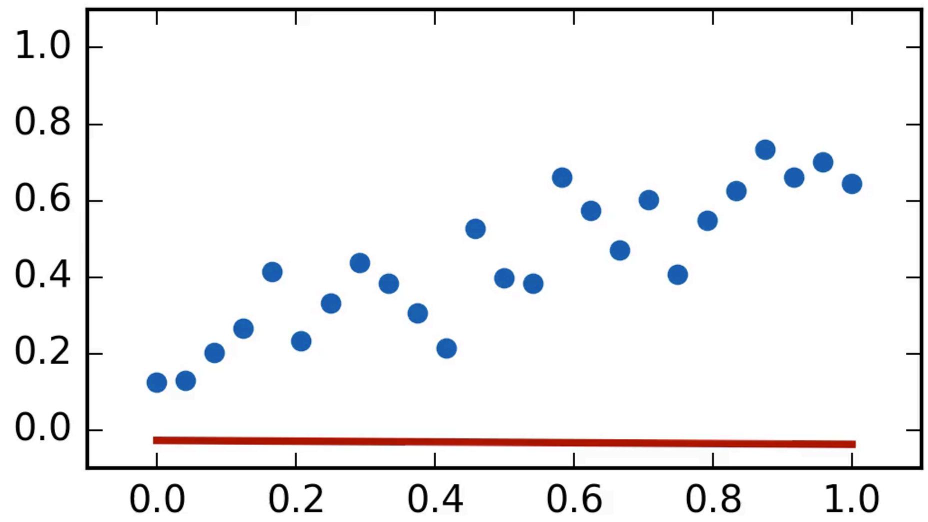
$\mathbf{x} \sim p(\mathbf{x}|\mathbf{y})$

$\mathbf{x}$	$\mathbf{y}$
program source code	program output

Perov and Wood.
"Automatic Sampler Discovery via Probabilistic Programming and Approximate Bayesian Computation"
AGI (2016).

Thinking Generatively about Discriminative Tasks

```
(defquery lin-reg [x-vals y-vals]
 (let [m (sample (normal 0 1))
       c  (sample (normal 0 1))
       f (fn [x] (+ (* m x) c))]
  (map (fn [x y]
          (observe
            (normal (f x) 0.1) y))
        x-vals y-vals))
  [m c])
```




```
(doquery :ipmcmc lin-reg data options)
```

([0.58 -0.05] [0.49 0.1] [0.55 0.05] [0.53 0.04] ….

(Re-?) Introduction to Bayesian Inference

A simple continuous example

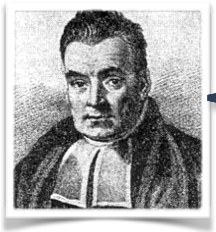
- Measure the temperature of some water using an inexact thermometer
- The actual water temperature x is somewhere near room temperature of 22°; we record an estimate y.

$$x \sim \mathrm{Normal}(22, 10)$$
$$y|x \sim \mathrm{Normal}(x, 1)$$

Easy question: what is $p(y \mid x = 25)$?

Hard question: what is $p(x \mid y = 25)$?

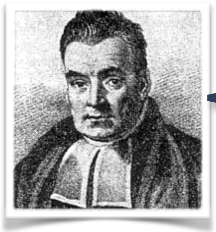
General problem:



$$\underbrace{p(x \mid y)}_{\text{Posterior}} = \underbrace{p(y \mid x)}_{\text{Likelihood}} \underbrace{p(x)}_{\text{Prior}} / p(y)$$

- Our *data* is given by *y*
- Our generative model specifies the prior and likelihood
- We are interested in answering questions about the *posterior* distribution of *p(x | y)*

General problem:



$$p(x \mid y) = p(y \mid x)p(x)/p(y)$$

Posterior Likelihood Prior

- Typically we are not trying to compute a probability density function for *p(x | y)* as our end goal

- Instead, we want to compute *expected values* of some function *f(x)* under the posterior distribution

Expectation

- Discrete and continuous:

$$\mathbb{E}[f] = \sum_x p(x) f(x)$$

$$\mathbb{E}[f] = \int p(x) f(x) \, \mathrm{d}x.$$

- Conditional on another random variable:

$$\mathbb{E}_x[f|y] = \sum_x p(x|y) f(x)$$

Key Monte Carlo identity

- We can approximate expectations using *samples* drawn from a distribution *p.* If we want to compute

$$\mathbb{E}[f] = \int p(x) f(x) \, \mathrm{d}x.$$

we can approximate it with a finite set of points sampled from *p(x)* using

$$\mathbb{E}[f] \simeq \frac{1}{N} \sum_{n=1}^{N} f(x_n)$$

which becomes exact as *N* approaches infinity.

How do we draw samples?

- Simple, well-known distributions: samplers exist (for the moment take as given)
- We will look at:

  1. Build samplers for complicated distributions out of samplers for simple distributions compositionally
  2. Rejection sampling
  3. Likelihood weighting
  4. Markov chain Monte Carlo

Ancestral sampling from a model

- In our example with estimating the water temperature, suppose we already know how to sample from a normal distribution.

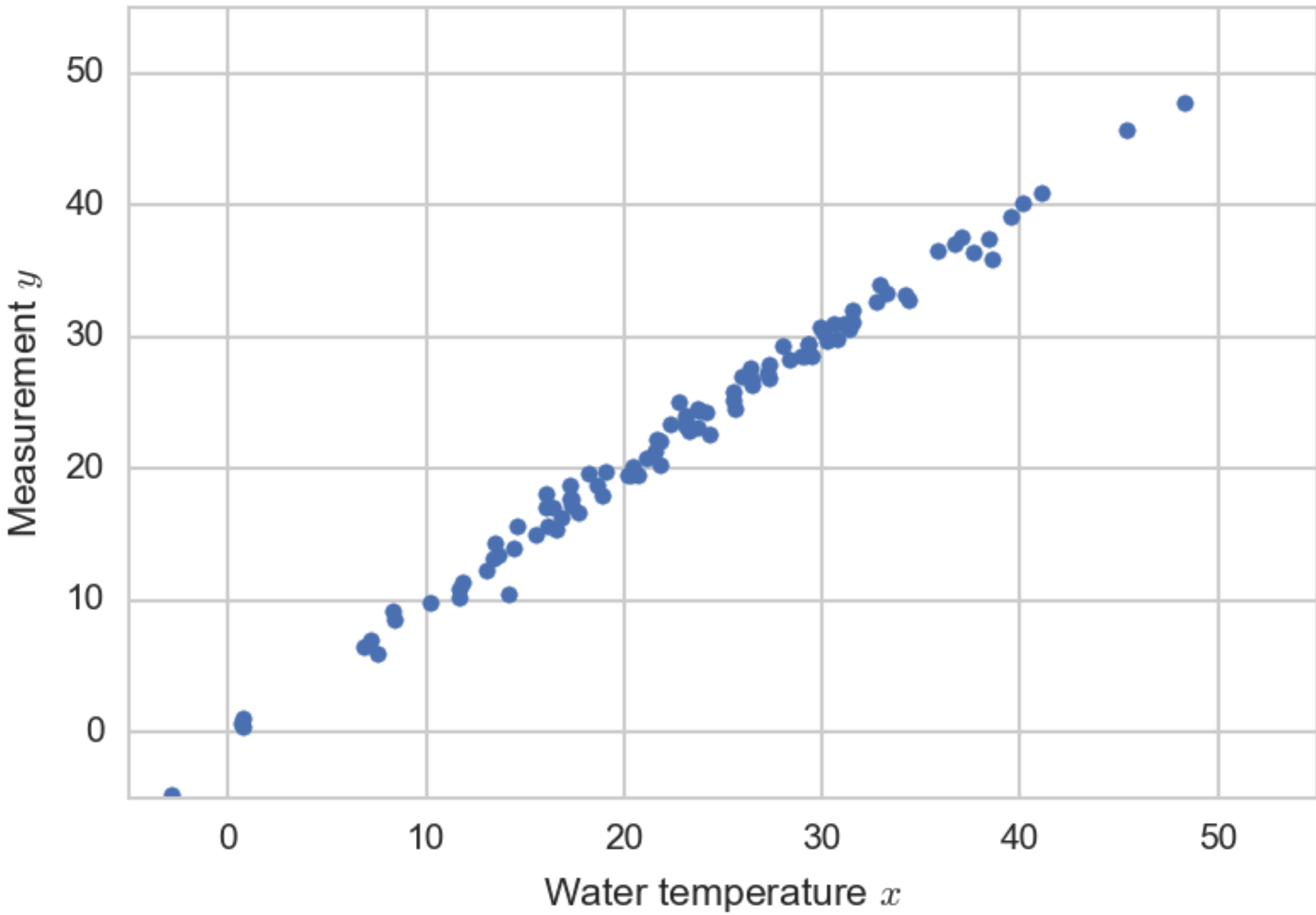
$$x \sim \text{Normal}(22, 10)$$

$$y|x \sim \text{Normal}(x, 1)$$

  We can sample *y* by literally simulating from the generative process: we first sample a "true" temperature *x*, and then we sample the observed *y*.

- This draws a sample from the **joint** distribution *p(x, y)*.

Samples from the joint distribution



Conditioning via rejection

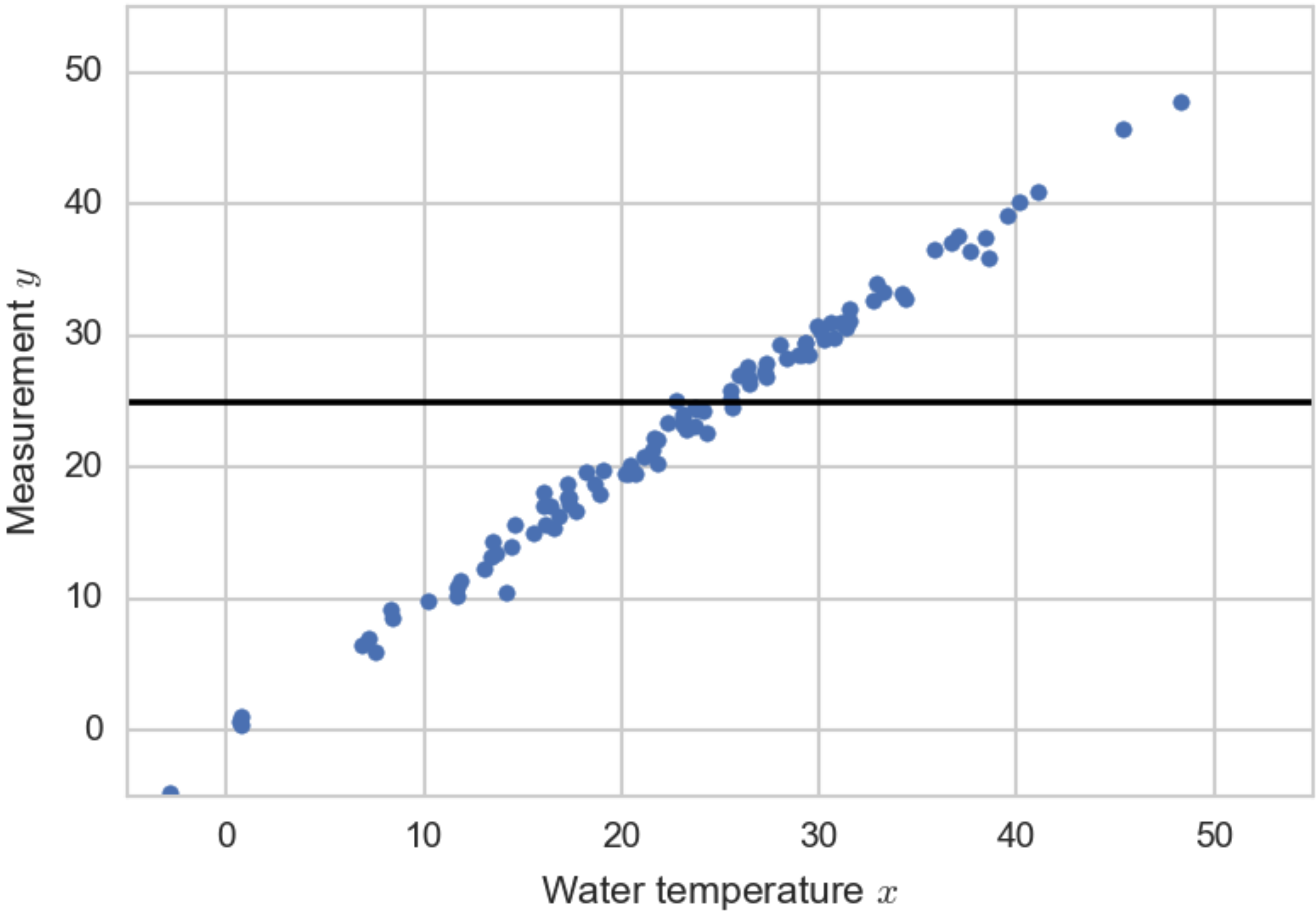
- What if we want to sample from a conditional distribution? The simplest form is via rejection.

- Use the ancestral sampling procedure to simulate from the generative process, draw a sample of x and a sample of y. These are drawn together from the joint distribution $p(x, y)$.

- To estimate the posterior $p(x \mid y = 25)$, we say that x is a sample from the posterior if its corresponding value $y = 25$.

- **Question:** is this a good idea?

Conditioning via rejection



Black bar shows measurement at *y = 25*.

How many of these samples from the joint have *y = 25*?

Conditioning via importance sampling

- One option is to sidestep sampling from the posterior *p(x | y = 3)* entirely, and draw from some proposal distribution *q(x)* instead.

- Instead of computing an expectation with respect to *p(x|y)*, we compute an expectation with respect to *q(x):*

$$\begin{aligned}
\mathbb{E}_{p(x|y)}[f(x)] &= \int f(x)p(x|y)\mathrm{d}x \\
&= \int f(x)p(x|y)\frac{q(x)}{q(x)}\mathrm{d}x \\
&= \mathbb{E}_{q(x)}\left[f(x)\frac{p(x|y)}{q(x)}\right]
\end{aligned}$$

Conditioning via importance sampling

- Define an "importance weight" $W(x) = \frac{p(x|y)}{q(x)}$
- Then, with $x_i \sim q(x)$

$$\mathbb{E}_{p(x|y)}[f(x)] = \mathbb{E}_{q(x)}\left[f(x)W(x)\right] \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)W(x_i)$$

- Expectations now computed using *weighted* samples from *q(x)*, instead of unweighted samples from *p(x|y)*

Conditioning via importance sampling

- Typically, can only evaluate *W(x)* up to a constant (but this is not a problem):

$$W(x_i) = \frac{p(x_i|y)}{q(x_i)} \qquad\qquad w(x_i) = \frac{p(x_i, y)}{q(x_i)}$$

- Approximation:

$$W(x_i) \approx \frac{w(x_i)}{\sum_{j=1}^{N} w(x_j)}$$

$$\mathbb{E}_{p(x|y)}[f(x)] \approx \sum_{i=1}^{N} \frac{w(x_i)}{\sum_{j=1}^{N} w(x_j)} f(x_i)$$

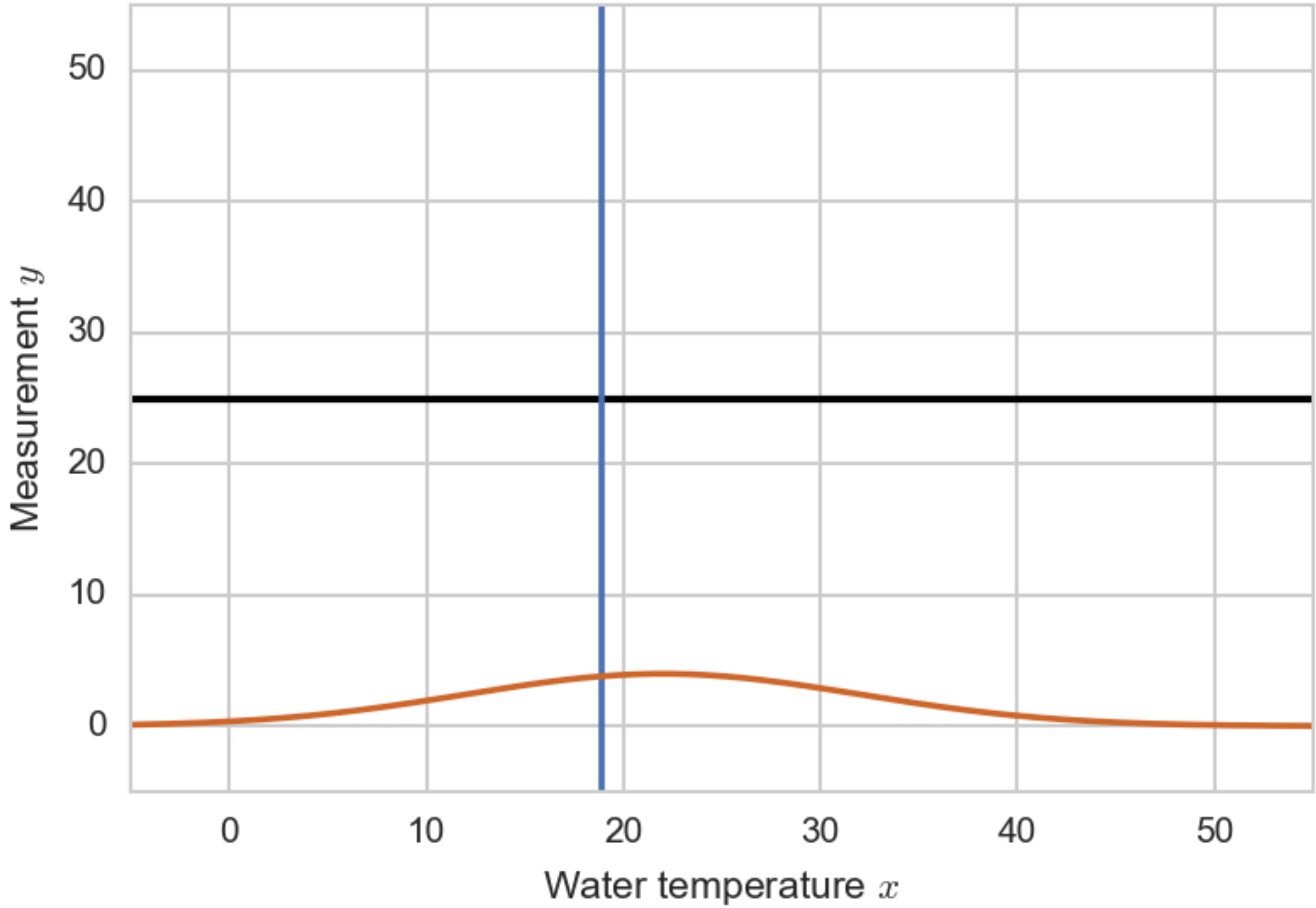
Conditioning via importance sampling

- We already have very simple proposal distribution we know how to sample from: the prior $p(x)$.

- The algorithm then resembles the rejection sampling algorithm, except instead of sampling both the latent variables and the observed variables, we only sample the latent variables

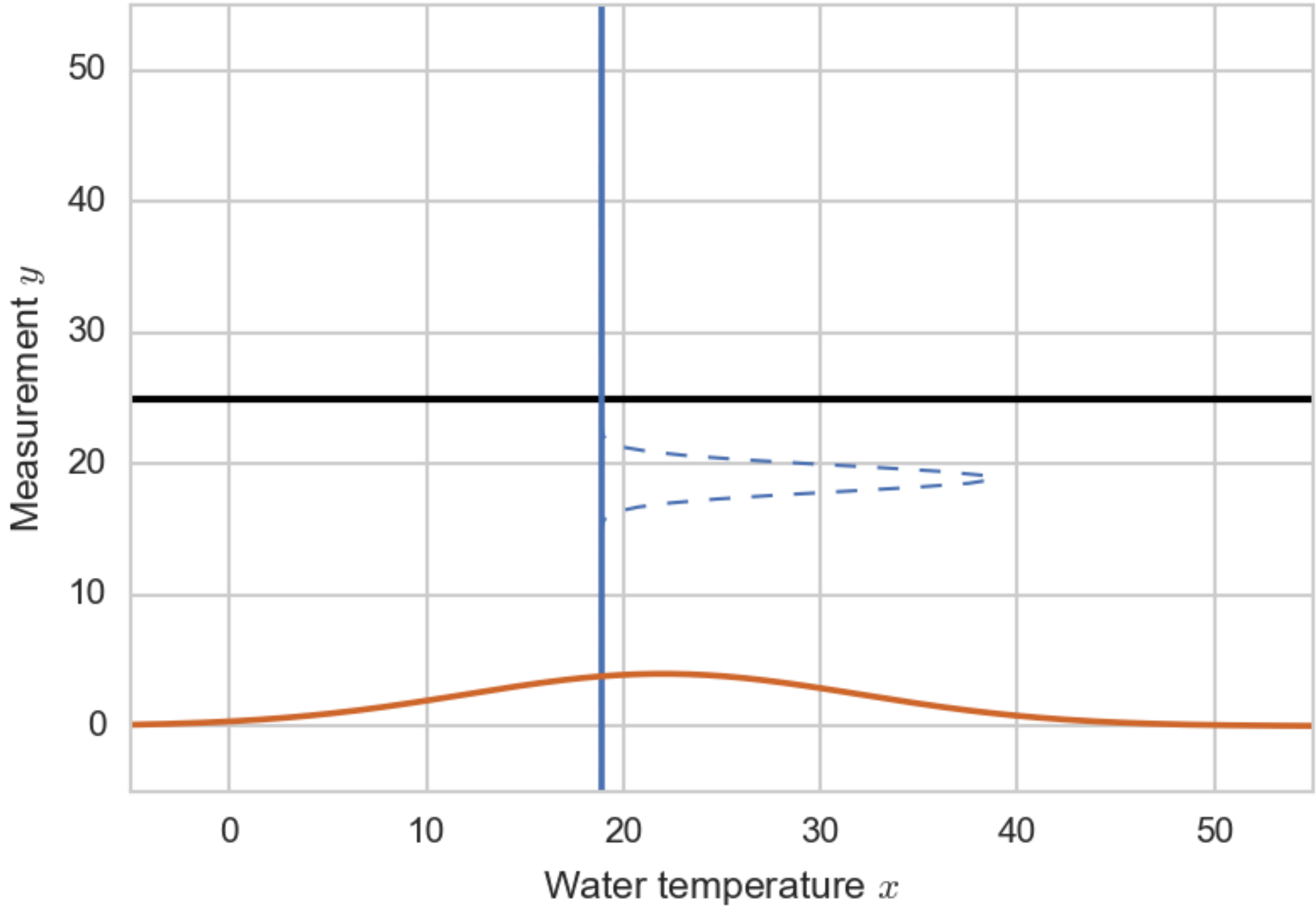
- Then, instead of a "hard" rejection step, we use the values of the latent variables and the data to assign "soft" weights to the sampled values.

Likelihood weighting schematic



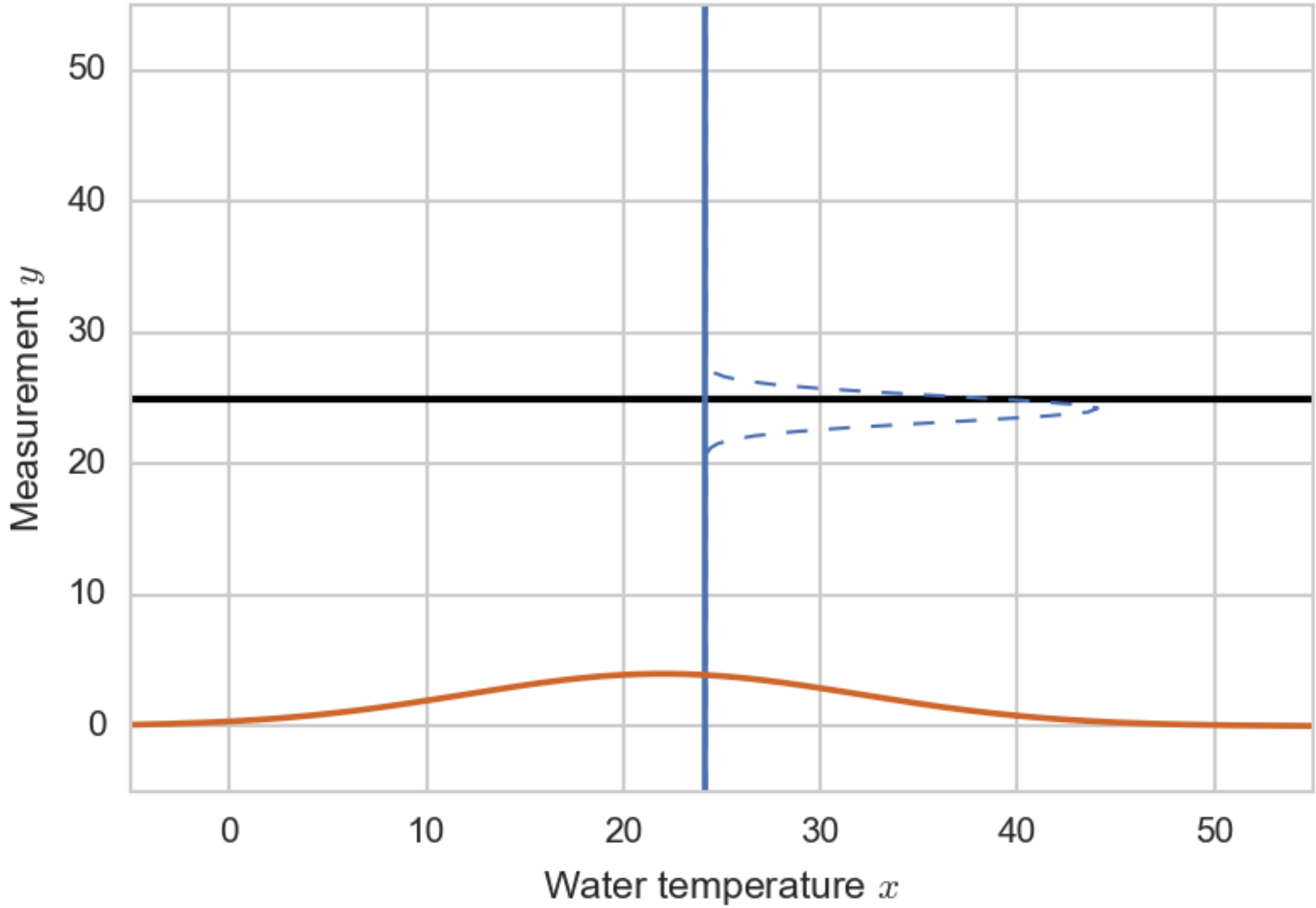
Draw a sample of *x* from the prior

Likelihood weighting schematic



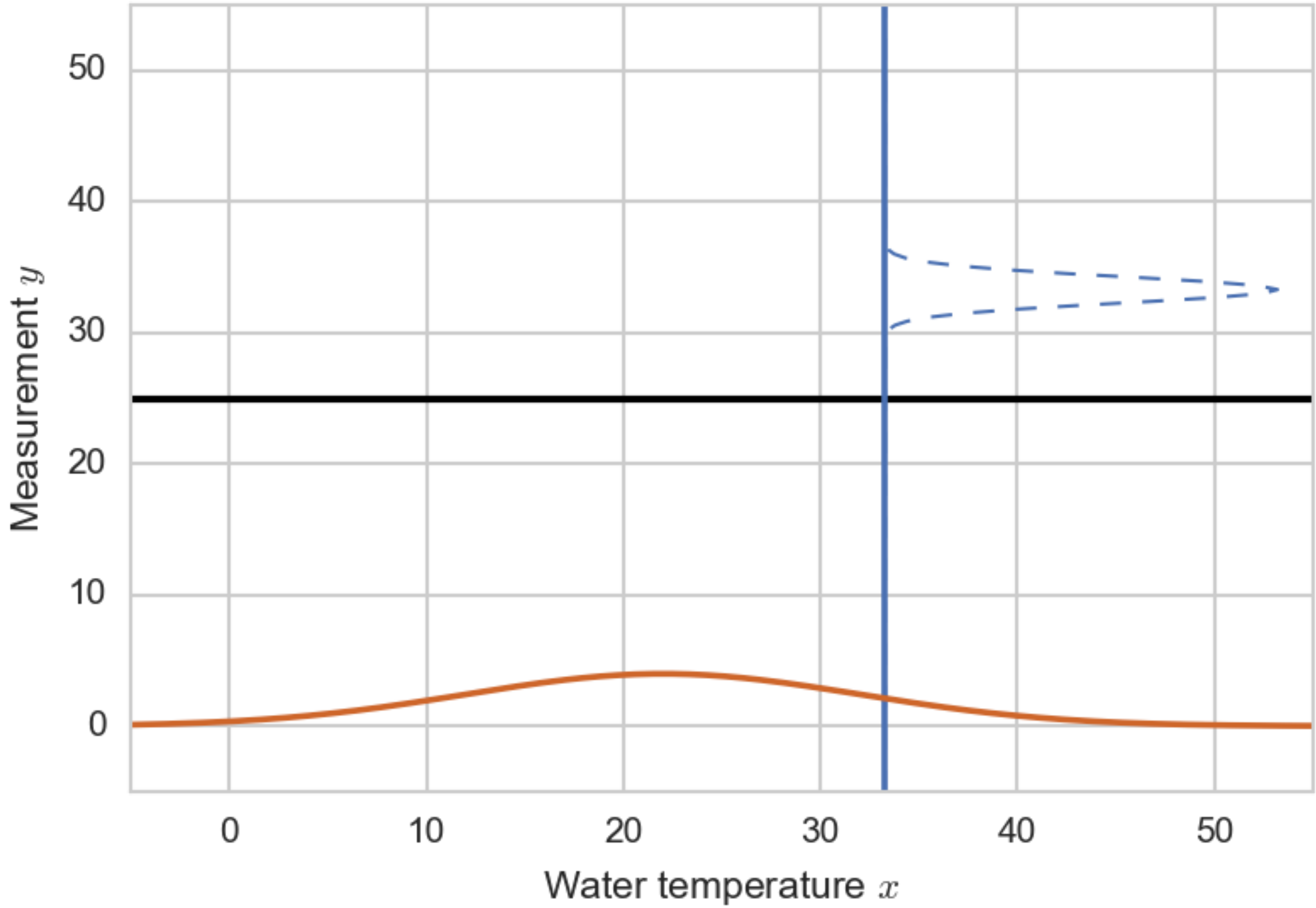
What does $p(y|x)$ look like for this sampled x?

Likelihood weighting schematic



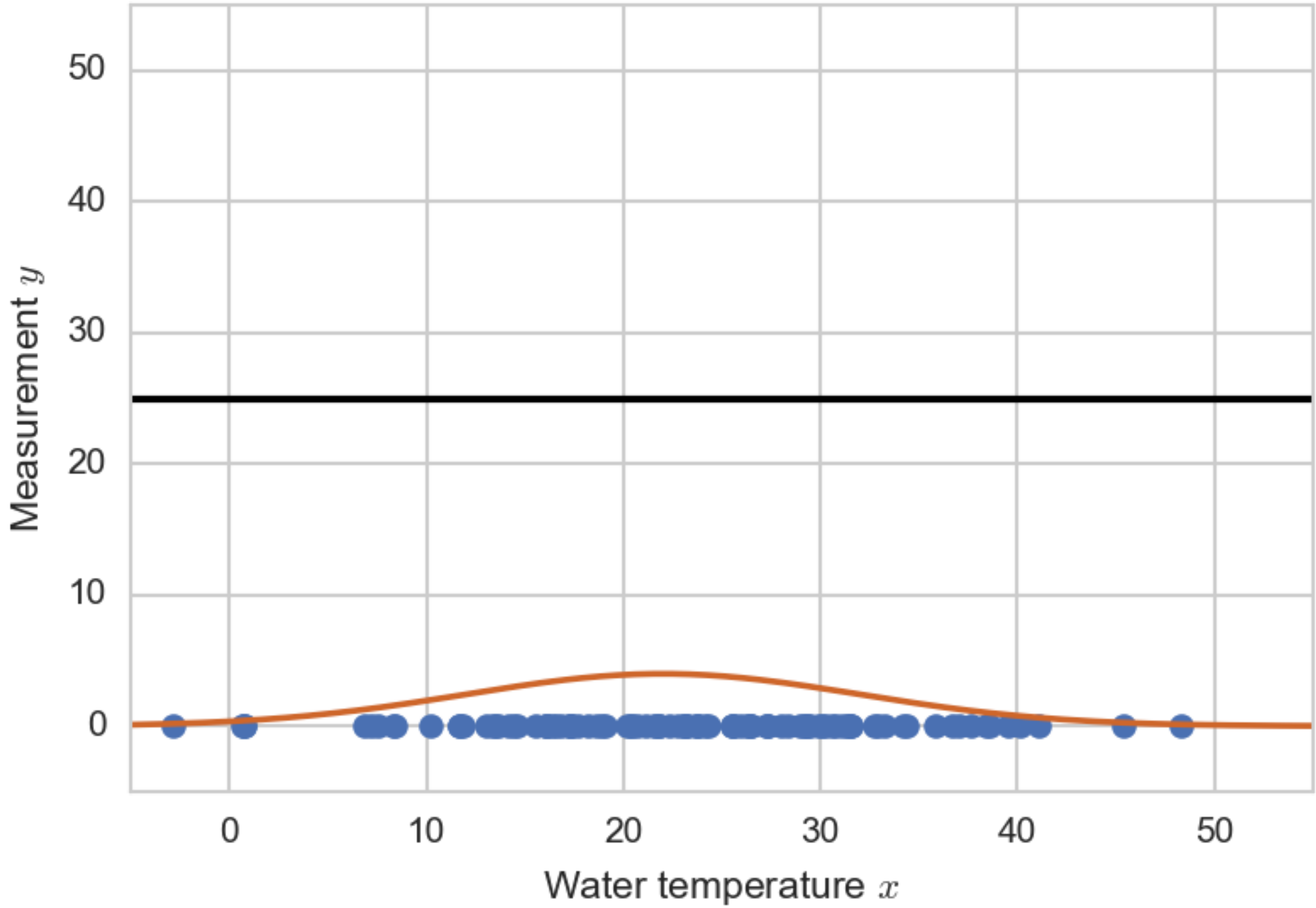
What does $p(y|x)$ look like for this sampled x?

Likelihood weighting schematic



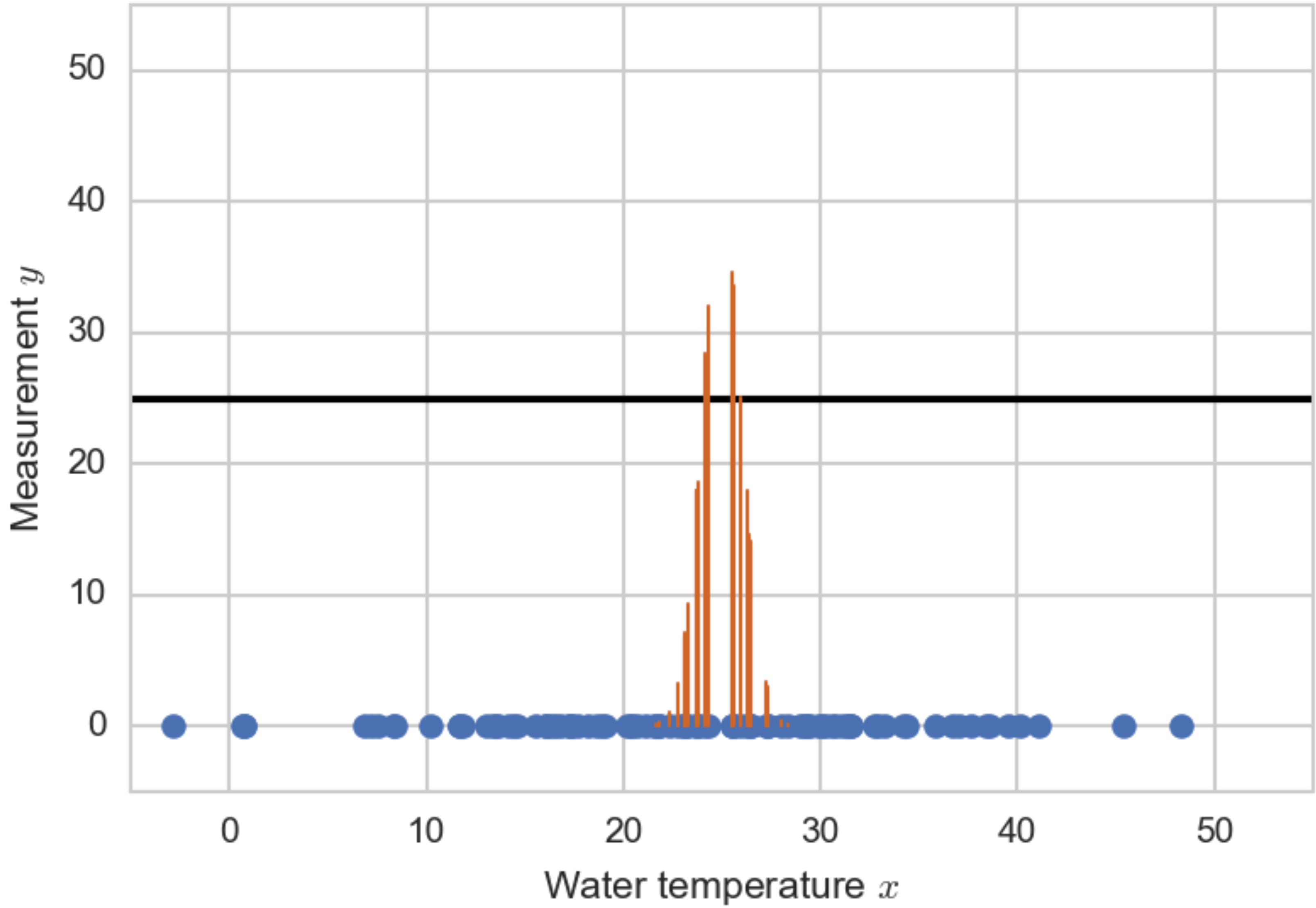
What does $p(y|x)$ look like for this sampled x?

Likelihood weighting schematic



Compute *p(y|x)* for *all* of our *x* drawn from the prior

Likelihood weighting schematic



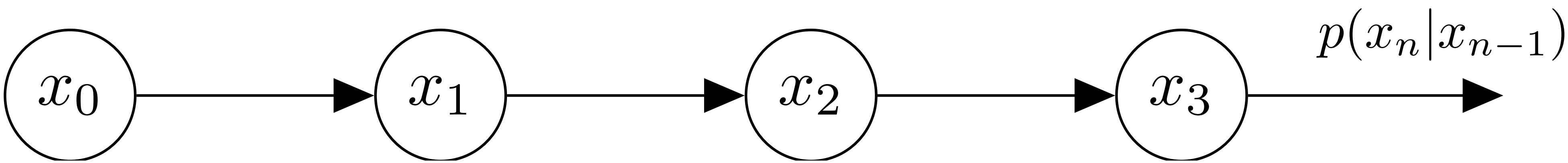
Assign weights (vertical bars) to samples
for a representation of the posterior

Conditioning via MCMC

- **Problem**: Likelihood weighting degrades poorly as the dimension of the latent variables increases, unless we have a very well-chosen proposal distribution *q(x)*.

- **An alternative**: Markov chain Monte Carlo (MCMC) methods draw samples from a target distribution by performing a biased random walk over the space of the latent variables *x*.

- Idea: create a Markov chain such that the sequence of states $x_0, x_1, x_2, \ldots$ are samples from $p(x \mid y)$



Conditioning via MCMC

- MCMC also uses a proposal distribution, but this proposal distribution makes **local** changes to the latent variables *x.* The proposal $q(x' \mid x)$ defines a conditional distribution over x' given a current value x.

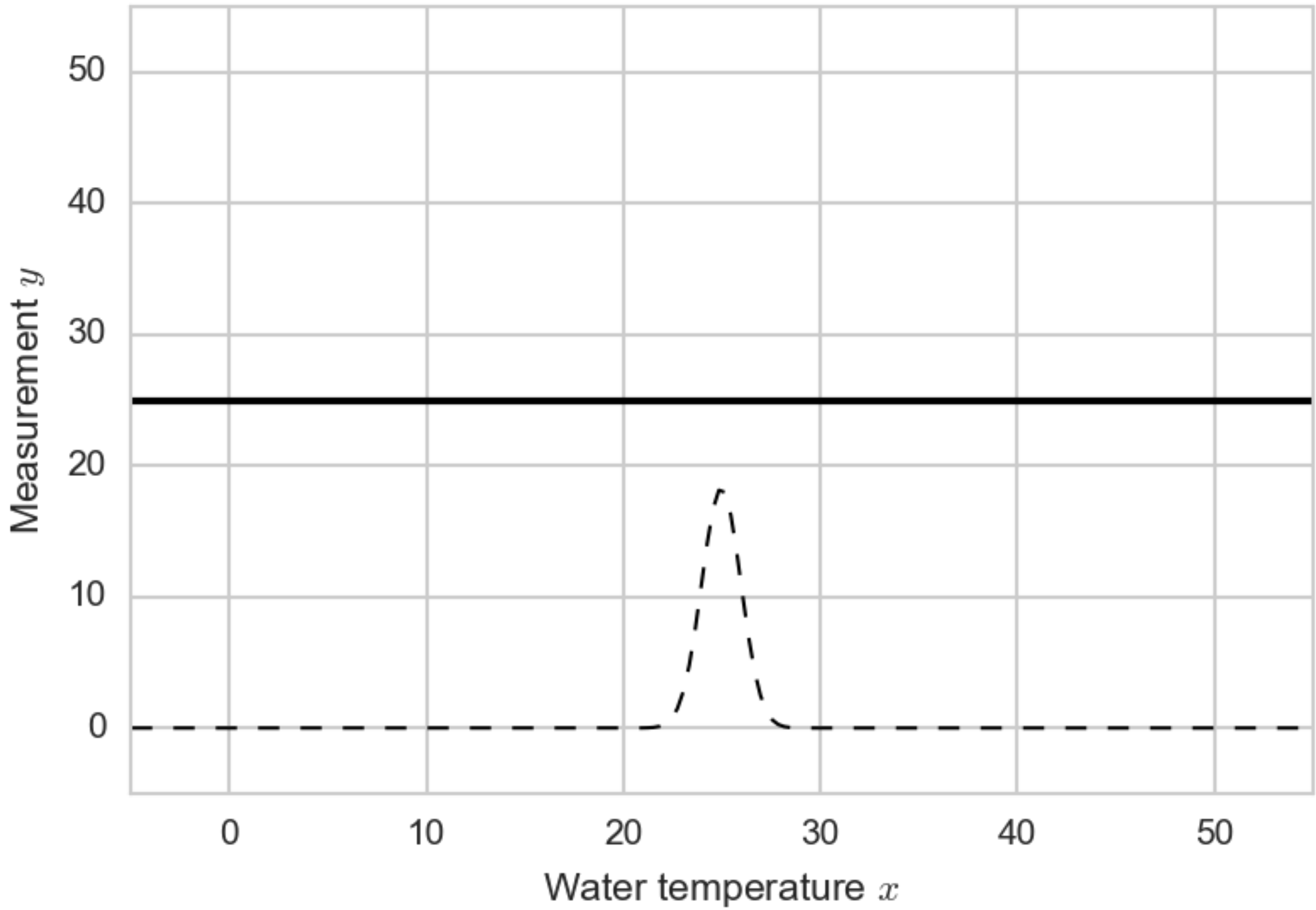
  - Typical choice: add small amount of Gaussian noise

- We use the proposal and the joint density to define an "acceptance ratio"

$$A(x \to x') = \min\left(1,\ \frac{p(x', y)q(x|x')}{p(x, y)q(x'|x)}\right)$$

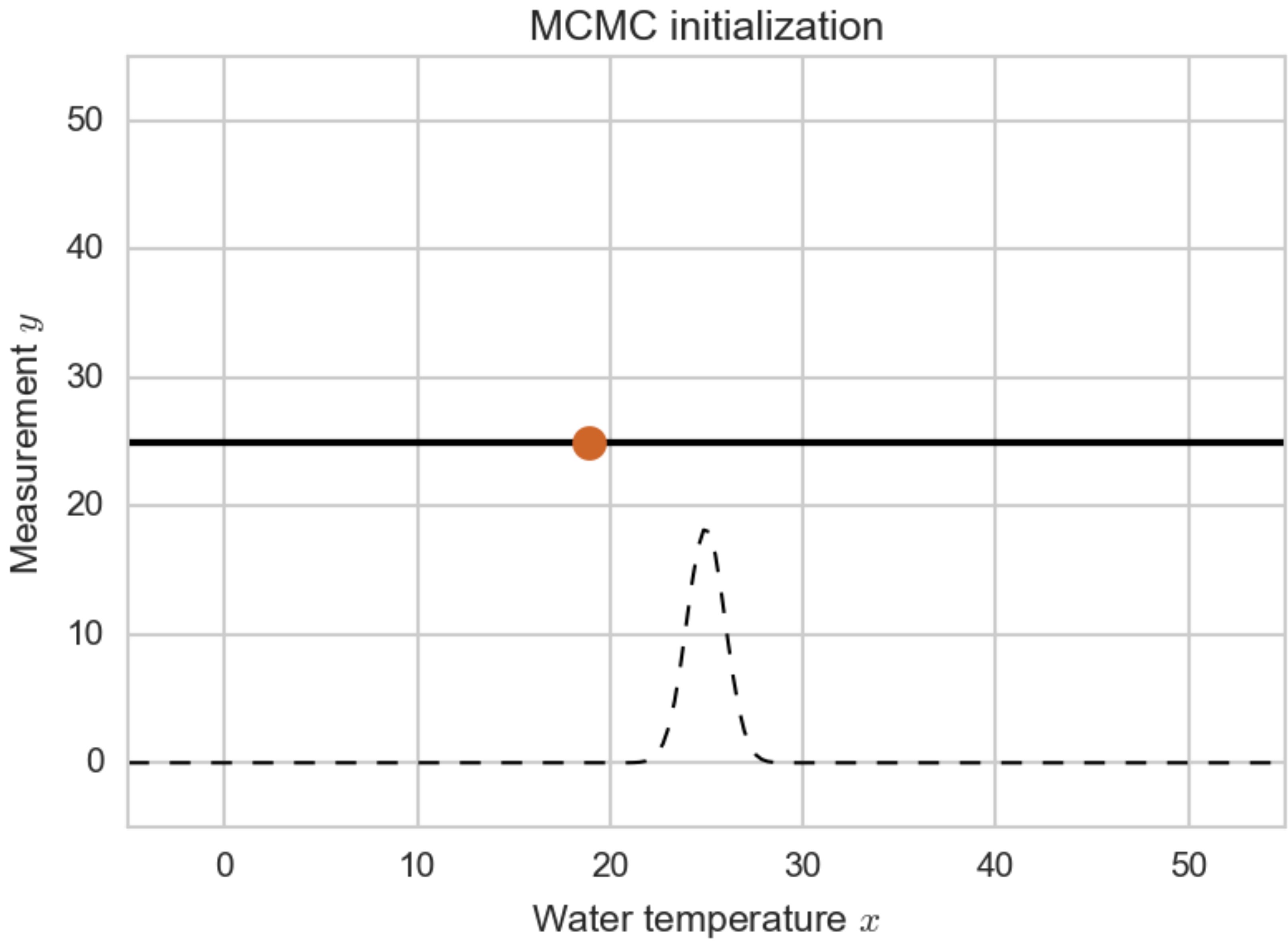
- Metropolis-Hastings: with probability A we "move" state with the new value x', otherwise we stay at x.

MCMC schematic



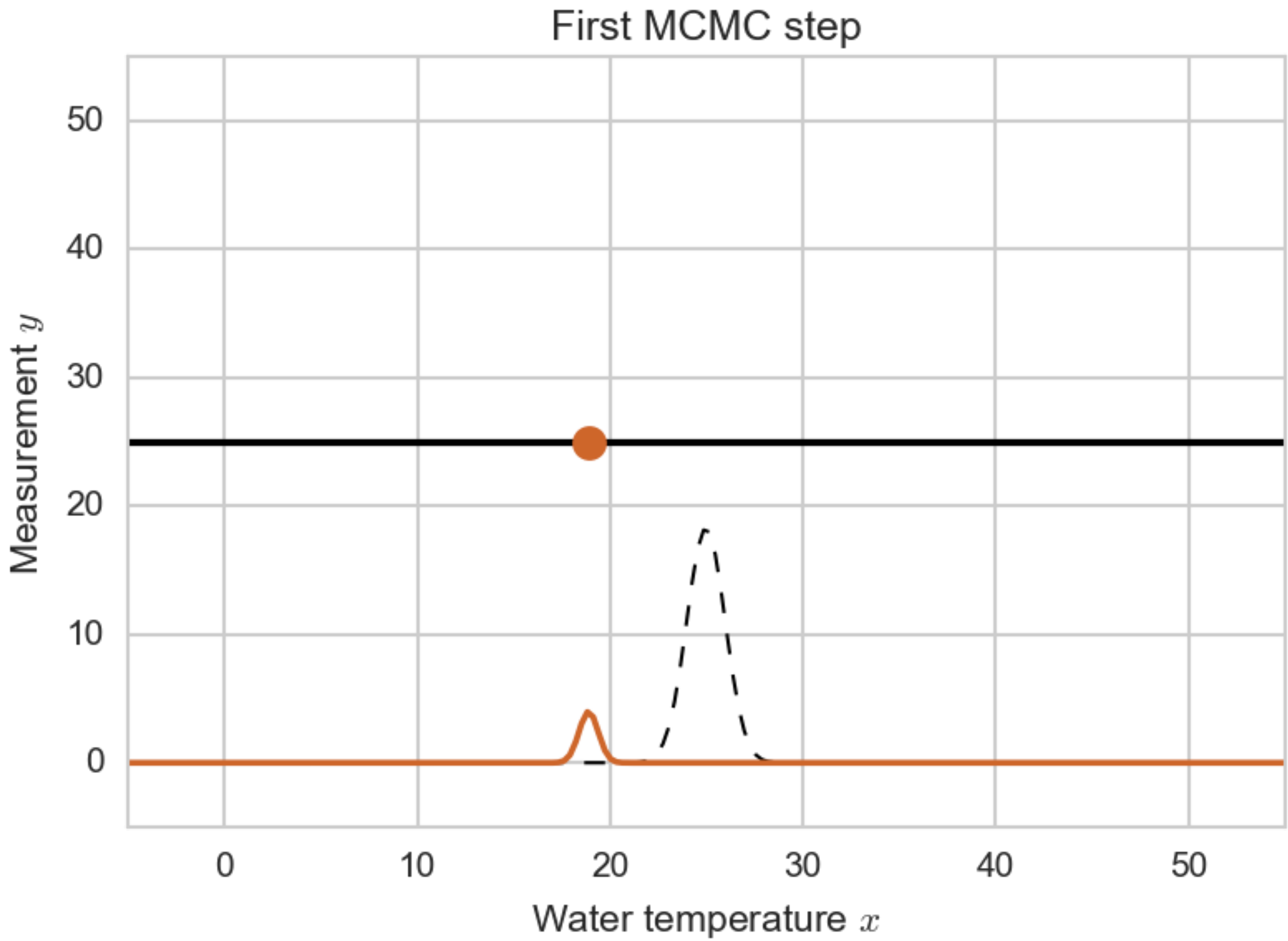
The (unnormalized) joint distribution *p(x,y)* is shown as a dashed line

MCMC schematic



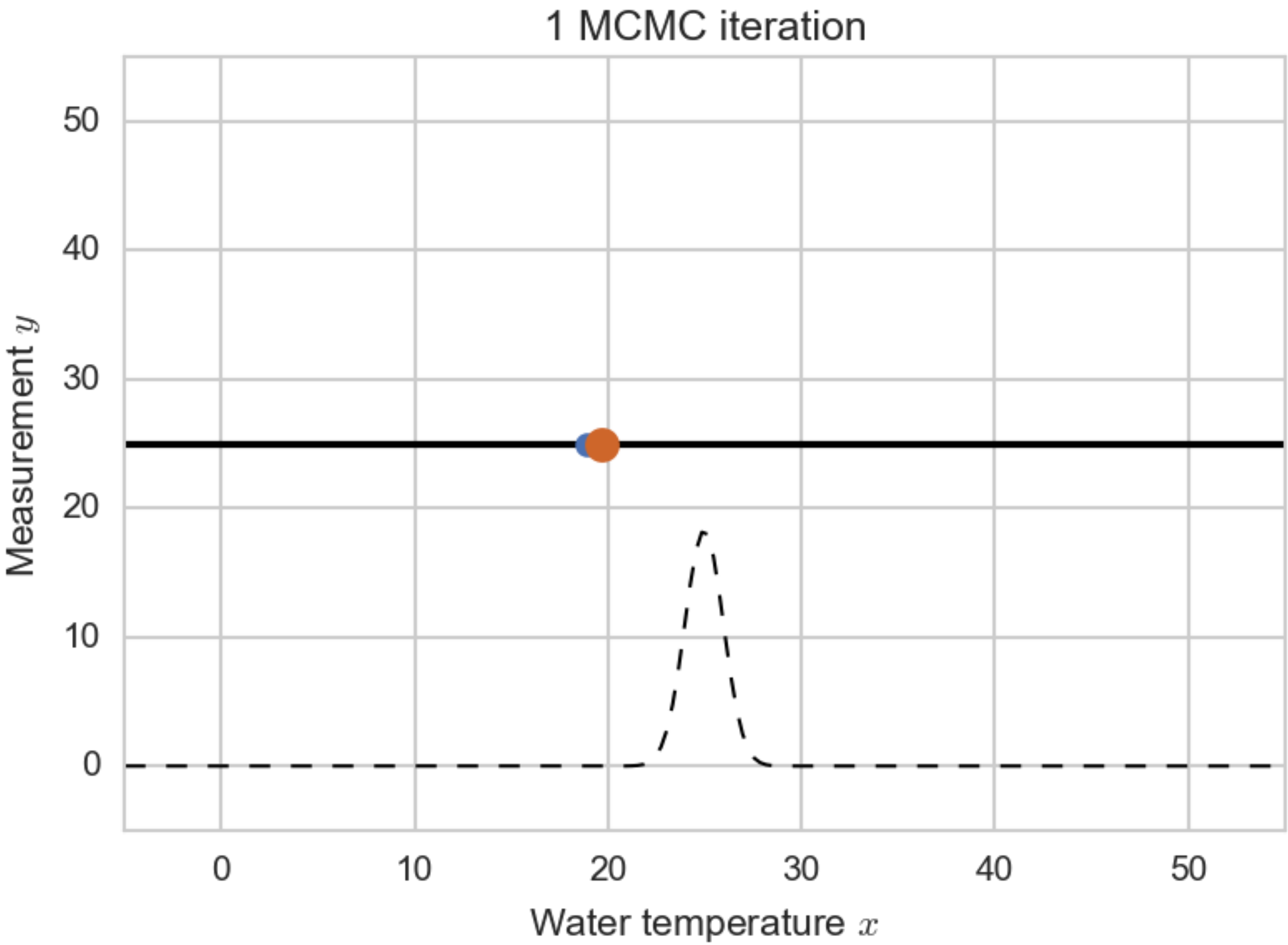
Initialize arbitrarily (e.g. with a sample from the prior)

MCMC schematic



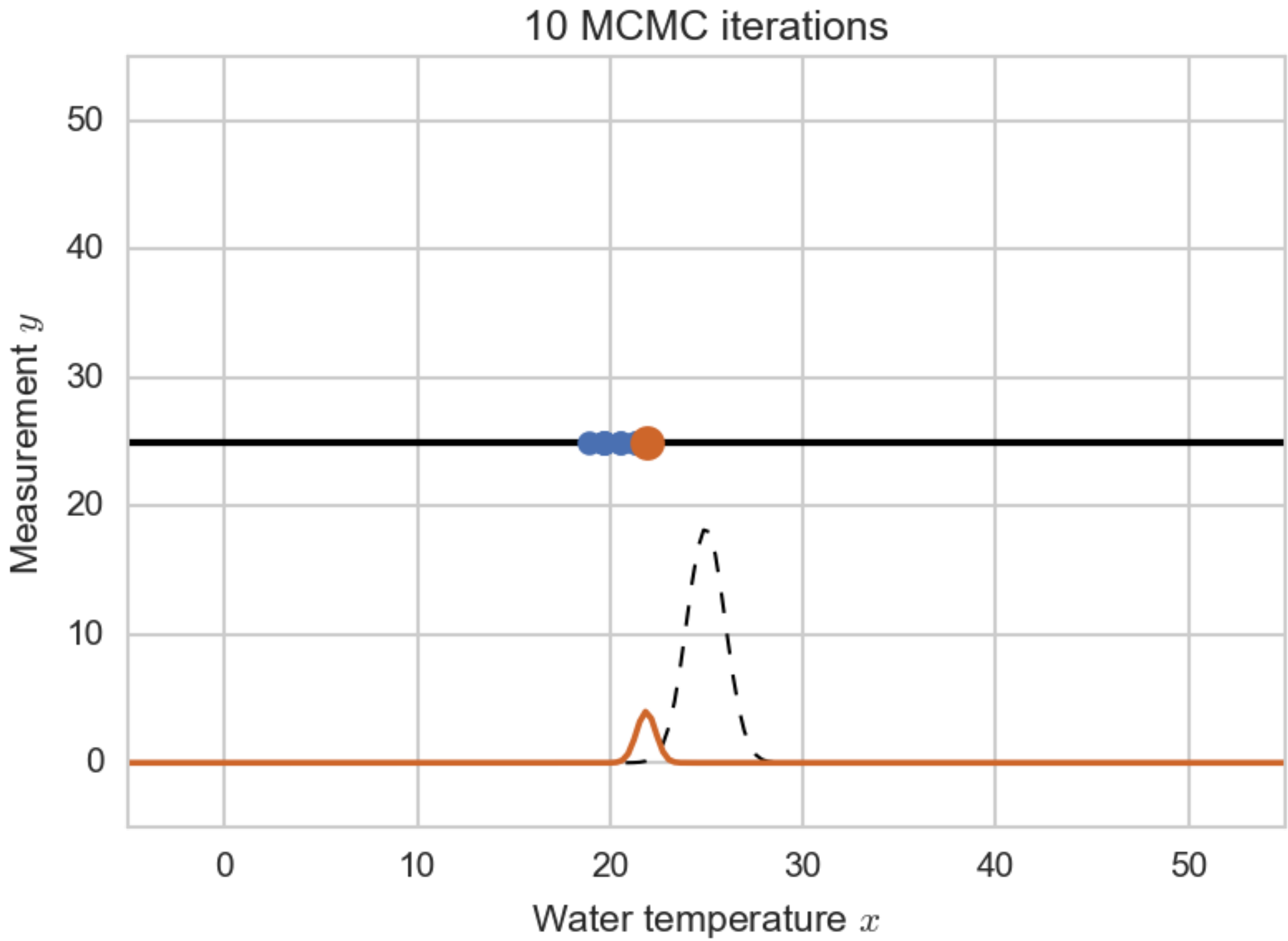
Propose a local move on *x* from a transition distribution

MCMC schematic



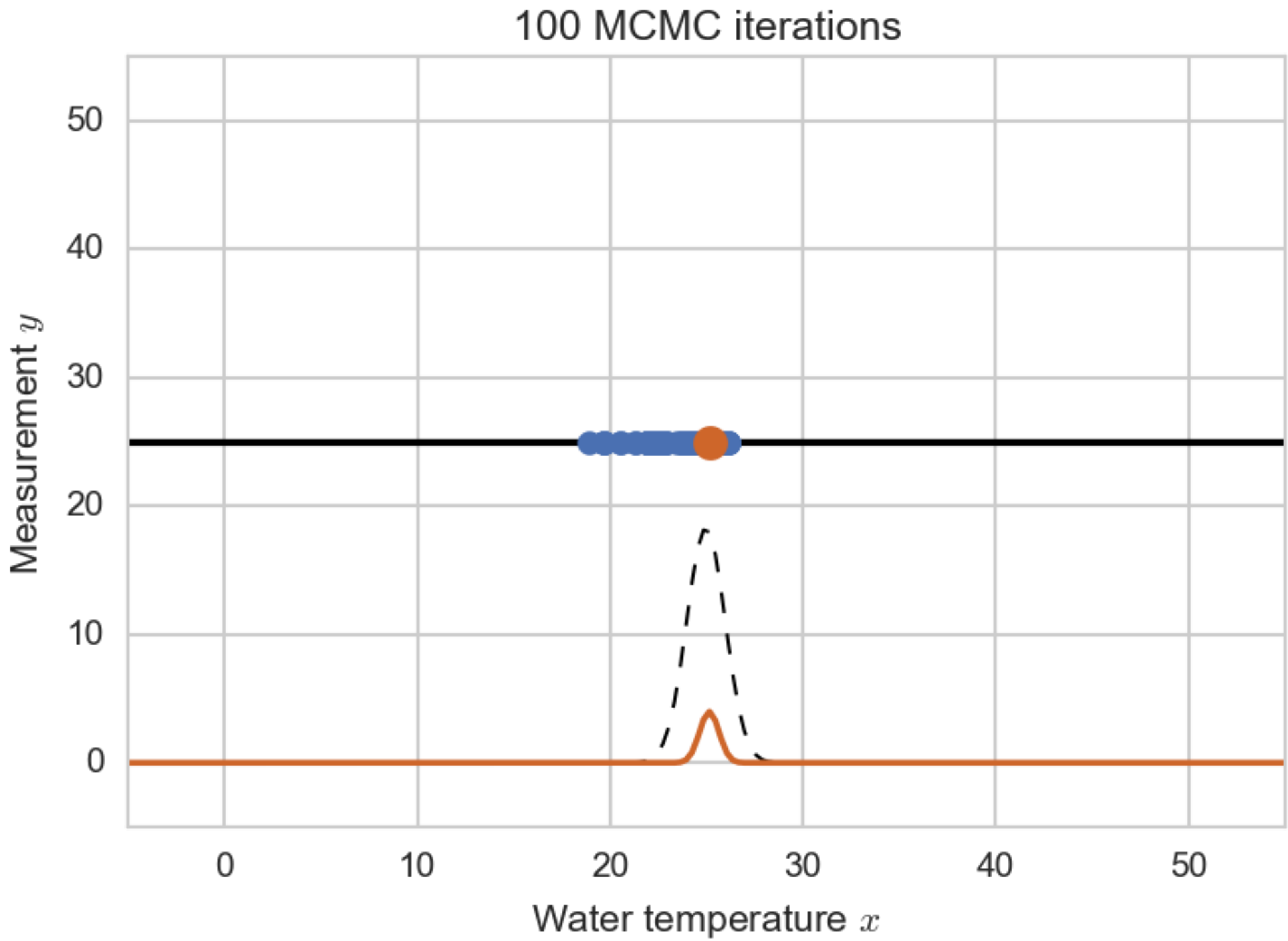
Here, we proposed a point in a region of higher probability density, and accepted

MCMC schematic



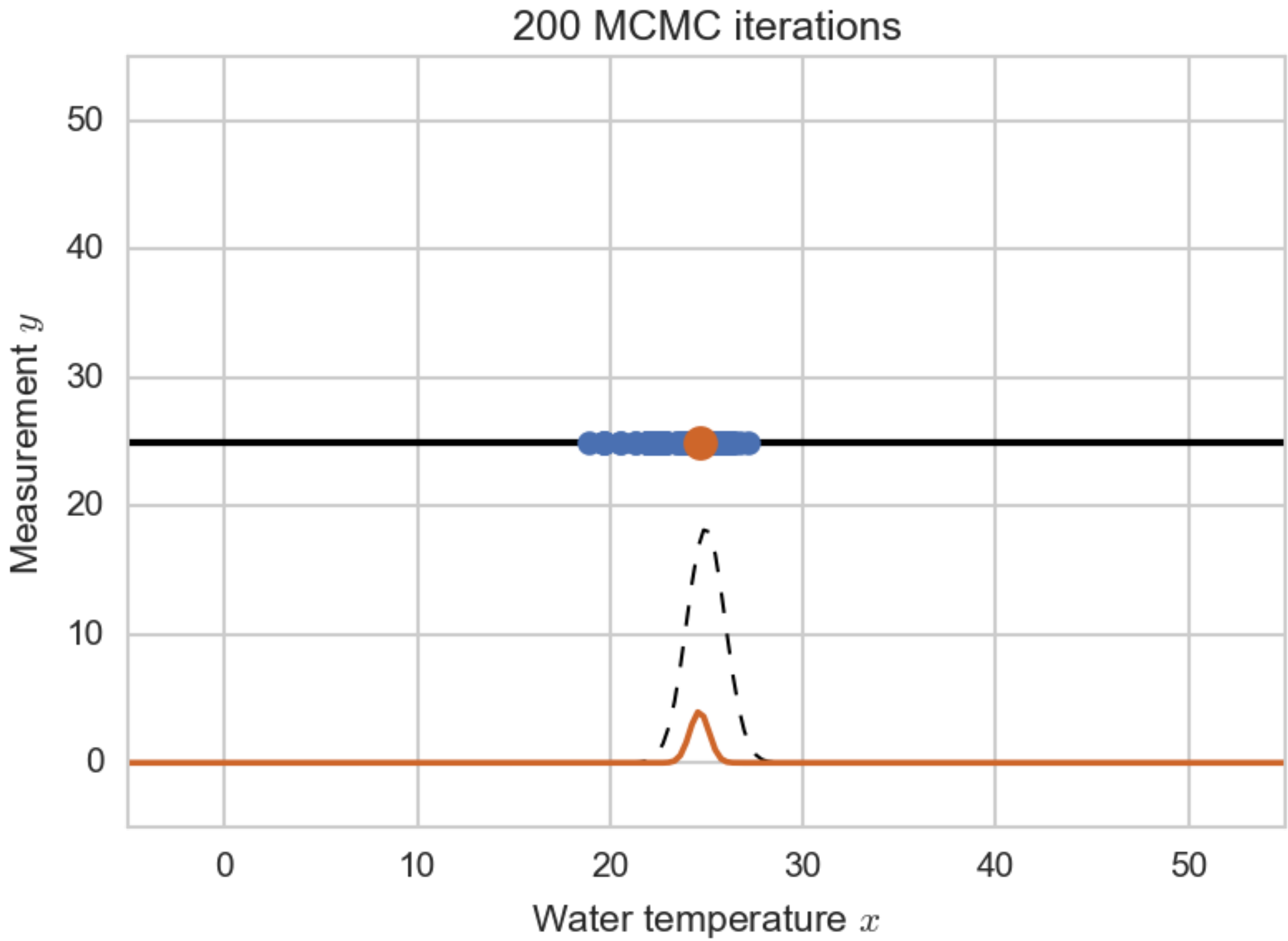
Continue: propose a local move, and accept or reject.
At first, this will look like a stochastic search algorithm!

MCMC schematic



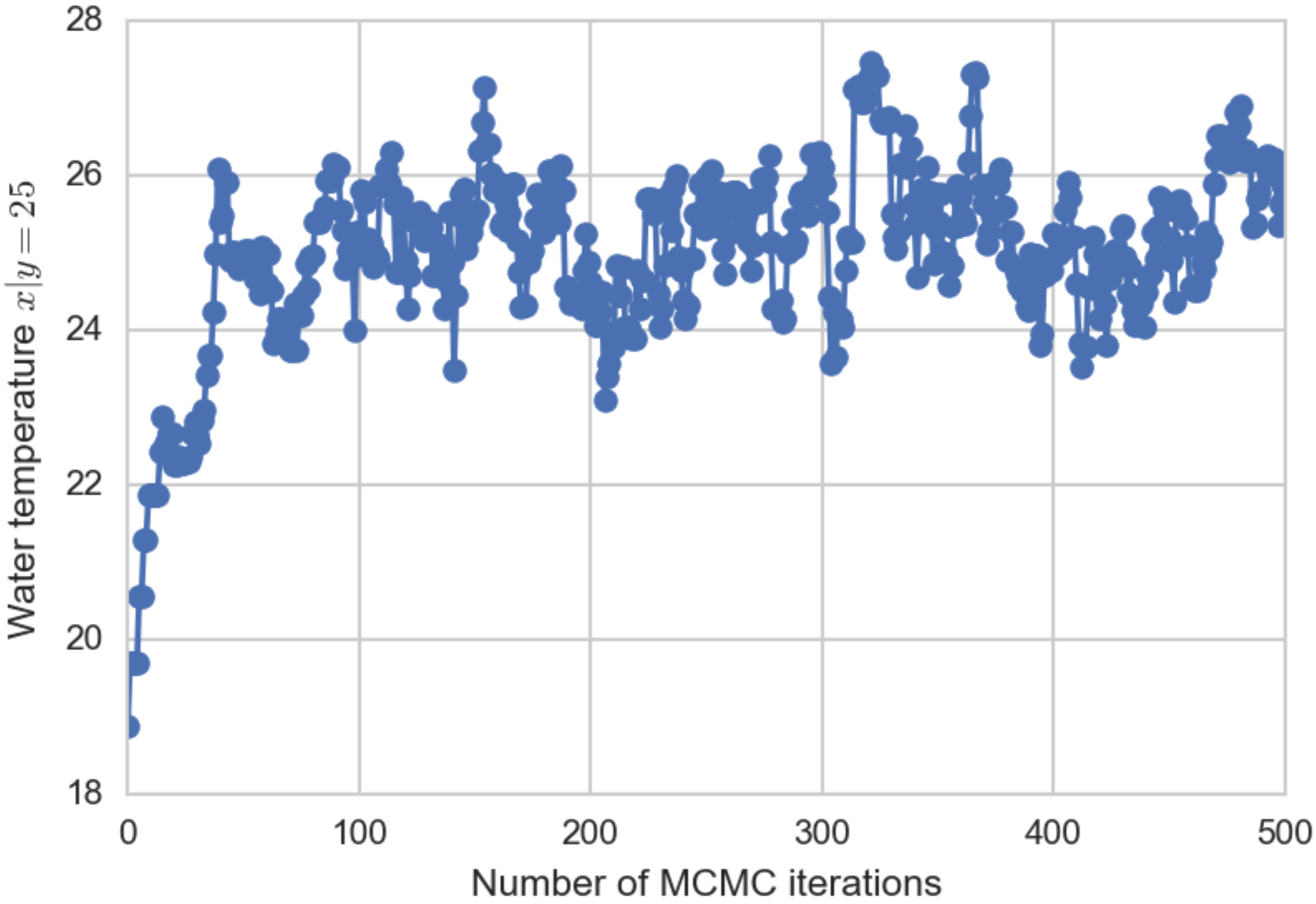
Once in a high-density region, it will explore the space

MCMC schematic



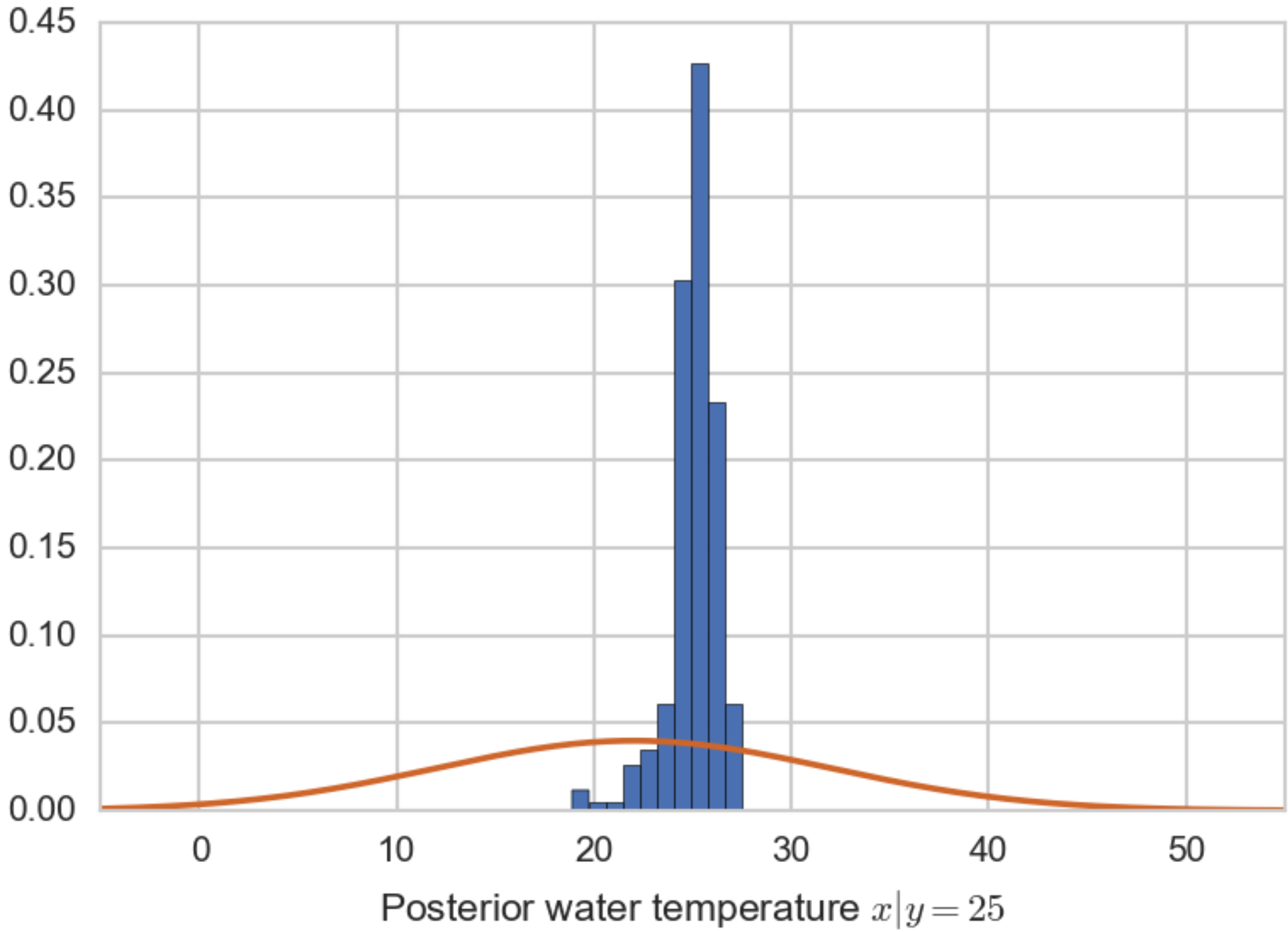
Once in a high-density region, it will explore the space

MCMC schematic



Helpful diagnostic: a “trace plot” of the path of the sampled values, as the number of MCMC iterations increases

MCMC schematic



Histogram of trace plot, overlaid on prior probability density

How It Works:
PPL **Inference**

Start With A Program

```
(let [z (sample (bernoulli 0.5))
      mu (if (= z 0) -1.0 1.0)
      d (normal mu 1.0)
      y 0.5]
  (observe d y)
  z)
```

Program

Semantically Agreed Mathematical Object

```
(let [z (sample (bernoulli 0.5))
      mu (if (= z 0) -1.0 1.0)
      d (normal mu 1.0)
      y 0.5]
  (observe d y)
  z)
```

Program

$$
\begin{aligned}
V &= \{z, y\},\\
A &= \{(z, y)\},\\
\mathcal{P} &= [z \mapsto (p_{\text{bern}}\ z\ 0.5),\\
&\qquad y \mapsto (p_{\text{norm}}\ y\ (\texttt{if}\ (=\ z\ 0)\ \texttt{-1.0}\ \texttt{1.0})\ \texttt{1.0})]\\
\mathcal{Y} &= [y \mapsto 0.5]\\
E &= z
\end{aligned}
$$

Mathematic Object

Rules of Inference

```
(let [z (sample (bernoulli 0.5))
      mu (if (= z 0) -1.0 1.0)
      d (normal mu 1.0)
      y 0.5]
  (observe d y)
  z)
```

$$
\begin{aligned}
V &= \{z, y\},\\
A &= \{(z, y)\},\\
\mathcal{P} &= [z \mapsto (p_{\text{bern}}\ z\ \texttt{0.5}),\\
&\qquad y \mapsto (p_{\text{norm}}\ y\ \texttt{(if (= } z \texttt{ 0) -1.0 1.0) 1.0)}]\\
\mathcal{Y} &= [y \mapsto 0.5]\\
E &= \ z
\end{aligned}
$$

Program **Mathematic Object**

$$
\frac{\begin{array}{ll}
\rho,\ \phi,\ e_1 \Downarrow G_1,\ E_1 & \rho,\ \phi,\ e_2 \Downarrow G_2,\ E_2\\
(V, A, \mathcal{P}, \mathcal{Y}) = G_1 \oplus G_2 & \text{Choose a fresh variable } v\\
F_1 = \textsc{Score}(E_1, v) \neq \bot & F = \texttt{(if } \phi\ F_1\ \texttt{1)}\\
Z = (\textsc{FreeVars}(F_1) \setminus \{v\}) \cap V & \textsc{FreeVars}(E_2) \cap V = \emptyset\\
B = \{(z, v) : z \in Z\} &
\end{array}}{\rho,\ \phi,\ \texttt{(observe } e_1\ e_2\texttt{)} \Downarrow (V \cup \{v\},\ A \cup B,\ P \oplus [v \mapsto F],\ \mathcal{Y} \oplus [v \mapsto E_2]),\ E_2}
$$

Big Step Operational Semantics

Intuitive Evaluation Perspective

```
(defquery example [y]
  (let [x (sample (beta 1 1))]  ; f(x)
    (observe (bernoulli x) y)   ; g(y|x)
    x))
```

- Syntactically denotes joint and conditioning

$$\gamma(\mathbf{x}) \triangleq p(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{N} g_i(y_i|\phi_i) \prod_{j=1}^{M} f_j(x_j|\theta_j)$$

- Evaluator characterizes

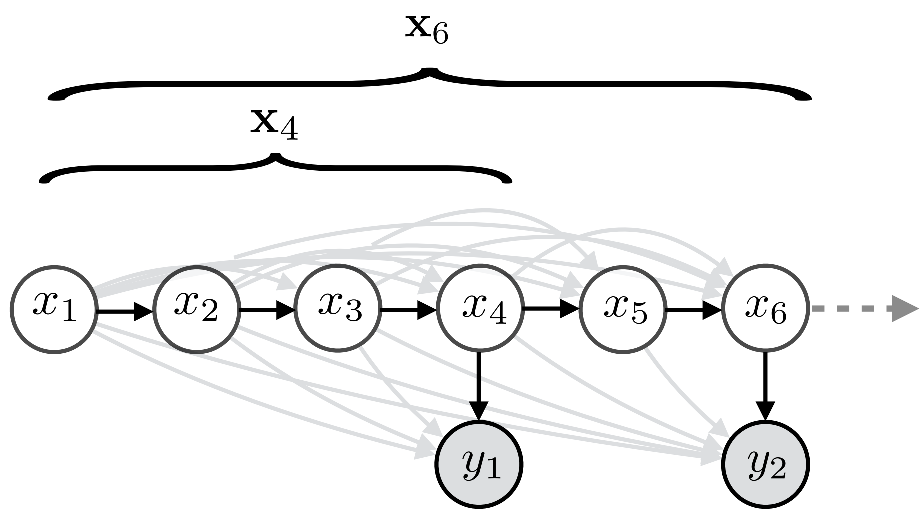
$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{y})}$$

Trace Probability

- Defined as (up to a normalization constant)

$$\gamma(\mathbf{x}) \triangleq p(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{N} g_i(y_i | \phi_i) \prod_{j=1}^{M} f_j(x_j | \theta_j)$$

- Simple notation hides complex dependency structure!



$\mathbf{x}_j = x_1 \times \cdots \times x_j$

etc

$$\gamma(\mathbf{x}) = p(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{N} \tilde{g}_i(\mathbf{x}_{n_i}) \left(y_i \middle| \tilde{\phi}_i(\mathbf{x}_{n_i}) \right) \prod_{j=1}^{M} \tilde{f}_j(\mathbf{x}_{j-1}) \left(x_j \middle| \tilde{\theta}_j(\mathbf{x}_{j-1}) \right)$$

Execution (Trace)-Based Inference

- Sequence of N **observe**'s

$$\{(g_i, \phi_i, y_i)\}_{i=1}^{N}$$

- Sequence of M **sample**'s

$$\{(f_j, \theta_j)\}_{j=1}^{M}$$

- Sequence of M sampled values

$$\{x_j\}_{j=1}^{M}$$

- Conditioned on these sampled values the entire trace is *deterministic*

Three Base Algorithms

- Likelihood Weighting
  - Importance sampling with prior as proposal
- Metropolis Hastings
- Sequential Monte Carlo

Likelihood Weighting

- Run K independent copies of program simulating from the prior

$$q(\mathbf{x}^k) = \prod_{j=1}^{M^k} f_j(x_j^k | \theta_j^k)$$

- Accumulate *unnormalized* weights (likelihoods)

$$w(\mathbf{x}^k) = \frac{\gamma(\mathbf{x}^k)}{q(\mathbf{x}^k)} = \prod_{i=1}^{N^k} g_i^k(y_i^k | \phi_i^k)$$

- Use in approximate (Monte Carlo) integration

$$W^k = \frac{w(\mathbf{x}^k)}{\sum_{\ell=1}^{K} w(\mathbf{x}^\ell)} \qquad \widehat{\mathbb{E}}_\pi[Q(\mathbf{x})] = \sum_{k=1}^{K} W^k Q(\mathbf{x}^k)$$

BLOG default inference engine:
http://bayesianlogic.github.io/pages/users-manual.html

Likelihood Weighting

- Run K independent copies of program simulating from the prior

$$q(\mathbf{x}^k) = \prod_{j=1}^{M^k} f_j(x_j^k | \theta_j^k)$$

- Accumulate *unnormalized* weights (likelihoods)

$$w(\mathbf{x}^k) = \frac{\gamma(\mathbf{x}^k)}{q(\mathbf{x}^k)} = \prod_{i=1}^{N^k} g_i^k(y_i^k | \phi_i^k)$$

- Use in approximate (Monte Carlo) integration

$$W^k = \frac{w(\mathbf{x}^k)}{\sum_{\ell=1}^{K} w(\mathbf{x}^\ell)} \qquad \widehat{\mathbb{E}}_\pi[Q(\mathbf{x})] = \sum_{k=1}^{K} W^k Q(\mathbf{x}^k)$$

BLOG default inference engine:
http://bayesianlogic.github.io/pages/users-manual.html

Likelihood Weighting

- Run K independent copies of program simulating from the prior

$$q(\mathbf{x}^k) = \prod_{j=1}^{M^k} f_j(x_j^k | \theta_j^k)$$

- Accumulate *unnormalized* weights (likelihoods)

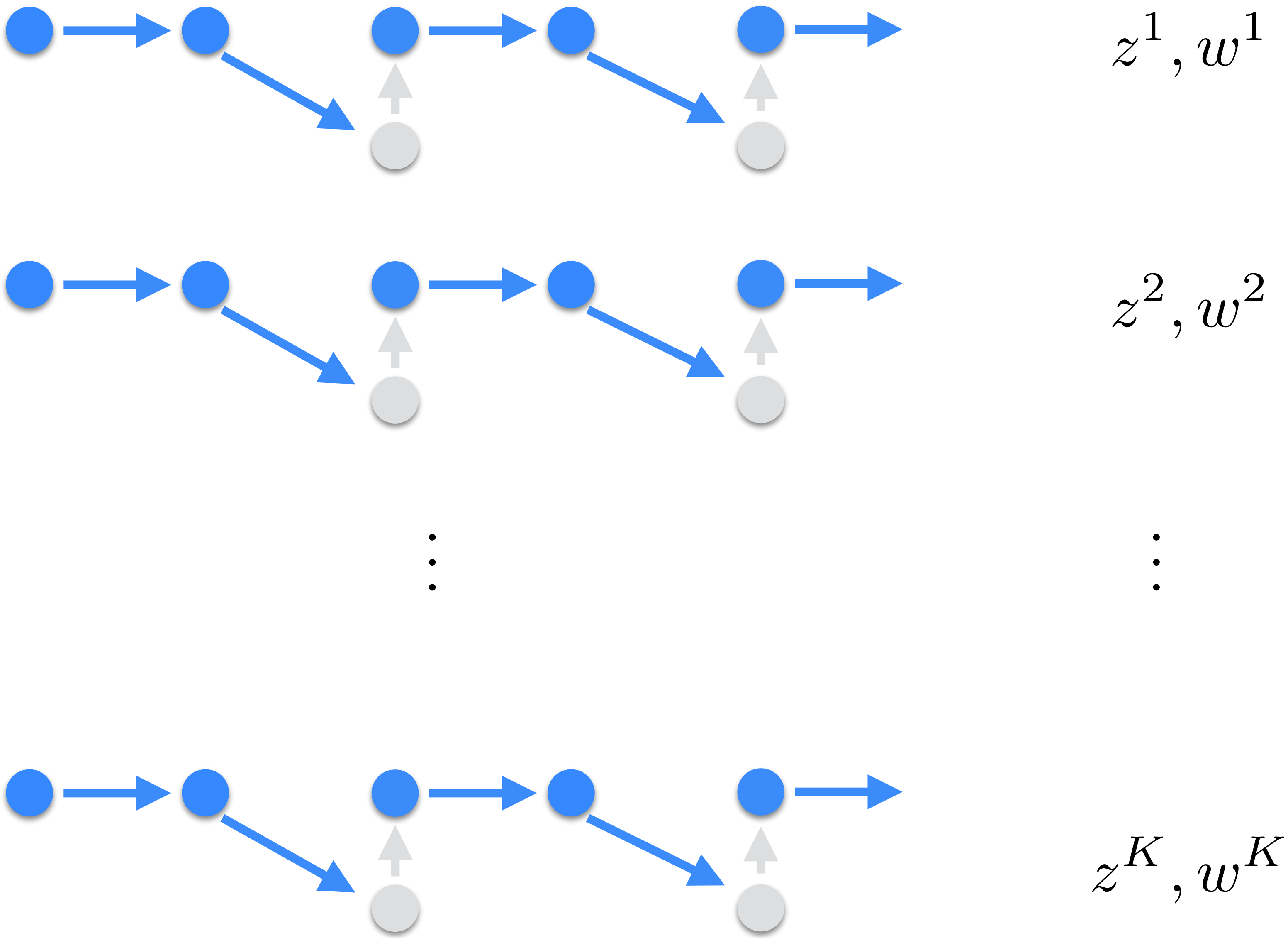
$$w(\mathbf{x}^k) = \frac{\gamma(\mathbf{x}^k)}{q(\mathbf{x}^k)} = \prod_{i=1}^{N^k} g_i^k(y_i^k | \phi_i^k)$$

- Use in approximate (Monte Carlo) integration

$$W^k = \frac{w(\mathbf{x}^k)}{\sum_{\ell=1}^{K} w(\mathbf{x}^\ell)} \qquad \widehat{\mathbb{E}}_\pi[Q(\mathbf{x})] = \sum_{k=1}^{K} W^k Q(\mathbf{x}^k)$$

BLOG default inference engine:
http://bayesianlogic.github.io/pages/users-manual.html

Likelihood Weighting Schematic



Metropolis Hastings = “Single Site” MCMC = LMH

Posterior distribution of execution traces is proportional to trace score with observed values plugged in

$$\gamma(\mathbf{x}) \triangleq p(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{N} g_i(y_i|\phi_i) \prod_{j=1}^{M} f_j(x_j|\theta_j) \qquad \pi(\mathbf{x}) \triangleq p(\mathbf{x}|\mathbf{y}) = \frac{\gamma(\mathbf{x})}{Z}$$

Metropolis-Hastings acceptance rule

$$\alpha = \min\left(1, \frac{\pi(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{\pi(\mathbf{x})q(\mathbf{x}'|\mathbf{x})}\right)$$

Need proposal

Goodman, Mansinghka, Roy, Bonawitz, and Tenenbaum “Church: a language for generative models.” UAI 2008.
Wingate, Stuhlmüller, Goodman “Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation” AISTATS 2011

LMH Proposal

Probability of new part of
proposed execution trace

$$q(\mathbf{x}'|\mathbf{x}^s) = \frac{1}{M^s}\kappa(x'_\ell|x^s_\ell)\prod_{j=\ell+1}^{M'} f'_j(x'_j|\theta'_j)$$

Number of samples in
original trace

LMH Acceptance Ratio

"Single site update" = sample from the prior = run program forward

$$\kappa(x'_m|x_m) = f_m(x'_m|\theta_m), \theta_m = \theta'_m$$

MH acceptance ratio

Number of sample statements in original trace

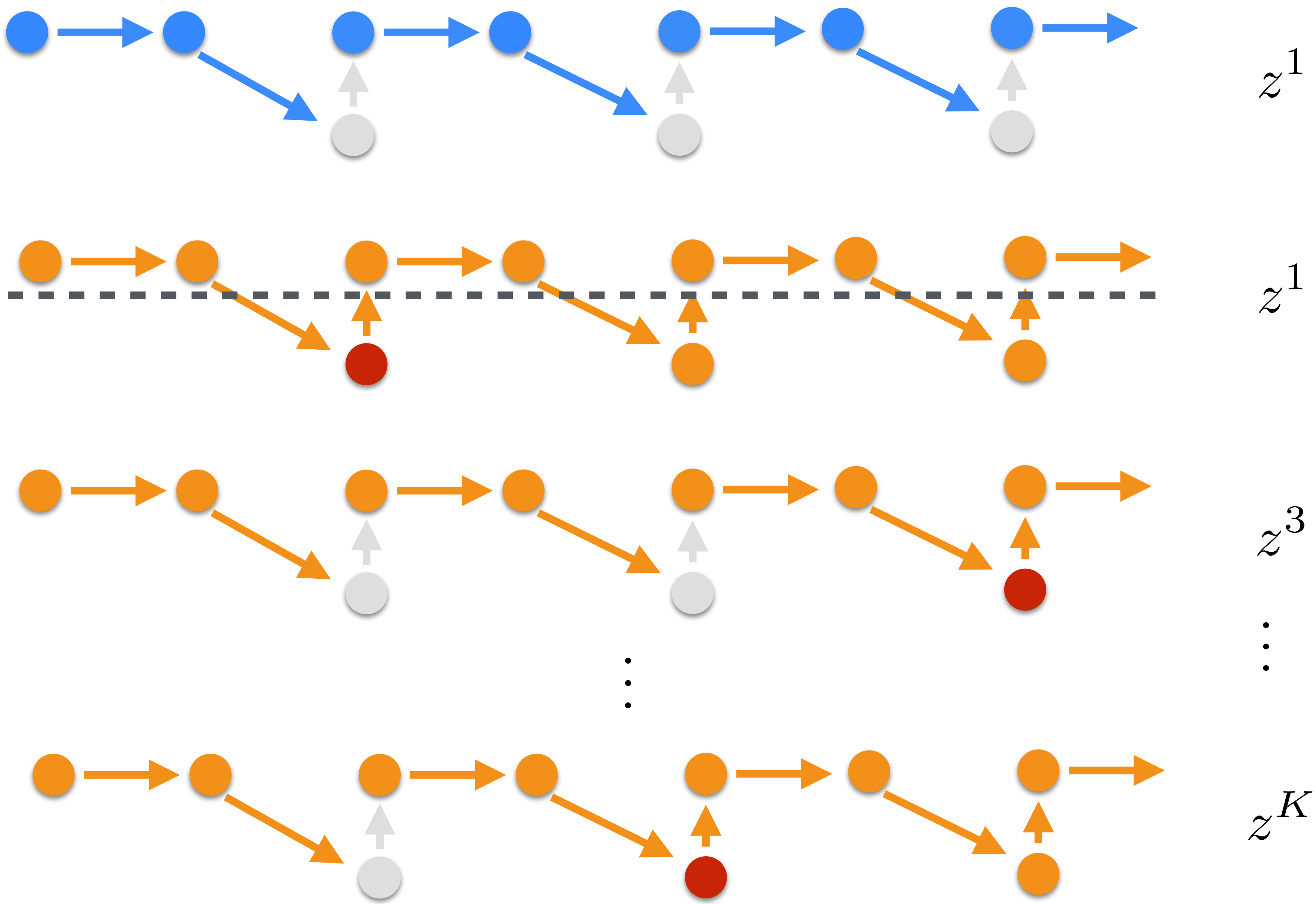
Probability of original trace continuation restarting proposal trace at m[th] sample

$$\alpha = \min\left(1, \frac{\gamma(\mathbf{x}')M\prod_{j=m}^{M} f_j(x_j|\theta_j)}{\gamma(\mathbf{x})M'\prod_{j=m}^{M'} f'_j(x'_j|\theta'_j)}\right)$$

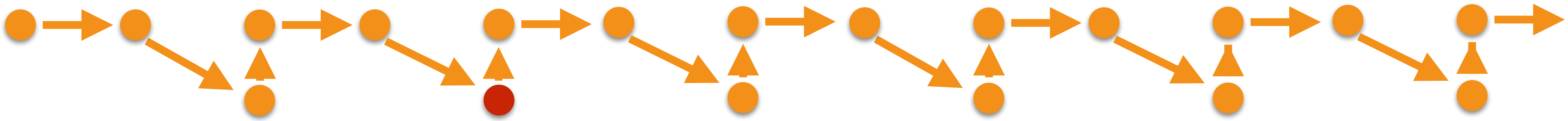
Number of sample statements in new trace

Probability of proposal trace continuation restarting original trace at m[th] sample

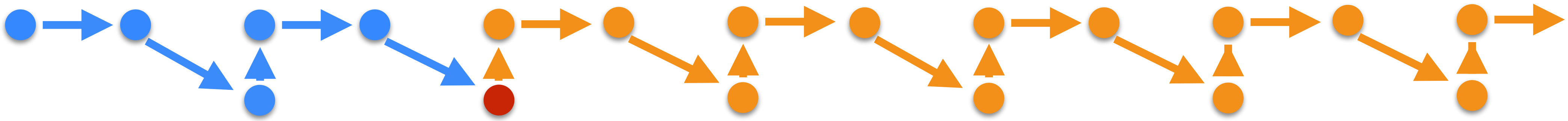
LMH Schematic



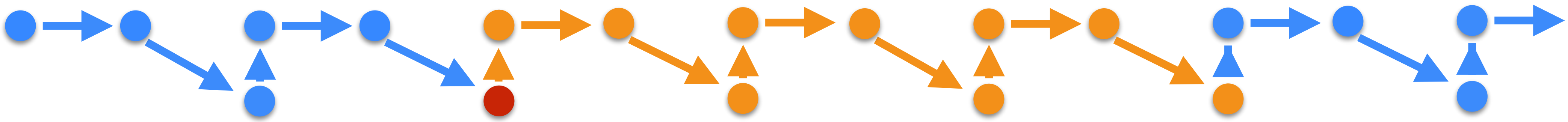
LMH Variants



D. Wingate, A. Stuhlmueller, and N. D. Goodman.
"Lightweight implementations of probabilistic programming languages via transformational compilation." AISTATS (2011).



with continuations: WebPPL
Anglican



"C3: Lightweight Incrementalized MCMC for Probabilistic Programs using Continuations and Callsite Caching."
D. Ritchie, A. Stuhlmuller, and N. D. Goodman. arXiv:1509.02151 (2015).

2015 : Probabilistic Programming

- Restricted (i.e. STAN, BUGS, infer.NET)
  - Easier inference problems -> fast
  - Impossible for users to denote some models
  - Fixed computation graph
- Unrestricted (i.e. Anglican, WebPPL)
  - Possible for users to denote all models
  - Harder inference problems -> slow
  - Dynamic computation graph

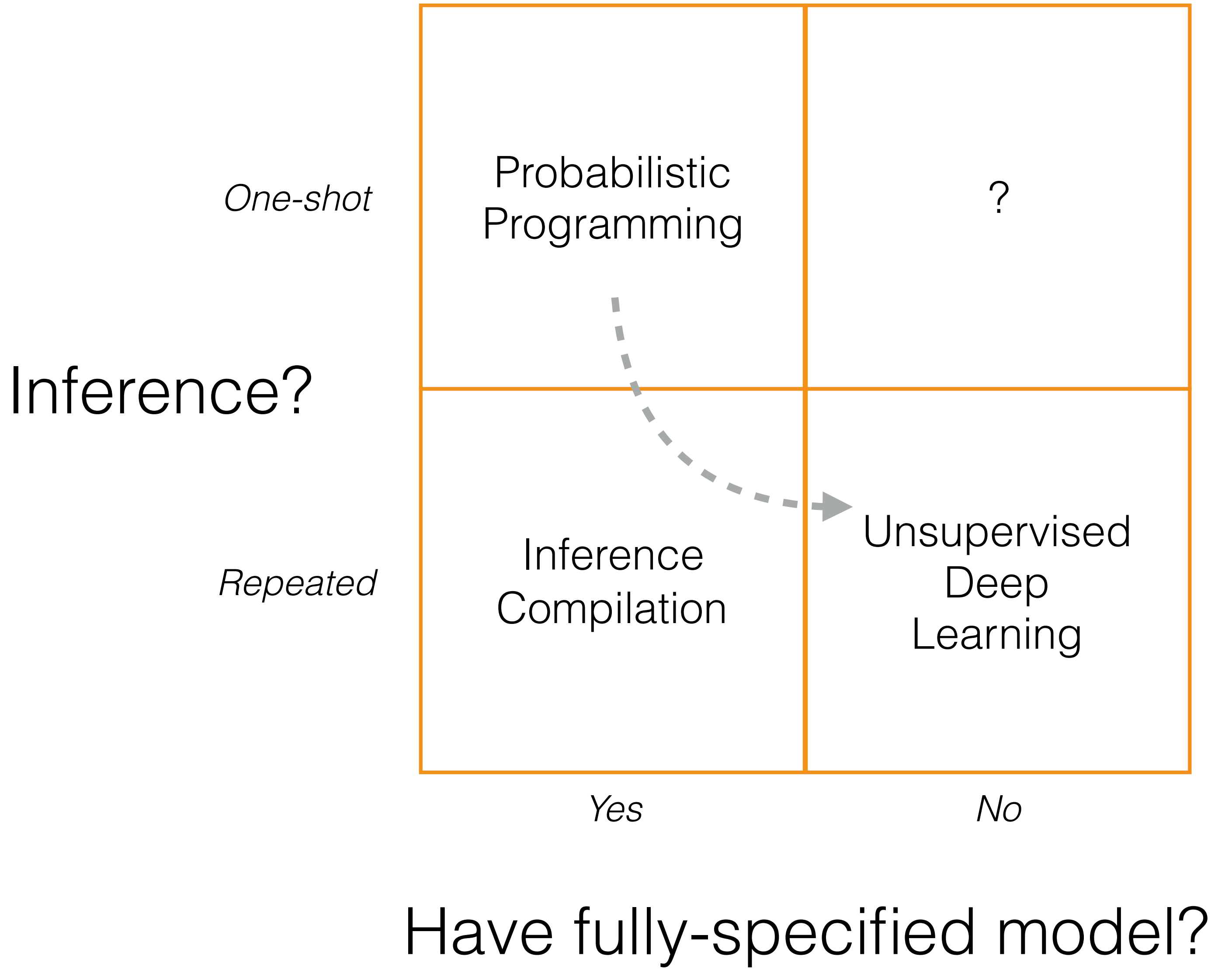
- Fixed, trusted model; one-shot inference

The AI/Repeated-Inference Challenge

"**Bayesian inference** is computationally expensive. Even approximate, sampling-based algorithms tend to take many iterations before they produce reasonable answers. In contrast, human recognition of words, objects, and scenes is extremely rapid, often taking only a few hundred milliseconds —only enough time for a **single pass from perceptual evidence to deeper interpretation**. Yet human perception and cognition are often well-described by **probabilistic inference in complex models**. How can we reconcile the speed of recognition with the expense of coherent probabilistic inference? How can we **build systems**, for applications like robotics and medical diagnosis, **that exhibit similarly rapid performance** at challenging inference tasks?"

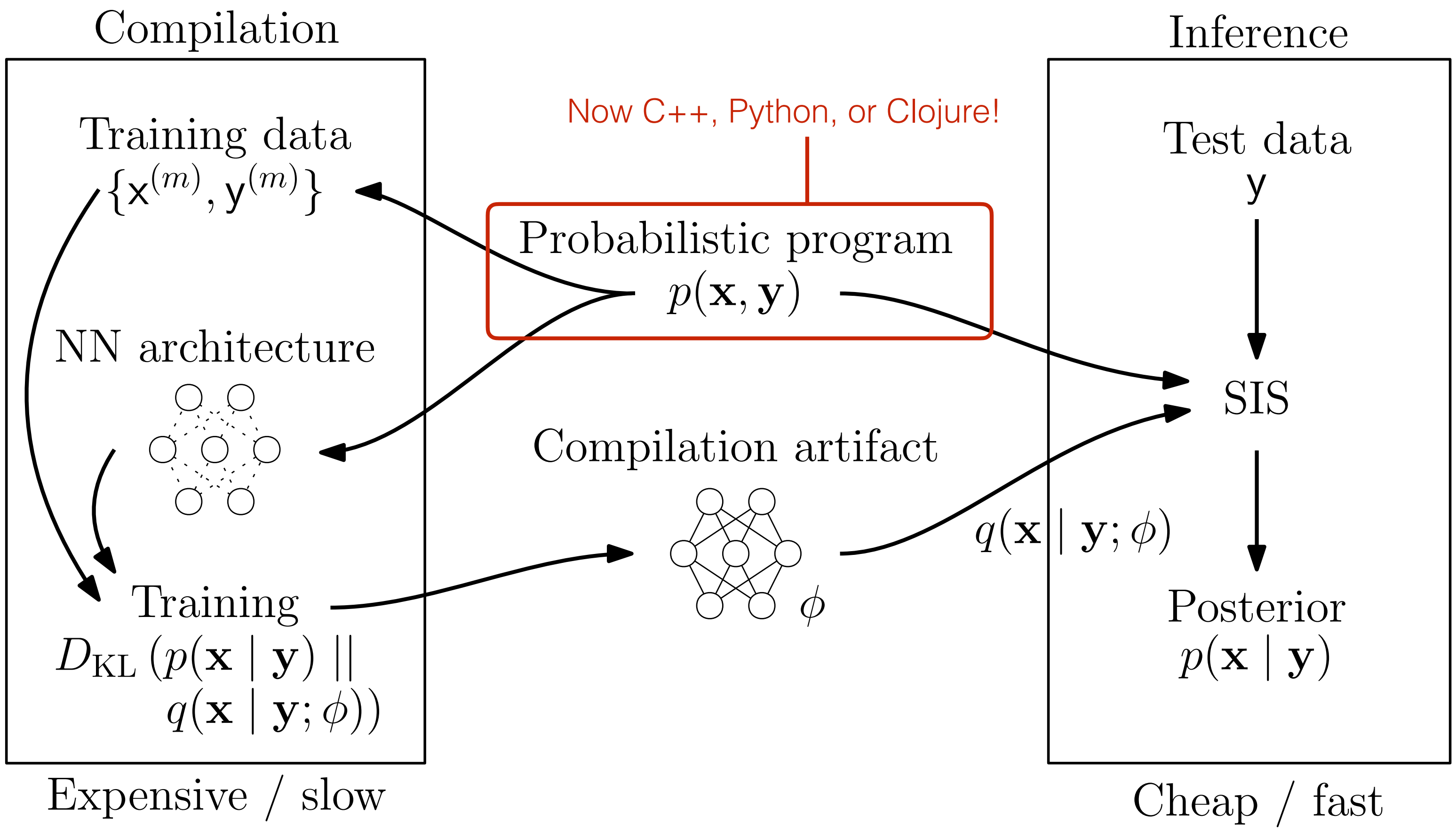
Stuhlmüller A, Taylor J, Goodman N. Learning stochastic inverses. In Advances in Neural Information Processing Systems 2013 (pp. 3048-3056).

Resulting Trend In Probabilistic Programming



Inference Compilation

Inference Compilation

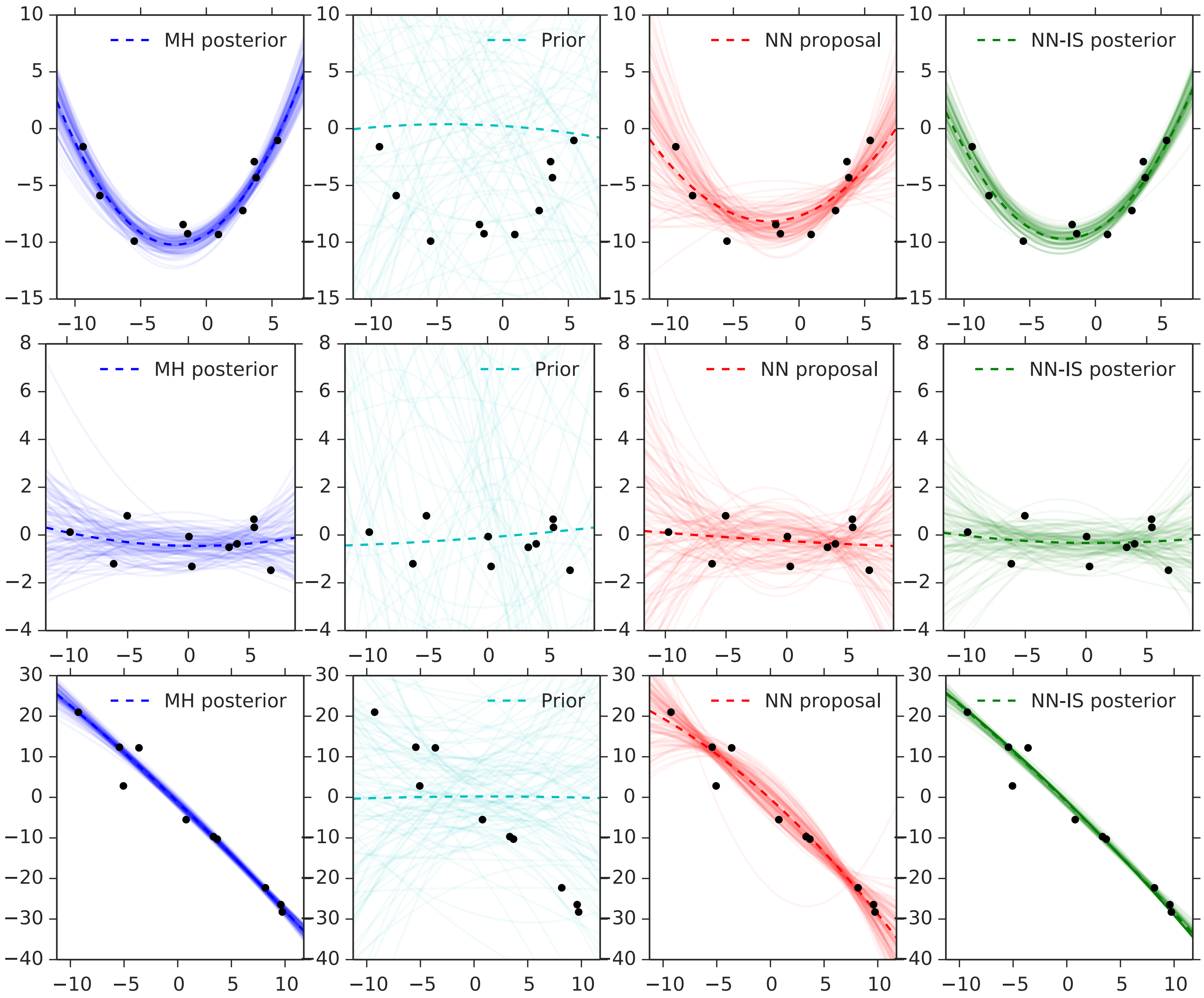


Input: an inference problem denoted in a probabilistic programming language

Output: a trained inference network (deep neural network "compilation artifact")

Le TA, Baydin AG, **Wood** F. Inference Compilation and Universal Probabilistic Programming. AISTATS. 2017.

Example Non-Conjugate Regression



Paige B, **Wood** F. Inference Networks for Sequential Monte Carlo in Graphical Models. ICML. JMLR W&CP 48: 3040-3049. 2016.

Captcha Breaking

Type		Baidu (2011)	Baidu (2013)	eBay	Yahoo	reCaptcha	Wikipedia	Facebook
Our method	RR	99.8%	99.9%	99.2%	98.4%	96.4%	93.6%	91.0%
	BT	72 ms	67 ms	122 ms	106 ms	78 ms	90 ms	90 ms
Bursztein et al. [15]	RR	38.68%	55.22%	51.39%	5.33%	22.67%	28.29%	
	BT	3.94 s	1.9 s	2.31 s	7.95 s	4.59 s		
Starostenko et al. [16]	RR				91.5%	54.6%		
	BT					$<$ 0.5 s		
Gao et al. [17]	RR	34%			55%	34%		
Gao et al. [18]	RR		51%		36%			
	BT		7.58 s		14.72 s			
Goodfellow et al. [6]	RR					99.8%		
Stark et al. [8]	RR					90%		

Facebook Captcha

Observed images		(W4kgvQ)	(uV7FeWB)	(MqhnpT)
Inference				
Training traces	10^7	W4kgvQ	uV7EeWB	MqhnpT
	10^6	WA4rjvQ	uV7FeWB	MypppT
	10^5	Woxewd9	mTTEMMm	RIrpES
	10^4	BKvu2Q	C9QDsoN	rS5FP2B

$40M raise

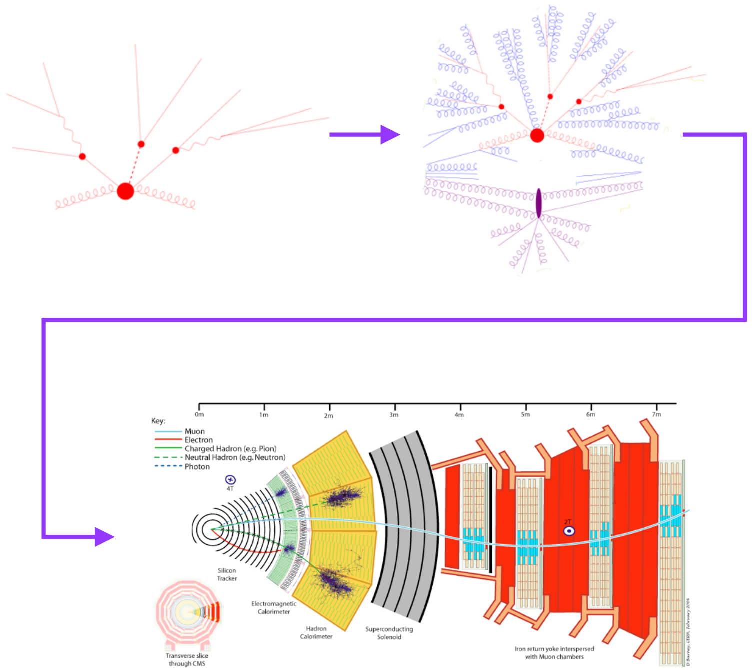
I’m Hiring

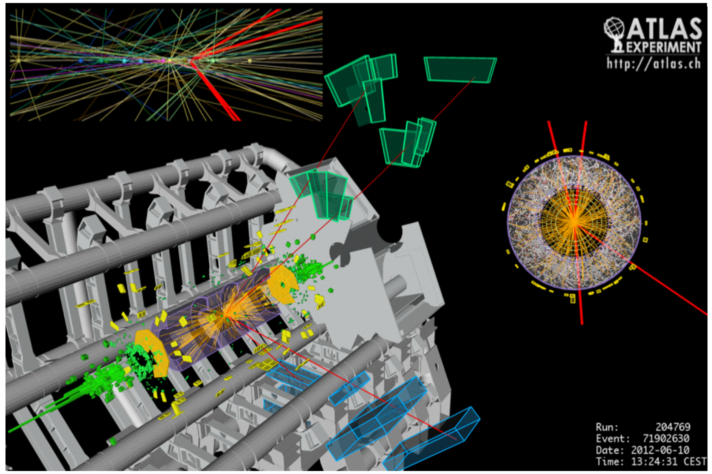
- Postdoc(s)
- PhD students

~$???; ’17-’20 New Physics Via ATLAS Simulator Inversion

e.g.
Sherpa

e.g.
Geant





$\mathbf{x}$	$\mathbf{y}$
event & detector simulators	ATLAS detector output

$1.8M USD; ’17-’21 — Hasty: A Generative Model Compiler

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

ABOUT US / OUR RESEARCH

Data-Driven Discovery of Models (D3M)

Mr. Wade Shen