From lazy evaluation to Gibbs sampling

Chung-chieh Shan
Indiana University

March 19, 2014

This work is supported by DARPA grant FA8750-14-2-0007.

Come to Indiana University to create essential abstractions and practical languages for clear, robust and efficient programs.

Dan Friedman

relational & logic languages, meta-circularity & reflection

Amr Sabry

quantum computing, type theory, information effects

Jeremy Siek

gradual typing, mechanized metatheory, high performance

Ryan Newton

streaming, distributed & GPU DSLs, Haskell deterministic parallelism

Chung-chieh Shan

probabilistic programming, semantics

Sam Tobin-Hochstadt

types for untyped languages, contracts, languages for the Web

Check out our work: Boost Libraries · Build to Order BLAS · C++ Concepts · Chapel Generics · HANSEI · JavaScript Modules · Racket & Typed Racket · miniKanren · LVars · monad-par · meta-par · WaveScript

http://lambda.soic.indiana.edu/

Today
Mar 11

67°F 40°F

CHANCE OF RAIN:
0%

Mostly Sunny

Wed
Mar 12

33° 9°

CHANCE OF SNOW:
90%

SNOWFALL
4-6 in

Snow / Wind

Probabilistic programming

Alice beat Bob at a game. Is she better than him at it?

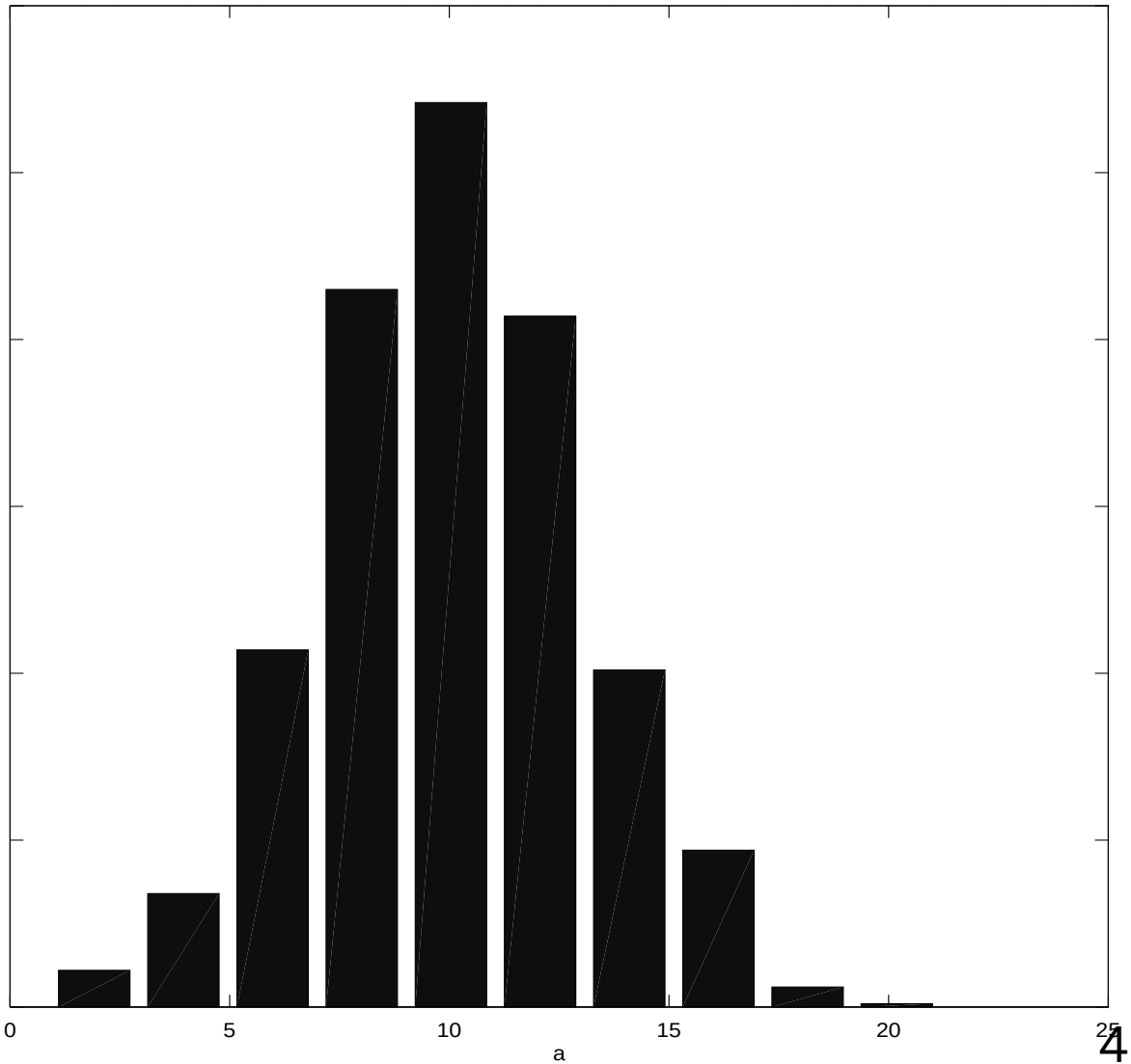
Generative story

Probabilistic programming

Alice beat Bob at a game. Is she better than him at it?

Generative story

```
a <- normal 10 3
```

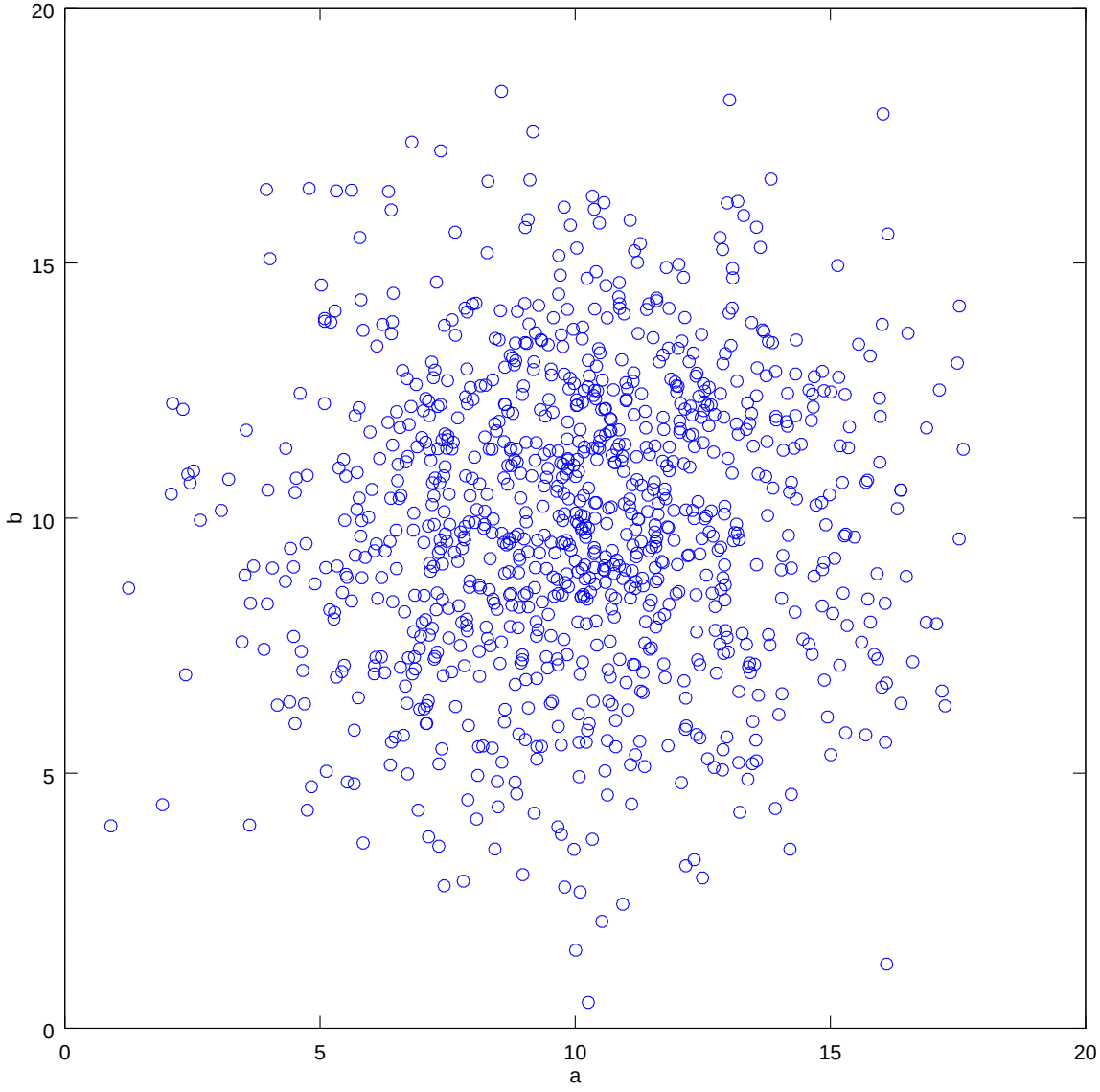


Probabilistic programming

Alice beat Bob at a game. Is she better than him at it?

Generative story

```
a <- normal 10 3
b <- normal 10 3
```

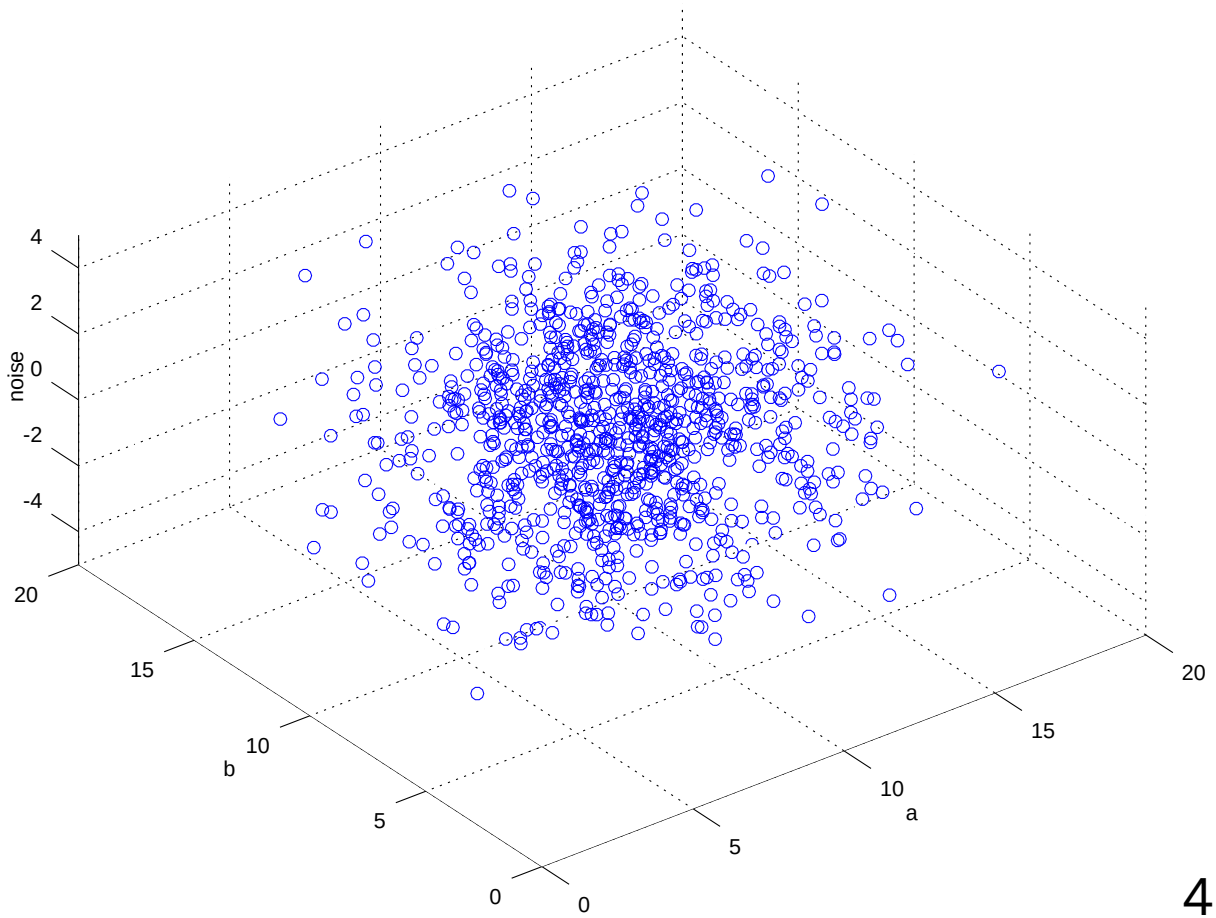


Probabilistic programming

Alice beat Bob at a game. Is she better than him at it?

Generative story

```
a <- normal 10 3
b <- normal 10 3
l <- normal  0 2
```



Probabilistic programming

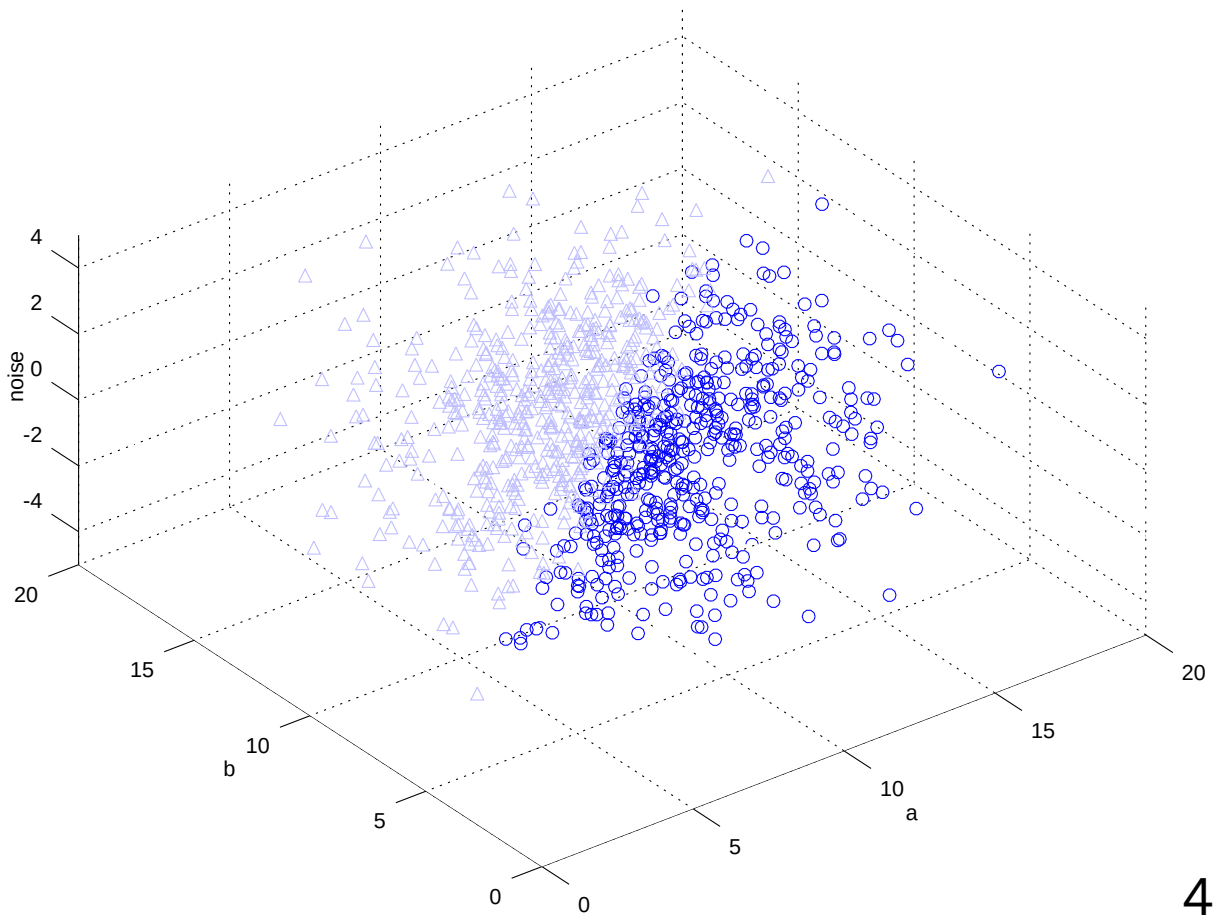
Alice beat Bob at a game. Is she better than him at it?

Generative story

```
a <- normal 10 3
b <- normal 10 3
l <- normal  0 2
```

Observed effect

```
condition (a-b > l)
```



Probabilistic programming

Alice beat Bob at a game. Is she better than him at it?

Generative story

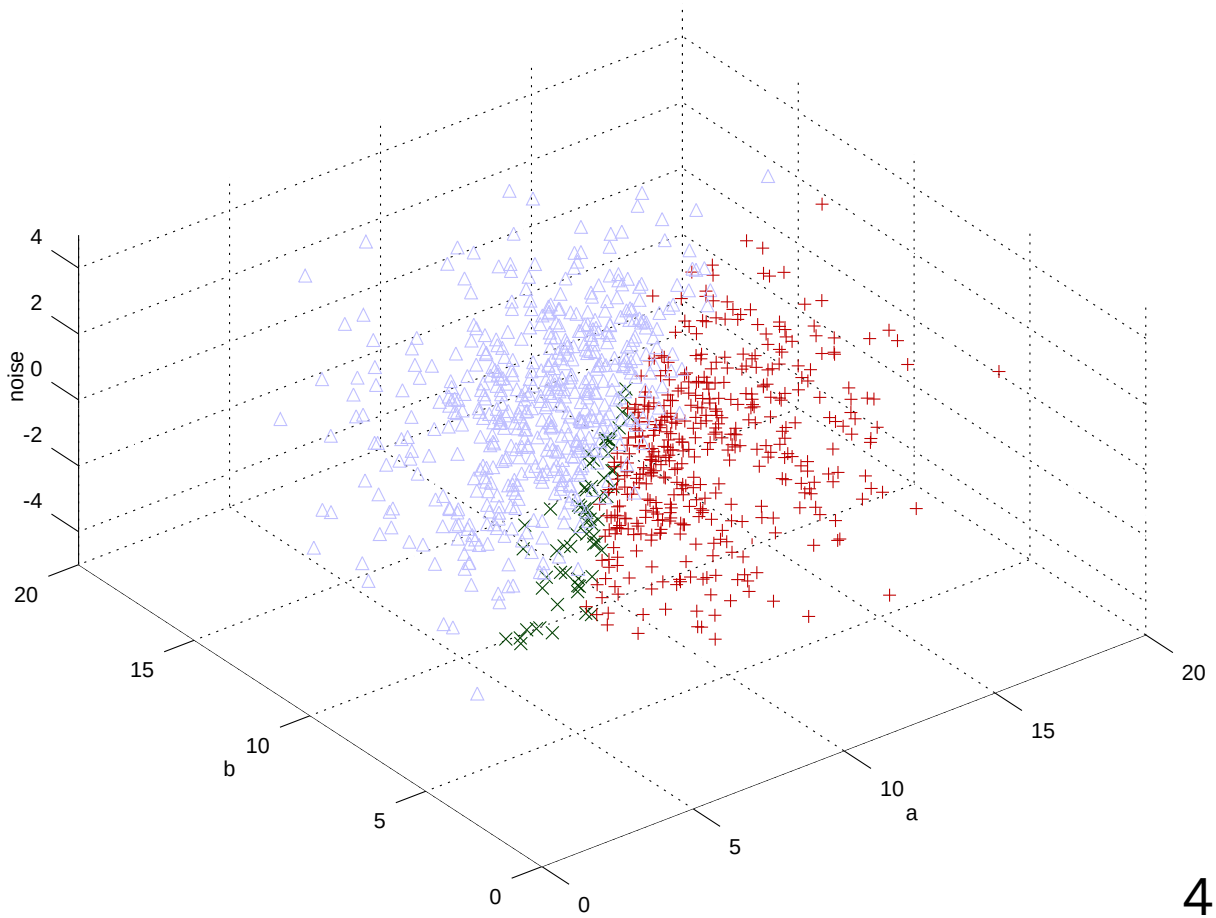
```
a <- normal 10 3
b <- normal 10 3
l <- normal  0 2
```

Observed effect

```
condition (a-b > l)
```

Hidden cause

```
return (a > b)
```



Probabilistic programming

Alice beat Bob at a game. Is she better than him at it?

Generative story

```
a <- normal 10 3
b <- normal 10 3
l <- normal  0 2
```

Observed effect

```
condition (a-b > l)
```

Hidden cause

```
return (a > b)
```

Denoted measure:

$$\lambda c. \int_{N(10,3)} da \int_{N(10,3)} db \int_{N(0,2)} dl \; \langle a - b > l \rangle \, c(a > b)$$

Sampling is hard. Let's do math!

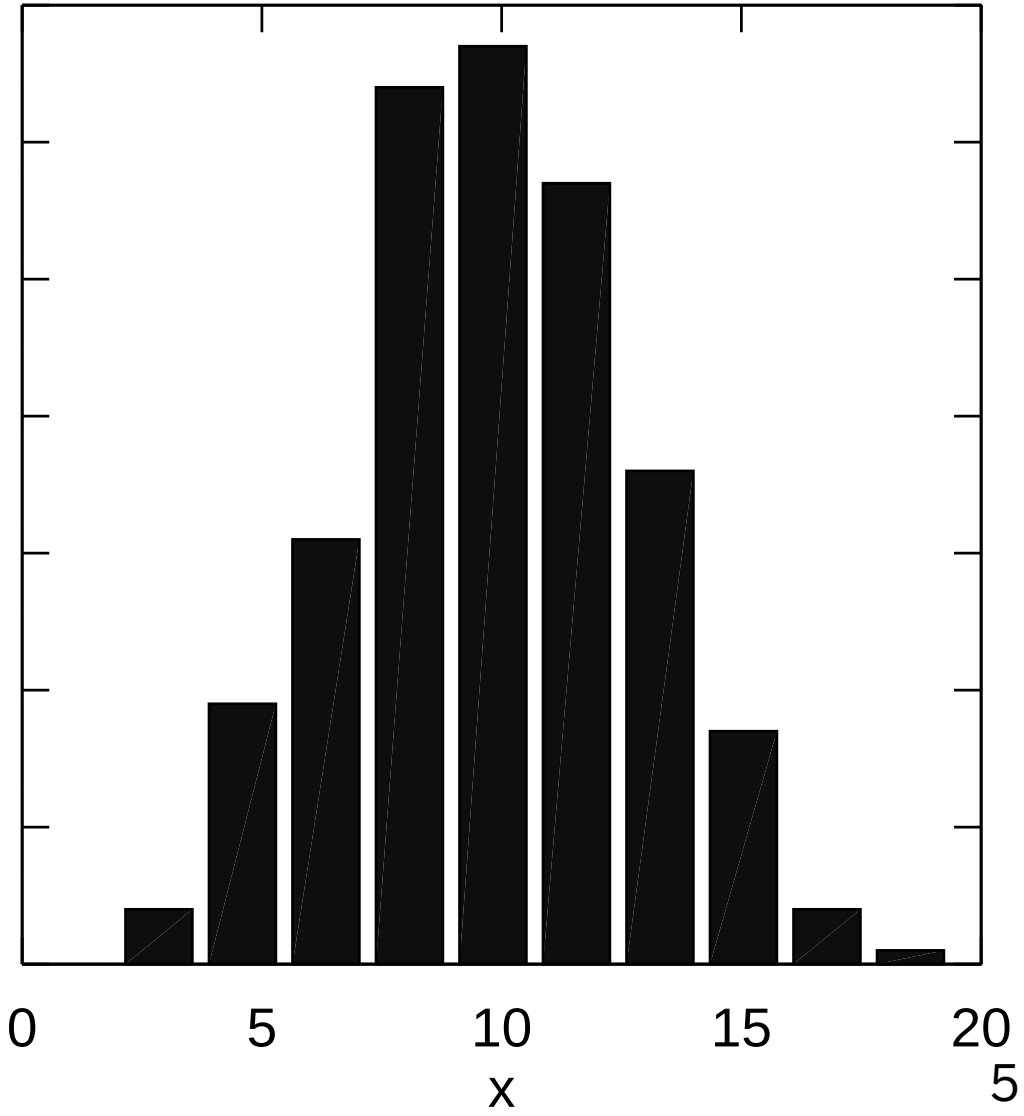
Filtering = tracking current state with uncertainty

Sampling is hard. Let's do math!

Filtering = tracking current state with uncertainty

Generative story

```
x  <- normal    10 3
```

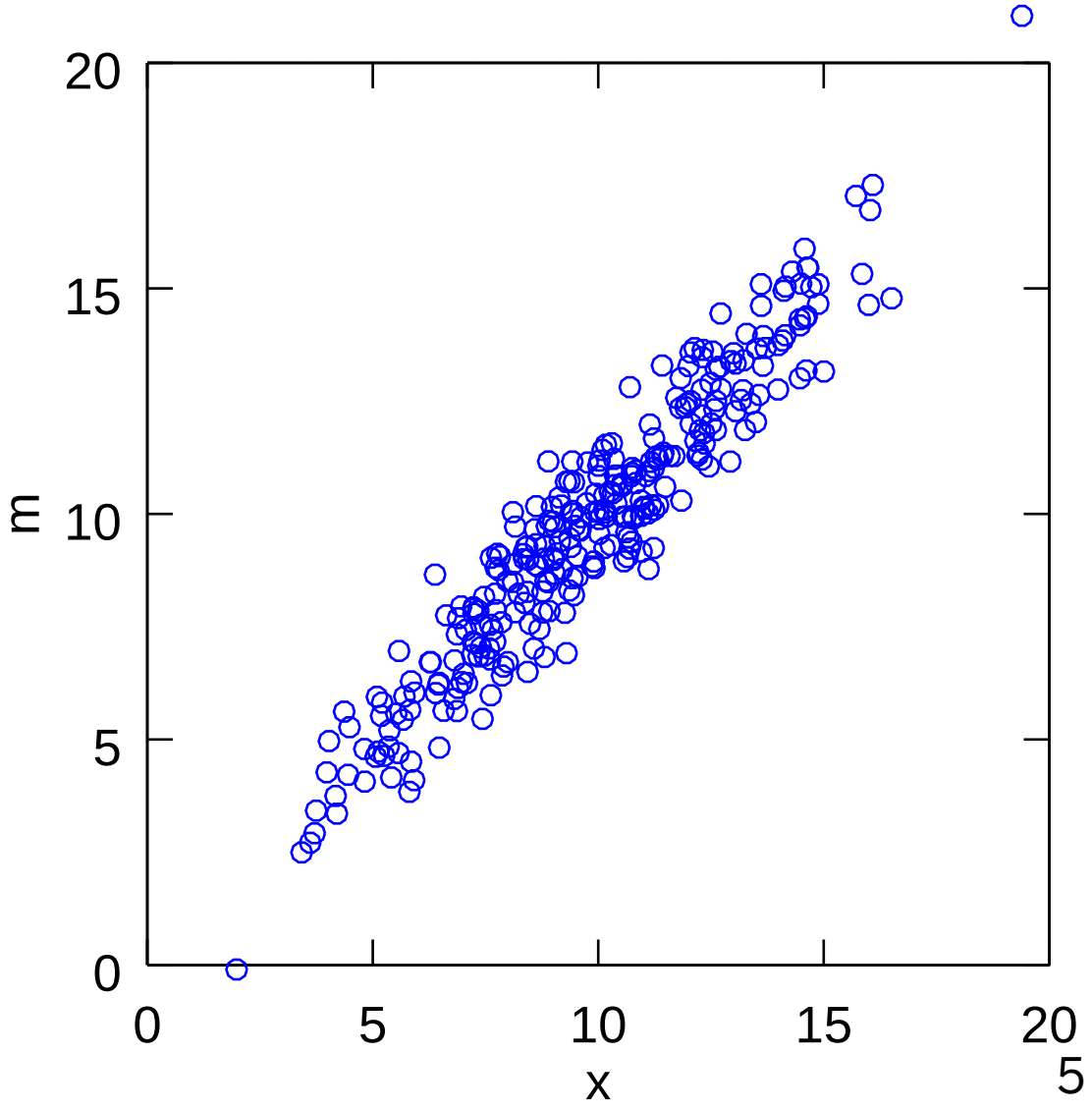


Sampling is hard. Let's do math!

Filtering = tracking current state with uncertainty

Generative story

```
x  <- normal   10 3
m  <- normal    x 1
```

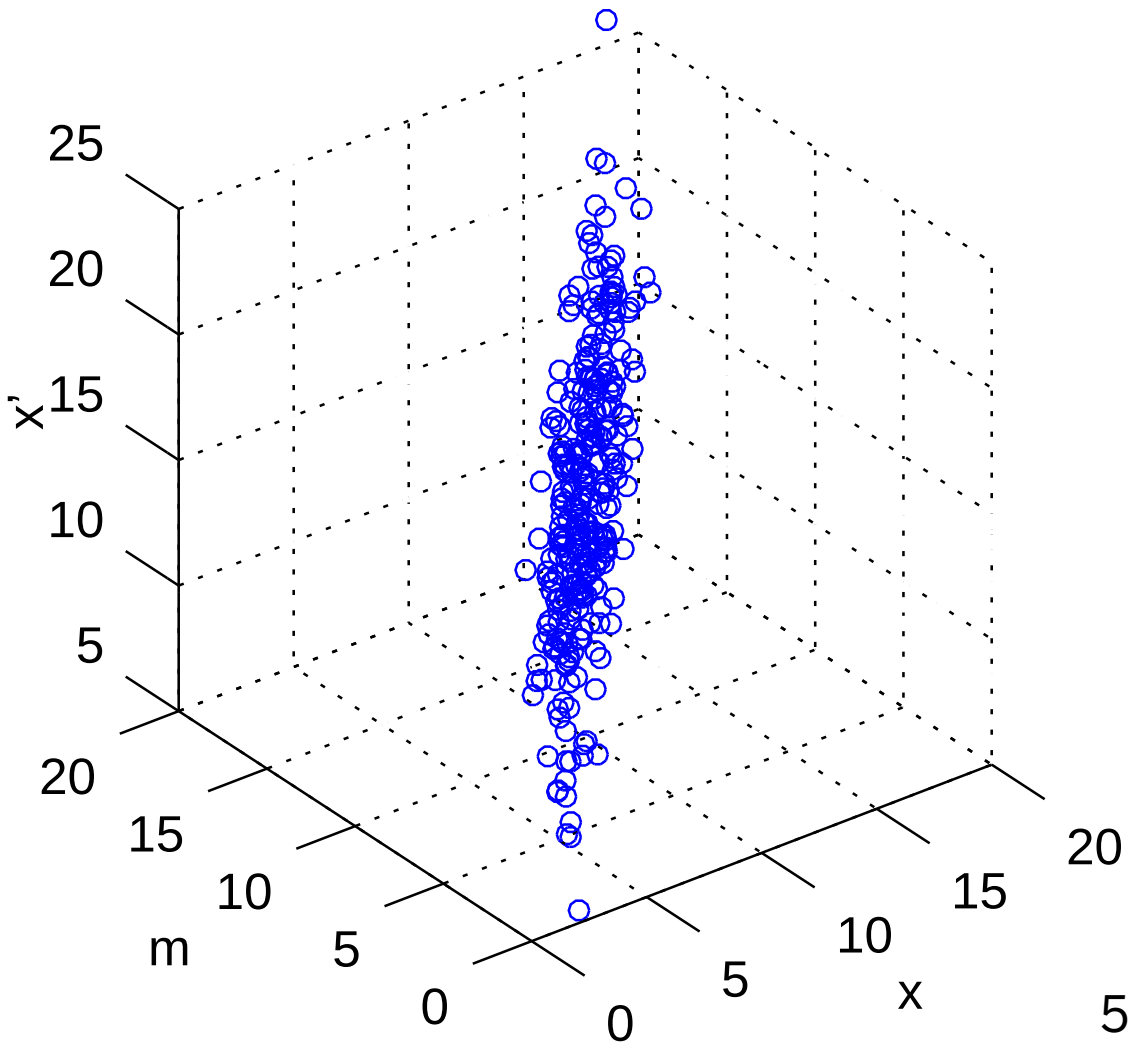


Sampling is hard. Let's do math!

Filtering = tracking current state with uncertainty

Generative story

```
x  <- normal     10 3
m  <- normal      x 1
x' <- normal (x+5) 2
```



Sampling is hard. Let's do math!

Filtering = tracking current state with uncertainty

Generative story

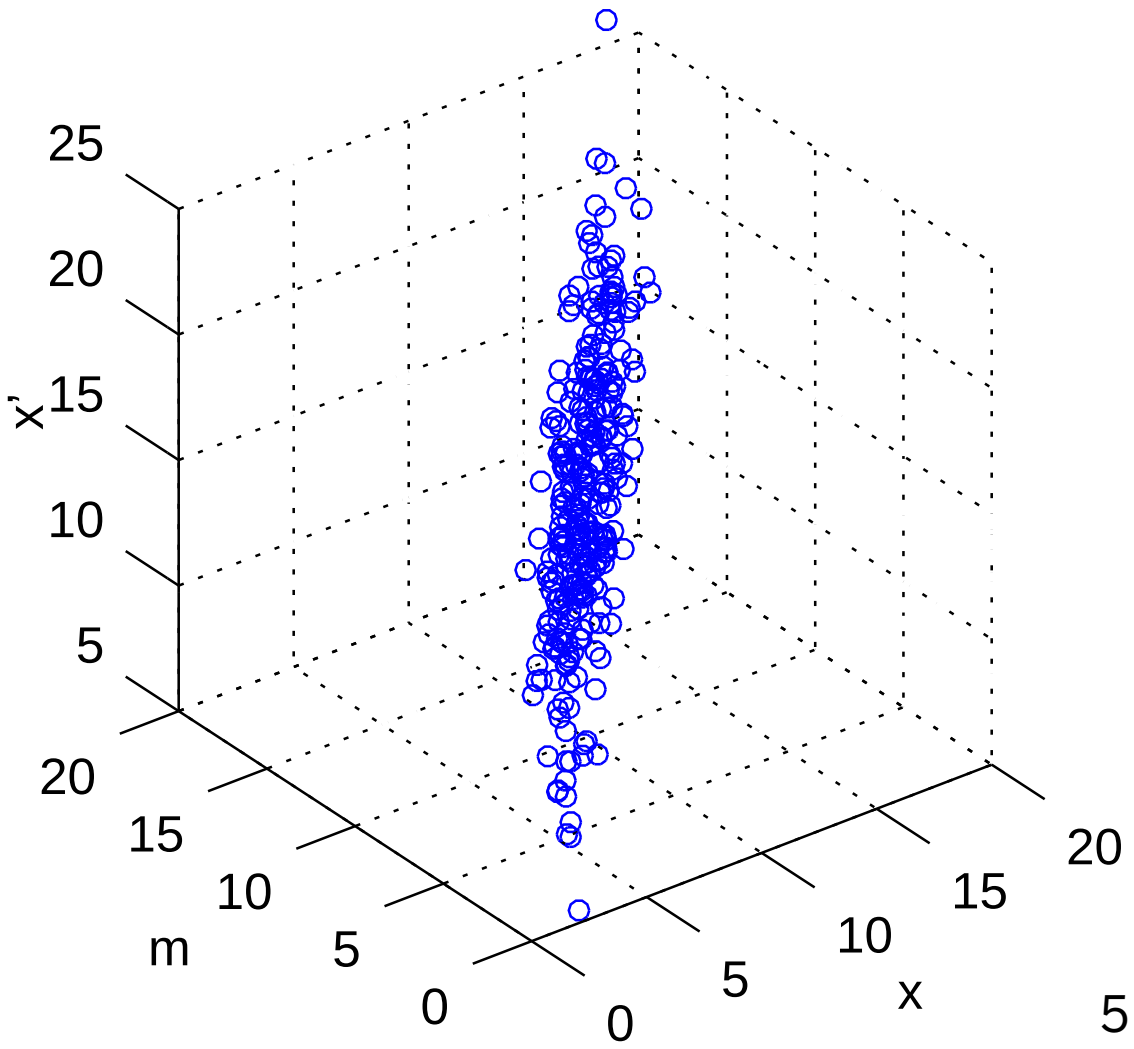
```
x  <- normal     10 3
m  <- normal      x 1
x' <- normal (x+5) 2
```

Observed effect

```
condition (m = 9)
```

Hidden cause

```
return x'
```



Sampling is hard. Let's do math!

Filtering = tracking current state with uncertainty
Conditioning = clamp first/outermost choice/integral

Generative story

```
x  <- normal    10 3
m  <- normal     x 1
x' <- normal (x+5) 2
```

$$\texttt{m <- normal } 10 \ \sqrt{10}$$

$$\texttt{x <- normal } \left(\tfrac{9}{10}\texttt{m} + \tfrac{1}{10}10\right) \ \sqrt{\tfrac{9}{10}}$$

Observed effect

```
condition (m = 9)
```

Hidden cause

```
return x'
```

Sampling is hard. Let's do math!

Filtering = tracking current state with uncertainty
Conditioning = clamp first/outermost choice/integral

Generative story

~~`x <- normal 10 3`~~
~~`m <- normal x 1`~~
`x' <- normal (x+5) 2`

~~`m <- normal 10` $\sqrt{10}$~~ `let m = 9`
`x <-` `normal` $(\frac{9}{10}\texttt{m} + \frac{1}{10}10)$ $\sqrt{\frac{9}{10}}$

Observed effect

~~`condition (m = 9)`~~

Hidden cause

`return x'`

Sampling is hard. Let's do math!

Filtering = tracking current state with uncertainty
Conditioning = clamp first/outermost choice/integral
Conjugacy = absorb one choice/integral into another

Generative story

```
x  <- normal 91/10 √(9/10)
x' <- normal (x+5) 2
```

Hidden cause

```
return x'
```

Sampling is hard. Let's do math!

Filtering = tracking current state with uncertainty
Conditioning = clamp first/outermost choice/integral
Conjugacy = absorb one choice/integral into another

Generative story

`x' <- normal` $\frac{141}{10}$ $\sqrt{\frac{49}{10}}$

Hidden cause

`return x'`

Math is hard. Let's go sampling!

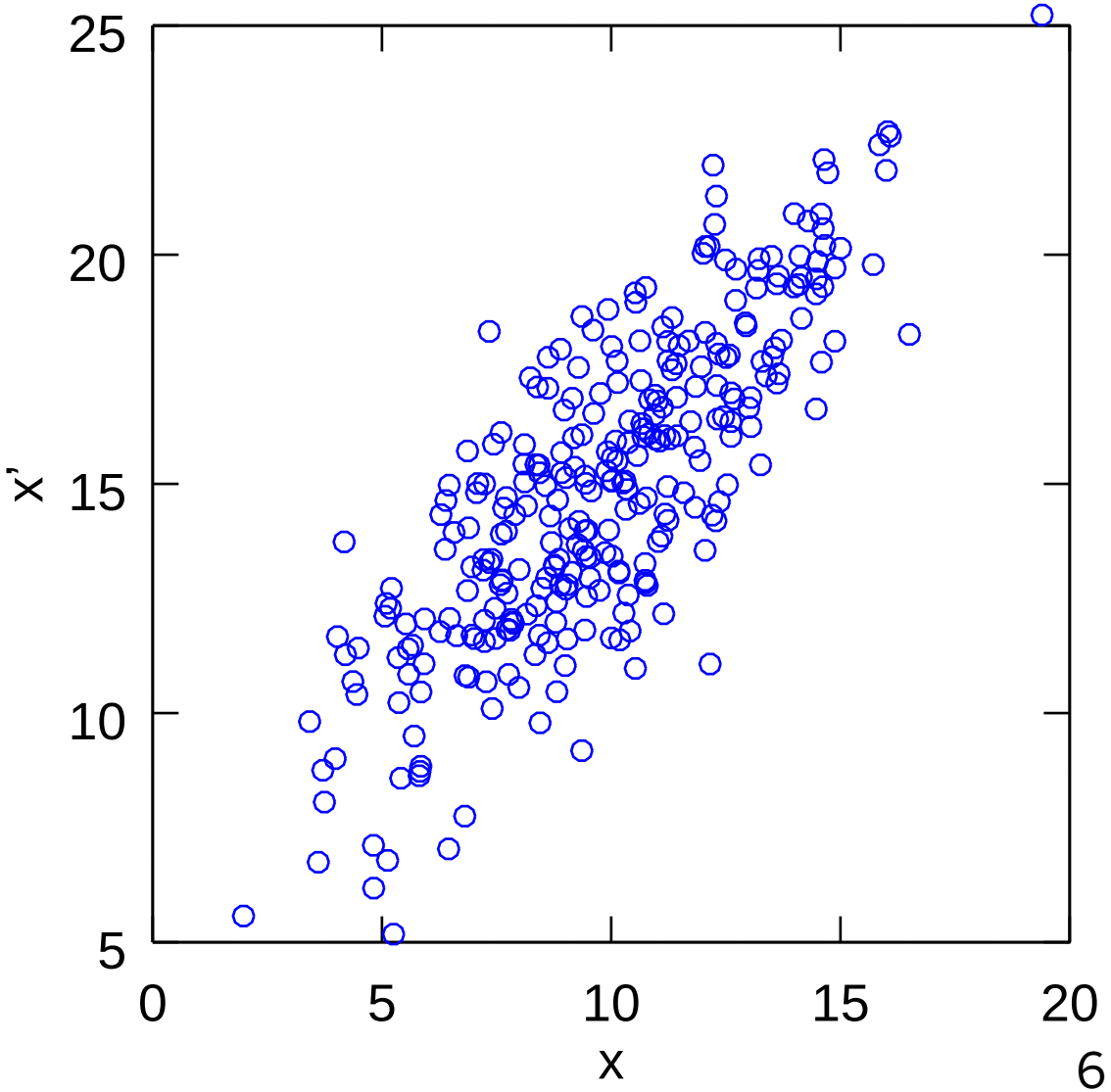
Each sample has an *importance weight*

Math is hard. Let's go sampling!

Each sample has an *importance weight*

Generative story

```
x  <- normal     10 3
m  <- normal      x 1
x' <- normal (x+5) 2

Observed effect

condition (m = 9)

Hidden cause

return x'
```



Math is hard. Let's go sampling!

Each sample has an *importance weight*:
How much did we rig our random choices to avoid rejection?

Generative story

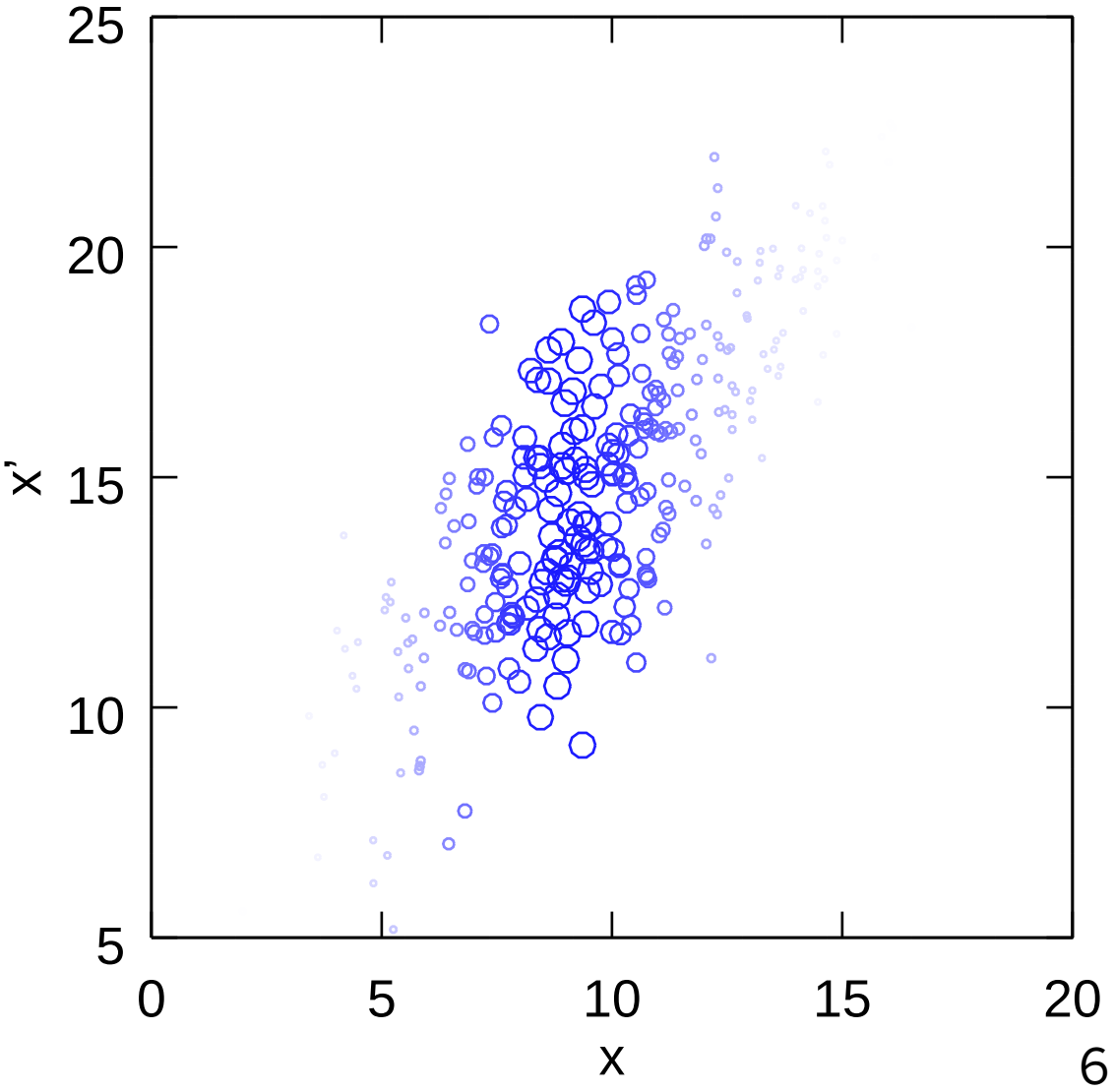
```
x  <- normal     10 3
m  <- normal      x 1
x' <- normal (x+5) 2
```

Observed effect

```
condition (m = 9)
```

Hidden cause

```
return x'
```



The story so far

1. Declarative program specifies generative story and observed effect.
2. We try mathematical optimizations, but still need to sample.
3. A sampler should generate a stream of samples (run-weight pairs) whose histogram matches the specified conditional distribution.
4. Importance sampling generates each sample independently.

Monte Carlo Markov Chain

For harder search problems,
keep the previous sampling run in memory, and
take a random walk that lingers around high-probability runs.

1. Initialise $x^{(0)}$.
2. For $i = 0$ to $N - 1$
   - Sample $u \sim \mathcal{U}_{[0,1]}$.
   - Sample $x^\star \sim q(x^\star | x^{(i)})$.
   - If $u < \mathcal{A}(x^{(i)}, x^\star) = \min\left\{1, \frac{p(x^\star)q(x^{(i)}|x^\star)}{p(x^{(i)})q(x^\star|x^{(i)})}\right\}$

     $$x^{(i+1)} = x^\star$$

     else

     $$x^{(i+1)} = x^{(i)}$$

Monte Carlo Markov Chain

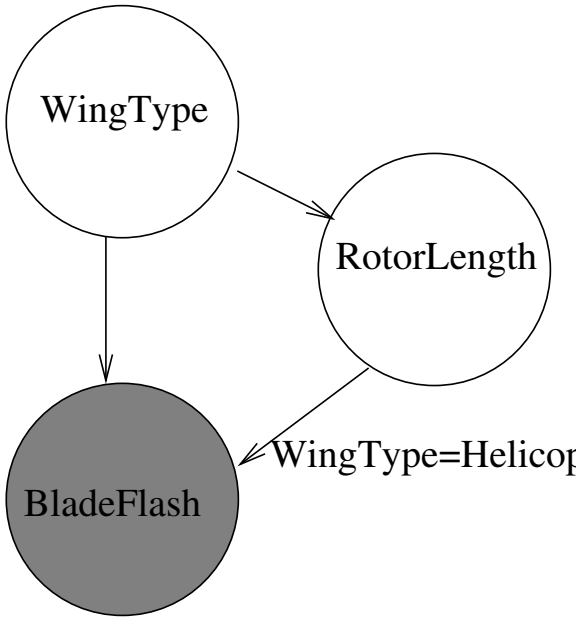
For harder search problems,
keep the previous sampling run in memory, and
take a random walk that lingers around high-probability runs.

1. Initialise $x^{(0)}$.
2. For $i = 0$ to $N - 1$
   - Sample $u \sim \mathcal{U}_{[0,1]}$.
   - Sample $x^\star \sim q(x^\star | x^{(i)})$.
   - If $u < \mathcal{A}(x^{(i)}, x^\star) = \min\left\{1, \frac{p(x^\star)q(x^{(i)}|x^\star)}{p(x^{(i)})q(x^\star|x^{(i)})}\right\}$

     $$x^{(i+1)} = x^\star$$

     else

     $$x^{(i+1)} = x^{(i)}$$



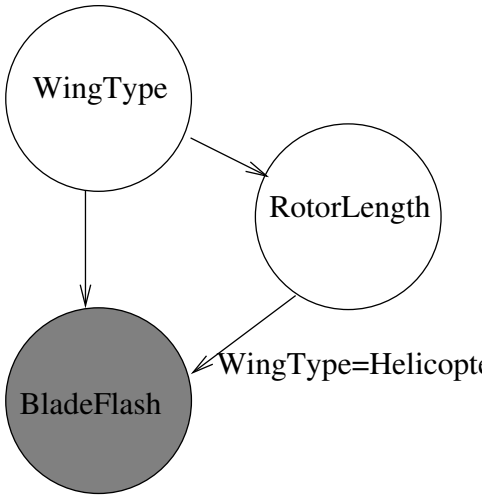
Want: 1. match dimensions 2. reject less 3. infinite domain

A lazy probabilistic language

```
data Code = Evaluate [Loc] ([Value] -> Code)
          | Allocate Code  (Loc     -> Code)
          | Generate [(Value, Prob)]

type Prob = Double

type Subloc = Int
type Loc = [Subloc]

data Value = Bool Bool | ...
```

A lazy probabilistic language

```
data Code = Evaluate [Loc] ([Value] -> Code)
          | Allocate Code  (Loc     -> Code)
          | Generate [(Value, Prob)]

bernoulli :: Prob -> Code
bernoulli p = Generate [(Bool True , p  ),
                        (Bool False, 1-p)]

example :: Code
example = Allocate (bernoulli 0.5) $ \w ->
          Allocate (bernoulli 0.5) $ \r ->
          Evaluate [w] $ \[Bool w] ->
          if w then Evaluate [r] $ \[Bool r] ->
                    if r then bernoulli 0.4
                         else bernoulli 0.8
               else bernoulli 0.2
```



Through the lens of lazy evaluation

To **match dimensions**, Wingate et al.'s MH sampler
reuses random choices in the heap from the previous run.
(*memoization*)

To **reject less**, Arora et al.'s Gibbs sampler
evaluates code in the context of its desired output.
(*destination passing*)

Summary

Probabilistic programming

- Denote measure by generative story
- Run backwards to infer cause from effect

Mathematical reasoning

- Define conditioning
- Reduce sampling
- Avoid rejection

Lazy evaluation

- Match dimensions (reversible jump)
- Reject less (Gibbs sampling)
- Infinite domain?