CS4340: Probabilistic Programming Seminar

Lecture 3

Recap of the previous lecture

- Probabilistic programs are
  - A powerful modelling tool
  - Programs with two special statements: sample and observe

Recap of the previous lecture

```
function  probabilisticHelloWorld():

        var coin1 = sample( Bernoulli(0.5) )
        var coin2 = sample( Bernoulli(0.5) )
        var coin3 = sample( Bernoulli(0.5) )
        observe( coin2 == 1 )

        return coin1 + coin2 + coin3
end
```

$$p(x, y) = \prod_{t=1}^{T} f_{a_t}(x_t \mid x_{1:t-1}) \prod_{n=1}^{N} g_n(y_n \mid x_{1:\tau(n)})$$

"Prior" probs
probabilities of sample

"Likelihood" probs
probabilities of observe

This lecture

How do we calculate $p(x, y)$ efficiently?

This lecture

How do we calculate $p(x, y)$ efficiently?

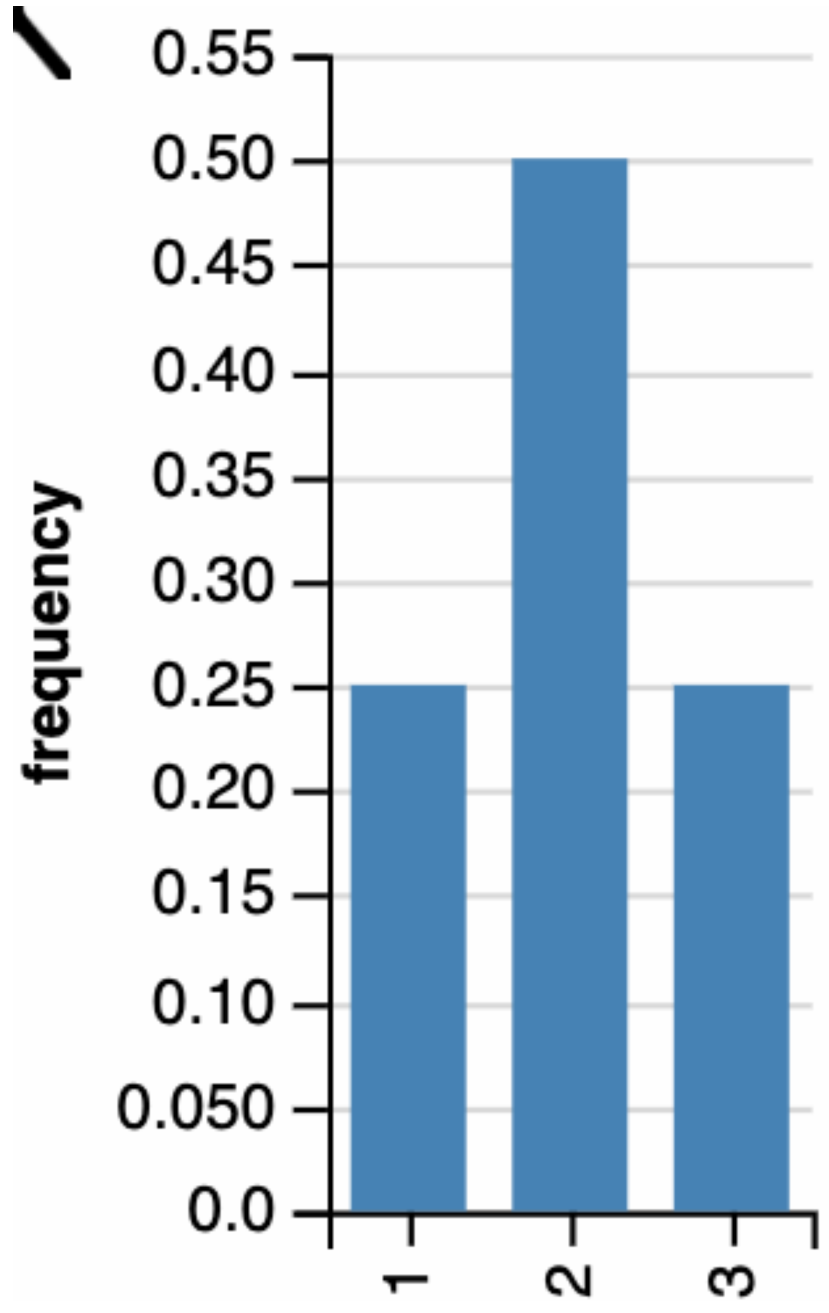
Importance sampling

Metropolis-Hastings MCMC

Particle filtering

Probabilistic inference

```
function  probabilisticHelloWorld():

        var coin1 = sample( Bernoulli(0.5) )
        var coin2 = sample( Bernoulli(0.5) )
        var coin3 = sample( Bernoulli(0.5) )
        observe( coin2 == 1 )

        return coin1 + coin2 + coin3
end
```



This lecture

How do we calculate $p(x, y)$ efficiently?

Importance sampling

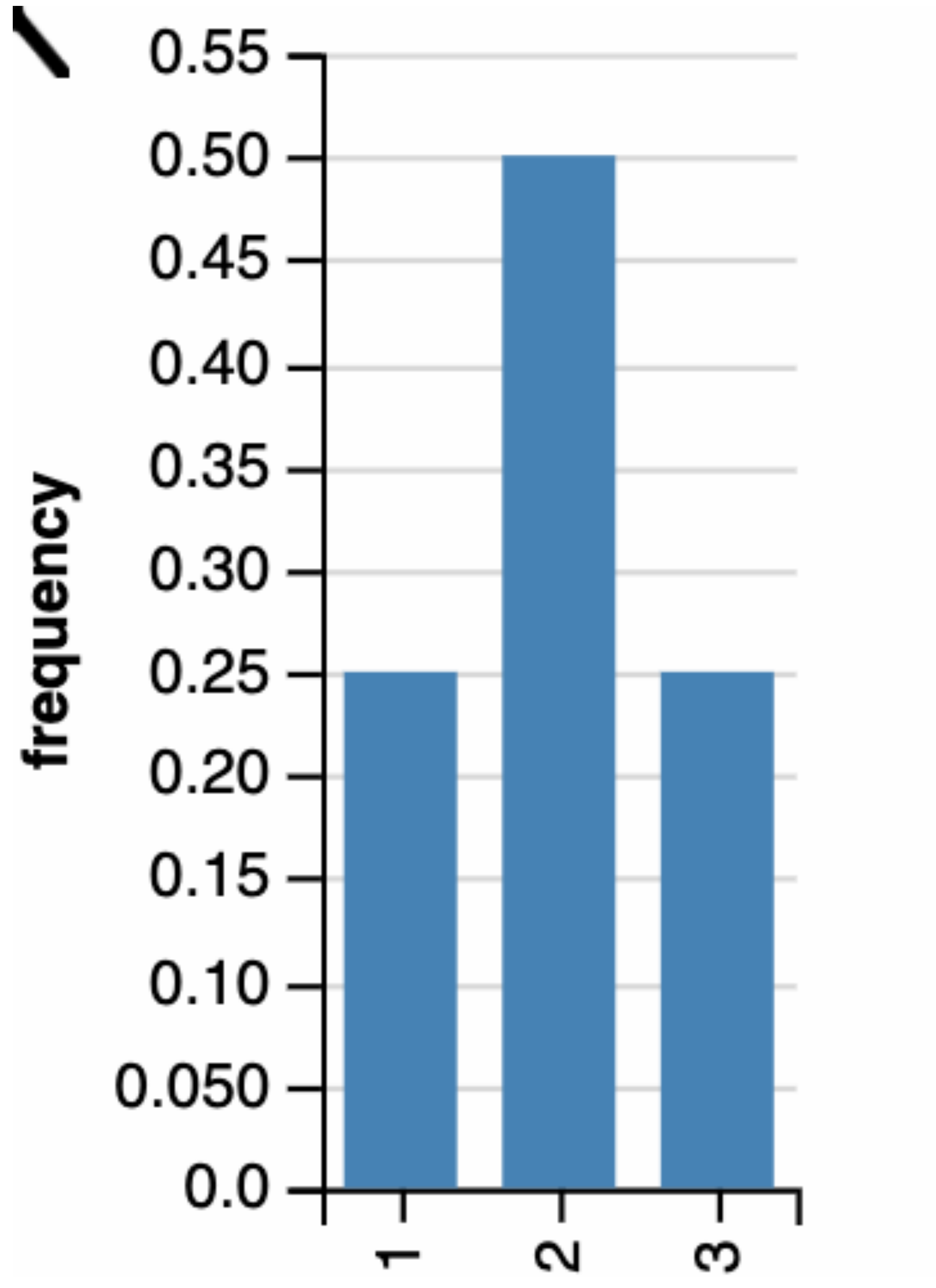
Metropolis-Hastings MCMC

Particle filtering

Probabilistic inference
Intuition

Probabilistic inference

```
function  probabilisticHelloWorld():

    var coin1 = sample( Bernoulli(0.5) )
    var coin2 = sample( Bernoulli(0.5) )
    var coin3 = sample( Bernoulli(0.5) )
    observe( coin2 == 1 )

    return coin1 + coin2 + coin3
end
```



Probabilistic inference: Enumeration

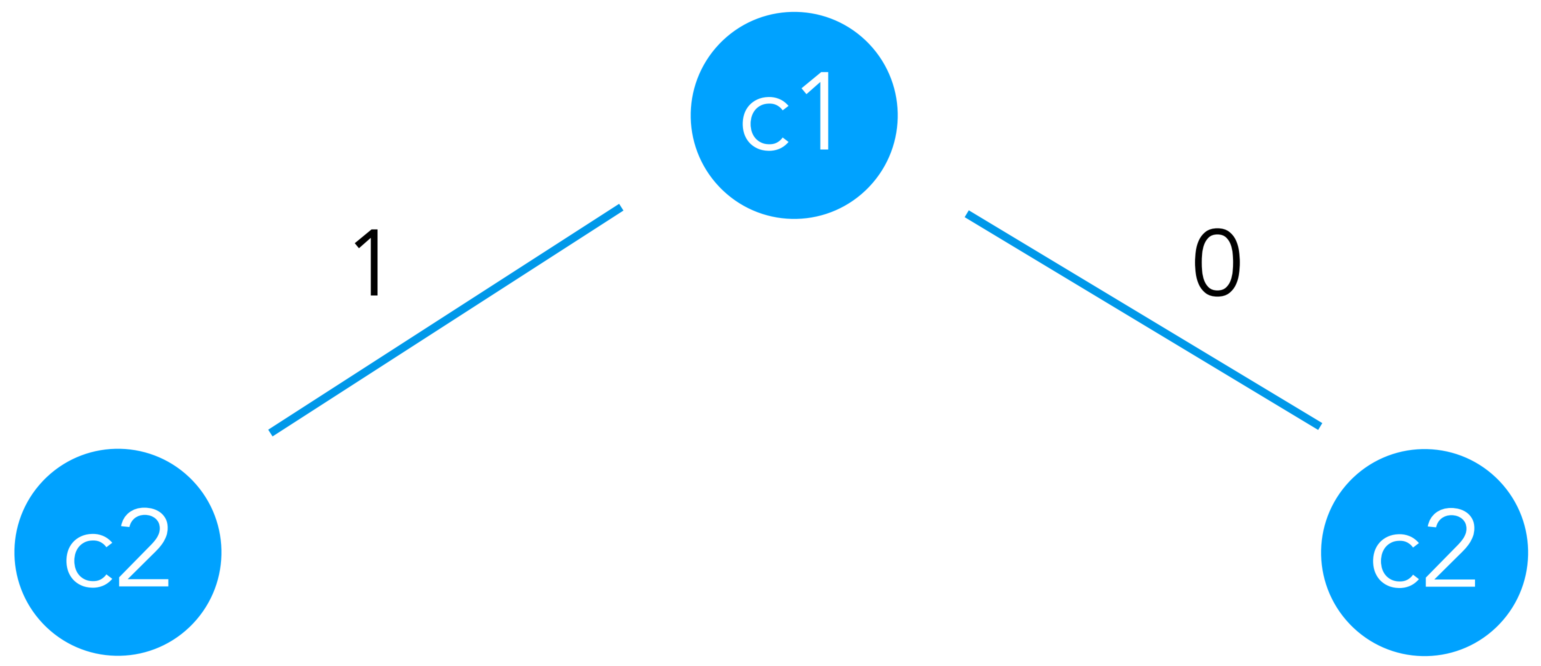
```
c1 ~ sample( Bernoulli(0.6) )
c2 ~ sample( Bernoulli(0.6) )
c3 ~ sample( Bernoulli(0.6) )

return c1 + c2 + c3
```

Probabilistic inference: Enumeration

```
c1 ~ sample( Bernoulli(0.6) )
c2 ~ sample( Bernoulli(0.6) )
c3 ~ sample( Bernoulli(0.6) )

return c1 + c2 + c3
```

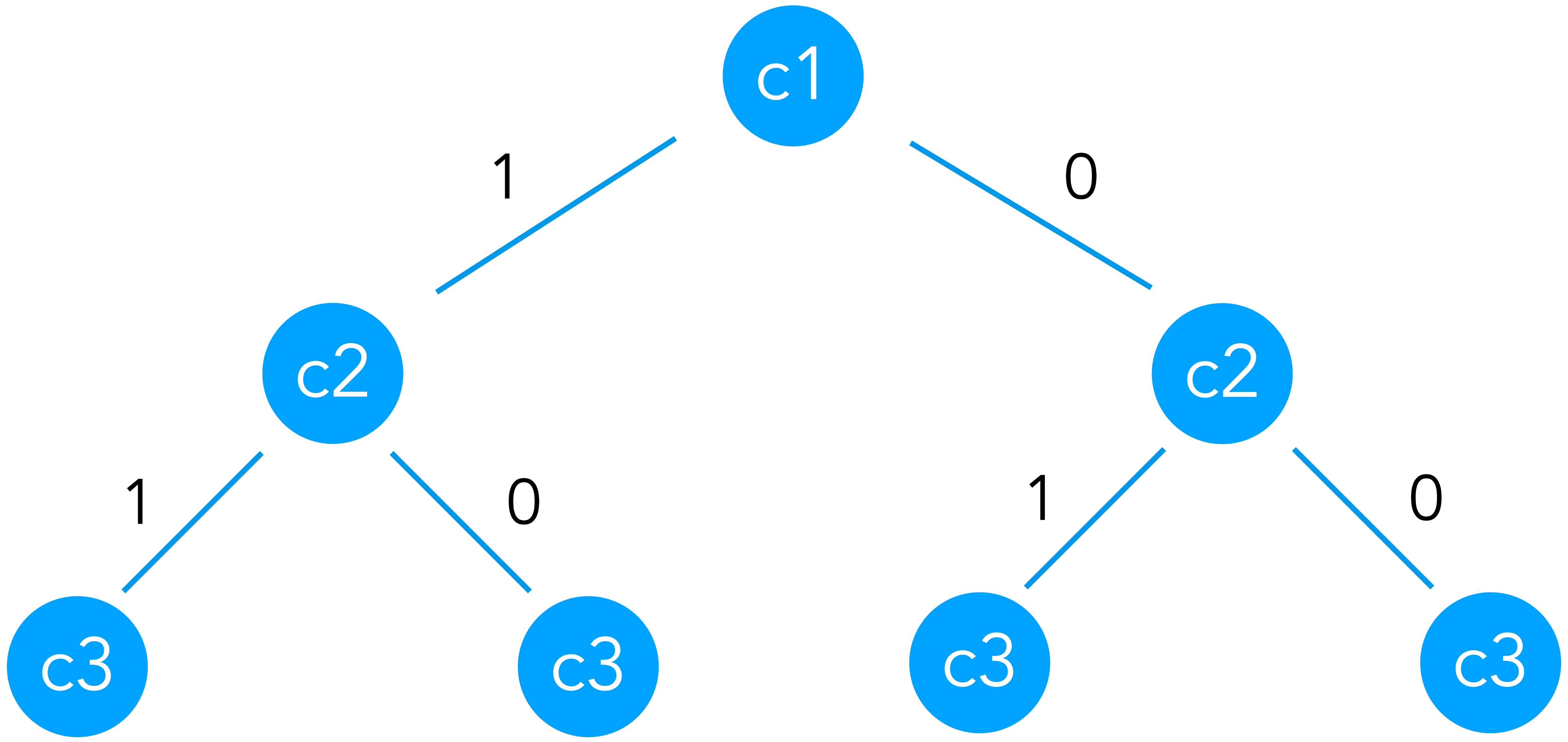
Probabilistic inference: Enumeration

```
c1 ~ sample( Bernoulli(0.6) )
c2 ~ sample( Bernoulli(0.6) )
c3 ~ sample( Bernoulli(0.6) )

return c1 + c2 + c3
```



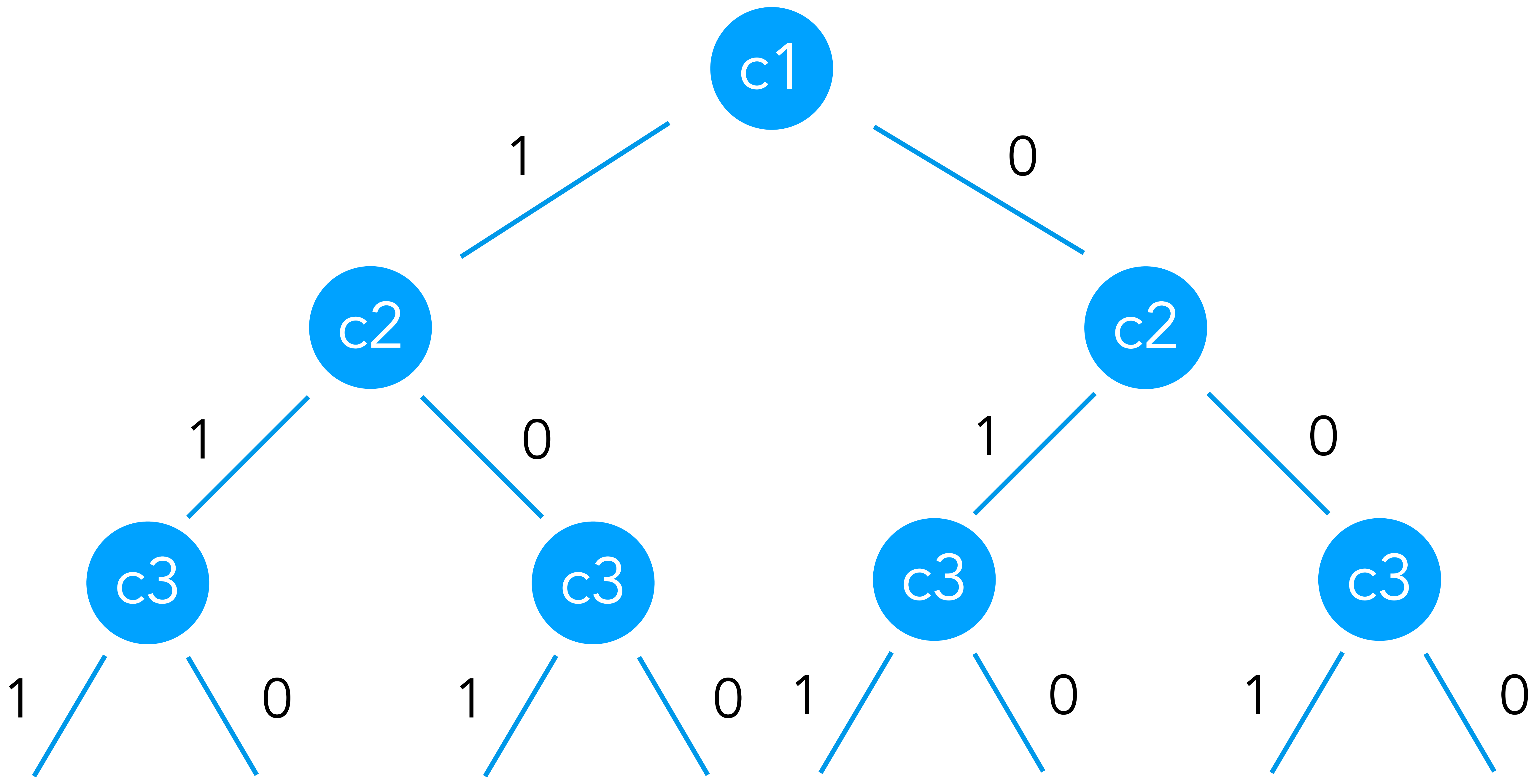
Probabilistic inference: Enumeration

```
c1 ~ sample( Bernoulli(0.6) )
c2 ~ sample( Bernoulli(0.6) )
c3 ~ sample( Bernoulli(0.6) )

return c1 + c2 + c3
```



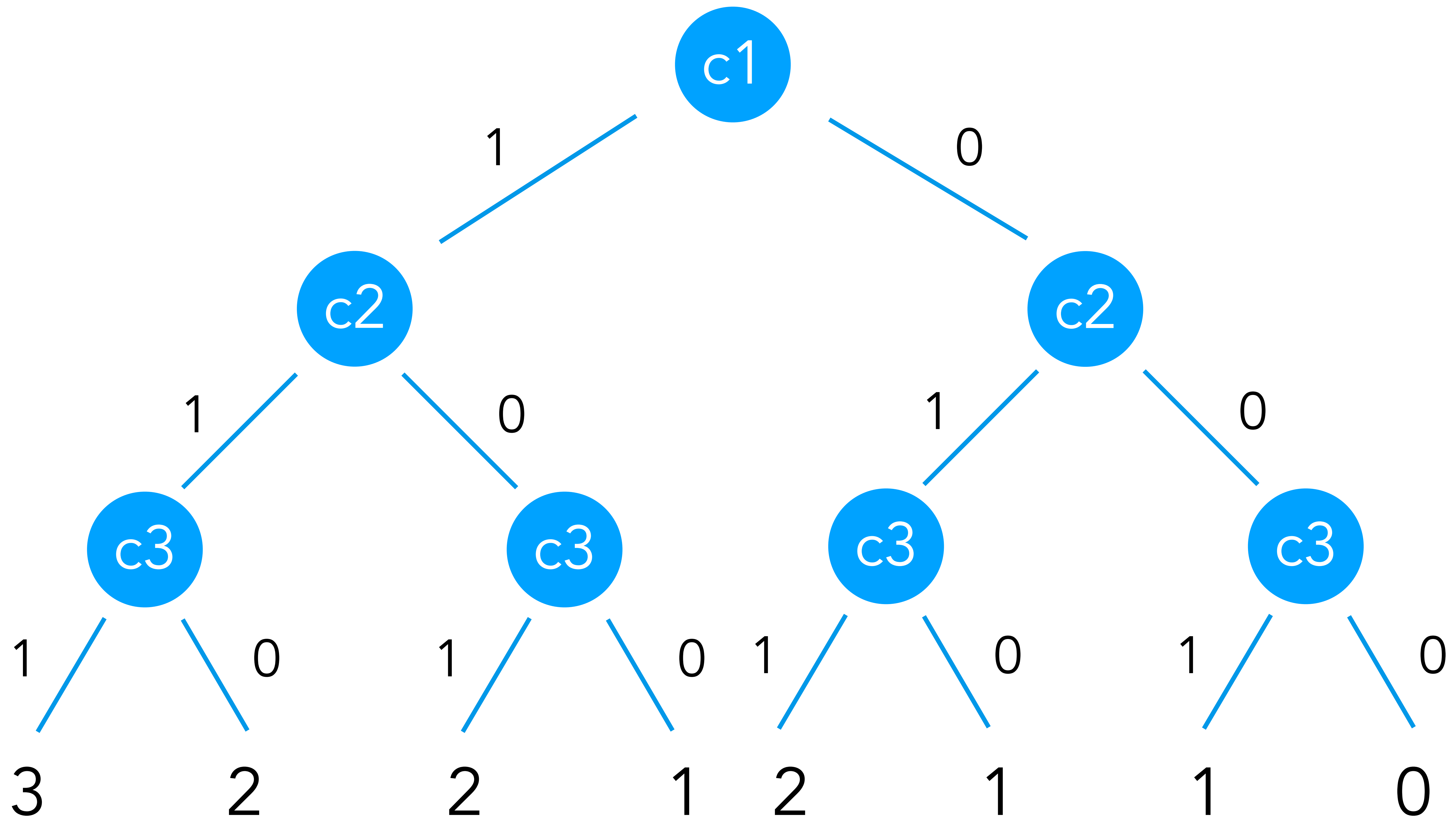
Probabilistic inference: Enumeration

```
c1 ~ sample( Bernoulli(0.6) )
c2 ~ sample( Bernoulli(0.6) )
c3 ~ sample( Bernoulli(0.6) )

return c1 + c2 + c3
```



Probabilistic inference: Enumeration

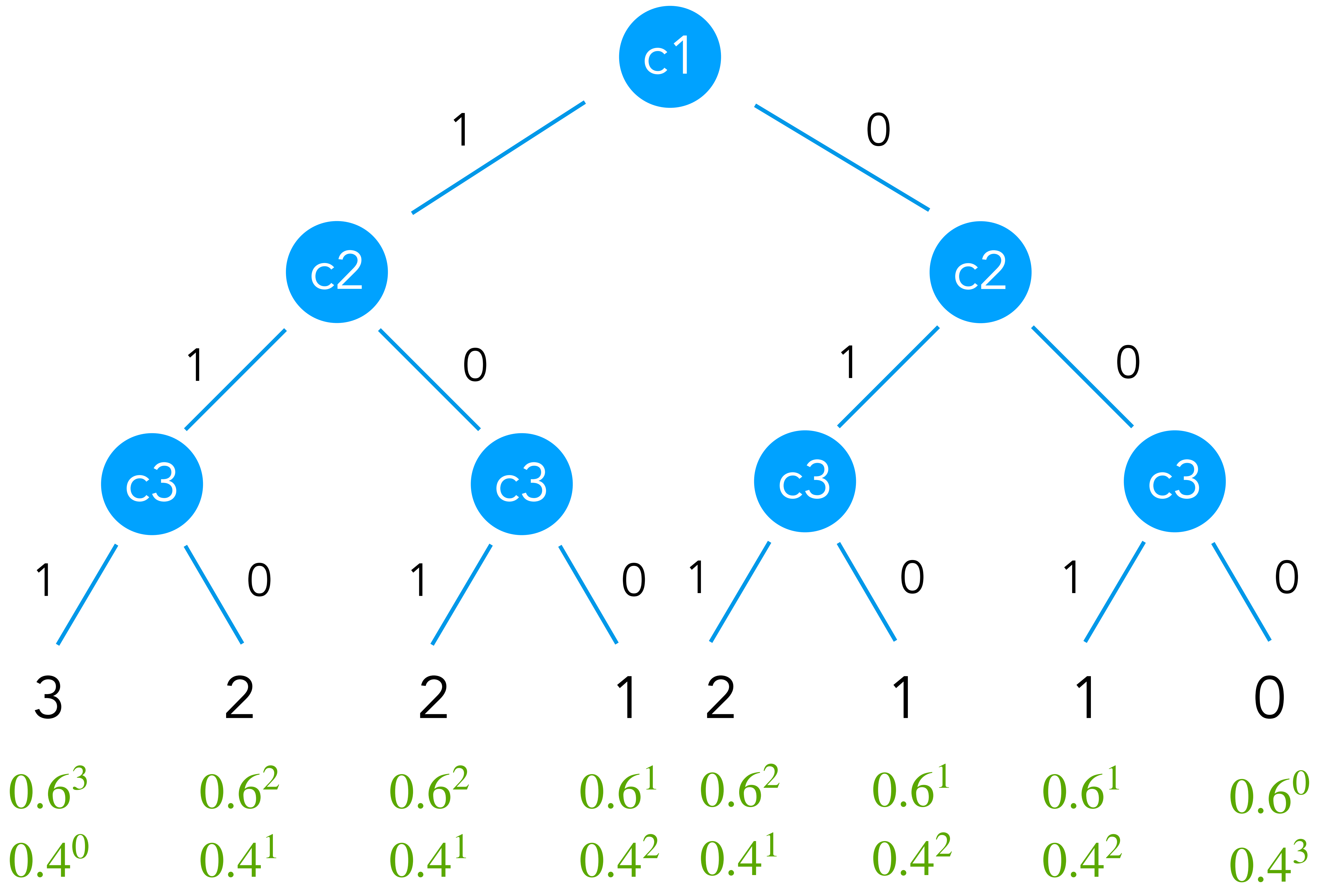
```
c1 ~ sample( Bernoulli(0.6) )
c2 ~ sample( Bernoulli(0.6) )
c3 ~ sample( Bernoulli(0.6) )

return c1 + c2 + c3
```



Probabilistic inference: Enumeration

c1 ~ sample(Bernoulli(0.6))
c2 ~ sample(Bernoulli(0.6))
c3 ~ sample(Bernoulli(0.6))

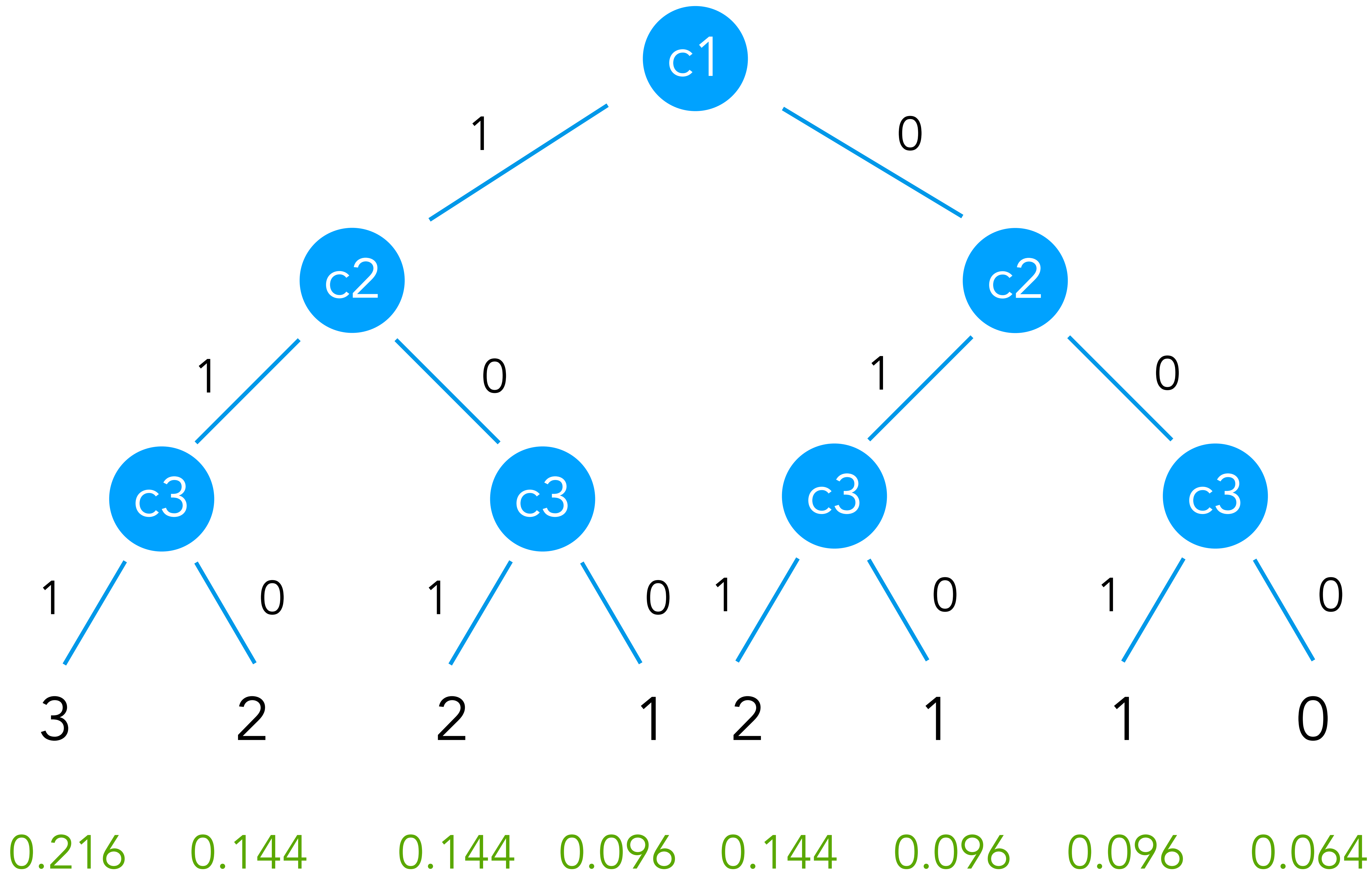
return c1 + c2 + c3



Probabilistic inference: Enumeration

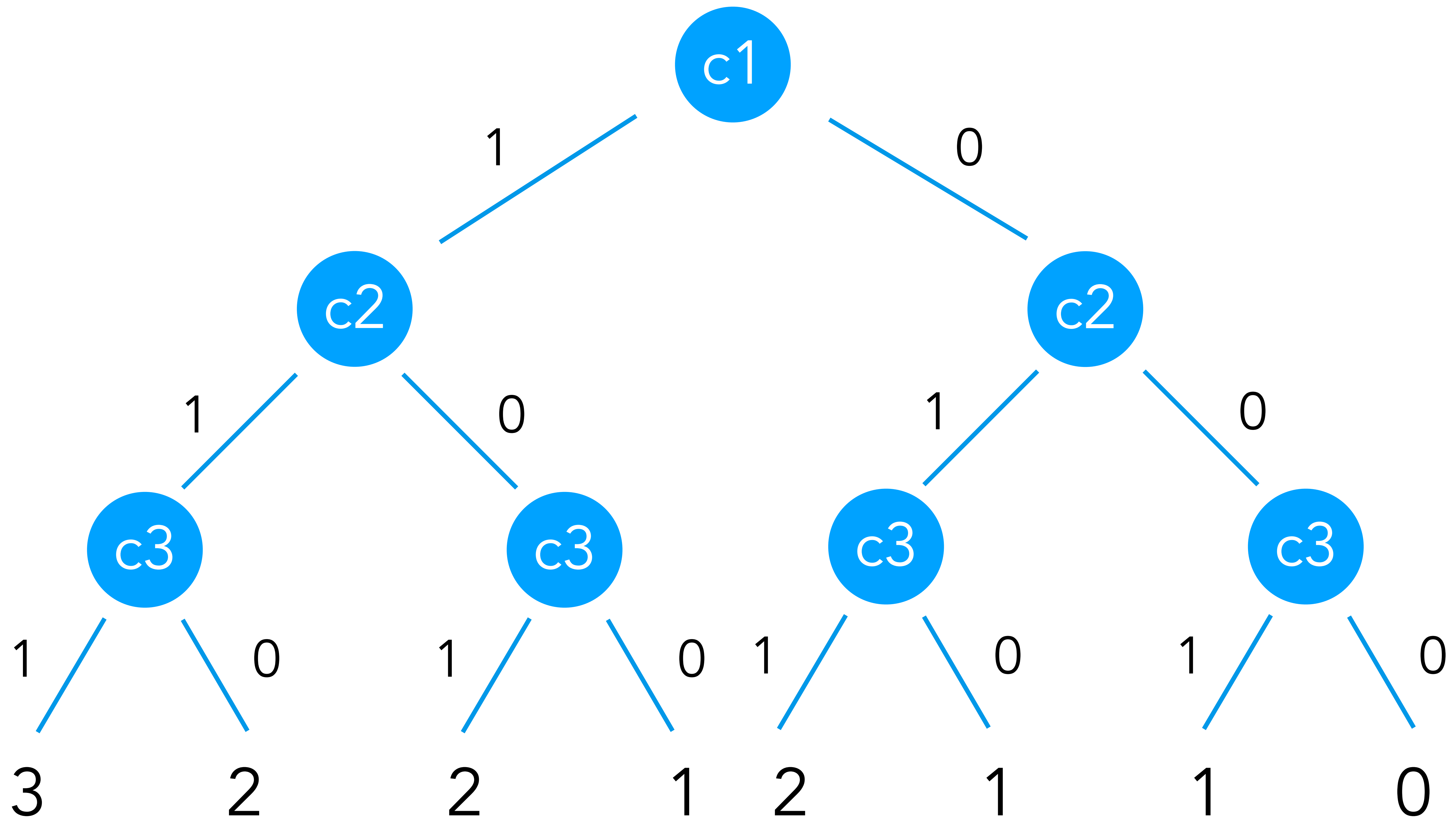
c1 ~ sample(Bernoulli(0.6))
c2 ~ sample(Bernoulli(0.6))
c3 ~ sample(Bernoulli(0.6))

return c1 + c2 + c3



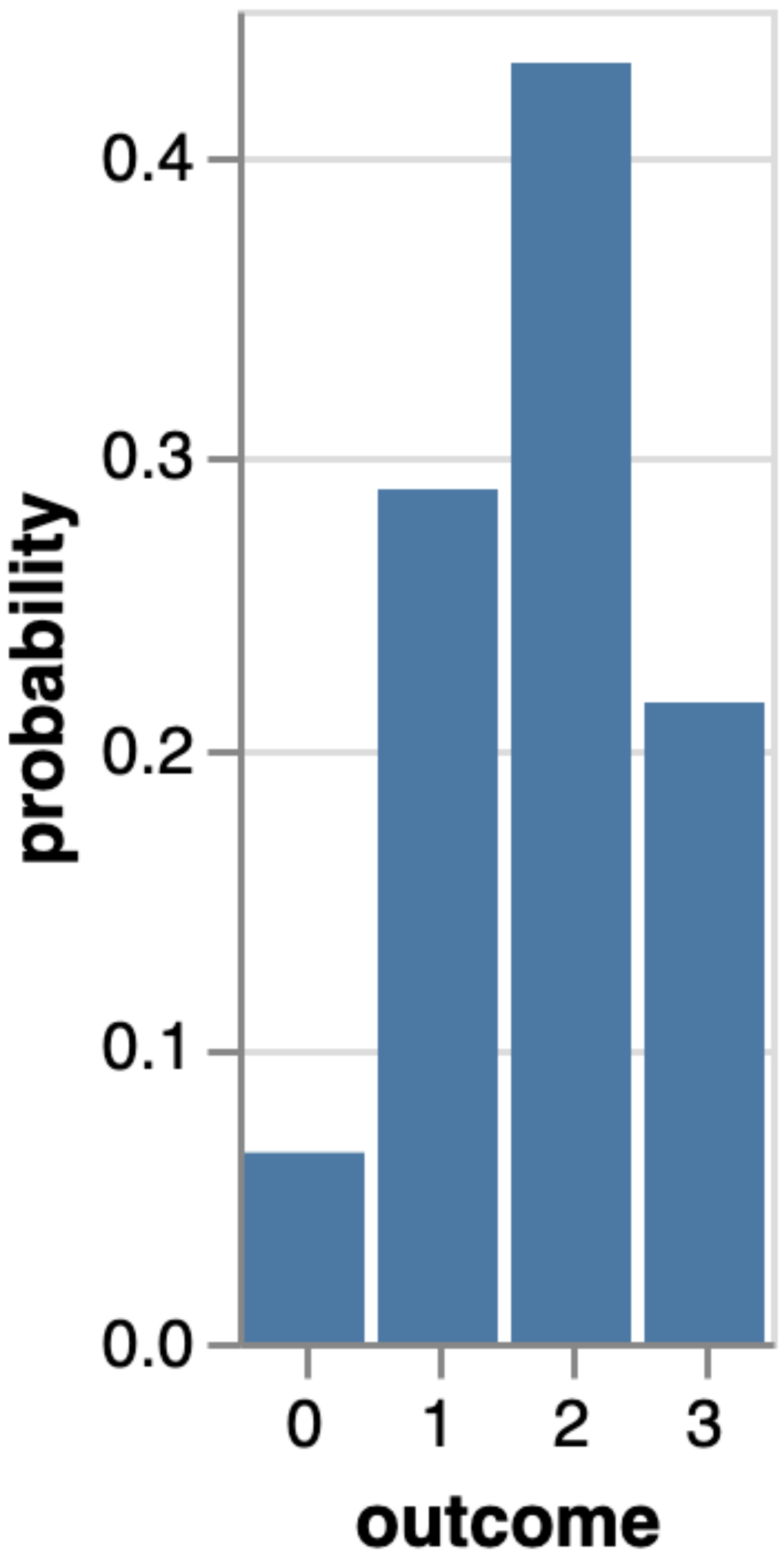
Probabilistic inference: Enumeration

```
c1 ~ sample( Bernoulli(0.6) )
c2 ~ sample( Bernoulli(0.6) )
c3 ~ sample( Bernoulli(0.6) )

return c1 + c2 + c3
```

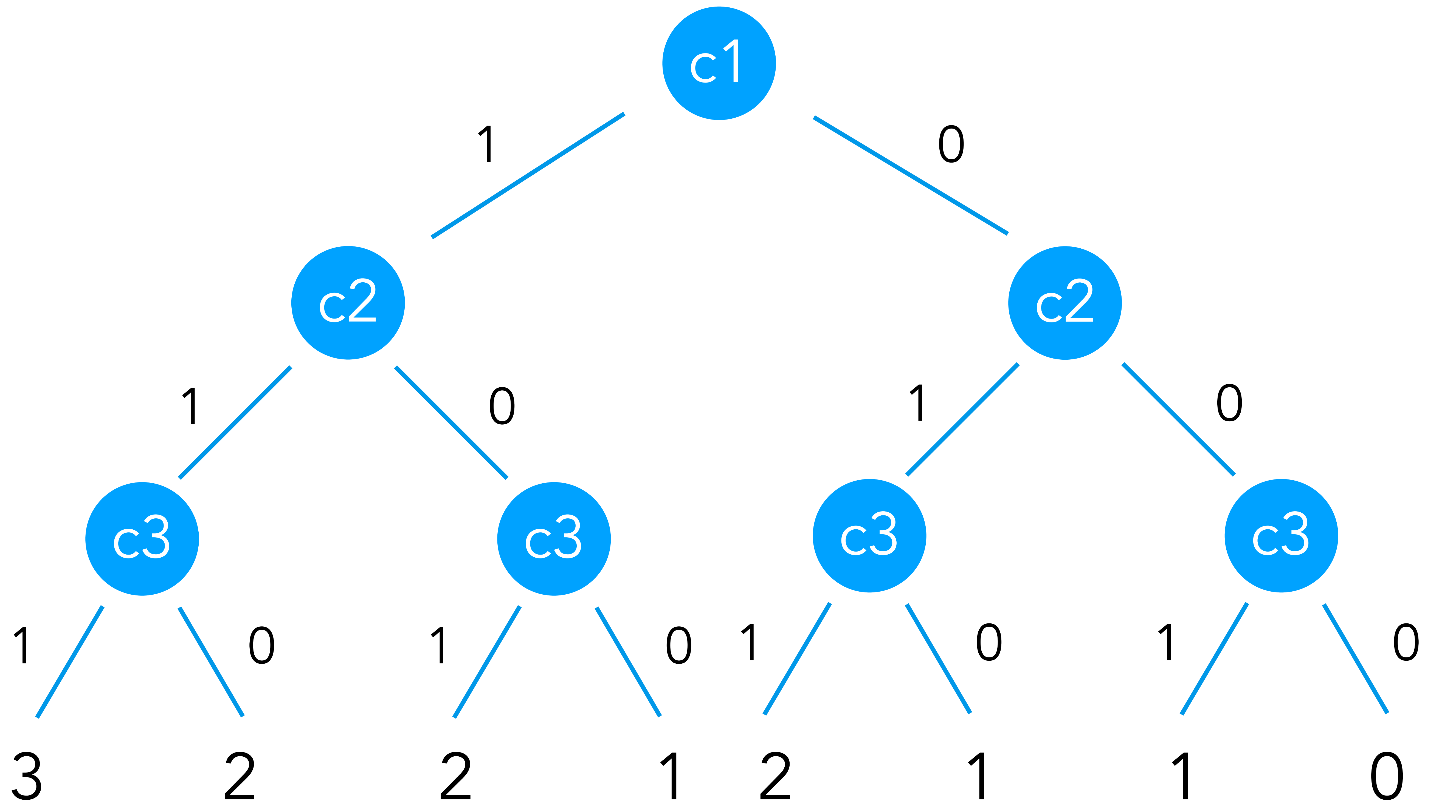


1.0 = 0.216 0.144 0.144 0.096 0.144 0.096 0.096 0.064

Probabilistic inference: Enumeration



noulli(0.6))
noulli(0.6))
noulli(0.6))
:3

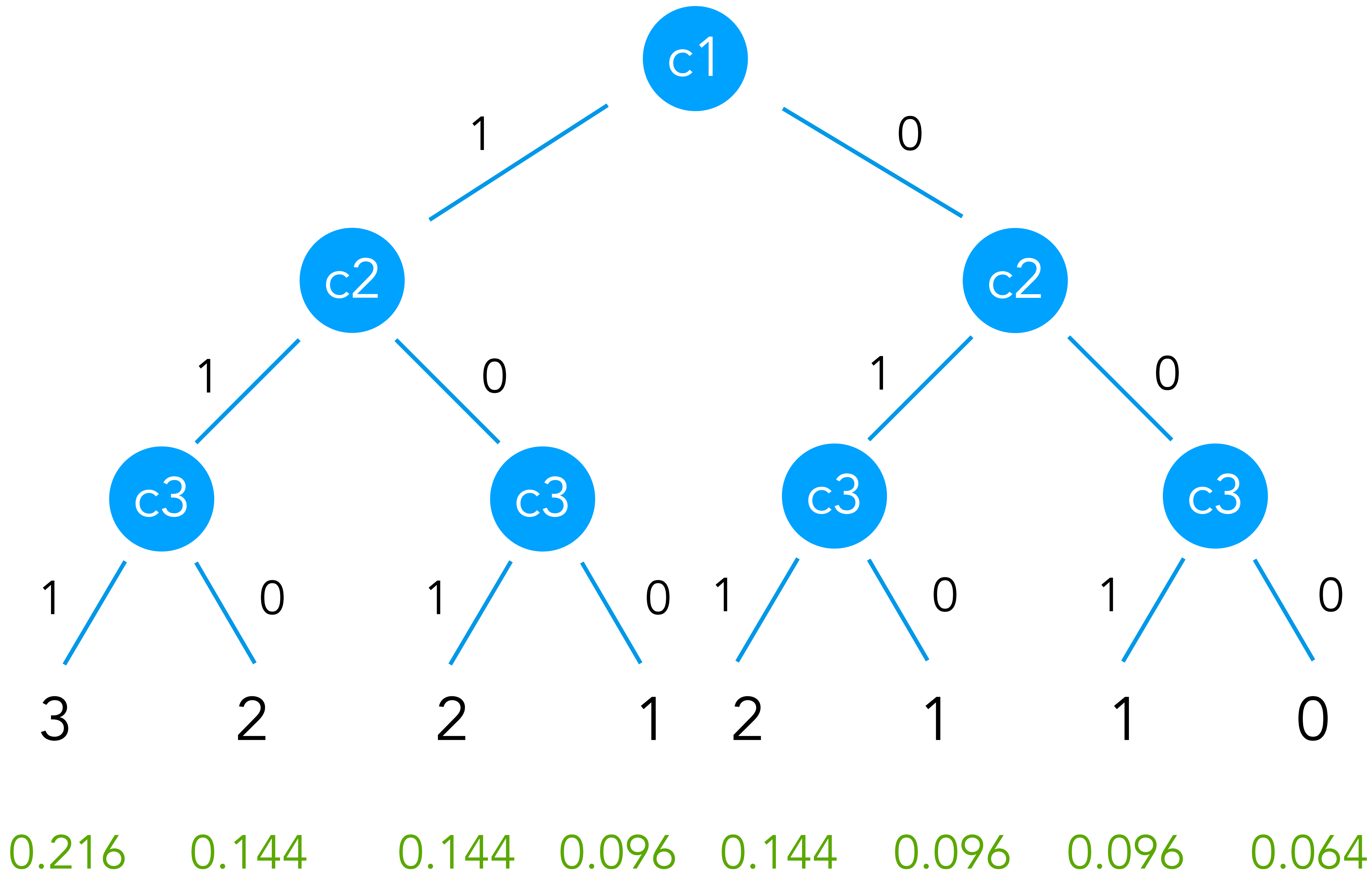


1.0 = 0.216 0.144 0.144 0.096 0.144 0.096 0.096 0.064

Probabilistic inference: Rules of inference

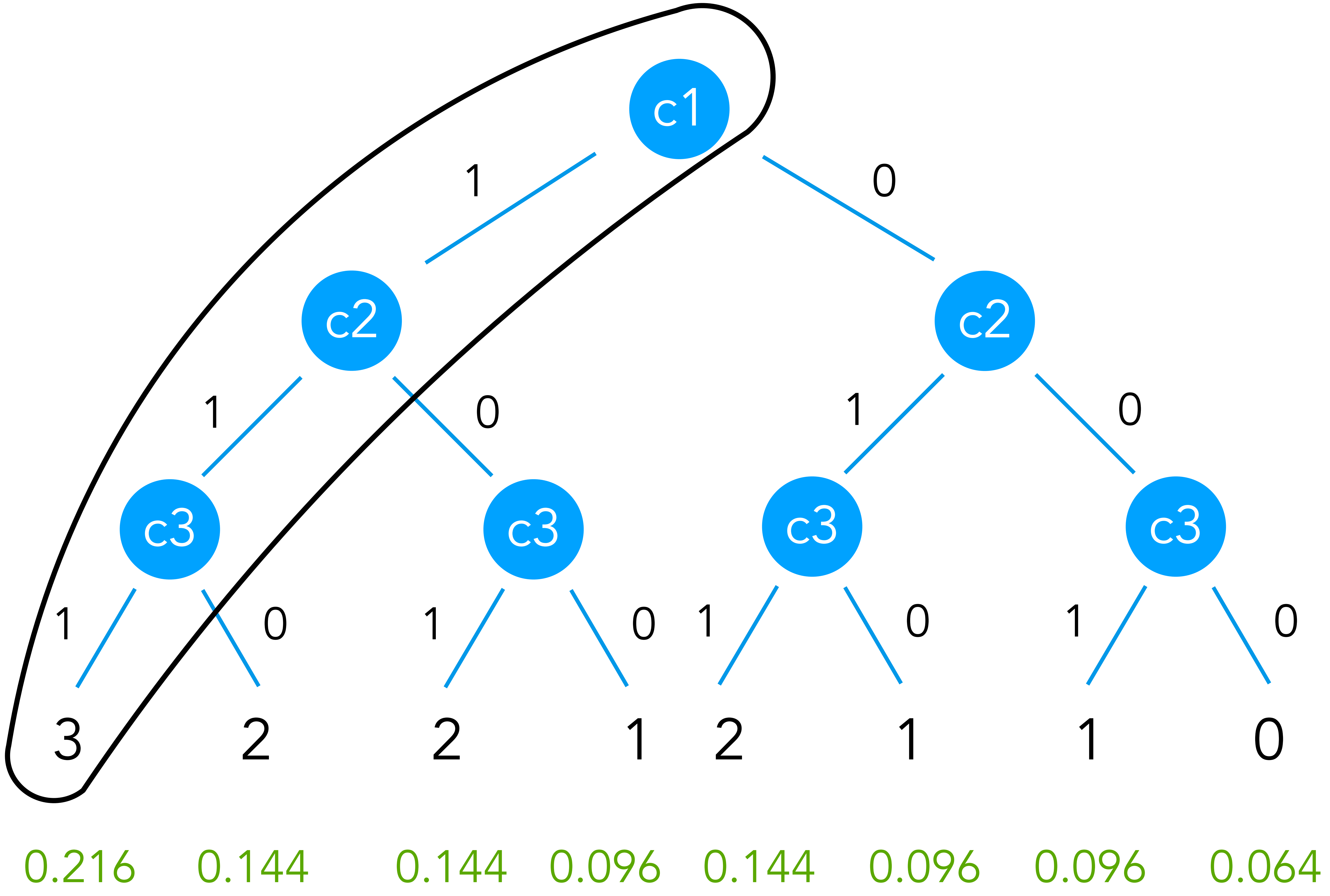
c1 ~ sample(Bernoulli(0.6))
c2 ~ sample(Bernoulli(0.6))
c3 ~ sample(Bernoulli(0.6))

return c1 + c2 + c3



Probabilistic inference: Rules of inference

Product rule:
Probs of random choices multiply



c1 ~ sample(Bernoulli(0.6))
c2 ~ sample(Bernoulli(0.6))
c3 ~ sample(Bernoulli(0.6))

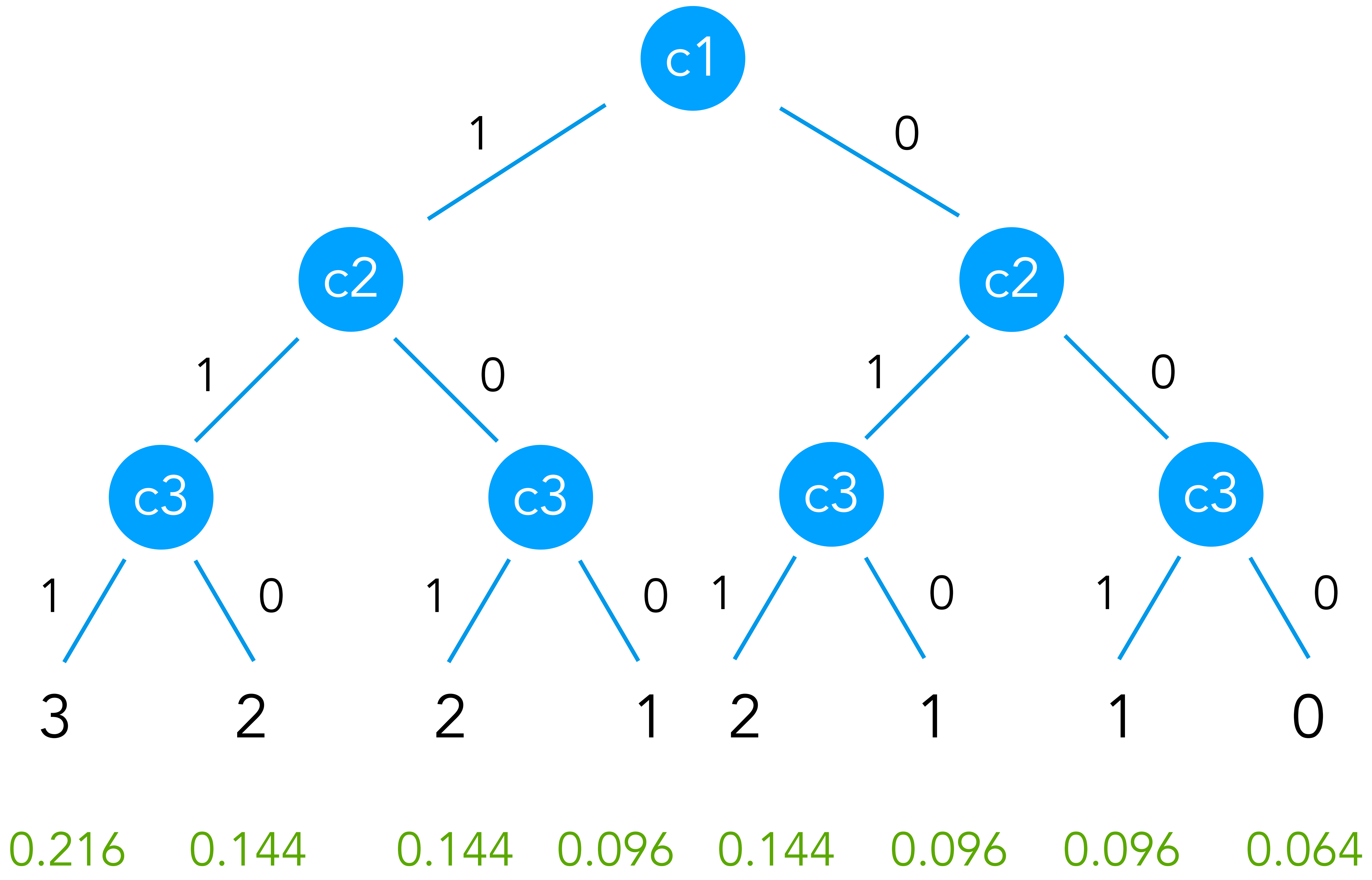
return c1 + c2 + c3

Probabilistic inference: Rules of inference

Product rule:
Probs of random choices multiply

c1 ~ sample(Bernoulli(0.6))
c2 ~ sample(Bernoulli(0.6))
c3 ~ sample(Bernoulli(0.6))

return c1 + c2 + c3



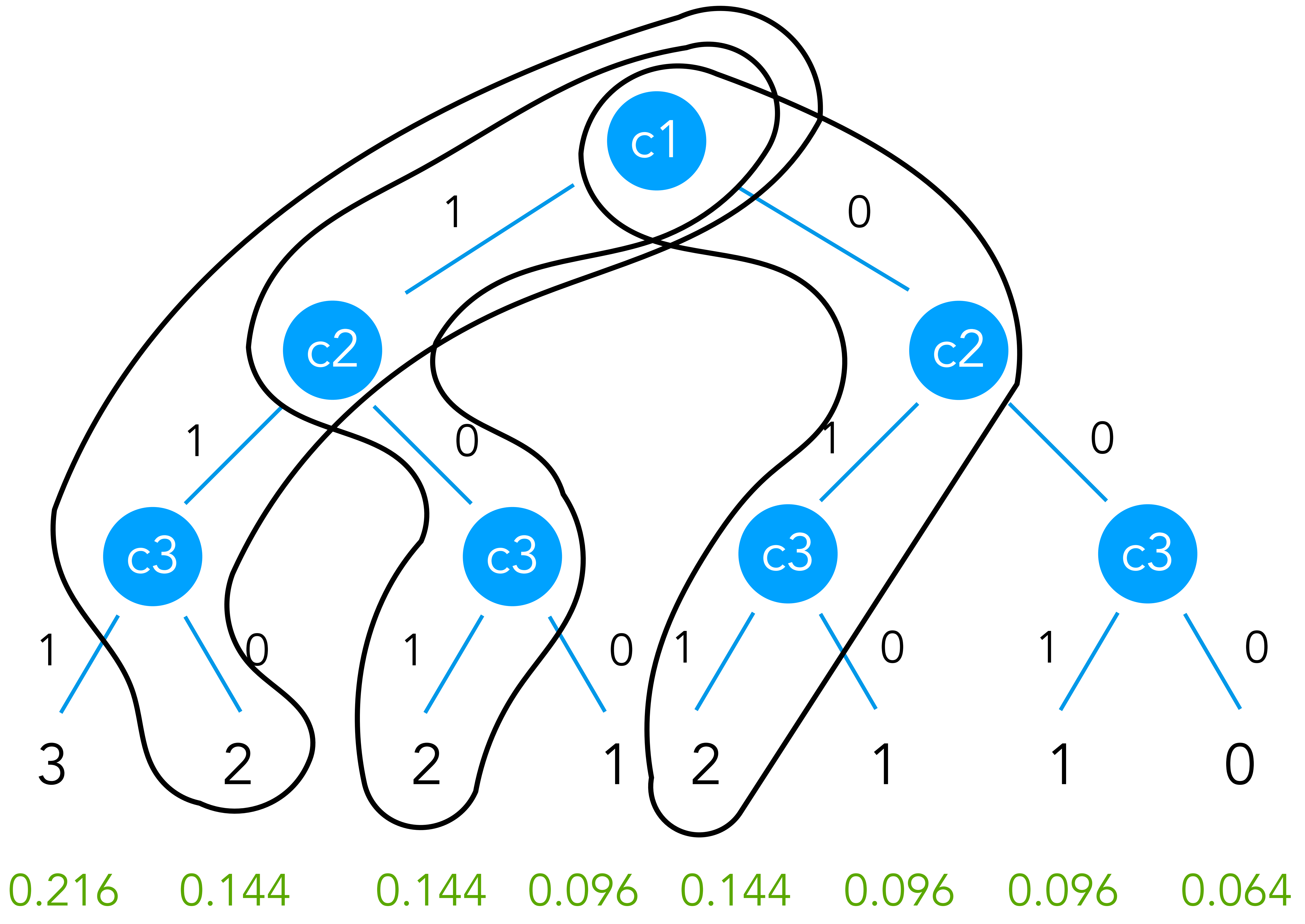
Probabilistic inference: Rules of inference

Product rule:
Probs of random choices multiply

Sum rule:
Probs of alternatives add

c1 ~ sample(Bernoulli(0.6))
c2 ~ sample(Bernoulli(0.6))
c3 ~ sample(Bernoulli(0.6))

return c1 + c2 + c3



A few exercises

```
A ~ sample( Bernoulli(0.5) )
B ~ sample( Bernoulli(0.5) )
C = [A, B]

return C
```

Probability of C = [true, false] ?

A few exercises

```
A ~ sample( Bernoulli(0.5) )
B ~ sample( A ? Bernoulli(0.3) : Bernoulli(0.7))
C = [A, B]

return C
```

Probability of C = [true, false] ?

A few exercises

```
C = sample( Bernoulli(0.5) ) || sample( Bernoulli(0.5) )

return C
```

Probability of C = true ?

Probabilistic inference: What if there are many choices?

```
c1 ~ sample( Bernoulli(0.6) )
c2 ~ sample( Bernoulli(0.6) )
c3 ~ sample( Bernoulli(0.6) )
……
c20 ~ sample( Bernoulli(0.6) )
c21 ~ sample( Bernoulli(0.6) )

return sum(c1 to c21)
```

Probabilistic inference: What if there are many choices?

```
c1 ~ sample( Bernoulli(0.6) )
c2 ~ sample( Bernoulli(0.6) )
c3 ~ sample( Bernoulli(0.6) )
……
c20 ~ sample( Bernoulli(0.6) )
c21 ~ sample( Bernoulli(0.6) )

return sum(c1 to c21)
```

Too many choices to consistently explore

Probabilistic inference: What if there are many choices?

```
c1 ~ sample( Bernoulli(0.6) )
c2 ~ sample( Bernoulli(0.6) )
c3 ~ sample( Bernoulli(0.6) )
……
c20 ~ sample( Bernoulli(0.6) )
c21 ~ sample( Bernoulli(0.6) )

return sum(c1 to c21)
```

Too many choices to consistently explore

We have to approximate: find a representative subset of executions

Probabilistic inference: What if there are many choices?

```
c1 ~ sample( Bernoulli(0.6) )
c2 ~ sample( Bernoulli(0.6) )
c3 ~ sample( Bernoulli(0.6) )
……
c20 ~ sample( Bernoulli(0.6) )
c21 ~ sample( Bernoulli(0.6) )

return sum(c1 to c21)
```

Too many choices to consistently explore

We have to approximate: find a representative subset of executions

We approximate with a fixed amount of executions

Probabilistic inference: What if there are many choices?

```
c1 ~ sample( Bernoulli(0.6) )
c2 ~ sample( Bernoulli(0.6) )
c3 ~ sample( Bernoulli(0.6) )
……
c20 ~ sample( Bernoulli(0.6) )
c21 ~ sample( Bernoulli(0.6) )

return sum(c1 to c21)
```

Too many choices to consistently explore

We have to approximate: find a representative subset of executions

We approximate with a fixed amount of executions

We don't 'care equally about all executions:
We care about likely outcome more

Probabilistic inference: What if there are many choices?

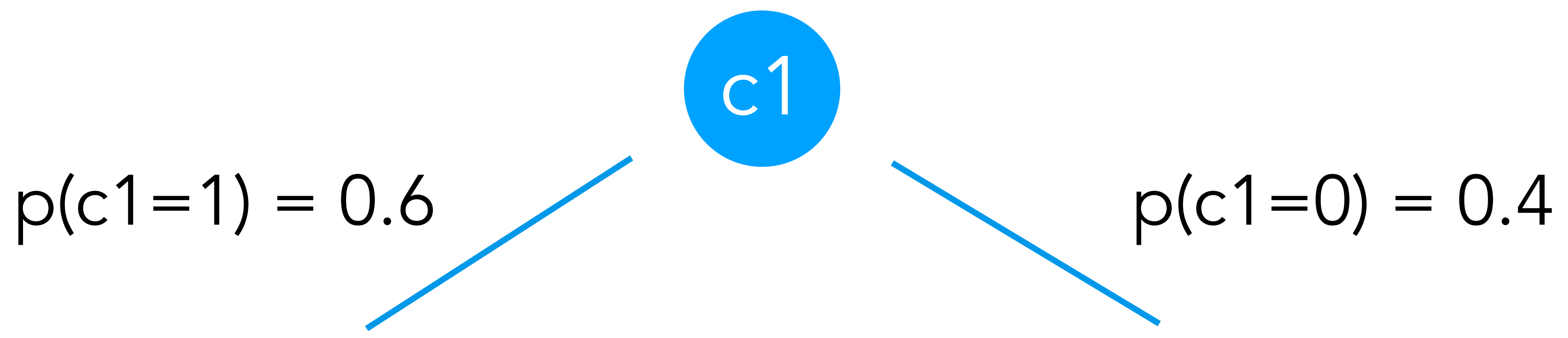
Strategy: order (partial!) executions according to the probabilities of choices

Probabilistic inference: What if there are many choices?

Strategy: order (partial!) executions according to the probabilities of choices

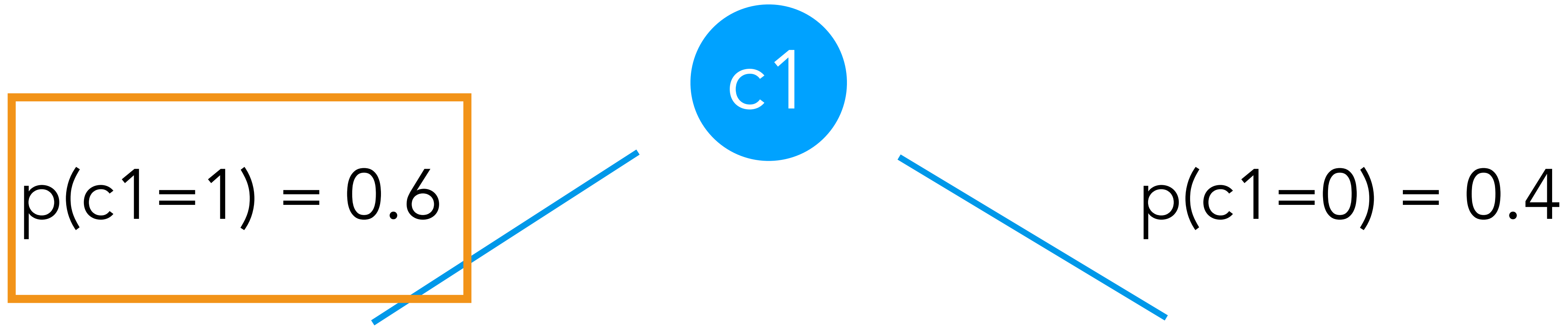
Probabilistic inference: What if there are many choices?

Strategy: order (partial!) executions according to the probabilities of choices



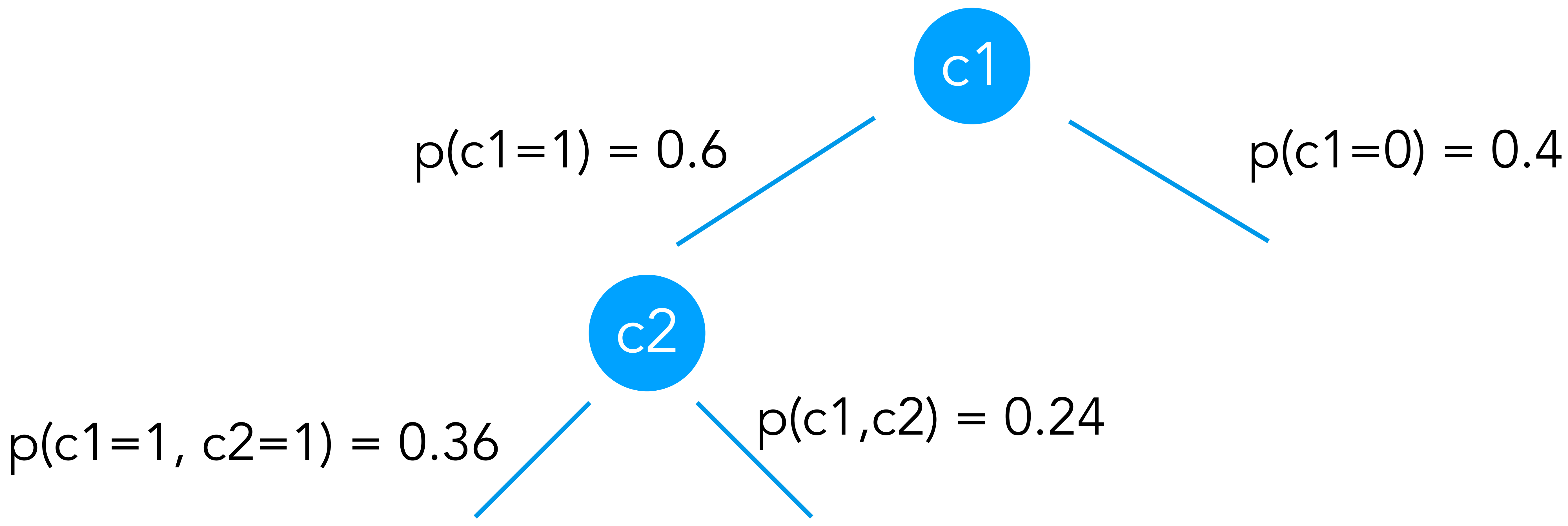
Probabilistic inference: What if there are many choices?

Strategy: order (partial!) executions according to the probabilities of choices



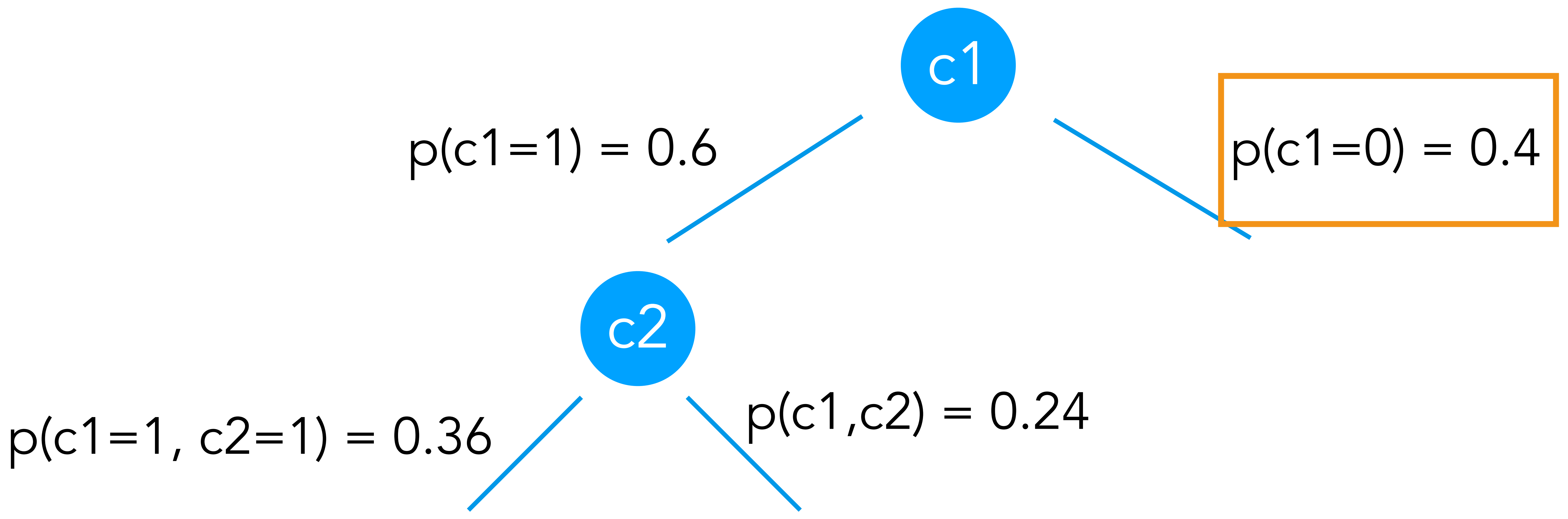
Probabilistic inference: What if there are many choices?

Strategy: order (partial!) executions according to the probabilities of choices



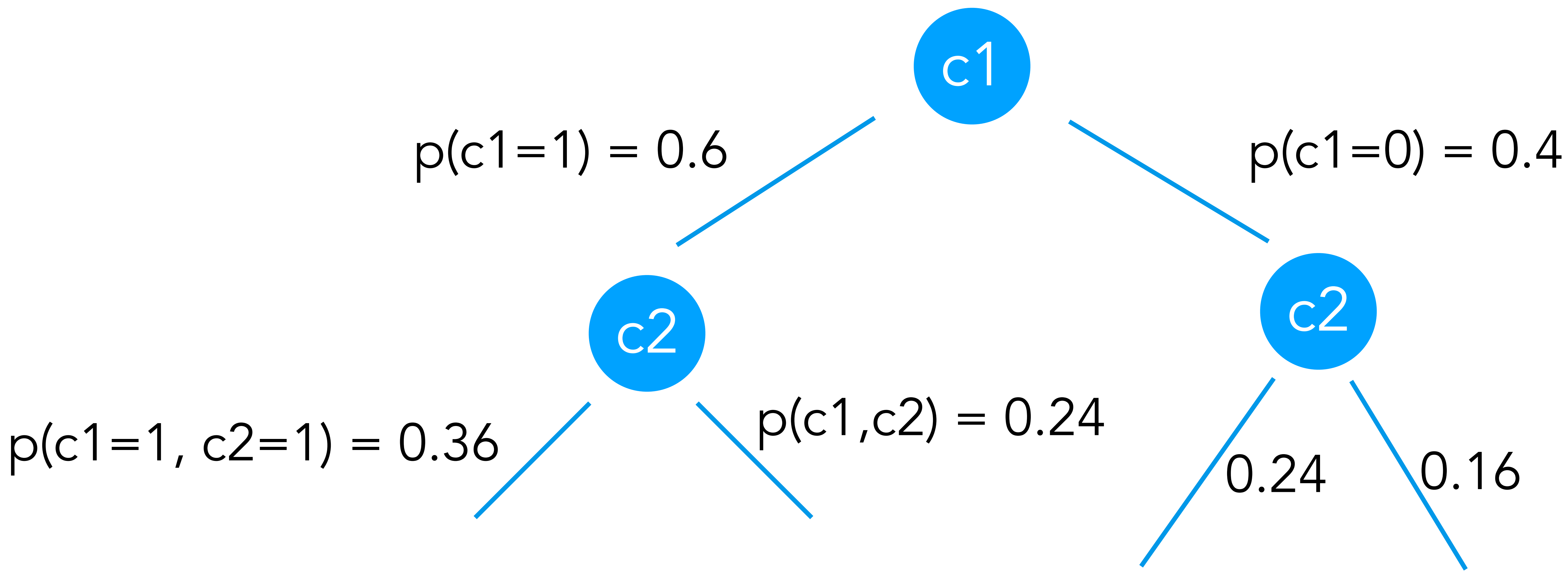
Probabilistic inference: What if there are many choices?

Strategy: order (partial!) executions according to the probabilities of choices



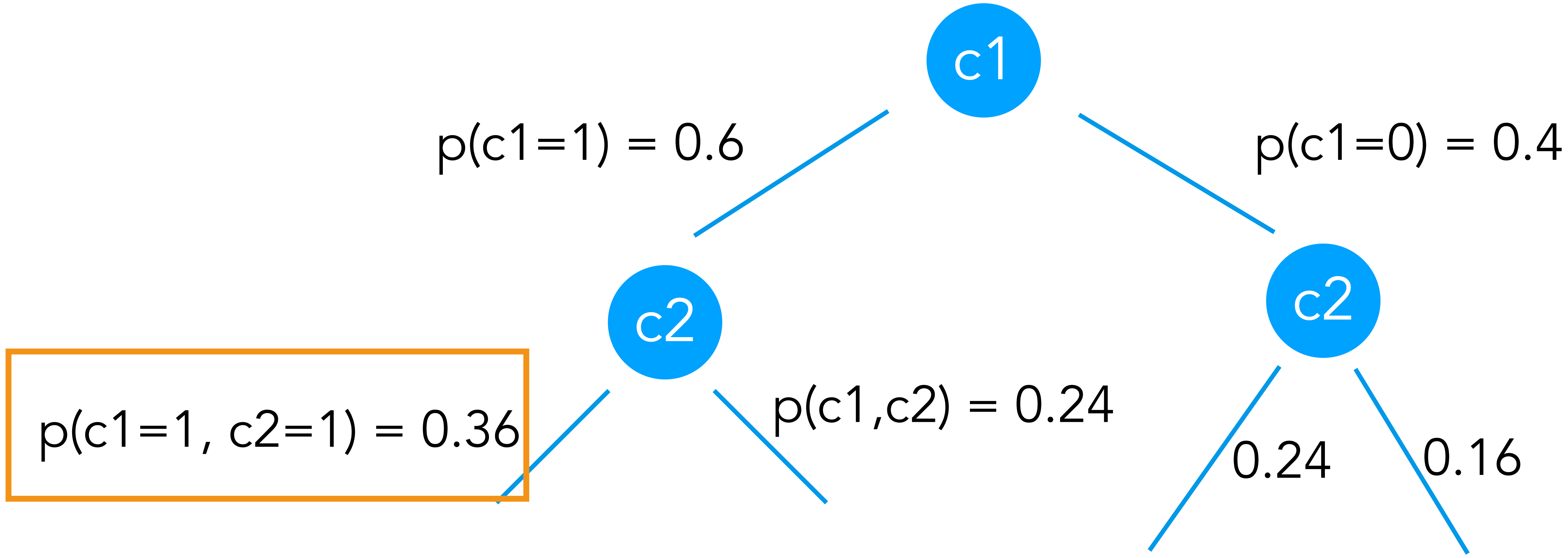
Probabilistic inference: What if there are many choices?

Strategy: order (partial!) executions according to the probabilities of choices



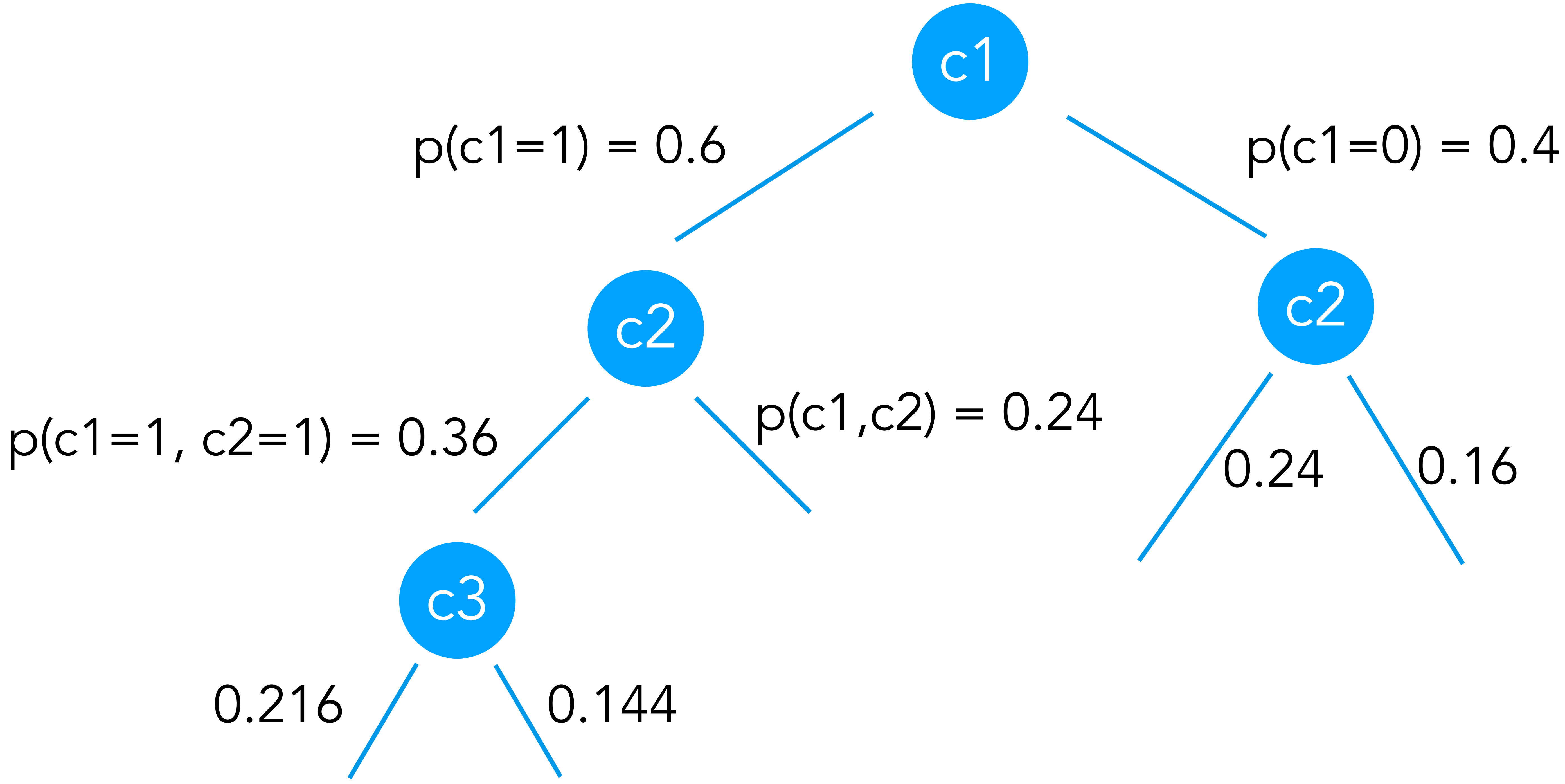
Probabilistic inference: What if there are many choices?

Strategy: order (partial!) executions according to the probabilities of choices



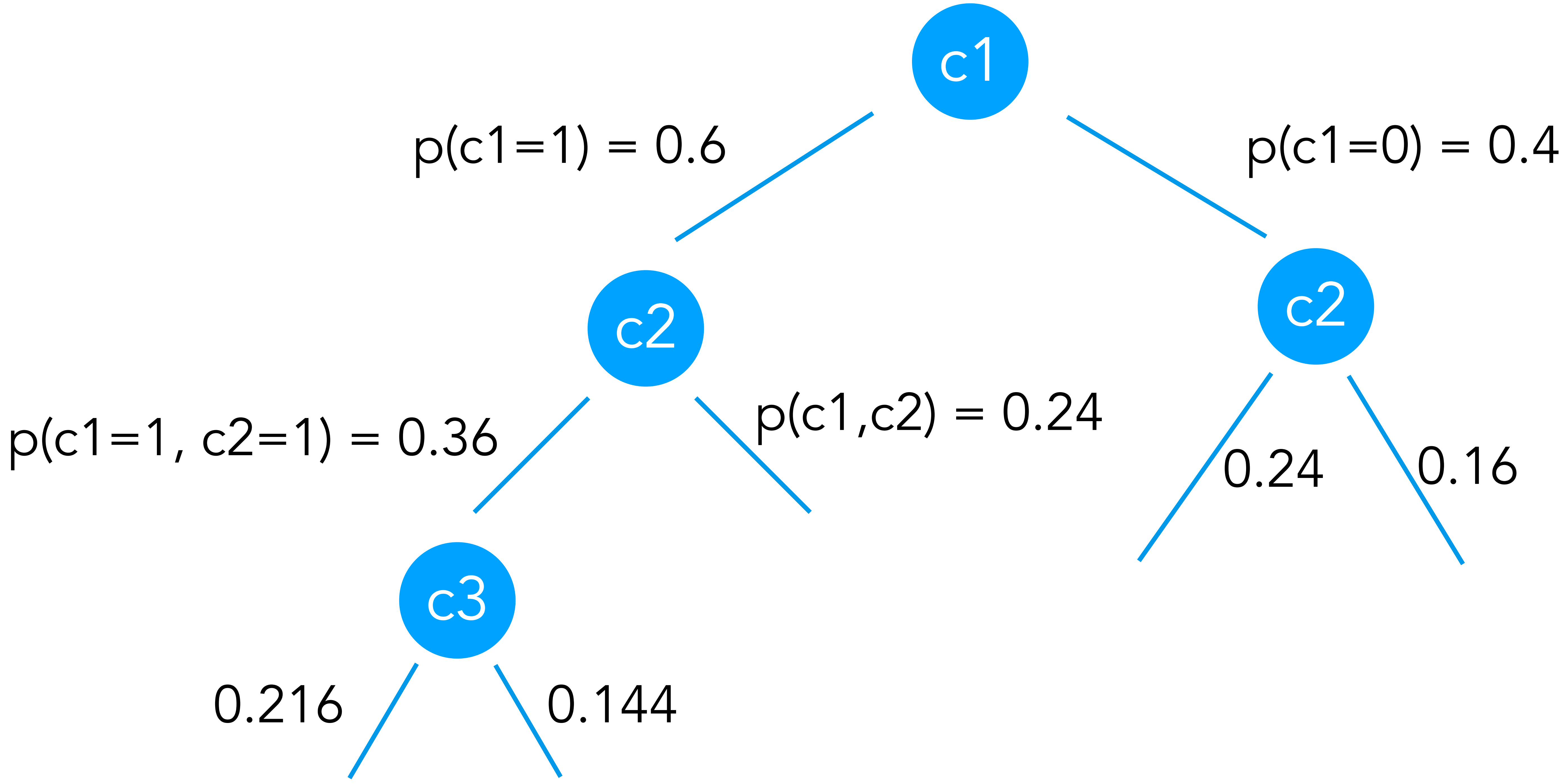
Probabilistic inference: What if there are many choices?

Strategy: order (partial!) executions according to the probabilities of choices



Probabilistic inference: What if there are many choices?

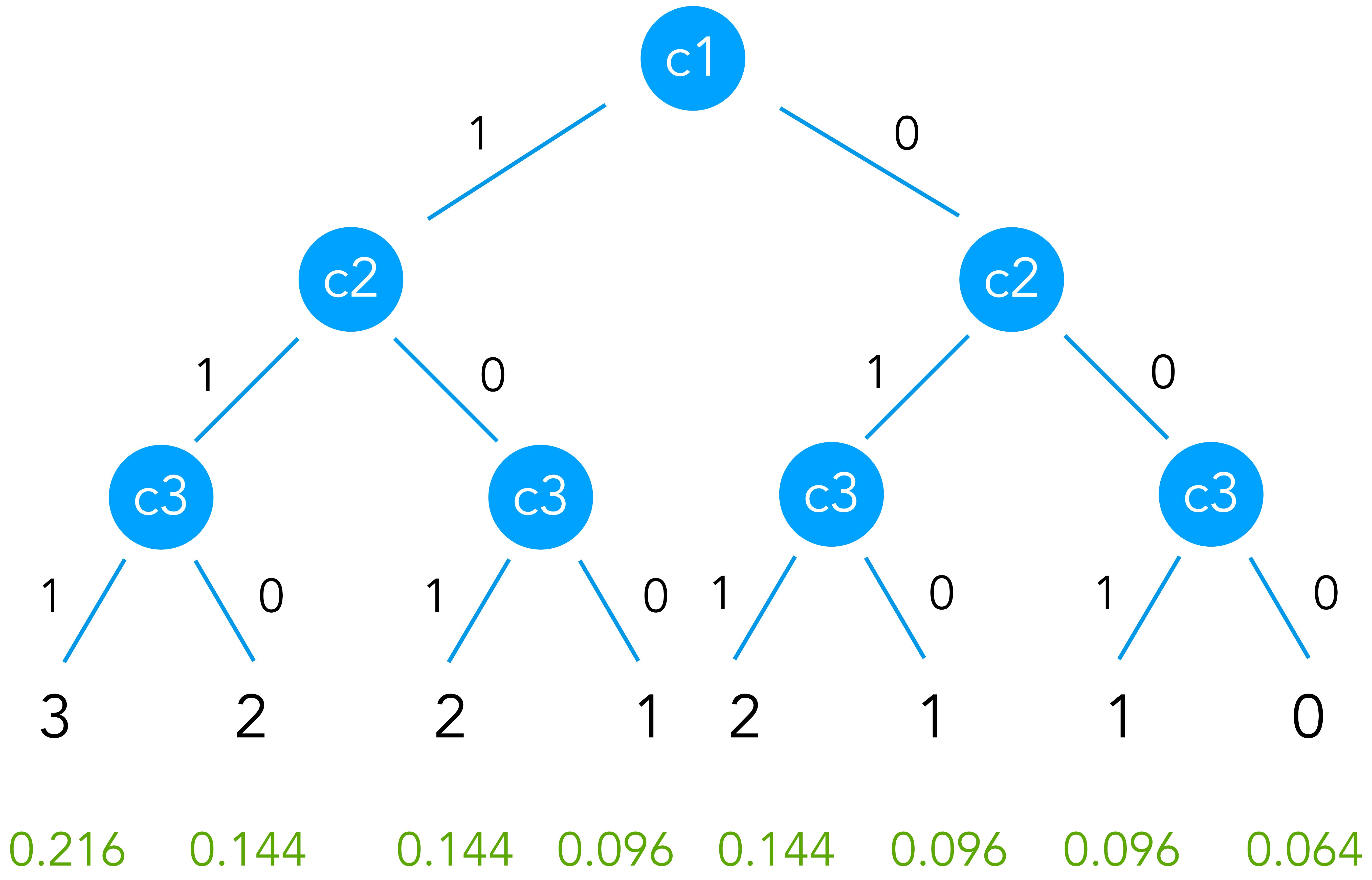
Strategy: order (partial!) executions according to the probabilities of choices



Continue until we collect K executions then normalise

Conditioning:

```
c1 ~ sample( Bernoulli(0.6) )
c2 ~ sample( Bernoulli(0.6) )
c3 ~ sample( Bernoulli(0.6) )

observe(c2 == 1)

return c1 + c2 + c3
```

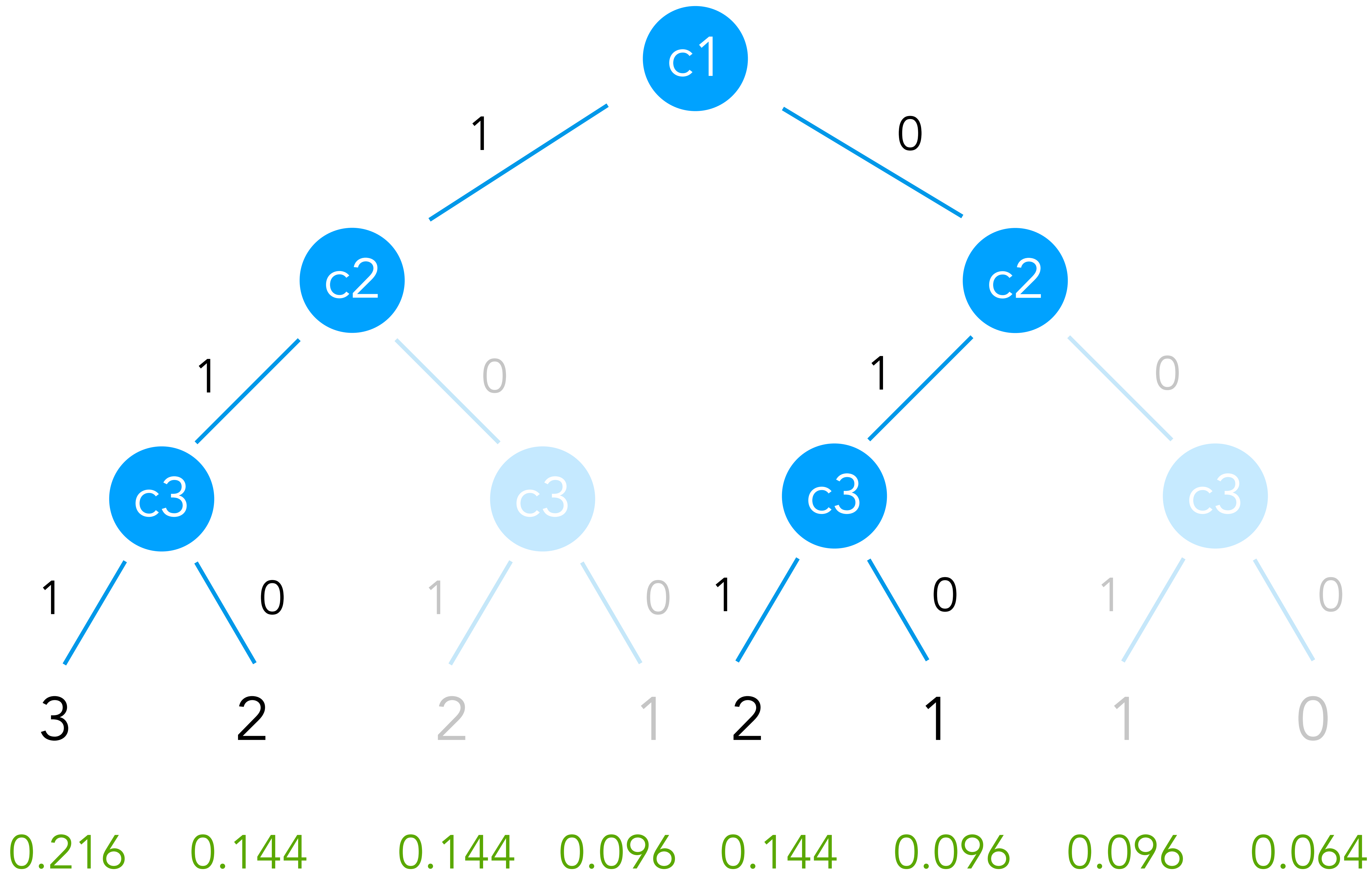


Conditioning: reject violating executions

c1 ~ sample(Bernoulli(0.6))
c2 ~ sample(Bernoulli(0.6))
c3 ~ sample(Bernoulli(0.6))

observe(c2 == 1)

return c1 + c2 + c3

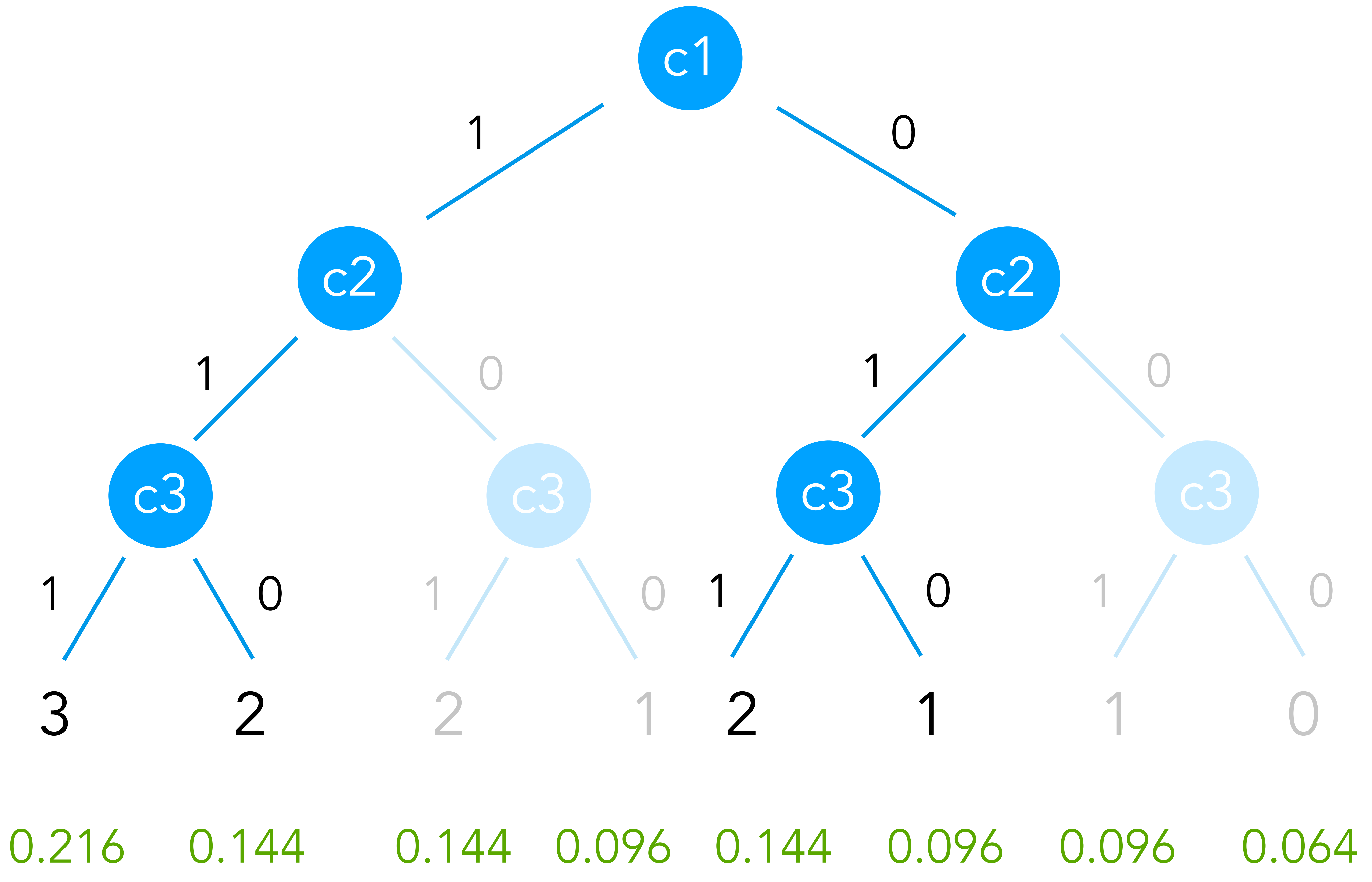


Conditioning: reject violating executions

Valid executions do not sum to 1 anymore

We need to adjust the probabilities according to the Bayes theorem

$$P(A = a \mid B = b) = \frac{P(A = a, B = b)}{P(B = b)}$$

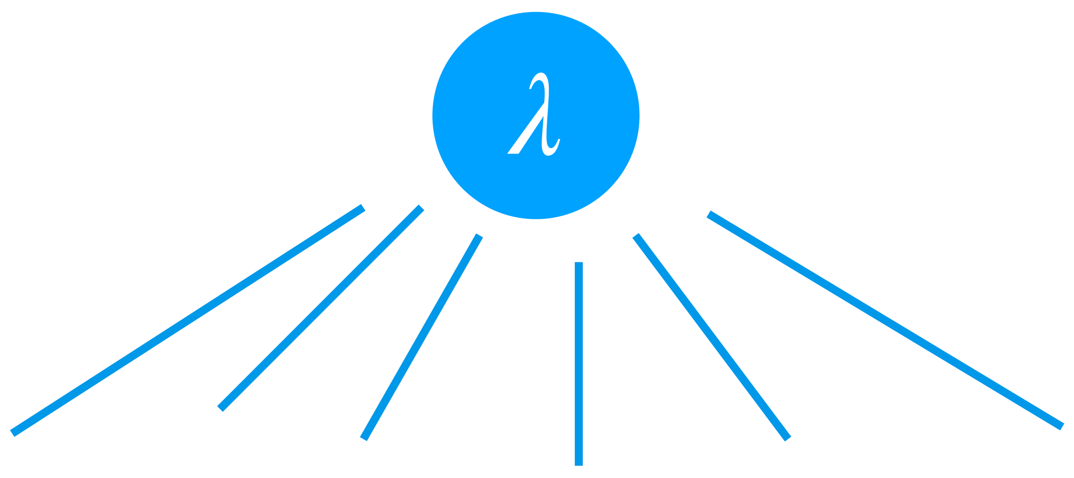


Probabilistic programs with continuous distributions

```
λ ~ sample( Normal(0.5, 1) )

c1 ~ sample( Bernoulli(λ) )
c2 ~ sample( Bernoulli(λ) )
c3 ~ sample( Bernoulli(λ) )

return c1 + c2 + c3
```

Probabilistic programs with continuous distributions

```
λ ~ sample( Normal(0.5, 1) )

c1 ~ sample( Bernoulli(λ) )
c2 ~ sample( Bernoulli(λ) )
c3 ~ sample( Bernoulli(λ) )

return c1 + c2 + c3
```

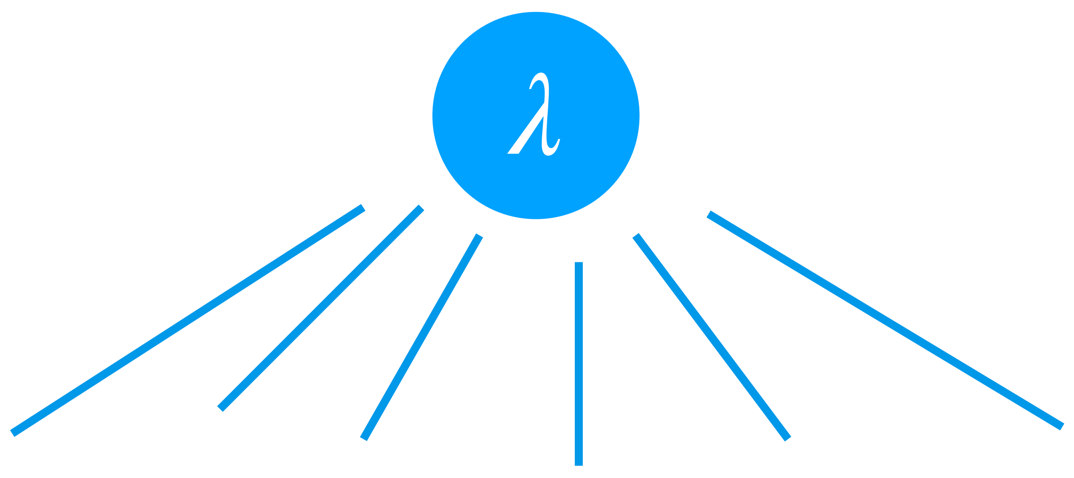


Probabilistic programs with continuous distributions

λ ~ sample(Normal(0.5, 1))

c1 ~ sample(Bernoulli(λ))
c2 ~ sample(Bernoulli(λ))
c3 ~ sample(Bernoulli(λ))

return c1 + c2 + c3



Infinite number of values

Probabilistic programs with continuous distributions

```
c1 ~ sample( Bernoulli(λ) )
c2 ~ sample( Bernoulli(λ) )

λ ~ sample( Normal(c1+c2, 0.1) )

observe(λ == 2)

return c1 + c2 + c3
```

Probabilistic programs with continuous distributions

```
c1 ~ sample( Bernoulli(λ) )
c2 ~ sample( Bernoulli(λ) )

λ ~ sample( Normal(c1+c2, 0.1) )

observe(λ == 2)

return c1 + c2 + c3
```

```
c1 ~ sample( Bernoulli(λ) )
c2 ~ sample( Bernoulli(λ) )

λ ~ sample( Normal(c1+c2, 0.1) )

observe(λ, Normal(2, 1))

return c1 + c2 + c3
```

Probabilistic programs with continuous distributions

```
c1 ~ sample( Bernoulli(λ) )
c2 ~ sample( Bernoulli(λ) )

λ ~ sample( Normal(c1+c2, 0.1) )

observe(λ == 2)

return c1 + c2 + c3
```

```
c1 ~ sample( Bernoulli(λ) )
c2 ~ sample( Bernoulli(λ) )

λ ~ sample( Normal(c1+c2, 0.1) )

observe(λ, Normal(2, 1))

return c1 + c2 + c3
```

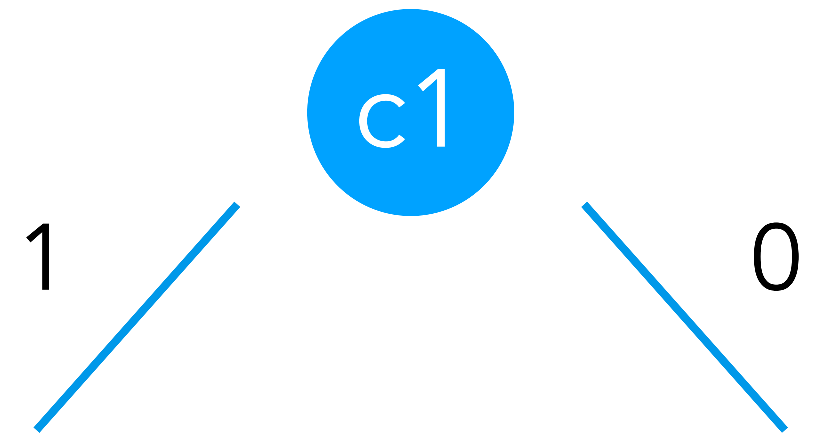
Probability that λ value is an observation from the distribution Normal(2,1)

Probabilistic programs with continuous distributions

```
c1 ~ sample( Bernoulli(0.6) )
c2 ~ sample( Bernoulli(0.4) )

λ ~ sample( Normal(c1+c2, 0.1) )

observe(λ, Normal(2, 1))

return c1 + c2 + c3
```

Probabilistic programs with continuous distributions

```
c1 ~ sample( Bernoulli(0.6) )
c2 ~ sample( Bernoulli(0.4) )

λ ~ sample( Normal(c1+c2, 0.1) )

observe(λ, Normal(2, 1))

return c1 + c2 + c3
```



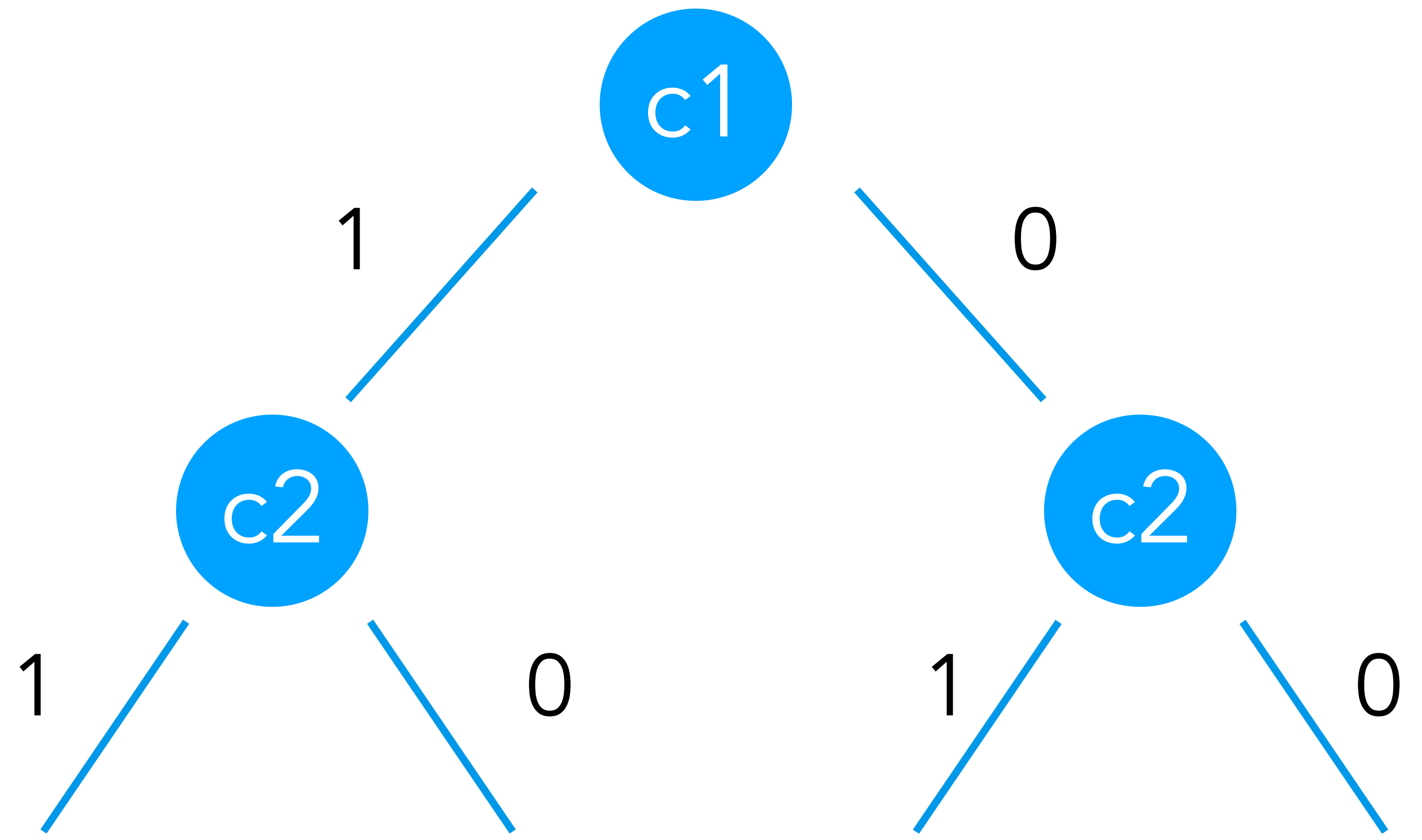
Probabilistic programs with continuous distributions

c1 ~ sample(Bernoulli(0.6))
c2 ~ sample(Bernoulli(0.4))

λ ~ sample(Normal(c1+c2, 0.1))

observe(λ, Normal(2, 1))

return c1 + c2 + c3



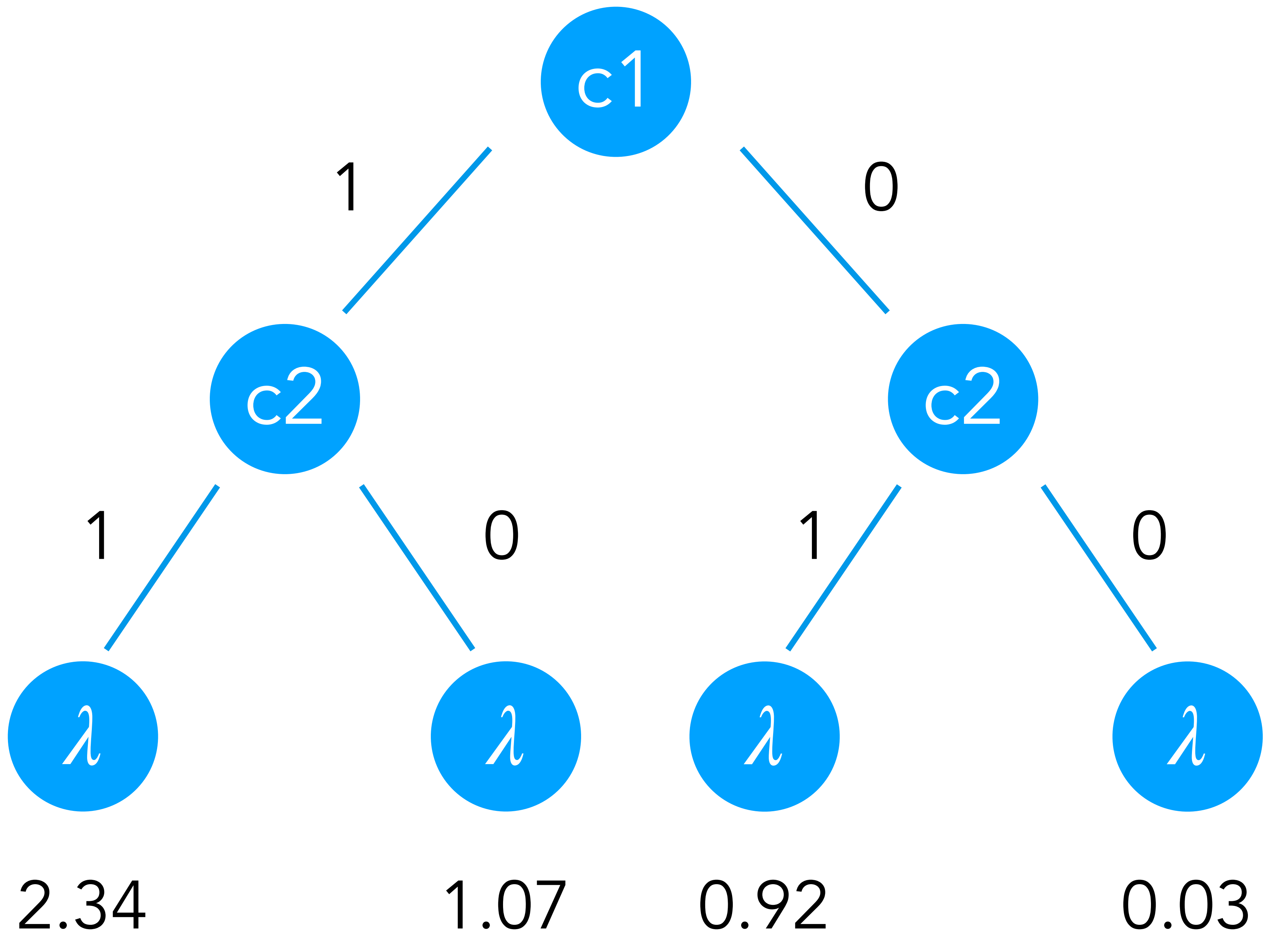
Probabilistic programs with continuous distributions

c1 ~ sample(Bernoulli(0.6))
c2 ~ sample(Bernoulli(0.4))

λ ~ sample(Normal(c1+c2, 0.1))

observe(λ, Normal(2, 1))

return c1 + c2 + c3



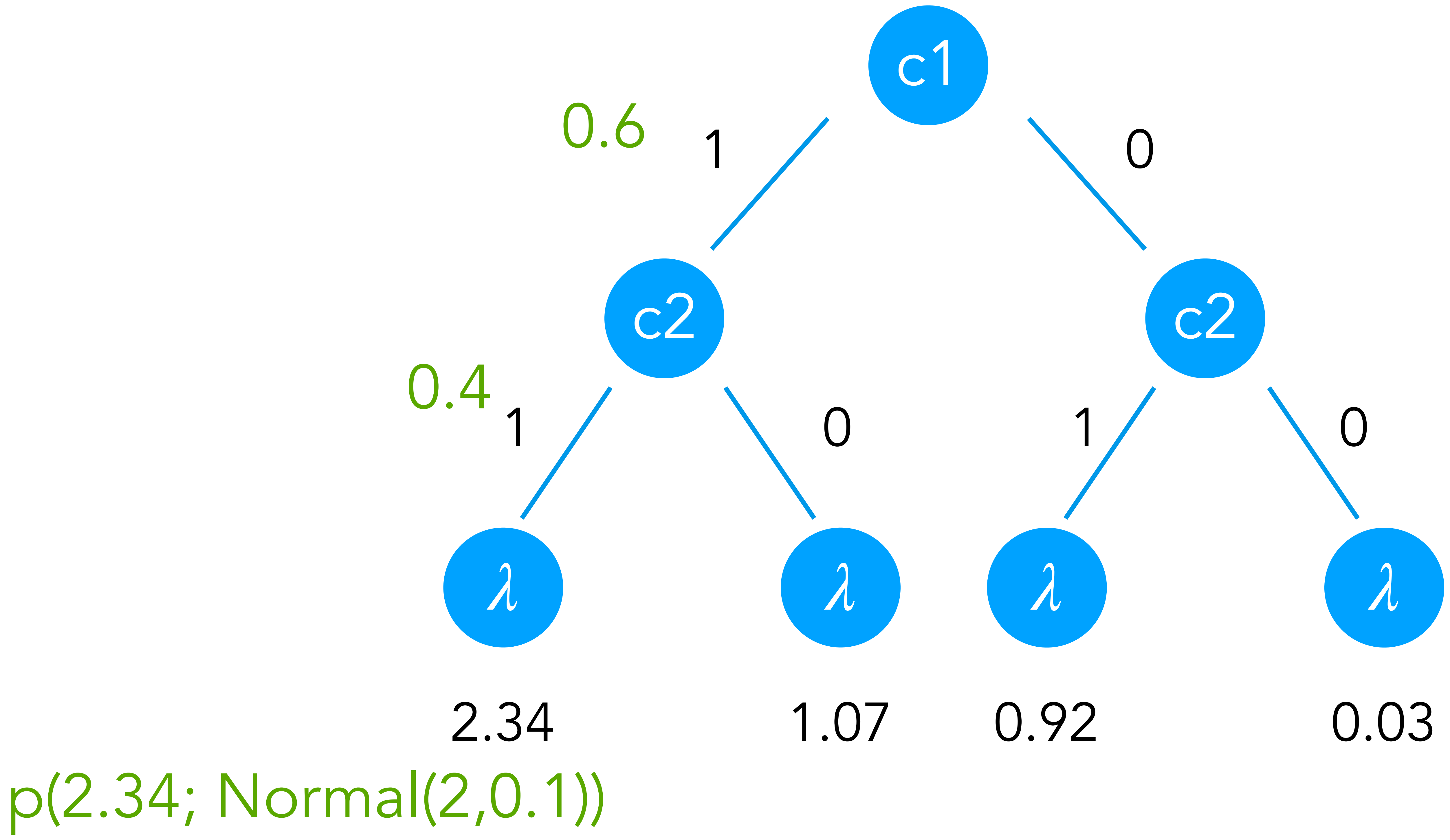
Probabilistic programs with continuous distributions

c1 ~ sample(Bernoulli(0.6))
c2 ~ sample(Bernoulli(0.4))

λ ~ sample(Normal(c1+c2, 0.1))

observe(λ, Normal(2, 1))

return c1 + c2 + c3



Desiderata for general inference techniques

General inference technique: doesn't care what is in the program

- All programming constructs (loops, conditions, …)
- All distributions (continuous and discrete)
- Finite and infinite distribution traces

Probabilistic inference
Grand tour

Preliminaries

```
λ ~ sample( Normal(0.5, 1) )

c1 ~ sample( Bernoulli(λ) )
c2 ~ sample( Bernoulli(λ) )
c3 ~ sample( Bernoulli(λ) )

observe(c1+c2+c3,  Dirac(2))

return λ
```

Preliminaries

```
λ ~ sample( Normal(0.5, 1) )

c1 ~ sample( Bernoulli(λ) )
c2 ~ sample( Bernoulli(λ) )
c3 ~ sample( Bernoulli(λ) )

observe(c1+c2+c3,  Dirac(2))

return λ
```

Program

Preliminaries

Probabilistic choice

```
λ ~ sample( Normal(0.5, 1) )

c1 ~ sample( Bernoulli(λ) )
c2 ~ sample( Bernoulli(λ) )
c3 ~ sample( Bernoulli(λ) )

observe(c1+c2+c3,  Dirac(2))

return λ
```

Program

Preliminaries

Probabilistic choice

λ ~ sample(Normal(0.5, 1))

c1 ~ sample(Bernoulli(λ))
c2 ~ sample(Bernoulli(λ))
c3 ~ sample(Bernoulli(λ))

Observations

observe(c1+c2+c3, Dirac(2))

return λ

Program

Preliminaries

Probabilistic choice

λ ~ sample(Normal(0.5, 1))

c1 ~ sample(Bernoulli(λ))
c2 ~ sample(Bernoulli(λ))
c3 ~ sample(Bernoulli(λ))

Observations

observe(c1+c2+c3, Dirac(2))

Outcome

return λ

Program

Preliminaries

Probabilistic choice

λ ~ sample(Normal(0.5, 1))

c1 ~ sample(Bernoulli(λ))

c2 ~ sample(Bernoulli(λ))

c3 ~ sample(Bernoulli(λ))

Observations

observe(c1+c2+c3, Dirac(2))

Outcome — return λ

Program

Trace: a state of all probabilistic choices in a program

λ : 0.43
c1: 0
c2 : 1
c3 : 0

Monte Carlo estimation

Randomly simulate a process

P(desired outcome) = E[desired outcome]

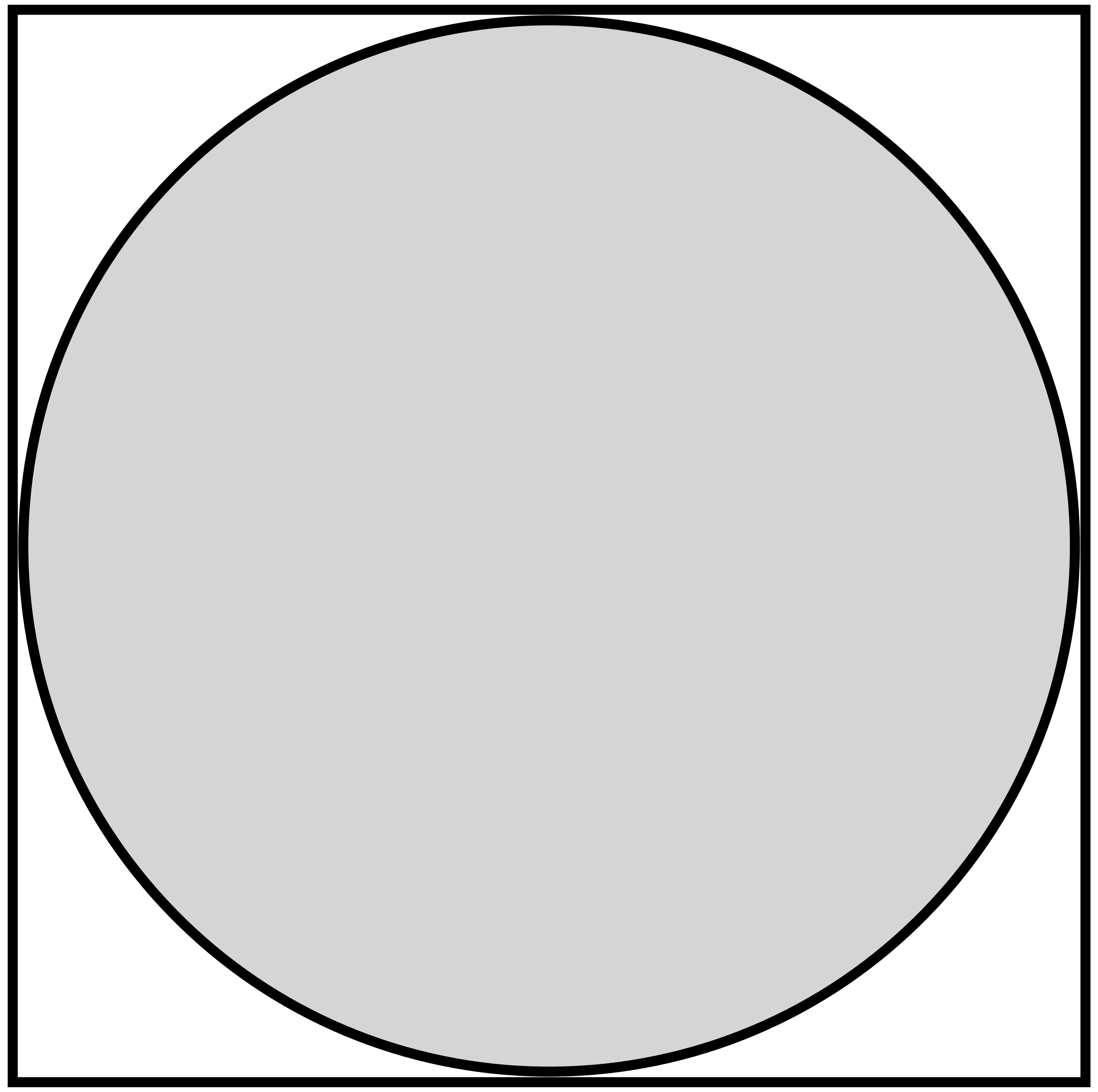
$$= \frac{\text{simulations with desired outcome}}{\text{all simulations}}$$

Monte Carlo estimation

Randomly simulate a process

P(desired outcome) = E[desired outcome]

$$= \frac{\text{simulations with desired outcome}}{\text{all simulations}}$$



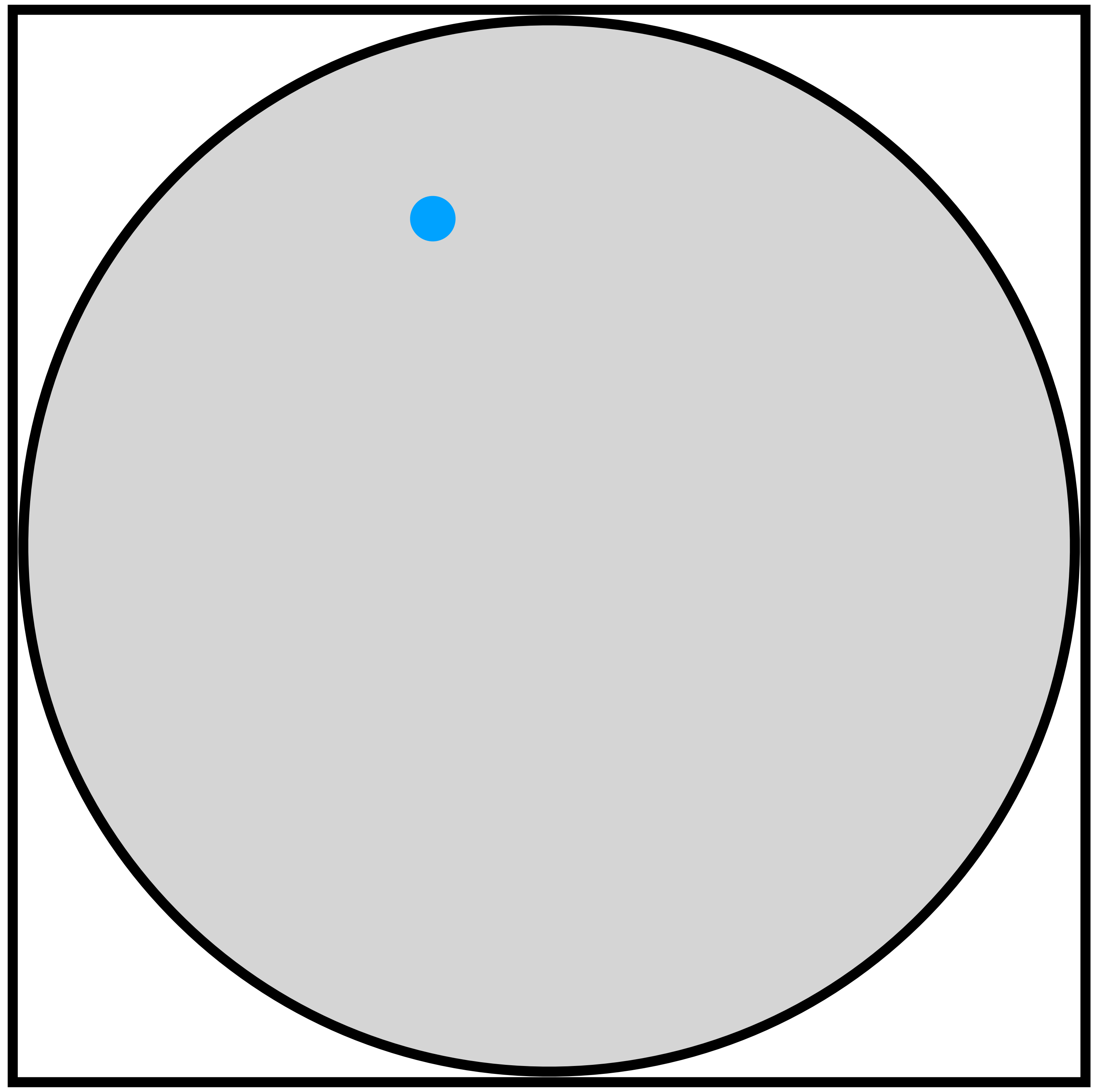
$$\pi = 4\frac{\text{area circle}}{\text{area square}} = 4\frac{\text{inner}}{\text{total}}$$

Monte Carlo estimation

Randomly simulate a process

P(desired outcome) = E[desired outcome]

$$= \frac{\text{simulations with desired outcome}}{\text{all simulations}}$$



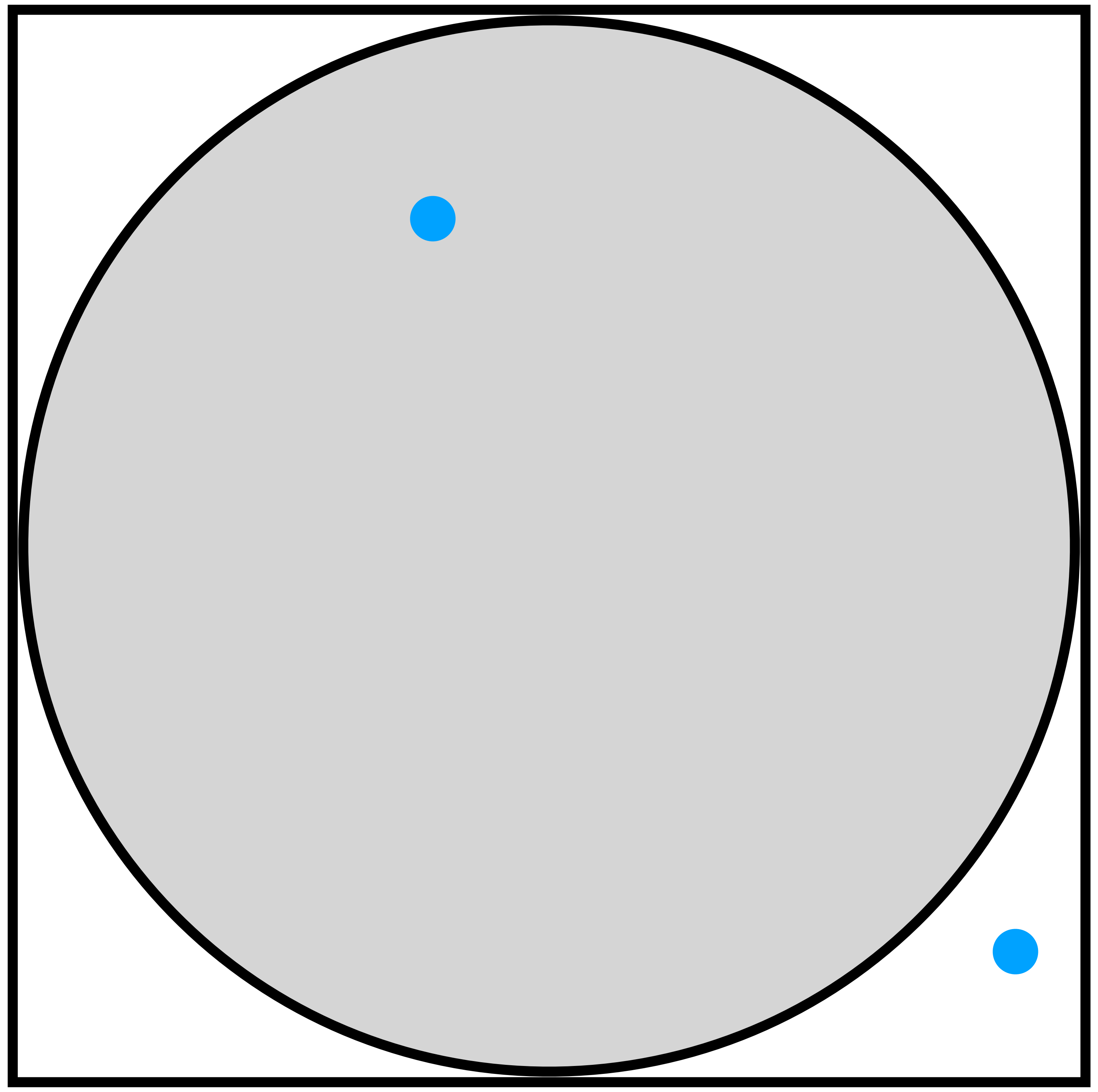
$$\pi = 4\frac{\text{area circle}}{\text{area square}} = 4\frac{\text{inner}}{\text{total}}$$

Monte Carlo estimation

Randomly simulate a process

P(desired outcome) = E[desired outcome]

$$= \frac{\text{simulations with desired outcome}}{\text{all simulations}}$$



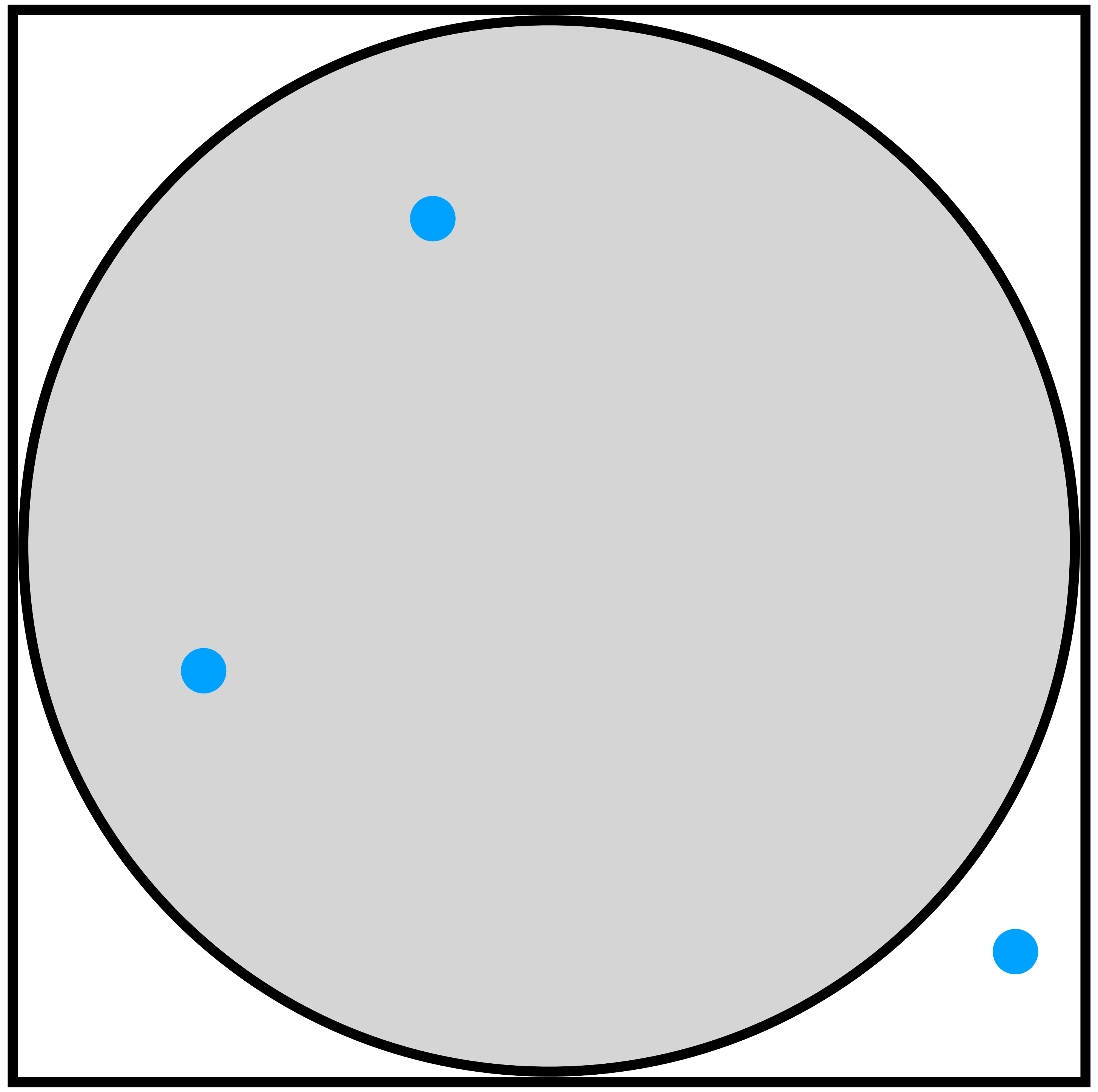
$$\pi = 4\frac{\text{area circle}}{\text{area square}} = 4\frac{\text{inner}}{\text{total}}$$

Monte Carlo estimation

Randomly simulate a process

P(desired outcome) = E[desired outcome]

$$= \frac{\text{simulations with desired outcome}}{\text{all simulations}}$$



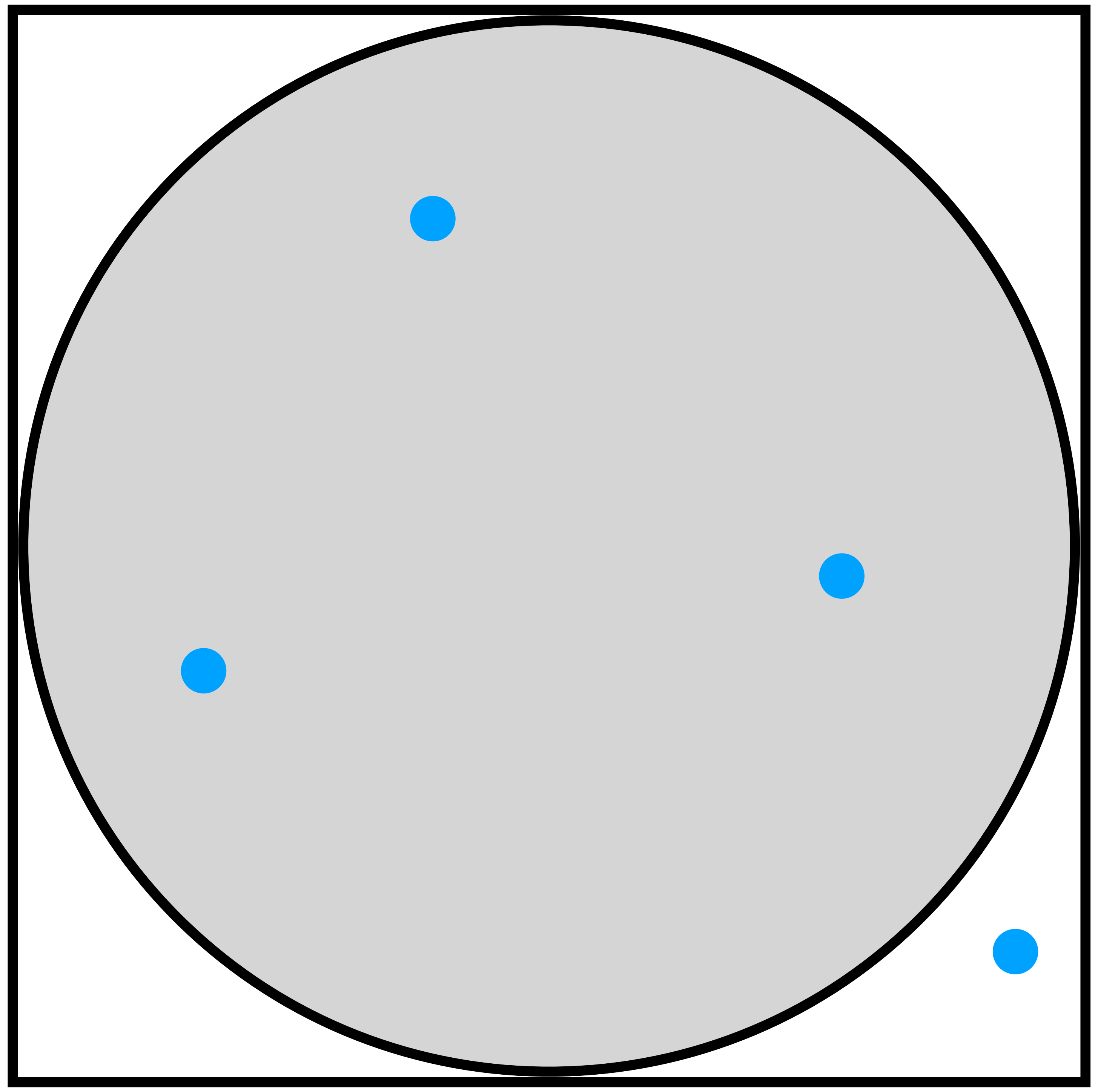
$$\pi = 4\frac{\text{area circle}}{\text{area square}} = 4\frac{\text{inner}}{\text{total}}$$

Monte Carlo estimation

Randomly simulate a process

P(desired outcome) = E[desired outcome]

$$= \frac{\text{simulations with desired outcome}}{\text{all simulations}}$$



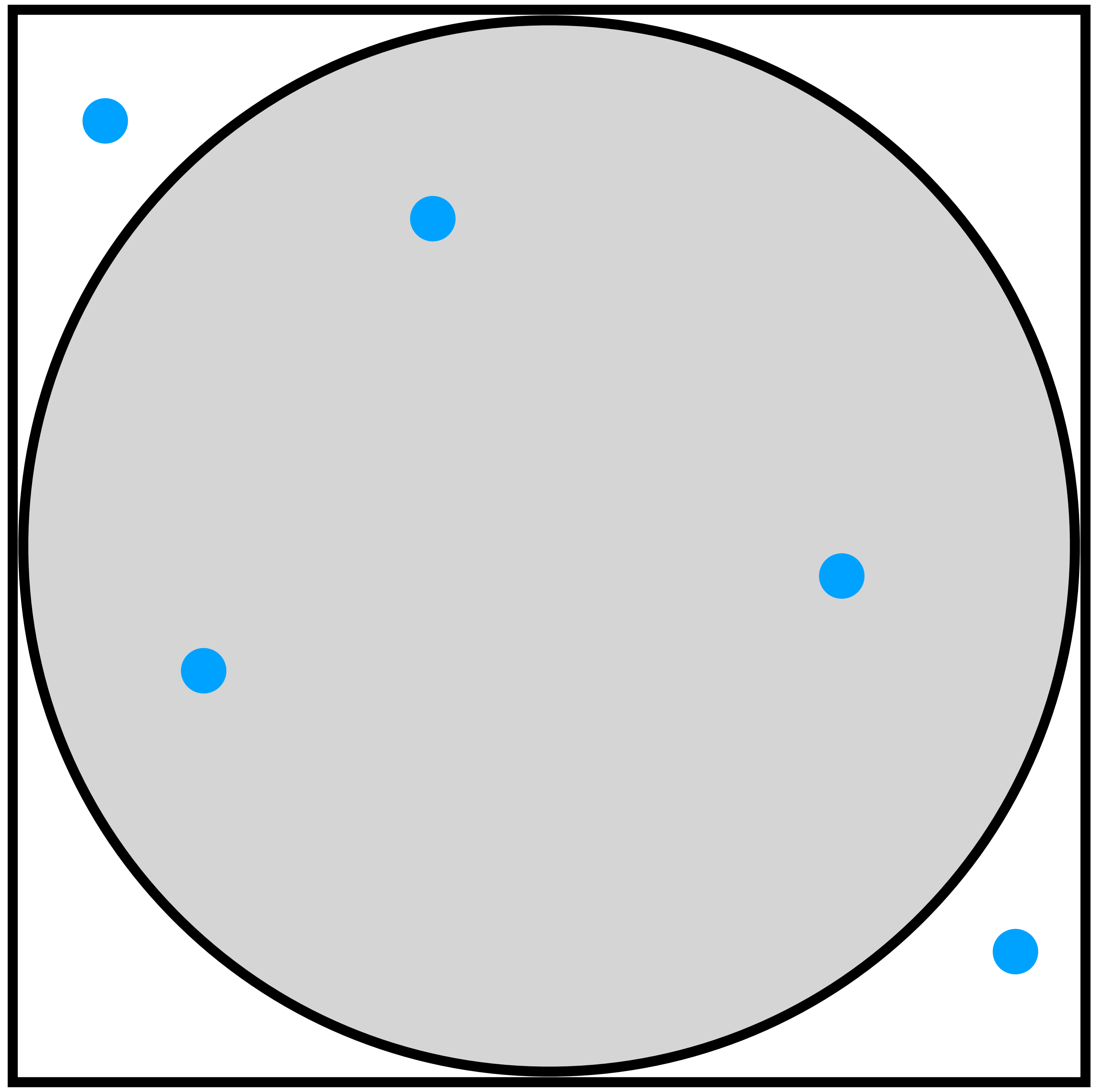
$$\pi = 4\frac{\text{area circle}}{\text{area square}} = 4\frac{\text{inner}}{\text{total}}$$

Monte Carlo estimation

Randomly simulate a process

P(desired outcome) = E[desired outcome]

$$= \frac{\text{simulations with desired outcome}}{\text{all simulations}}$$



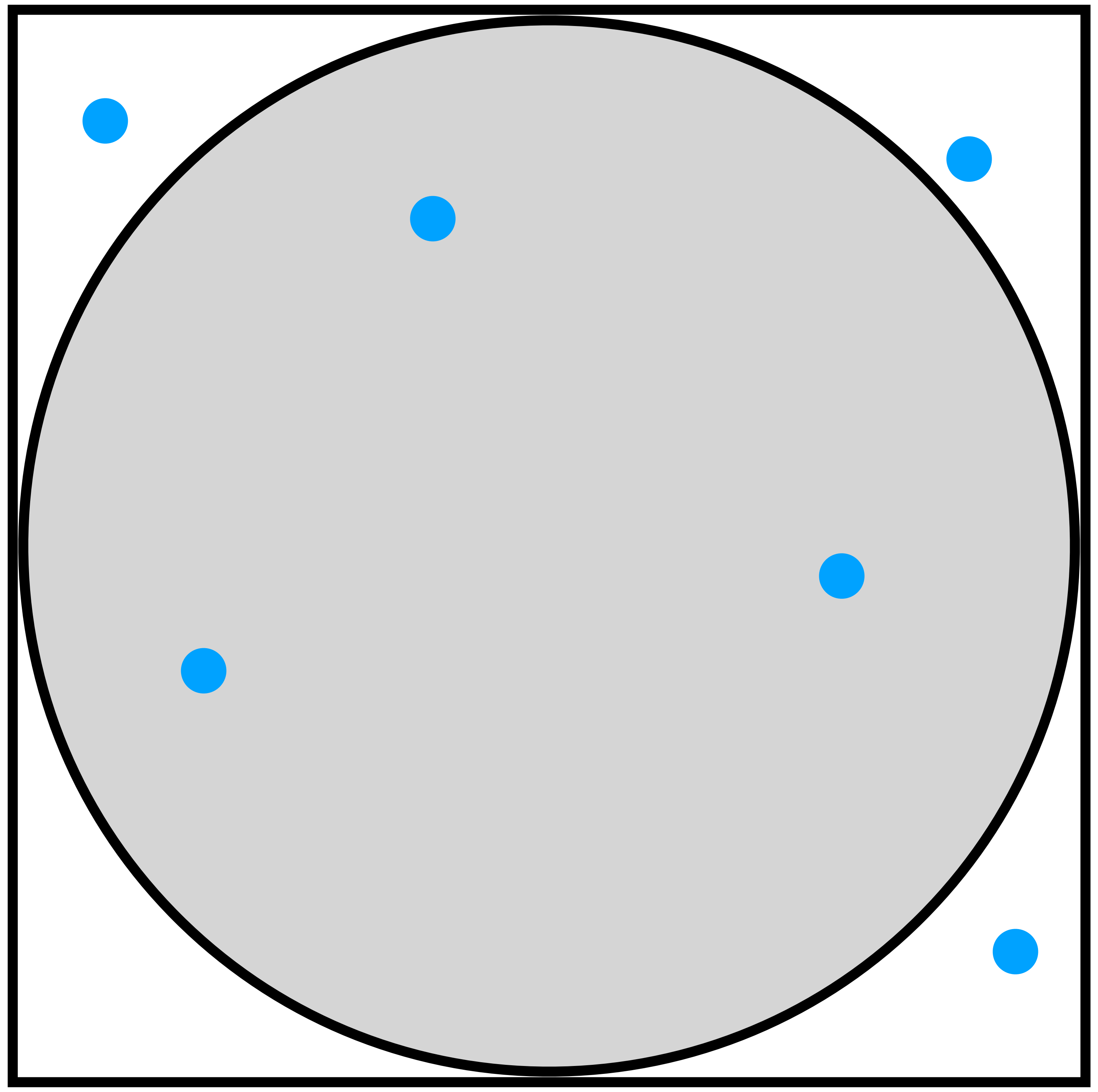
$$\pi = 4\frac{\text{area circle}}{\text{area square}} = 4\frac{\text{inner}}{\text{total}}$$

Monte Carlo estimation

Randomly simulate a process

P(desired outcome) = E[desired outcome]

$$= \frac{\text{simulations with desired outcome}}{\text{all simulations}}$$



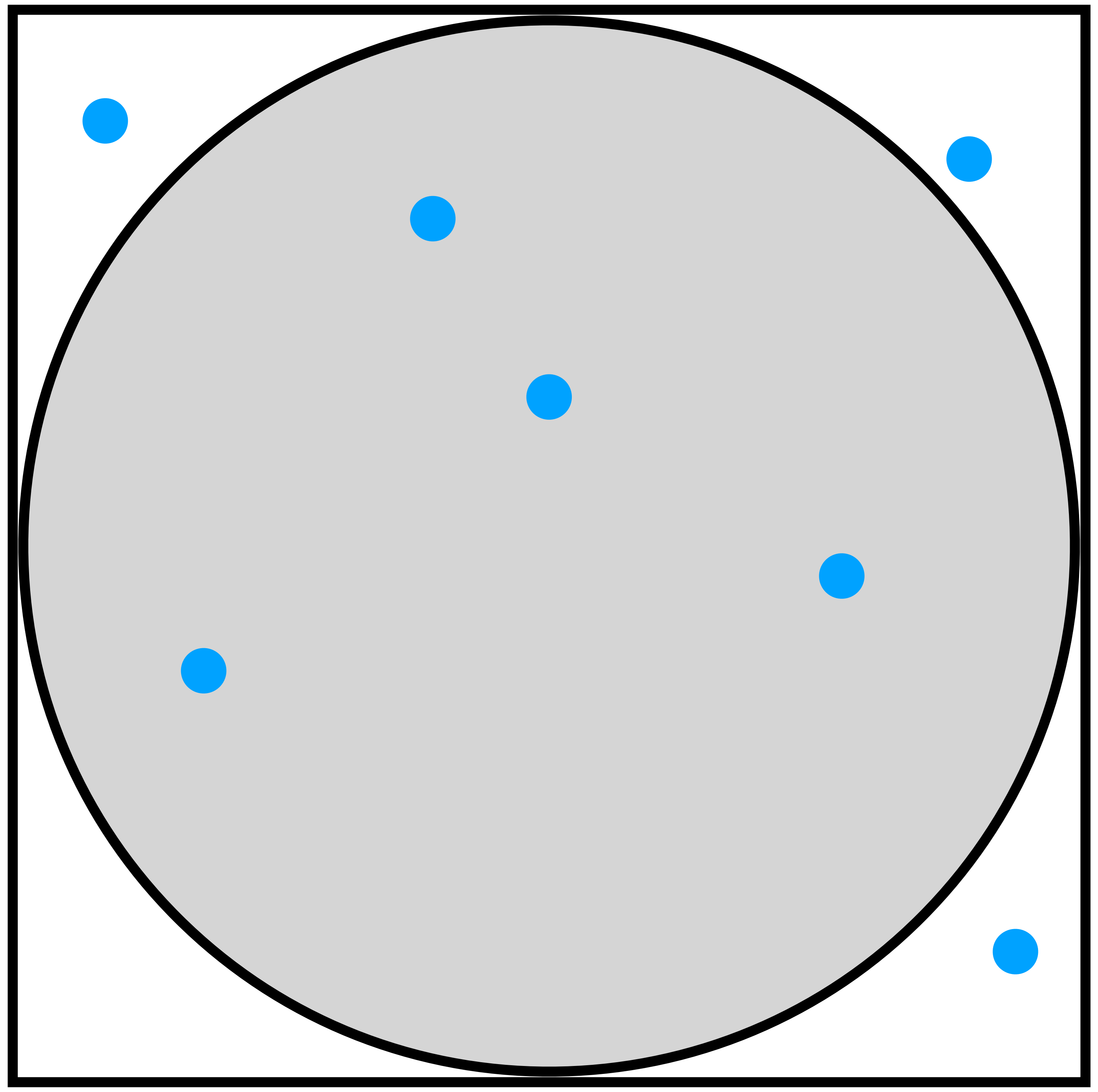
$$\pi = 4\frac{\text{area circle}}{\text{area square}} = 4\frac{\text{inner}}{\text{total}}$$

Monte Carlo estimation

Randomly simulate a process

P(desired outcome) = E[desired outcome]

$$= \frac{\text{simulations with desired outcome}}{\text{all simulations}}$$



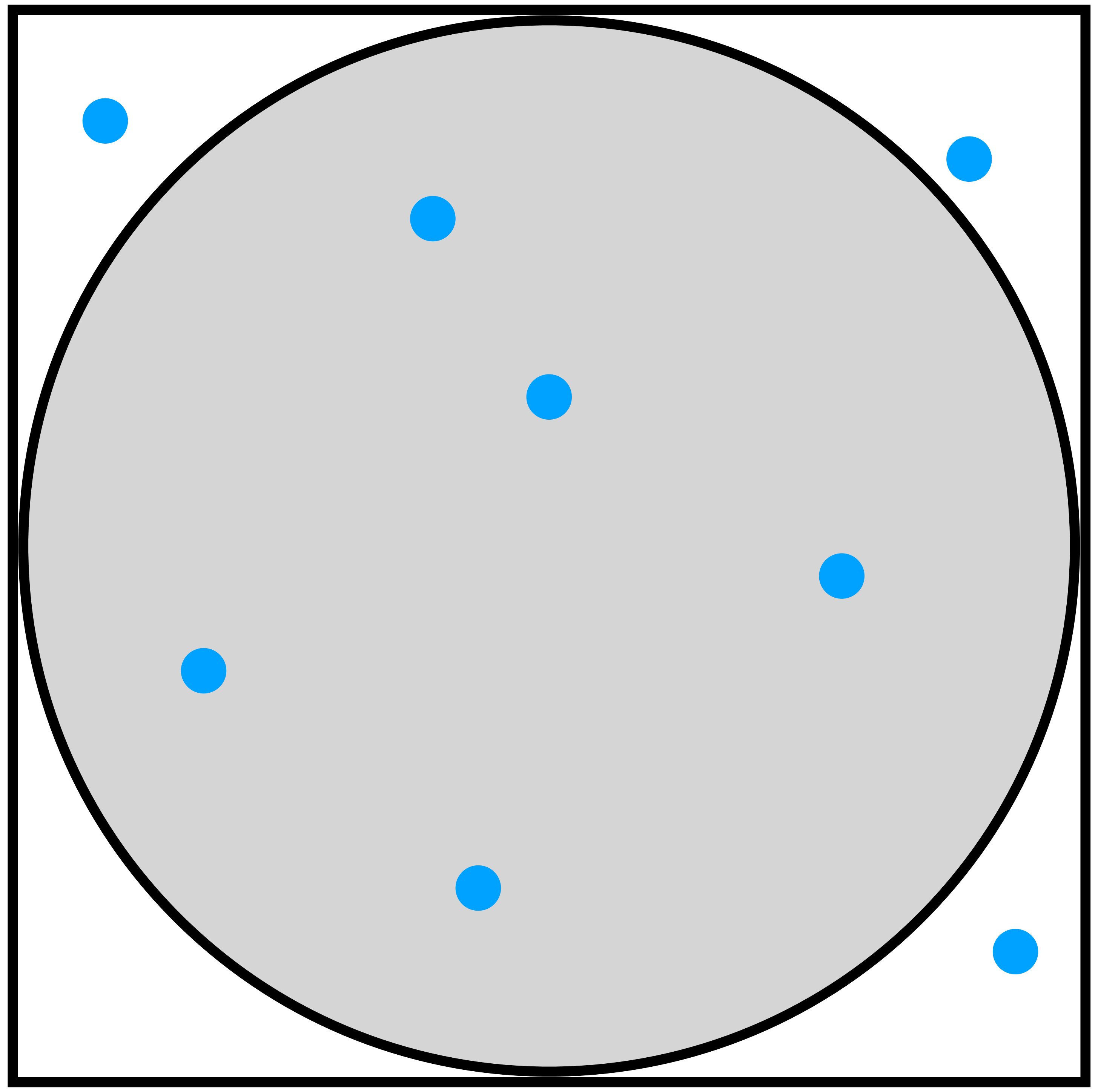
$$\pi = 4\frac{\text{area circle}}{\text{area square}} = 4\frac{\text{inner}}{\text{total}}$$

Monte Carlo estimation

Randomly simulate a process

P(desired outcome) = E[desired outcome]

$$= \frac{\text{simulations with desired outcome}}{\text{all simulations}}$$



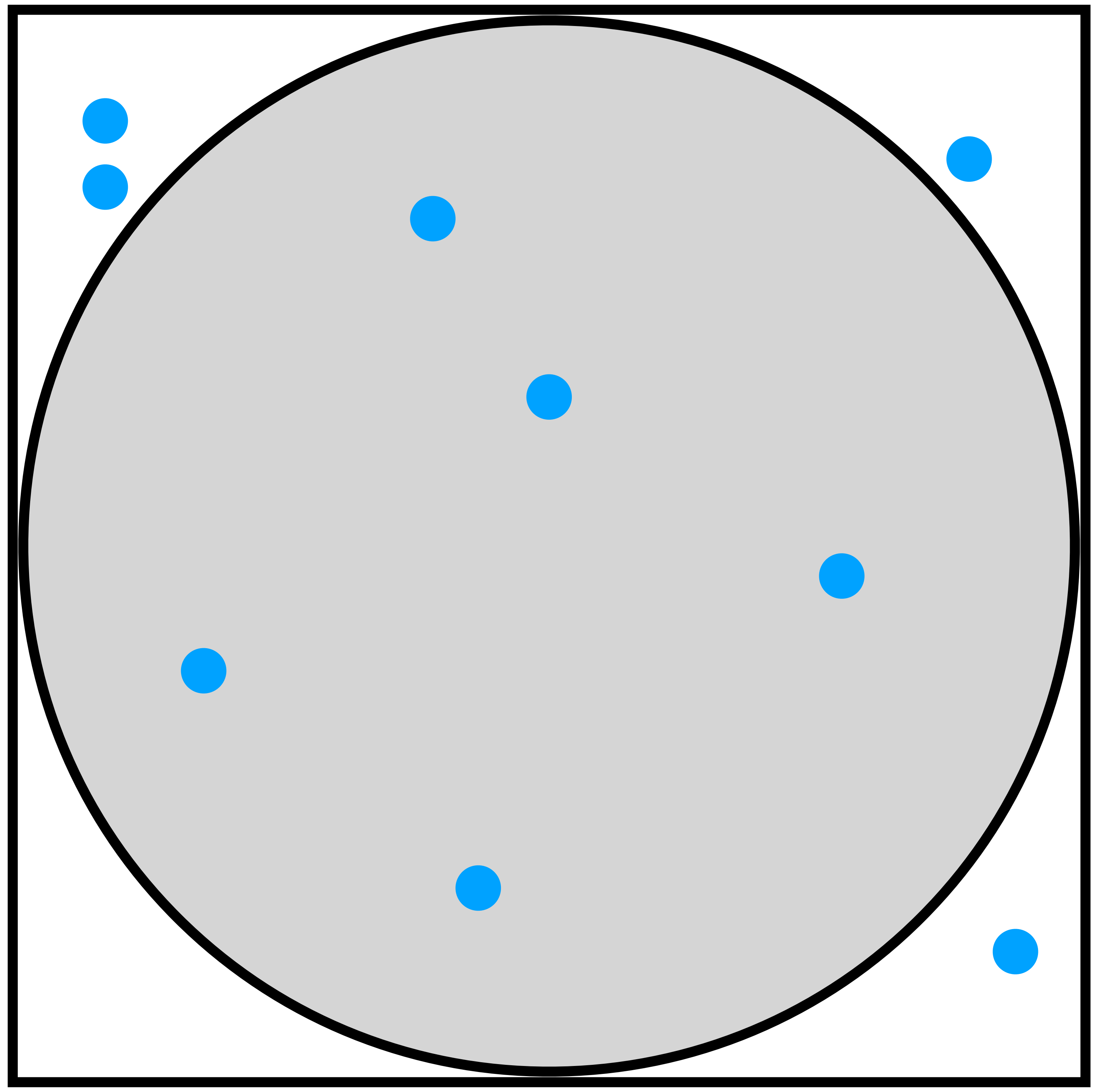
$$\pi = 4\frac{\text{area circle}}{\text{area square}} = 4\frac{\text{inner}}{\text{total}}$$

Monte Carlo estimation

Randomly simulate a process

P(desired outcome) = E[desired outcome]

$$= \frac{\text{simulations with desired outcome}}{\text{all simulations}}$$



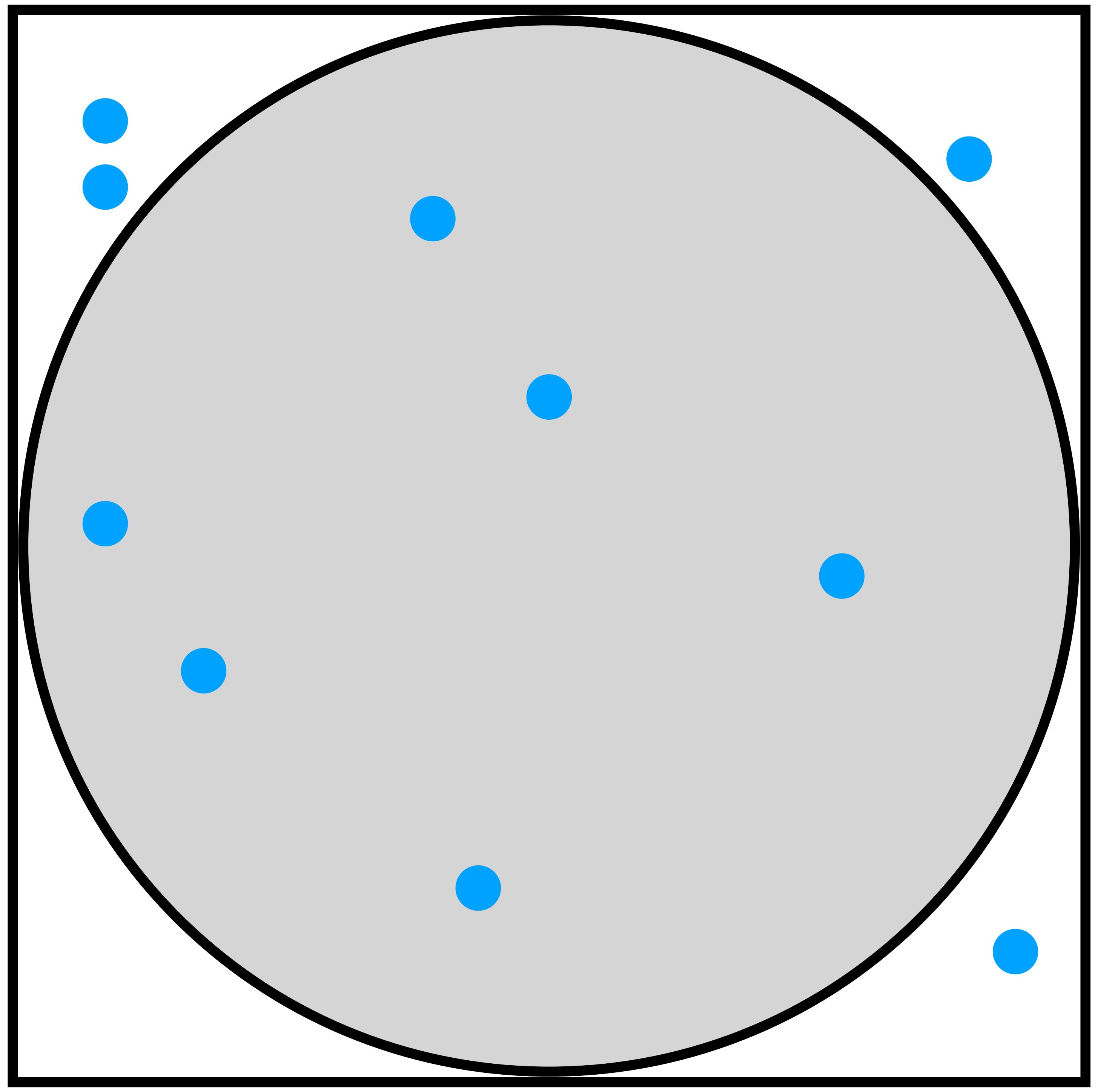
$$\pi = 4\frac{\text{area circle}}{\text{area square}} = 4\frac{\text{inner}}{\text{total}}$$

Monte Carlo estimation

Randomly simulate a process

P(desired outcome) = E[desired outcome]

$$= \frac{\text{simulations with desired outcome}}{\text{all simulations}}$$



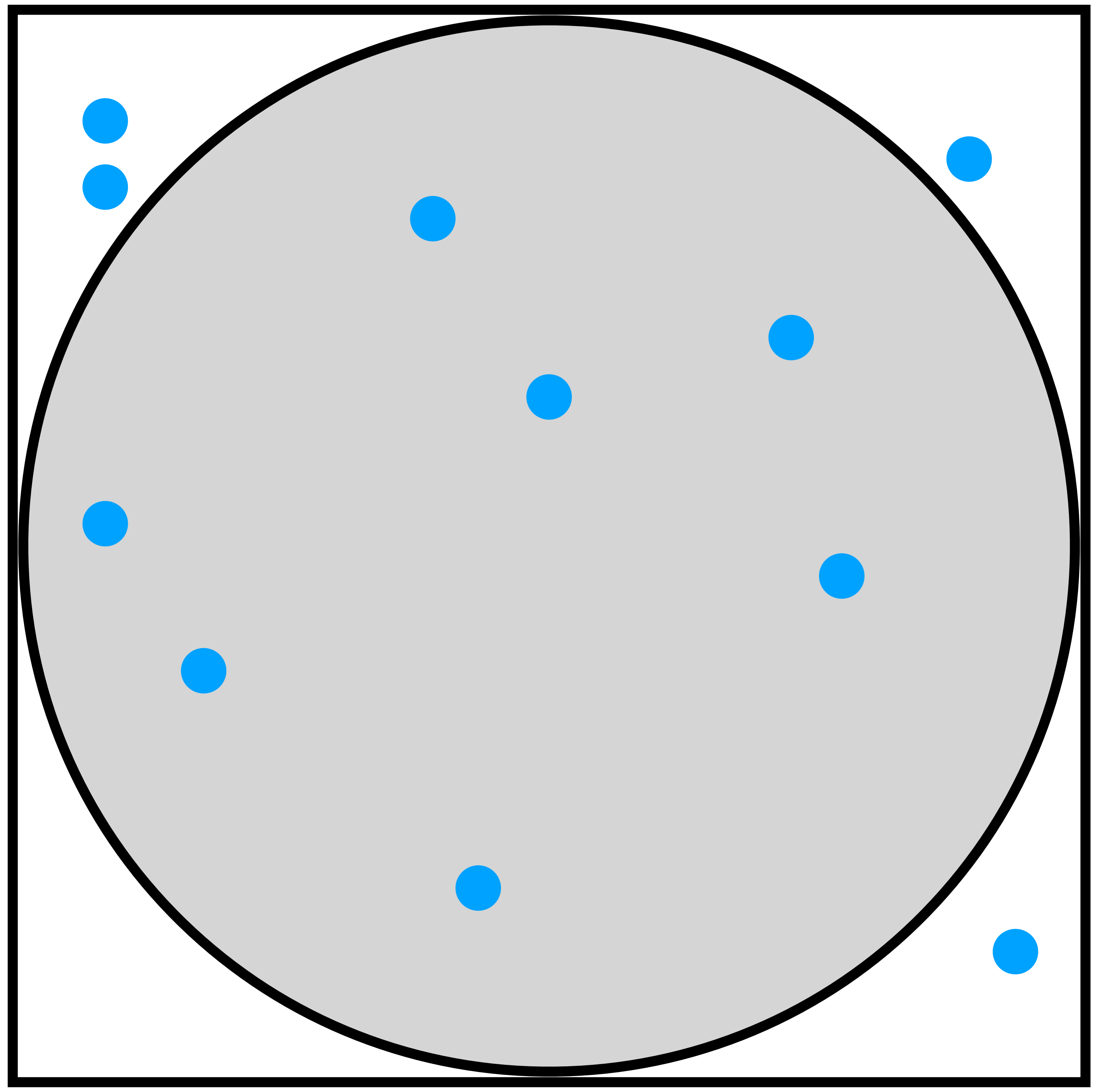
$$\pi = 4\frac{\text{area circle}}{\text{area square}} = 4\frac{\text{inner}}{\text{total}}$$

Monte Carlo estimation

Randomly simulate a process

P(desired outcome) = E[desired outcome]

$$= \frac{\text{simulations with desired outcome}}{\text{all simulations}}$$



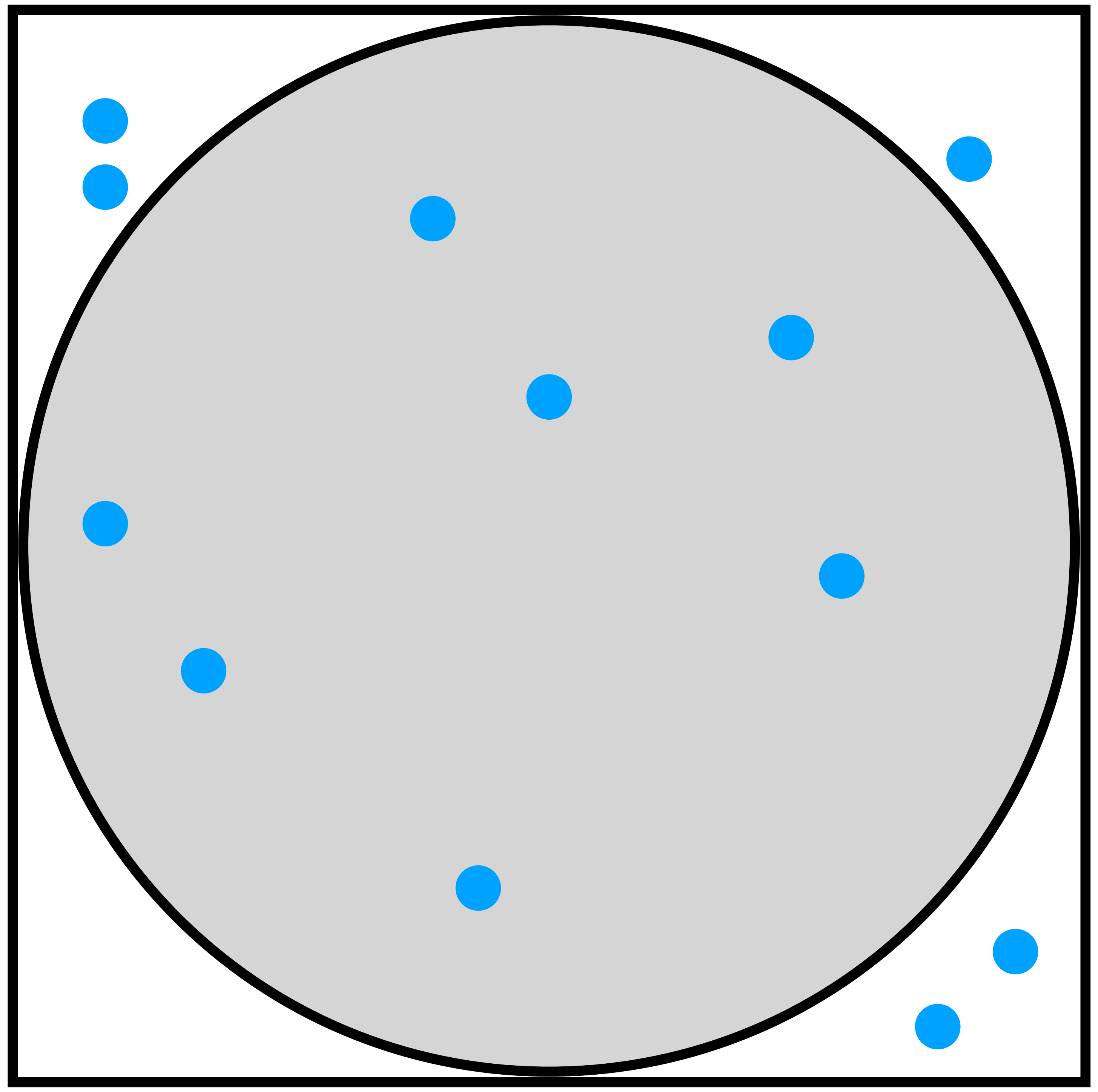
$$\pi = 4\frac{\text{area circle}}{\text{area square}} = 4\frac{\text{inner}}{\text{total}}$$

Monte Carlo estimation

Randomly simulate a process

P(desired outcome) = E[desired outcome]

$$= \frac{\text{simulations with desired outcome}}{\text{all simulations}}$$



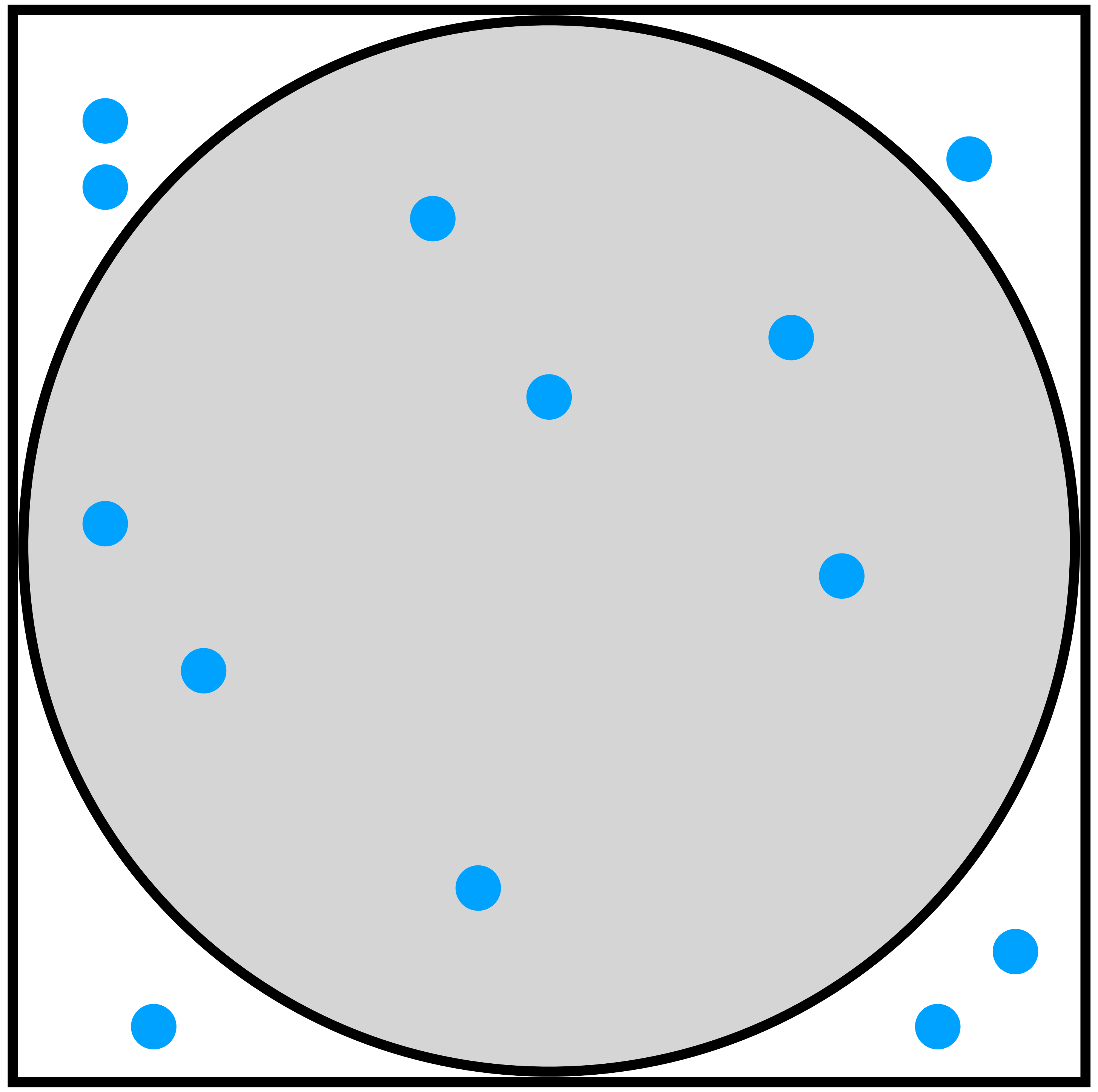
$$\pi = 4\frac{\text{area circle}}{\text{area square}} = 4\frac{\text{inner}}{\text{total}}$$

Monte Carlo estimation

Randomly simulate a process

P(desired outcome) = E[desired outcome]

$$= \frac{\text{simulations with desired outcome}}{\text{all simulations}}$$



$$\pi = 4\frac{\text{area circle}}{\text{area square}} = 4\frac{\text{inner}}{\text{total}}$$

Running example

Measure a heat source in a factory with 3 different sensors

```
heat ~ sample( Normal(56,10) )
sensor1 ~ sample( Normal(heat,3) )
sensor2 ~ sample( Normal(heat,5) )
sensor3 ~ sample( Normal(heat,5) )

observe( sensor2, Normal(43,2) )

return heat
```

Probabilistic inference
Importance sampling

Importance sampling

Execute the probabilistic program N times

Treat the distribution over outcomes as the empirical distribution

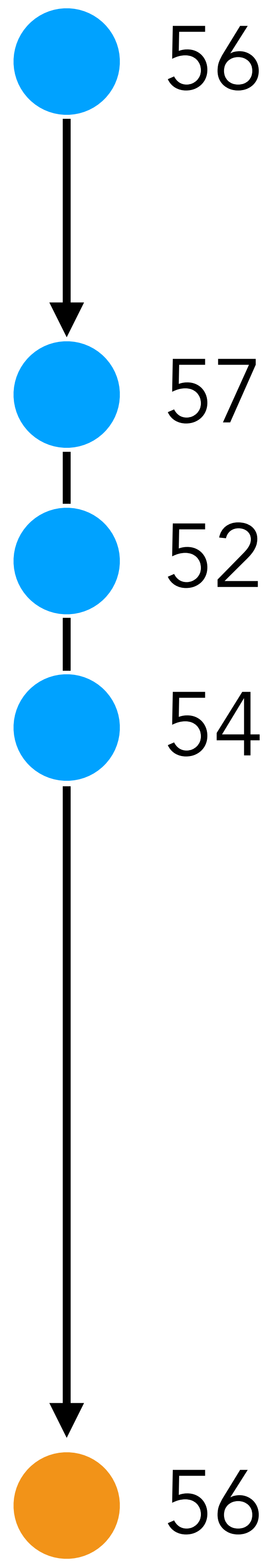
```
heat ~ sample( Normal(56,10) )

sensor1 ~ sample( Normal(heat,3) )
sensor2 ~ sample( Normal(heat,5) )
sensor3 ~ sample( Normal(heat,5) )

observe( sensor2, Normal(43,2) )

return heat
```

Importance sampling

Execute the probabilistic program *N* times

Treat the distribution over outcomes as the empirical distribution

```
heat ~ sample( Normal(56,10) )

sensor1 ~ sample( Normal(heat,3) )
sensor2 ~ sample( Normal(heat,5) )
sensor3 ~ sample( Normal(heat,5) )

observe( sensor2, Normal(43,2) )

return heat
```

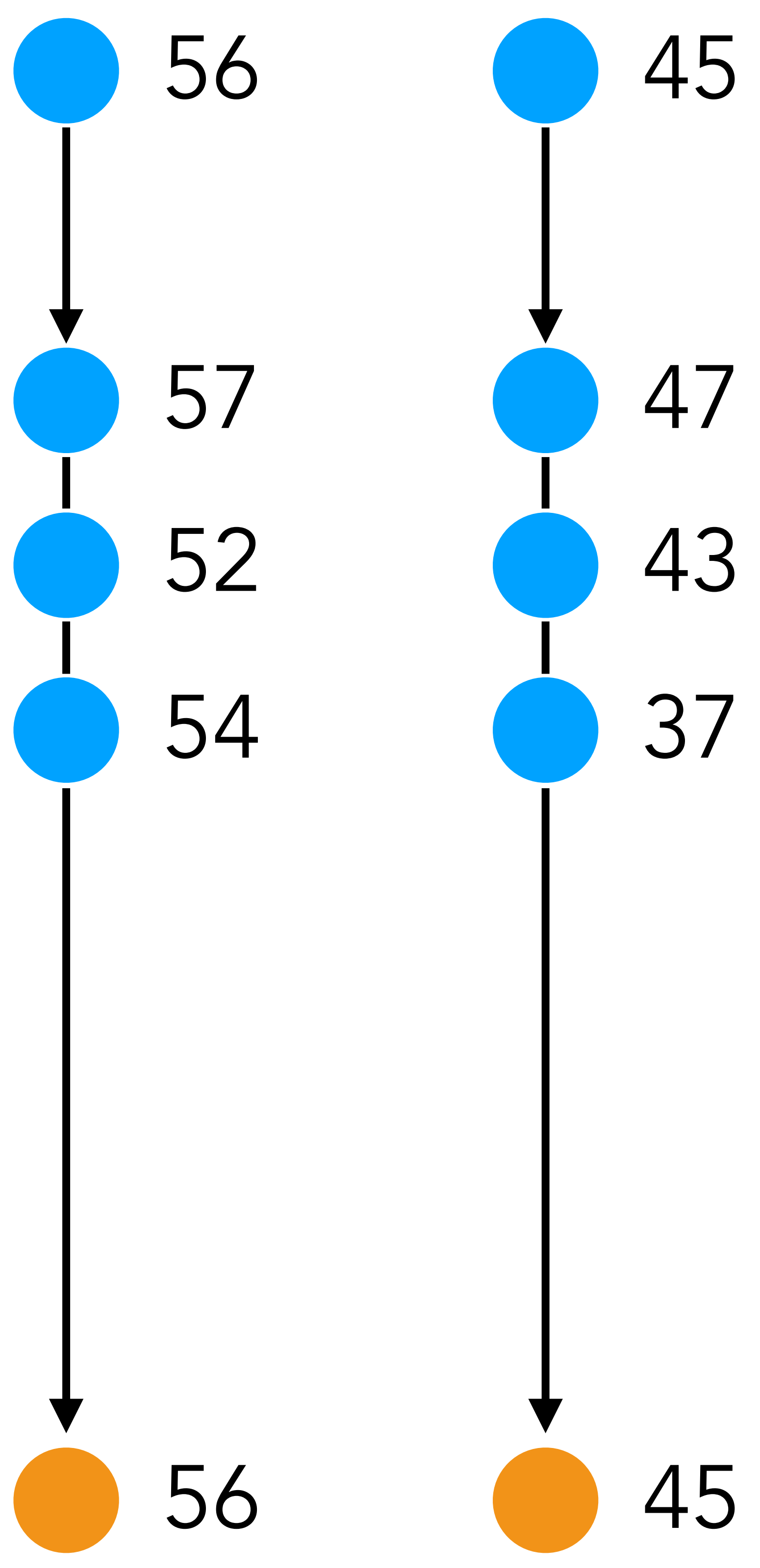


Importance sampling

Execute the probabilistic program *N* times

Treat the distribution over outcomes as the empirical distribution

```
heat ~ sample( Normal(56,10) )

sensor1 ~ sample( Normal(heat,3) )
sensor2 ~ sample( Normal(heat,5) )
sensor3 ~ sample( Normal(heat,5) )

observe( sensor2, Normal(43,2) )

return heat
```

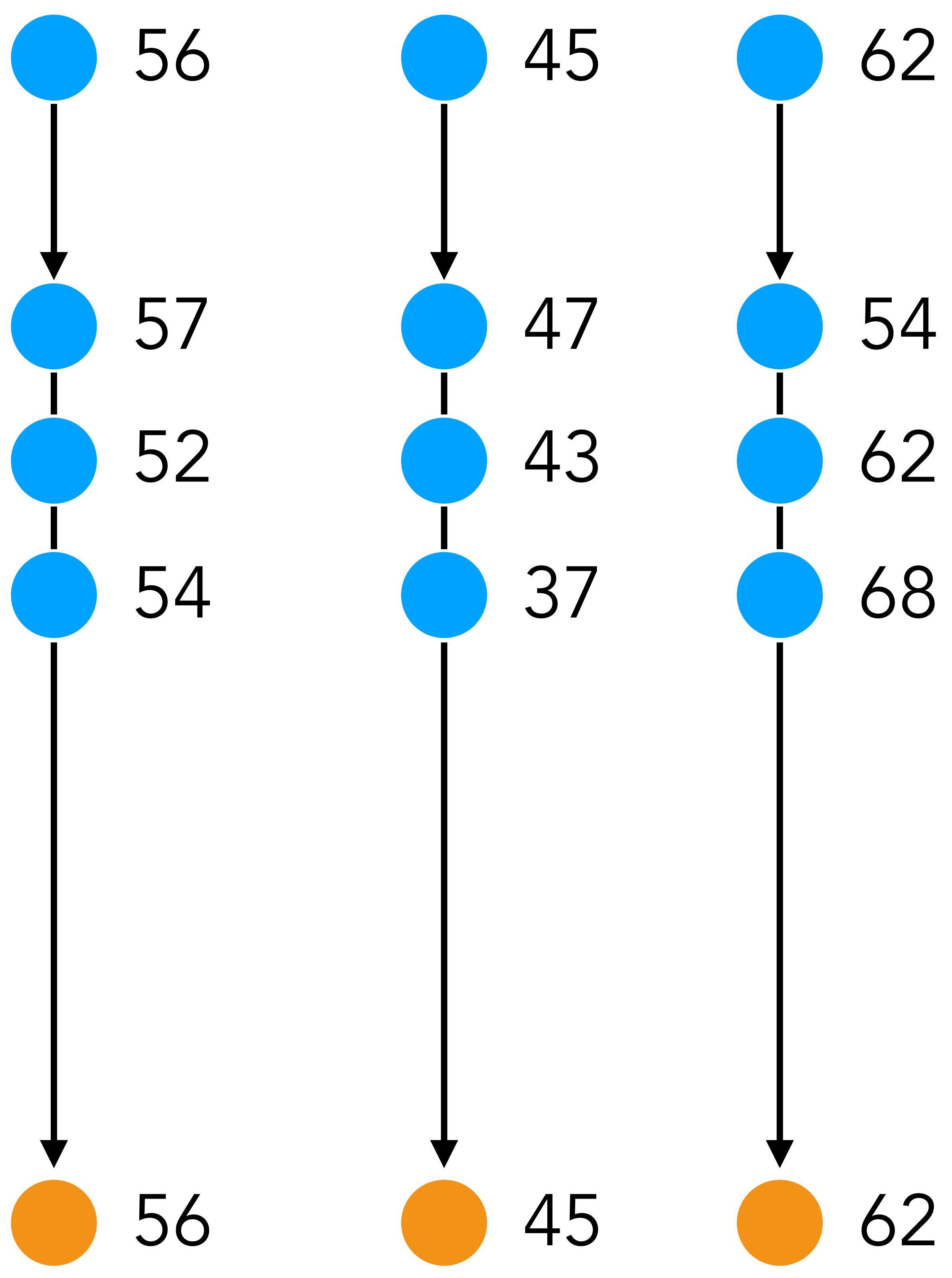


Importance sampling

Execute the probabilistic program *N* times

Treat the distribution over outcomes as the empirical distribution

```
heat ~ sample( Normal(56,10) )

sensor1 ~ sample( Normal(heat,3) )
sensor2 ~ sample( Normal(heat,5) )
sensor3 ~ sample( Normal(heat,5) )

observe( sensor2, Normal(43,2) )

return heat
```



Importance sampling

Execute the probabilistic program *N* times

Treat the distribution over outcomes as the empirical distribution

heat ~ sample(Normal(56,10))

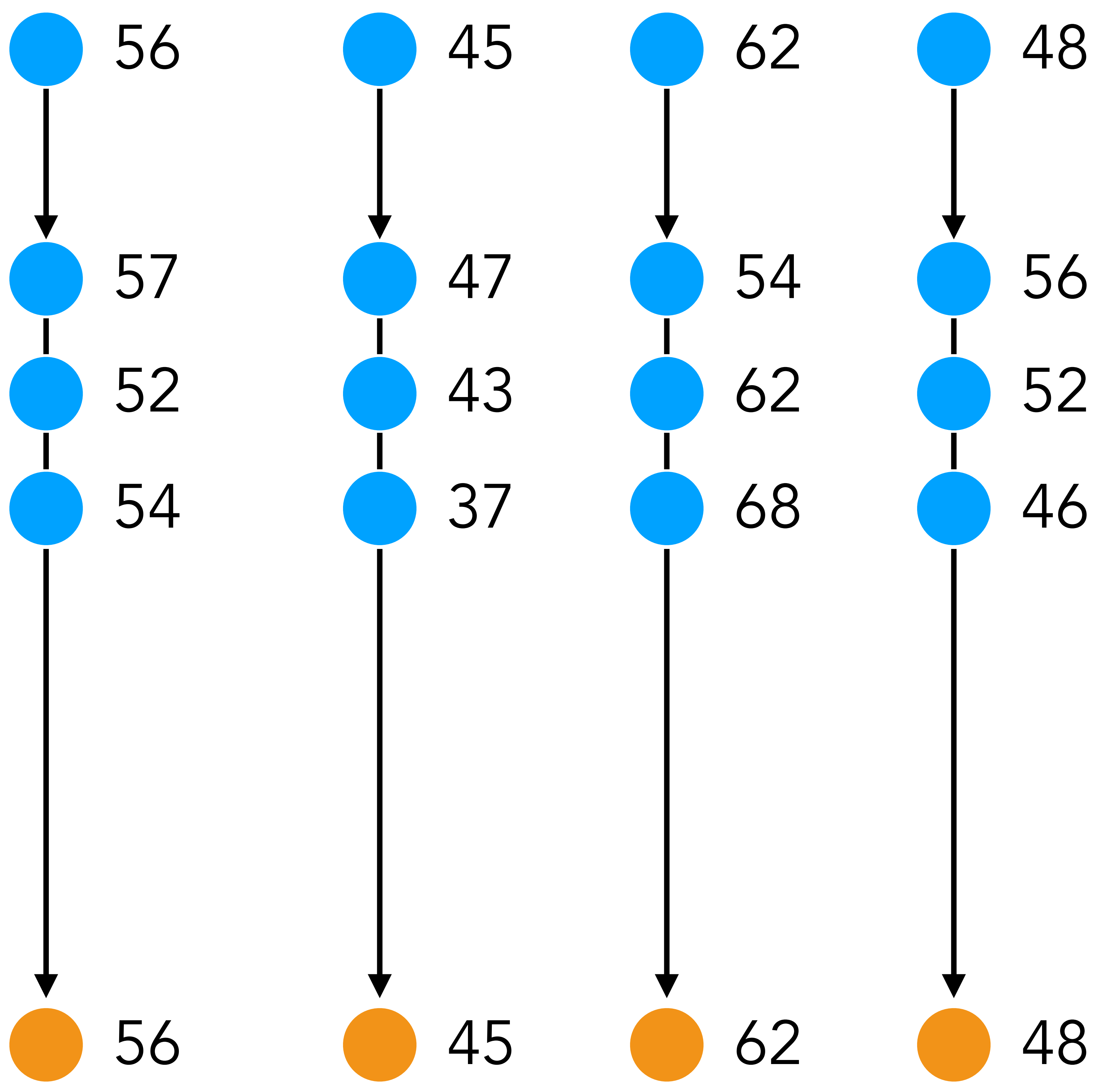
sensor1 ~ sample(Normal(heat,3))

sensor2 ~ sample(Normal(heat,5))

sensor3 ~ sample(Normal(heat,5))

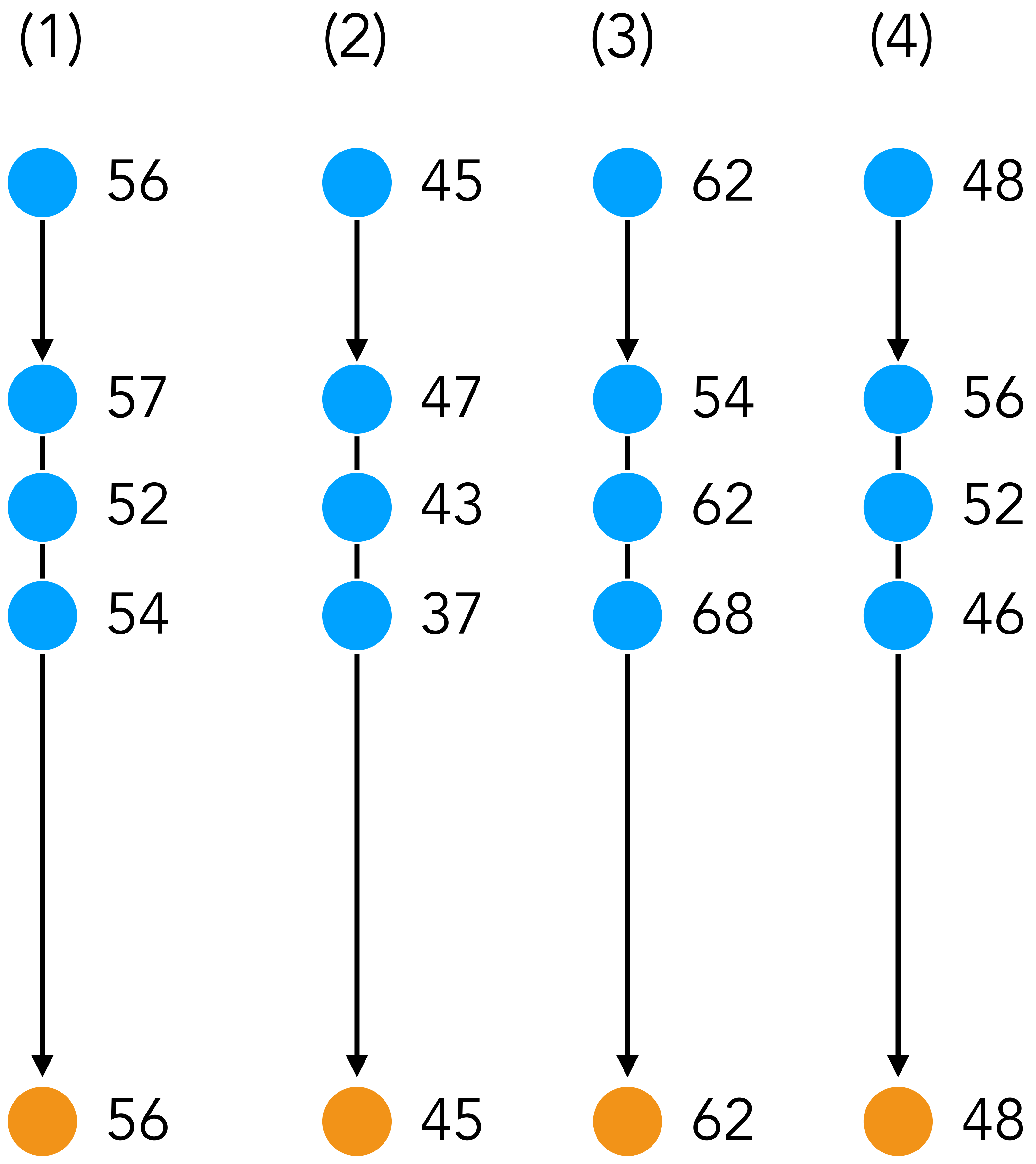
observe(sensor2, Normal(43,2))

return heat



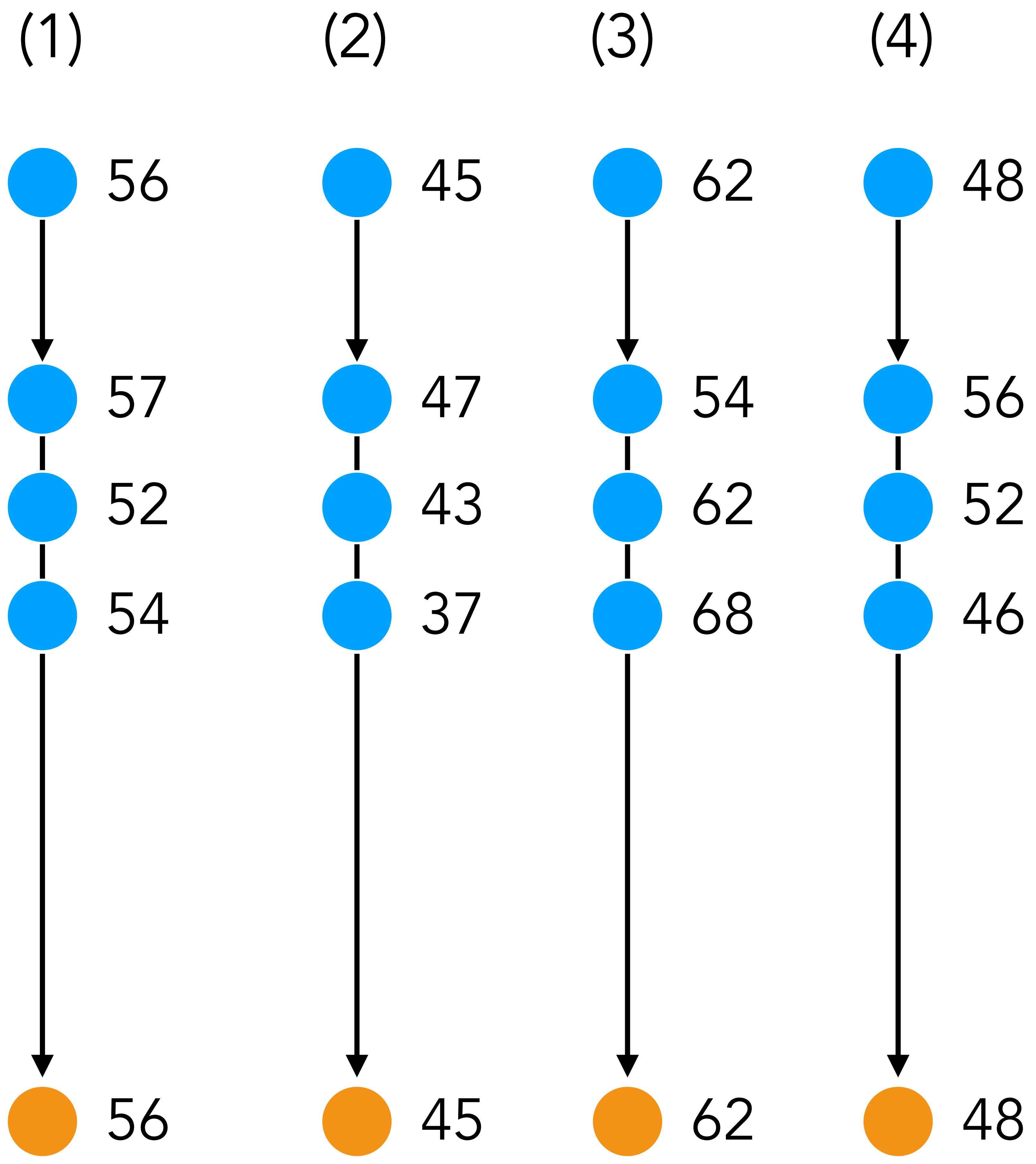
Importance sampling

Samples obtained by executing a program are from *p(heat)* not *p(heat | sensor2 = 43) !*



Importance sampling

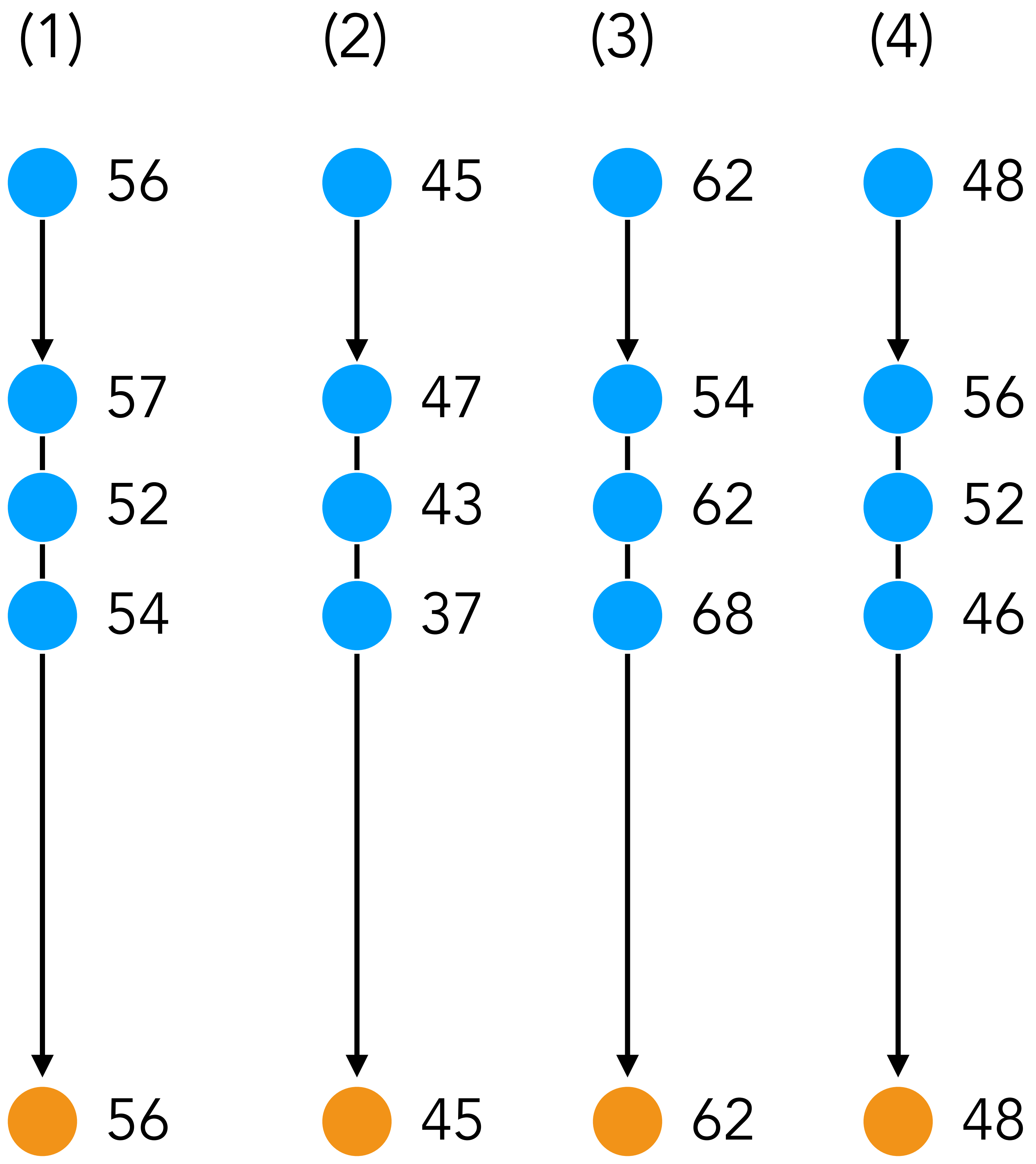
Samples obtained by executing a program are from *p(heat)* not *p(heat | sensor2 = 43)* !



We can fix this by weighting each execution proportionally to how much it agrees with observe(sensor2, Normal(43,2))

Importance sampling

Samples obtained by executing a program are from *p(heat)* not *p(heat | sensor2 = 43)* !



We can fix this by weighting each execution proportionally to how much it agrees with observe(sensor2, Normal(43,2))

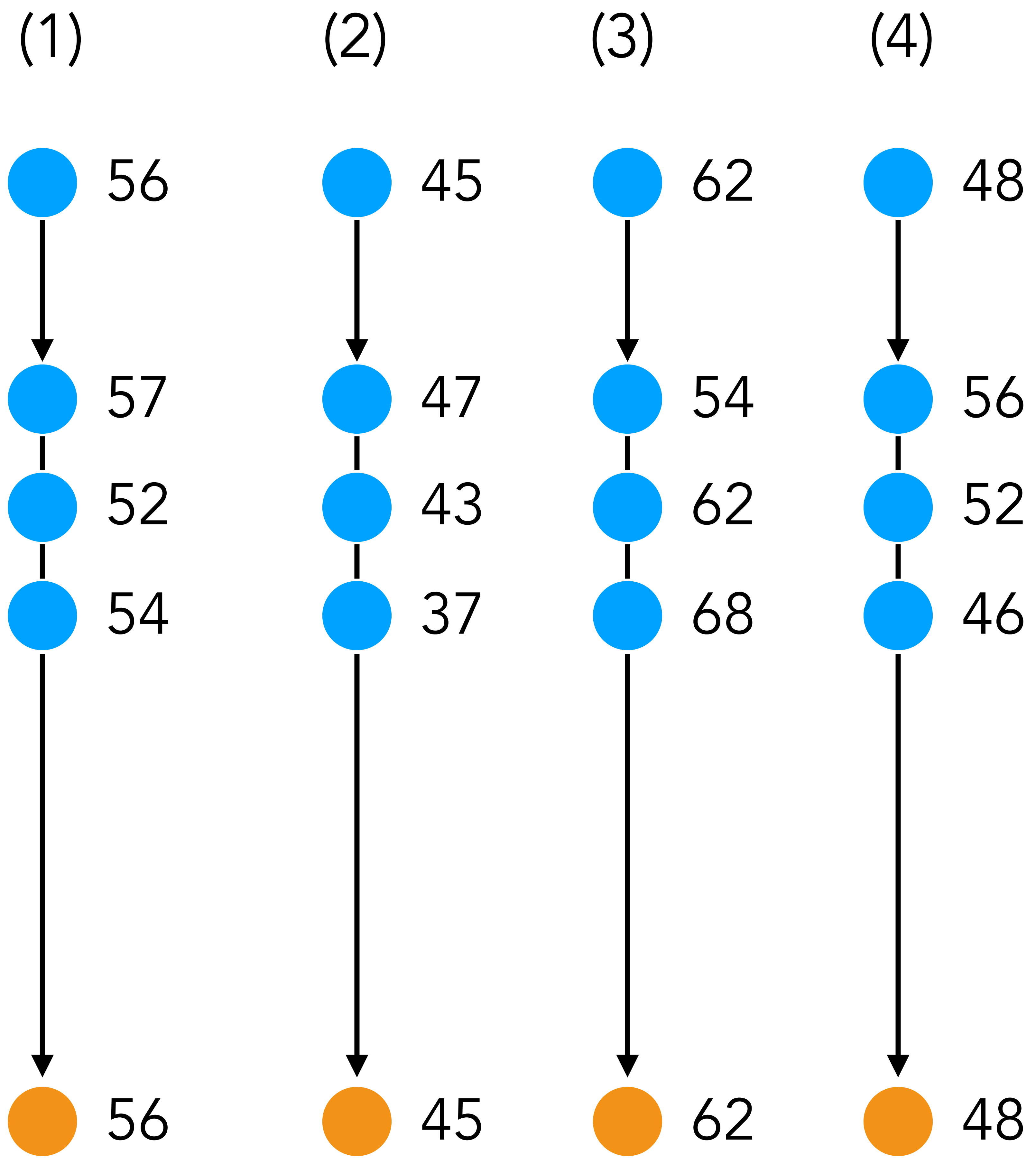
$heat^4 = 48$

$sensor2^4 = 52$

$W^4 = p(sensor2 = 52;\ Normal(43,2))$

Importance sampling

Samples obtained by executing a program are from *p(heat)* not *p(heat | sensor2 = 43)* !



We can fix this by weighting each execution proportionally to how much it agrees with observe(sensor2, Normal(43,2))

$heat^4 = 48$

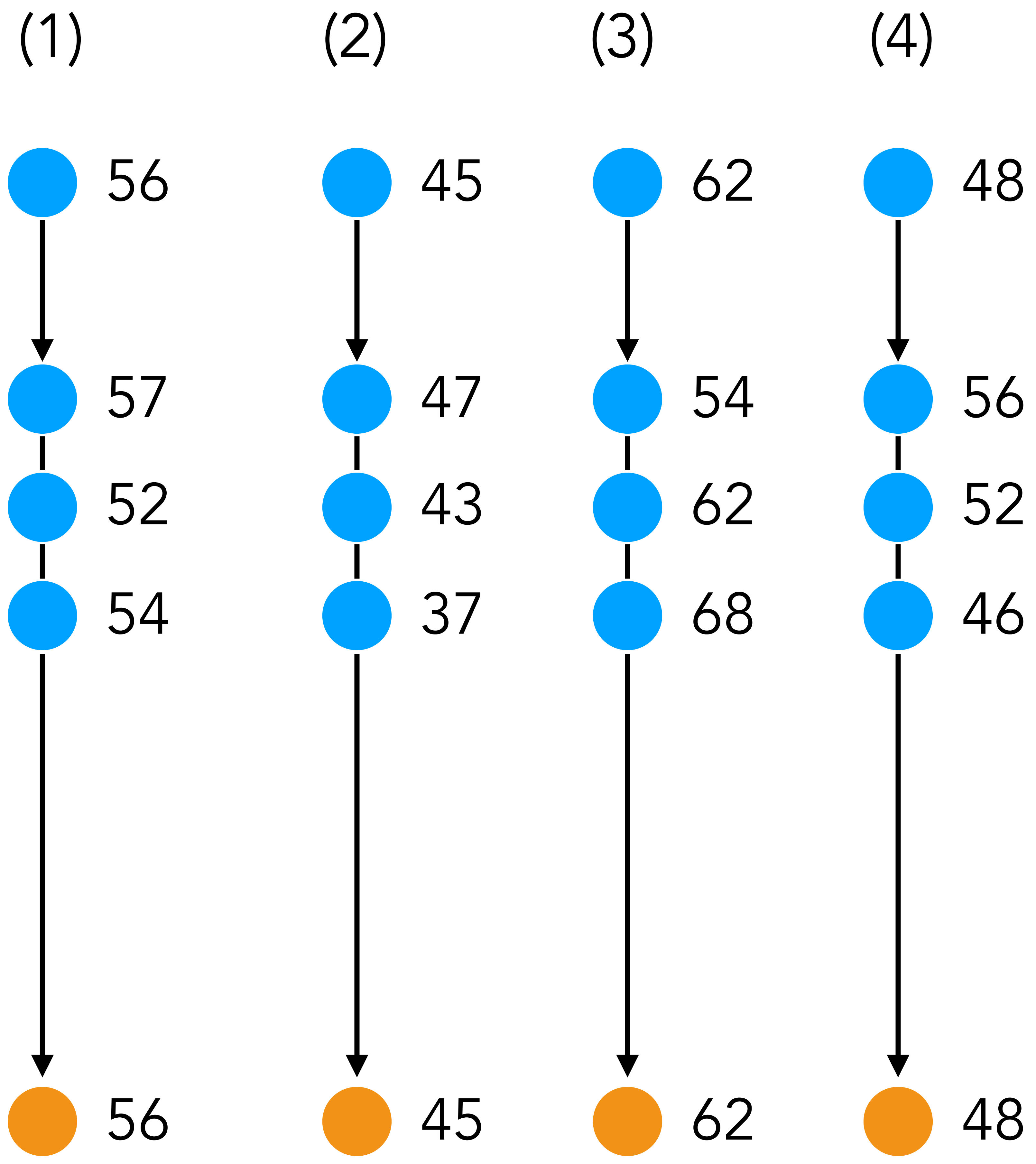
$sensor2^4 = 52$

$W^4 = p(sensor2 = 52; Normal(43,2))$

$W^3 = p(sensor2 = 62; Normal(43,2))$

Importance sampling

Samples obtained by executing a program are from *p(heat)* not *p(heat | sensor2 = 43)* !



We can fix this by weighting each execution proportionally to how much it agrees with observe(sensor2, Normal(43,2))

heat^4 = 48

sensor2^4 = 52

W^4 = p(sensor2 = 52; Normal(43,2))

W^3 = p(sensor2 = 62; Normal(43,2))

Probability of any outcome *i* becomes: $\text{p(outcome}^i) = \dfrac{W^i}{\sum_{k=1}^{L} W^k}$

Importance sampling: why can we re-weight?

Discrete and continuous expectations

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

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

Conditional on another variable

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

Importance sampling: why can we re-weight?

Sidestep sampling from the posterior *p(heat | sensor2 = 43)* entirely,
and draw from some proposal distribution *q(heat)* instead

Any distribution that is easy to sample from

Instead of computing an expectation with respect to *p(heat | sensor2)*,
We compute an expectation with respect to *q(heat)*

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

Importance sampling: why can we re-weight?

We define an "importance weight" $W(x) = \dfrac{p(x \mid y)}{q(x)}$

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 are now computed using *weighted* samples from $q(x)$,
instead of unweighted samples from $p(x \mid y)$

Importance sampling: why can we re-weight?

One problem left: we cannot evaluate the weight just yet

$$W(x) = \frac{p(x \mid y)}{q(x)}$$

We did all this to avoid calculating this term

But we can evaluate it up to a constant

$$w(x) = \frac{p(x, y)}{q(x)}$$

Approximation

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

Importance sampling: why can we re-weight?

We already have a very simple proposal distribution we know how to sample from:

the prior $p(x)$

Importance sampling: why can we re-weight?

We already have a very simple proposal distribution we know how to sample from:

the prior $p(x)$

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

Importance sampling: why can we re-weight?

We already have a very simple proposal distribution we know how to sample from:

the prior $p(x)$

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

Then, instead of a “hard” rejection step, we use the values of the latent variables and that data to assign “soft” weights to the sampled values

Properties of importance sampling

General inference technique: doesn’t care what is in the program

- All programming constructs (loops, conditions, …)
- All distributions (continuous and discrete)
- Finite and infinite distribution traces

Properties of importance sampling

General inference technique: doesn’t care what is in the program

- All programming constructs (loops, conditions, …)
- All distributions (continuous and discrete)
- Finite and infinite distribution traces

Properties of importance sampling

General inference technique: doesn’t care what is in the program

- All programming constructs (loops, conditions, …) 👍
- All distributions (continuous and discrete) 👍
- Finite and infinite distribution traces

Properties of importance sampling

General inference technique: doesn’t care what is in the program

- All programming constructs (loops, conditions, …) 👍
- All distributions (continuous and discrete) 👍
- Finite and infinite distribution traces 👍

Properties of importance sampling

Importance sampling degrades poorly as the dimension of the latent variables increases, unless we have a very well-chosen proposal distribution $q(x)$

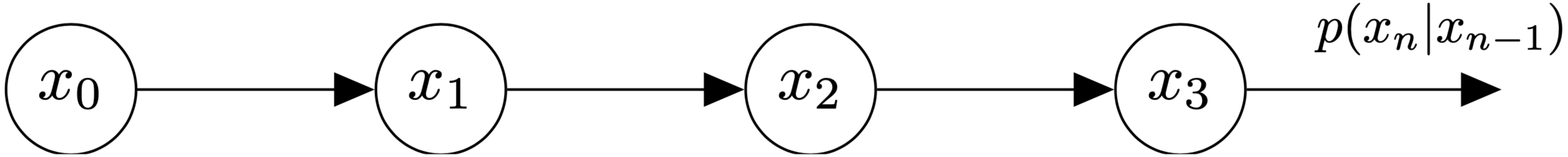
If the posterior distribution is 'peaky', we need a lot of luck to end up in the high-probability region

Probabilistic inference
Metropolis-Hastings MCMC

Metropolis-Hastings

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

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



Metropolis-Hastings

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

One step = one sample (execution)

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

$$x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \xrightarrow{p(x_n|x_{n-1})}$$

Metropolis-Hastings

Use proposal distribution to make **local** changes to the latent variables (the trace).

$q(x'|x)$ then defines a conditional distribution over x' given a current value x

Do we keep the new trace?

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

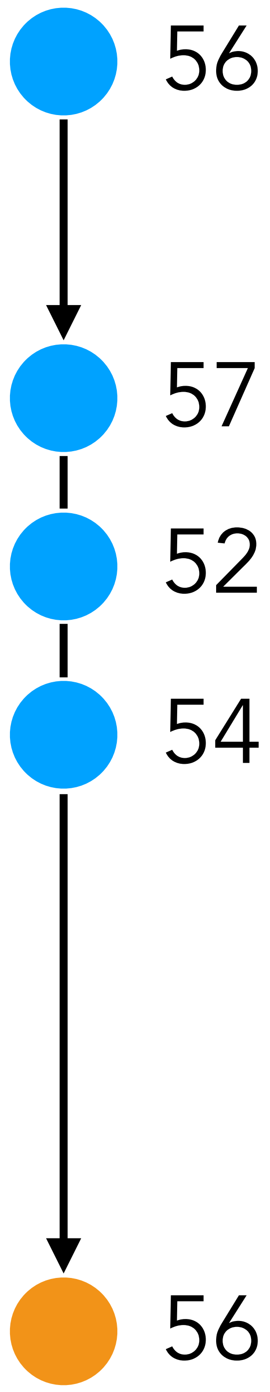
Yes, with probability *A*

Metropolis-Hastings

Use proposal distribution to make **local** changes to the latent variables (the trace).

$q(x'|x)$ then defines a conditional distribution over x' given a current value x

Generate the initial trace with e.g. IS



Do we keep the new trace?

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

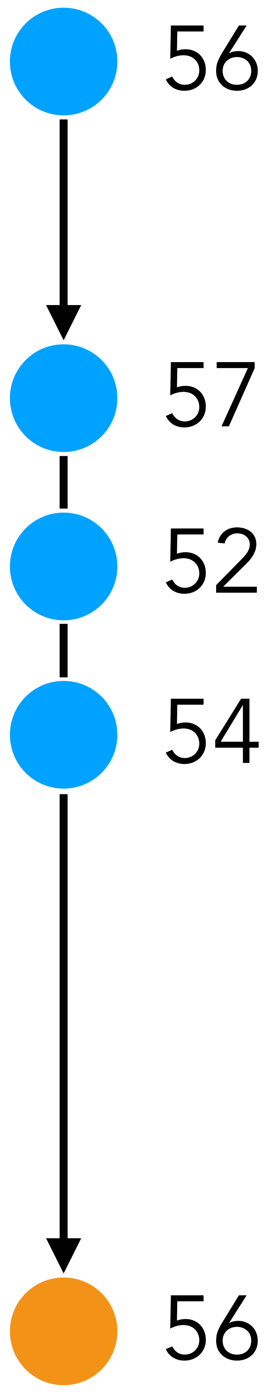
Yes, with probability A

Metropolis-Hastings

Use proposal distribution to make **local** changes to the latent variables (the trace).

$q(x'|x)$ then defines a conditional distribution over x' given a current value x

Generate the initial trace with e.g. IS



Pick a variable to modify

Do we keep the new trace?

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

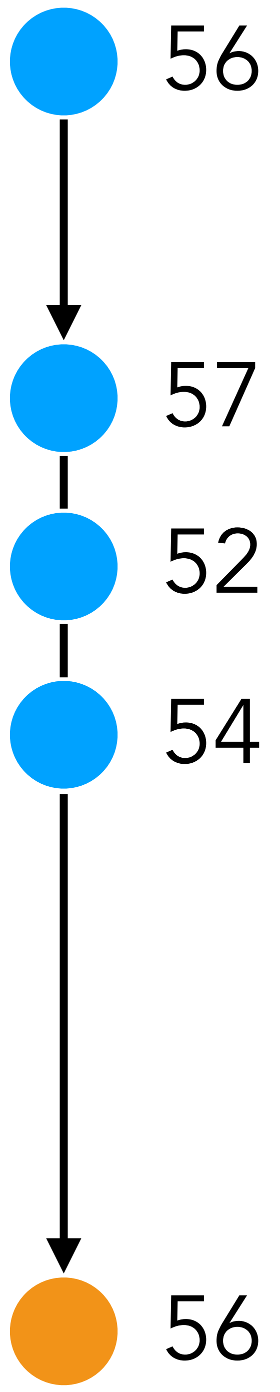
Yes, with probability *A*

Metropolis-Hastings

Use proposal distribution to make **local** changes to the latent variables (the trace).

$q(x' \mid x)$ then defines a conditional distribution over x' given a current value x

Generate the initial trace with e.g. IS



Pick a variable to modify

Modify the value by e.g. adding a small amount of noise

52 + Normal(0,2)

Do we keep the new trace?

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

Yes, with probability *A*

Metropolis-Hastings

Use proposal distribution to make **local** changes to the latent variables (the trace).

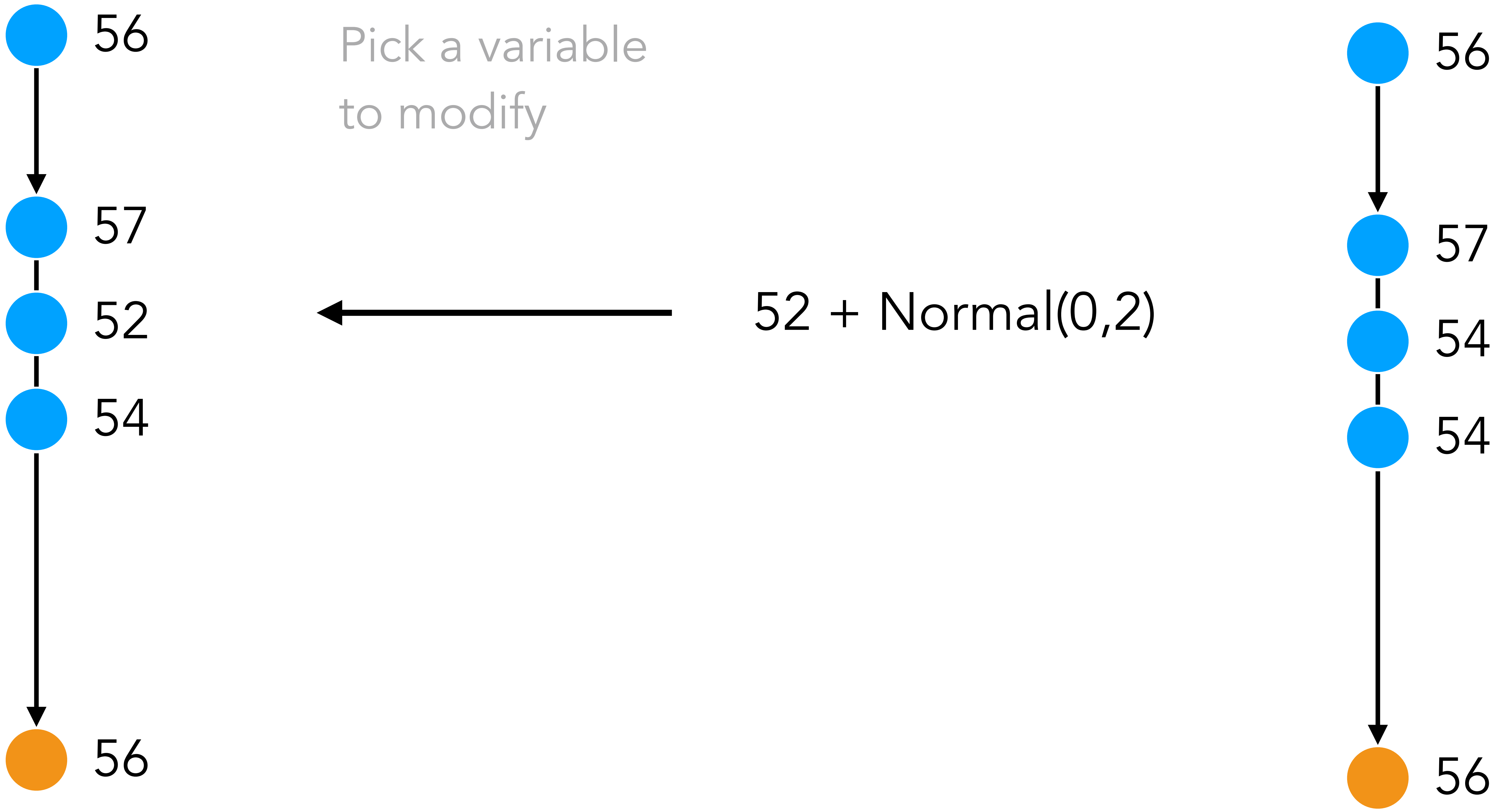
$q(x' \mid x)$ then defines a conditional distribution over x' given a current value x

Generate the initial trace with e.g. IS

Pick a variable to modify

Modify the value by e.g. adding a small amount of noise

New trace

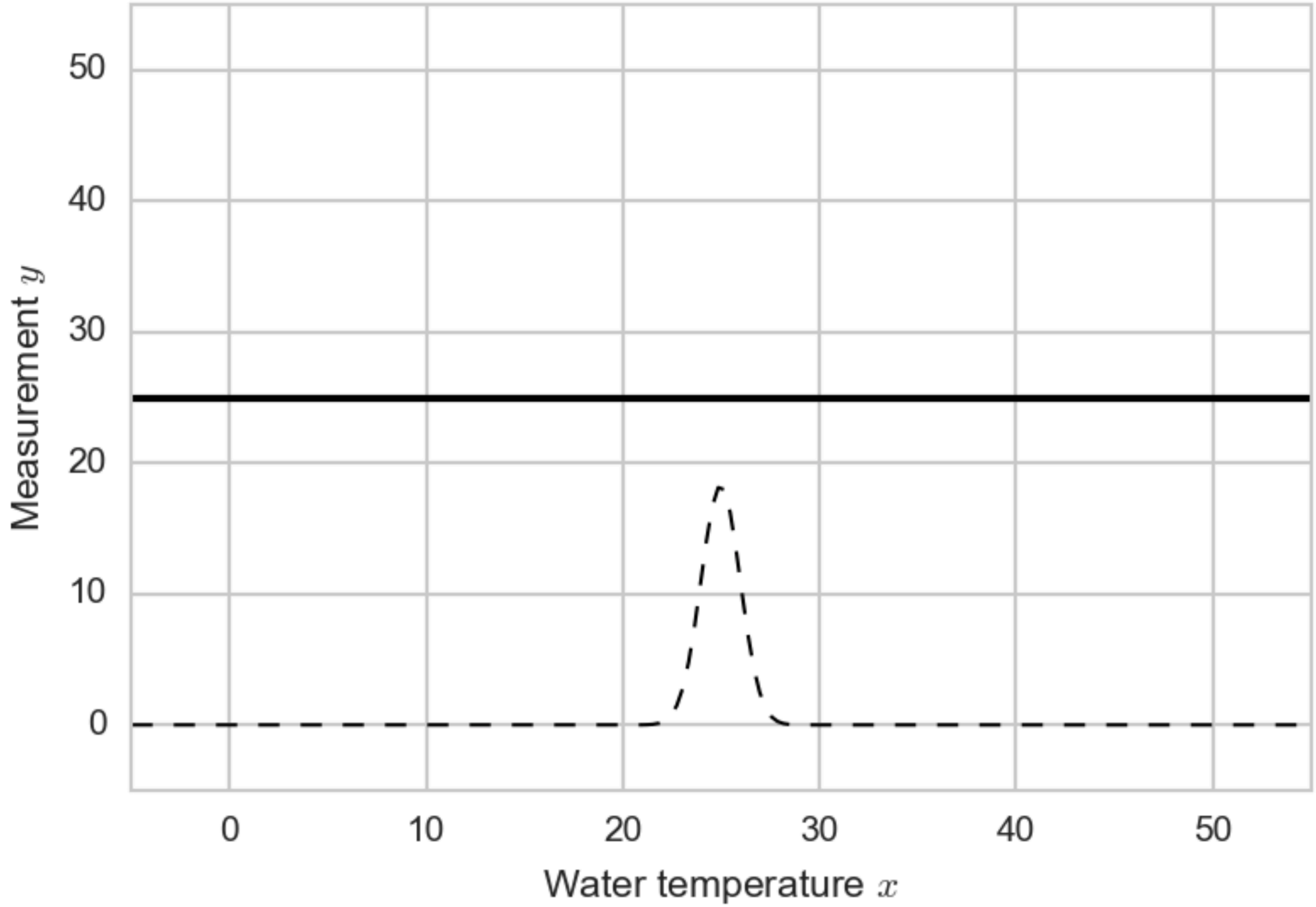


Do we keep the new trace?

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

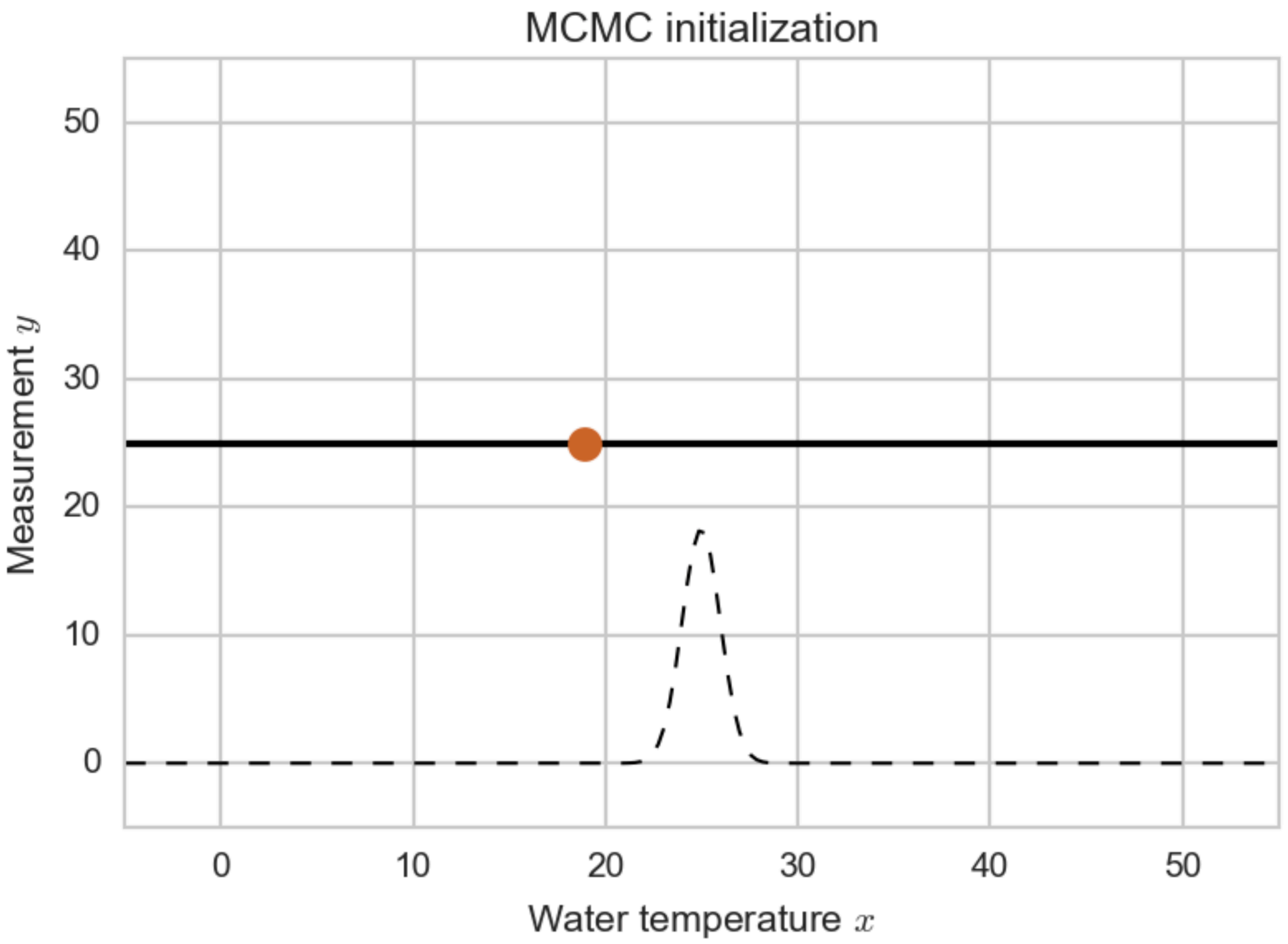
Yes, with probability *A*

Metropolis-Hastings: an illustration



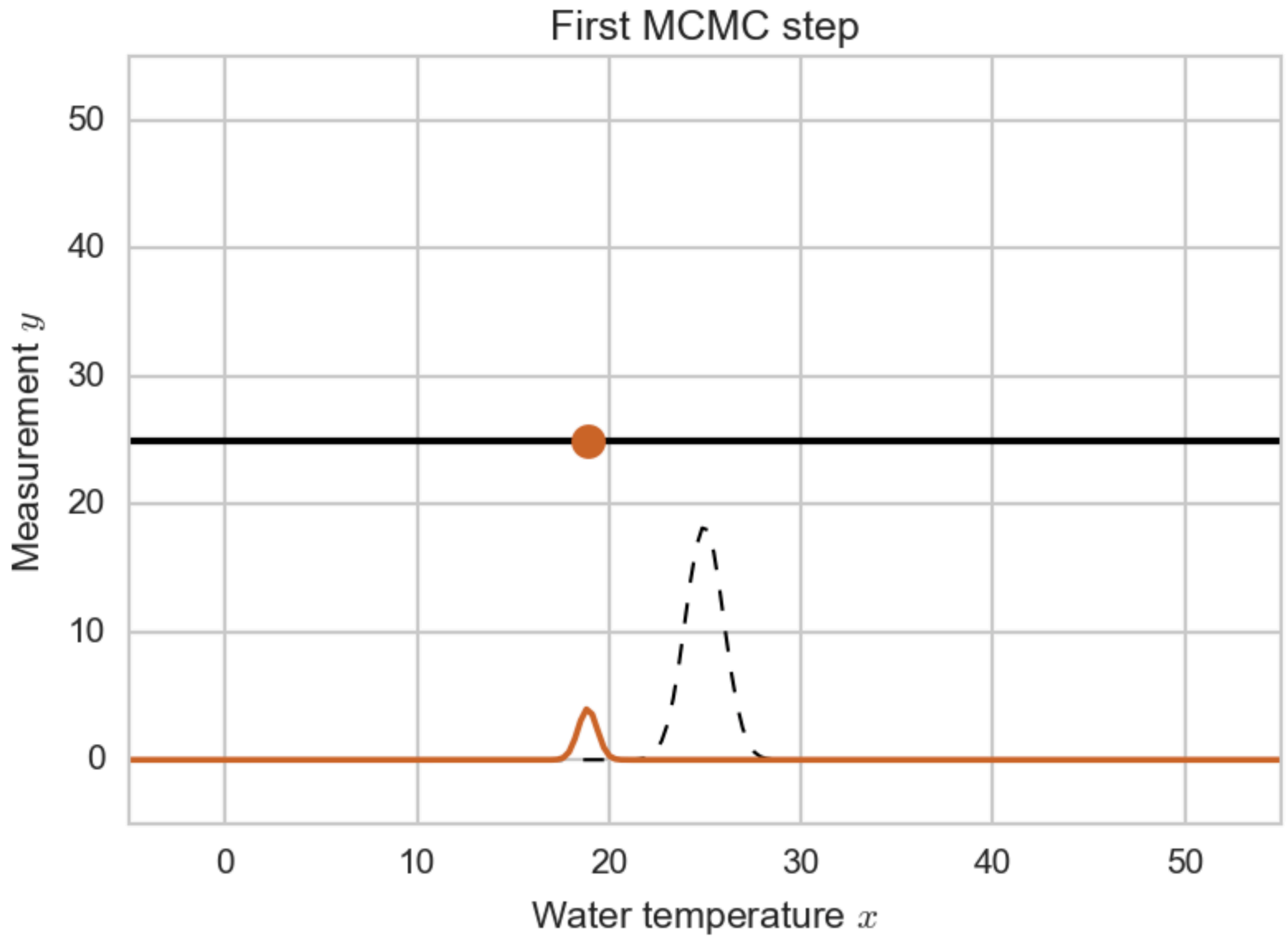
The (unnormalized) joint distribution *p(x,y)*
is shown as a dashed line

Metropolis-Hastings: an illustration



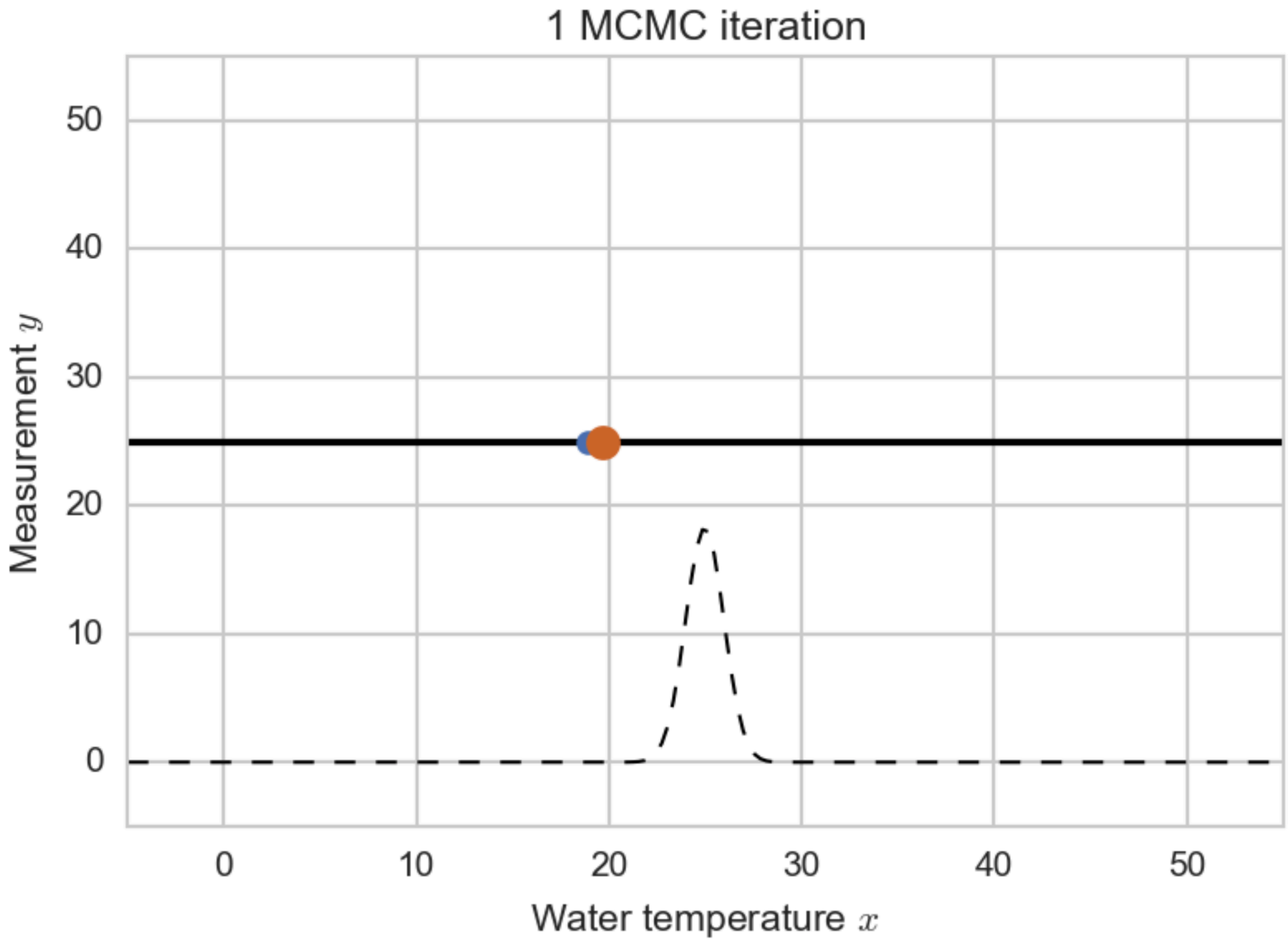
Initialize arbitrarily (e.g. with a sample from the prior)

Metropolis-Hastings: an illustration



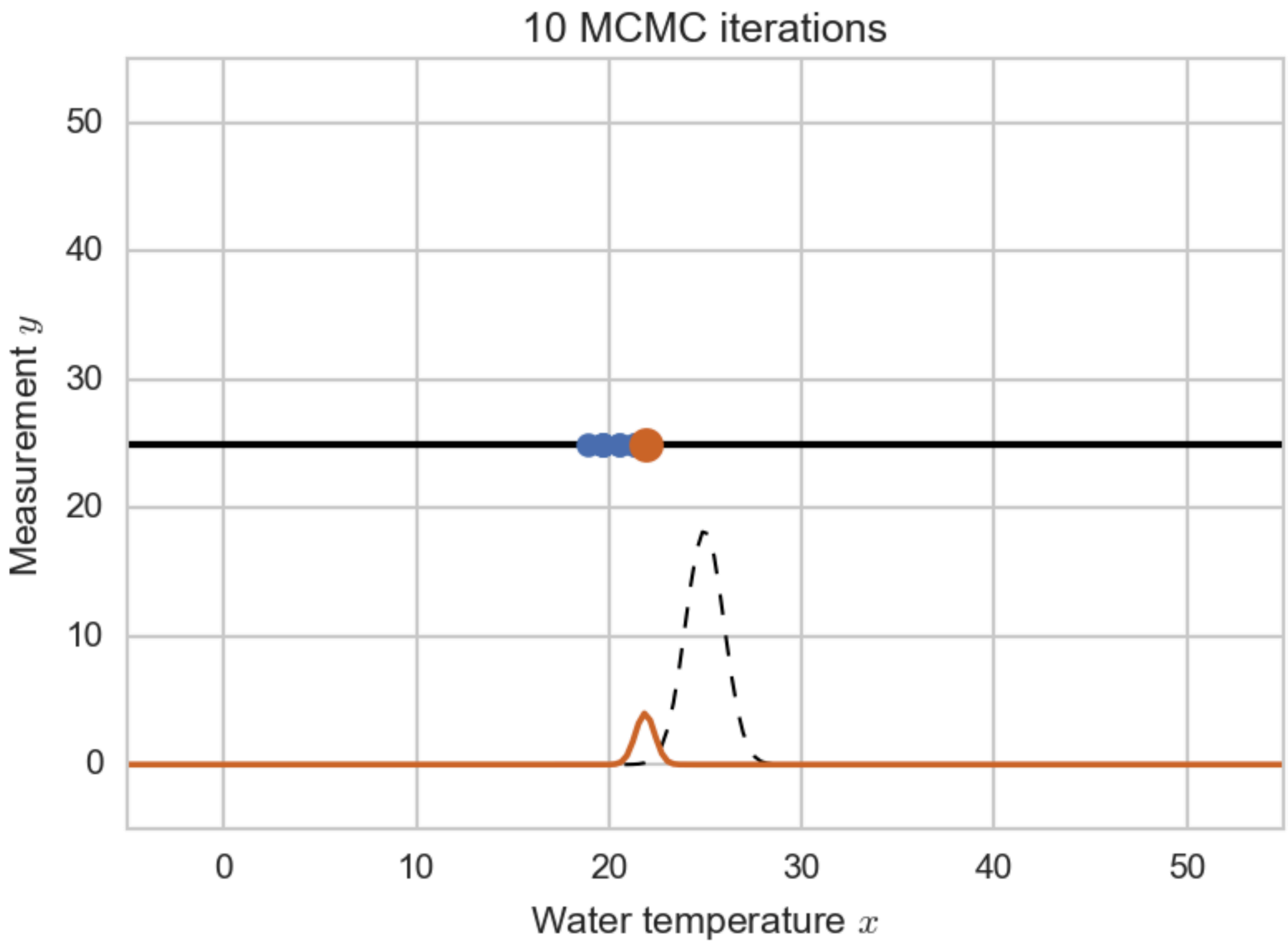
Propose a local move on *x* from a transition distribution

Metropolis-Hastings: an illustration



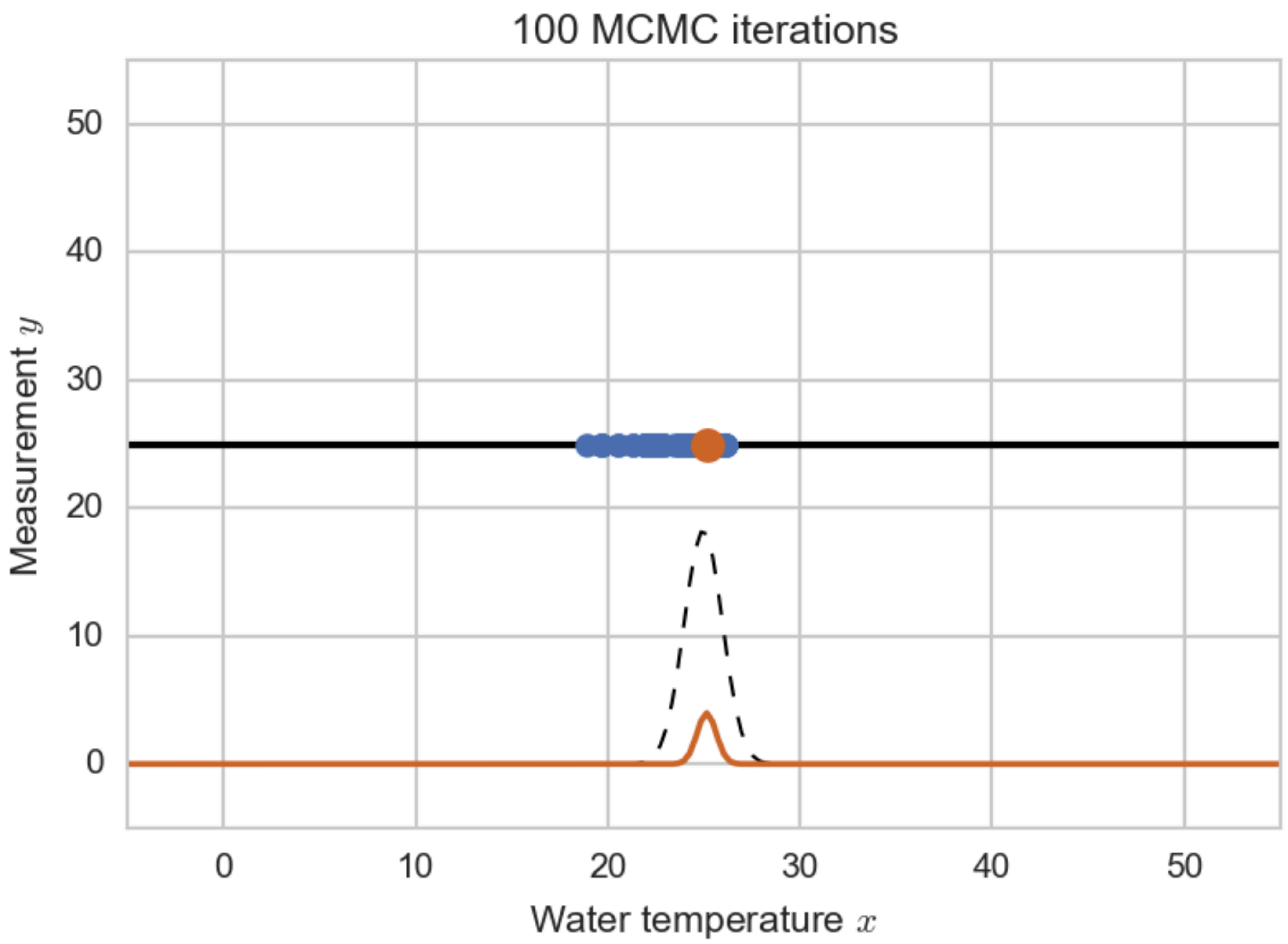
Here, we proposed a point in a region of higher probability density, and accepted

Metropolis-Hastings: an illustration



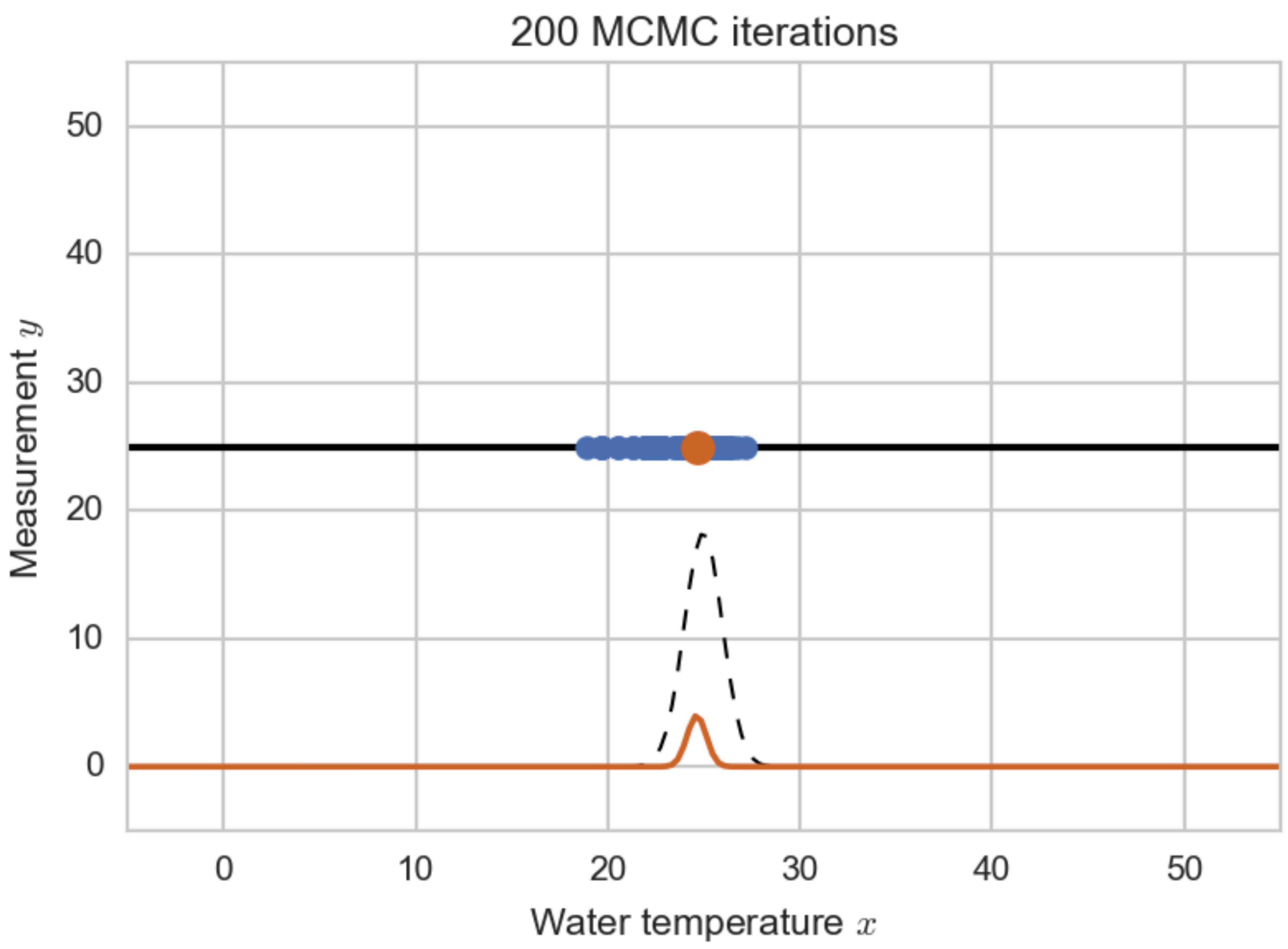
Continue: propose a local move, and accept or reject.
At first, this will look like a stochastic search algorithm!

Metropolis-Hastings: an illustration



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

Metropolis-Hastings: an illustration



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

Metropolis-Hastings MCMC: why can we re-weight?

The main technical requirement for MCMC is that the transition kernel leaves the posterior invariant

If we sample $X \sim p(X|Y)$ and then generate a new sample $X' \sim q(X'|X, Y)$ from the transition kernel, X and X' come from the same distribution

It is sufficient that the kernel satisfies the *detailed balance* criteria

$$q(X'|X, Y)p(X|Y) = q(X|X', Y)p(X'|Y)$$

We have to be able to go back to X from X'

Acceptance criterion ensures that!

Properties of Metropolis-Hastings

General inference technique: doesn't care what is in the program

- All programming constructs (loops, conditions, …)
- All distributions (continuous and discrete)
- Finite and infinite distribution traces

Properties of Metropolis-Hastings

General inference technique: doesn't care what is in the program

- All programming constructs (loops, conditions, …) 
- All distributions (continuous and discrete)
- Finite and infinite distribution traces

Properties of Metropolis-Hastings

General inference technique: doesn't care what is in the program

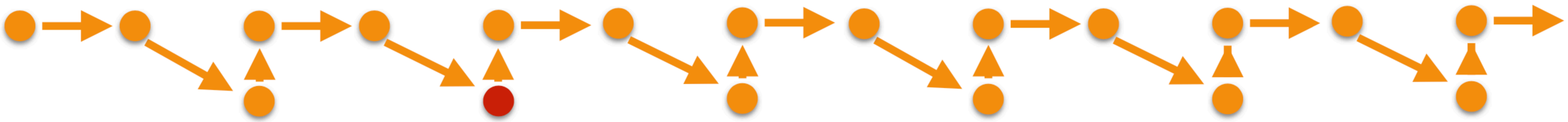
- All programming constructs (loops, conditions, …) 👍
- All distributions (continuous and discrete) 👍
- Finite and infinite distribution traces

Properties of Metropolis-Hastings

General inference technique: doesn't care what is in the program

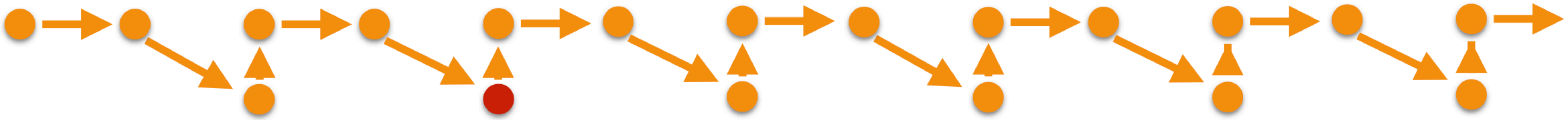
- All programming constructs (loops, conditions, …)
- All distributions (continuous and discrete)
- Finite and infinite distribution traces

Metropolis-Hastings: computational efficiency

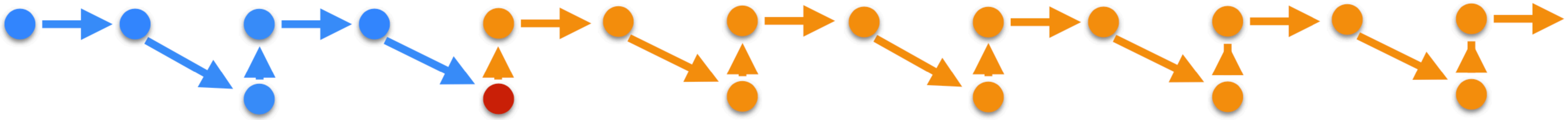


D. Wingate, A. Stuhlmueller, and N. D. Goodman.
"Lightweight implementations of probabilistic programming languages via transformational compilation." AISTATS (2011).

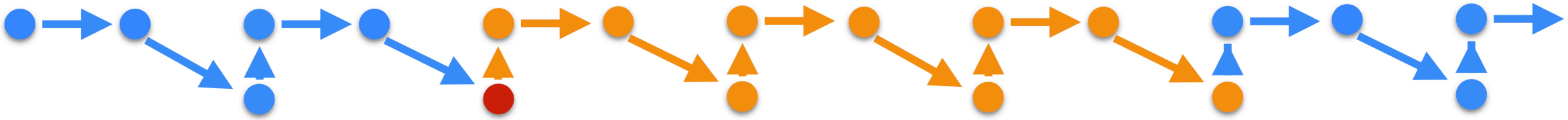
Metropolis-Hastings: computational efficiency



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).

Metropolis-Hastings: properties

Metropolis-Hastings: properties

Metropolis-Hastings: properties

✓ Makes small changes to traces

✗

Metropolis-Hastings: properties

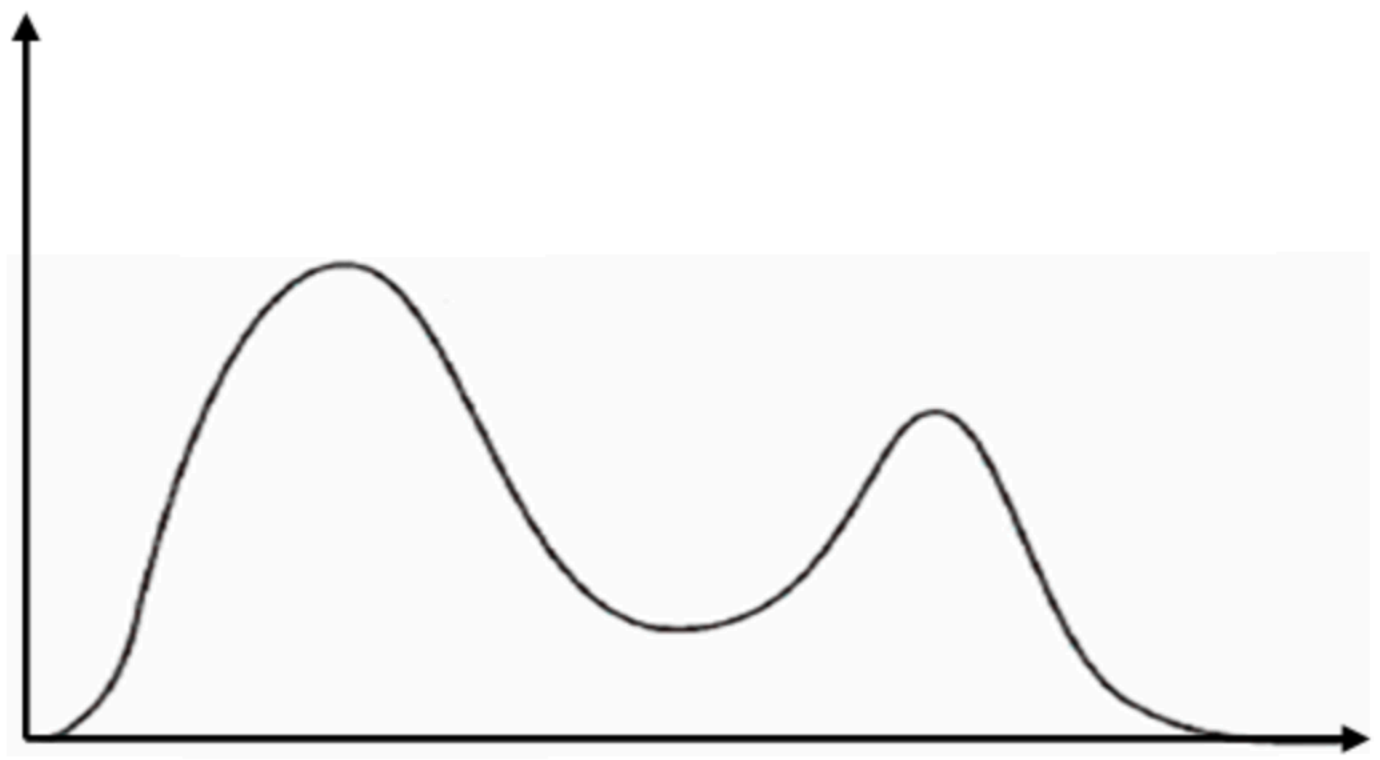
Makes small changes to traces

Gradually goes to better traces

Metropolis-Hastings: properties

Makes small changes to traces

Gradually goes to better traces



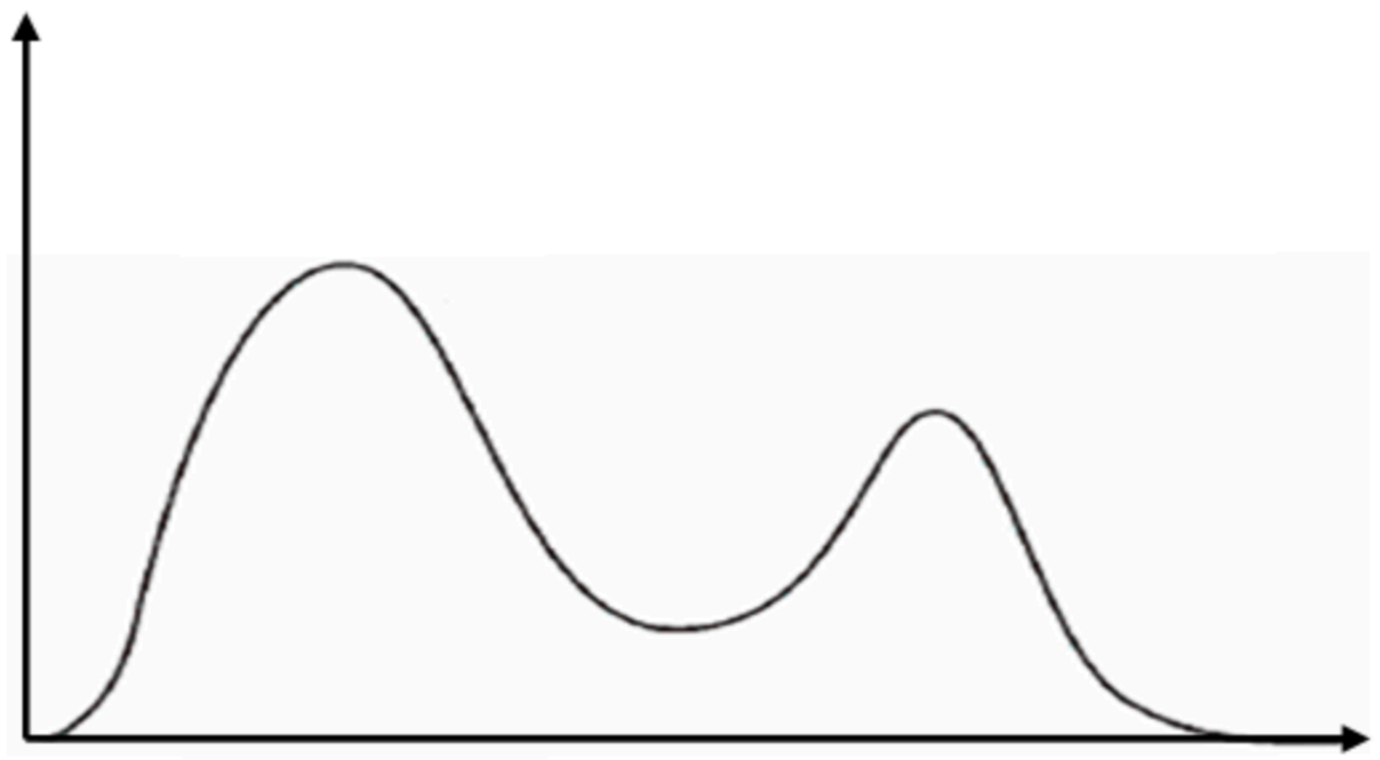
Metropolis-Hastings: properties

Makes small changes to traces

Gradually goes to better traces

It might be difficult to capture a complex distribution in small steps

Especially when choices are correlated



Probabilistic inference
Metropolis-Hastings MCMC

Importance sampling: makes all choices at once

Metropolis-Hastings: modify one choice at a time

Importance sampling: makes all choices at once

Metropolis-Hastings: modify one choice at a time

Can we do better?

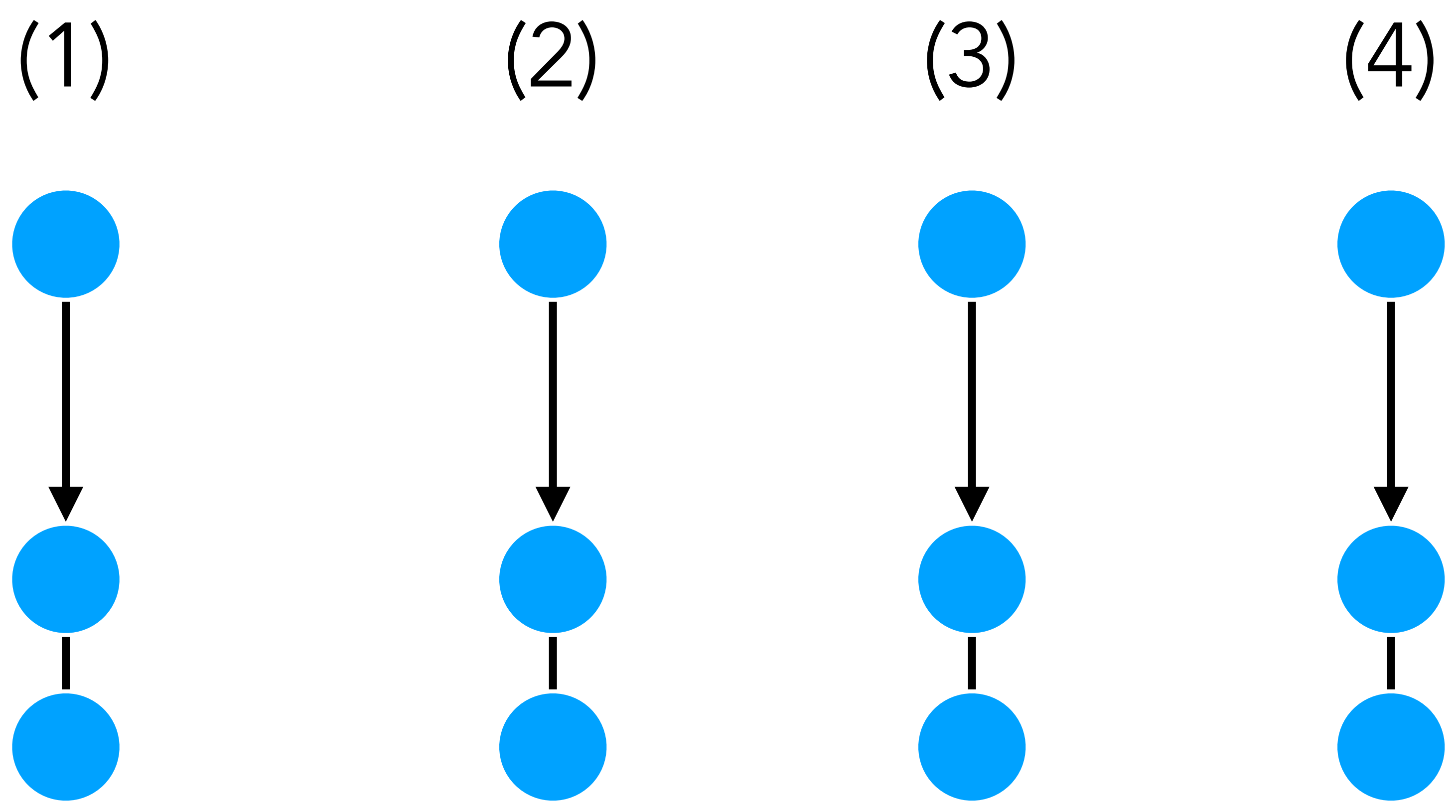
Particle filters

Particle filters

(1) (2) (3) (4)

Initialise N traces/programs

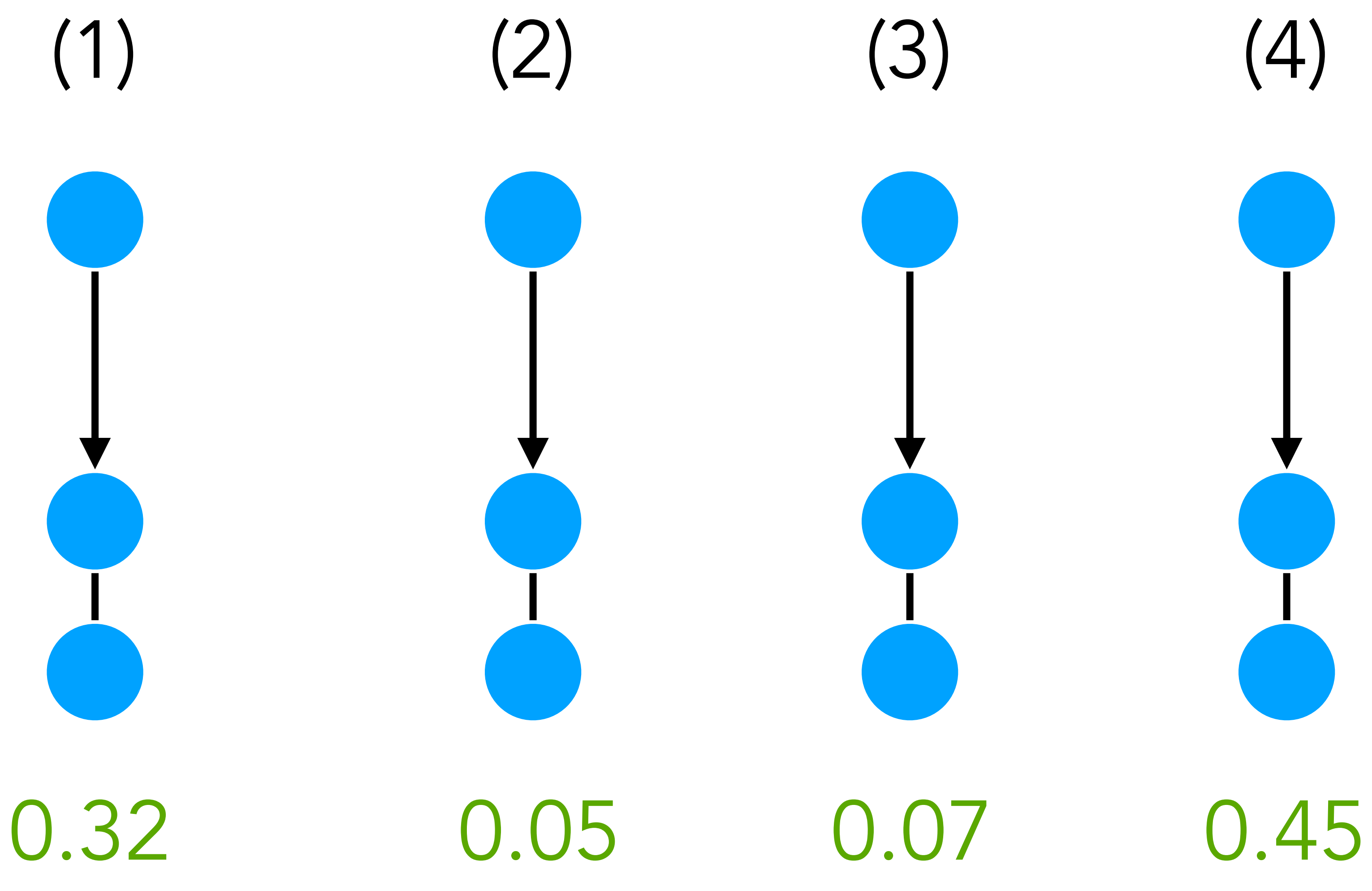
Particle filters



Initialise N traces/programs

Run them until the first observe statement

Particle filters

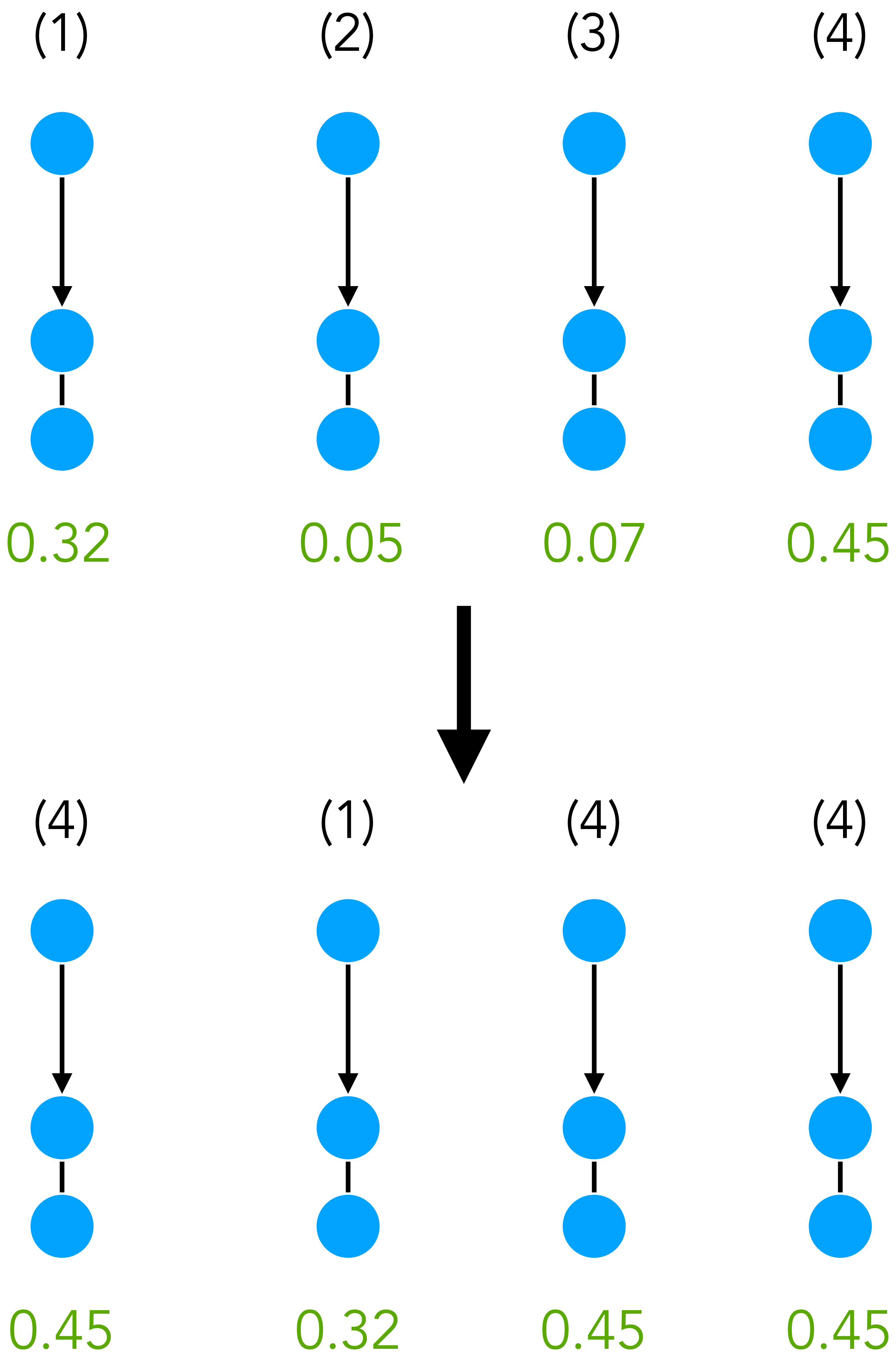


Initialise N traces/programs

Run them until the first observe statement

And check how well they match the observations

Particle filters



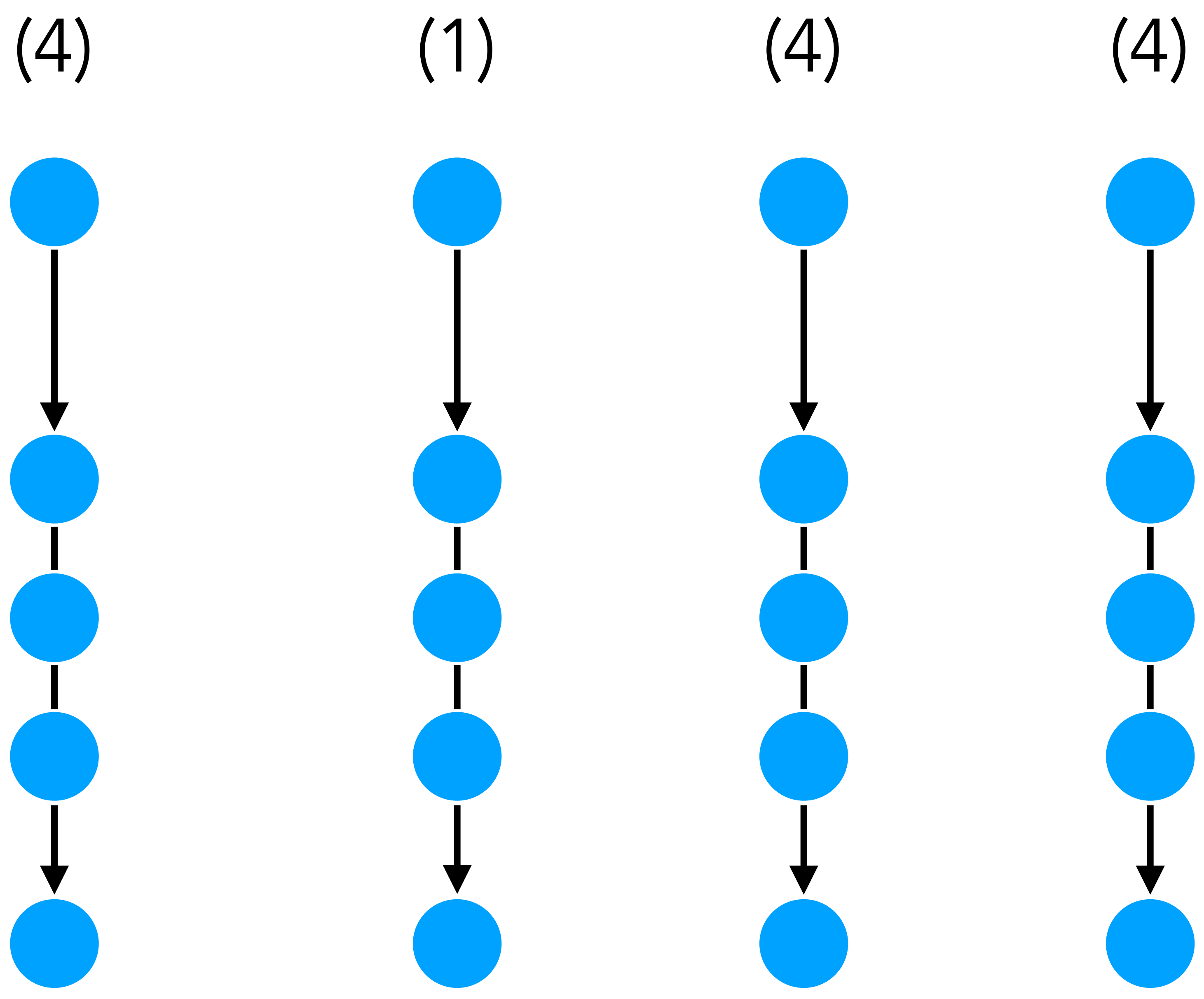
Initialise N traces/programs

Run them until the first observe statement

And check how well they match the observations

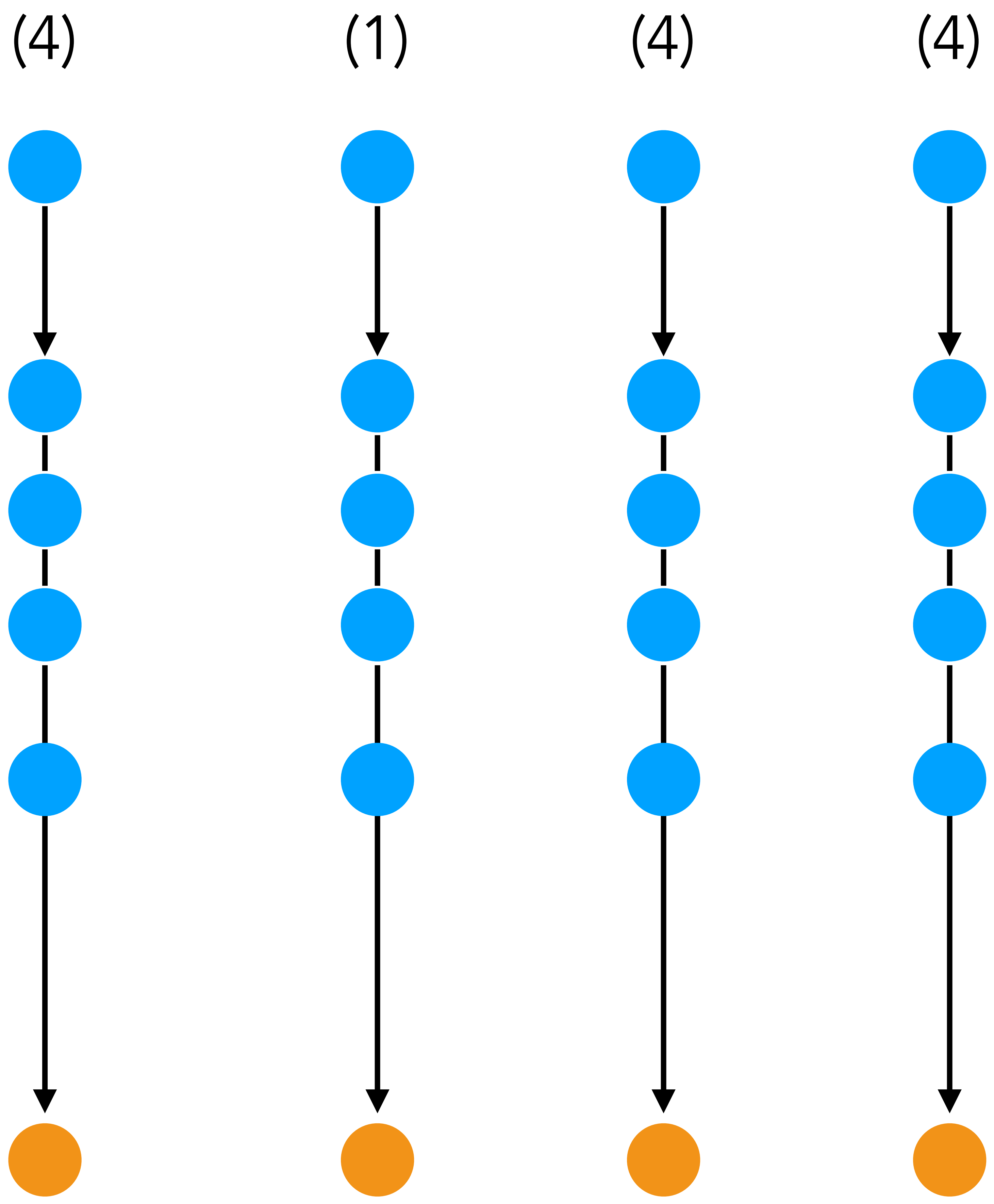
Then, resample **traces** with replacement proportional to how well they match observations

Particle filters



Run until the next observe and resample

Particle filters



Run until the next observe and resample

Continue until each program is finished

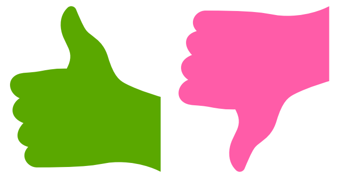
Properties of Particle filtering

General inference technique: doesn’t care what is in the program

- All programming constructs (loops, conditions, …)
- All distributions (continuous and discrete)
- Finite and infinite distribution traces

Properties of Particle filtering

General inference technique: doesn't care what is in the program

- All programming constructs (loops, conditions, …) 
- All distributions (continuous and discrete)
- Finite and infinite distribution traces

Properties of Particle filtering

General inference technique: doesn’t care what is in the program

- All programming constructs (loops, conditions, …)  
- All distributions (continuous and discrete) 
- Finite and infinite distribution traces

Properties of Particle filtering

General inference technique: doesn’t care what is in the program

- All programming constructs (loops, conditions, …) 👍 👎
- All distributions (continuous and discrete) 👍
- Finite and infinite distribution traces 👍 👎

Particle filters

Particle filters

Particle filters

Particle filters

✓ Good mixture between IS and MH

✗

Particle filters

Good mixture between IS and MH

Often very good solution to complex programs

Particle filters

Good mixture between IS and MH

Often very good solution to complex programs

Doesn’t not support every programs

Particle filters

Good mixture between IS and MH

Often very good solution to complex programs

Doesn’t not support every programs

What to do if all initial samples are bad?

Particle filters

Good mixture between IS and MH

Often very good solution to complex programs

Doesn’t not support every programs

What to do if all initial samples are bad?

→ Particle filters with rejuvenation (perform Metropolis-Hastings on the traces before sampling)

Summary

Calculating $p(x, y)$ exactly is not possible for non-toy problems

We have to rely on Monte Carlo approximations

Inference procedures need to be able to handle *any* kind of program

Importance sampling

Metropolis-Hastings MCMC

Particle filtering