Probabilistic Programming Languages

Guillaume Baudart

SIESTE - ENS Lyon

Probabilistic programming languages

Probabilistic programming languages

General purpose programming languages extended with probabilistic constructs

- `sample`: draw a sample from a distribution
- `assume`, `factor`, `observe`: condition the model on inputs (e.g., observed data)
- `infer`: compute the posterior distribution of a model given the inputs

Probabilistic programming languages

General purpose programming languages extended with probabilistic constructs

- `sample`: draw a sample from a distribution
- `assume`, `factor`, `observe`: condition the model on inputs (e.g., observed data)
- `infer`: compute the posterior distribution of a model given the inputs

Multiple examples:

- Church, Anglican (lisp, clojure), 2008
- WebPPL (javascript), 2014
- Pyro/NumPyro (python), 2017/2019
- Gen (julia), 2018
- ProbZelus (Zelus), 2019
- ...

Probabilistic programming languages

General purpose programming languages extended with probabilistic constructs

- `sample`: draw a sample from a distribution
- `assume`, `factor`, `observe`: condition the model on inputs (e.g., observed data)
- `infer`: compute the posterior distribution of a model given the inputs

Multiple examples:

- Church, Anglican (lisp, clojure), 2008
- WebPPL (javascript), 2014
- Pyro/NumPyro (python), 2017/2019
- Gen (julia), 2018
- ProbZelus (Zelus), 2019
- ...

More and more, incorporating new ideas:

- New inference techniques, e.g., stochastic variational inference (SVI)
- Interaction with neural nets (deep probabilistic programming)

Demo: mu-ppl

Bayesian reasoning

$$p(x \mid y_1, \ldots, y_n) = \frac{p(x)\, p(y_1, \ldots, y_n \mid x)}{p(y_1, \ldots, y_n)} \qquad \text{(Bayes' theorem)}$$

$$\propto p(x)\, p(y_1, \ldots, y_n \mid x) \qquad \text{(Data are constants)}$$

Thomas Bayes (1701-1761)

https://en.wikipedia.org/wiki/Thomas_Bayes

Bayesian reasoning

Bayesian Inference: learn parameters from data

- Latent parameter x
- Observed data $y_1, \ldots, y_n$

$$p(x \mid y_1, \ldots, y_n) = \frac{p(x)\, p(y_1, \ldots, y_n \mid x)}{p(y_1, \ldots, y_n)} \qquad \text{(Bayes' theorem)}$$

$$\propto p(x)\, p(y_1, \ldots, y_n \mid x) \qquad \text{(Data are constants)}$$

Thomas Bayes (1701-1761)

https://en.wikipedia.org/wiki/Thomas_Bayes

Bayesian reasoning

Bayesian Inference: learn parameters from data

- Latent parameter x
- Observed data $y_1, \dots, y_n$

$$\underbrace{p(x \mid y_1, \dots, y_n)}_{\text{posterior}} = \frac{p(x)\, p(y_1, \dots, y_n \mid x)}{p(y_1, \dots, y_n)} \qquad \text{(Bayes' theorem)}$$

$$\propto \underbrace{p(x)}_{\text{prior}} \, \underbrace{p(y_1, \dots, y_n \mid x)}_{\text{likelihood}} \qquad \text{(Data are constants)}$$

Thomas Bayes (1701-1761)

https://en.wikipedia.org/wiki/Thomas_Bayes

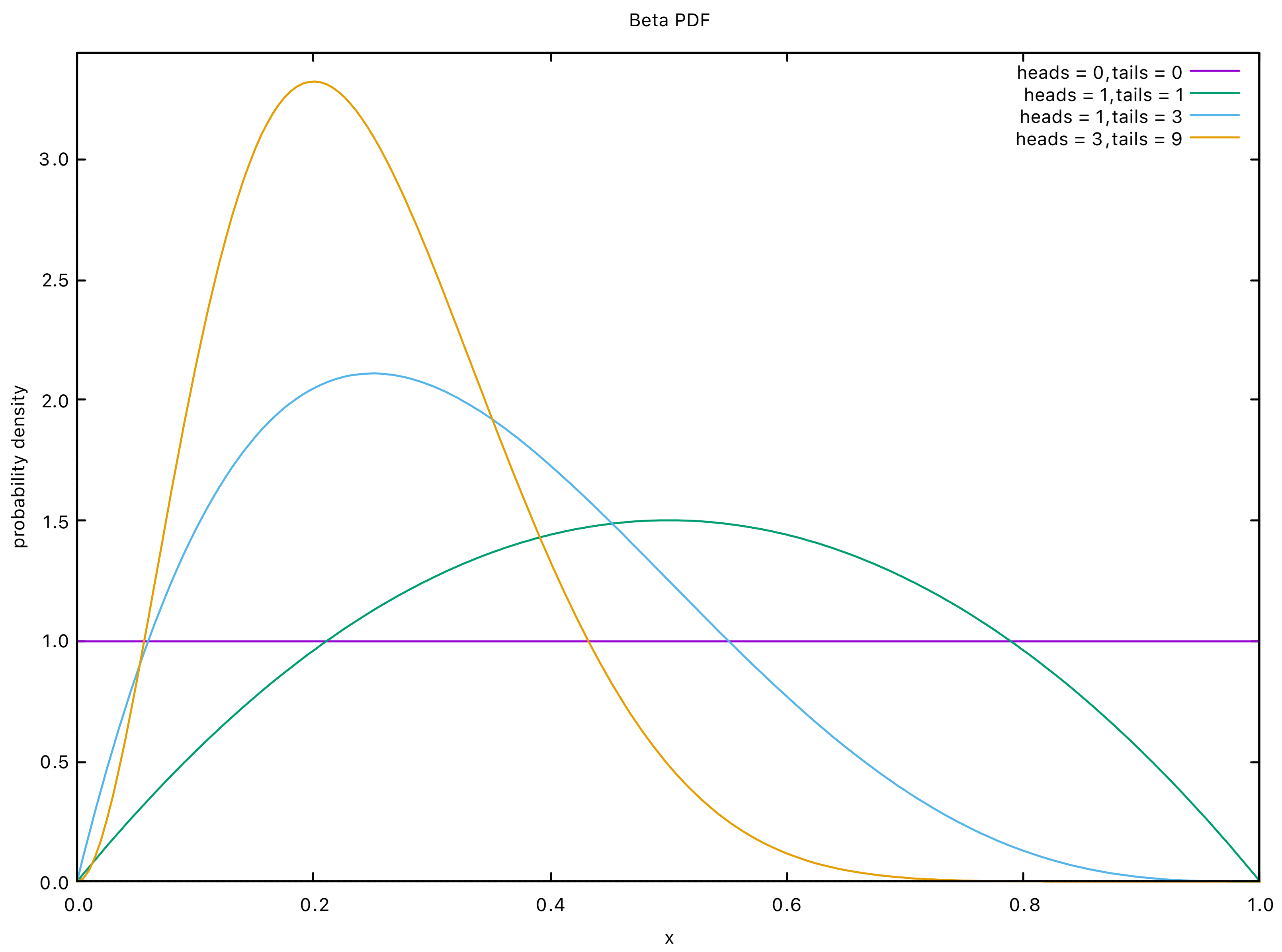
Example: coin

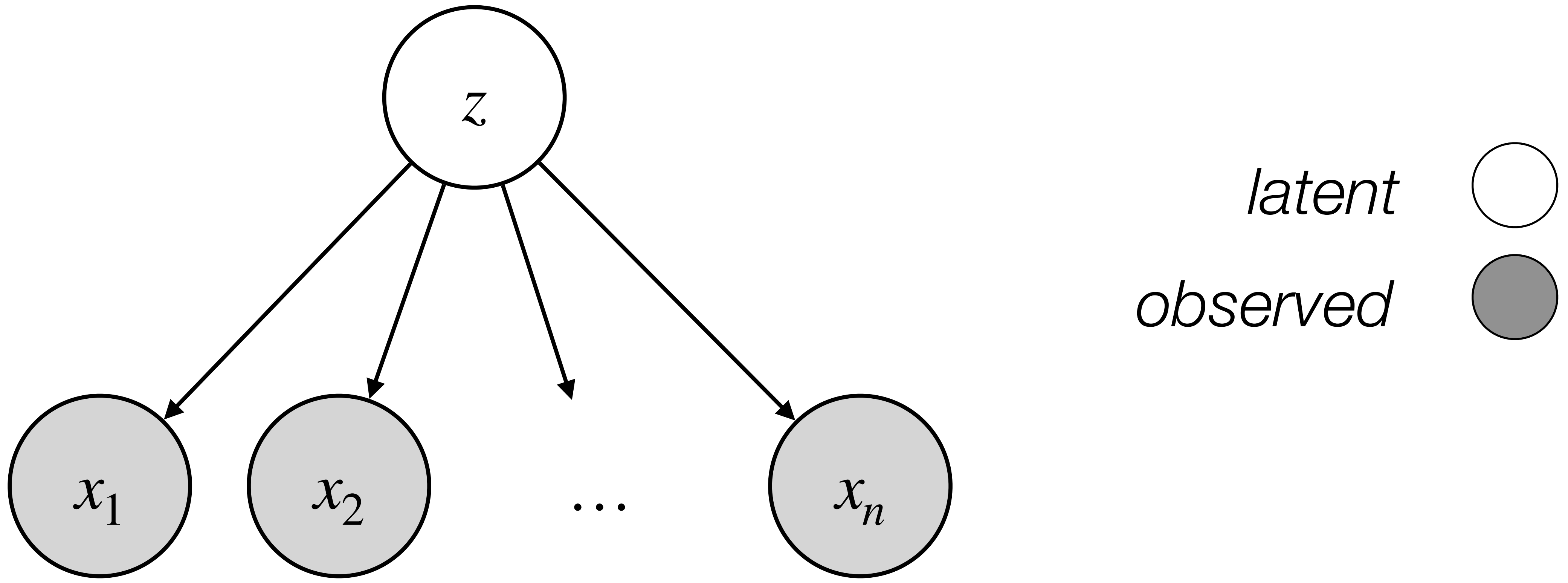
Consider a series of coin tosses

- Observations: head or tail
- Each toss is independant
- What is the probability of getting head at the next toss?

Probabilistic model

- Prior: $z \sim \mathit{Uniform}(0, 1)$
- Observations: for $i \in [1, n]$, $x_i \sim \mathit{Bernoulli}(z)$
- Posterior: $p(z \mid x_1, \ldots, x_n)$?





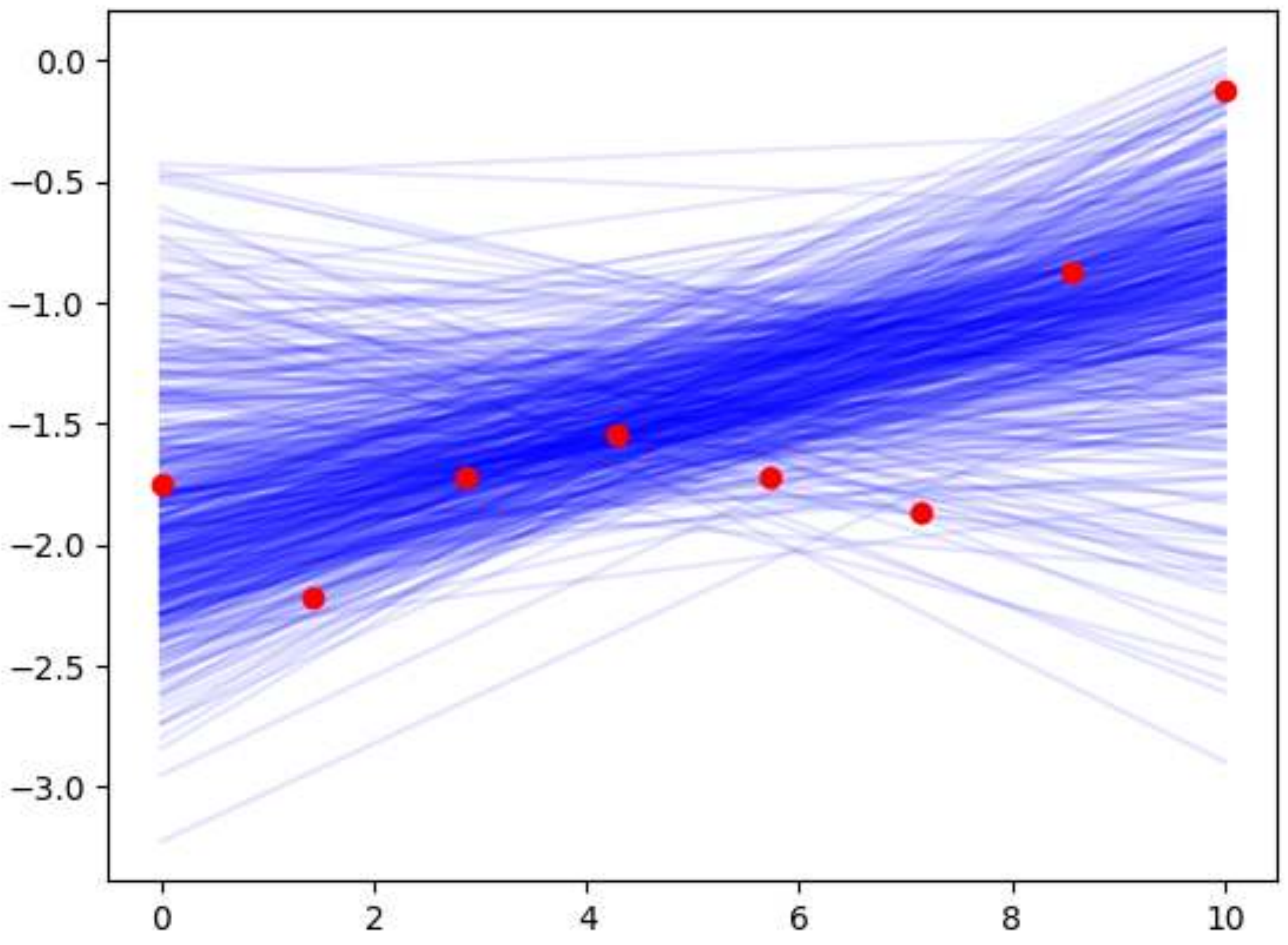
Example: linear regression

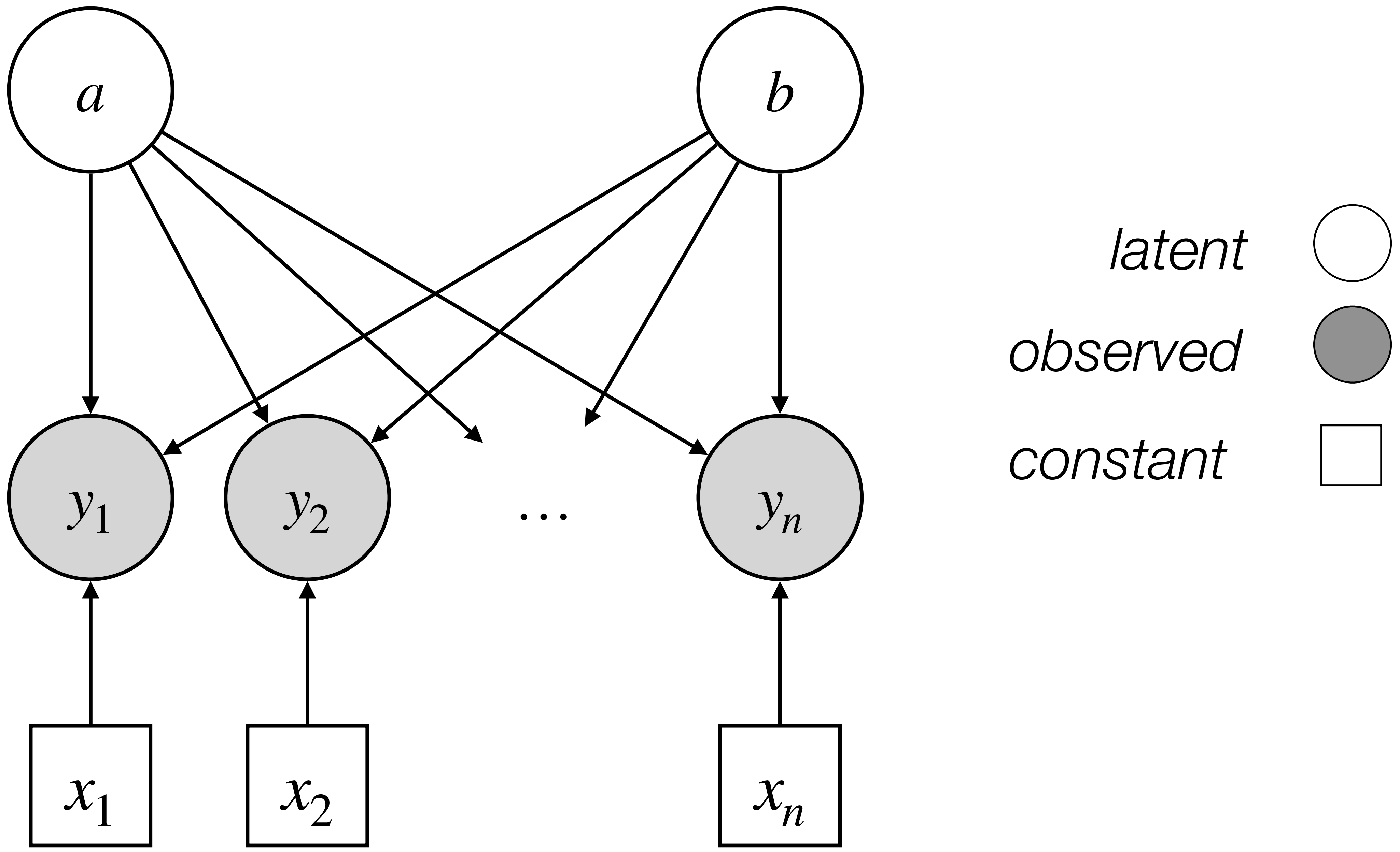
Consider a series of observations

- Observation: point (x, y)
- Each point is independent from others
- Find the distribution of possible regressions

Probabilistic model

- Prior: $a \sim \mathcal{N}(0,1)$, and $b \sim \mathcal{N}(0,1)$
- Observations: for $i \in [1,n]$, $y_i \sim \mathcal{N}(a \times x_i + b, \sigma)$
- Posterior: $p(a, b \mid (x_1, y_1) \ldots, (x_n, y_n))$?





Probabilistic programming

$$p(x \mid y_1, \ldots, y_n) = \frac{p(x)\, p(y_1, \ldots, y_n \mid x)}{p(y_1, \ldots, y_n)} \qquad \text{(Bayes' theorem)}$$

$$\propto p(x)\, p(y_1, \ldots, y_n \mid x) \qquad \text{(Data are constants)}$$

Thomas Bayes (1701-1761)

https://en.wikipedia.org/wiki/Thomas_Bayes

Probabilistic programming

Bayesian Inference: learn parameters from data

- Latent parameter x
- Observed data $y_1, \dots, y_n$

$$\underbrace{p(x \mid y_1, \dots, y_n)}_{\text{posterior}} = \frac{p(x)\, p(y_1, \dots, y_n \mid x)}{p(y_1, \dots, y_n)} \qquad \text{(Bayes' theorem)}$$

$$\propto \underbrace{p(x)}_{\text{prior}}\, \underbrace{p(y_1, \dots, y_n \mid x)}_{\text{likelihood}} \qquad \text{(Data are constants)}$$

Thomas Bayes (1701-1761)

https://en.wikipedia.org/wiki/Thomas_Bayes

Probabilistic programming

Bayesian Inference: learn parameters from data

- Latent parameter x
- Observed data $y_1, \ldots, y_n$

$$\underbrace{p(x \mid y_1, \ldots, y_n)}_{\text{posterior}} = \frac{p(x)\, p(y_1, \ldots, y_n \mid x)}{p(y_1, \ldots, y_n)} \qquad \text{(Bayes' theorem)}$$

$$\propto \underbrace{p(x)}_{\text{prior}}\, \underbrace{p(y_1, \ldots, y_n \mid x)}_{\text{likelihood}} \qquad \text{(Data are constants)}$$

Probabilistic constructs

- `x = sample(d)`: introduce a random variable `x` of distribution `d`
- `observe(d, y)`: condition on the fact that `y` was sampled from `d`
- `infer(m, y)`: compute posterior distribution of `m` given `y`

Thomas Bayes (1701-1761)

https://en.wikipedia.org/wiki/Thomas_Bayes

Probabilistic programming

Bayesian Inference: learn parameters from data

- Latent parameter x
- Observed data $y_1, \ldots, y_n$

Notation: $x \sim \mu$

- parameter (output): `sample(mu)`
- observation (input): `observe(mu)`

$$p(x \mid y_1, \ldots, y_n) = \frac{p(x)\, p(y_1, \ldots, y_n \mid x)}{p(y_1, \ldots, y_n)} \quad \text{(Bayes' theorem)}$$

posterior

$$\propto p(x)\, p(y_1, \ldots, y_n \mid x) \quad \text{(Data are constants)}$$

prior *likelihood*

Thomas Bayes (1701-1761)

Probabilistic constructs

- `x = sample(d)`: introduce a random variable `x` of distribution `d`
- `observe(d, y)`: condition on the fact that `y` was sampled from `d`
- `infer(m, y)`: compute posterior distribution of `m` given `y`

https://en.wikipedia.org/wiki/Thomas_Bayes

Probabilistic programming

Bayesian Inference: learn parameters from data

- Latent parameter x
- Observed data $y_1, \ldots, y_n$

$$p(x \mid y_1, \ldots, y_n) = \frac{p(x)\, p(y_1, \ldots, y_n \mid x)}{p(y_1, \ldots, y_n)} \qquad \text{(Bayes' theorem)}$$

posterior

$$\propto p(x)\, p(y_1, \ldots, y_n \mid x) \qquad \text{(Data are constants)}$$

prior likelihood

```
def model(y1, … , yn):
  x = sample prior
  observe (likelihood(x), (y1, … , yn))
  return x

infer(model, (y1, … , yn))
```

Thomas Bayes (1701-1761)

https://en.wikipedia.org/wiki/Thomas_Bayes

Probabilistic programming

A Bayesian model is describe by a program

Bayesian Inference: learn parameters from data

- Latent parameter x
- Observed data $y_1, \ldots, y_n$

$$\underbrace{p(x \mid y_1, \ldots, y_n)}_{\text{posterior}} = \frac{p(x)\, p(y_1, \ldots, y_n \mid x)}{p(y_1, \ldots, y_n)} \qquad \text{(Bayes' theorem)}$$

$$\propto \underbrace{p(x)}_{\text{prior}}\, \underbrace{p(y_1, \ldots, y_n \mid x)}_{\text{likelihood}} \qquad \text{(Data are constants)}$$

```
def model(y1, ... , yn):
  x = sample prior
  observe (likelihood(x), (y1, ... , yn))
  return x

infer(model, (y1, ... , yn))
```

Thomas Bayes (1701-1761)

https://en.wikipedia.org/wiki/Thomas_Bayes

Probabilistic programming

Probabilistic constructs

- `sample`: draw a sample from a distribution
- `assume`, `factor`, `observe`: condition the model on inputs (e.g., observed data)
- `infer`: compute the posterior distribution of a model given the inputs

More general than classic Bayesian reasoning

```
def weird() =
  b = sample(Bernoulli(0.5))
  mu = 0.5 if (b = 1) else 1.0
  theta = sample(Gaussian(mu, 1.0))
  if theta > 0.:
    observe (Gaussian(mu, 0.5), theta)
    return theta
  else:
    return weird ()
```

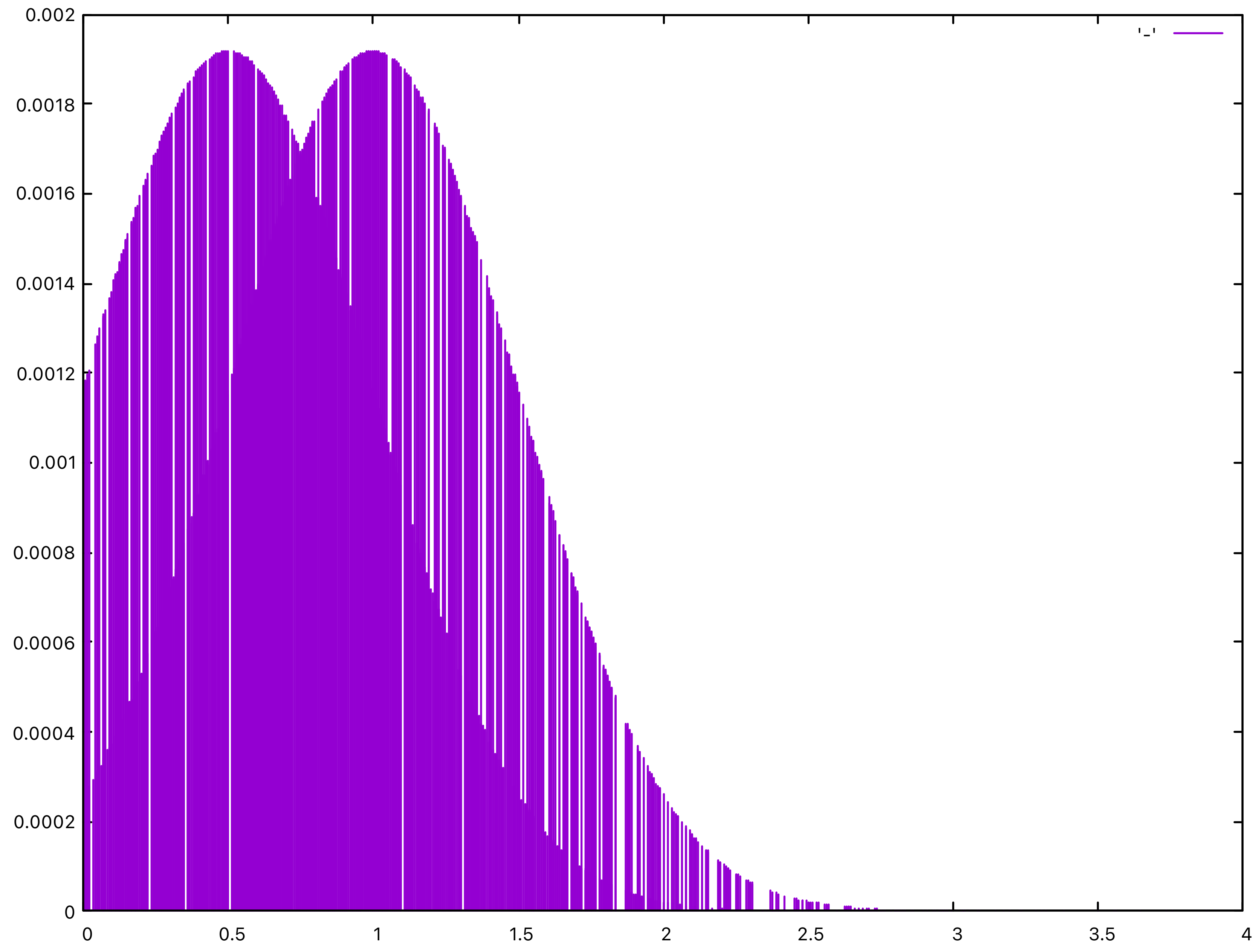
Probabilistic programming

Probabilistic constructs

- `sample`: draw a sample from a distribution
- `assume`, `factor`, `observe`: condition the model on inputs (e.g., observed data)
- `infer`: compute the posterior distribution of a model given the inputs

More general than classic Bayesian reasoning

```
def weird() =
  b = sample(Bernoulli(0.5))
  mu = 0.5 if (b = 1) else 1.0
  theta = sample(Gaussian(mu, 1.0))
  if theta > 0.:
    observe (Gaussian(mu, 0.5), theta)
    return theta
  else:
    return weird ()
```



Probabilistic programming

Probabilistic constructs

- `sample`: draw a sample from a distribution
- `assume`, `factor`, `observe`: condition the model on inputs (e.g., observed data)
- `infer`: compute the posterior distribution of a model given the inputs

More general than classic Bayesian reasoning

But: general case....

$$p(\theta \mid x_1, \ldots x_n) = \frac{p(\theta)\; p(x_1, \ldots, x_n \mid \theta)}{p(x_1, \ldots, x_n)}$$ (Bayes' theorem)

$$= \frac{p(\theta)\; p(x_1, \ldots, x_n \mid \theta)}{\int p(\theta)\; p(x_1, \ldots, x_n \mid \theta) d\theta}$$

Probabilistic programming

Probabilistic constructs

- `sample`: draw a sample from a distribution
- `assume`, `factor`, `observe`: condition the model on inputs (e.g., observed data)
- `infer`: compute the posterior distribution of a model given the inputs

More general than classic Bayesian reasoning

But: general case....

$$p(\theta \mid x_1, \ldots x_n) = \frac{p(\theta)\ p(x_1, \ldots, x_n \mid \theta)}{p(x_1, \ldots, x_n)}$$

(Bayes' theorem)

$$= \frac{\boxed{p(\theta)}\ \boxed{p(x_1, \ldots, x_n \mid \theta)}}{\int p(\theta)\ p(x_1, \ldots, x_n \mid \theta) d\theta} =$$

Probabilistic programming

Probabilistic constructs

- `sample`: draw a sample from a distribution
- `assume`, `factor`, `observe`: condition the model on inputs (e.g., observed data)
- `infer`: compute the posterior distribution of a model given the inputs

More general than classic Bayesian reasoning

But: general case....

$$p(\theta \mid x_1, \ldots x_n) = \frac{p(\theta)\ p(x_1, \ldots, x_n \mid \theta)}{p(x_1, \ldots, x_n)}$$

(Bayes' theorem)

$$= \frac{p(\theta)\ p(x_1, \ldots, x_n \mid \theta)}{\int p(\theta)\ p(x_1, \ldots, x_n \mid \theta) d\theta}$$

=

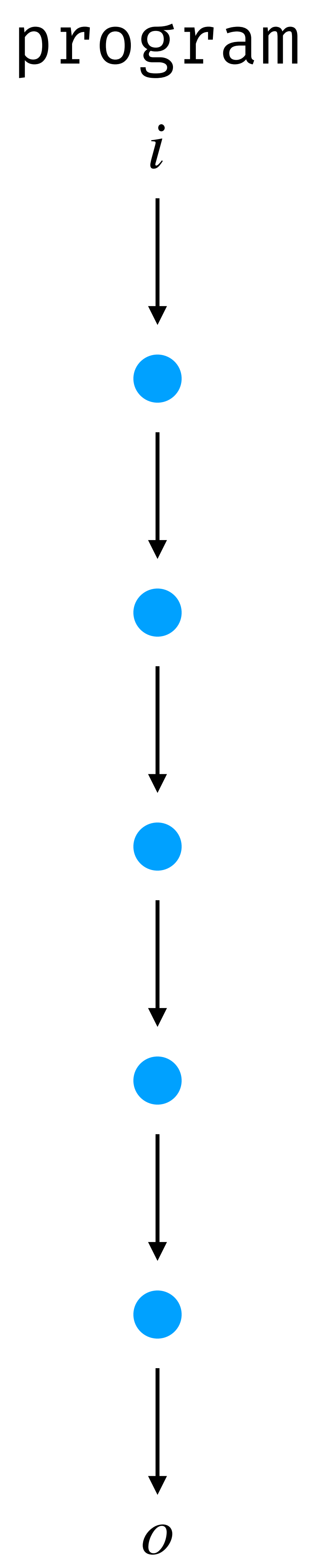
intractable...

Semantics

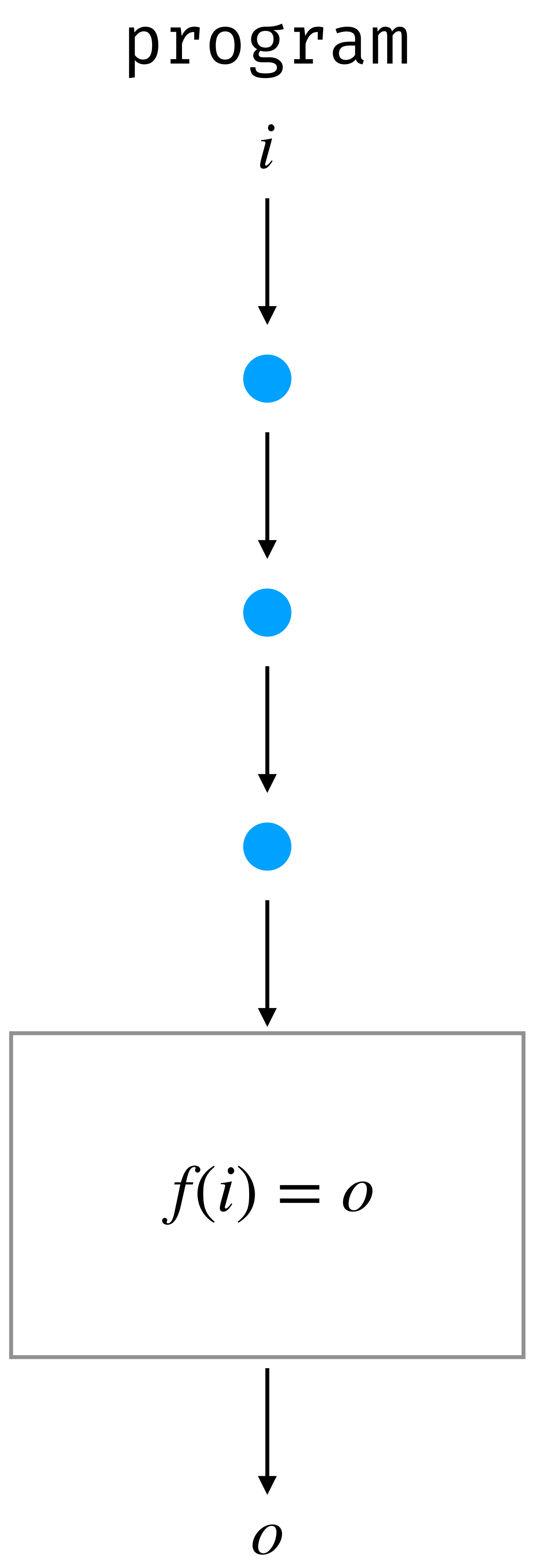
Probabilistic Programming Languages

`infer :` $(\alpha \to \beta) \to \alpha \to \beta$ `dist`

`infer` : $(\alpha \to \beta) \to \alpha \to \beta$ `dist`

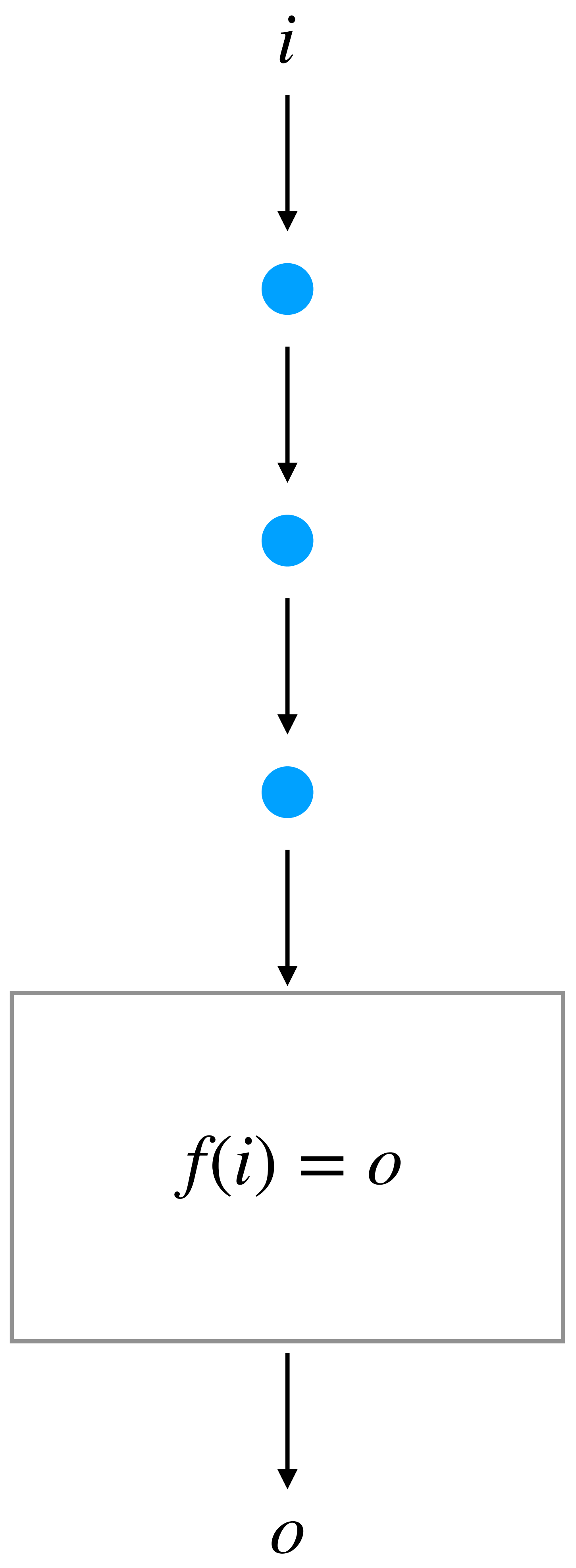


$\texttt{infer} : (\alpha \to \beta) \to \alpha \to \beta\ \texttt{dist}$



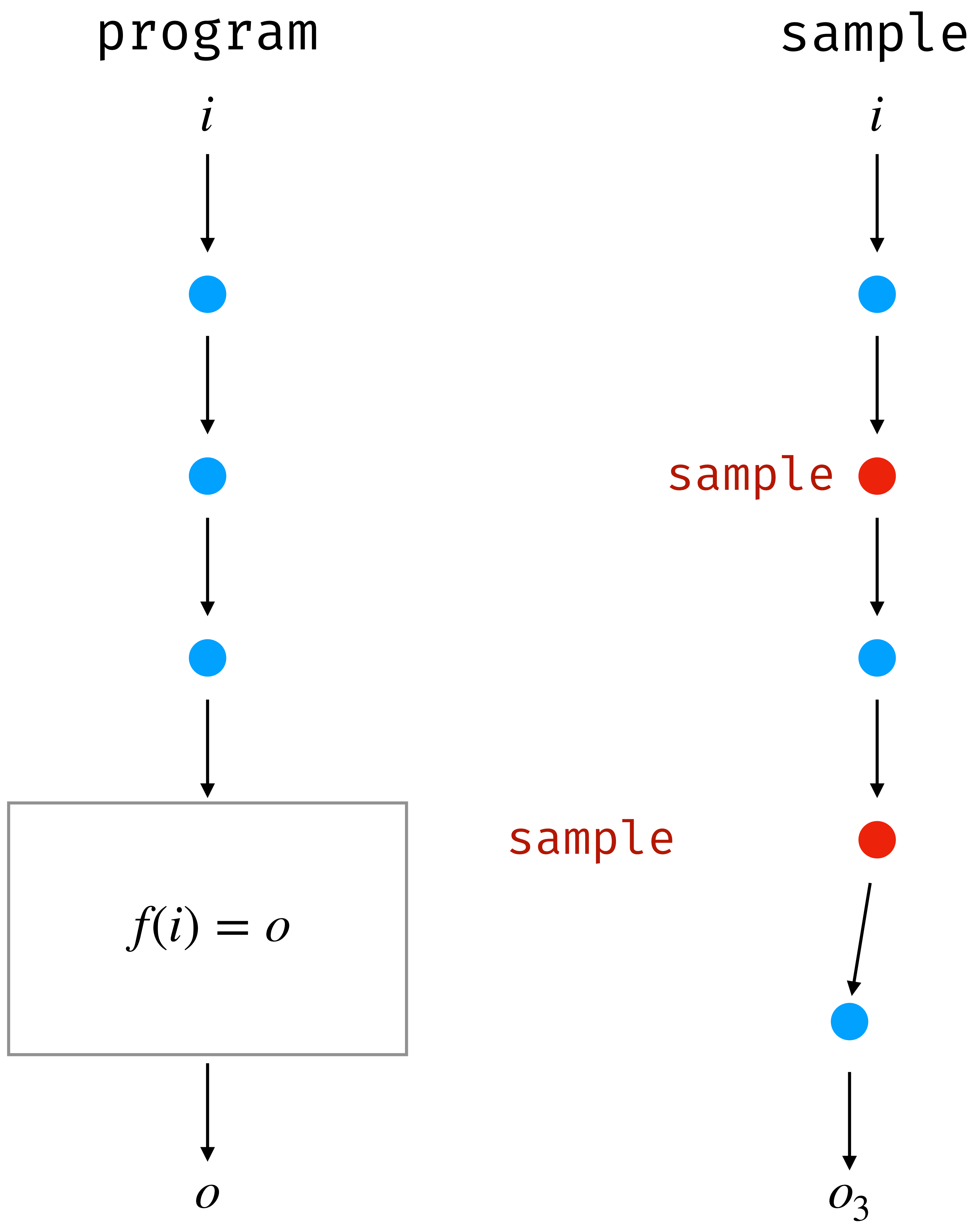
infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist

program

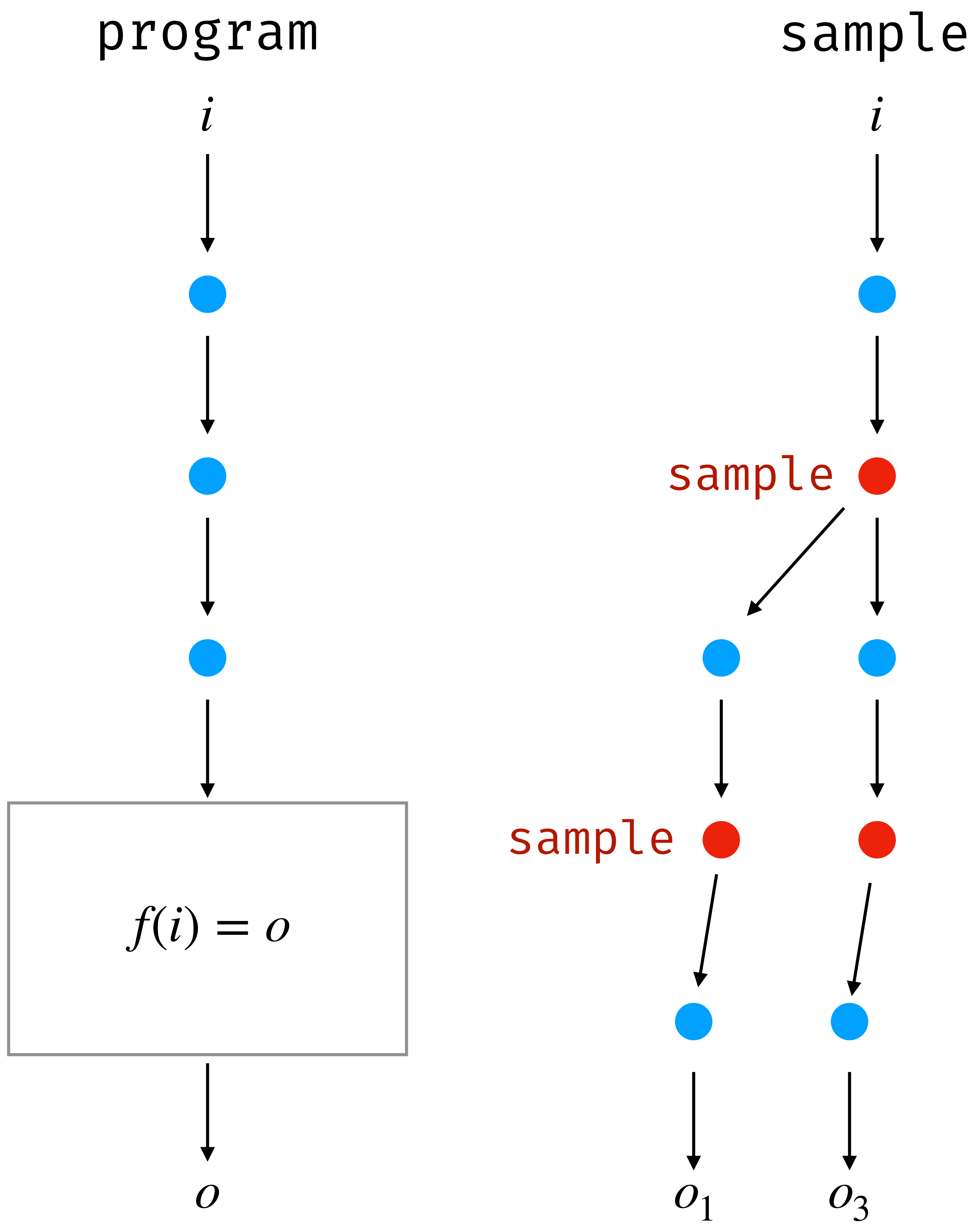


sample

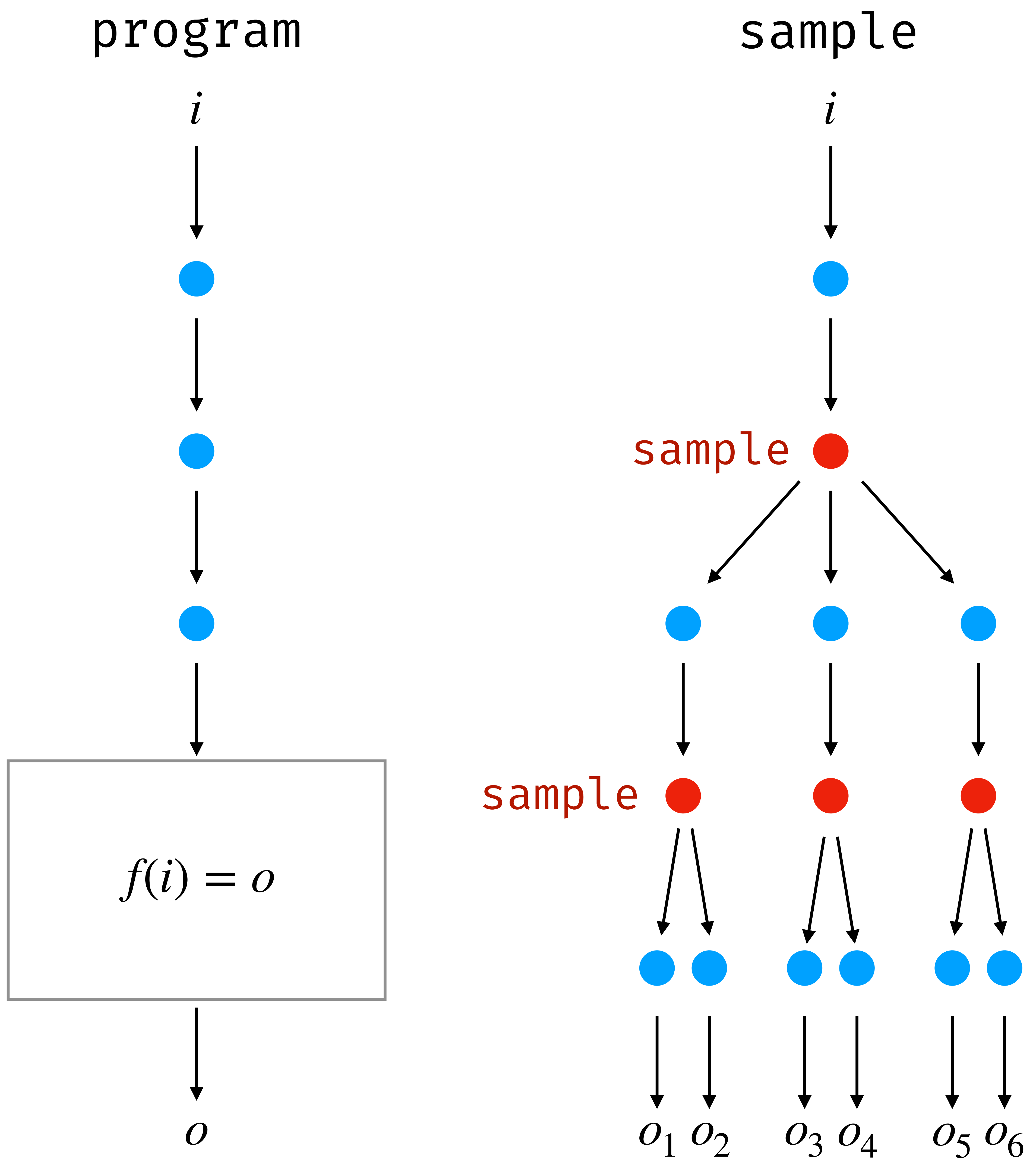
infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist



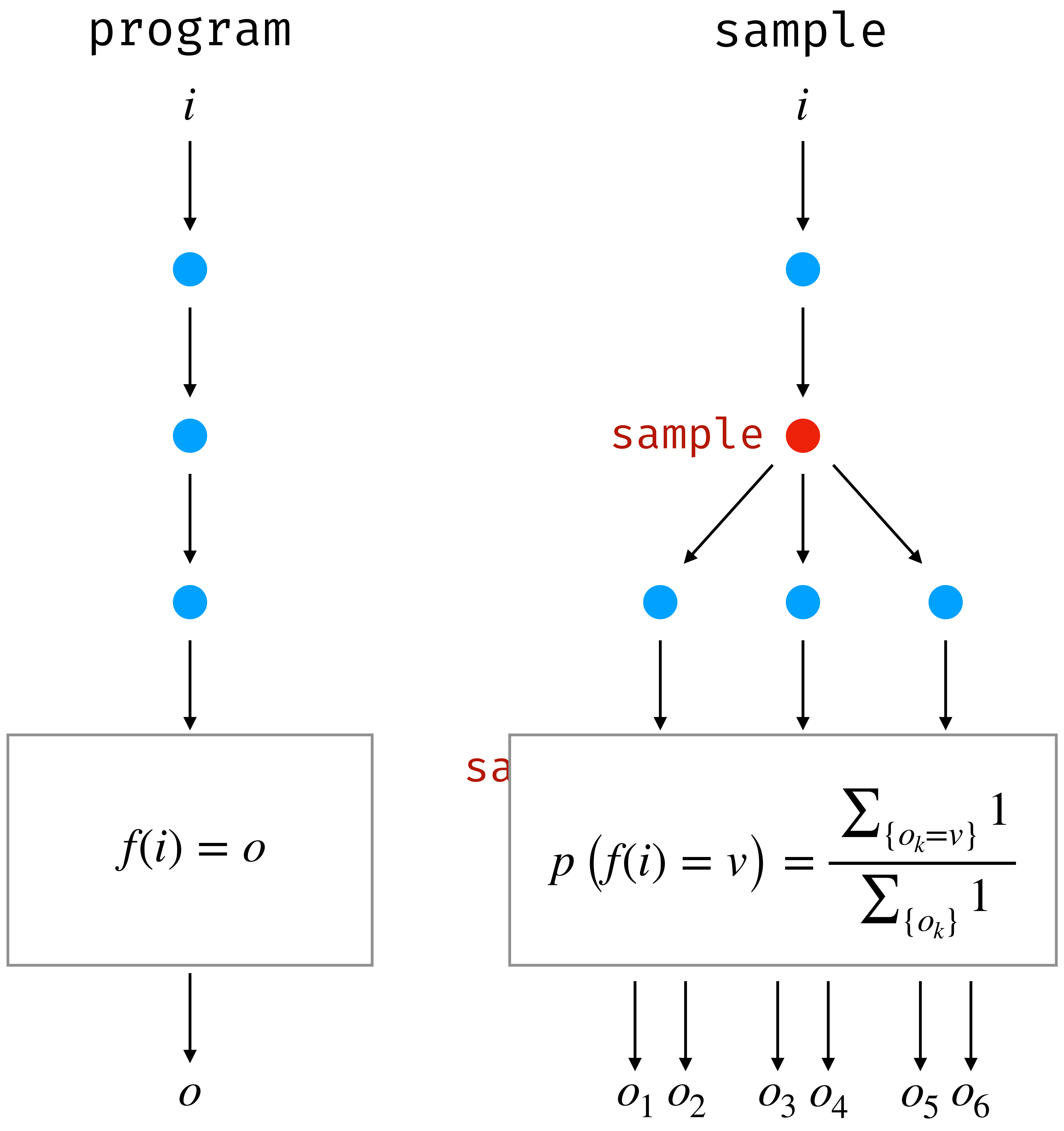
`infer` : $(\alpha \to \beta) \to \alpha \to \beta$ `dist`



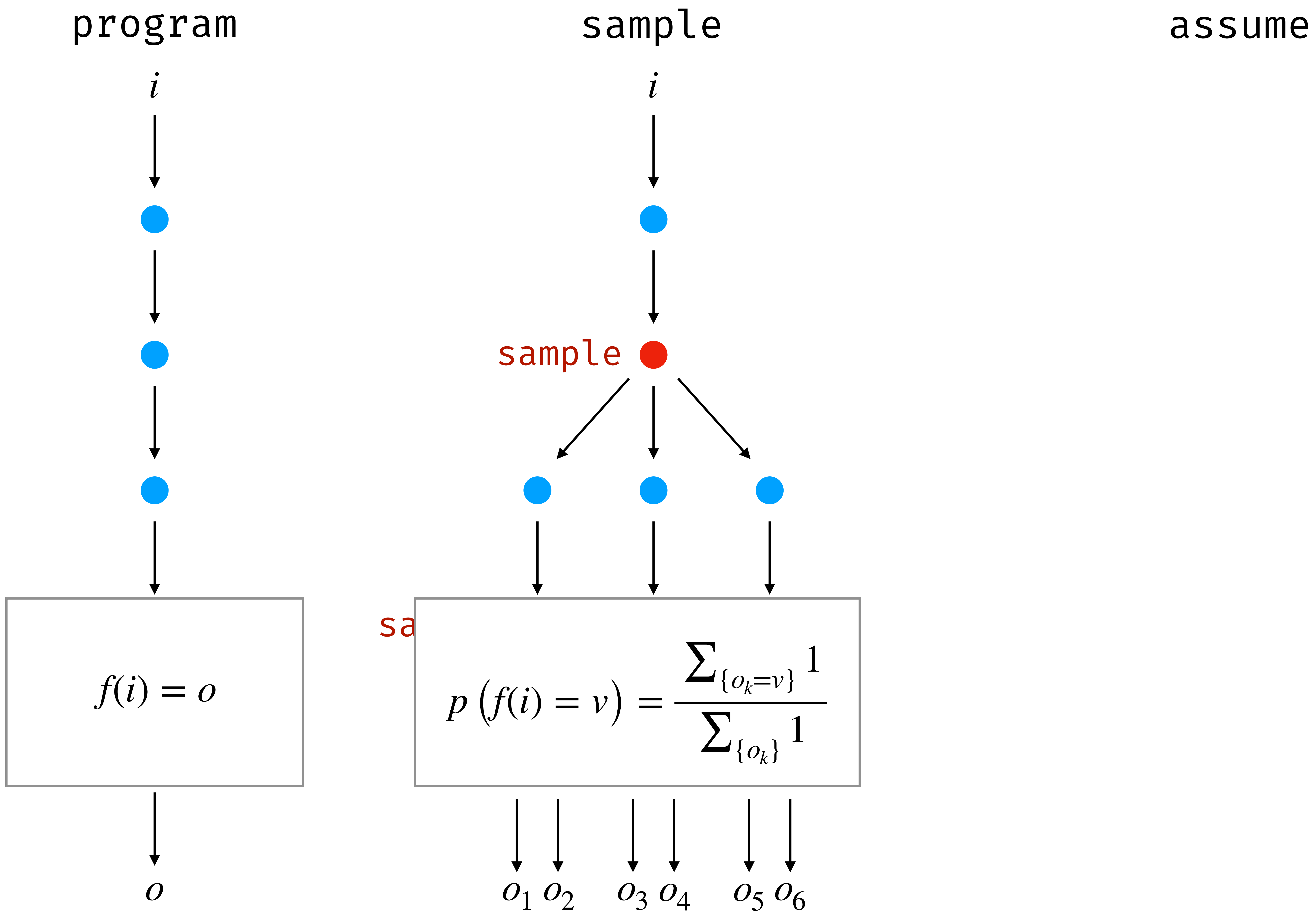
`infer` : $(\alpha \to \beta) \to \alpha \to \beta$ `dist`



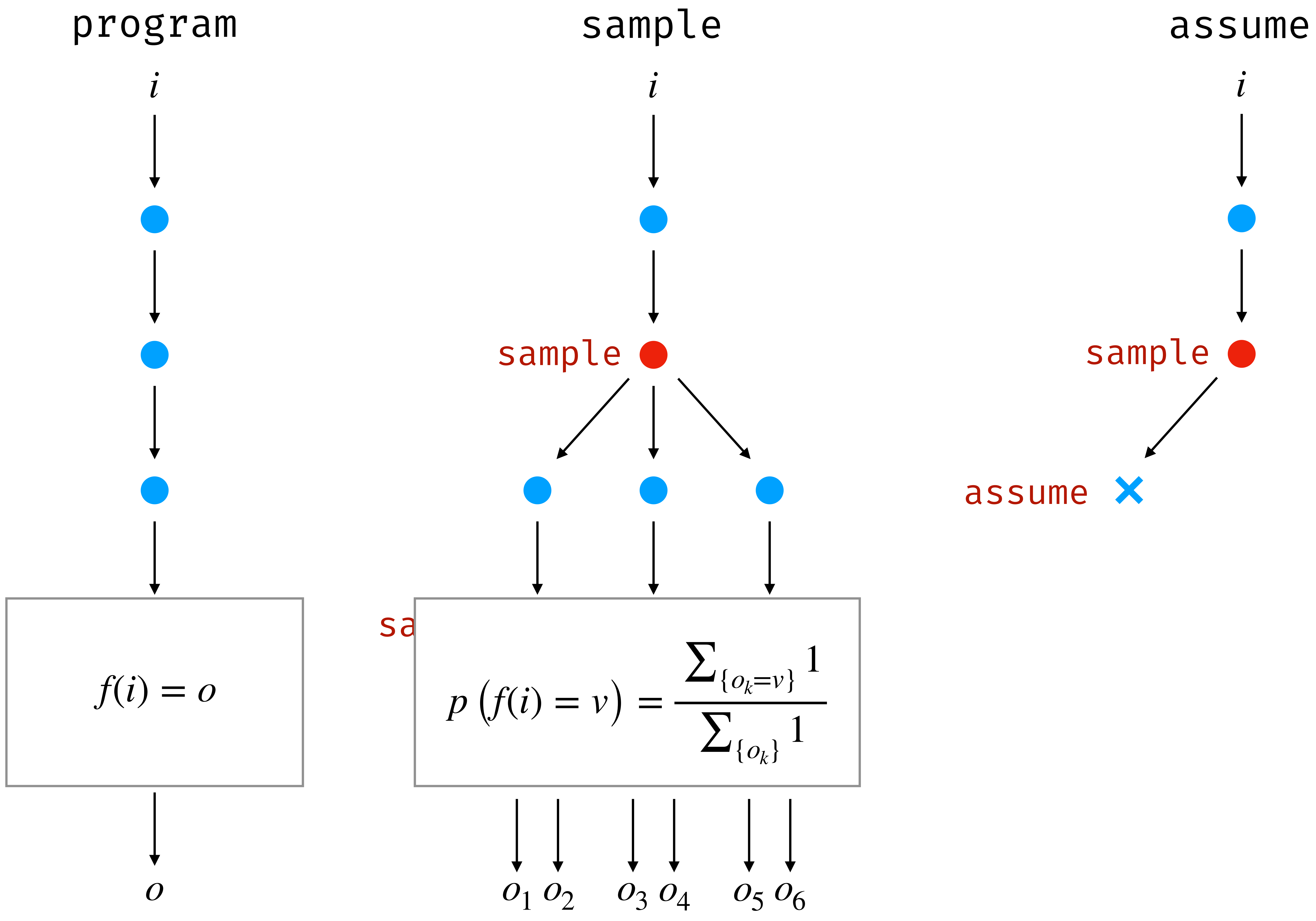
`infer` : $(\alpha \to \beta) \to \alpha \to \beta$ `dist`



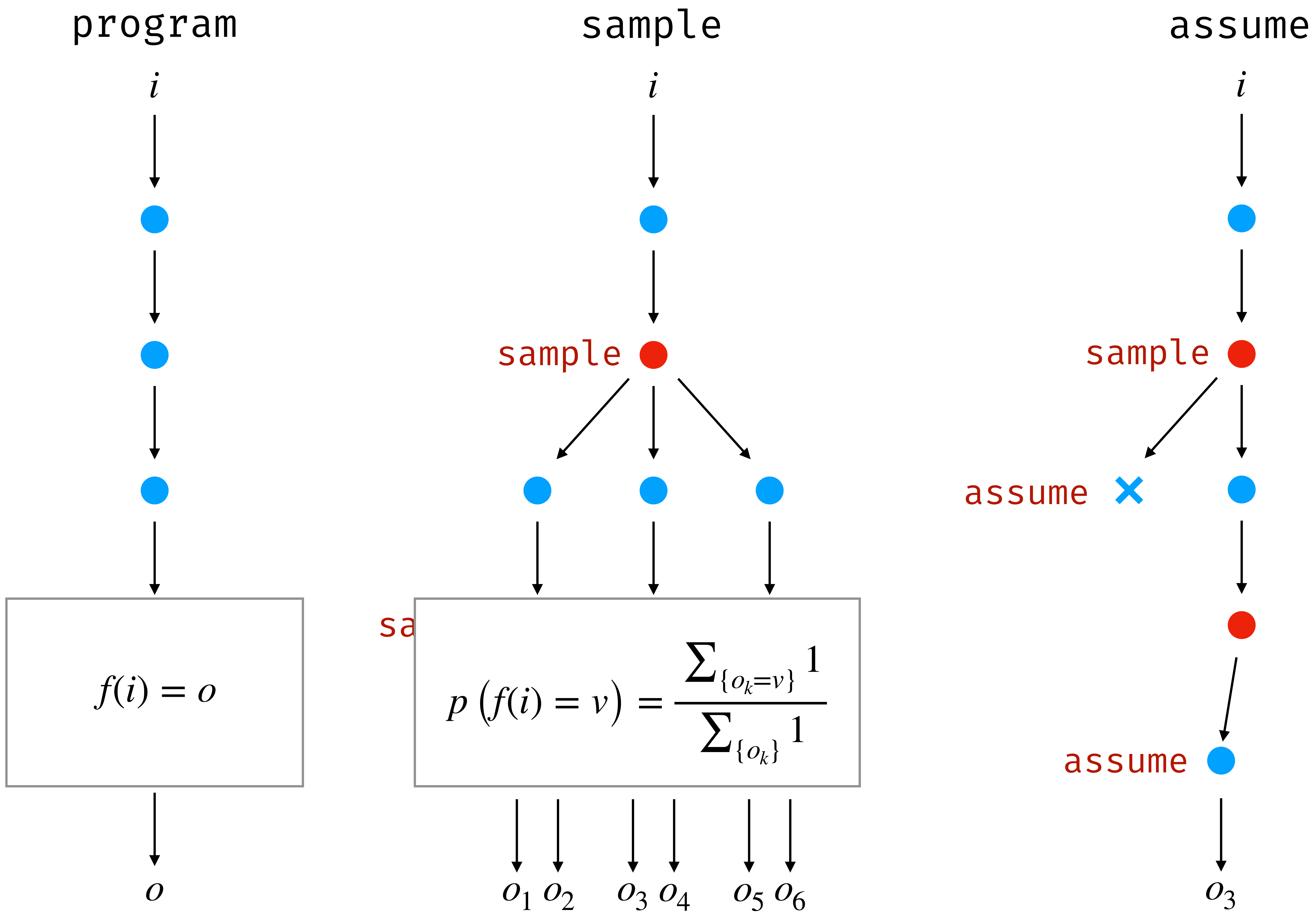
`infer` : $(\alpha \to \beta) \to \alpha \to \beta$ `dist`



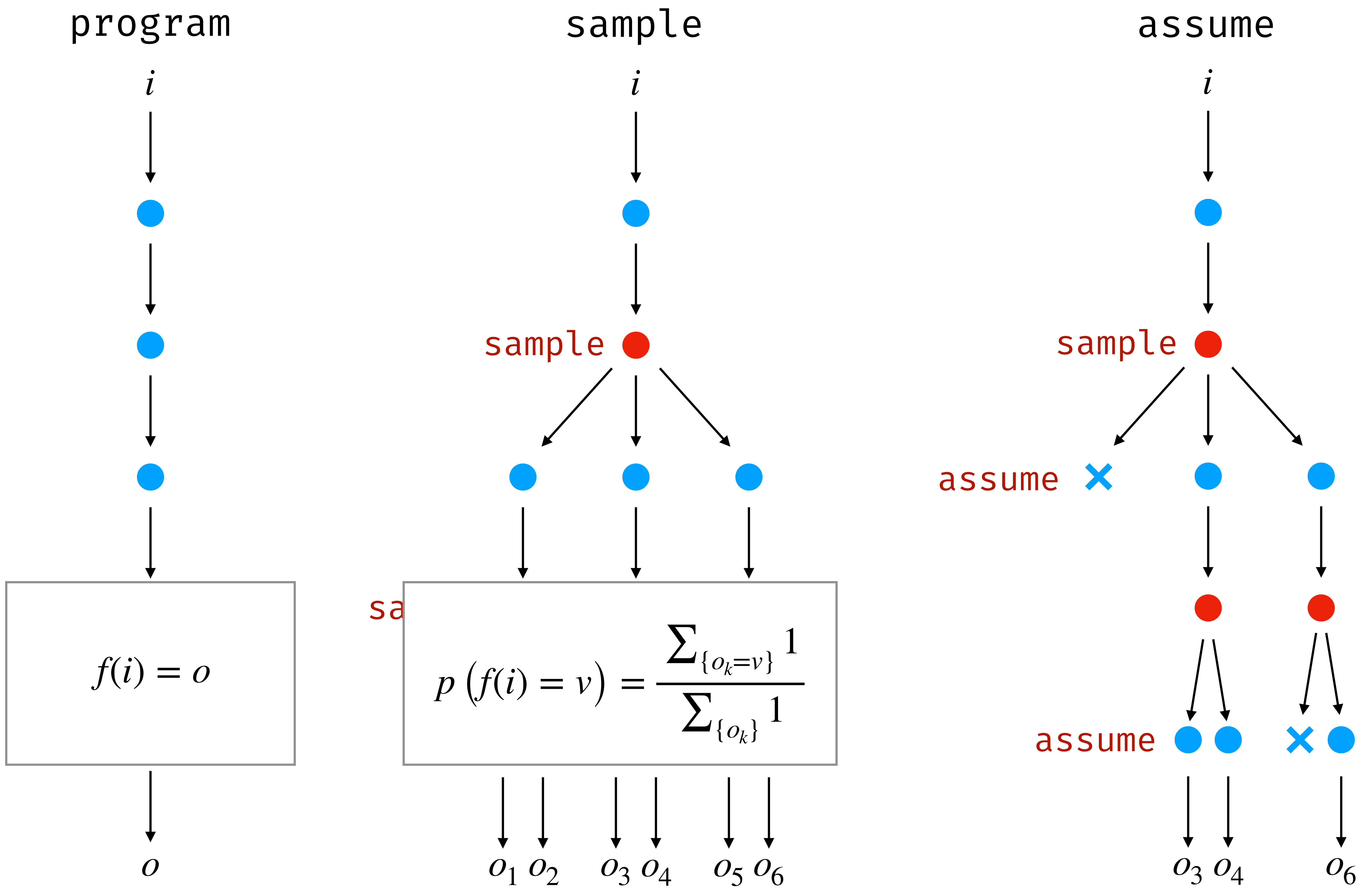
infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist



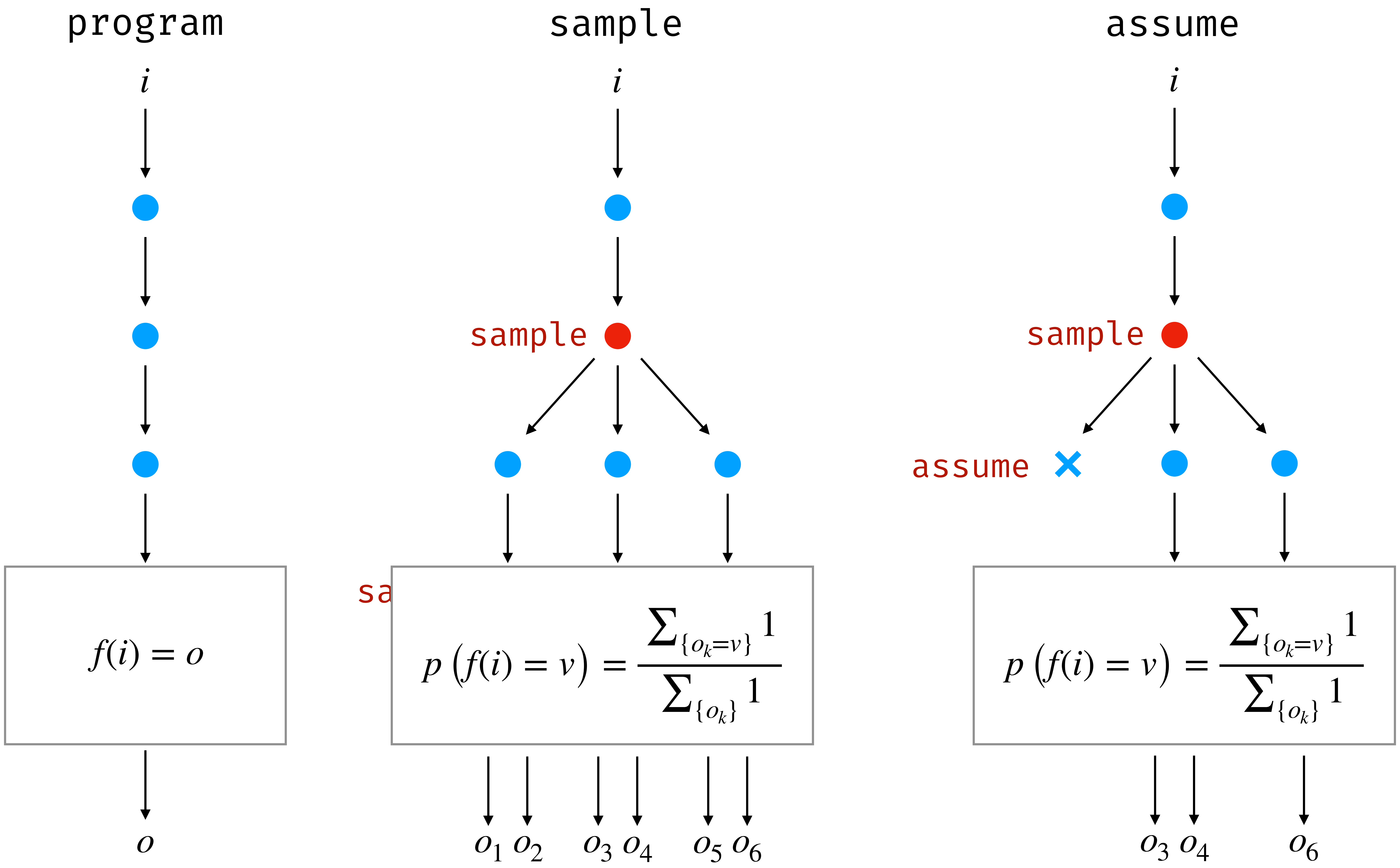
$\texttt{infer} : (\alpha \to \beta) \to \alpha \to \beta\ \texttt{dist}$



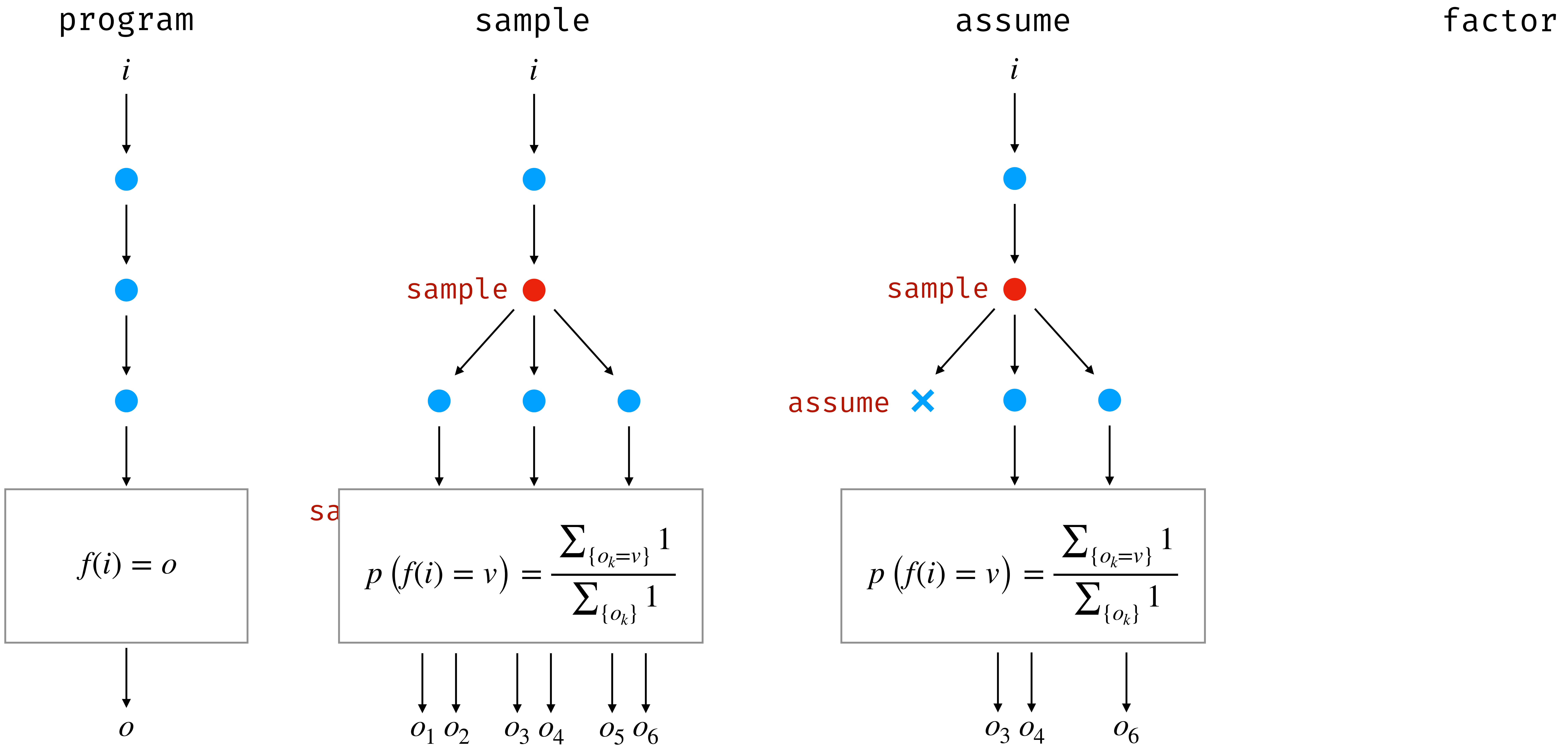
$$\texttt{infer} : (\alpha \to \beta) \to \alpha \to \beta\ \texttt{dist}$$



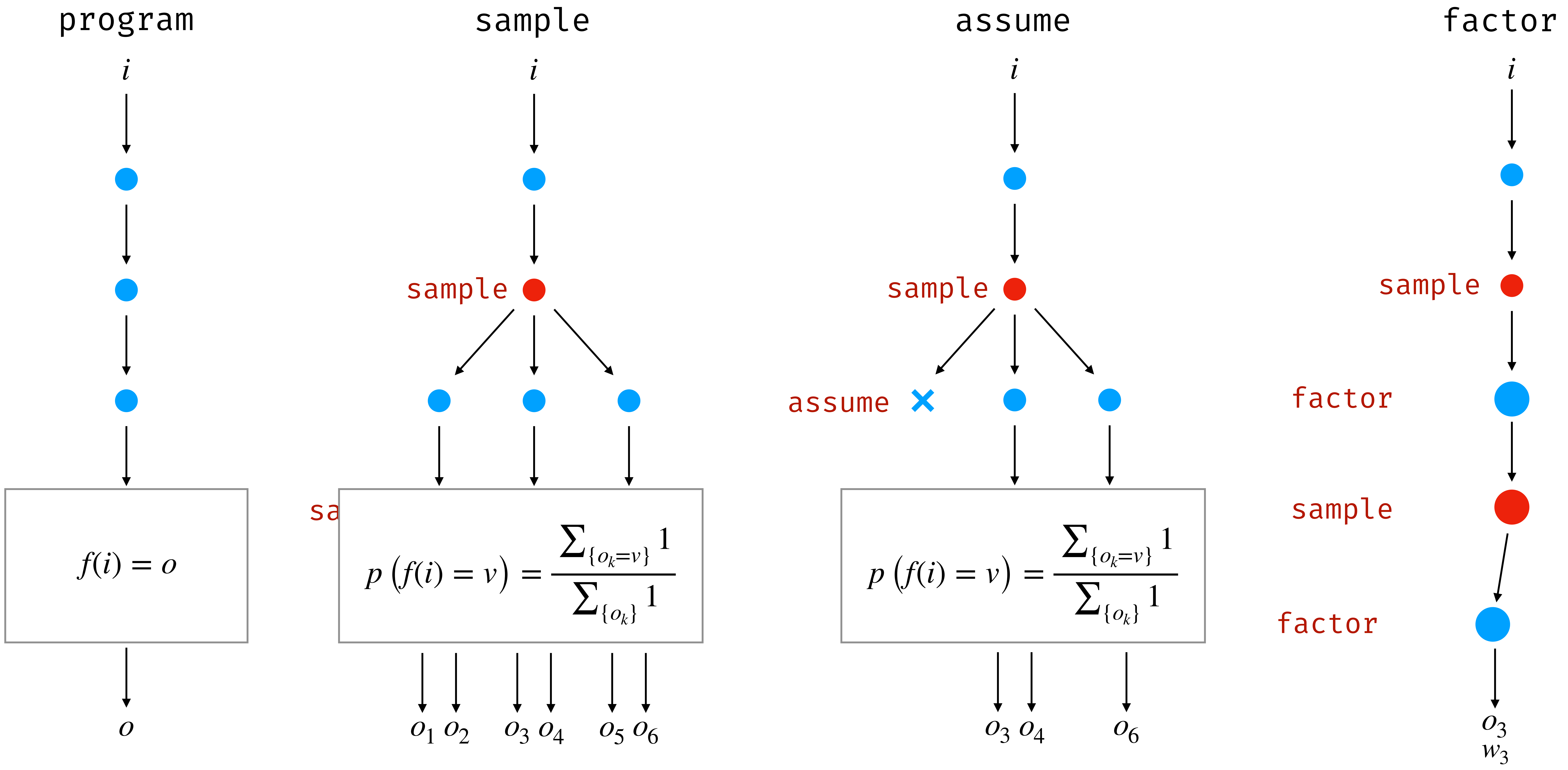
infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist



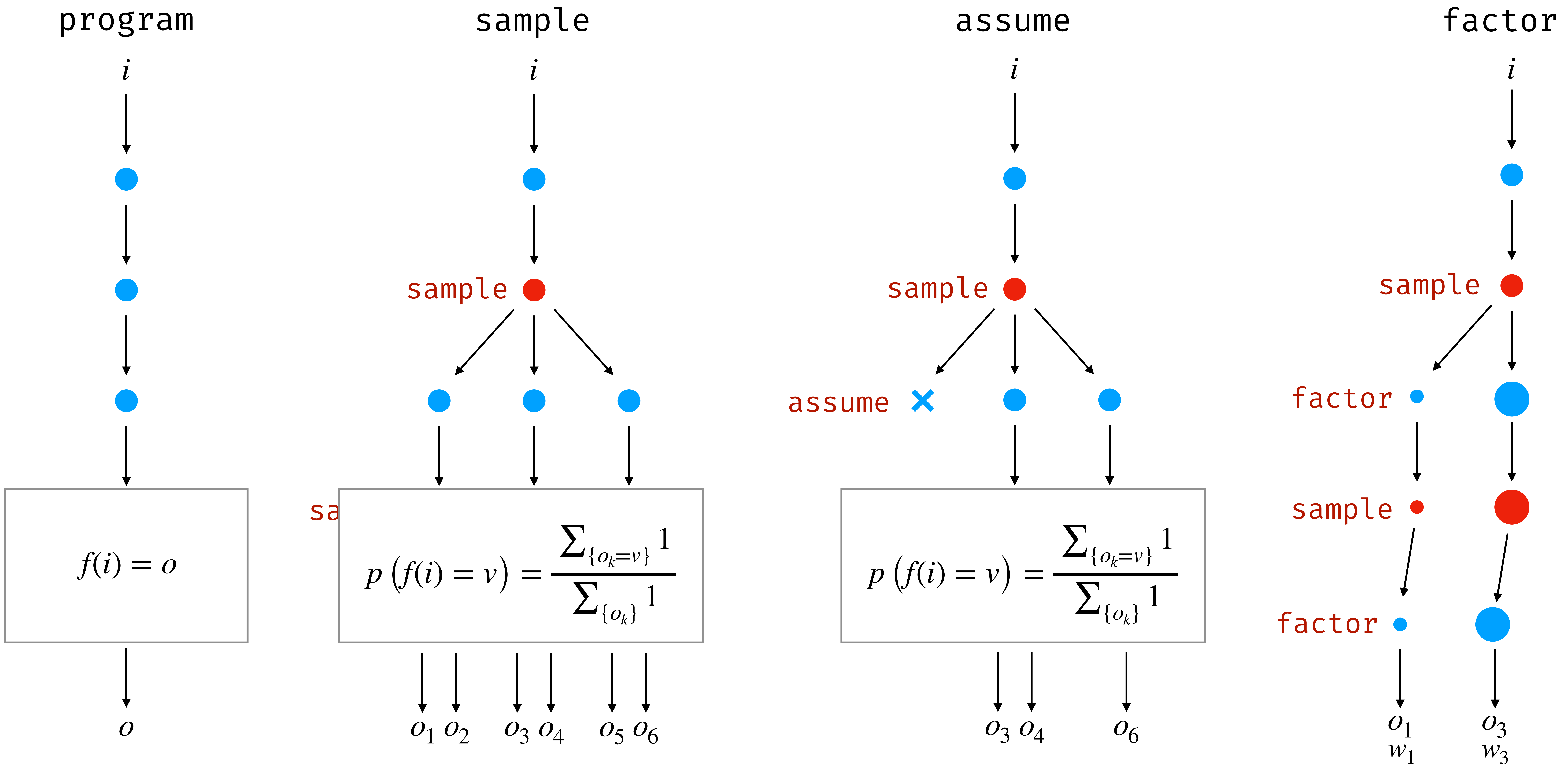
`infer` : $(\alpha \to \beta) \to \alpha \to \beta$ `dist`



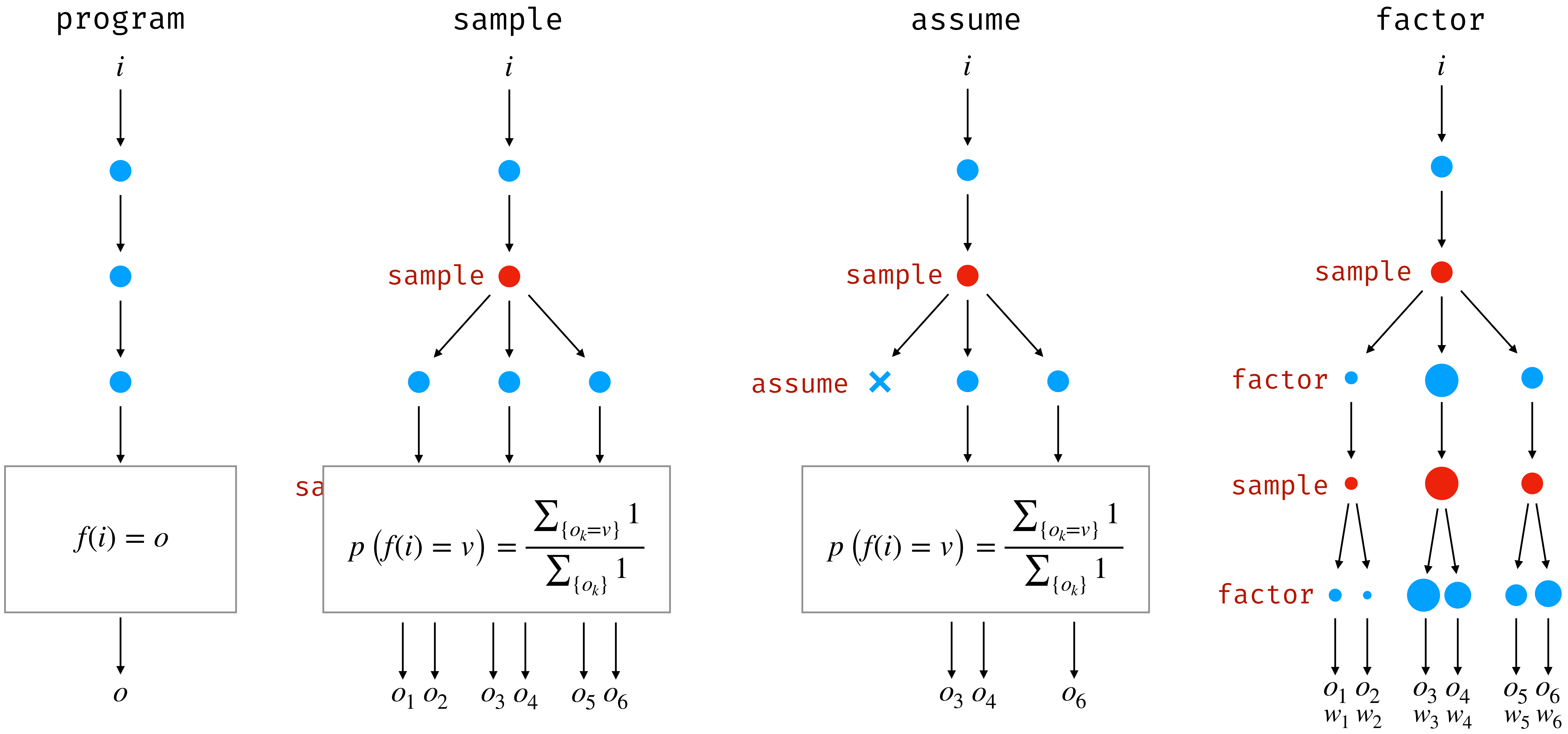
`infer` : $(\alpha \to \beta) \to \alpha \to \beta$ `dist`



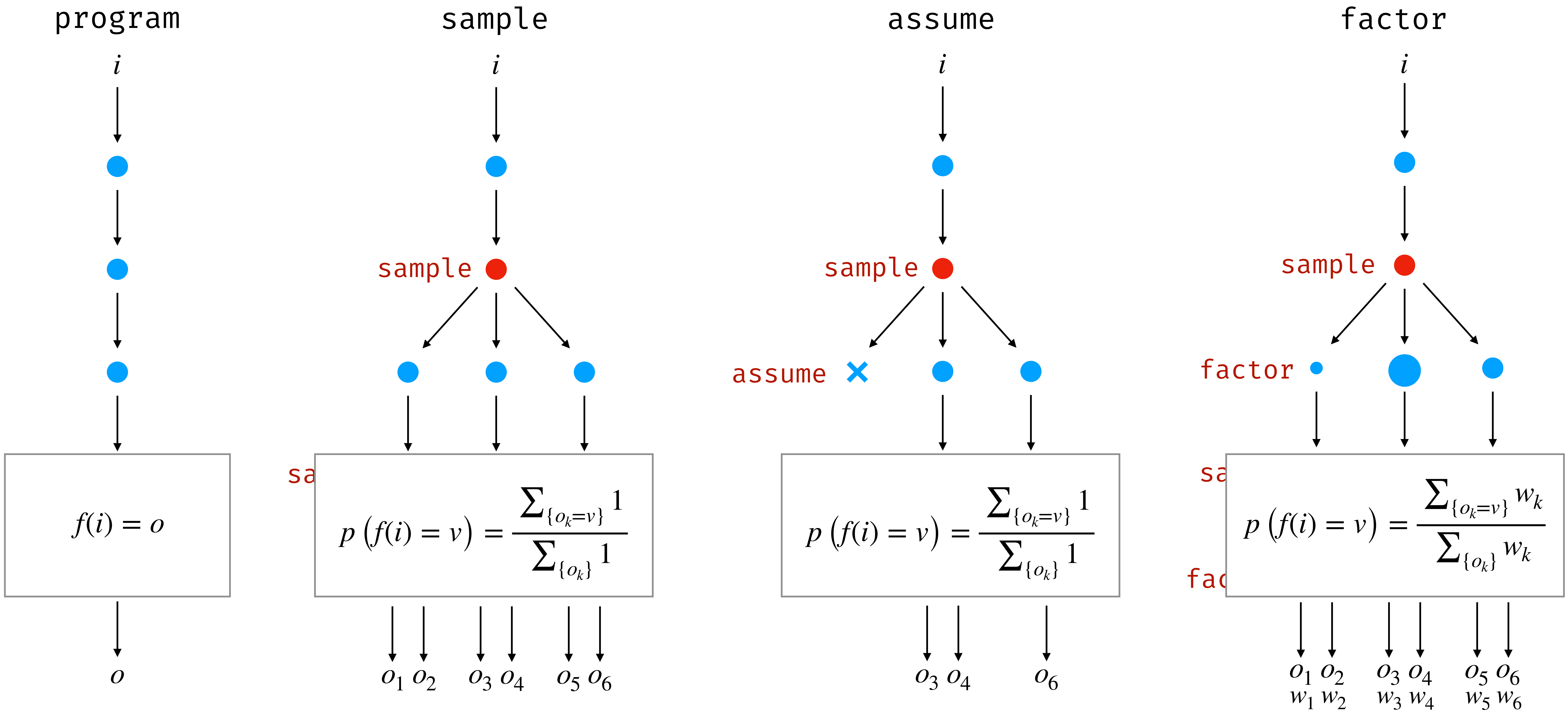
infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist



infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist



infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist



Deterministic vs. probabilistic semantics

output value

$$[\![\text{prog}]\!]_{\gamma} : t$$

environment

Deterministic vs. probabilistic semantics

output value

$$[\![\text{prog}]\!]_{\gamma} : t$$

environment

$$[\![2+5]\!]_{[]} = 7$$

$$[\![\mathsf{x}+\mathsf{y}]\!]_{[\mathsf{x}\leftarrow 2,\mathsf{y}\leftarrow 5]} = 7$$

Deterministic vs. probabilistic semantics

output value

$$[\![\mathsf{prog}]\!]_\gamma : t$$

environment

$$[\![2 + 5]\!]_{[]} = 7$$

$$[\![\mathsf{x} + \mathsf{y}]\!]_{[\mathsf{x} \leftarrow 2, \mathsf{y} \leftarrow 5]} = 7$$

measure over possible outputs

$$\{\!\![\mathsf{model}]\!\!\}_\gamma : \mathsf{Meas}([\![t]\!])$$

environment

Deterministic vs. probabilistic semantics

output valu

$$[\![\text{prog}]\!]_\gamma : t$$

environment

$$[\![2 + 5]\!]_{[]} = 7$$

$$[\![x + y]\!]_{[x \leftarrow 2, y \leftarrow 5]} = 7$$

Measures

Maps a set of possible outcomes to a positive score: $\mu : \Sigma_X \to [0,\infty]$

Probability distributions are normalized measures, i.e., $\mu(\top) = 1$

Bernoulli distribution (discrete)

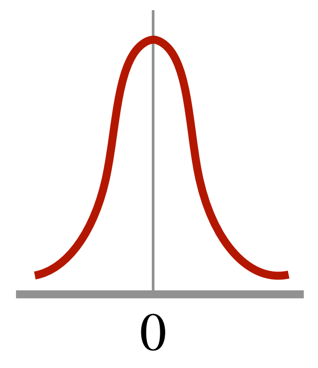
$\mathscr{B}(0.3)(\{1\}) = 0.3$

$\mathscr{B}(0.3)(\{0,1\}) = 1$

Normal distribution (continuous)

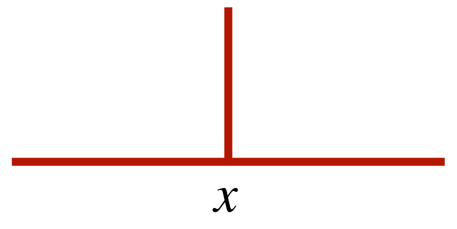
$\mathscr{N}(0,1)([0,1]) \approx 0.34$

$\mathscr{N}(0,1)((-\infty,0]) = 0.5$



Dirac delta measure

$$\delta_x(U) = \begin{cases} 1 & \text{if } x \in U \\ 0 & \text{otherwise} \end{cases}$$



Deterministic vs. probabilistic semantics

output value

$$[\![\mathsf{prog}]\!]_\gamma : t$$

environment

measure over possible outputs

$$\{\![\mathsf{model}]\!\}_\gamma : \mathsf{Meas}([\![t]\!])$$

environment

$$[\![2 + 5]\!]_{[]} = 7$$

$$[\![x + y]\!]_{[x \leftarrow 2, y \leftarrow 5]} = 7$$

Deterministic vs. probabilistic semantics

output value

$$[\![\text{prog}]\!]_\gamma : t$$

environment

measure over possible outputs

$$\{\!|\text{model}|\!\}_\gamma : \mathsf{Meas}([\![t]\!])$$

environment

$[\![2+5]\!]_{[]} = 7$

$[\![\mathtt{x}+\mathtt{y}]\!]_{[\mathtt{x}\leftarrow 2, \mathtt{y}\leftarrow 5]} = 7$

$\{\!|2+5|\!\}_{[]}(U) = \delta(7)(U)$

$\{\!|\mathtt{sample}(\mathtt{normal}(\mathtt{x},\mathtt{y}))|\!\}_{[\mathtt{x}\leftarrow 1, \mathtt{y}\leftarrow 0]}(U) = \mathcal{N}(0,1)(U)$

(Un)normalized measures

measure over possible outputs

$$\{\!\![\text{model}]\!\!\}_{\gamma} : \mathsf{Meas}([\![t]\!])$$

environment

Unnormalized measure

(Un)normalized measures

measure over possible outputs

$$\{\!\{\text{model}\}\!\}_\gamma : \textbf{Meas}(\llbracket t \rrbracket)$$

environment

Unnormalized measure

$$\llbracket \texttt{infer}(\text{model}) \rrbracket_\gamma = \frac{\{\!\{\text{model}\}\!\}_\gamma}{\{\!\{\text{model}\}\!\}_\gamma \left(\llbracket \mathit{typeOf}(\text{model}) \rrbracket\right)}$$

normalize over all possible values

Distribution

(Un)normalized measures

measure over possible outputs

$$\{\!\!\{\text{model}\}\!\!\}_\gamma : \mathbf{Meas}(\llbracket t \rrbracket)$$

environment

Unnormalized measure

$$\llbracket \texttt{infer}(\text{model}) \rrbracket_\gamma = \frac{\{\!\!\{\text{model}\}\!\!\}_\gamma}{\{\!\!\{\text{model}\}\!\!\}_\gamma \left(\llbracket \mathit{typeOf}(\text{model}) \rrbracket\right)}$$

Distribution

normalize over all possible values

Approximate inference

Probabilistic Programming Languages

Importance sampling

Importance sampling

Inference algorithm

- Run a set of n independent executions
- `sample`: draw a sample from a distribution
- `factor`: associate a score to the current execution
- `infer`: gather output values v_i and score W_i to approximate the posterior distribution

$$p_i = \frac{W_i}{\sum_{1 \leq i \leq n} W_i}$$

Importance sampling

Inference algorithm

- Run a set of n independent executions
- `sample`: draw a sample from a distribution
- `factor`: associate a score to the current execution
- `infer`: gather output values v_i and score W_i to approximate the posterior distribution

$$p_i = \frac{W_i}{\sum_{1 \leq i \leq n} W_i}$$

Conditioning

- `assume(p)`: sets the score to 0 if `p` is false
- `observe(d,v)`: multiply the score by the likelihood of `d` at `v` (density function of `d`)

Importance sampling

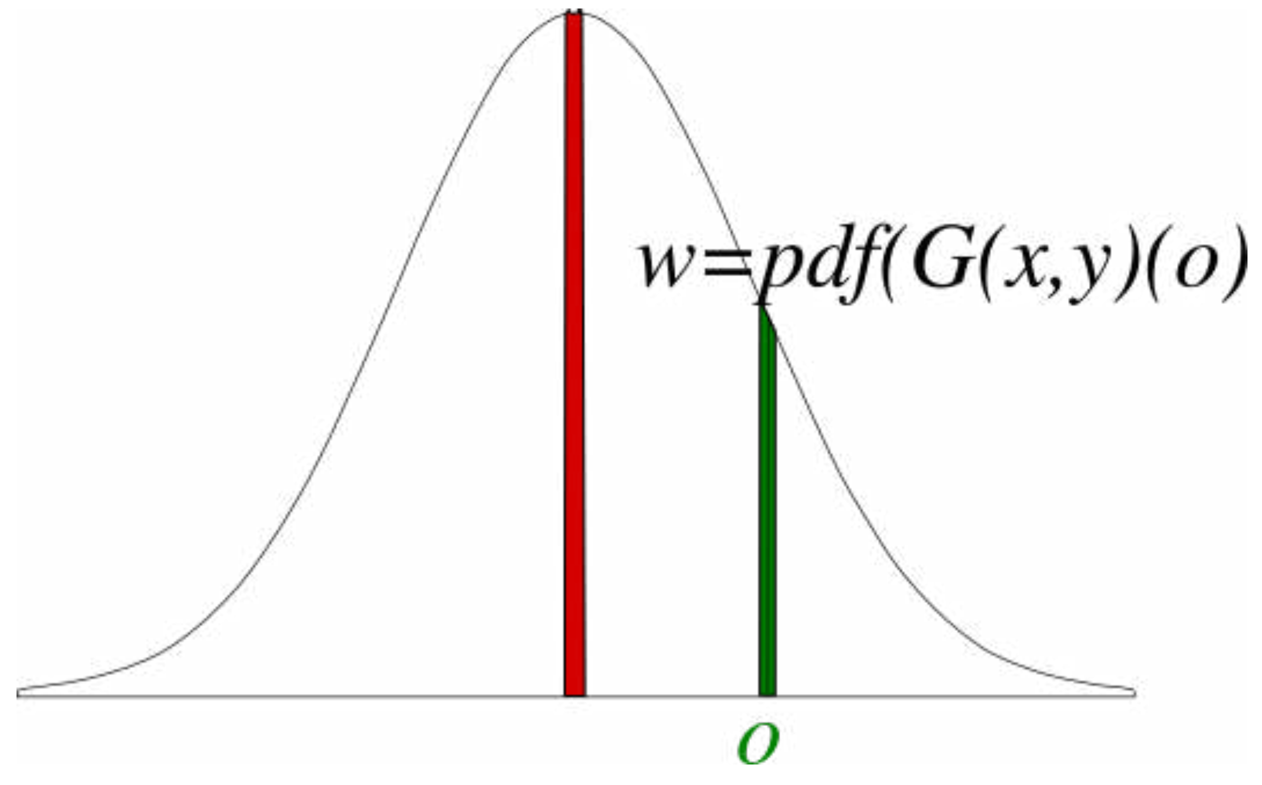
Inference algorithm

- Run a set of n independent executions
- `sample`: draw a sample from a distribution
- `factor`: associate a score to the current execution
- `infer`: gather output values v_i and score W_i to approximate the posterior distribution

$$p_i = \frac{W_i}{\sum_{1 \leq i \leq n} W_i}$$

Conditioning

- `assume(p)`: sets the score to 0 if `p` is false
- `observe(d,v)`: multiply the score by the likelihood of `d` at `v` (density function of `d`)



Importance sampling

0
```
def model(data):
  ...
  x = sample ...
  ...
  factor ...
  ...
  return output
```

1
```
def model(data):
  ...
  x = sample ...
  ...
  factor ...
  ...
  return output
```

2
```
def model(data):
  ...
  x = sample ...
  ...
  factor ...
  ...
  return output
```

Importance sampling

0
```
def model(data):
  ...
  x = sample ...
  ...
  factor ...
  ...
  return output
```

1
```
def model(data):
  ...
  x = sample ...
  ...
  factor ...
  ...
  return output
```

2
```
def model(data):
  ...
  x = sample ...
  ...
  factor ...
  ...
  return output
```

```
o0, 0.66
```

Importance sampling

0

```
def model(data):
  ...
  x = sample ...
  ...
  factor ...
  ...
  return output
```

1

```
def model(data):
  ...
  x = sample ...
  ...
  factor ...
  ...
  return output
```

2

```
def model(data):
  ...
  x = sample ...
  ...
  factor ...
  ...
  return output
```

o0, 0.66

Importance sampling

0

```
def model(data):
  ...
  x = sample ...
  ...
  factor ...
  ...
  return output
```

1

```
def model(data):
  ...
  x = sample ...
  ...
  factor ...
  ...
  return output
```

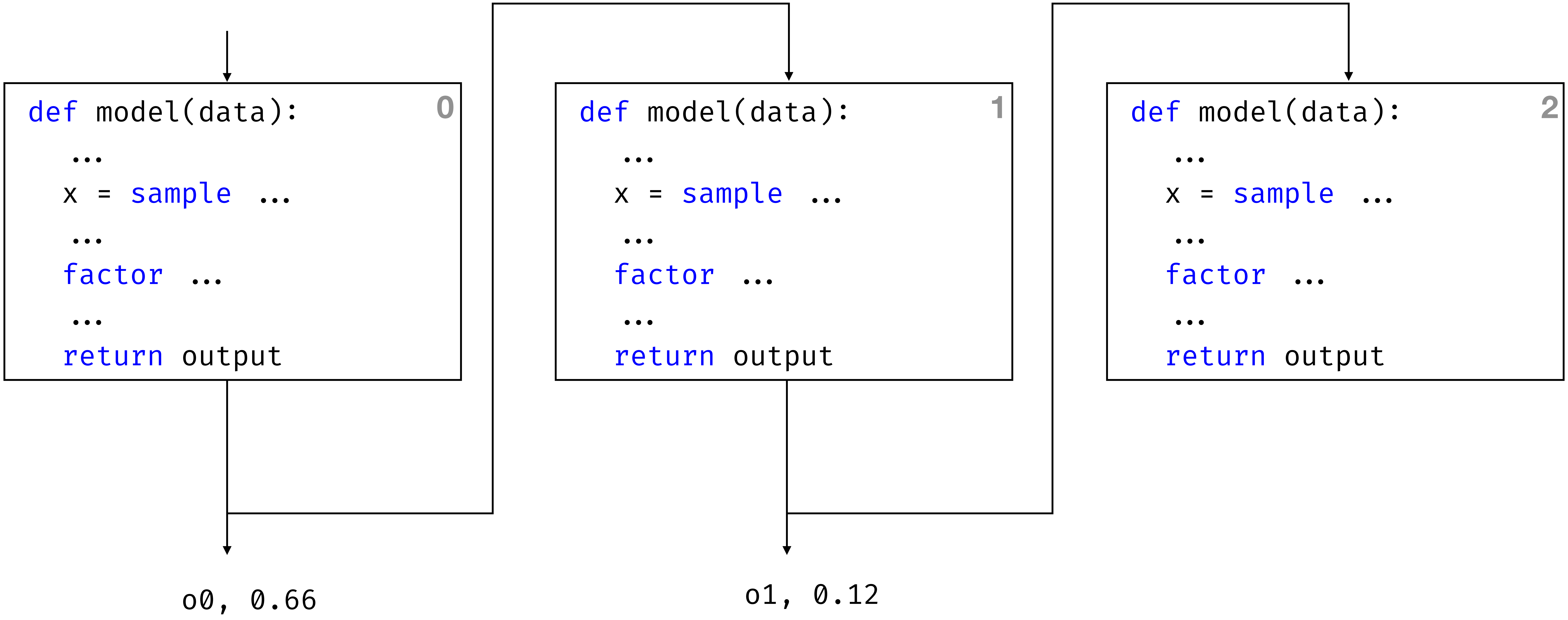
2

```
def model(data):
  ...
  x = sample ...
  ...
  factor ...
  ...
  return output
```

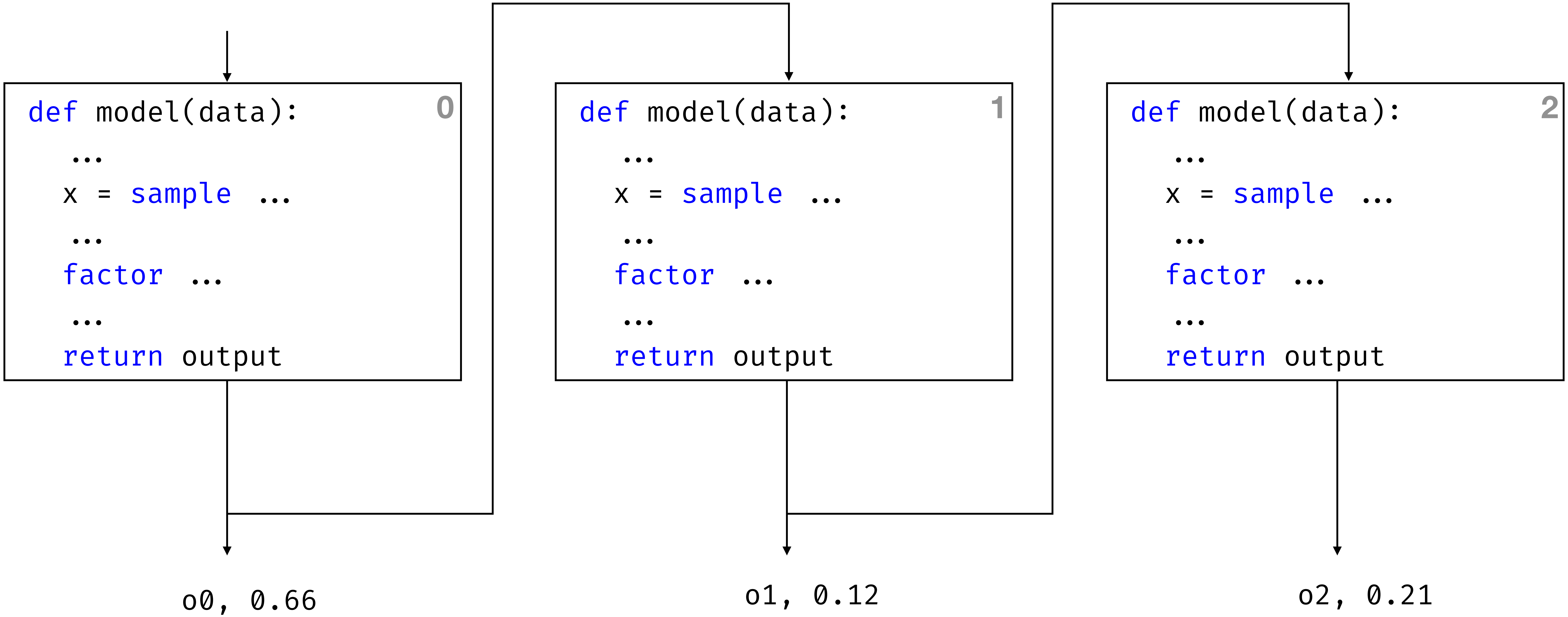
`o0, 0.66`

`o1, 0.12`

Importance sampling



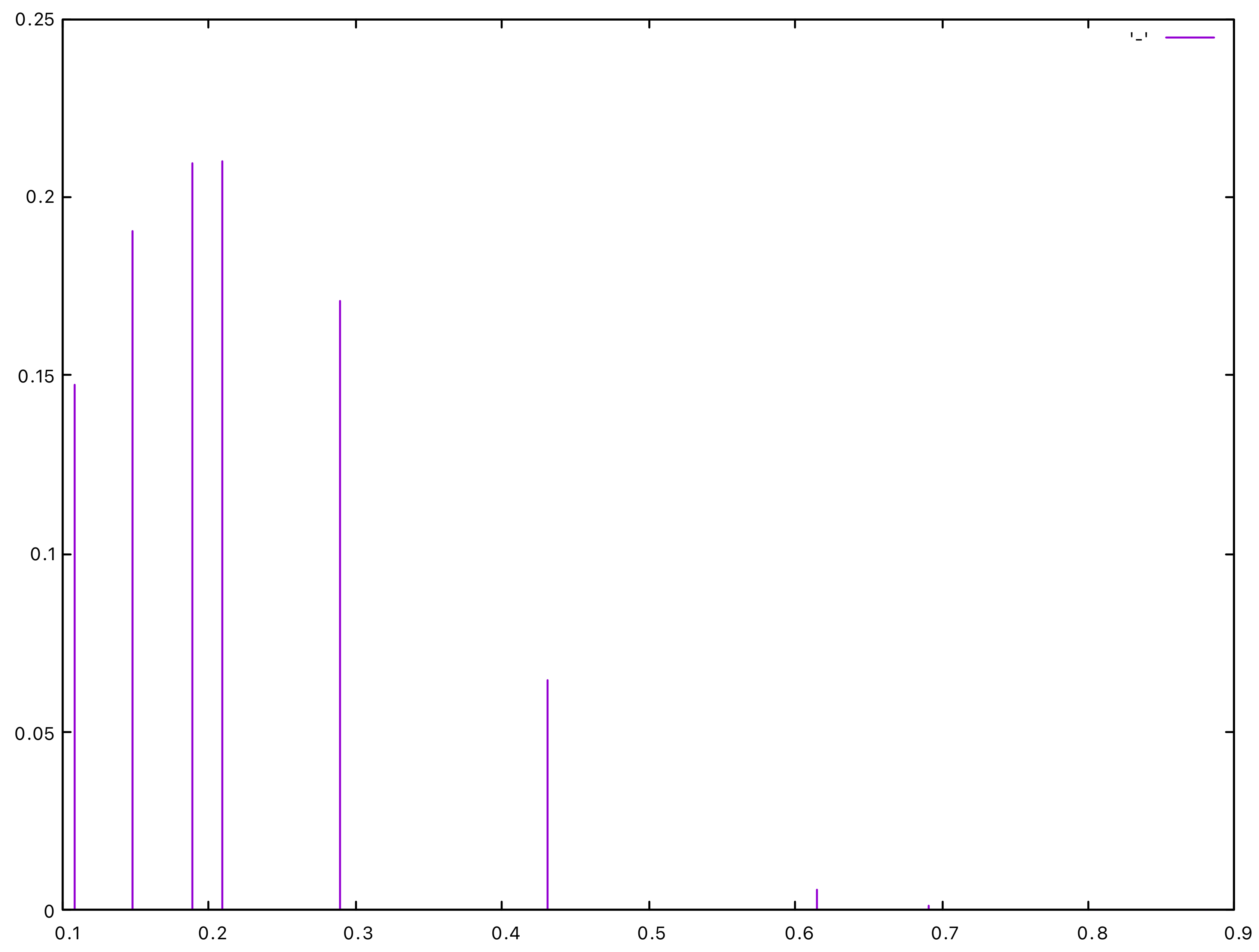
Importance sampling



Example: coin

```
from mu_ppl import *

def coin(obs: list[int]) → float:
  p = sample(Uniform(0, 1))
  for o in obs:
    observe(Bernoulli(p), o)
  return p

with ImportanceSampling(num_particles=10):
  dist = infer(coin, [0, 0, 0, 0, 0, 0, 0, 0, 1, 1])
  viz(dist)
```

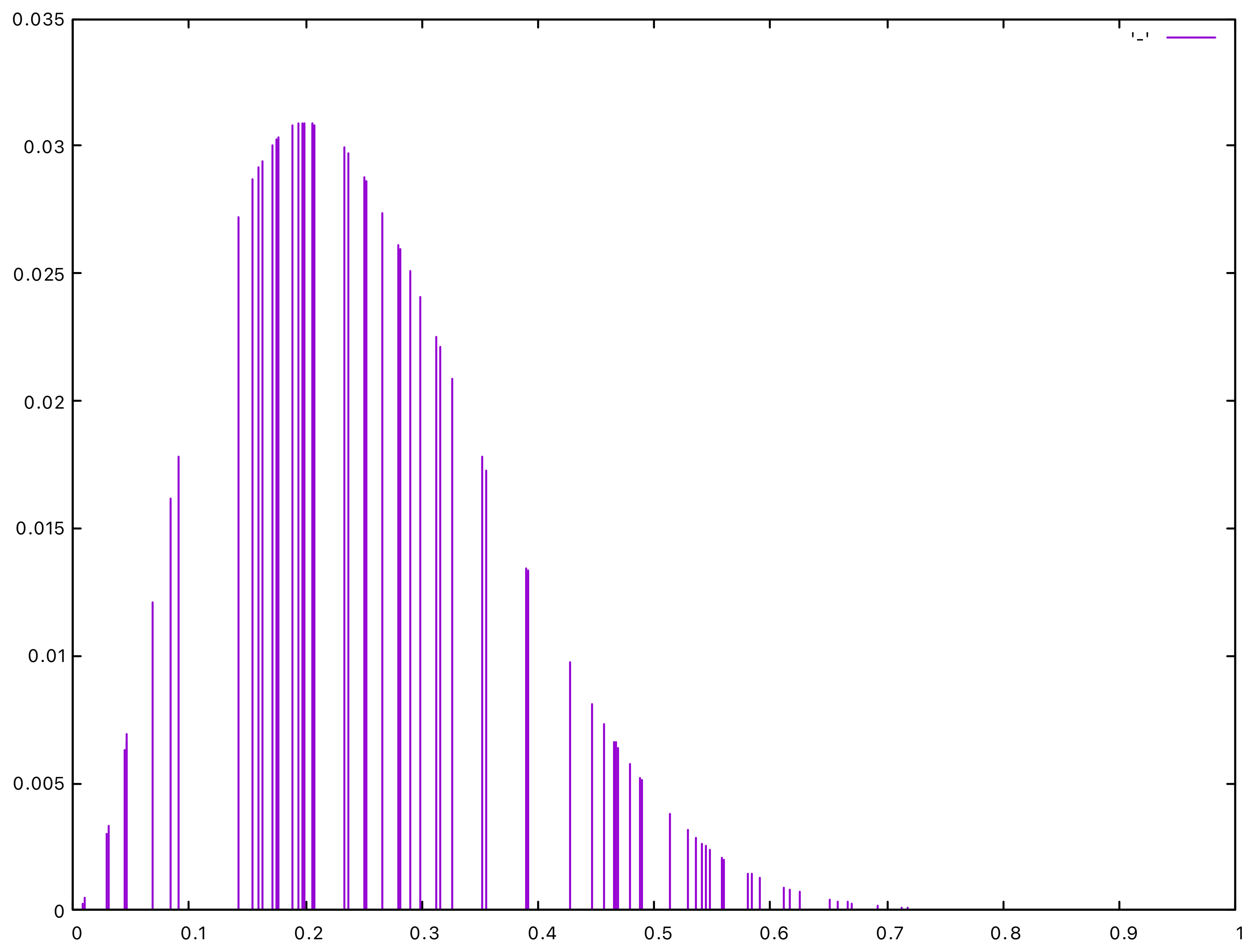
10 particles



Example: coin

```
from mu_ppl import *

def coin(obs: list[int]) → float:
  p = sample(Uniform(0, 1))
  for o in obs:
    observe(Bernoulli(p), o)
  return p

with ImportanceSampling(num_particles=100):
  dist = infer(coin, [0, 0, 0, 0, 0, 0, 0, 0, 1, 1])
  viz(dist)
```

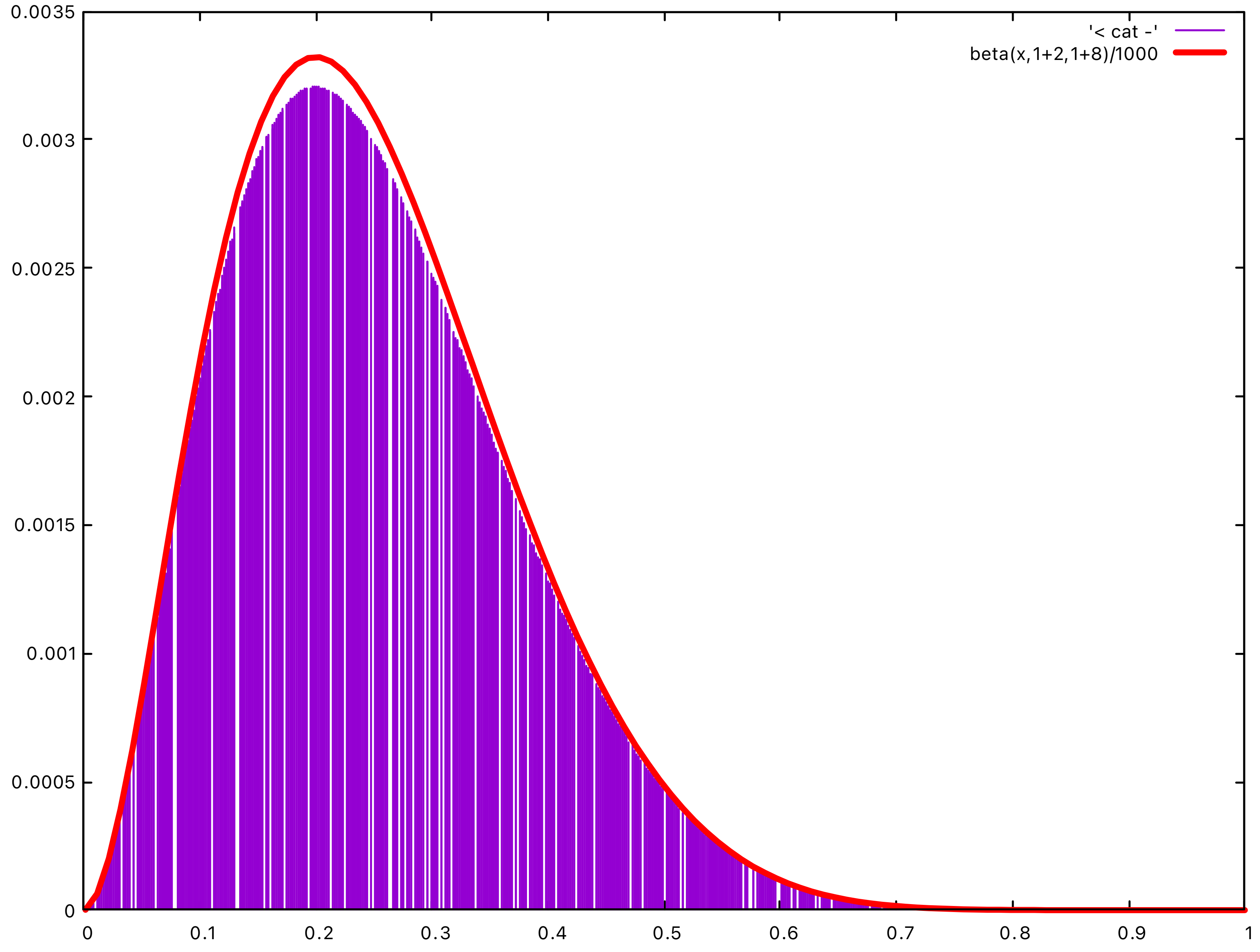
100 particles



Example: coin

```python
from mu_ppl import *

def coin(obs: list[int]) → float:
  p = sample(Uniform(0, 1))
  for o in obs:
    observe(Bernoulli(p), o)
  return p

with ImportanceSampling(num_particles=1000):
  dist = infer(coin, [0, 0, 0, 0, 0, 0, 0, 0, 1, 1])
  viz(dist)
```

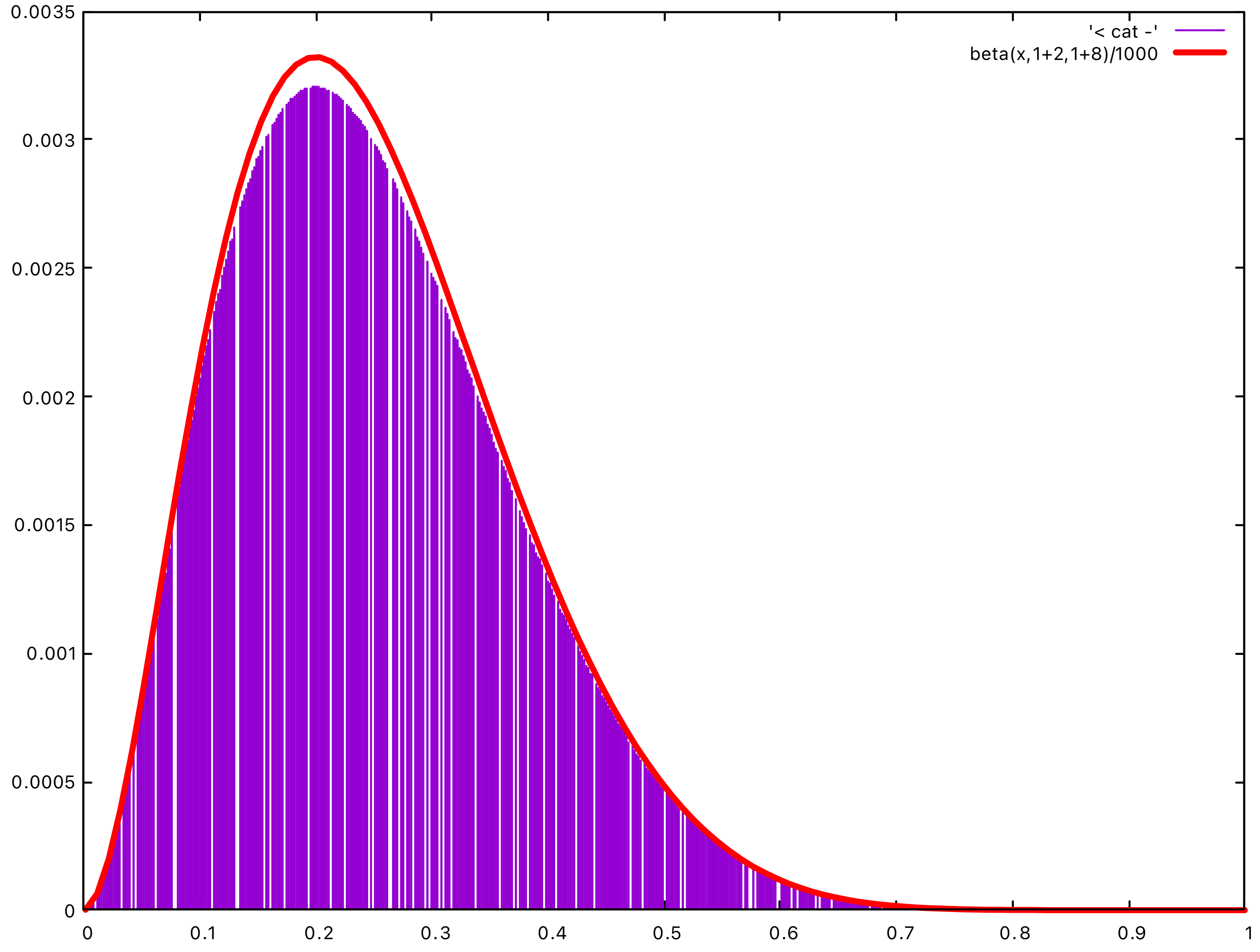
1000 particles



Example: coin

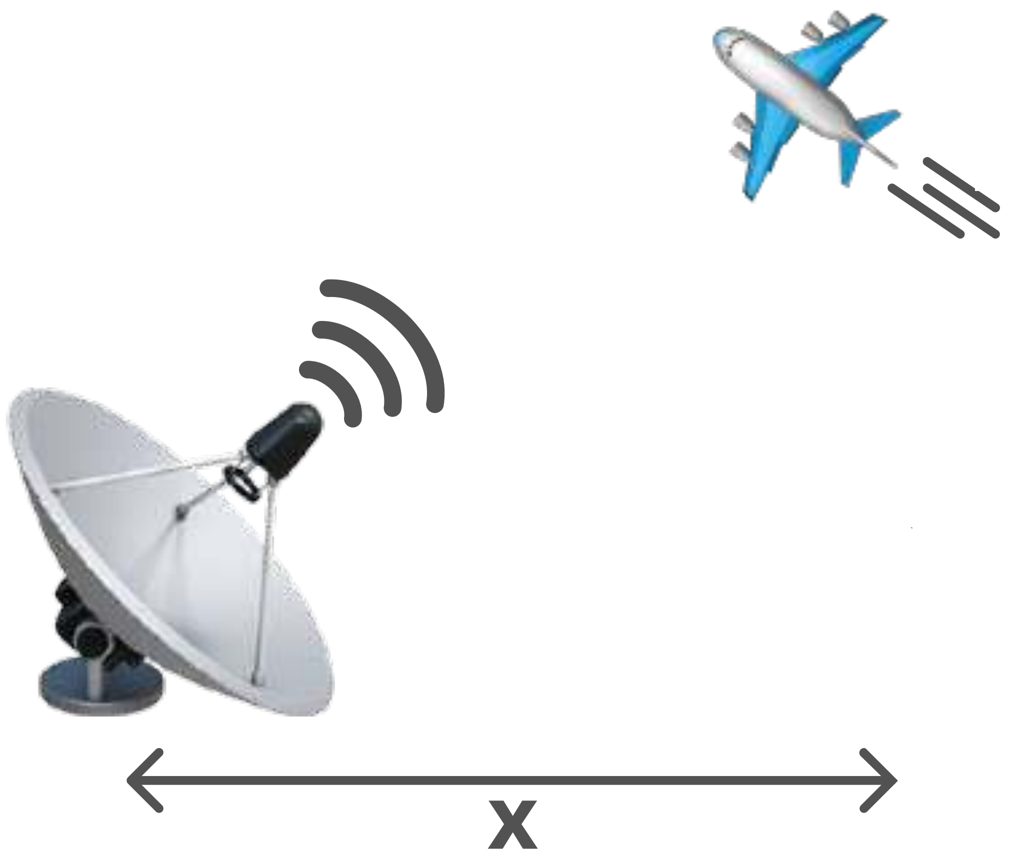
```python
from mu_ppl import *

def coin(obs: list[int]) → float:
  p = sample(Uniform(0, 1))
  for o in obs:
    observe(Bernoulli(p), o)
  return p

with ImportanceSampling(num_particles=1000):
  dist = infer(coin, [0, 0, 0, 0, 0, 0, 0, 0, 1, 1])
  viz(dist)
```

1000 particles

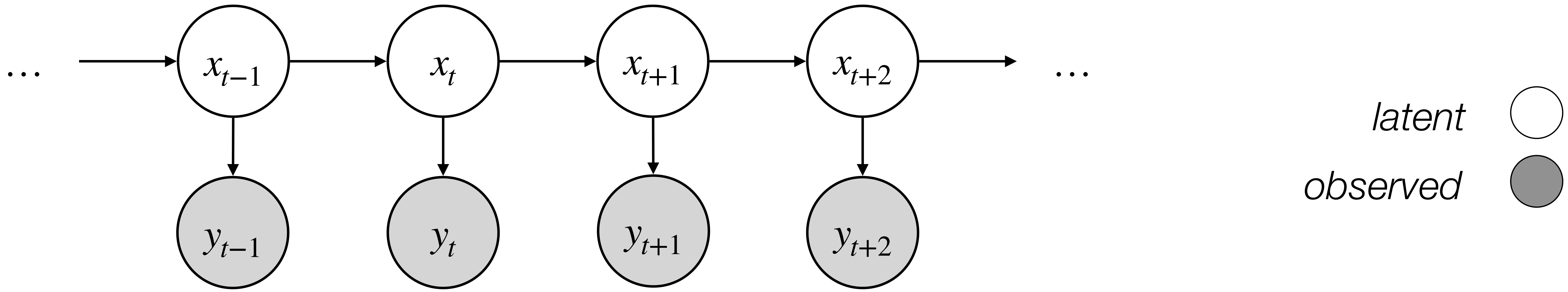


Exact solution: Beta(#heads + 1, #tails + 1)

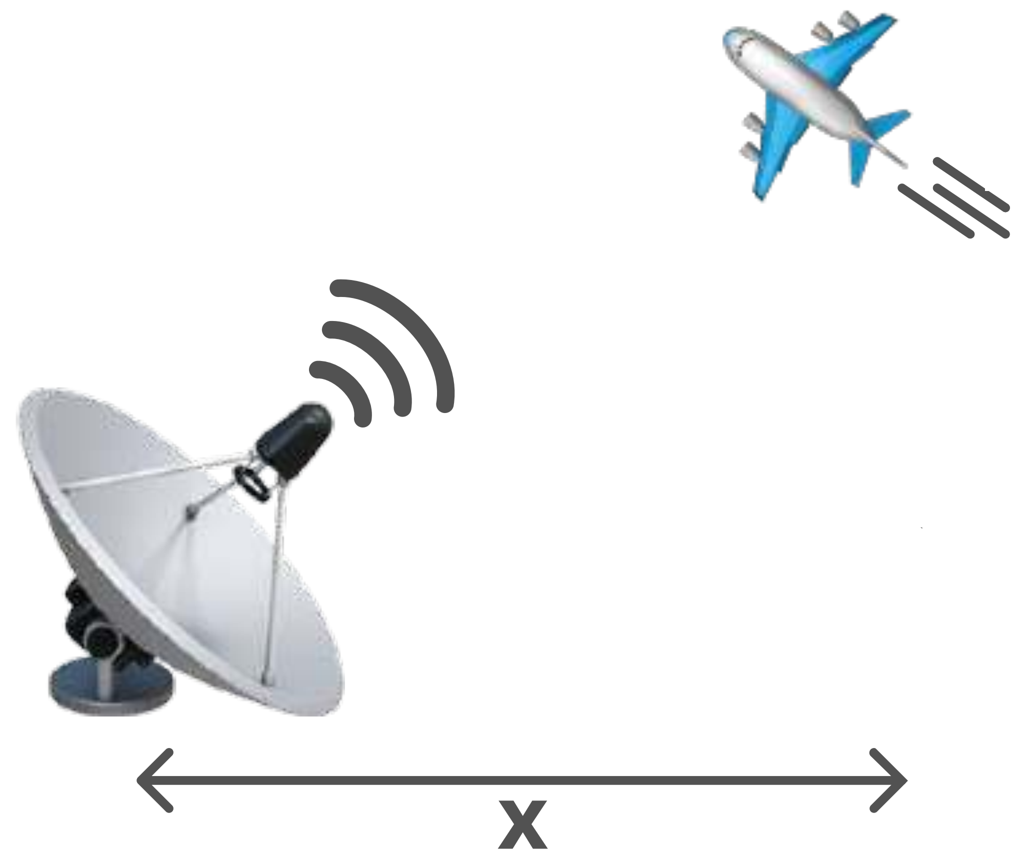
Example: tracking



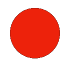

```python
def tracker(data: list[float]) -> list[float]:
  acc = [0]
  for y in data:                          # at each time step
    pre_x = acc[-1]
    x = sample(Gaussian(pre_x, 1))        # random walk
    observe(Gaussian(x, 1), y)            # condition on observation
    acc.append(x)
  return acc
```



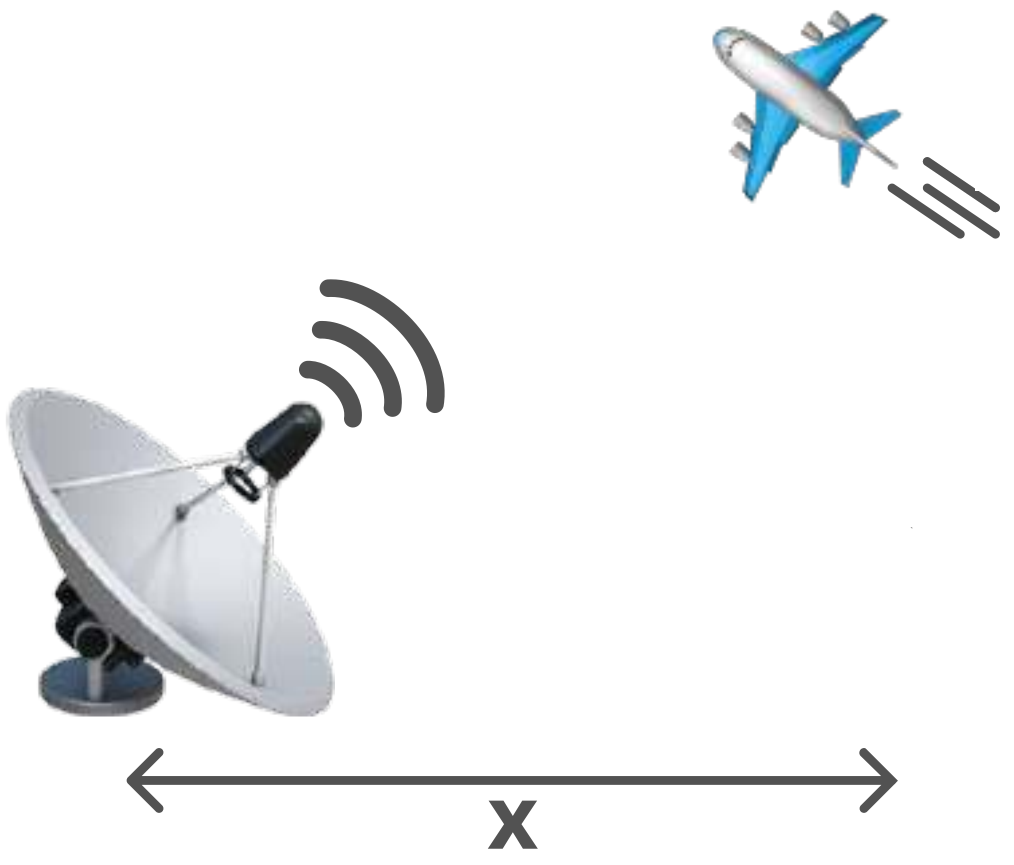
Example: tracking




```python
def tracker(data: list[float]) → list[float]:
  acc = [0]
  for y in data:                          # at each time step
    pre_x = acc[-1]
    x = sample(Gaussian(pre_x, 1))        # random walk
    observe(Gaussian(x, 1), y)            # condition on observation
    acc.append(x)
  return acc
```

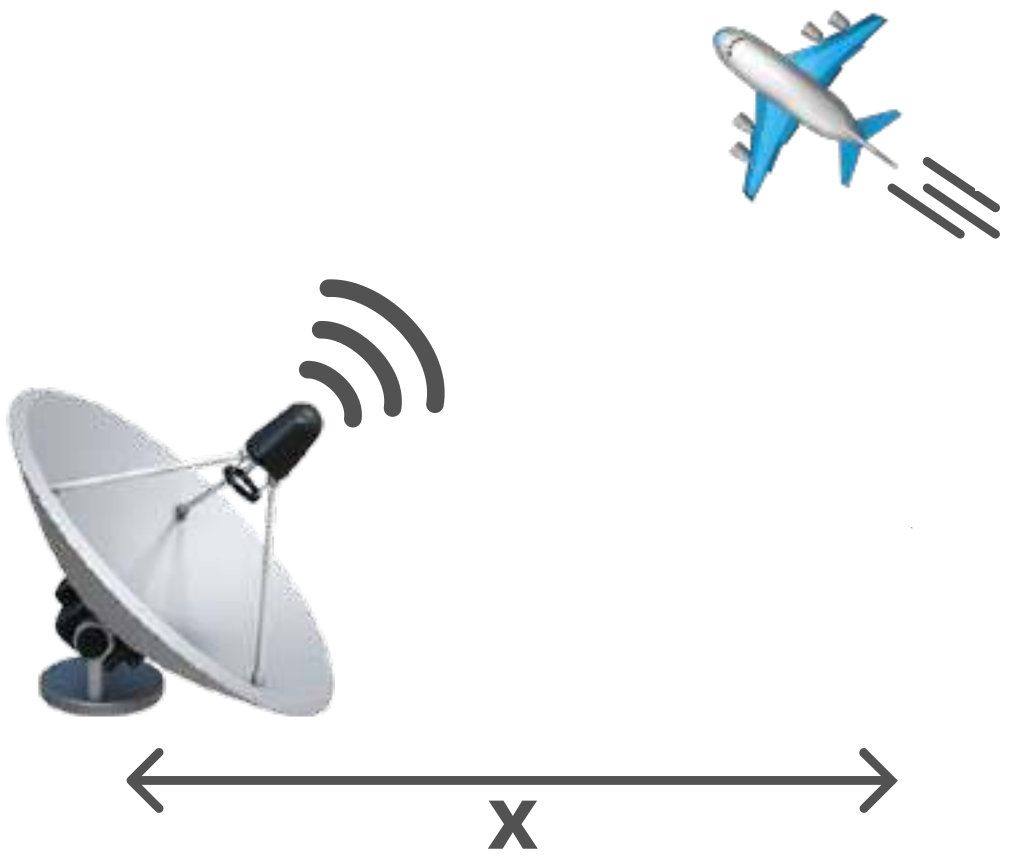


Example: tracking

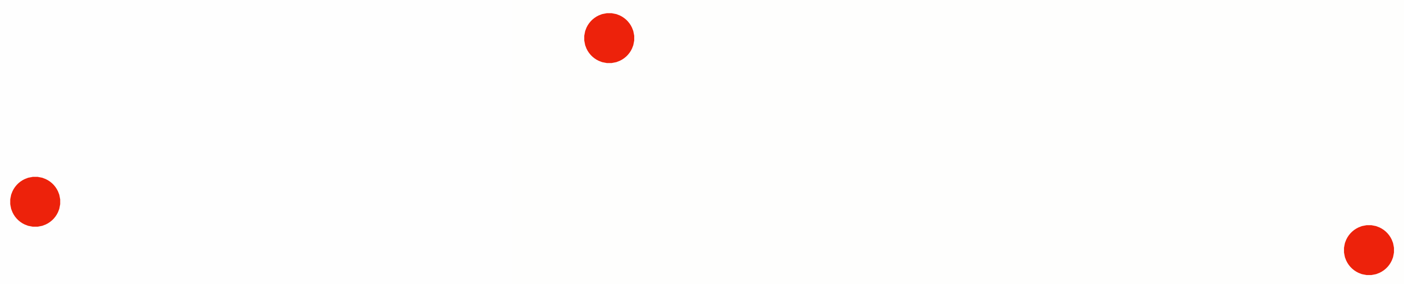



```
def tracker(data: list[float]) → list[float]:
  acc = [0]
  for y in data:                       # at each time step
    pre_x = acc[-1]
    x = sample(Gaussian(pre_x, 1))    # random walk
    observe(Gaussian(x, 1), y)        # condition on observation
    acc.append(x)
  return acc
```

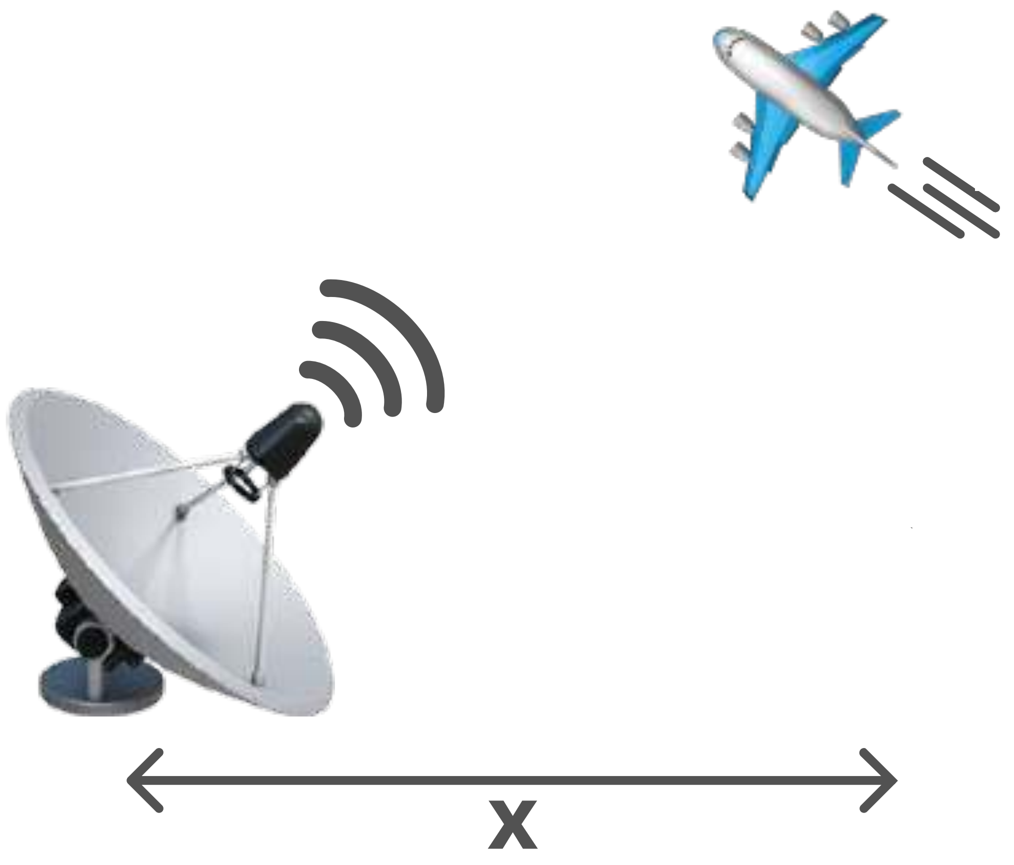
Example: tracking




```
def tracker(data: list[float]) → list[float]:
  acc = [0]
  for y in data:                        # at each time step
    pre_x = acc[-1]
    x = sample(Gaussian(pre_x, 1))      # random walk
    observe(Gaussian(x, 1), y)          # condition on observation
    acc.append(x)
  return acc
```



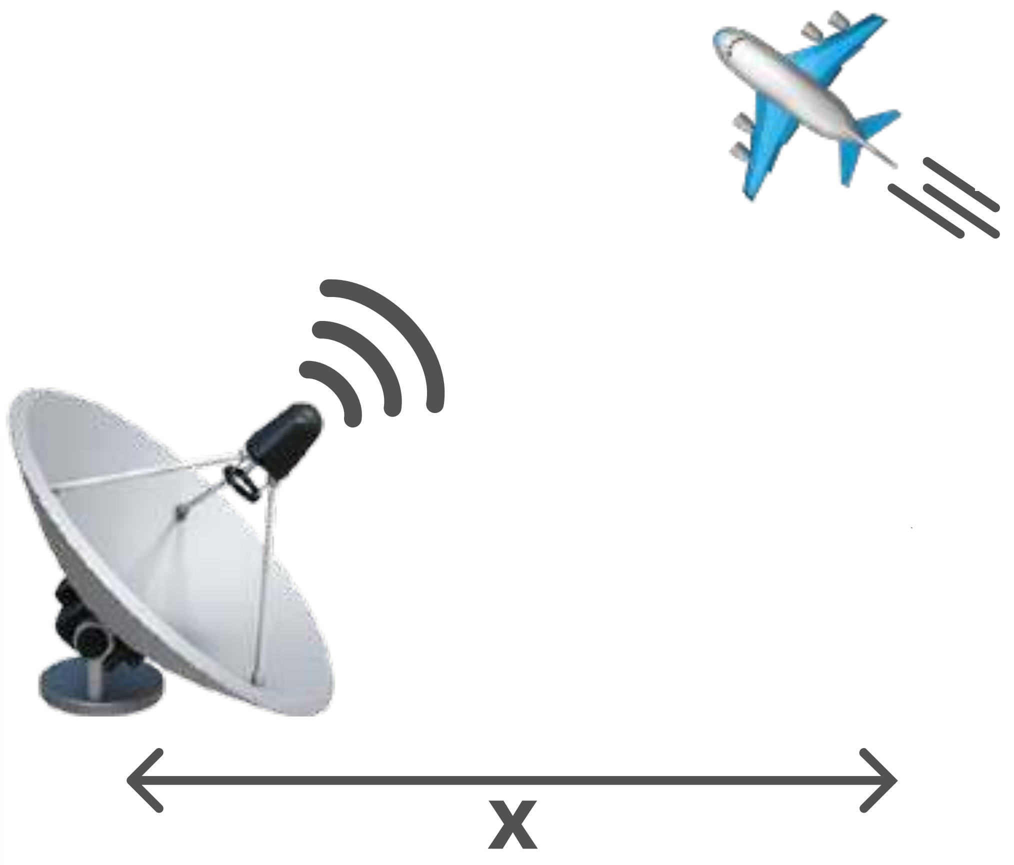
Example: tracking



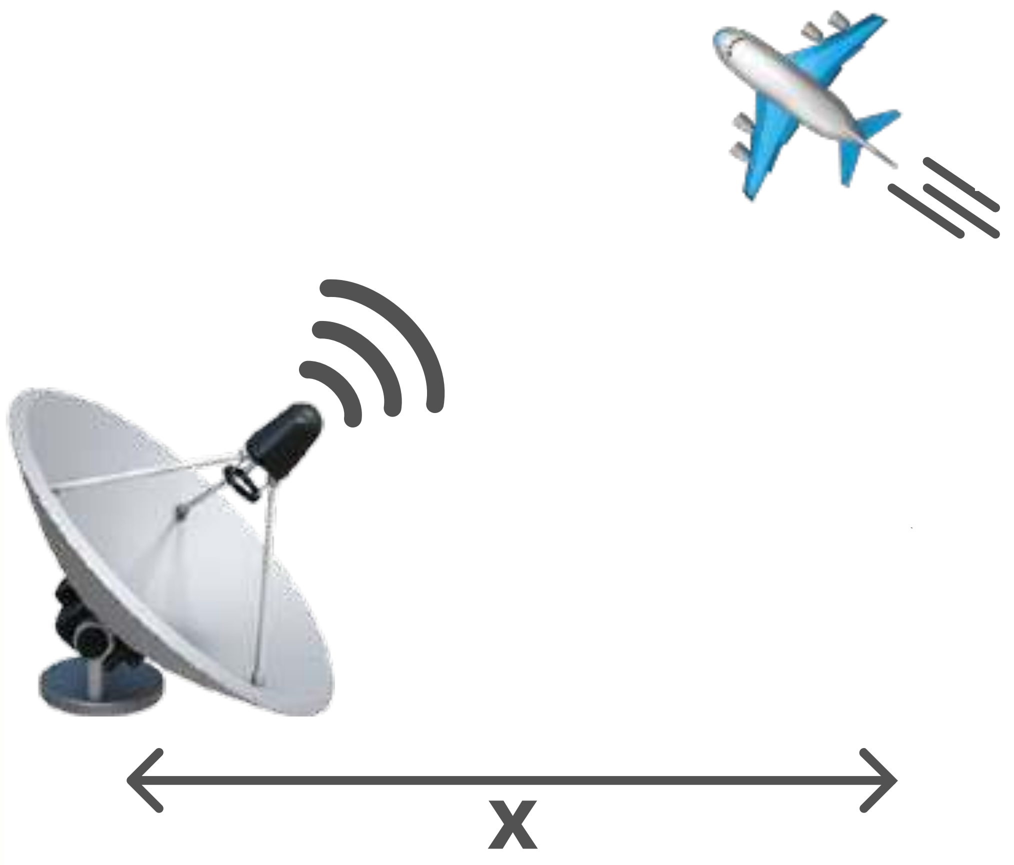

```
def tracker(data: list[float]) → list[float]:
  acc = [0]
  for y in data:                        # at each time step
    pre_x = acc[-1]
    x = sample(Gaussian(pre_x, 1))      # random walk
    observe(Gaussian(x, 1), y)          # condition on observation
    acc.append(x)
  return acc
```

Example: tracking

```python
def tracker(data: list[float]) → list[float]:
  acc = [0]
  for y in data:                          # at each time step
    pre_x = acc[-1]
    x = sample(Gaussian(pre_x, 1))        # random walk
    observe(Gaussian(x, 1), y)            # condition on observation
    acc.append(x)
  return acc
```



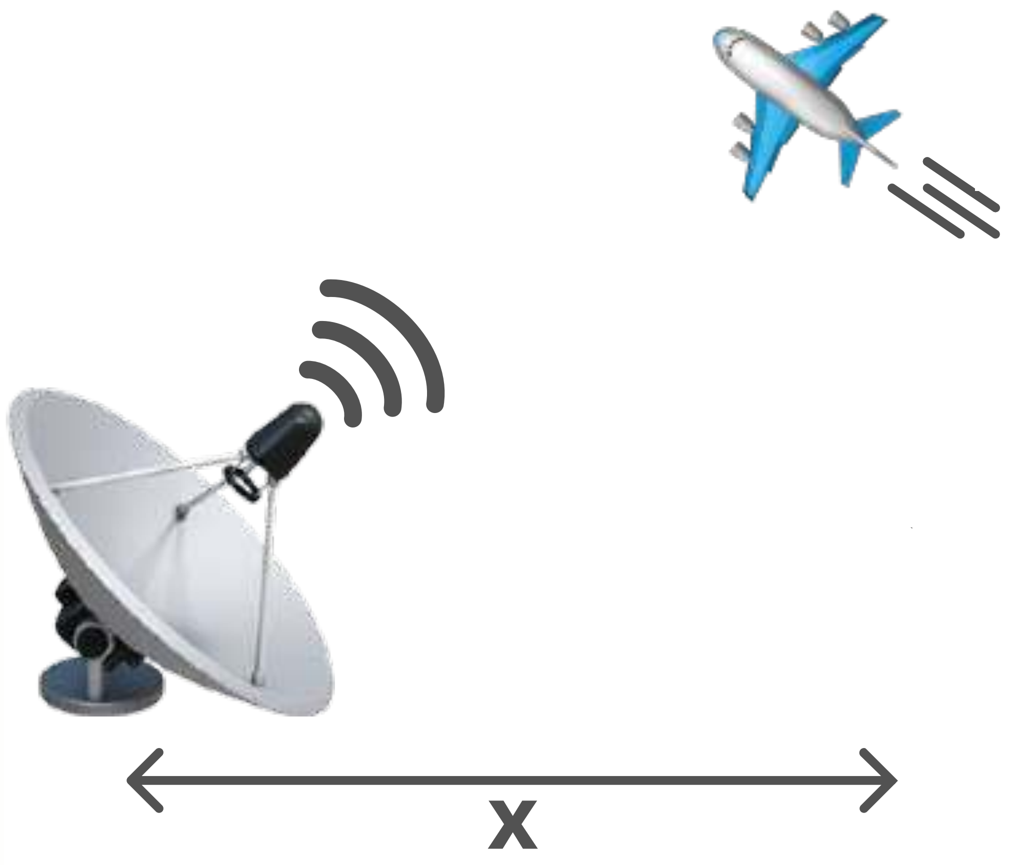
Example: tracking




```python
def tracker(data: list[float]) → list[float]:
  acc = [0]
  for y in data:                         # at each time step
    pre_x = acc[-1]
    x = sample(Gaussian(pre_x, 1))       # random walk
    observe(Gaussian(x, 1), y)           # condition on observation
    acc.append(x)
  return acc
```

Bad estimation

Example: tracking




```python
def tracker(data: list[float]) → list[float]:
  acc = [0]
  for y in data:                    # at each time step
    pre_x = acc[-1]
    x = sample(Gaussian(pre_x, 1))  # random walk
    observe(Gaussian(x, 1), y)      # condition on observation
    acc.append(x)
  return acc
```

Problem:

- Each particle does a random walk and compute a score
- No re-calibration based on observations
- Score decreases at each factor statement

Bad estimation

The curse of dimensionality

Sven Sachsalber, *Looking for a needle in the Haystack*, 2014

The curse of dimensionality

Problem becomes harder as the dimension increases

Basic inference: importance sampling

- Performances decrease exponentially when the dimension increases
- Only use for low-dimension models

Sven Sachsalber, *Looking for a needle in the Haystack*, 2014

The curse of dimensionality

Problem becomes harder as the dimension increases

Basic inference: importance sampling

- Performances decrease exponentially when the dimension increases
- Only use for low-dimension models

17h45mn

Sven Sachsalber, *Looking for a needle in the Haystack*, 2014

The curse of dimensionality

Problem becomes harder as the dimension increases

Basic inference: importance sampling

- Performances decrease exponentially when the dimension increases
- Only use for low-dimension models

How to mitigate this problem?

- Make assumptions about the posterior distributions
- Break the problem into simpler, smaller problems

17h45mn

Sven Sachsalber, *Looking for a needle in the Haystack*, 2014

Particle filter

Particle filter

Inference algorithm

- Run a set of n independent executions
- `sample`: draw a sample from a distribution
- `factor`: associate a score to the current execution
- `infer`: gather output values v_i and score W_i to approximate the posterior distribution

$$p_i = \frac{W_i}{\sum_{1 \leq i \leq n} W_i}$$

Conditioning

- `assume(p)`: sets the score to 0 if `p` is false
- `observe(d,v)`: multiply the score by the likelihood of `d` at `v` (density function of `d`)

Particle filter

Inference algorithm

- Run a set of n independent executions
- `sample`: draw a sample from a distribution
- `factor`: associate a score to the current execution
- `infer`: gather output values v_i and score W_i to approximate the posterior distribution

$$p_i = \frac{W_i}{\sum_{1 \leq i \leq n} W_i}$$

Conditioning

- `assume(p)`: sets the score to 0 if `p` is false
- `observe(d,v)`: multiply the score by the likelihood of `d` at `v` (density function of `d`)

Add a resampling step

- Checkpoint: stop the particles in the middle of the execution
- Compute the score of the particles to compute a distribution
- Re-sample a new set of particles from this distribution

Particle filter

0

```
def model(data):
  ...
  x = sample ...
  ...
  factor ...
```

```
  ...
  return output
```

1

```
def model(data):
  ...
  x = sample ...
  ...
  factor ...
```

```
  ...
  return output
```

2

```
def model(data):
  ...
  x = sample ...
  ...
  factor ...
```

```
  ...
  return output
```

Particle filter

0

```
def model(data):
   ...
   x = sample  ...
   ...
   factor  ...
```

```
   ...
   return output
```

1

```
def model(data):
   ...
   x = sample  ...
   ...
   factor  ...
```

```
   ...
   return output
```

2

```
def model(data):
   ...
   x = sample  ...
   ...
   factor  ...
```

```
   ...
   return output
```

0.66

Particle filter

0

```
def model(data):
  ...
  x = sample  ...
  ...
  factor  ...
```

```
  ...
  return output
```

1

```
def model(data):
  ...
  x = sample  ...
  ...
  factor  ...
```

```
  ...
  return output
```

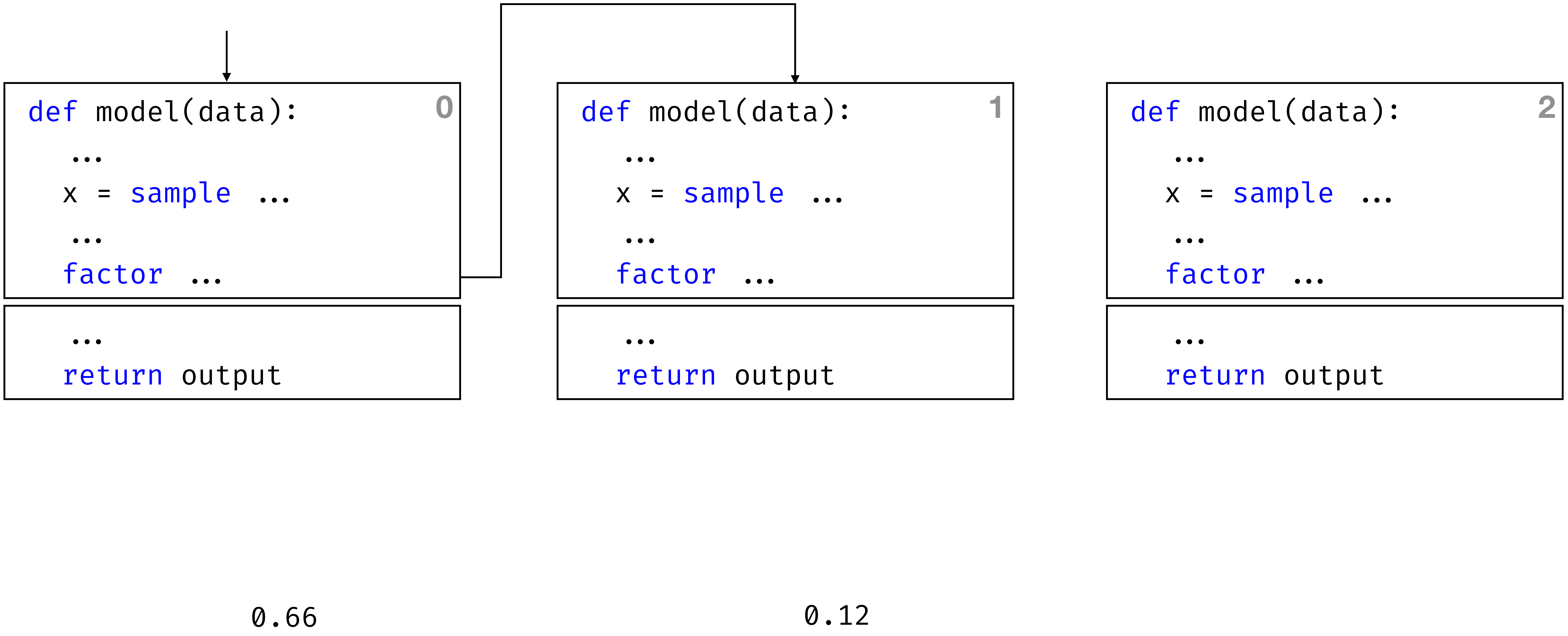
2

```
def model(data):
  ...
  x = sample  ...
  ...
  factor  ...
```

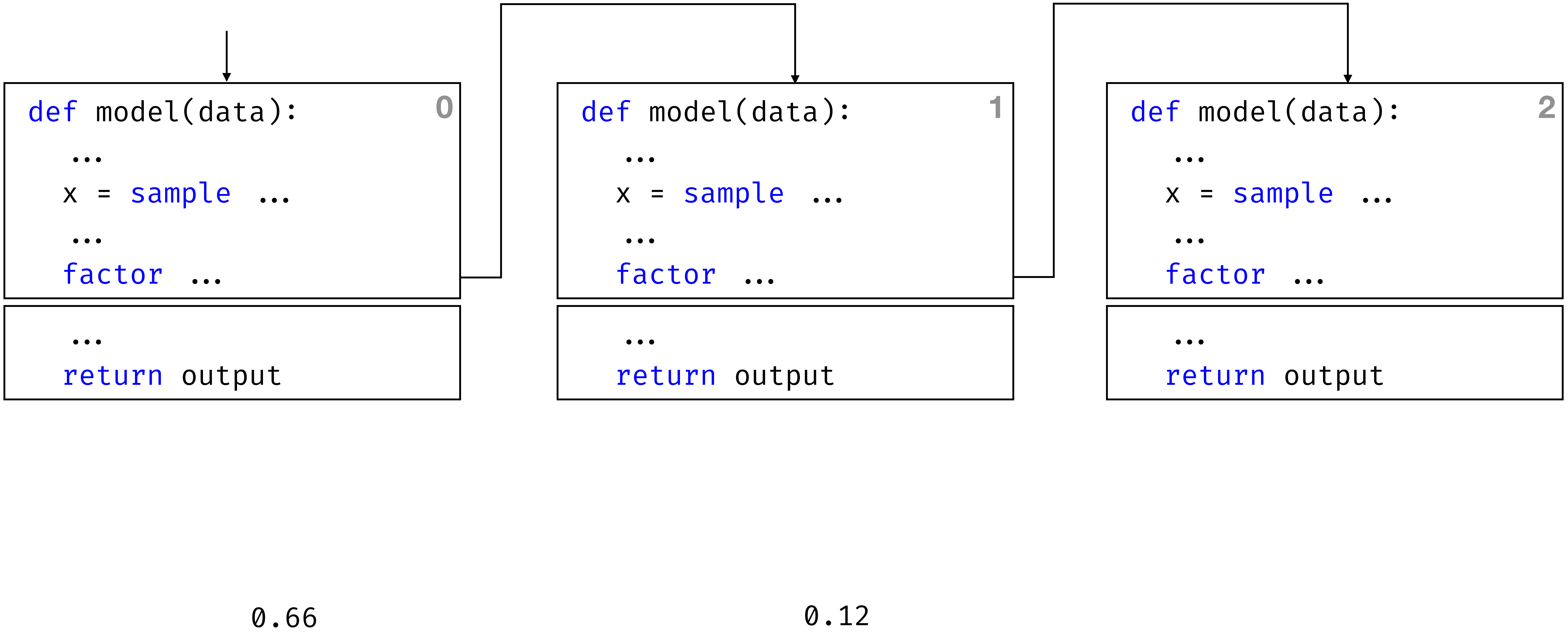
```
  ...
  return output
```

0.66

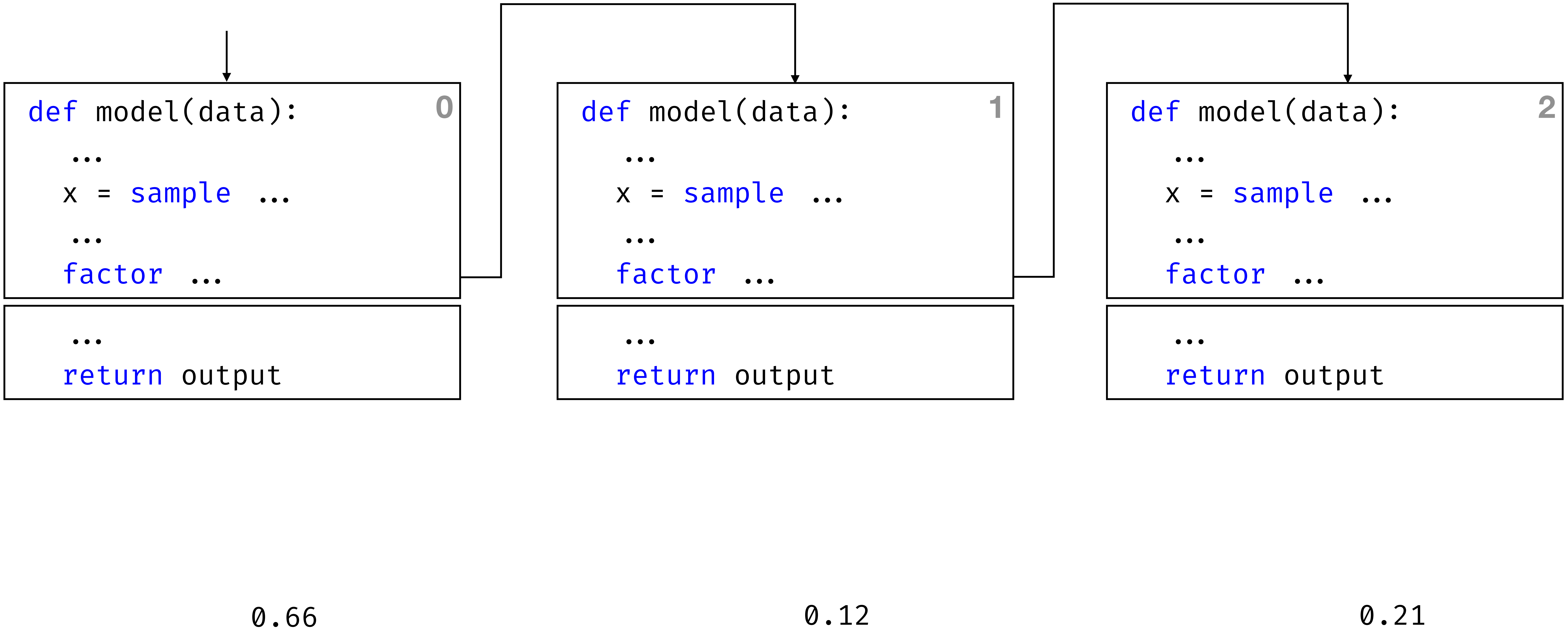
Particle filter



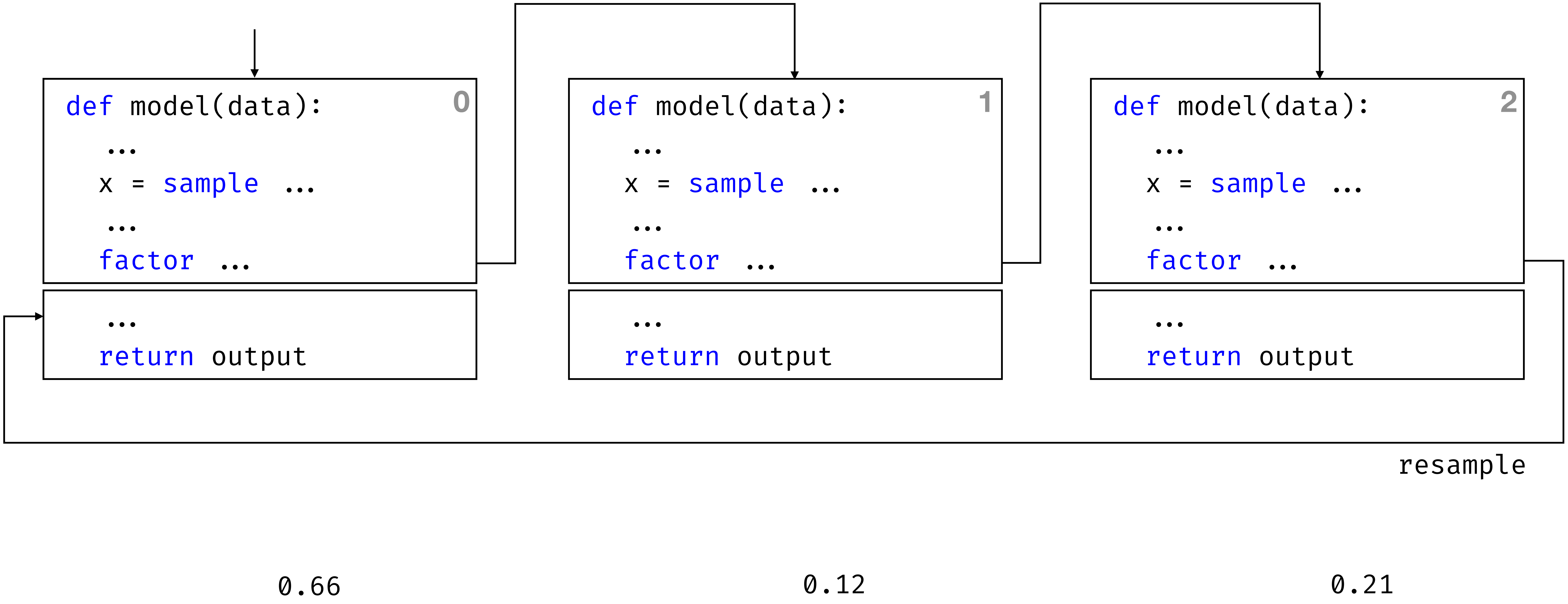
Particle filter



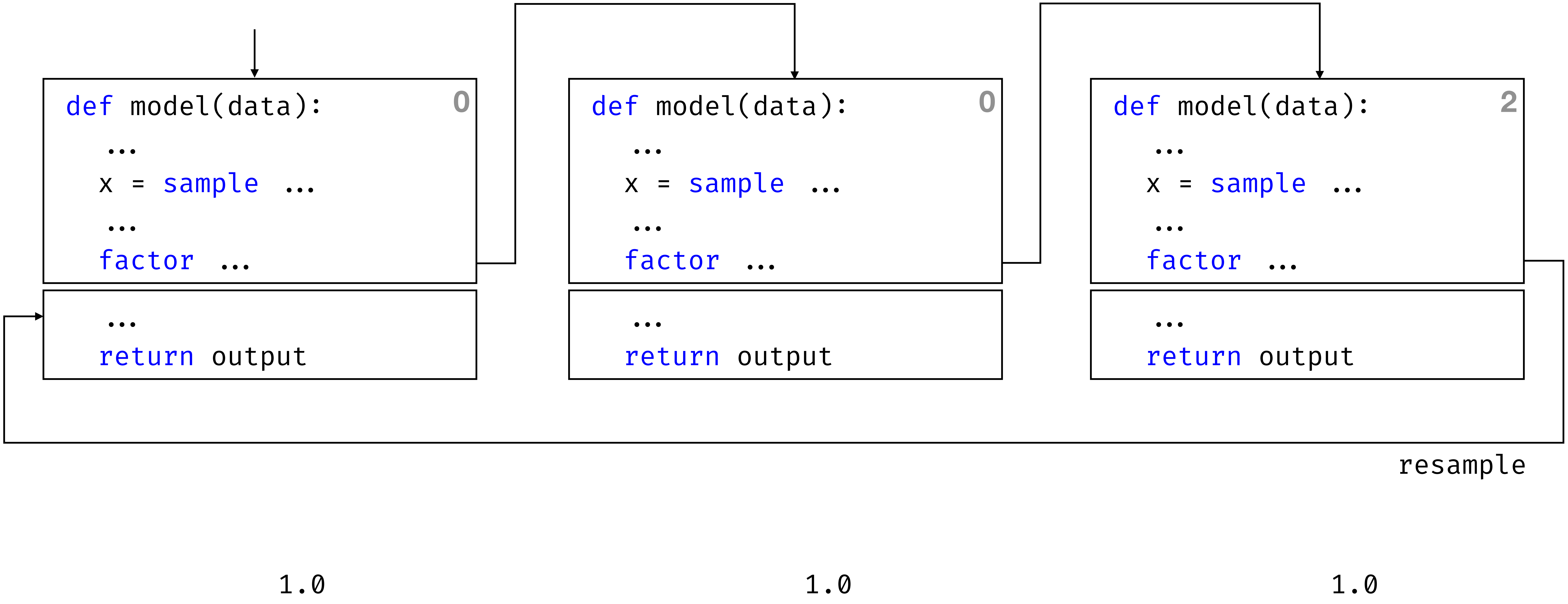
Particle filter



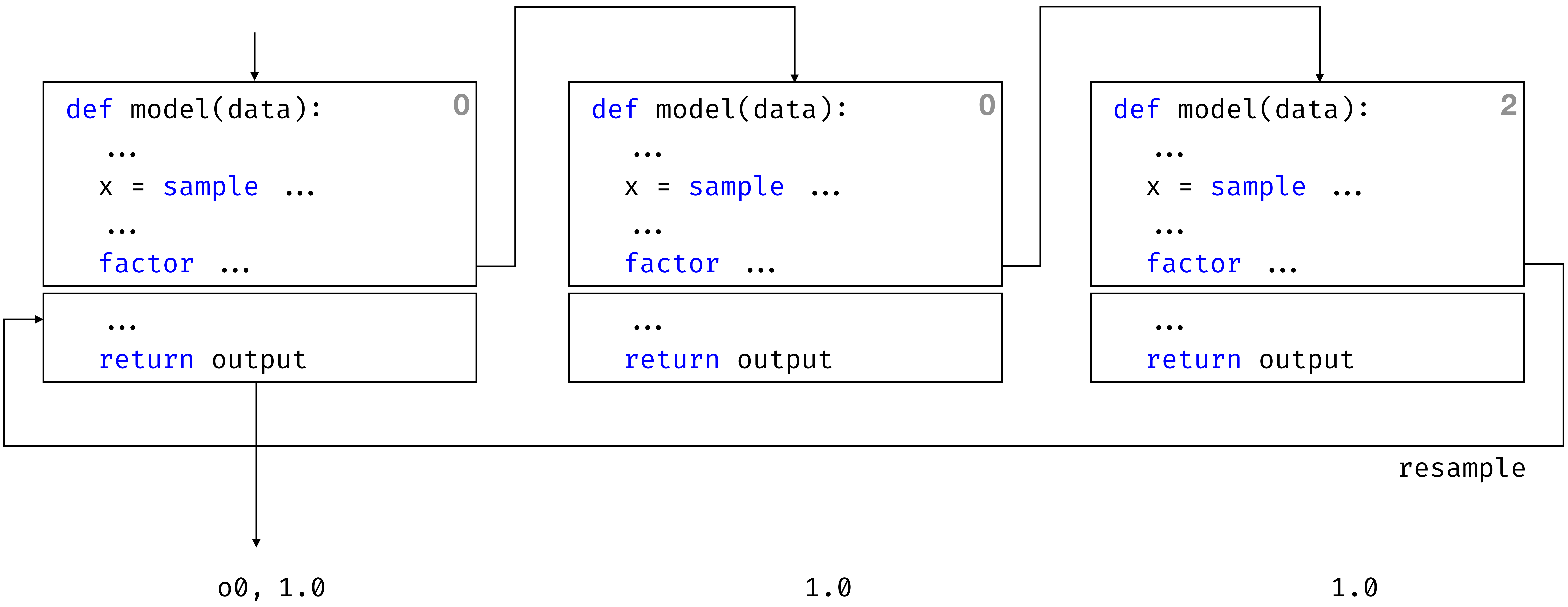
Particle filter



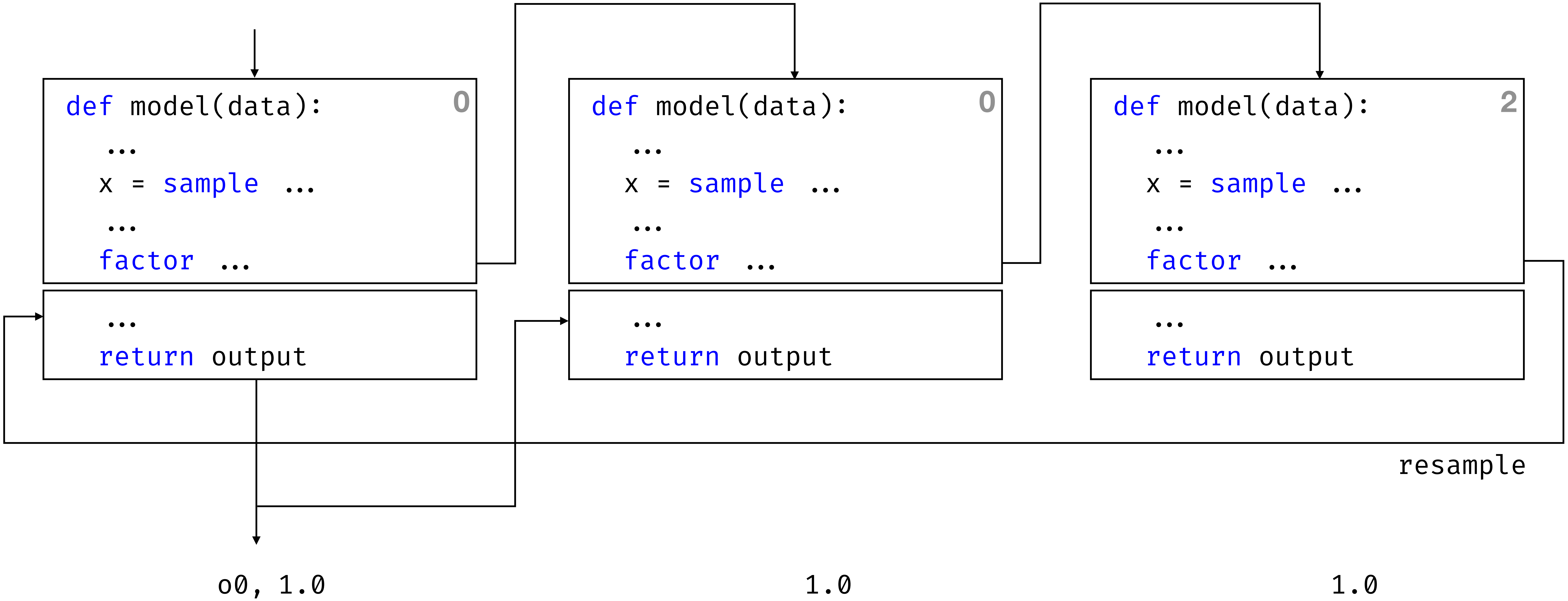
Particle filter



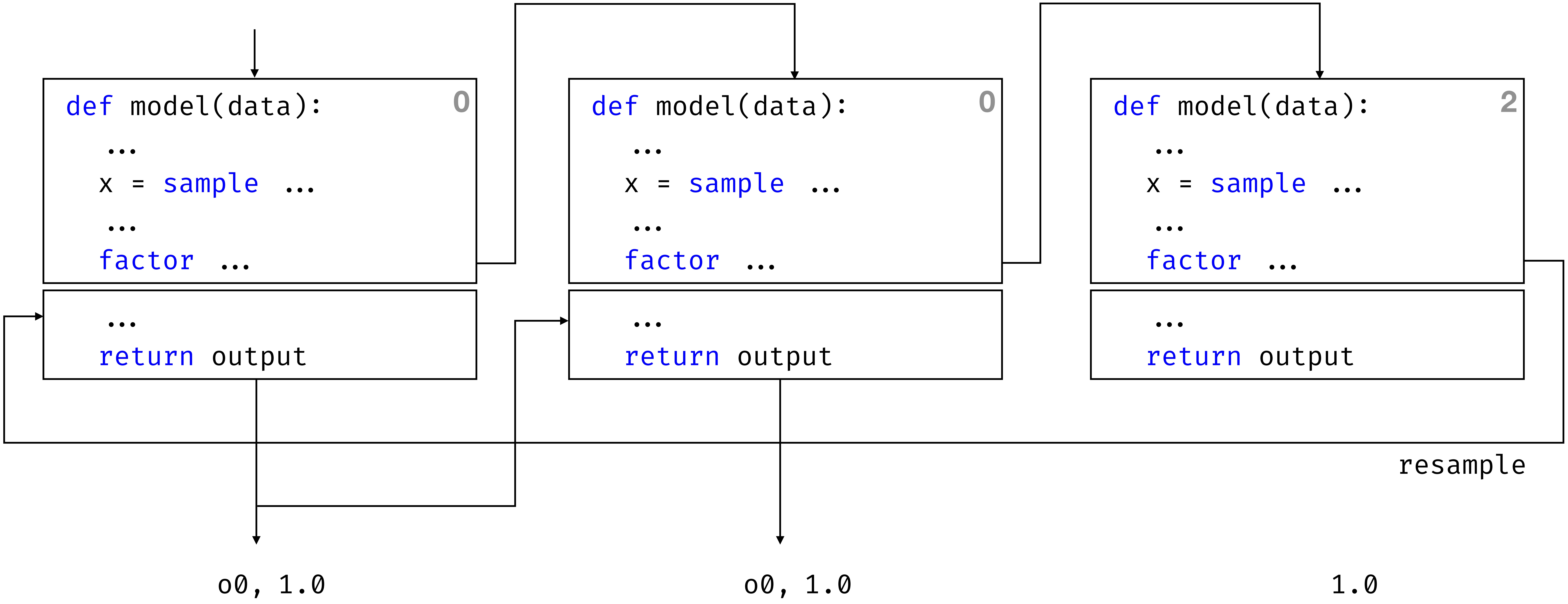
Particle filter



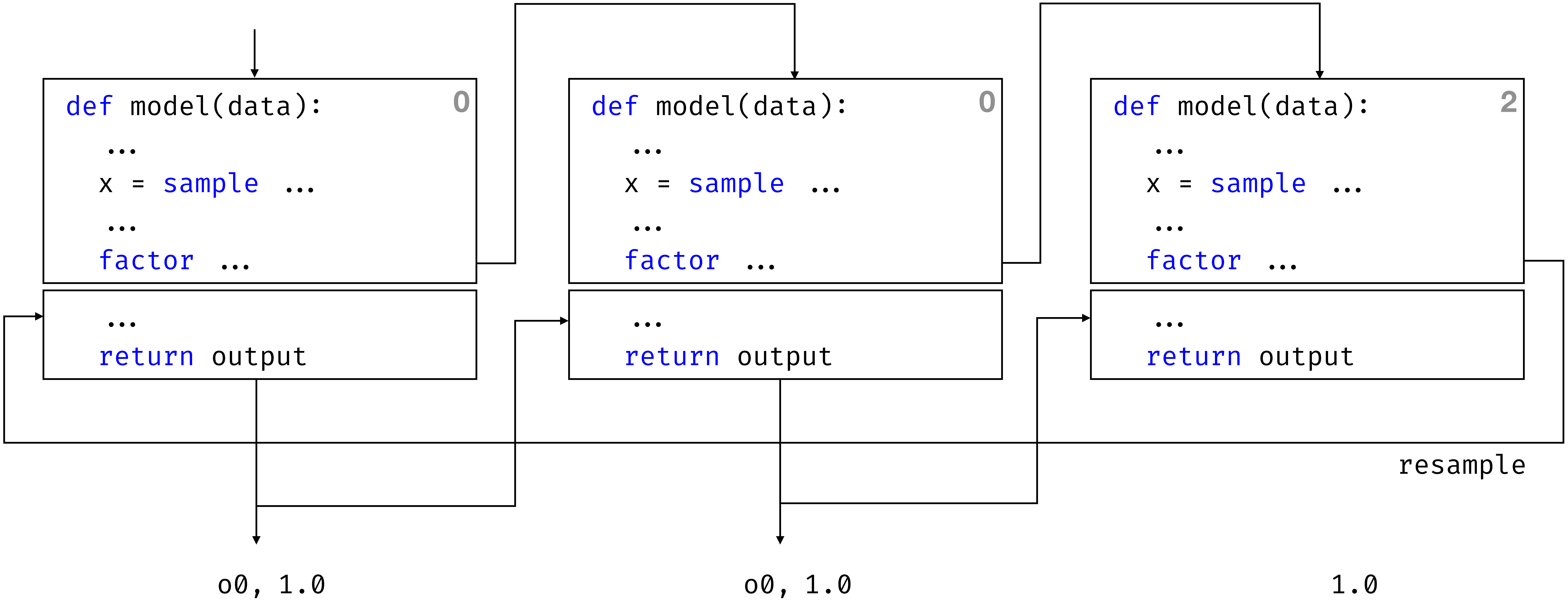
Particle filter



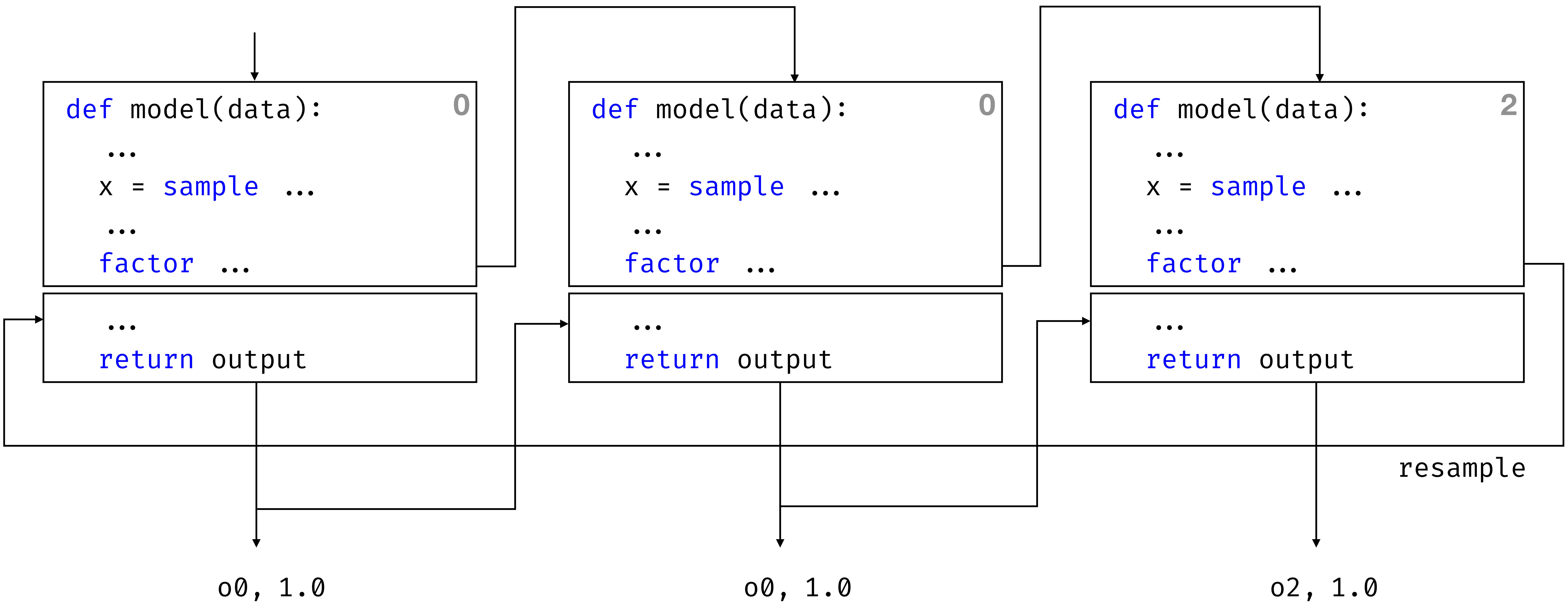
Particle filter



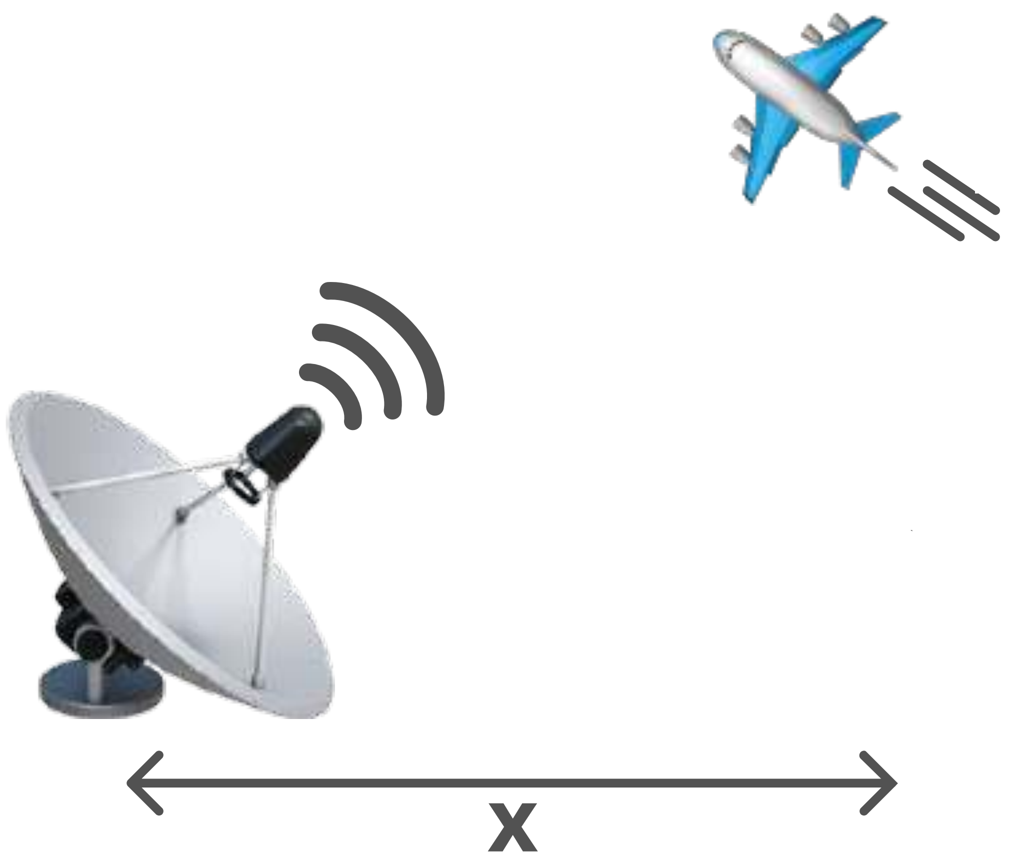
Particle filter



Particle filter



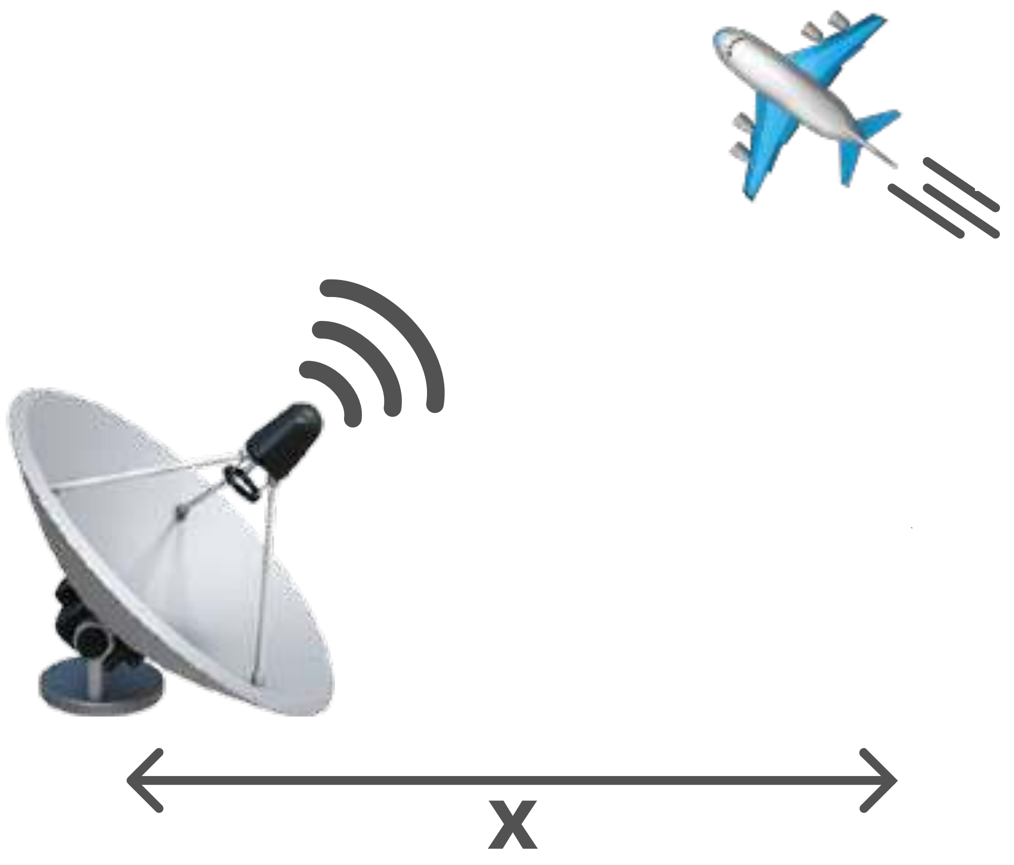
Example: tracking



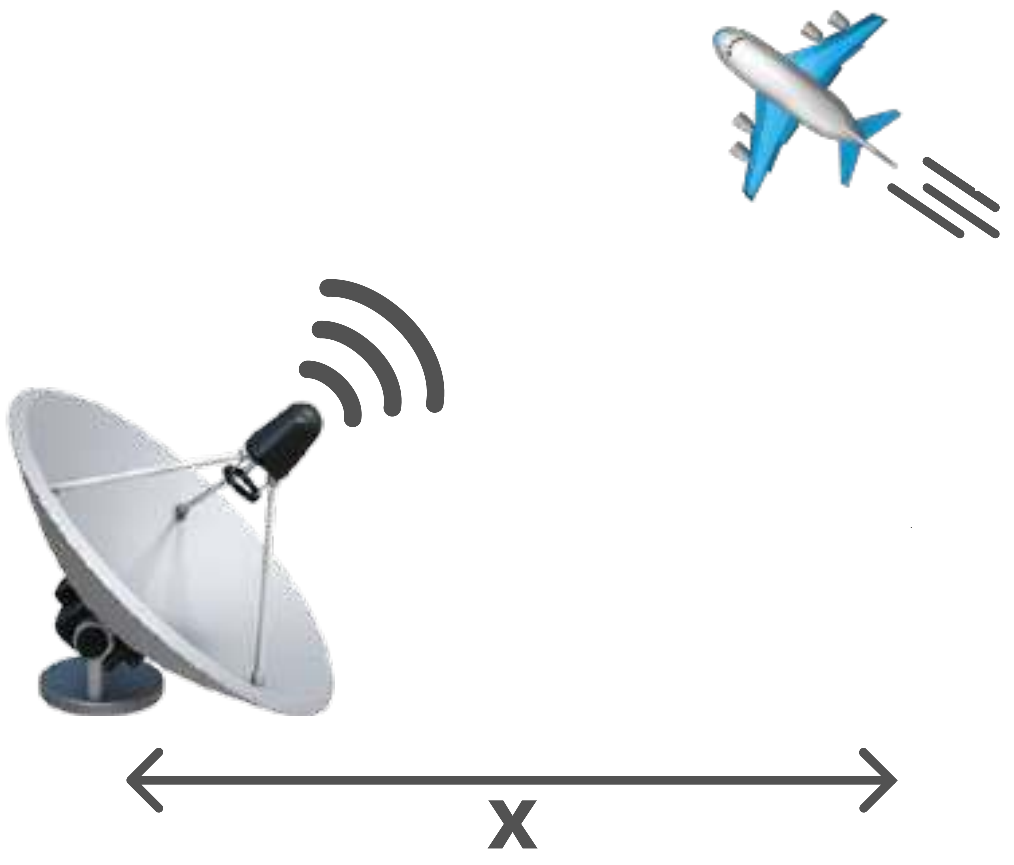

```
def tracker(data: list[float]) → list[float]:
  acc = [0]
  for y in data:                        # at each time step
    pre_x = acc[-1]
    x = sample(Gaussian(pre_x, 1))      # random walk
    observe(Gaussian(x, 1), y)          # condition on observation
    acc.append(x)
  return acc
```

Example: tracking

```
def tracker(data: list[float]) → list[float]:
  acc = [0]
  for y in data:                           # at each time step
    pre_x = acc[-1]
    x = sample(Gaussian(pre_x, 1))         # random walk
    observe(Gaussian(x, 1), y)             # condition on observation
    acc.append(x)
  return acc
```

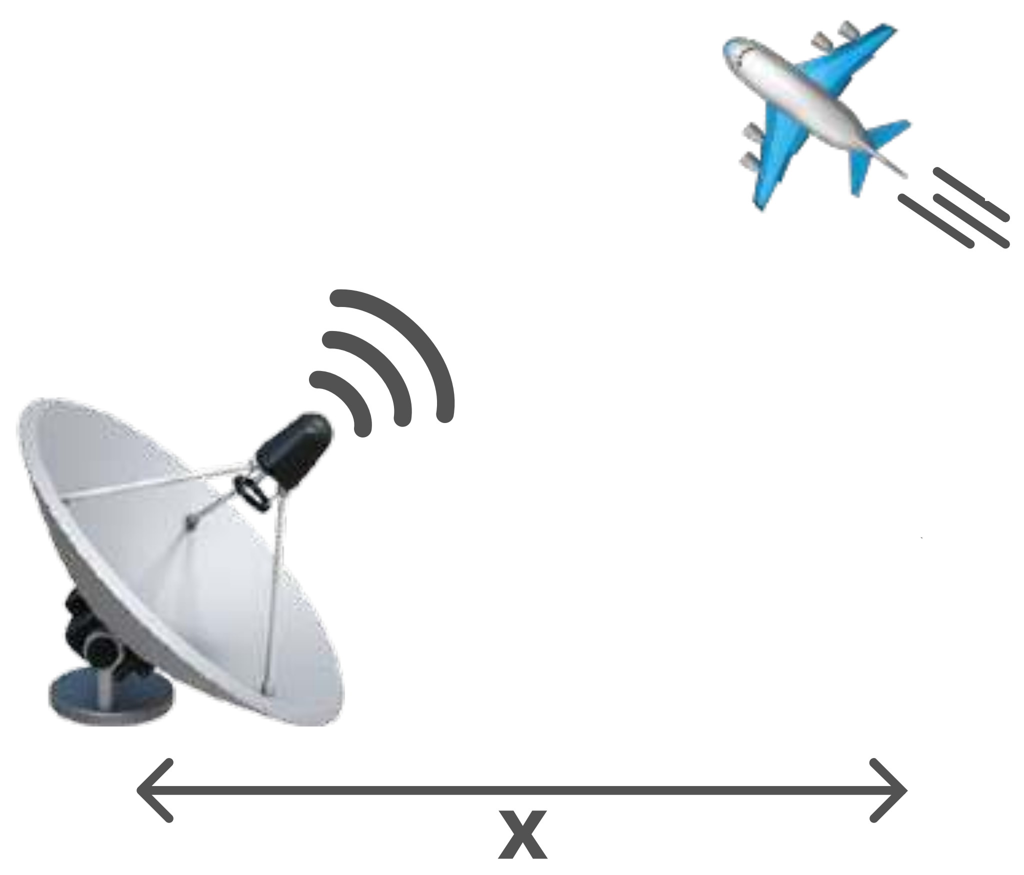


Example: tracking

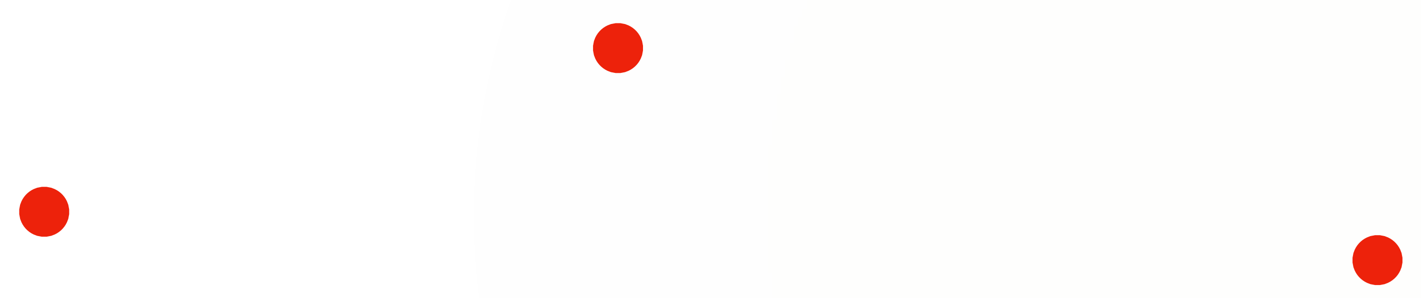



```python
def tracker(data: list[float]) → list[float]:
  acc = [0]
  for y in data:                          # at each time step
    pre_x = acc[-1]
    x = sample(Gaussian(pre_x, 1))        # random walk
    observe(Gaussian(x, 1), y)            # condition on observation
    acc.append(x)
  return acc
```

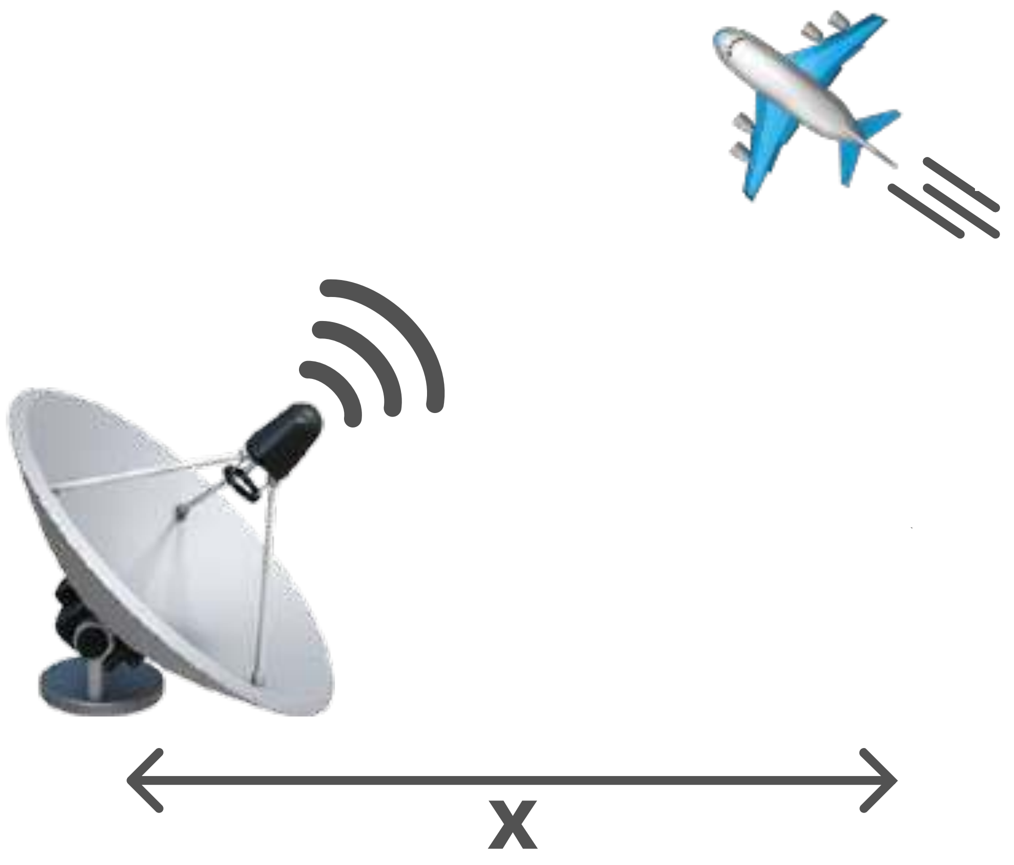
Example: tracking



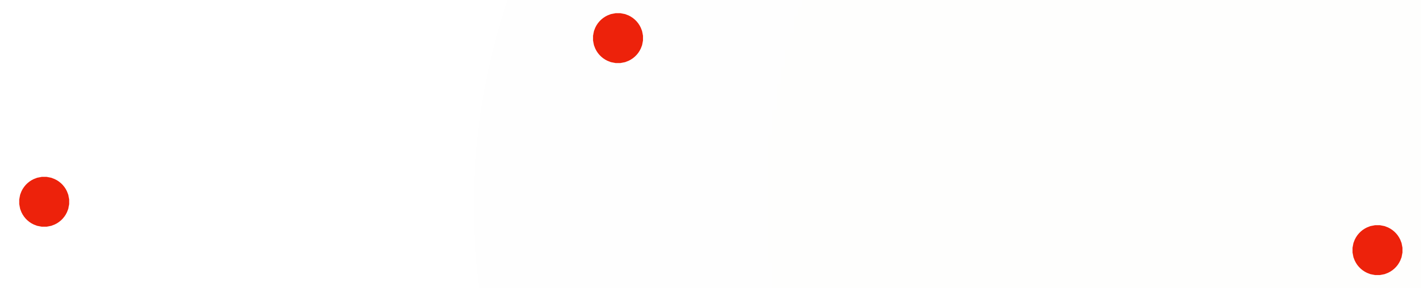

```
def tracker(data: list[float]) → list[float]:
  acc = [0]
  for y in data:                        # at each time step
    pre_x = acc[-1]
    x = sample(Gaussian(pre_x, 1))      # random walk
    observe(Gaussian(x, 1), y)          # condition on observation
    acc.append(x)
  return acc
```



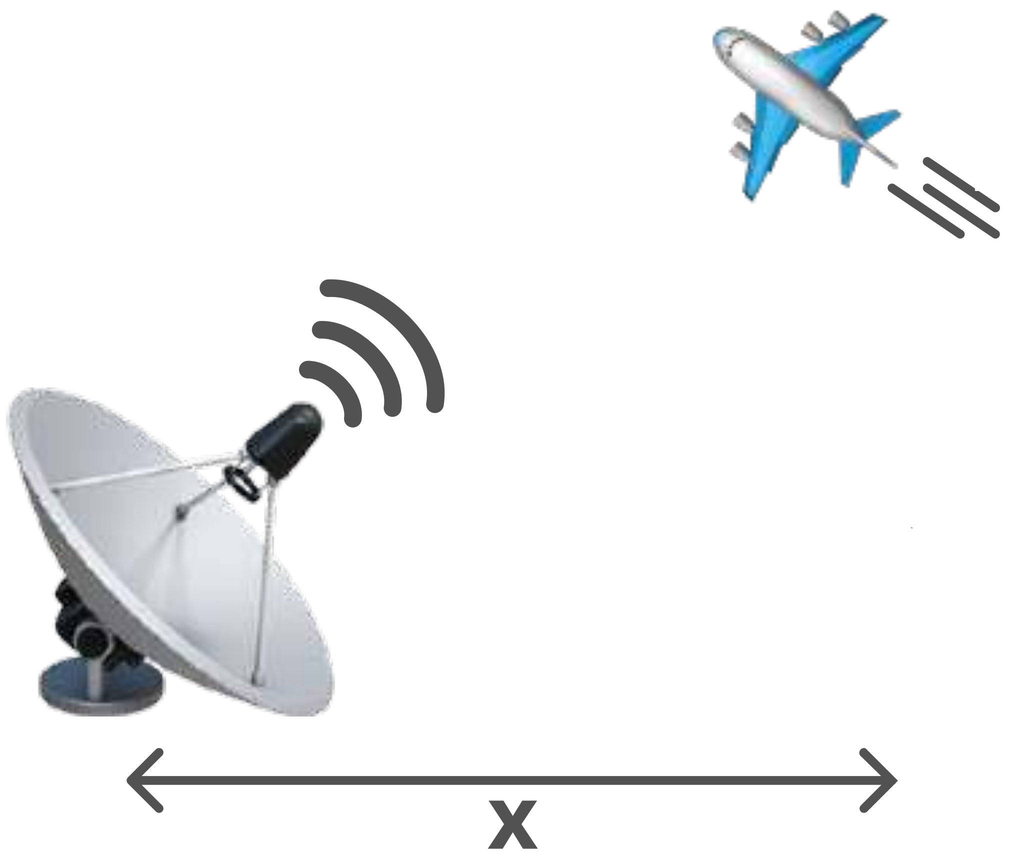
Example: tracking




```python
def tracker(data: list[float]) -> list[float]:
  acc = [0]
  for y in data:                          # at each time step
    pre_x = acc[-1]
    x = sample(Gaussian(pre_x, 1))        # random walk
    observe(Gaussian(x, 1), y)            # condition on observation
    acc.append(x)
  return acc
```

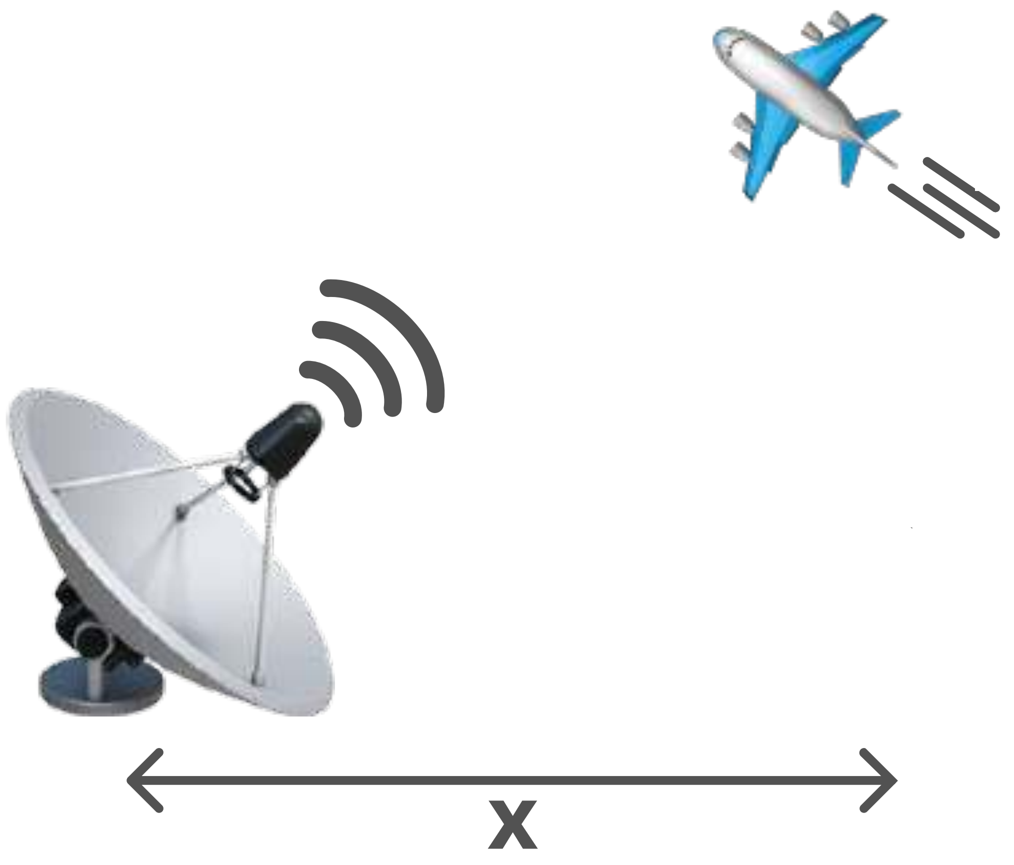


Example: tracking



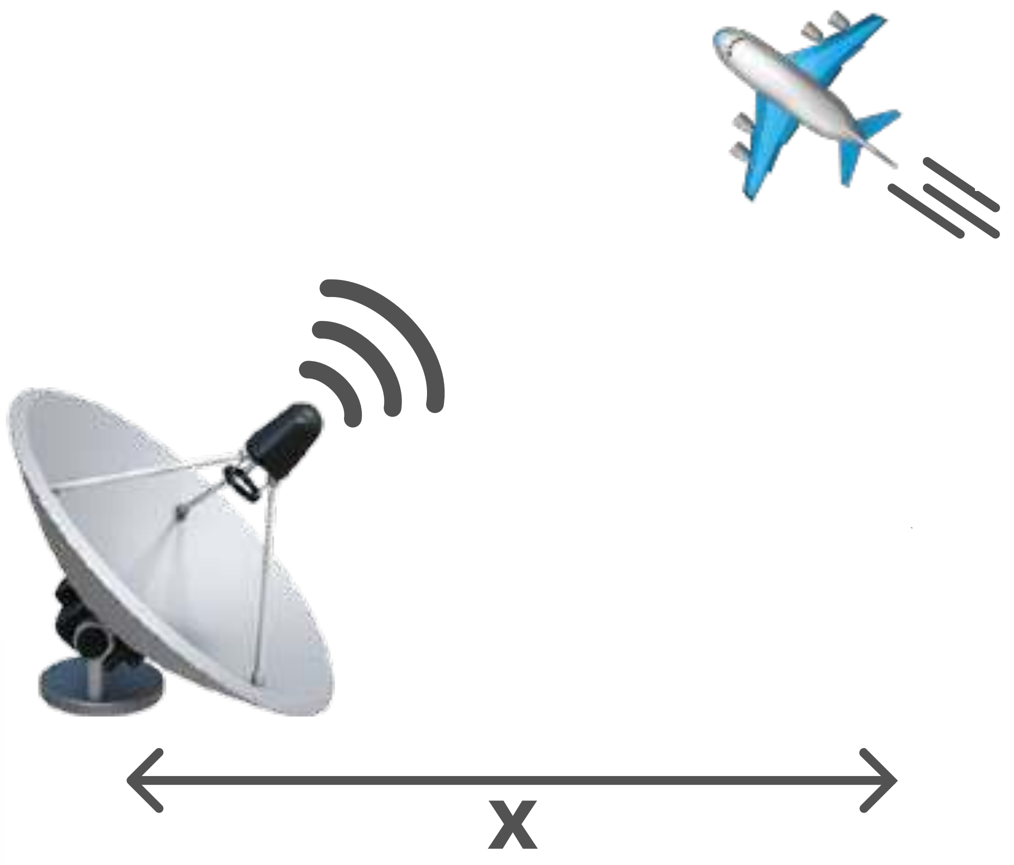

```
def tracker(data: list[float]) → list[float]:
  acc = [0]
  for y in data:                        # at each time step
    pre_x = acc[-1]
    x = sample(Gaussian(pre_x, 1))      # random walk
    observe(Gaussian(x, 1), y)          # condition on observation
    acc.append(x)
  return acc
```

Example: tracking

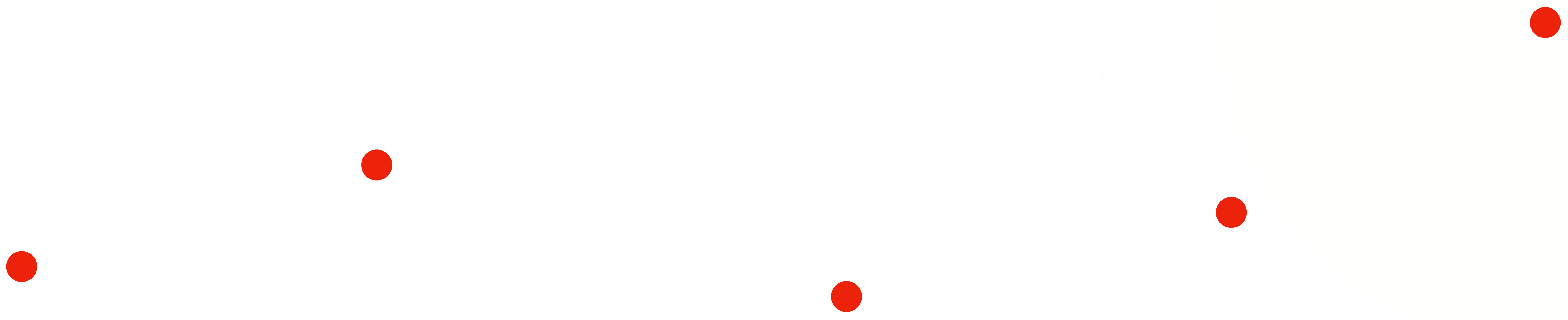



```python
def tracker(data: list[float]) -> list[float]:
  acc = [0]
  for y in data:                          # at each time step
    pre_x = acc[-1]
    x = sample(Gaussian(pre_x, 1))        # random walk
    observe(Gaussian(x, 1), y)            # condition on observation
    acc.append(x)
  return acc
```

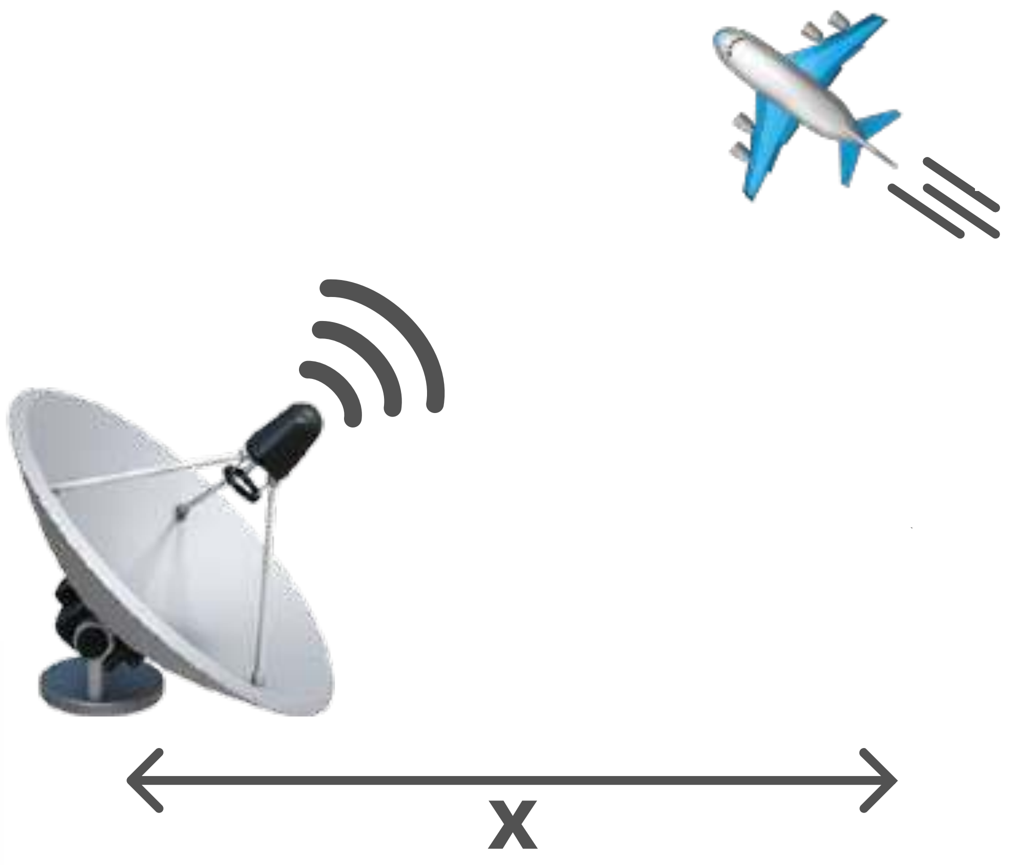
Example: tracking



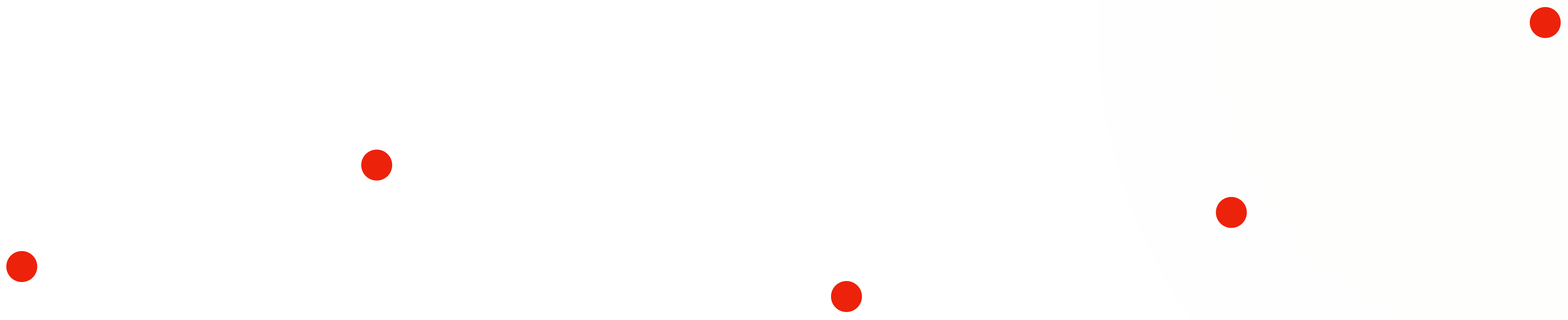

```
def tracker(data: list[float]) → list[float]:
  acc = [0]
  for y in data:                         # at each time step
    pre_x = acc[-1]
    x = sample(Gaussian(pre_x, 1))       # random walk
    observe(Gaussian(x, 1), y)           # condition on observation
    acc.append(x)
  return acc
```



Example: tracking




```python
def tracker(data: list[float]) → list[float]:
  acc = [0]
  for y in data:                       # at each time step
    pre_x = acc[-1]
    x = sample(Gaussian(pre_x, 1))     # random walk
    observe(Gaussian(x, 1), y)         # condition on observation
    acc.append(x)
  return acc
```



Better estimation

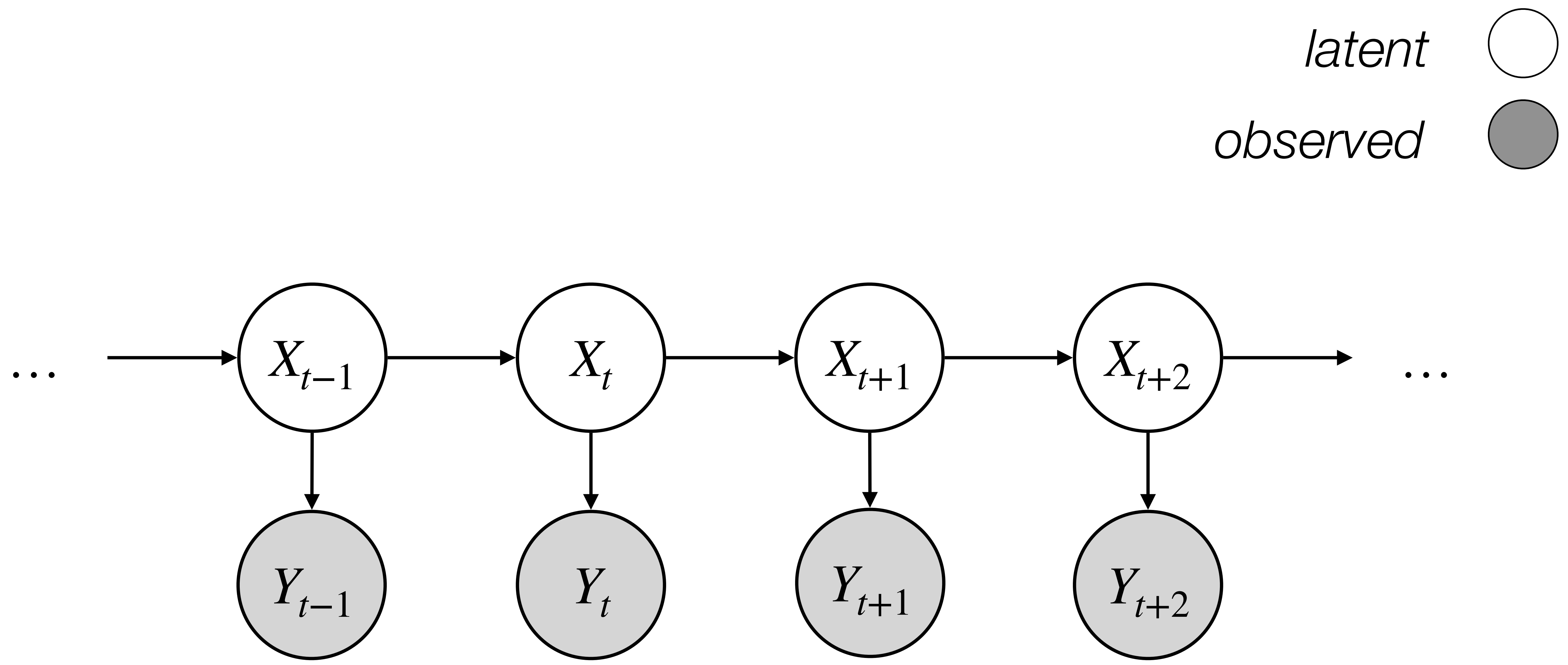
Digression: Kalman filter

Probabilistic Programming Languages

Example: tracker

Model

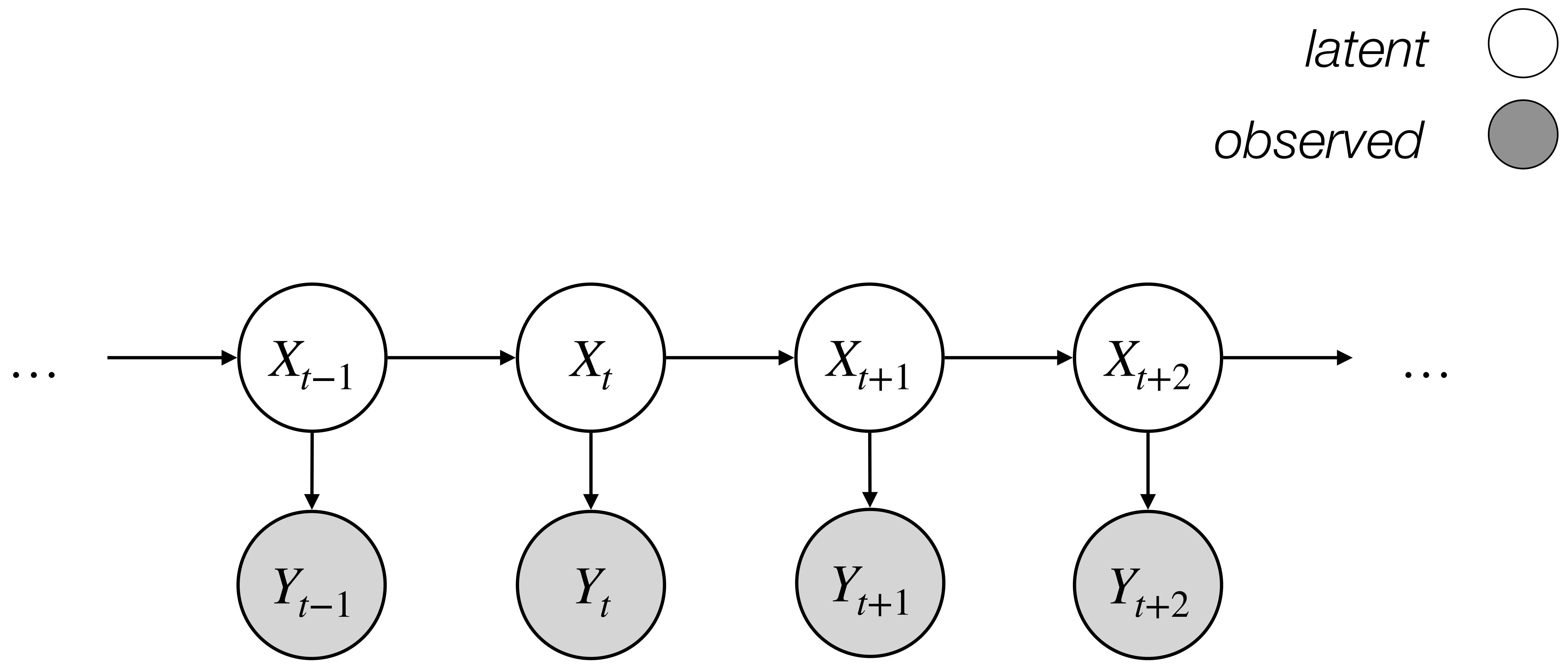
- Linear motion: $X_k \sim \mathcal{N}(FX_{k-1}, Q)$
- Observation: $Y_k \sim \mathcal{N}(HX_k, R)$



E.g., with Q and R constant noise matrices

- $X_k = \begin{pmatrix} p_k \\ v_k \end{pmatrix}$ (position, velocity)
- $F = \begin{pmatrix} 1 & dt \\ 0 & 1 \end{pmatrix}$ (discrete integration)
- $H = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ (projection)

Example: tracker



Model

- Linear motion: $X_k \sim \mathcal{N}(FX_{k-1}, Q)$
- Observation: $Y_k \sim \mathcal{N}(HX_k, R)$

E.g., with Q and R constant noise matrices

- $X_k = \begin{pmatrix} p_k \\ v_k \end{pmatrix}$ (position, velocity)
- $F = \begin{pmatrix} 1 & dt \\ 0 & 1 \end{pmatrix}$ (discrete integration)
- $H = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ (projection)

$$
\begin{aligned}
X_k^{\text{pred}} &= FX_{k-1}^{\text{est}} \\
X_k^{\text{est}} &= X_k^{\text{pred}} + K_k(Y_k - HX_k^{\text{pred}}) \\
S_k &= (R + HP_k^{\text{pred}}H^T)^{-1} \\
K_k &= P_k^{\text{pred}}H^T S_k \\
P_k^{\text{pred}} &= Q + FP_{k-1}^{\text{est}}F^T \\
P_k^{\text{est}} &= P_k^{\text{pred}} - K_k H P_k^{\text{pred}}
\end{aligned}
$$

Solution: Kalman filter

Demo

Semi-symbolic inference

Probabilistic Programming Languages

Semi-symbolic inference

Simple particles filters can be impractical

- Require lot of computing power
- Poor approximation

Exact inference is often possible

Rao-Blackwellized particle filters

- Each particle maintains symbolic distributions
- Perform as much exact computation as possible
- Fall back to a particle filter when symbolic computation fails

Main idea

- Keep track of conjugacy relationships
- Incorporate observations analytically
- Sample only when necessary

Semi-symbolic inference

Simple particles filters can be impractical

- Require lot of computing power
- Poor approximation

Exact inference is often possible

Rao-Blackwellized particle filters

- Each particle maintains symbolic distributions
- Perform as much exact computation as possible
- Fall back to a particle filter when symbolic computation fails

Main idea

- Keep track of conjugacy relationships
- Incorporate observations analytically
- Sample only when necessary

Example: Conjugate Gaussians

$$x \sim \mathcal{N}(\mu_0, \sigma_0^2)$$
$$y \sim \mathcal{N}(x, \sigma^2)$$

$$x \mid (y = v) \sim \mathcal{N}(\mu_1, \sigma_1^2)$$
$$\mu_1 = \left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2}\right)^{-1} \left(\frac{\mu_0}{\sigma_0^2} + \frac{v}{\sigma^2}\right)$$
$$\sigma_1^2 = \left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2}\right)^{-1}$$

Semi-symbolic inference

```
for y in data:
  pre_x = acc[-1]
  x = sample(gaussian(pre_x, 1))
  observe(gaussian(x, 1), y)
```

Semi-symbolic inference

```
for y in data:
  pre_x = acc[-1]
  x = sample(gaussian(pre_x, 1))
  observe(gaussian(x, 1), y)
```

$t = 0$

Semi-symbolic inference

```
for y in data:
  pre_x = acc[-1]
  x = sample(gaussian(pre_x, 1))
  observe(gaussian(x, 1), y)
```

$t = 0$

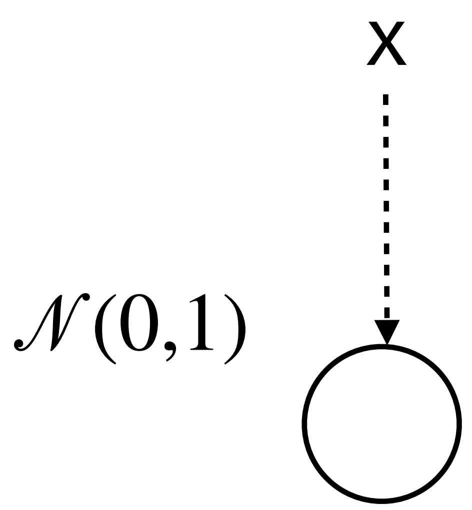
```
sample (gaussian (0, 1))
```

Semi-symbolic inference

```
for y in data:
  pre_x = acc[-1]
  x = sample(gaussian(pre_x, 1))
  observe(gaussian(x, 1), y)
```

$t = 0$

```
sample (gaussian (0, 1))
```

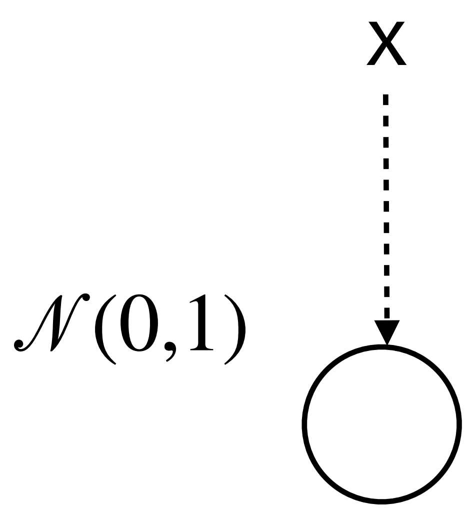


Semi-symbolic inference

```
for y in data:
  pre_x = acc[-1]
  x = sample(gaussian(pre_x, 1))
  observe(gaussian(x, 1), y)
```

$t = 0$

```
sample (gaussian (0, 1))
observe (gaussian (x, 1), 3)
```

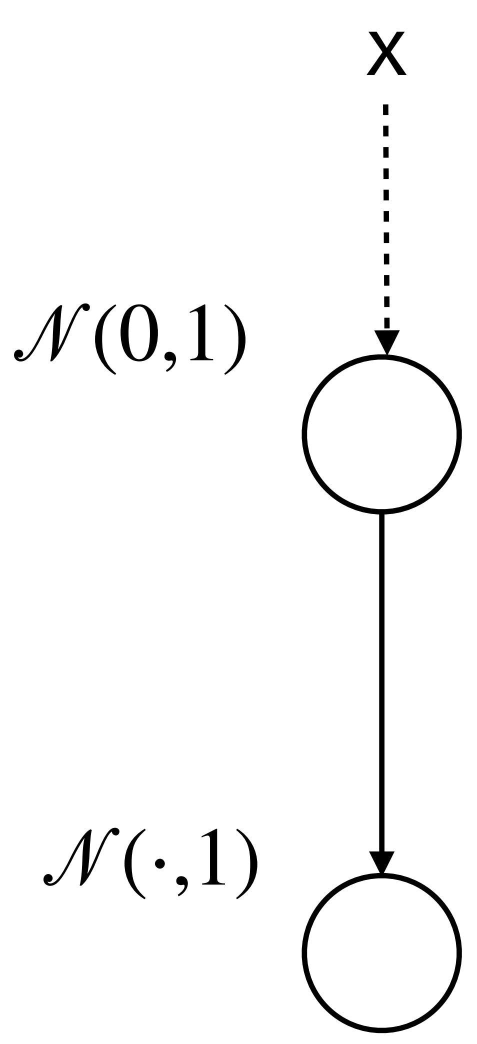


Semi-symbolic inference

```
for y in data:
  pre_x = acc[-1]
  x = sample(gaussian(pre_x, 1))
  observe(gaussian(x, 1), y)
```

$t = 0$

```
sample (gaussian (0, 1))
observe (gaussian (x, 1), 3)
```

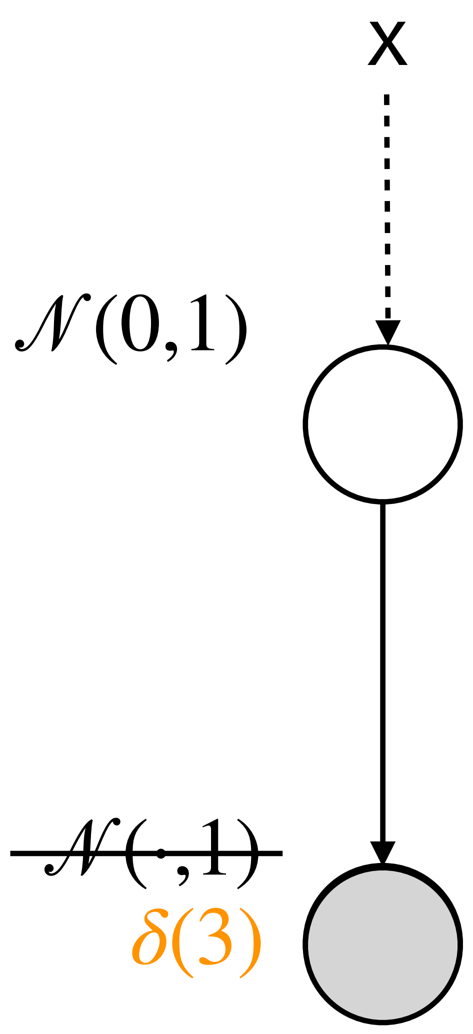


Semi-symbolic inference

```
for y in data:
  pre_x = acc[-1]
  x = sample(gaussian(pre_x, 1))
  observe(gaussian(x, 1), y)
```

$t = 0$

```
sample (gaussian (0, 1))
observe (gaussian (x, 1), 3)
```

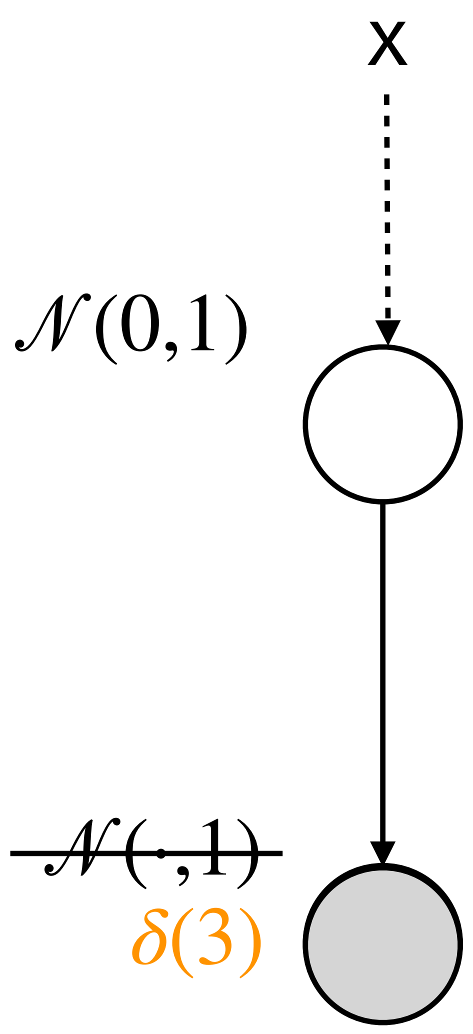


Semi-symbolic inference

```
for y in data:
  pre_x = acc[-1]
  x = sample(gaussian(pre_x, 1))
  observe(gaussian(x, 1), y)
```

$t = 0$

```
sample (gaussian (0, 1))
observe (gaussian (x, 1), 3)
```



Example: Conjugate Gaussians

$$x \sim \mathcal{N}(\mu_0, \sigma_0^2)$$
$$y \sim \mathcal{N}(x, \sigma^2)$$

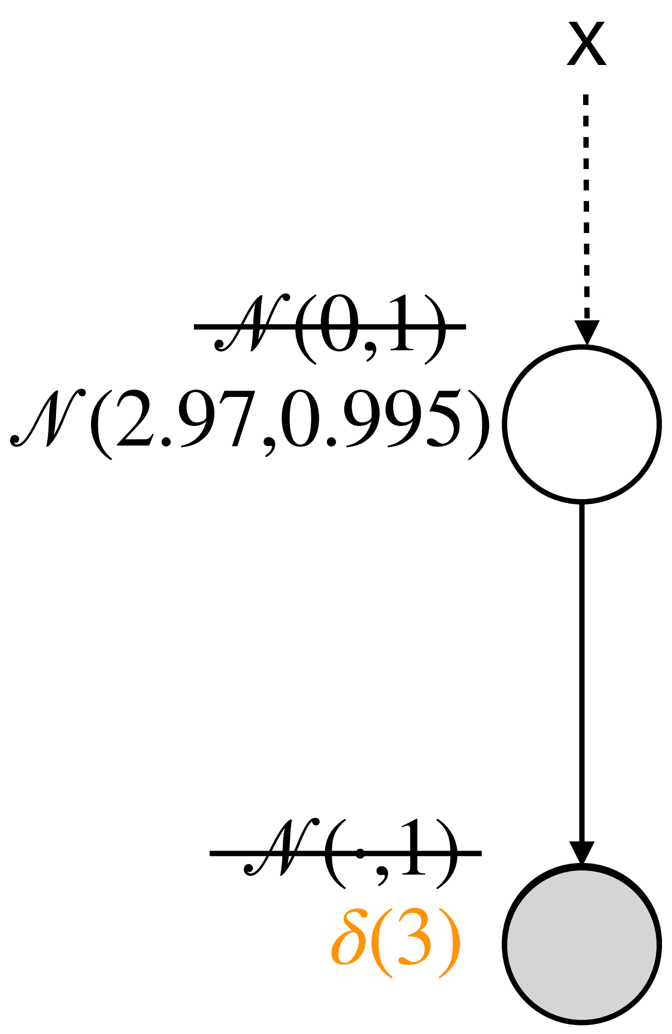
$$x \mid (y = v) \sim \mathcal{N}(\mu_1, \sigma_1^2)$$
$$\mu_1 = \left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2}\right)^{-1} \left(\frac{\mu_0}{\sigma_0^2} + \frac{v}{\sigma^2}\right)$$
$$\sigma_1^2 = \left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2}\right)^{-1}$$

Semi-symbolic inference

```
for y in data:
  pre_x = acc[-1]
  x = sample(gaussian(pre_x, 1))
  observe(gaussian(x, 1), y)
```

$t = 0$

```
sample (gaussian (0, 1))
observe (gaussian (x, 1), 3)
```



Example: Conjugate Gaussians

$$x \sim \mathcal{N}(\mu_0, \sigma_0^2)$$
$$y \sim \mathcal{N}(x, \sigma^2)$$

$$x \mid (y = v) \sim \mathcal{N}(\mu_1, \sigma_1^2)$$

$$\mu_1 = \left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2}\right)^{-1} \left(\frac{\mu_0}{\sigma_0^2} + \frac{v}{\sigma^2}\right)$$

$$\sigma_1^2 = \left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2}\right)^{-1}$$

Semi-symbolic inference

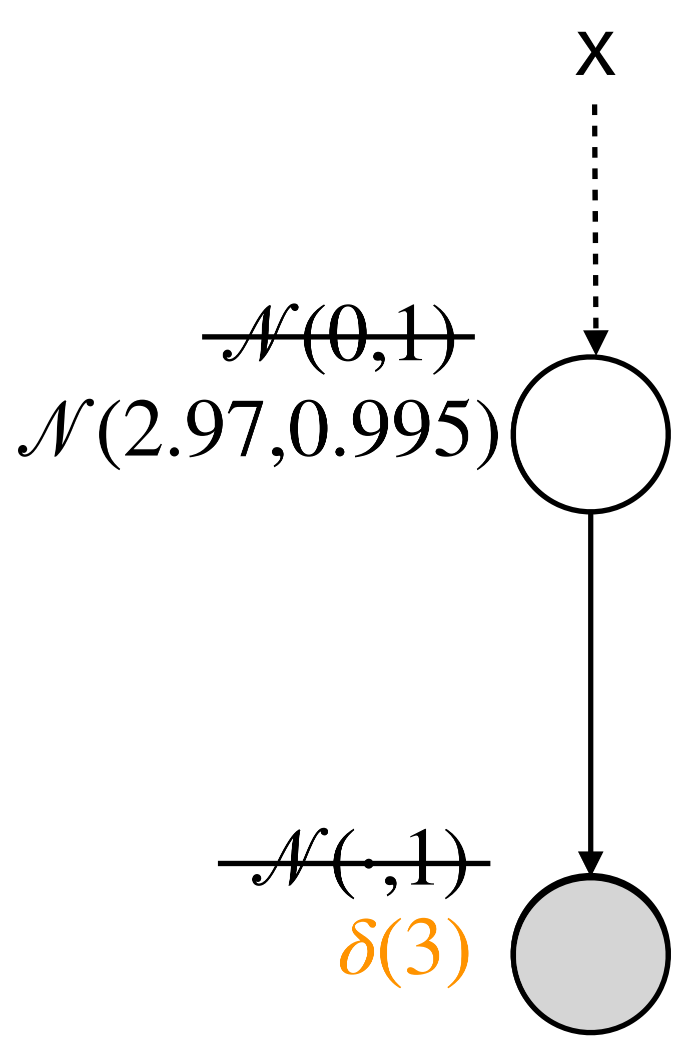
```
for y in data:
  pre_x = acc[-1]
  x = sample(gaussian(pre_x, 1))
  observe(gaussian(x, 1), y)
```

$t = 0$

```
sample (gaussian (0, 1))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
sample (gaussian (pre x, 1))
```



Semi-symbolic inference

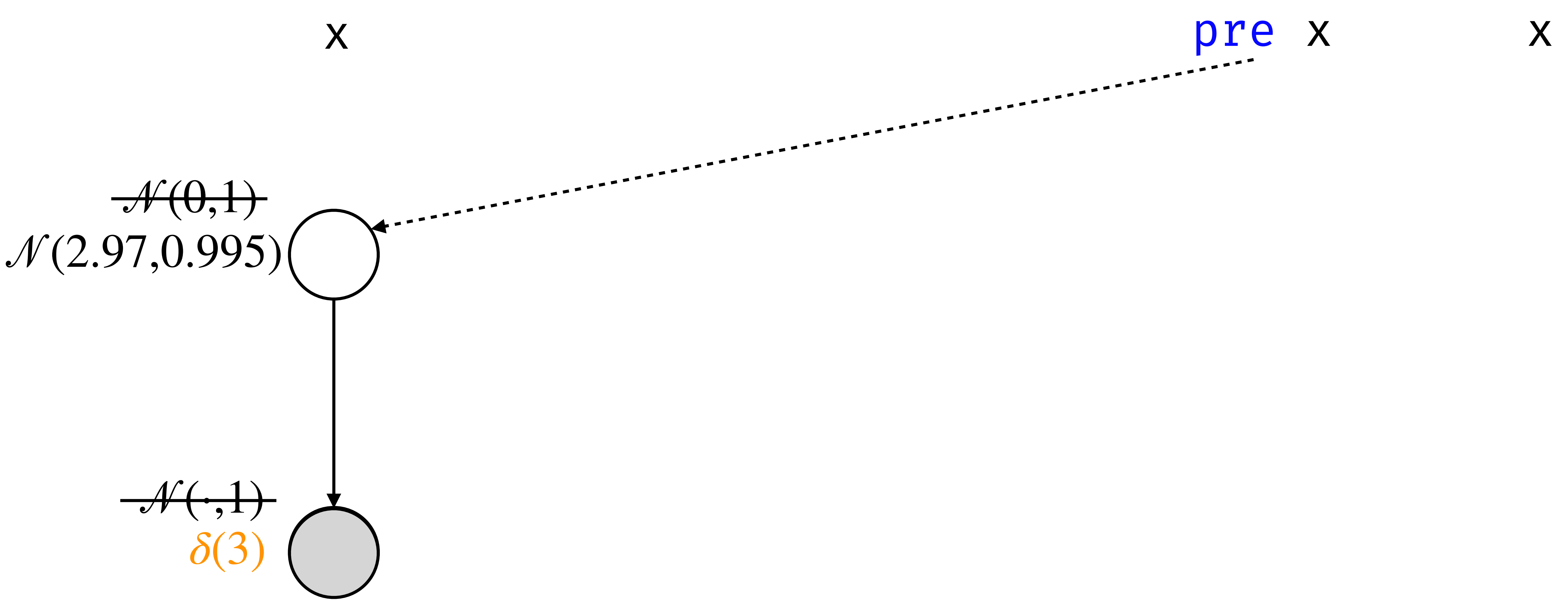
```
for y in data:
  pre_x = acc[-1]
  x = sample(gaussian(pre_x, 1))
  observe(gaussian(x, 1), y)
```

$t = 0$

```
sample (gaussian (0, 1))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
sample (gaussian (pre x, 1))
```



Semi-symbolic inference

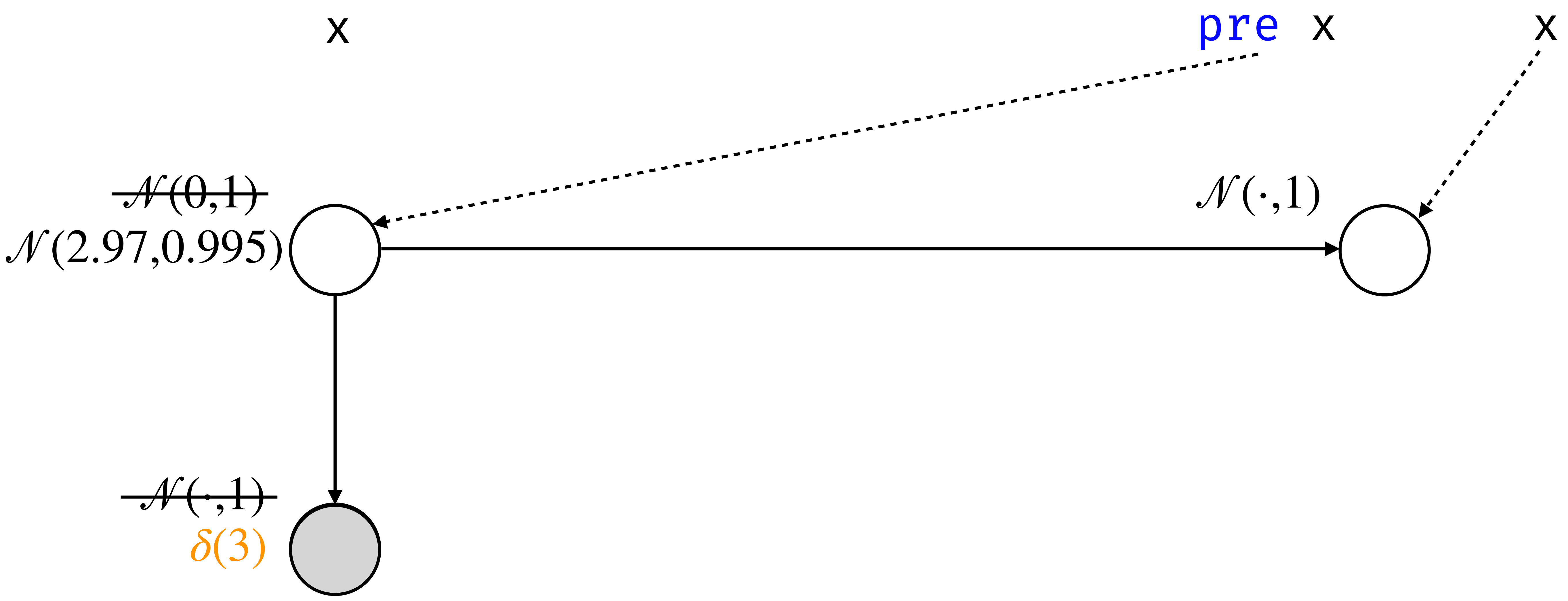
```
for y in data:
  pre_x = acc[-1]
  x = sample(gaussian(pre_x, 1))
  observe(gaussian(x, 1), y)
```

$t = 0$

```
sample (gaussian (0, 1))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
sample (gaussian (pre x, 1))
```



Semi-symbolic inference

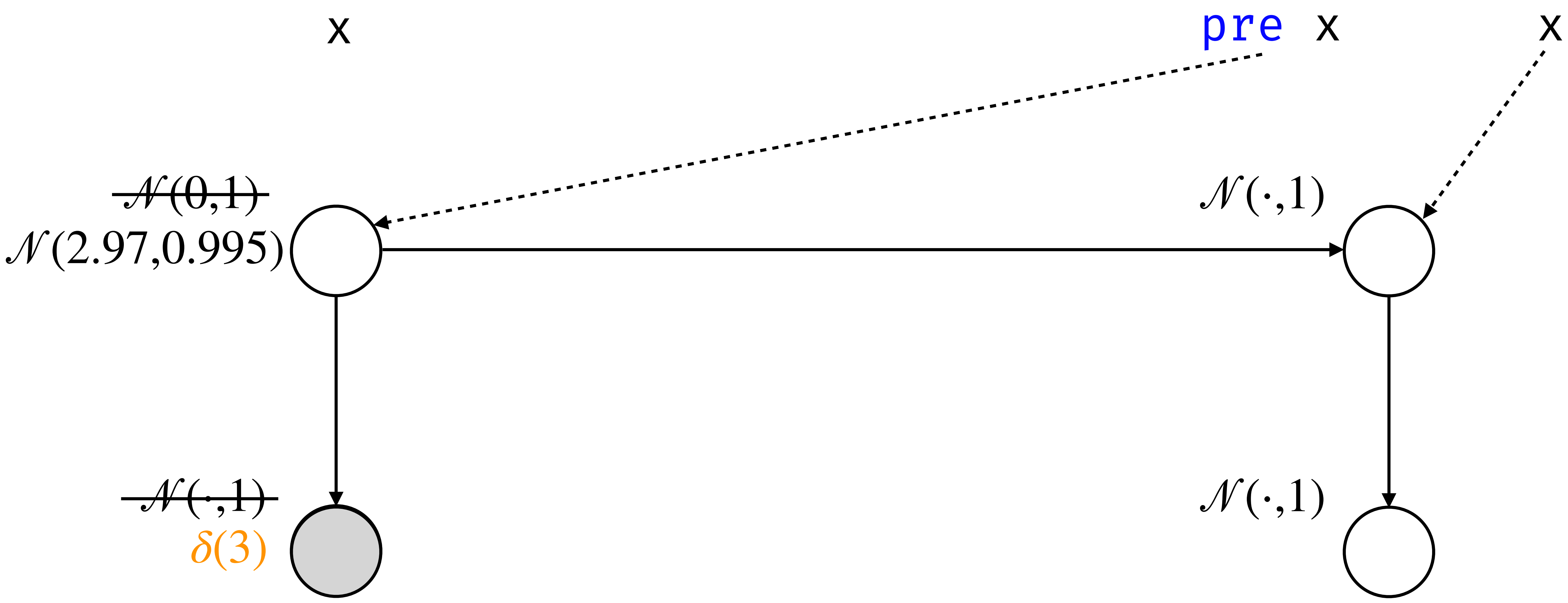
```
for y in data:
  pre_x = acc[-1]
  x = sample(gaussian(pre_x, 1))
  observe(gaussian(x, 1), y)
```

$t = 0$

```
sample (gaussian (0, 1))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), 5)
```



Semi-symbolic inference

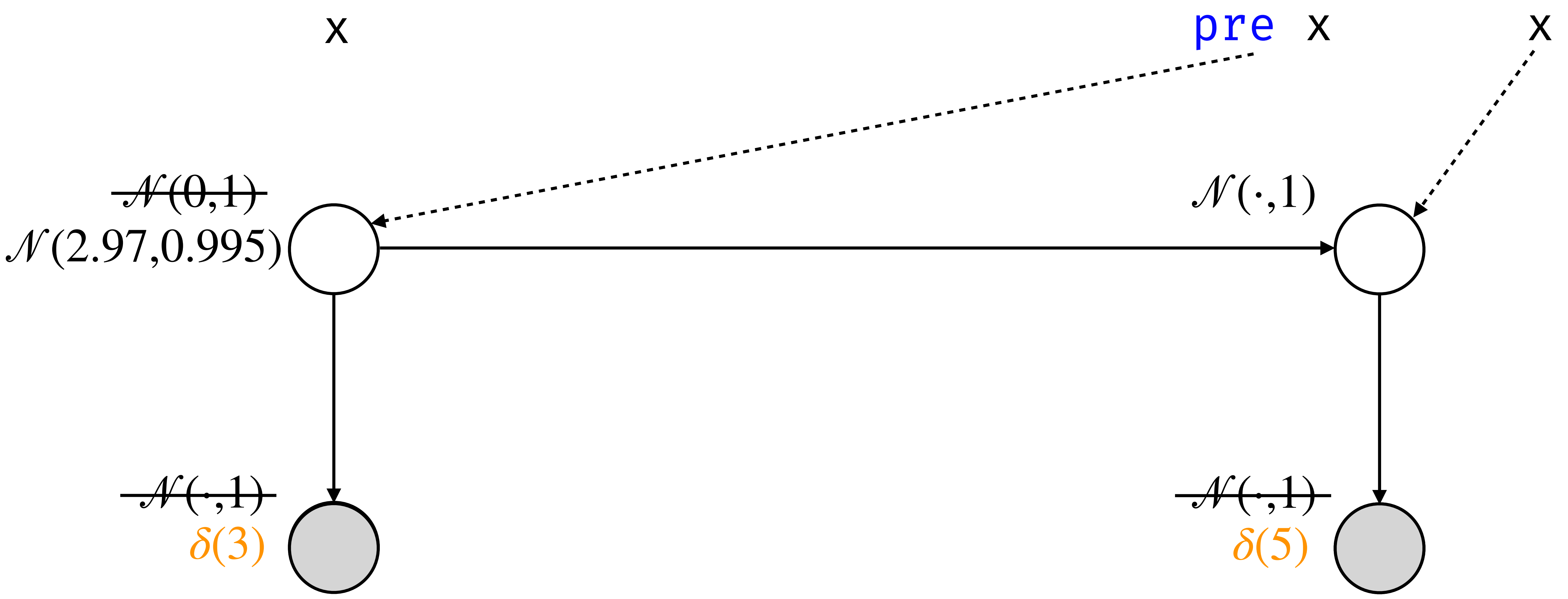
```
for y in data:
  pre_x = acc[-1]
  x = sample(gaussian(pre_x, 1))
  observe(gaussian(x, 1), y)
```

$t = 0$

```
sample (gaussian (0, 1))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), 5)
```



Semi-symbolic inference

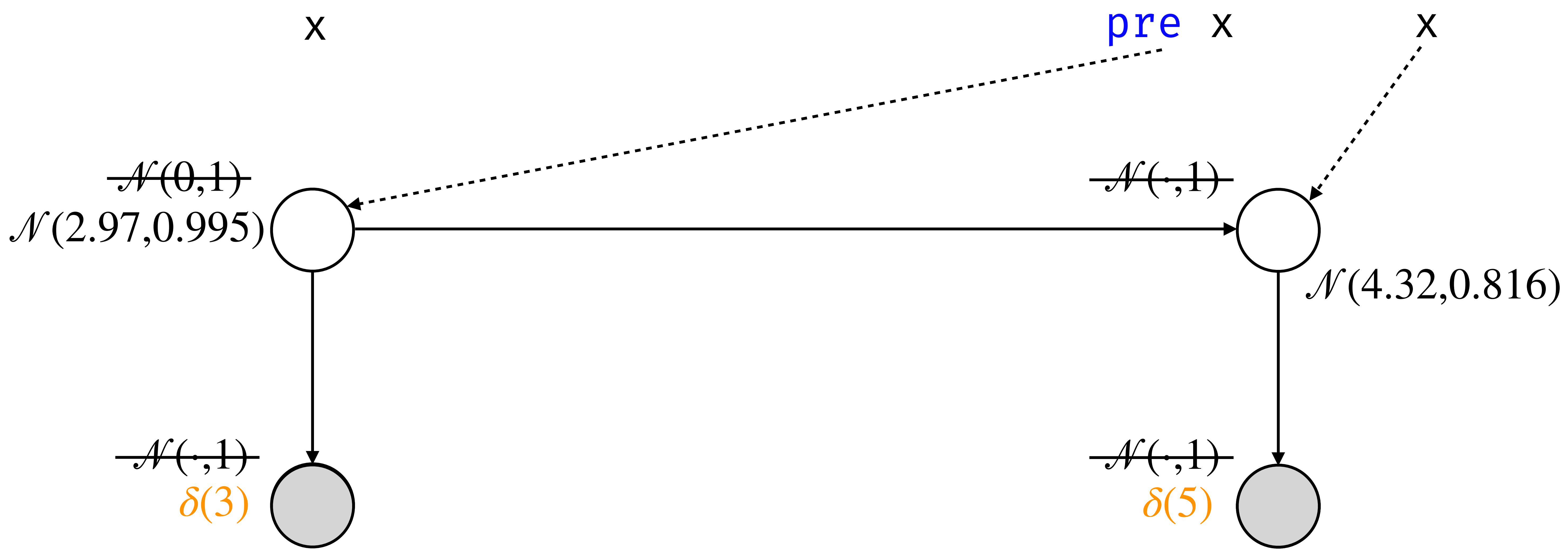
```
for y in data:
  pre_x = acc[-1]
  x = sample(gaussian(pre_x, 1))
  observe(gaussian(x, 1), y)
```

$t = 0$

```
sample (gaussian (0, 1))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), 5)
```



Semi-symbolic inference

```
for y in data:
  pre_x = acc[-1]
  x = sample(gaussian(pre_x, 1))
  observe(gaussian(x, 1), y)
```

$t = 0$

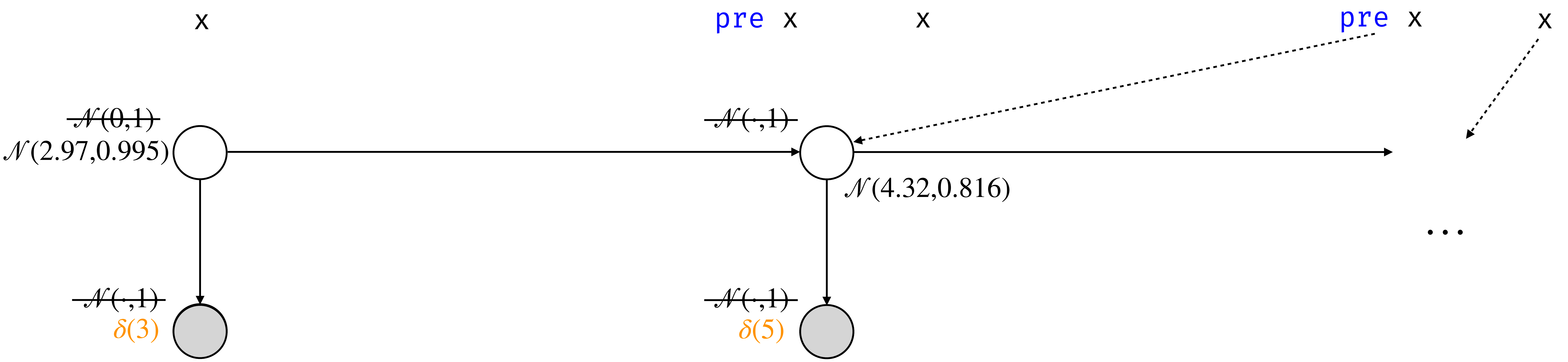
```
sample (gaussian (0, 1))
observe (gaussian (x, 1), 3)
```

$t = 1$

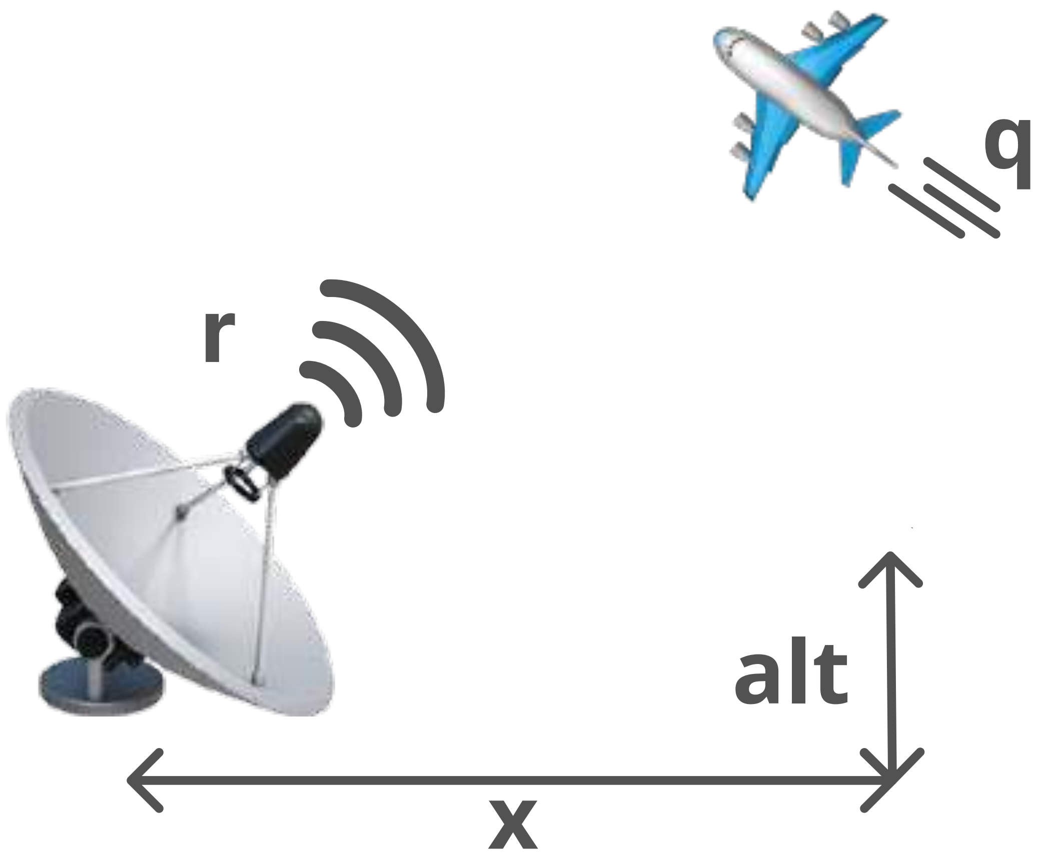
```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), 5)
```

$t = 2$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), …)
```

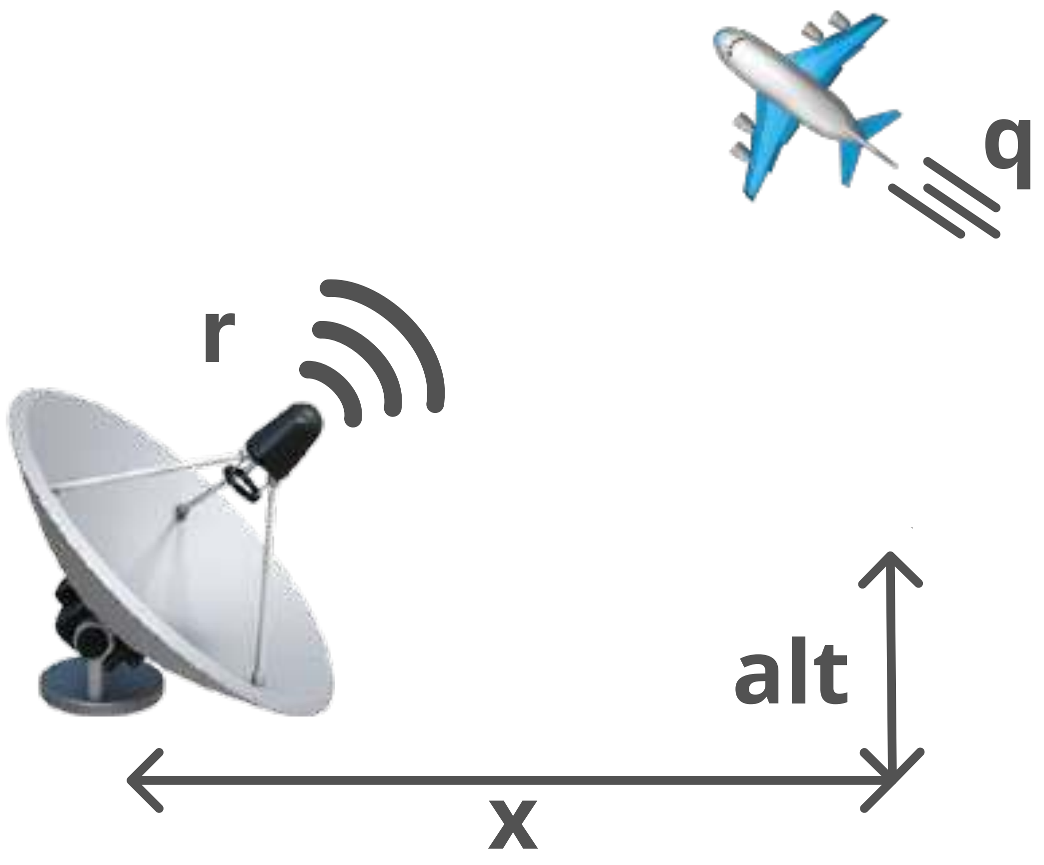


Example: tracker++



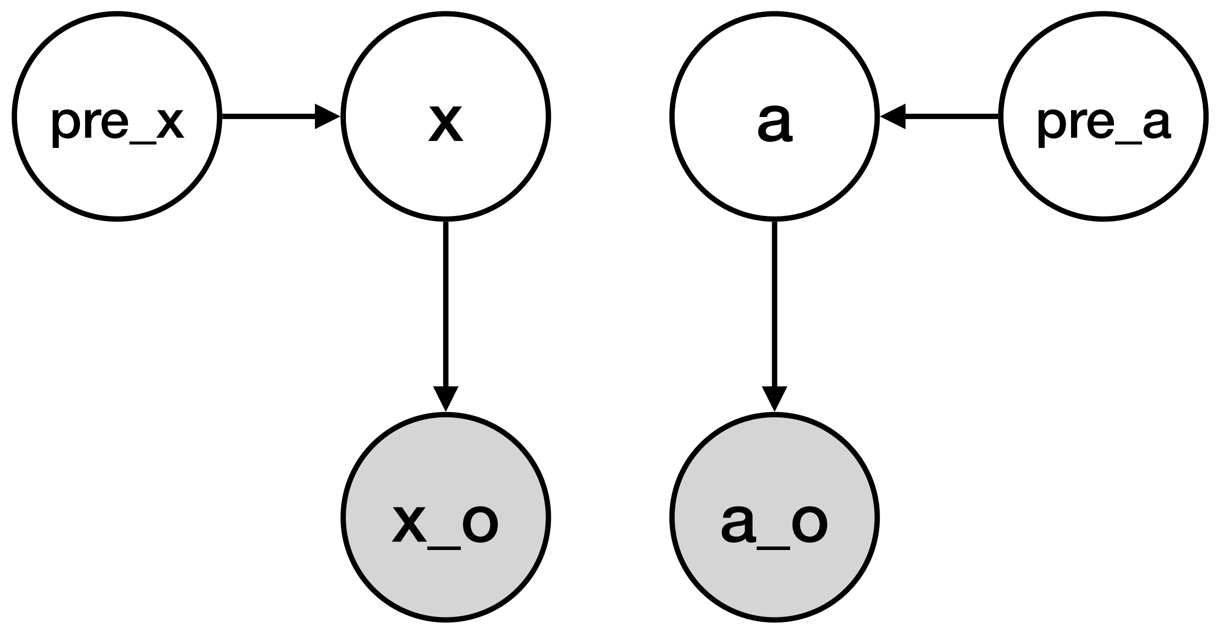

```
def tracker(data: list[X * A]) → list[X * A]:

  # q and r are constants

  acc = [(0,0)]
  for (xo, ao) in data:                  # at each time step
    pre_x, pre_a = acc[-1]
    x = sample(Gaussian(pre_x, q))       # random walk
    a = sample(Gaussian(pre_a, q))


    observe(Gaussian(x, r), xo)          # condition on observations
    observe(Gaussian(a, r), ao)
    acc.append((x, a))

  return acc
```

Example: tracker++

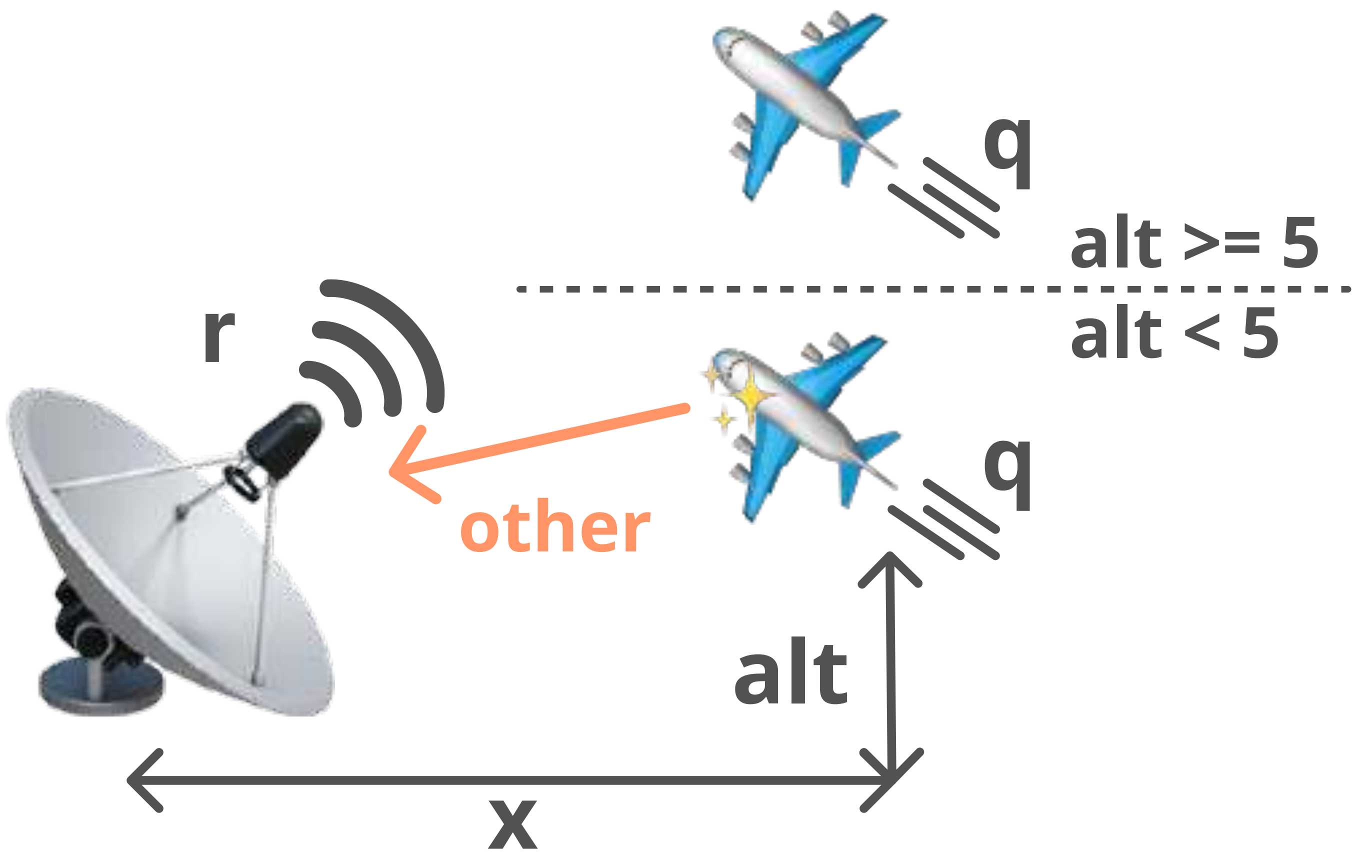



```
def tracker(data: list[X * A]) → list[X * A]:

  # q and r are constants

  acc = [(0,0)]
  for (xo, ao) in data:                # at each time step
    pre_x, pre_a = acc[-1]
    x = sample(Gaussian(pre_x, q))     # random walk
    a = sample(Gaussian(pre_a, q))


    observe(Gaussian(x, r), xo)        # condition on obse
    observe(Gaussian(a, r), ao)
    acc.append((x, a))

  return acc
```

Exact solution



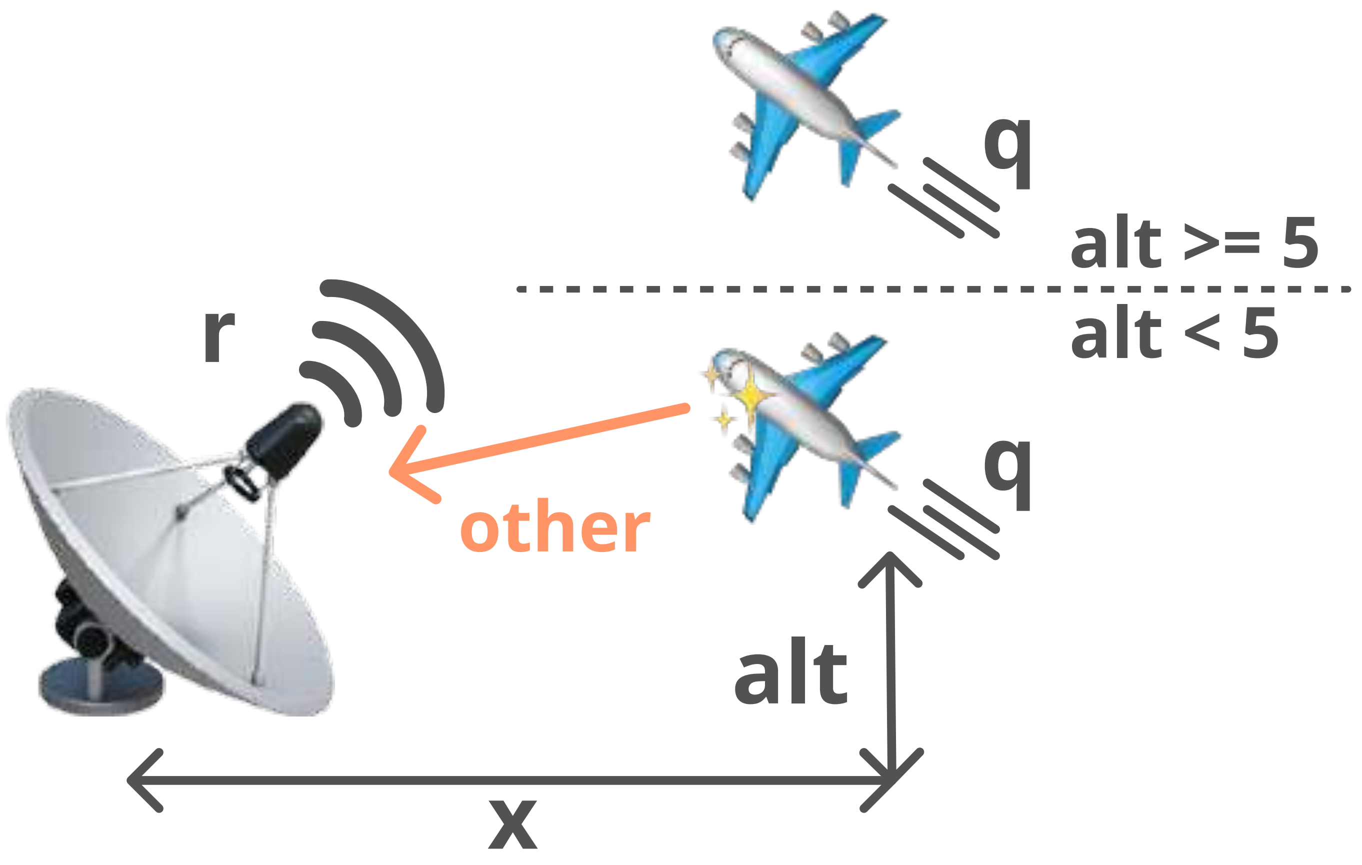
Example: tracker++




```
def tracker(data: list[X * A]) → list[X * A]:

  # q and v are constants

  acc = [(0,0)]
  for (xo, ao) in data:                  # at each time step
    pre_x, pre_a = acc[-1]
    x = sample(Gaussian(pre_x, q))       # random walk
    a = sample(Gaussian(pre_a, q))
    other = sample(InvGamma(1., 1.)) # spkiking noise at low alt
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)          # condition on observations
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc
```

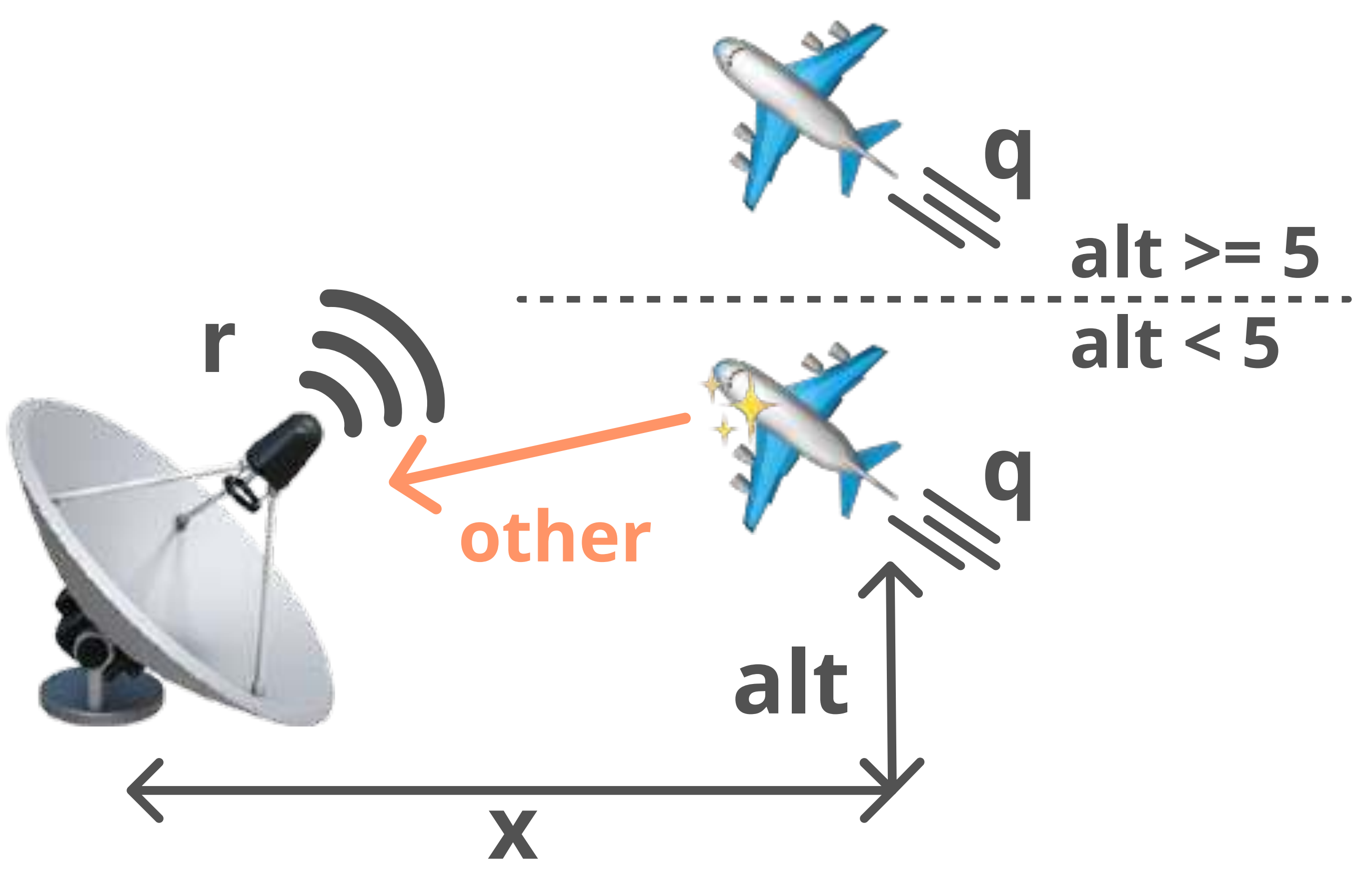
Example: tracker++

```
def tracker(data: list[X * A]) → list[X * A] * float * float:
  q = sample(InvGamma(1., 1.)) # move noise
  r = sample(InvGamma(1., 1.)) # observation noise

  acc = [(0,0)]
  for (xo, ao) in data:                  # at each time step
    pre_x, pre_a = acc[-1]
    x = sample(Gaussian(pre_x, q))       # random walk
    a = sample(Gaussian(pre_a, q))
    other = sample(InvGamma(1., 1.)) # spkiking noise at low alt
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)          # condition on observations
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

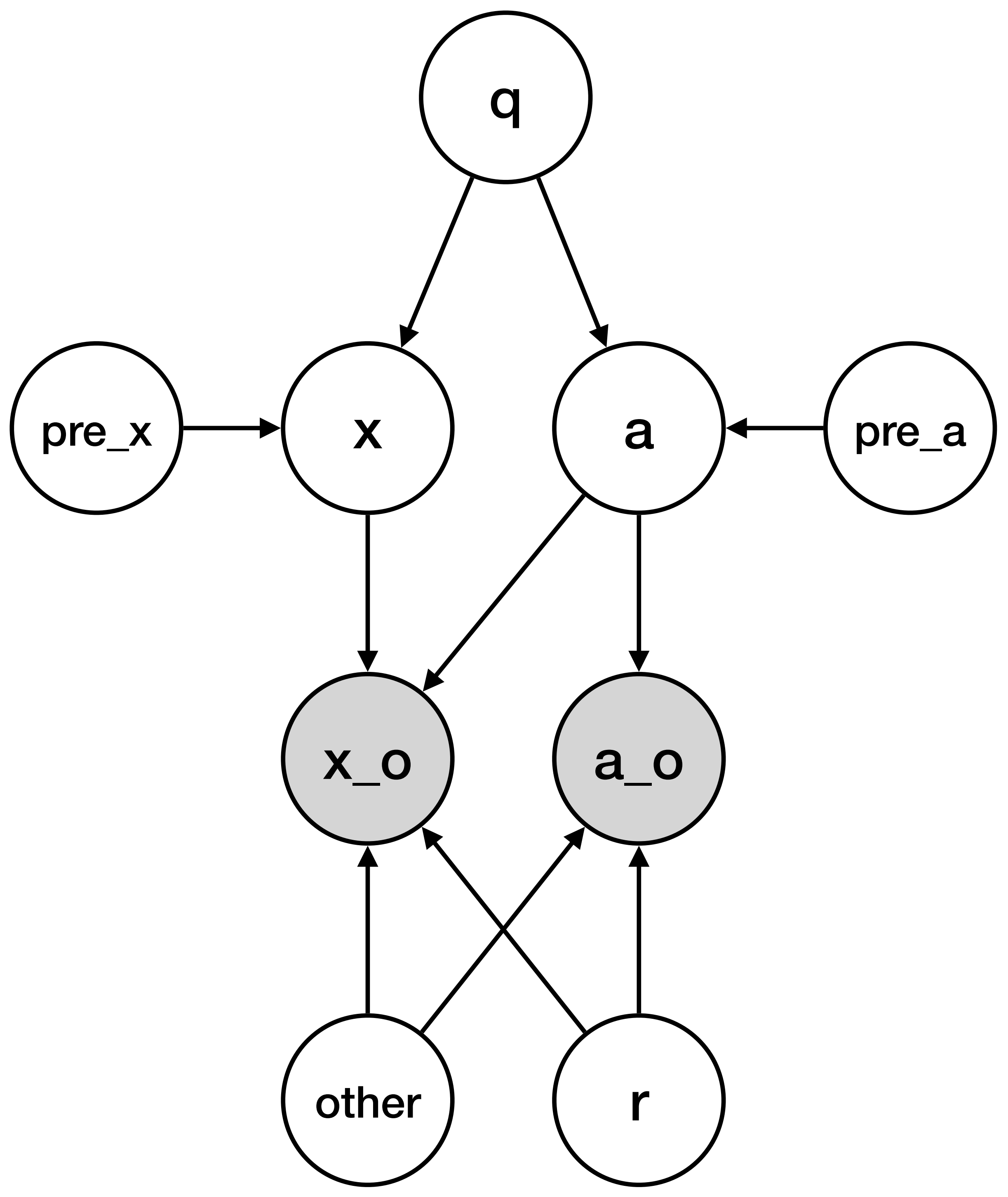


Example: tracker++

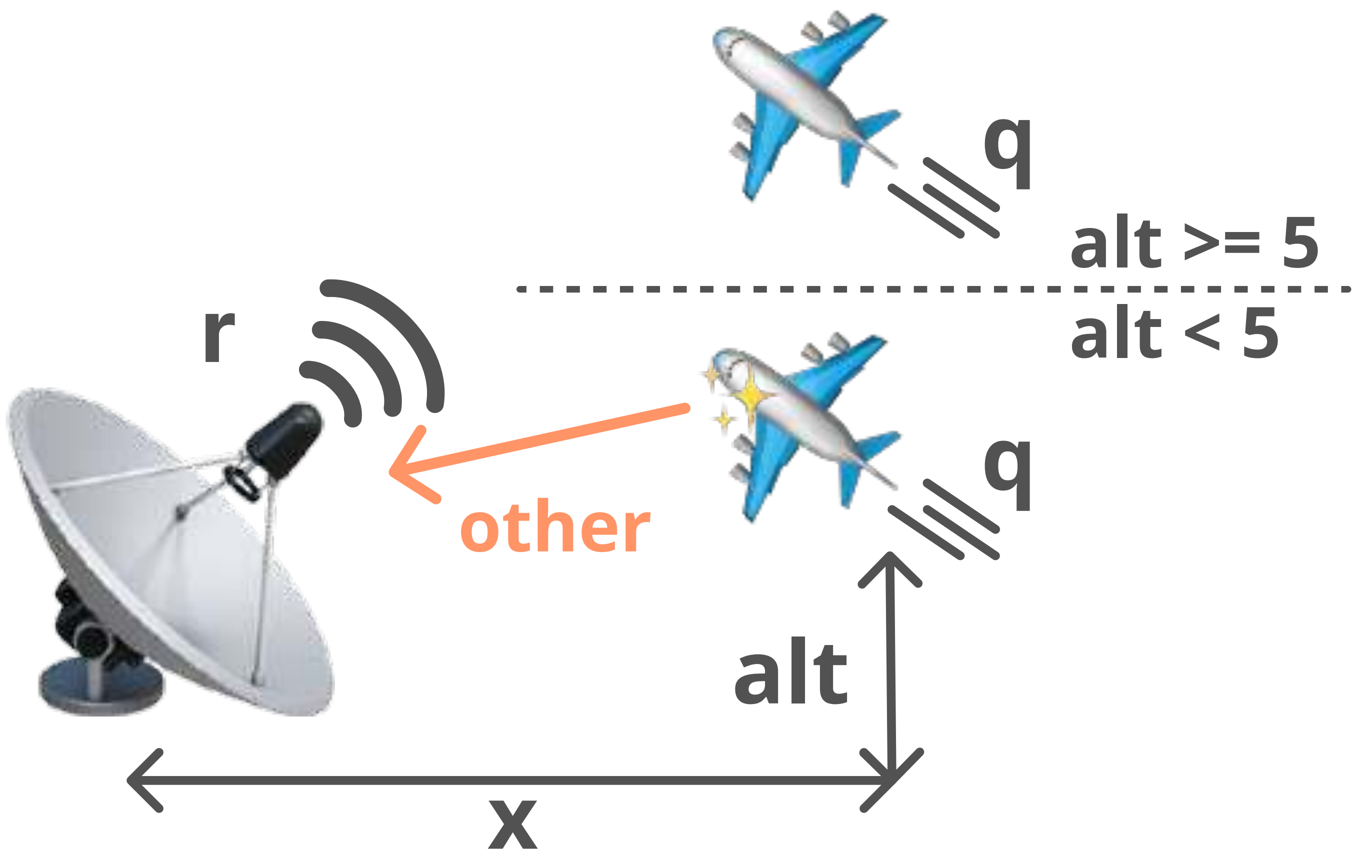
```
def tracker(data: list[X * A]) → list[X * A] * float * float:
  q = sample(InvGamma(1., 1.)) # move noise
  r = sample(InvGamma(1., 1.)) # observation noise

  acc = [(0,0)]
  for (xo, ao) in data:                    # at each time step
    pre_x, pre_a = acc[-1]
    x = sample(Gaussian(pre_x, q))         # random walk
    a = sample(Gaussian(pre_a, q))
    other = sample(InvGamma(1., 1.)) # spkiking noise at
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)            # condition on obse
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```



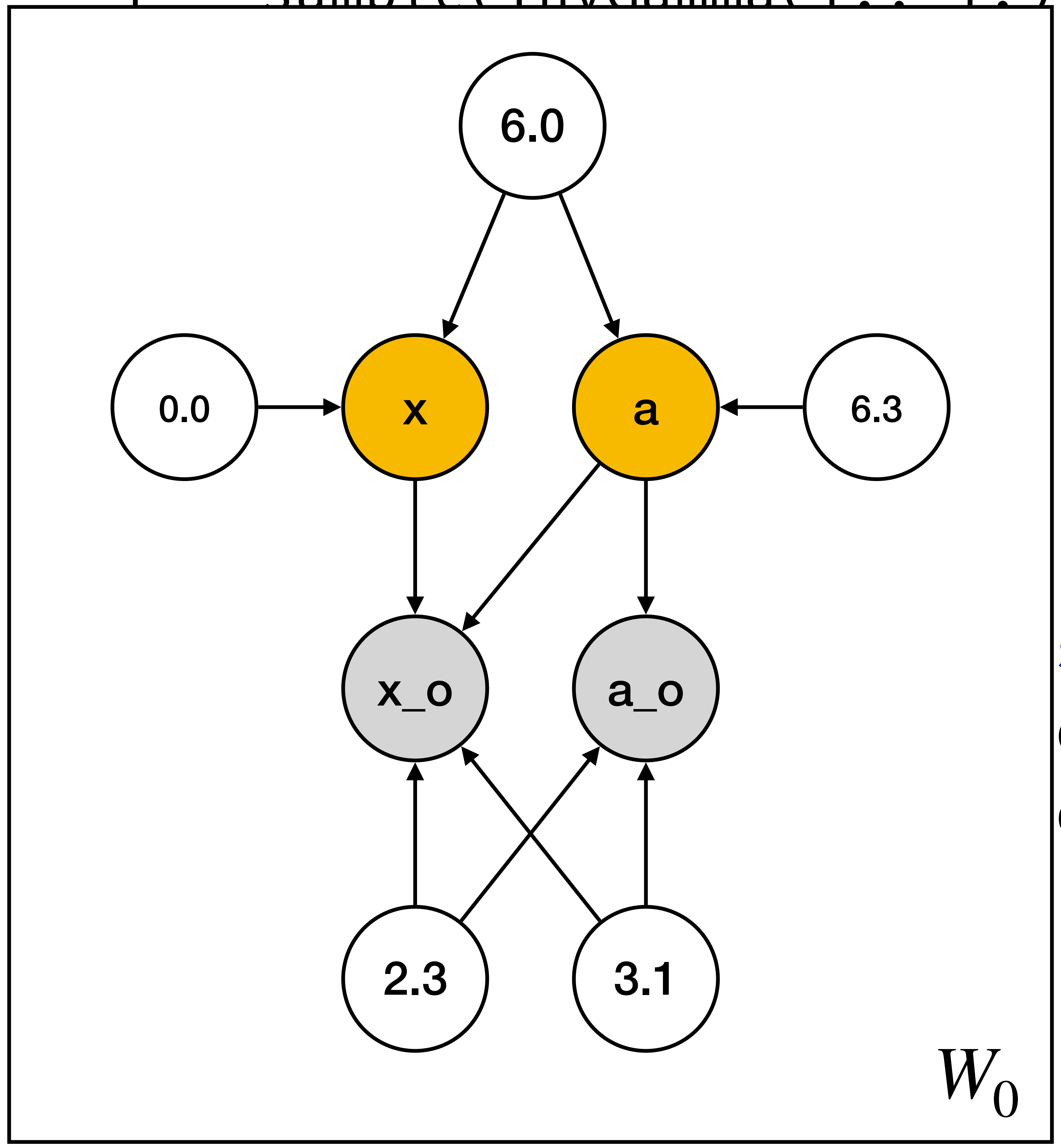
No analytical solution



Semi-symbolic inference

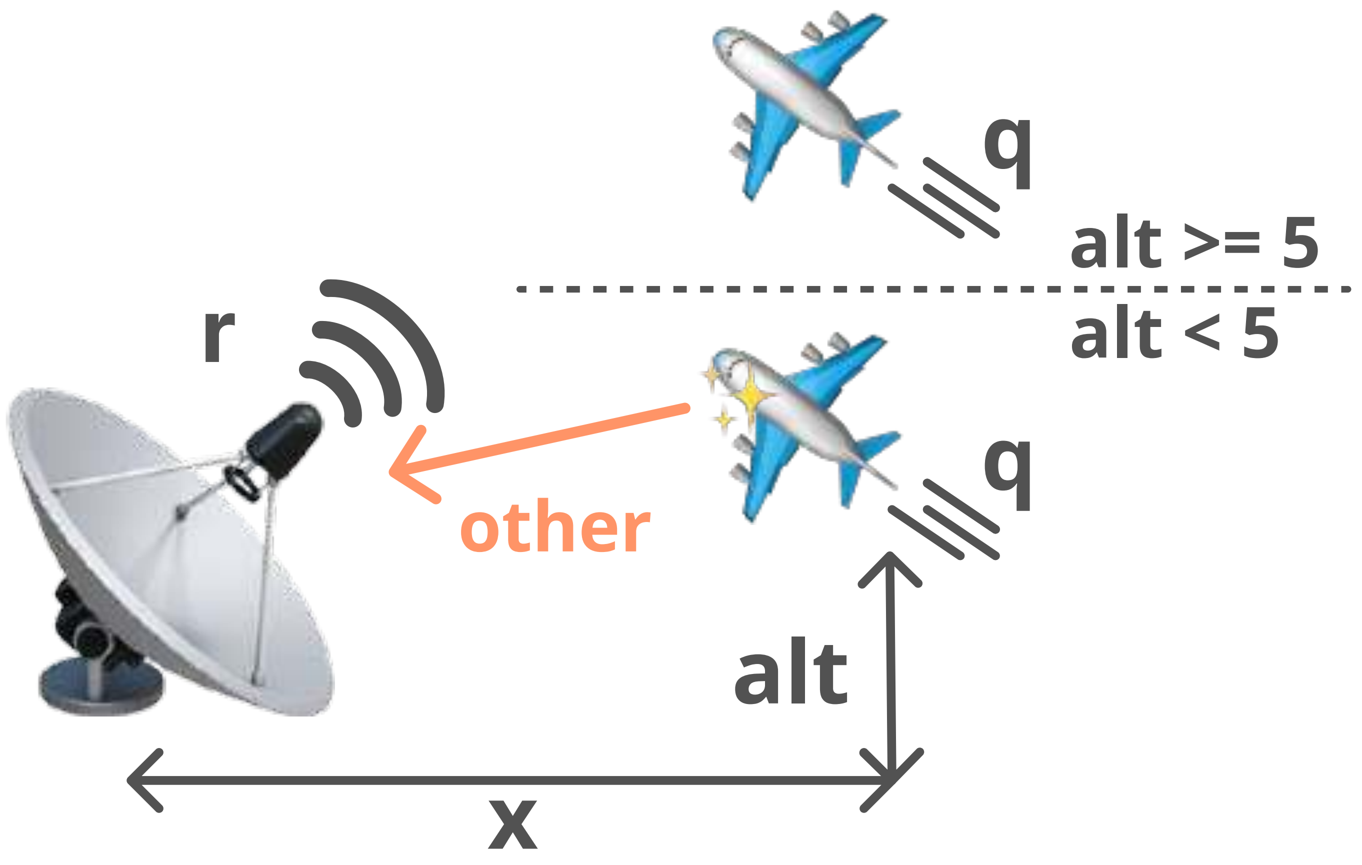



```
def tracker(data: list[X * A]) → list[X * A] * float * float:
  q = sample(InvGamma(1., 1.)) # move noise
  r = sample(InvGamma(1., 1.)) # observation noise
```

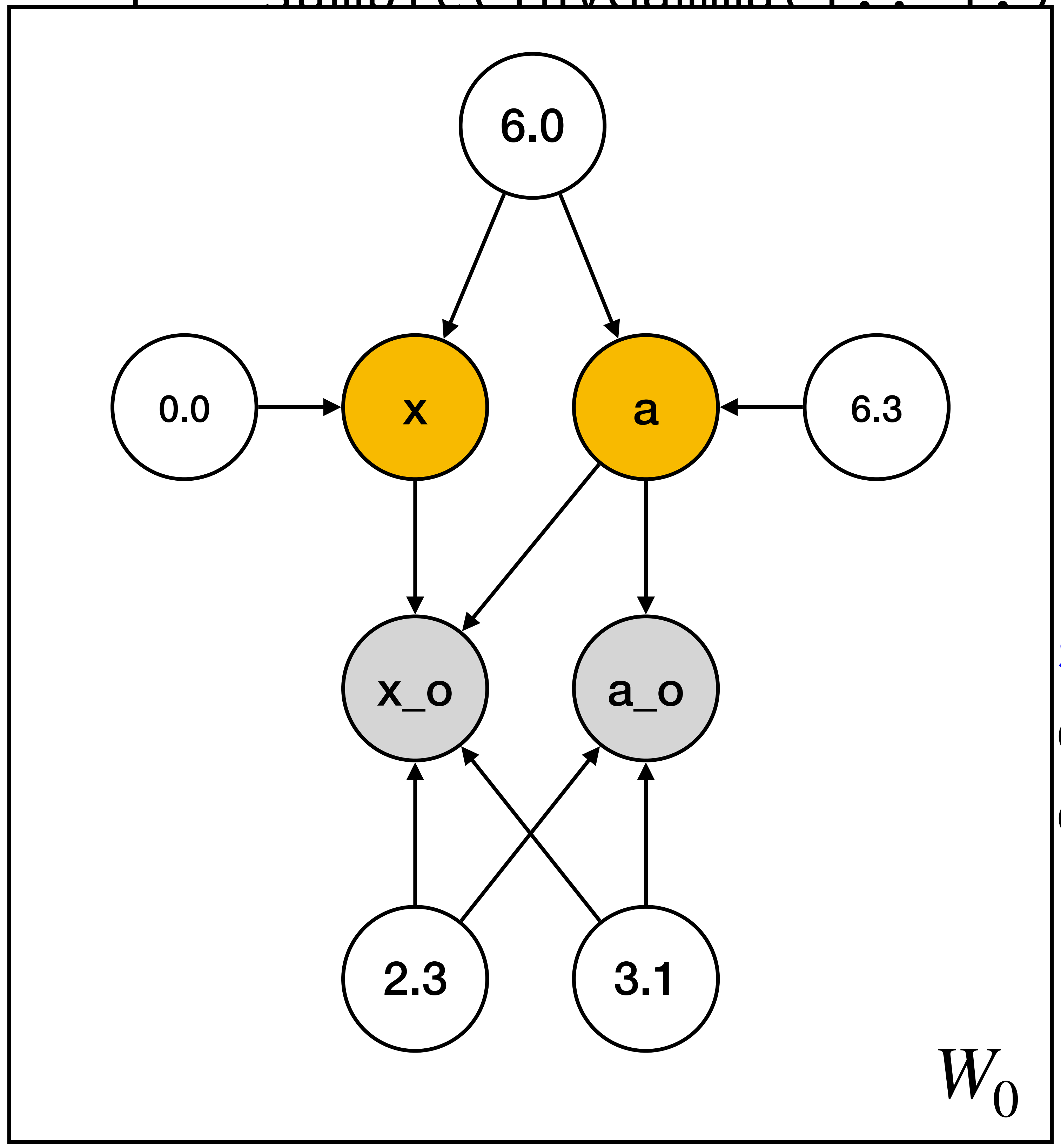


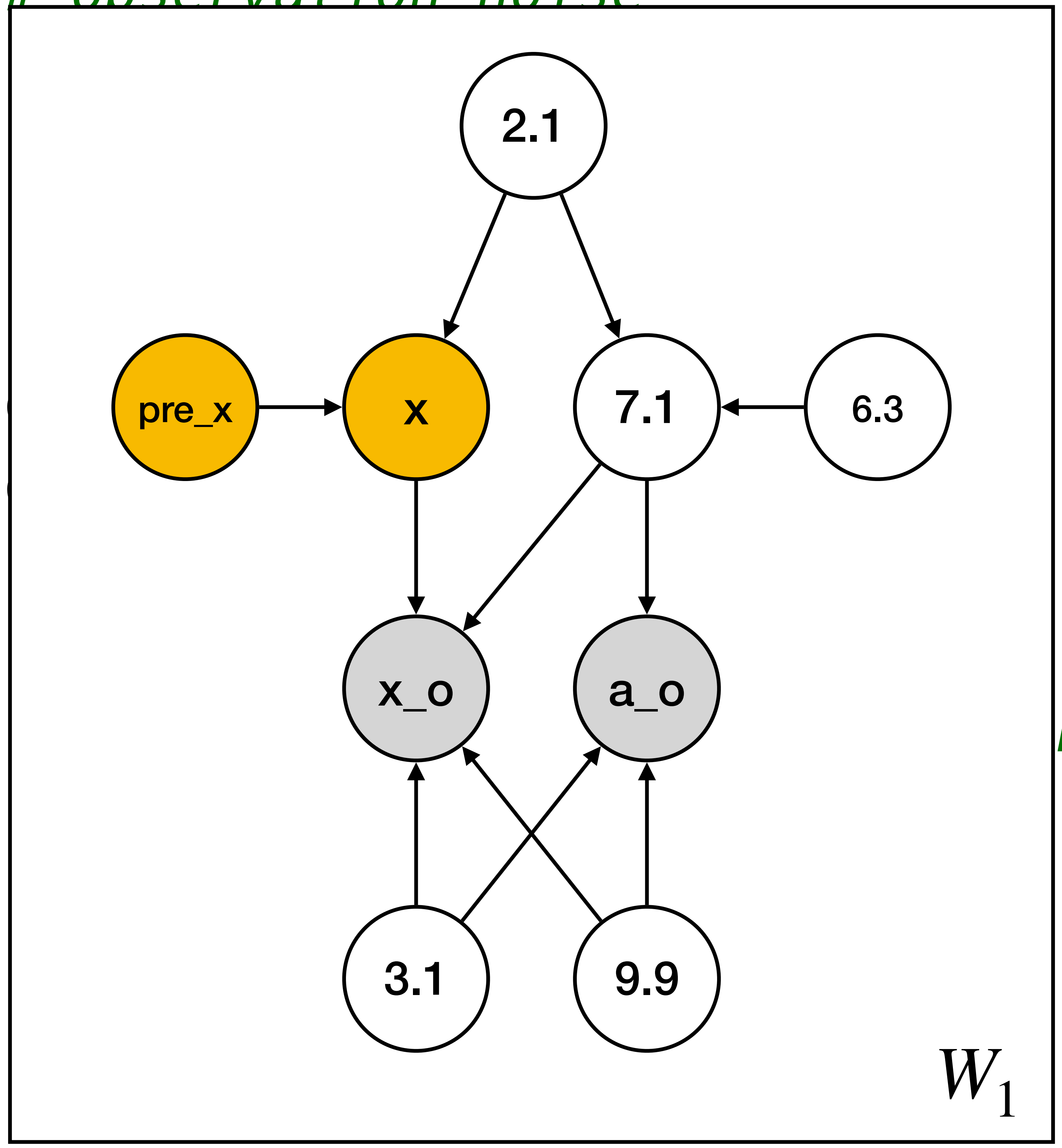

```
                   # at each time step

, q))    # random walk
, q))
., 1.)) # spkiking noise at low alt
se r
o)       # condition on observations
o)
```

Semi-symbolic inference



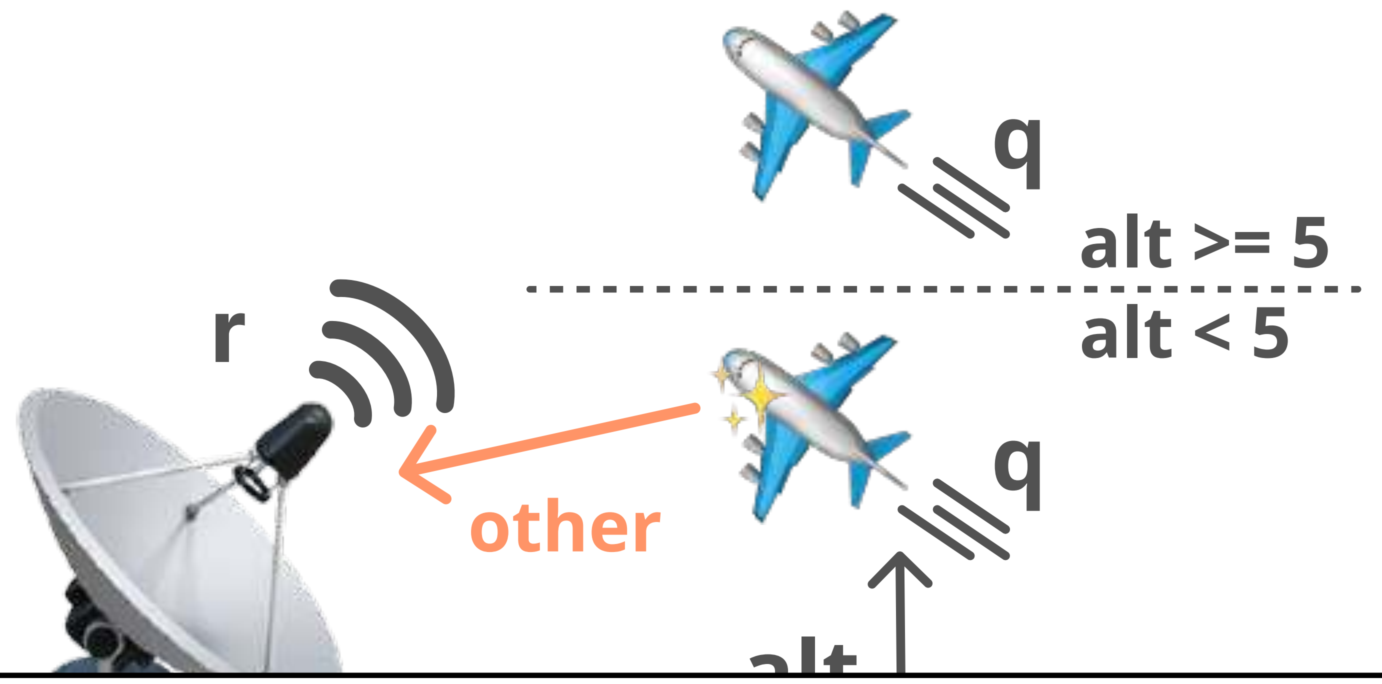

```
def tracker(data: list[X * A]) → list[X * A] * float * float:
  q = sample(InvGamma(1., 1.)) # move noise
  r = sample(InvGamma(1., 1.)) # observation noise
```

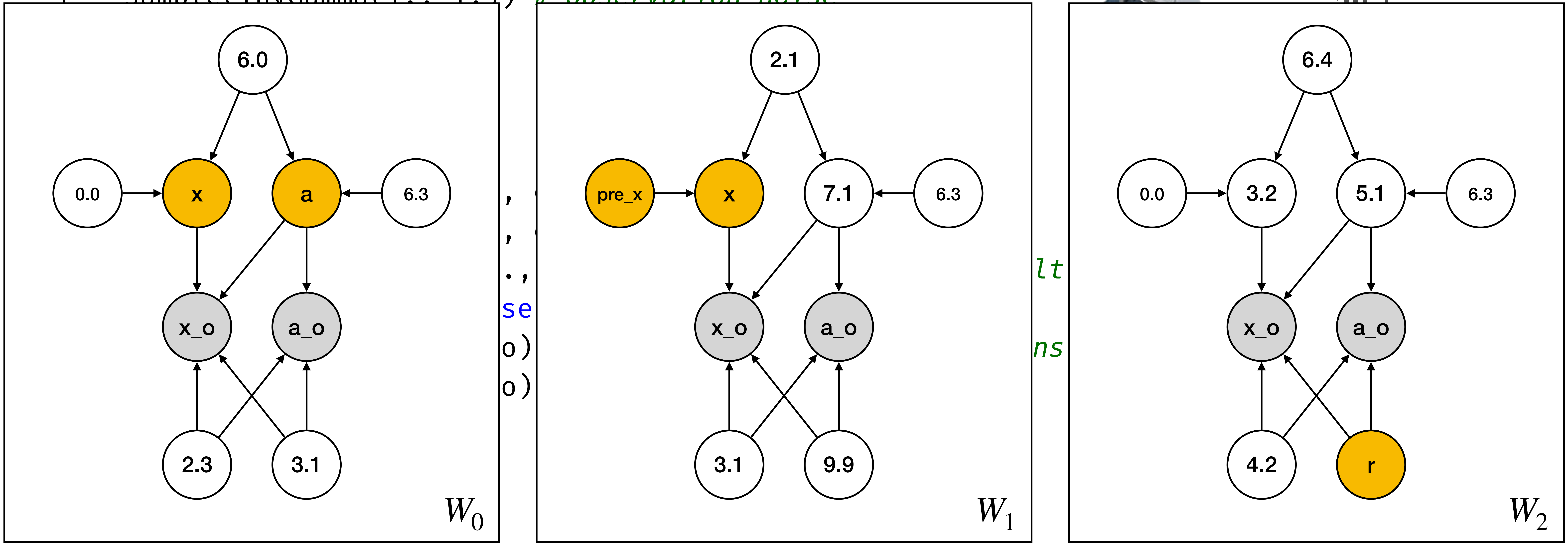




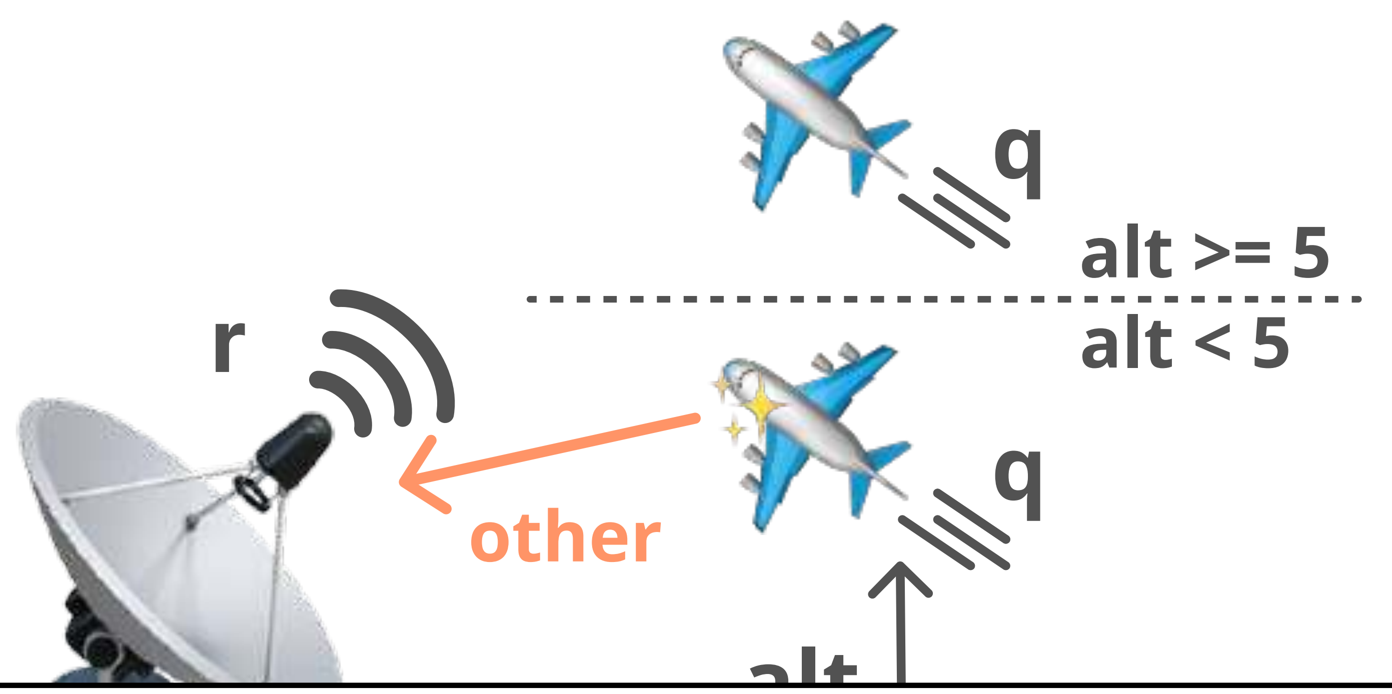
Semi-symbolic inference

```
def tracker(data: list[X * A]) → list[X * A] * float * float:
  q = sample(InvGamma(1., 1.)) # move noise
  r = sample(InvGamma(1., 1.)) # observation noise
```

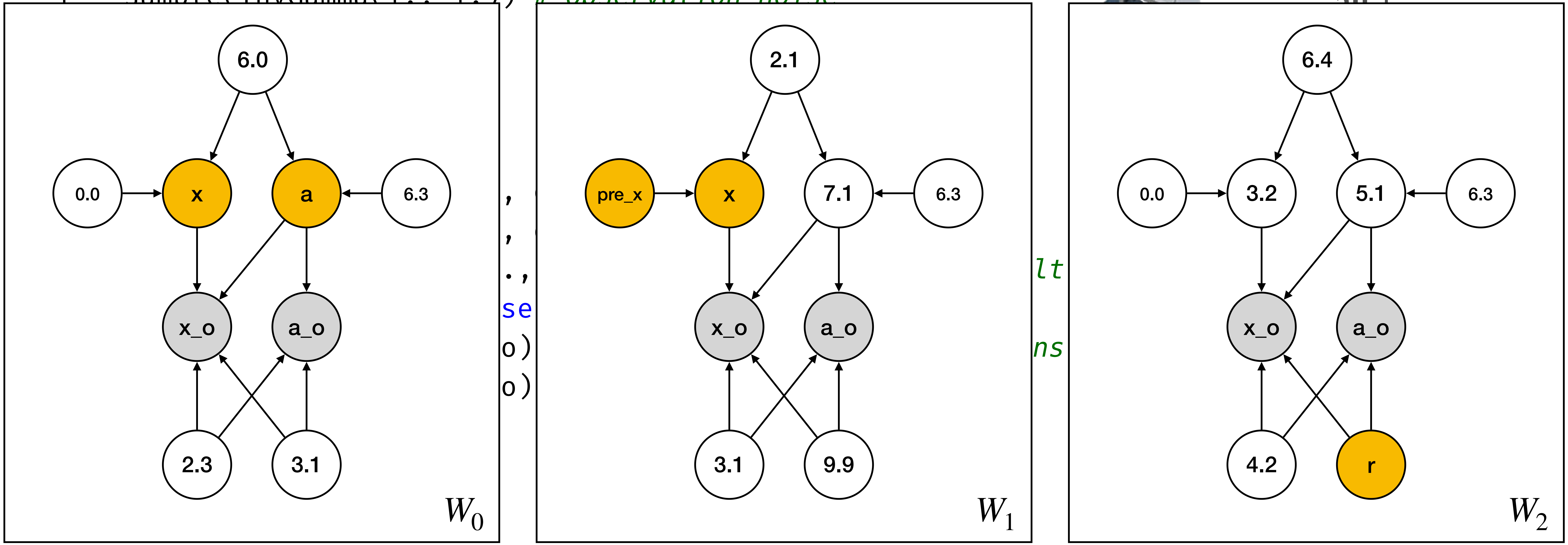




Semi-symbolic inference

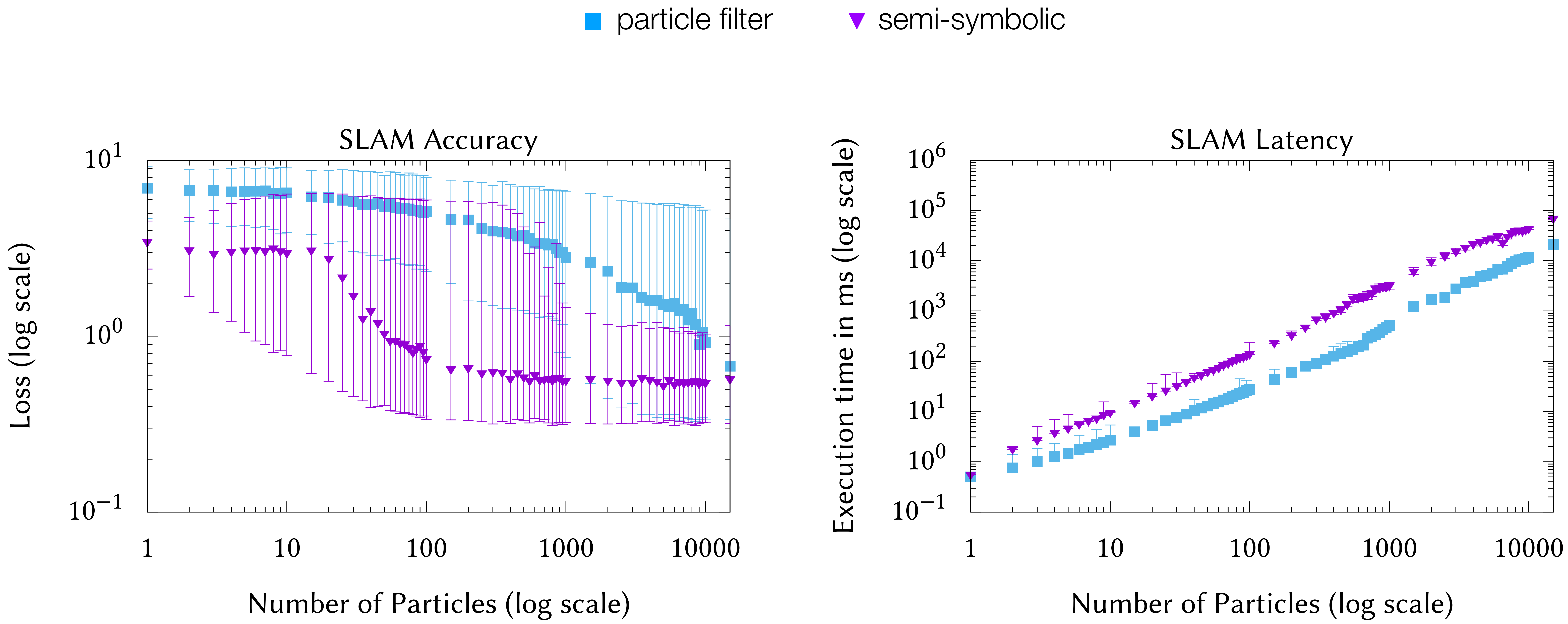



```
def tracker(data: list[X * A]) → list[X * A] * float * float:
  q = sample(InvGamma(1., 1.)) # move noise
  r = sample(InvGamma(1., 1.)) # observation noise
```



Resample: particle filtering

Speed vs. accuracy



Benchmarks

Time to reach a target accuracy

Baseline:
SDS with 1,000 particles

SSI computations cost more, but can outperform PF

	PF		DS		SSI	
MODEL	# PART.	TIME (MS)	# PART.	TIME (MS)	# PART.	TIME (MS)
Beta-Bernoulli	200	23.05 (22.54-23.90)	✓ 1	0.28 (0.27-0.28)	✓ 1	0.65 (0.65-0.66)
Gaussian-Gaussian	3000	877.44 (689.04-890.68)	150	62.99 (61.69-65.86)	150	182.57 (181.40-183.54)
Kalman-1D	15	3.27 (3.26-3.28)	✓ 1	0.33 (0.33-0.34)	✓ 1	1.15 (1.14-1.15)
Outlier	700	222.27 (220.76-223.84)	65	43.81 (43.45-46.48)	65	125.88 (125.27-128.08)
Robot	85	771.32 (767.98-775.51)	✓ 1	91.44 (90.94-92.07)	✓ 1	96.40 (96.21-97.53)
SLAM		✗	800	2812.55 (2755.99-2853.89)	800	5649.30 (5619.59-5675.81)
MTT		✗	60	2889.11 (2615.76-3244.30)	60	4457.79 (4068.35-4996.20)
Tree	150	35.55 (35.41-35.68)	90	58.83 (58.55-59.74)	✓ 1	2.67 (2.66-2.70)
Wheels	550	246.48 (245.06-248.75)	550	699.12 (672.25-713.64)	✓ 1	8.04 (8.00-8.10)
Delayed GPS	150	1221.00 (1218.76-1230.67)	9	304.73 (303.17-306.31)	✓ 1	108.55 (108.02-109.07)

Inference plan

Probabilistic Programming Languages

Probabilistic inference by program transformation in Hakaru (system description)*

Praveen Narayanan[1], Jacques Carette[2], Wren Romano[1], Chung-chieh Shan[1], and Robert Zinkov[1]

[1] Indiana University {pravnar,wrengr,ccshan,zinkov}@indiana.edu
[2] McMaster University carette@mcmaster.ca

Semi-symbolic Inference for Efficient Streaming Probabilistic Programming

ERIC ATKINSON, MIT, USA
CHARLES YUAN, MIT, USA
GUILLAUME BAUDART, ENS – PSL University – CNRS – Inria, France
LOUIS MANDEL, IBM Research, USA
MICHAEL CARBIN, MIT, USA

Delayed Sampling and Automatic Rao–Blackwellization of Probabilistic Programs

Lawrence M. Murray
Uppsala University

Daniel Lundén
KTH Royal Institute of Technology

Jan Kudlicka
Uppsala University

David Broman
KTH Royal Institute of Technology

Thomas B. Schön
Uppsala University

Automatic Rao-Blackwellization for Sequential Monte Carlo with Belief Propagation

Waïss Azizian[1] **Guillaume Baudart**[2] **Marc Lelarge**[3]

Automatically Marginalized MCMC in Probabilistic Programming

Jinlin Lai[1] **Javier Burroni**[1] **Hui Guan**[1] **Daniel Sheldon**[1]

Inference plan

```
def tracker(data):
  q:sample = sample(InvGamma(1., 1.))
  r:sample = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:symbolic = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

Symbolic x plan

Inference plan

```
def tracker(data):
  q:sample = sample(InvGamma(1., 1.))
  r:sample = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:symbolic = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

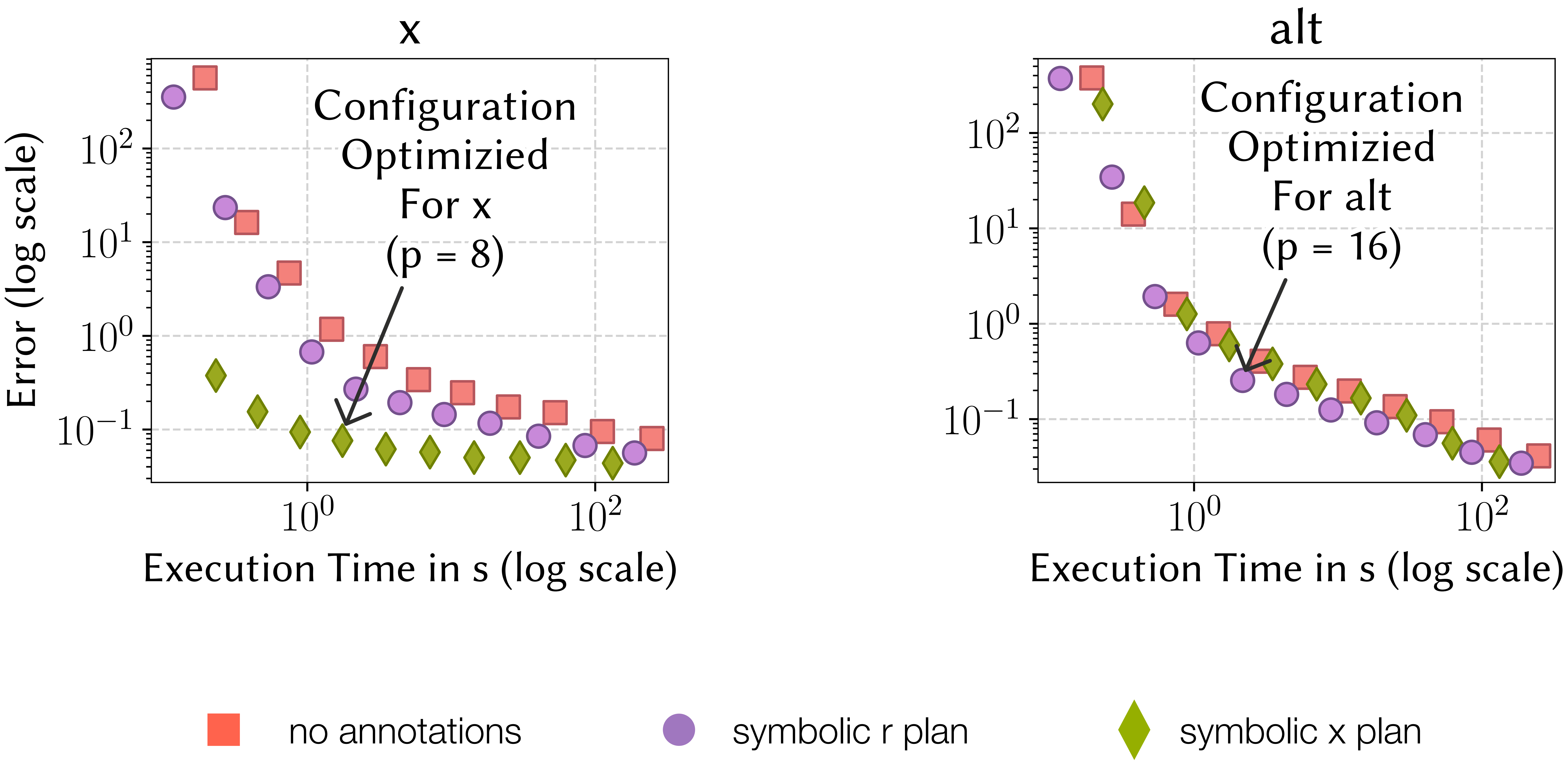
Symbolic x plan

```
def tracker(data):
  q:sample = sample(InvGamma(1., 1.))
  r:symbolic = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:sample = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

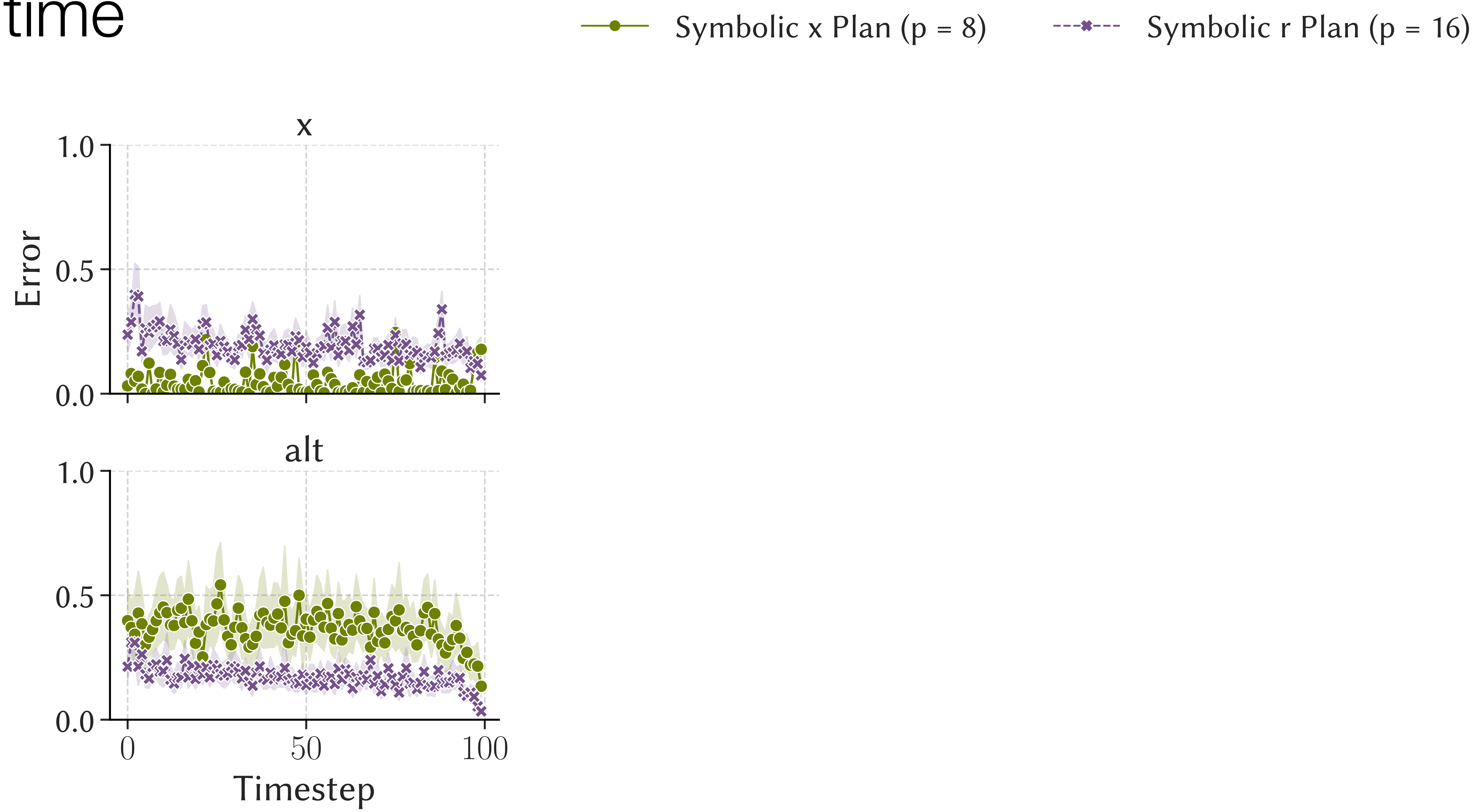
Symbolic r plan

Trade-off accuracy / speed

Max runtime: 3-seconds

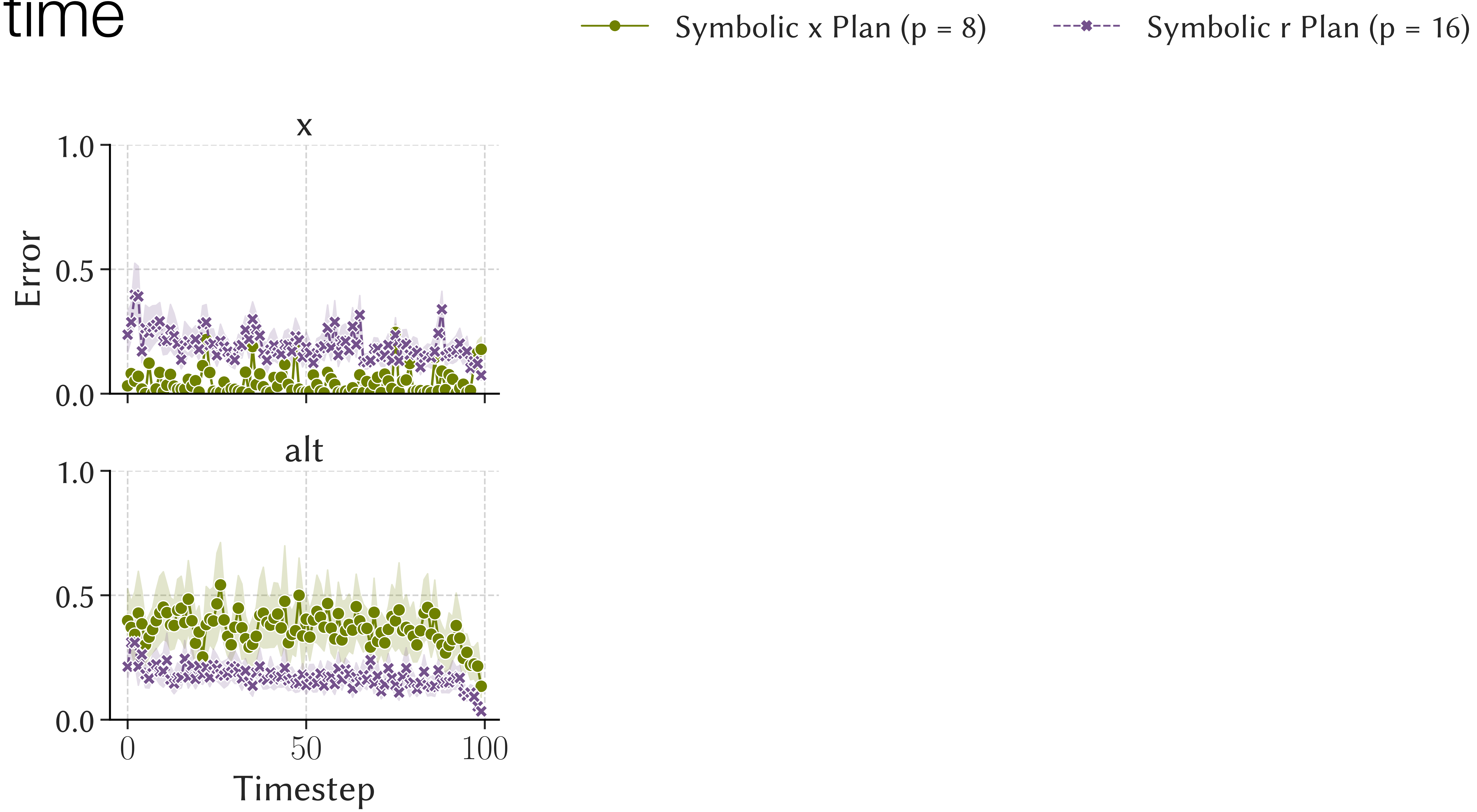


Runtime



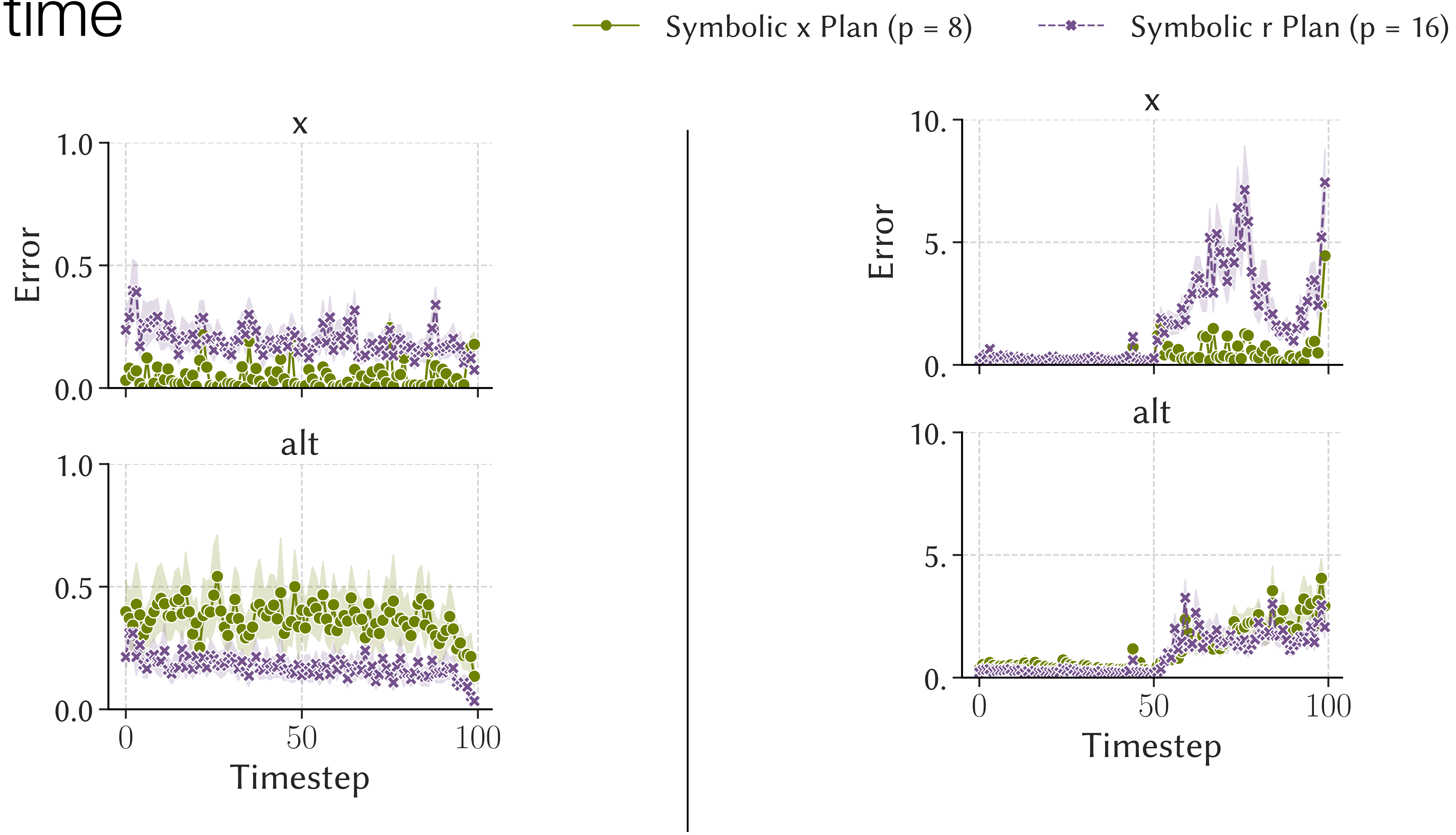
Symbolic x Plan (p = 8)
Symbolic r Plan (p = 16)
x
alt
Error
1.0
0.5
0.0
0
50
100
Timestep

Runtime



alt > 10

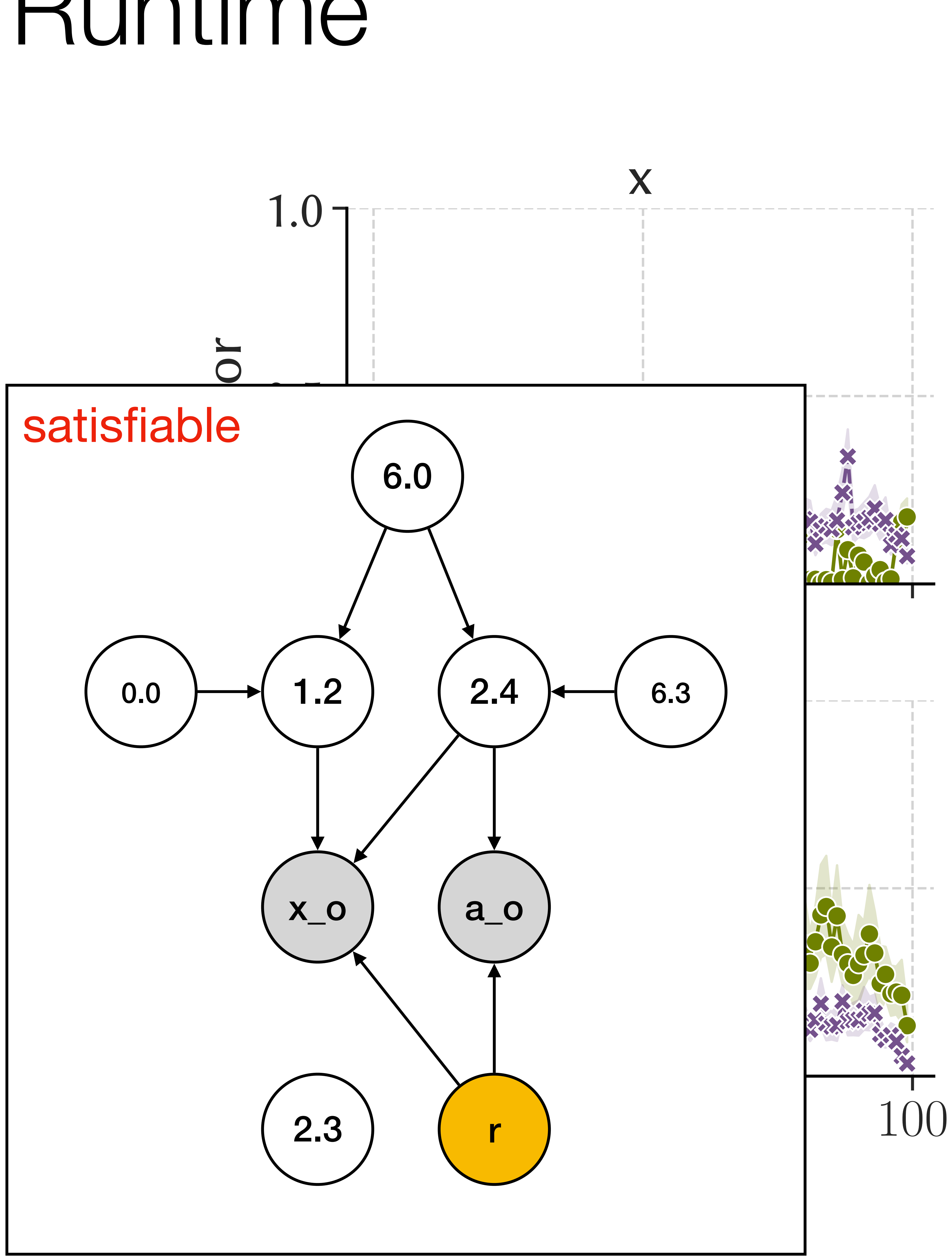
Runtime



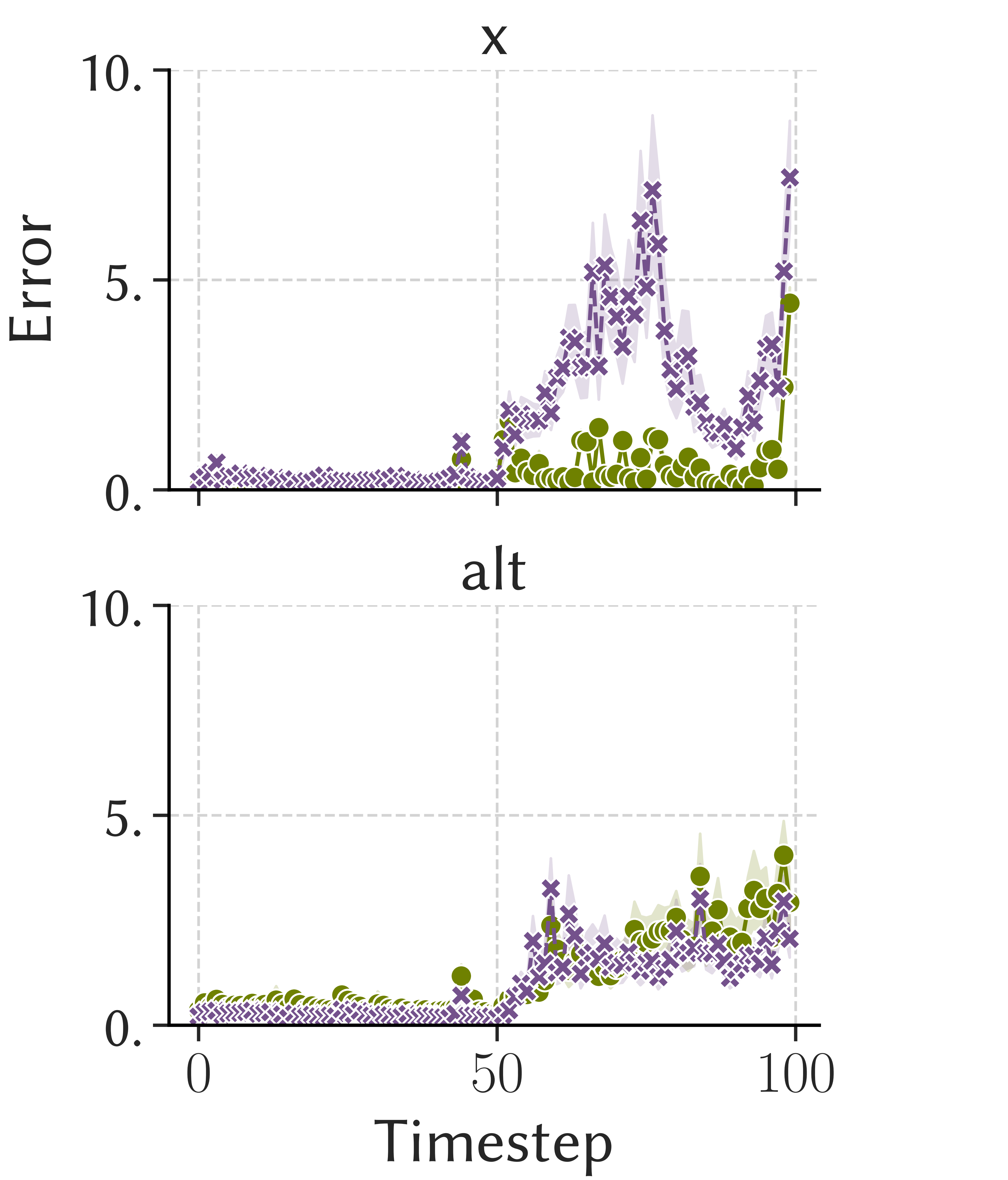
alt > 10

alt < 10

Runtime



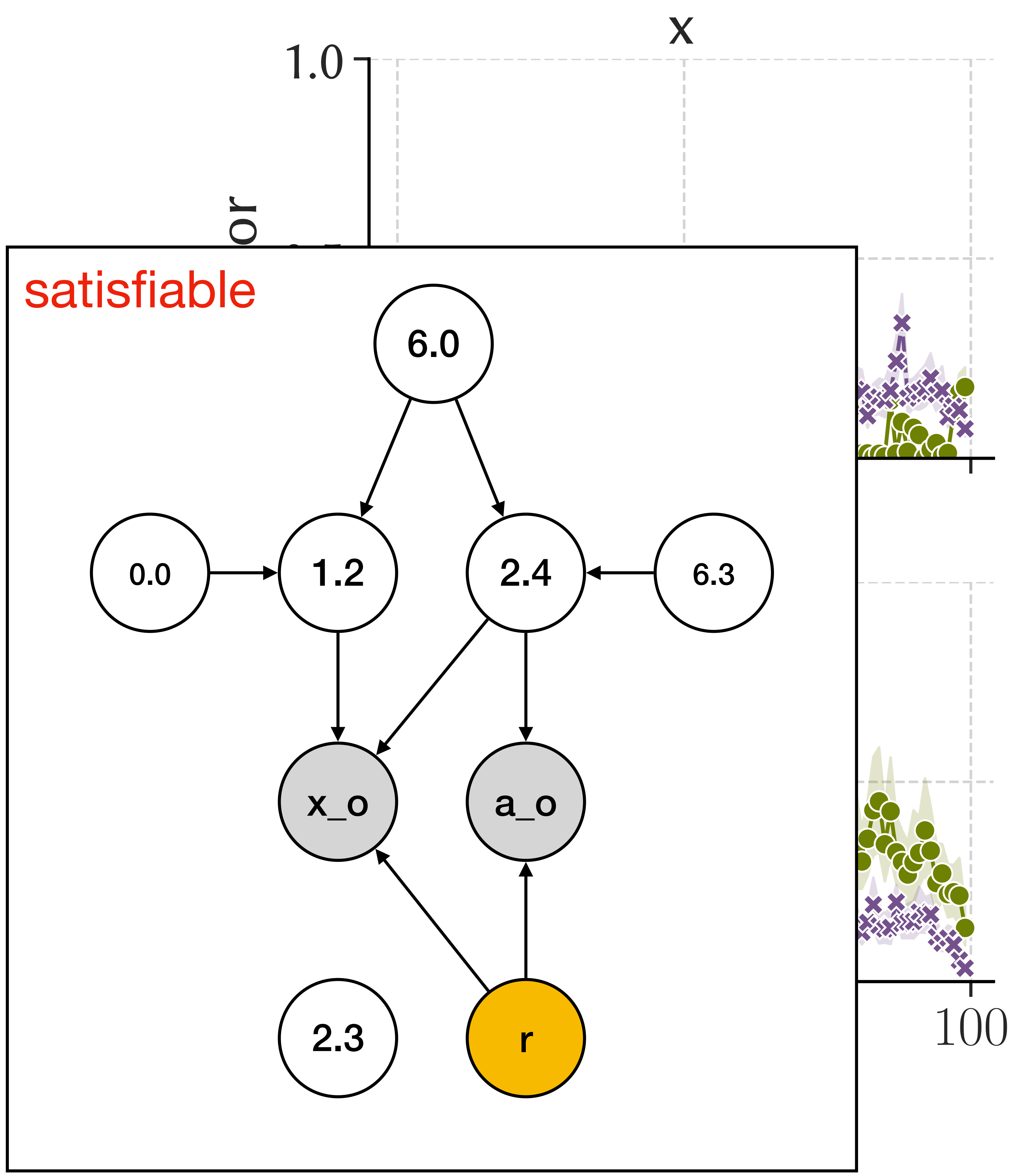
alt > 10



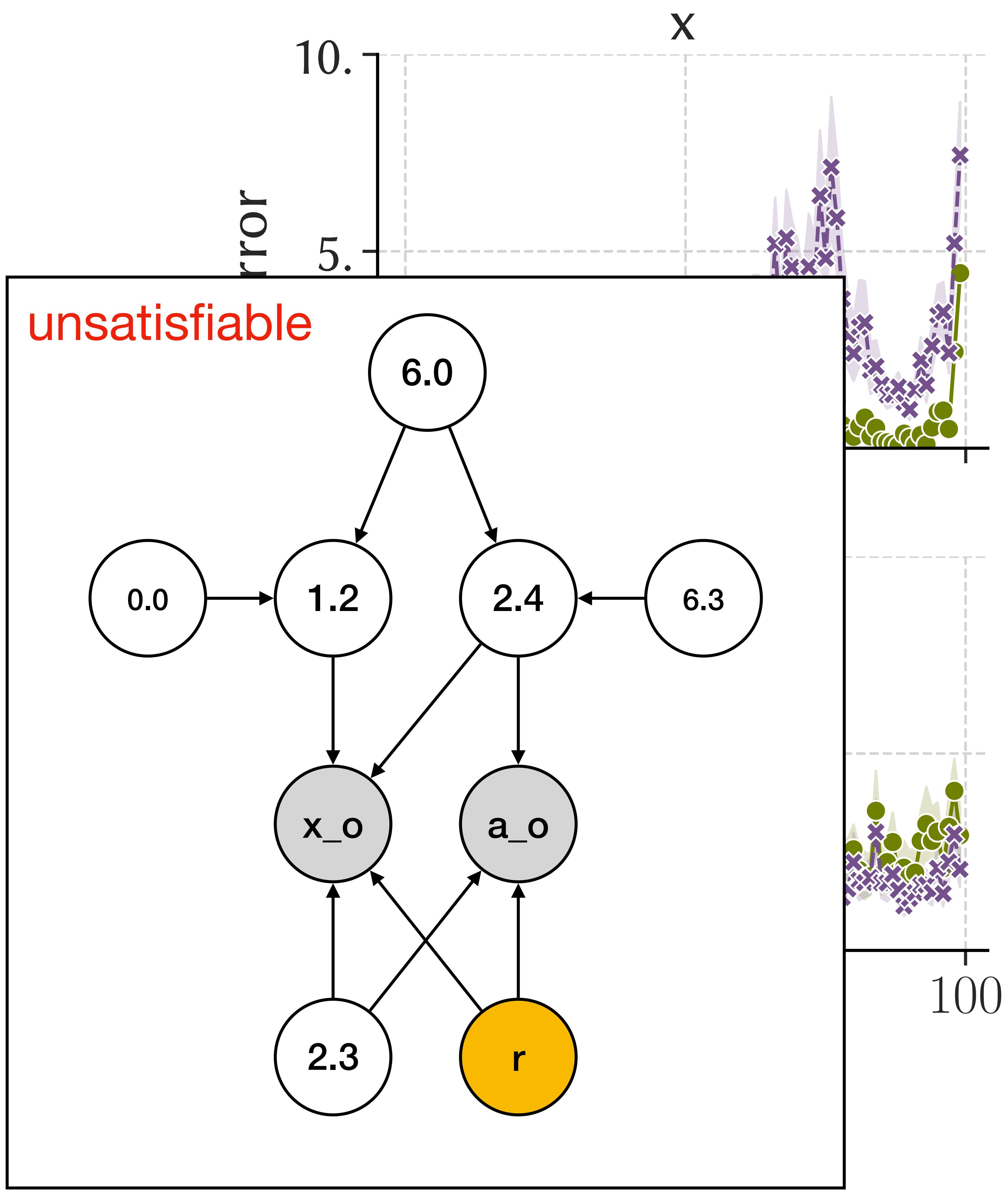
alt < 10

Runtime

Symbolic x Plan (p = 8) Symbolic r Plan (p = 16)

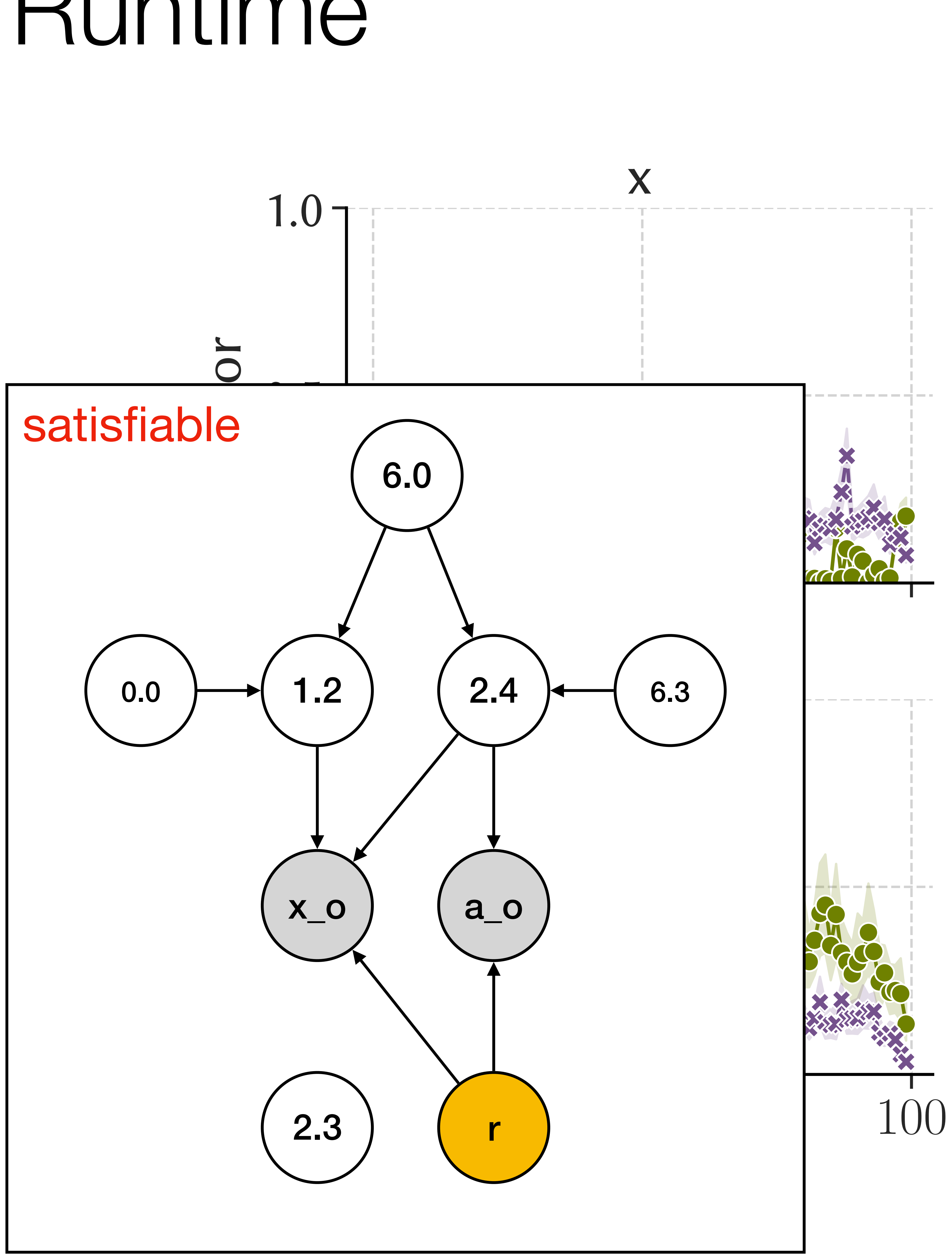


alt > 10

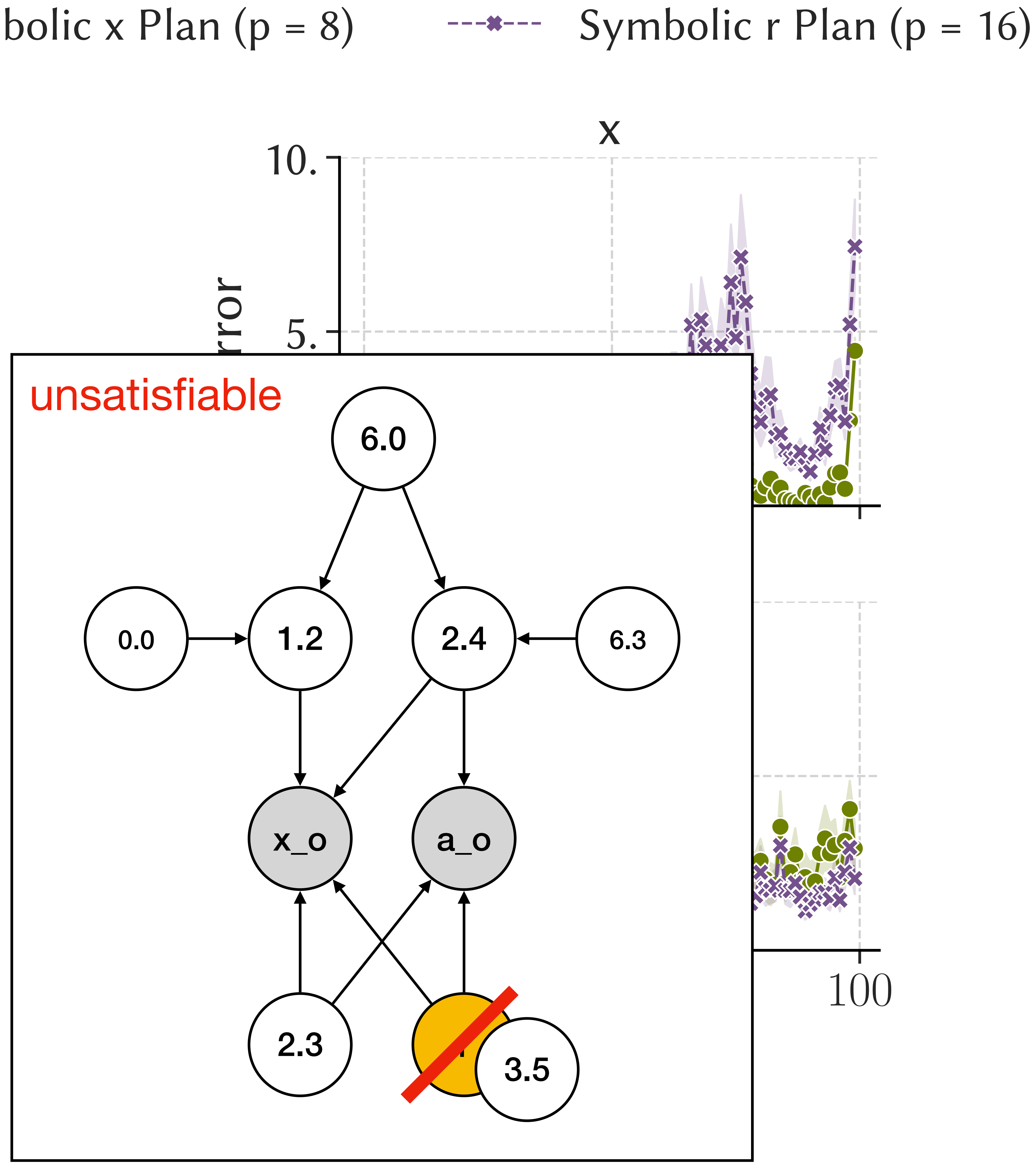


alt < 10

Runtime



alt > 10



alt < 10

Plan satisfiability analysis

Statically determine if a plan is satisfiable

- Check all possible executions of a program
- Formalized using abstract interpretation

Abstract symbolic expressions

```
D ::= N(E, E) | Bernoulli(E) |InvGamma(E, E) | Delta(E) | DeltaS(E)
    | UnkD(S) | TopD

E ::= r | c | x | E + E | E - E | E * E | E / E
    | UnkC | UnkE(S) | TopE
```

Soundness: If the analysis says that the plan is satisfiable, all possible executions are valid

Plan satisfiability analysis

Statically determine if a plan is satisfiable

- Check all possible executions of a program
- Formalized using abstract interpretation

Abstract symbolic expressions

```
D ::= N(E, E) | Bernoulli(E) |InvGamma(E, E) | Delta(E) | DeltaS(E)
    | UnkD(S) | TopD

E ::= r | c | x | E + E | E - E | E * E | E / E
    | UnkC | UnkE(S) | TopE
```

Soundness: If the analysis says that the plan is satisfiable, all possible executions are valid

Possible false alarms

Plan satisfiability analysis

```
def tracker(data: list[X * A]) → list[X * A] * float * float:
  q:sample = sample(InvGamma(1., 1.))
  r:symbolic = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:sample = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

Plan satisfiability analysis

$\hat{X}_r$

```
def tracker(data: list[X * A]) → list[X * A] * float * float:
  q:sample = sample(InvGamma(1., 1.))
  r:symbolic = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:sample = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

Plan satisfiability analysis

```
def tracker(data: list[X ⋆ A]) → list[X ⋆ A] ⋆ float ⋆ float:
  q:sample = sample(InvGamma(1., 1.))
  r:symbolic = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:sample = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

$\hat{X}_r$

UnkC

Plan satisfiability analysis

```
def tracker(data: list[X * A]) → list[X * A] * float * float:
  q:sample = sample(InvGamma(1., 1.))
  r:symbolic = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:sample = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

$\hat{X}_r$

UnkC

$\hat{X}_r$

Plan satisfiability analysis

```
def tracker(data: list[X * A]) → list[X * A] * float * float:
  q:sample = sample(InvGamma(1., 1.))
  r:symbolic = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:sample = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

$\hat{X}_r$

UnkC

$\hat{X}_r \hat{+}$ UnkC

$\hat{X}_r$

Plan satisfiability analysis

```
def tracker(data: list[X * A]) → list[X * A] * float * float:
  q:sample = sample(InvGamma(1., 1.))
  r:symbolic = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:sample = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

$\hat{X}_r$

UnkC

UnkE($\{\hat{X}_r\}$)

$\hat{X}_r \hat{+}$ UnkC

$\hat{X}_r$

Plan satisfiability analysis

```
def tracker(data: list[X * A]) → list[X * A] * float * float:
  q:sample = sample(InvGamma(1., 1.))
  r:symbolic = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:sample = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

$\hat{X}_r$

UnkC

$\mathrm{UnkE}(\{\hat{X}_r\})$

$\hat{X}_r \hat{+} \mathrm{UnkC}$

$\hat{X}_r$

$\mathcal{N}(\mathrm{UnkC}, \mathrm{UnkE}(\{\hat{X}_r\}))$

Plan satisfiability analysis

```
def tracker(data: list[X * A]) → list[X * A] * float * float:
  q:sample = sample(InvGamma(1., 1.))
  r:symbolic = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:sample = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

$\hat{X}_r$

UnkC

$\mathtt{UnkE}(\{\hat{X}_r\})$

$\hat{X}_r \mathbin{\hat{+}} \mathtt{UnkC}$

$\hat{X}_r$

$\mathcal{N}(\mathtt{UnkC}, \mathtt{UnkE}(\{\hat{X}_r\}))$

unsatisfiable

Plan satisfiability analysis

```
def tracker(data):
  q:sample = sample(InvGamma(1., 1.))
  r:sample = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:symbolic = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

Plan satisfiability analysis

$\hat{X}_x$

```
def tracker(data):
  q:sample = sample(InvGamma(1., 1.))
  r:sample = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:symbolic = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

Plan satisfiability analysis

$\hat{X}_x$

UnkC

```
def tracker(data):
  q:sample = sample(InvGamma(1., 1.))
  r:sample = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:symbolic = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

Plan satisfiability analysis

```
def tracker(data):
  q:sample = sample(InvGamma(1., 1.))
  r:sample = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:symbolic = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

$\hat{X}_x$

UnkC

UnkC $\hat{+}$ UnkC

Plan satisfiability analysis

```
def tracker(data):
  q:sample = sample(InvGamma(1., 1.))
  r:sample = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:symbolic = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

$\hat{X}_x$

UnkC

UnkC $\hat{+}$ UnkC

UnkC

Plan satisfiability analysis

```
def tracker(data):
  q:sample = sample(InvGamma(1., 1.))
  r:sample = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:symbolic = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

$\hat{X}_x$

UnkC

UnkC

$\mathrm{UnkC} \mathbin{\hat{+}} \mathrm{UnkC}$

UnkC

Plan satisfiability analysis

```
def tracker(data):
  q:sample = sample(InvGamma(1., 1.))
  r:sample = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:symbolic = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

$\hat{X}_x$

UnkC

UnkC

UnkC $\hat{+}$ UnkC

UnkC

$\mathcal{N}(\hat{X}_x, \mathrm{UnkC})$

Plan satisfiability analysis

```
def tracker(data):
  q:sample = sample(InvGamma(1., 1.))
  r:sample = sample(InvGamma(1., 1.))

  acc = [(0,0)]
  for (xo, ao) in data:
    pre_x, pre_a = acc[-1]
    x:symbolic = sample(Gaussian(pre_x, q))
    a:sample = sample(Gaussian(pre_a, q))
    other:sample = sample(InvGamma(1., 1.))
    v = r + other if a < 5 else r
    observe(Gaussian(x, v), xo)
    observe(Gaussian(a, v), ao)
    acc.append((x, a))

  return acc, q, r
```

$\hat{X}_x$

UnkC

UnkC

$\text{UnkC} \mathbin{\hat{+}} \text{UnkC}$

UnkC

$\mathcal{N}(\hat{X}_x, \text{UnkC})$

satisfiable

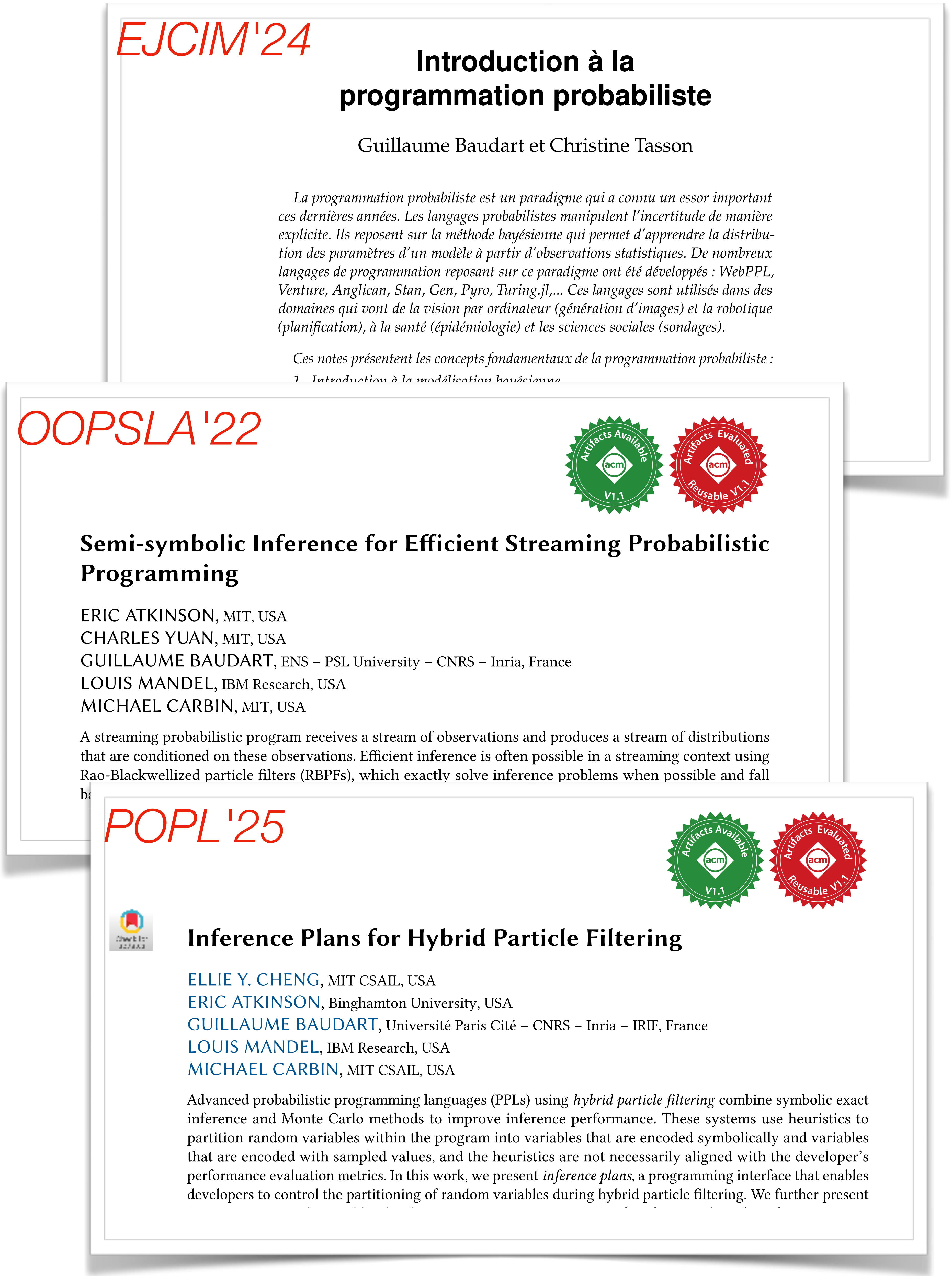
Takeaway

I - Probabilistic programming
- A program describes a distribution
- We can sample from distributions and condition on data
- Inference computes the distribution: Intractable problem in general

II - Approximate inference
- Importance sampling: weighted sampler returns pairs (value, score)
- Particle filtering: add checkpoints and resample particles

III - Semi-symbolic inference
- Each particle performs symbolic computations
- Fall back to a particle filter when symbolic computation fails

IV - Inference plan
- Control symbolic vs. approximate inference with annotations
- Static analysis checks if a plan is satisfiable



EJCIM'24

Introduction à la programmation probabiliste

Guillaume Baudart et Christine Tasson

La programmation probabiliste est un paradigme qui a connu un essor important ces dernières années. Les langages probabilistes manipulent l'incertitude de manière explicite. Ils reposent sur la méthode bayésienne qui permet d'apprendre la distribution des paramètres d'un modèle à partir d'observations statistiques. De nombreux langages de programmation reposant sur ce paradigme ont été développés : WebPPL, Venture, Anglican, Stan, Gen, Pyro, Turing.jl,... Ces langages sont utilisés dans des domaines qui vont de la vision par ordinateur (génération d'images) et la robotique (planification), à la santé (épidémiologie) et les sciences sociales (sondages).

Ces notes présentent les concepts fondamentaux de la programmation probabiliste :

OOPSLA'22

Semi-symbolic Inference for Efficient Streaming Probabilistic Programming

ERIC ATKINSON, MIT, USA
CHARLES YUAN, MIT, USA
GUILLAUME BAUDART, ENS – PSL University – CNRS – Inria, France
LOUIS MANDEL, IBM Research, USA
MICHAEL CARBIN, MIT, USA

A streaming probabilistic program receives a stream of observations and produces a stream of distributions that are conditioned on these observations. Efficient inference is often possible in a streaming context using Rao-Blackwellized particle filters (RBPFs), which exactly solve inference problems when possible and fall

POPL'25

Inference Plans for Hybrid Particle Filtering

ELLIE Y. CHENG, MIT CSAIL, USA
ERIC ATKINSON, Binghamton University, USA
GUILLAUME BAUDART, Université Paris Cité – CNRS – Inria – IRIF, France
LOUIS MANDEL, IBM Research, USA
MICHAEL CARBIN, MIT CSAIL, USA

Advanced probabilistic programming languages (PPLs) using *hybrid particle filtering* combine symbolic exact inference and Monte Carlo methods to improve inference performance. These systems use heuristics to partition random variables within the program into variables that are encoded symbolically and variables that are encoded with sampled values, and the heuristics are not necessarily aligned with the developer's performance evaluation metrics. In this work, we present *inference plans*, a programming interface that enables developers to control the partitioning of random variables during hybrid particle filtering. We further present