Delayed Sampling and Automatic Rao–Blackwellization of Probabilistic Programs

Lawrence Murray

Uppsala University

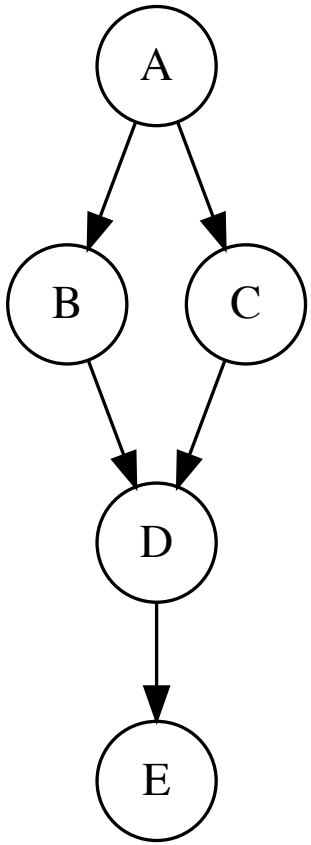
+ Daniel Lundén (KTH)
+ Jan Kudlicka (Uppsala)
+ David Broman (KTH)
+ Thomas Schön (Uppsala)

Outline

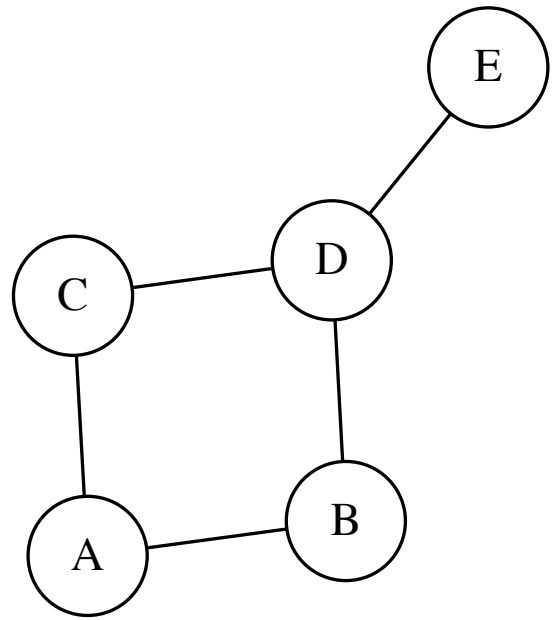
1. Graphical models $\longrightarrow$ probabilistic programs.
2. Monte Carlo methods for probabilistic programs.
3. Delayed sampling to automate variance reduction techniques.
4. The Birch probabilistic programming language (birch-lang.org).
5. Examples.

Graphical models

(a) Directed

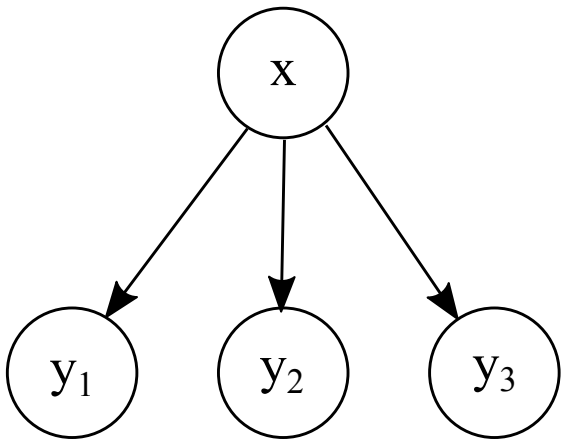


(b) Undirected

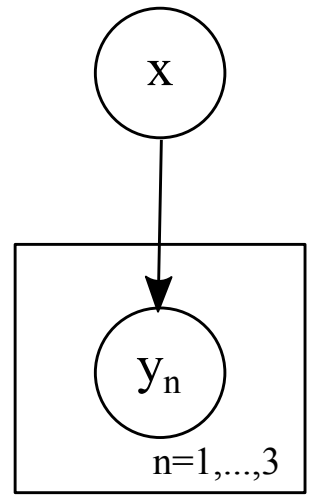


Graphical models

(a) Without plate notation



(b) With plate notation



Graphical models

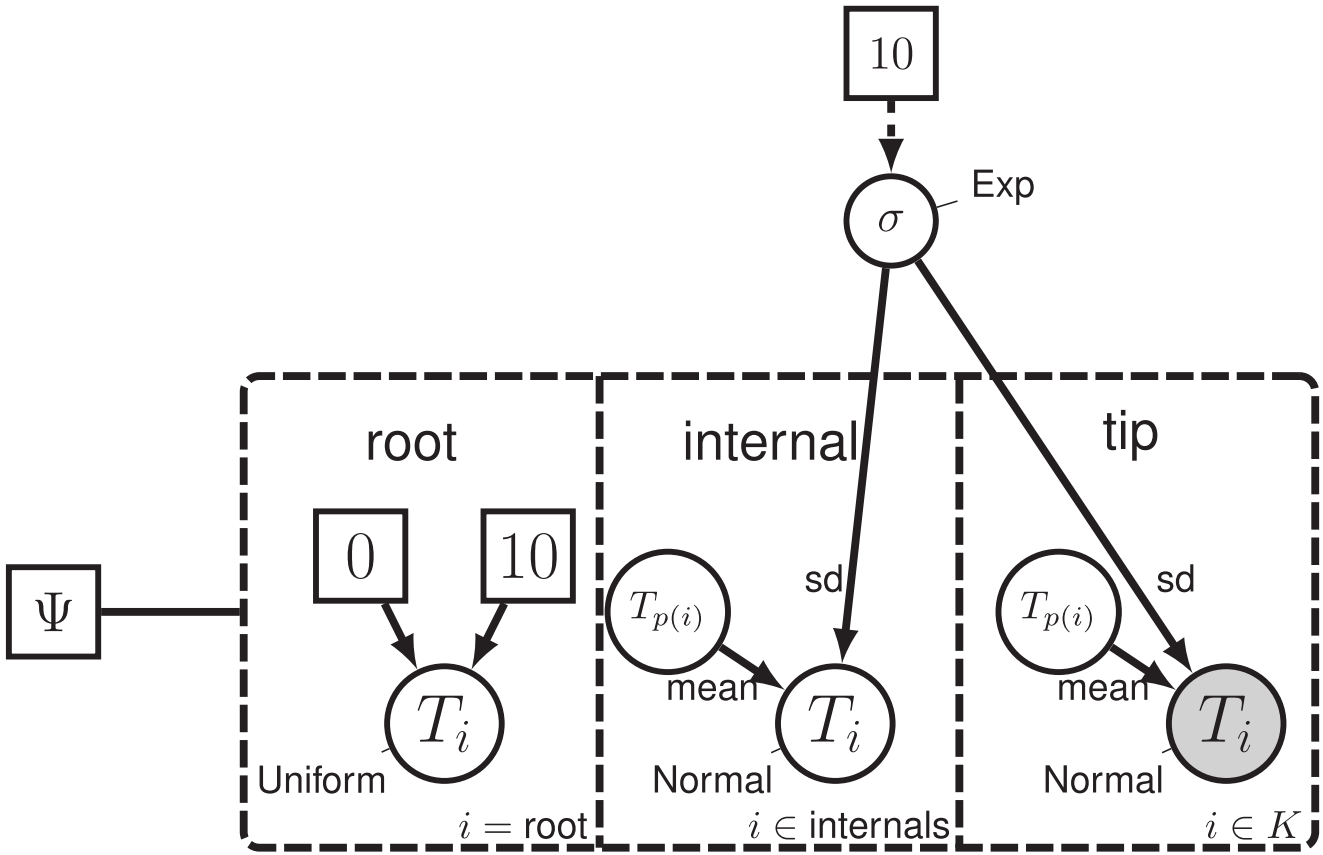


Figure: S. Höhna, M. J. Landis, T. A. Heath, B. Boussau, N. Lartillot, B. R. Moore, J. P. Huelsenbeck, and F. Ronquist. Revbayes: Bayesian phylogenetic inference using graphical models and an interactive model-specification language. Systematic, 65(4):726–736, 2016.

doi: 10.1093/sysbio/syw021

Graphical models

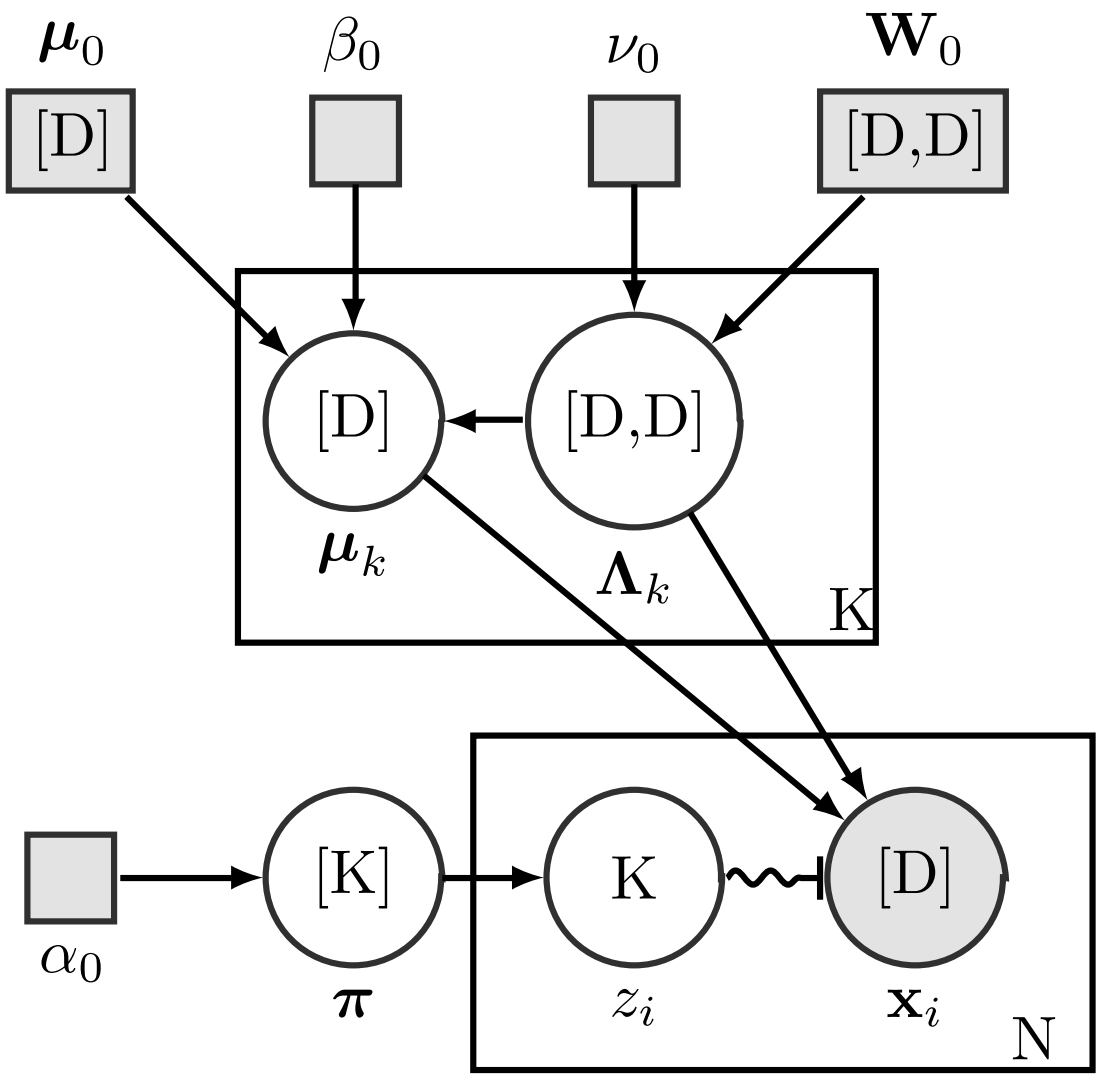
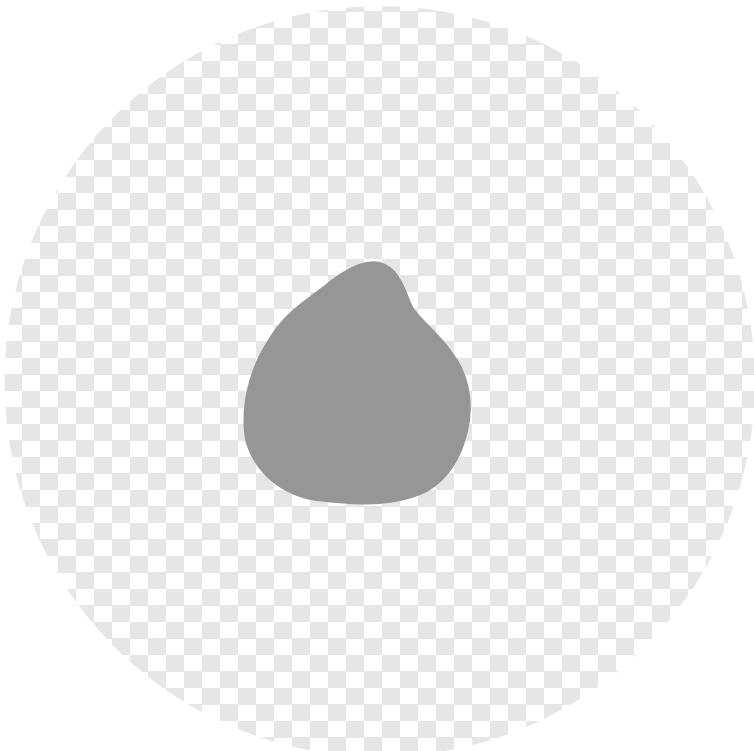


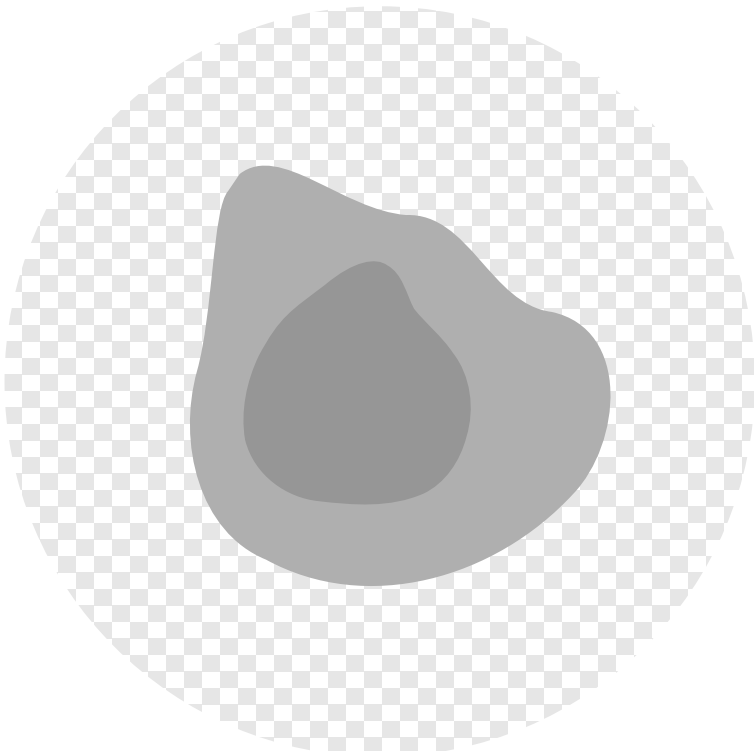
Figure: Benwing `https://commons.wikimedia.org/wiki/File:Bayesian-gaussian-mixture.svg`

Graphical models $\longrightarrow$ probabilistic programs

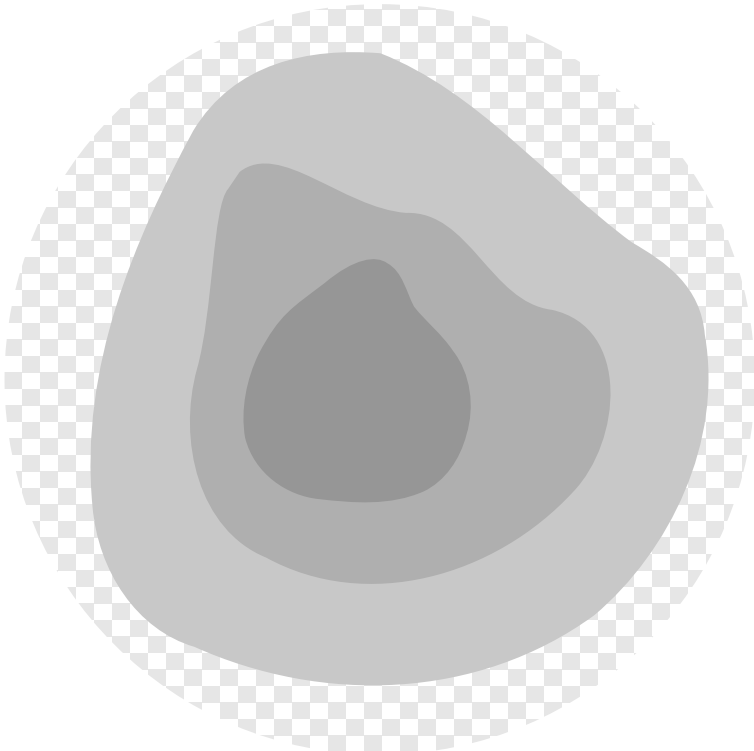
Graphical models $\longrightarrow$ probabilistic programs



Graphical models $\longrightarrow$ probabilistic programs



Graphical models $\longrightarrow$ probabilistic programs



Graphical models $\longrightarrow$ probabilistic programs

The most expressive languages are known as **universal**.

Also known as **Turing complete**.

Models written in such languages are **universal probabilistic programs**.

These are the most expressive languages for model specification, but also the most difficult for which to do inference.

Inference

As a probabilistic program runs, its memory state evolves **dynamically** and **stochastically** in time.

- Let $k = 1, 2, 3, \ldots$ denote a sequence of **checkpoints**.
- Let $(X_k)_{k=1}^{\infty}$ denote the (memory) state of the running program at these checkpoints, where $X_k \in \mathbb{X}_k$.
- The state transitions according to $X_k \mid x_{k-1} \sim p_k(dx_k \mid x_{k-1})$.
- At each checkpoint we can manipulate the running program: pause execution, inspect memory state, consider distributions over that state, modify that state. This is what facilitates inference.

Two checkpoint types

A typical formulation uses two checkpoint types:

- **sample** to sample a random variable.
- **observe** to condition on a random variable having some value.

Two checkpoint types

A typical formulation uses two checkpoint types:

- **sample** to sample a random variable.
- **observe** to condition on a random variable having some value.

If we simulate at **sample** checkpoints and update a weight at **observe** checkpoints, we have an **importance sampler**.

Two checkpoint types

A typical formulation uses two checkpoint types:

- **sample** to sample a random variable.
- **observe** to condition on a random variable having some value.

If we simulate at **sample** checkpoints and update a weight at **observe** checkpoints, we have an **importance sampler**.

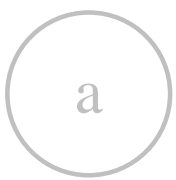
If we also run N instances of the program simultaneously and add resampling, we have a **bootstrap particle filter**.

Canonical Example with Two Checkpoints

Program	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	

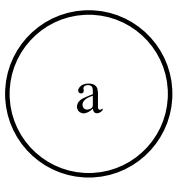
Canonical Example with Two Checkpoints

Program	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	**sample** a
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



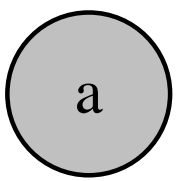
Canonical Example with Two Checkpoints

Program	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	**sample** a
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



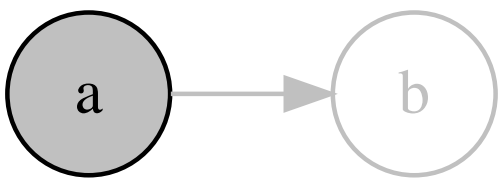
Canonical Example with Two Checkpoints

Program	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	**sample** a
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



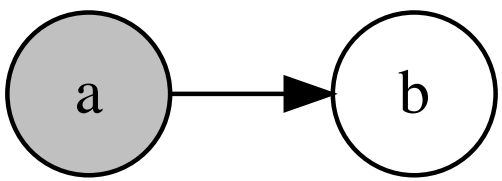
Canonical Example with Two Checkpoints

Program	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	**sample** b
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



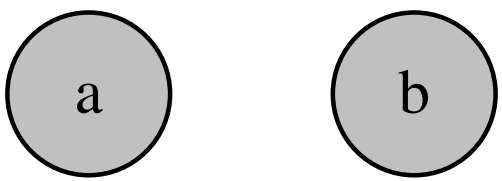
Canonical Example with Two Checkpoints

Program	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	**sample** b
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



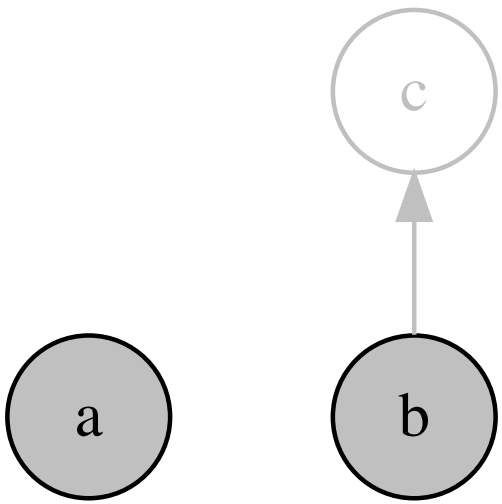
Canonical Example with Two Checkpoints

Program	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	**sample** b
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



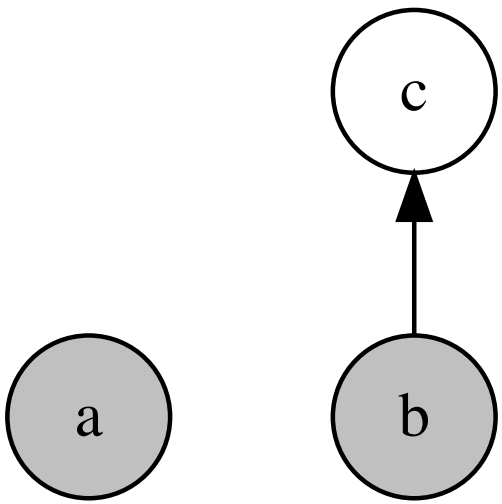
Canonical Example with Two Checkpoints

Program	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	**sample** c
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



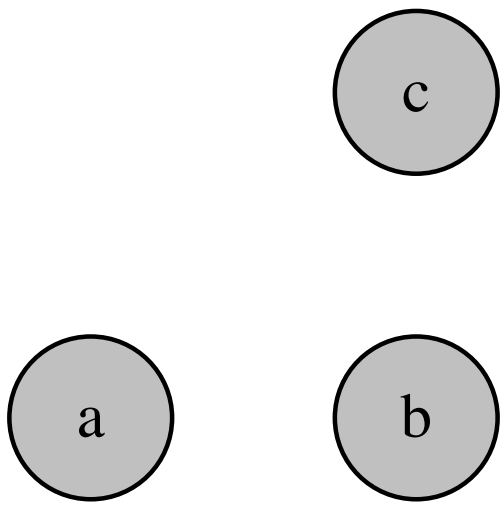
Canonical Example with Two Checkpoints

Program	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	**sample** c
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



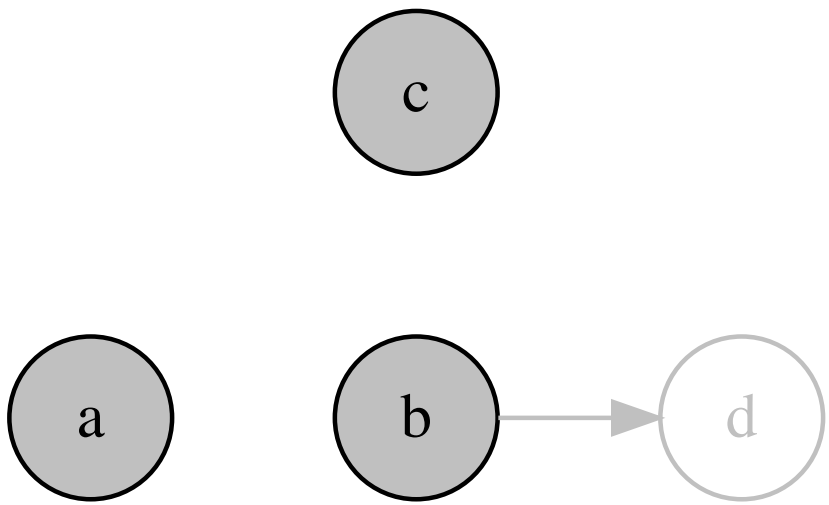
Canonical Example with Two Checkpoints

Program	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	**sample** c
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



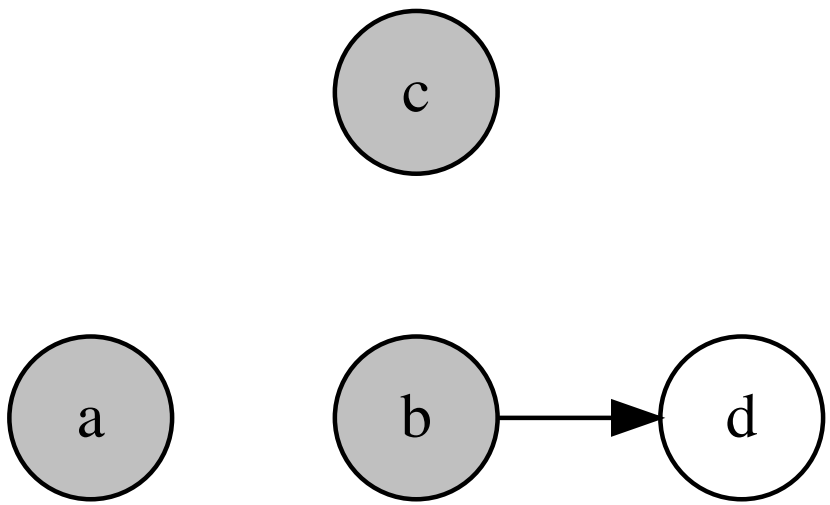
Canonical Example with Two Checkpoints

Program	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	**sample** d
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



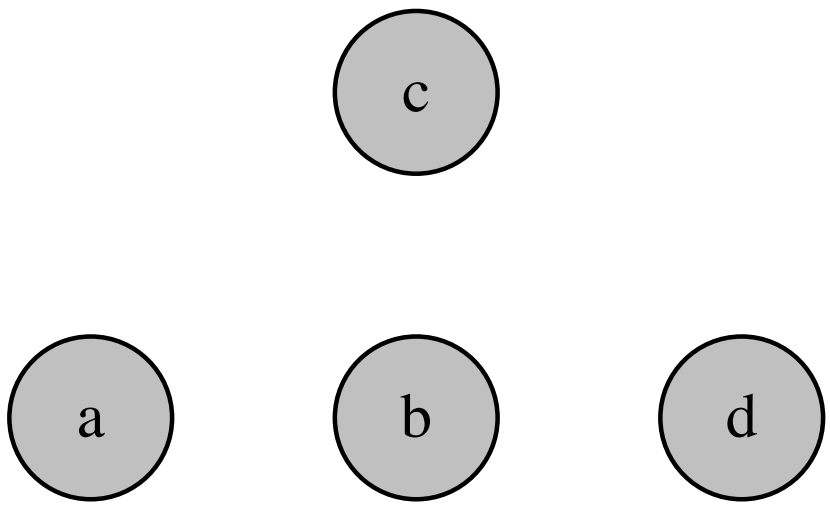
Canonical Example with Two Checkpoints

Program	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	**sample** d
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



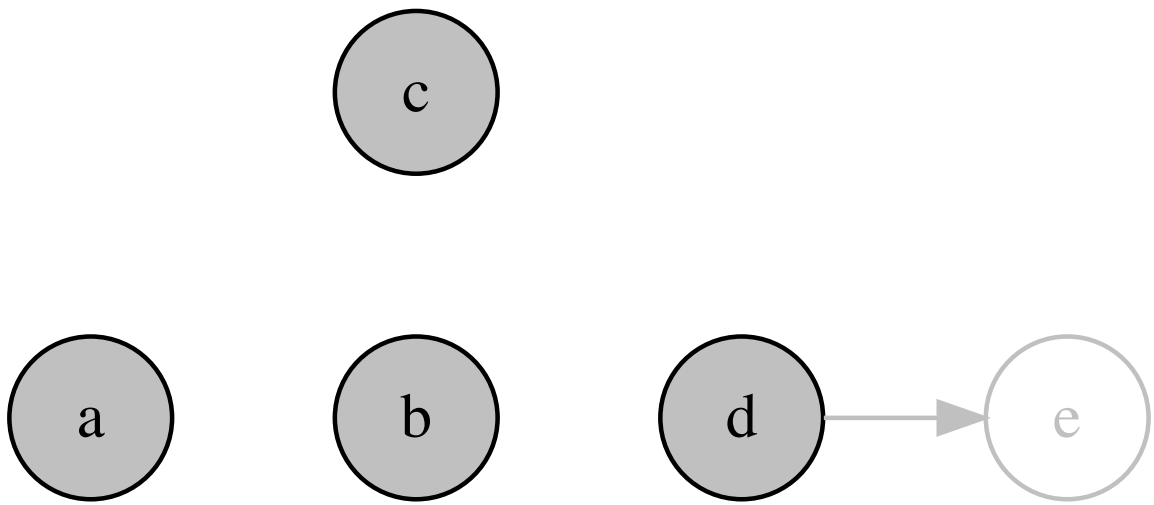
Canonical Example with Two Checkpoints

Program	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	**sample** d
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



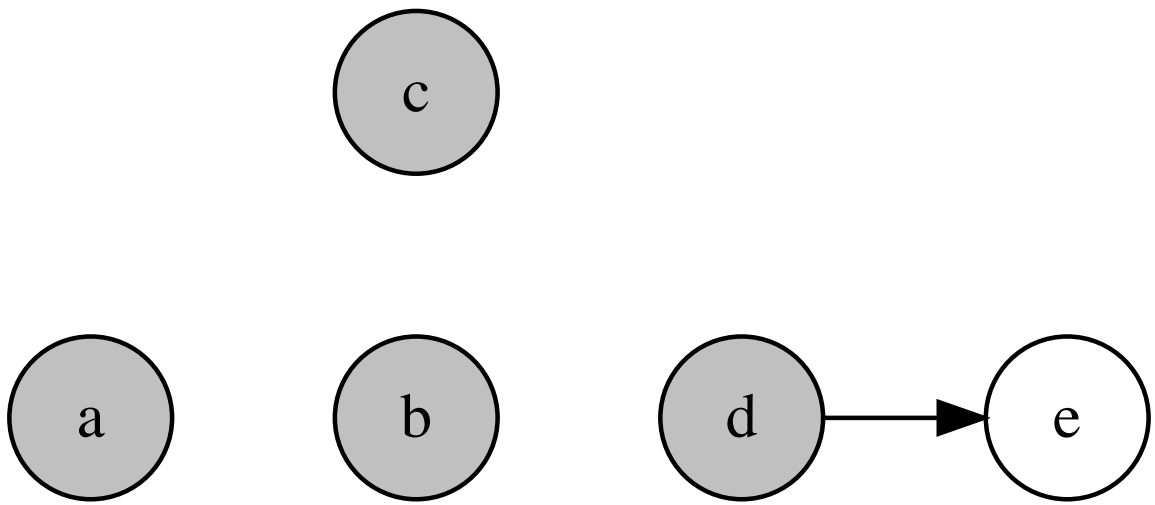
Canonical Example with Two Checkpoints

Program	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	**observe** e
`stdout.print(c);`	



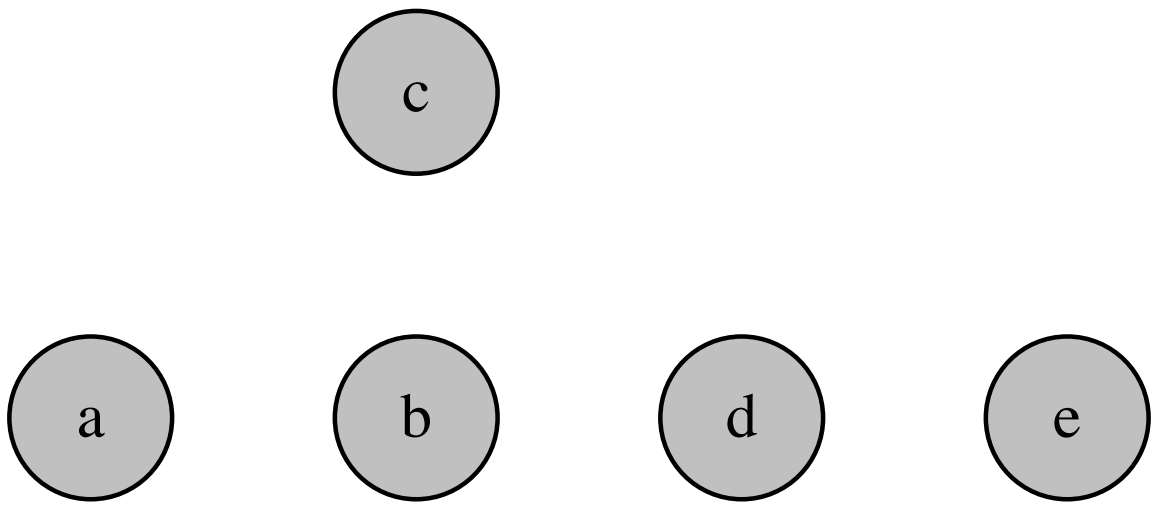
Canonical Example with Two Checkpoints

Program	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	**observe** e
`stdout.print(c);`	



Canonical Example with Two Checkpoints

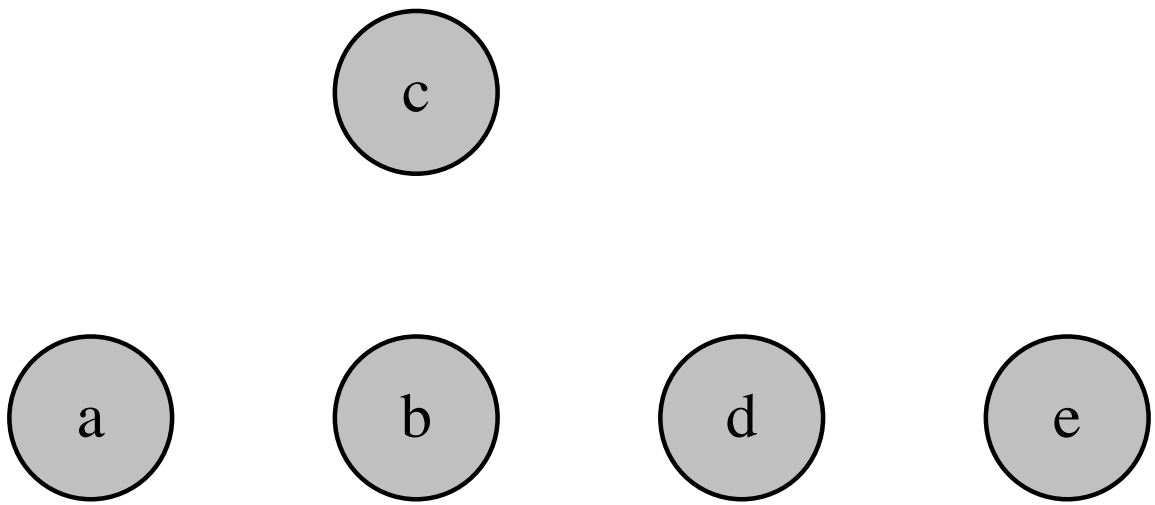
Program	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	**observe** e
`stdout.print(c);`	



Canonical Example with Two Checkpoints

Program	Checkpoint

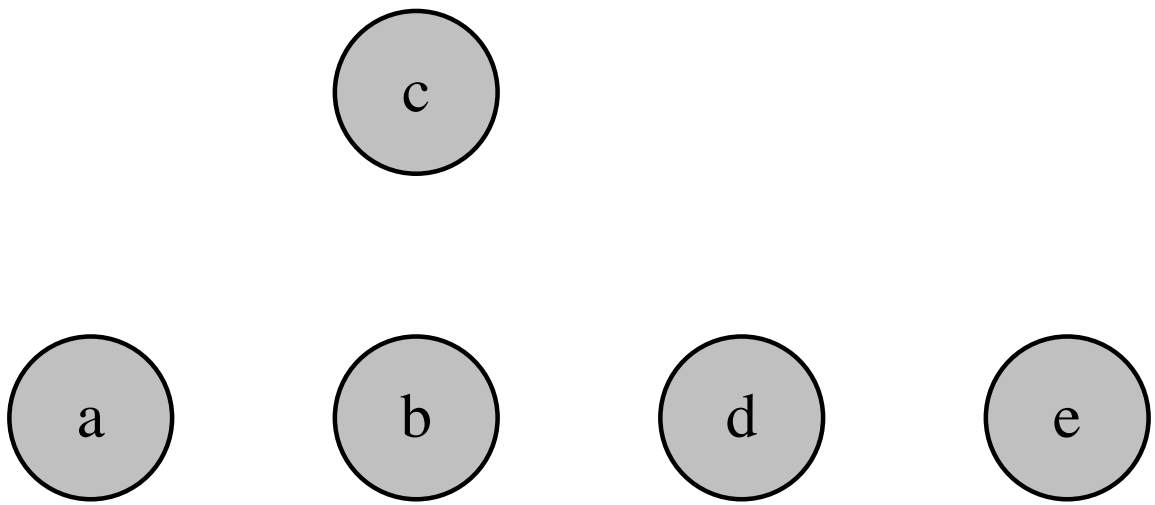
```
a ~ Gaussian(0.0, 1.0);
b ~ Gaussian(a, 1.0);
c ~ Gaussian(b, 1.0);
d ~ Gaussian(b, 1.0);
e ~ Gaussian(d, 1.0);
stdout.print(c);
```



Canonical Example with Two Checkpoints

Program	Checkpoint

```
a ~ Gaussian(0.0, 1.0);
b ~ Gaussian(a, 1.0);
c ~ Gaussian(b, 1.0);
d ~ Gaussian(b, 1.0);
e ~ Gaussian(d, 1.0);
stdout.print(c);
```



The problem

This **eager sampling** approach misses opportunities for variance reduction such as Rao–Blackwellization, variable elimination, and locally-optimal proposals.

- We would like to apply such improvements in an automatic way that does not require modification of the probabilistic program.
- But we don't know the model structure until we execute the program.
- So some heuristic decisions must be made without complete knowledge of the model structure.

Delayed sampling

We use three checkpoint types:

- **assume** to initialize a random variable with some distribution.
- **observe** to condition, given some value for a random variable.
- **value** when an actual variate is required.

Delayed sampling

We use three checkpoint types:

- **assume** to initialize a random variable with some distribution.
- **observe** to condition, given some value for a random variable.
- **value** when an actual variate is required.

Between **assume** and **value**, the distribution for a random variable can be updated using analytical relationships, such as conjugate priors.

The graph

- Alongside the state X, we maintain a graph $G = (V, E)$ that evolves dynamically as the program executes. At any time it represents only a fraction of the full model.
- We partition V into three disjoint sets according to three states:

$I \subseteq V$, the set of nodes in an 

state,

$M \subseteq V$, the set of nodes in a 

state, and

$R \subseteq V$, the set of nodes in a 

state.

- Each node transitions through the states in this order.

The graph

- **Initialize** inserts a new node v into the graph. If v has a parent, then the edge (u, v) is inserted.

  This moves v to the 

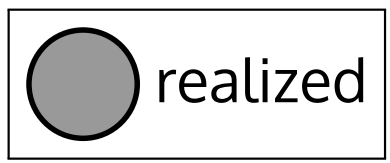
  state.

- **Marginalize** updates the distribution of node v by marginalizing over its parent.

  This moves v to the 

  state.

- **Sample** or **Observe** assigns a value to v by either sampling or observing, respectively.

  This moves v to the 

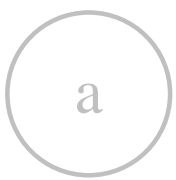
  state.

Canonical Example with Three Checkpoints

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	

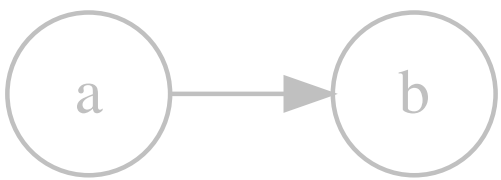
Canonical Example with Three Checkpoints

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	**assume** a
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



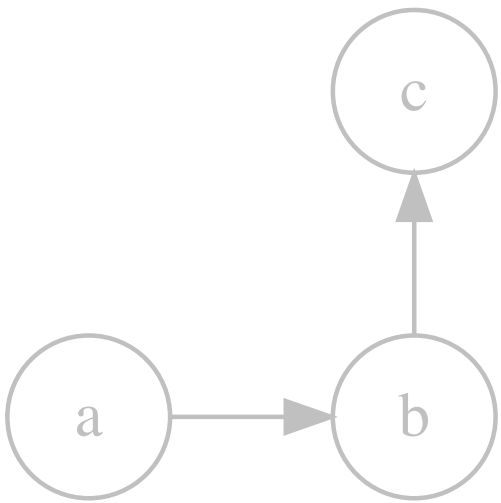
Canonical Example with Three Checkpoints

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	**assume** b
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



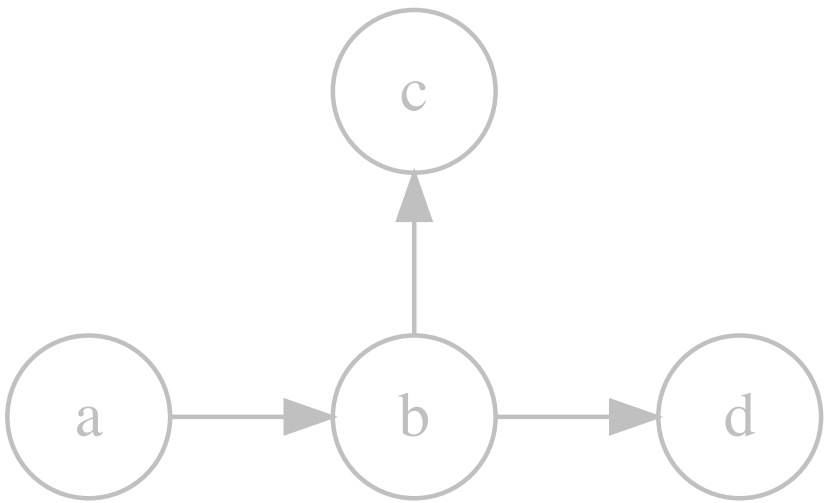
Canonical Example with Three Checkpoints

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	**assume** `c`
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



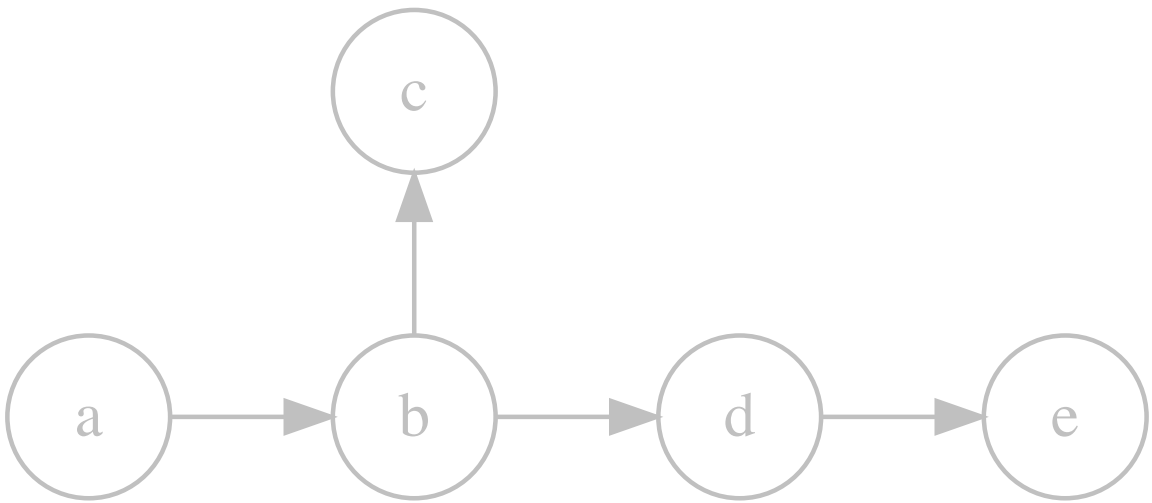
Canonical Example with Three Checkpoints

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	**assume** d
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



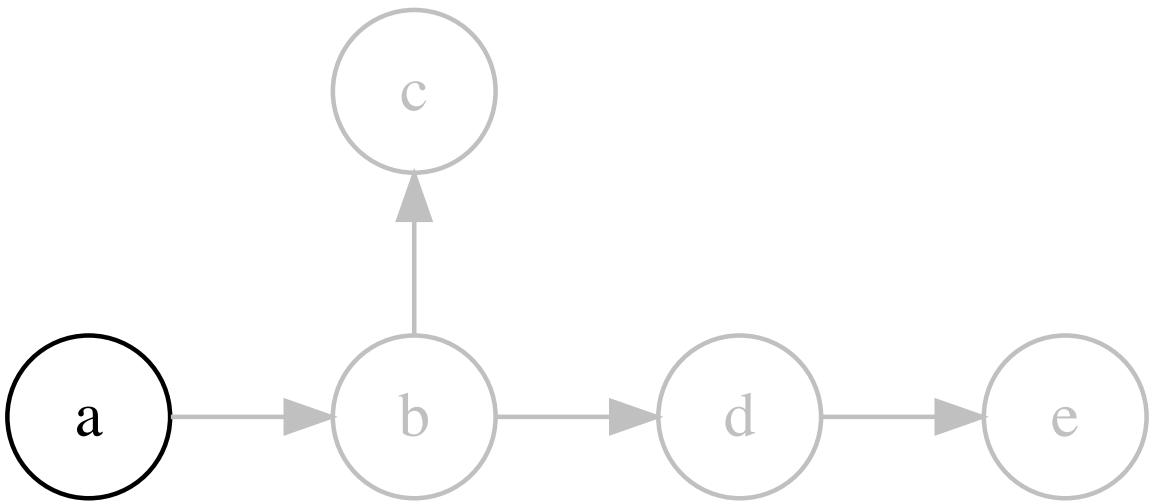
Canonical Example with Three Checkpoints

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	**observe** e
`stdout.print(c);`	



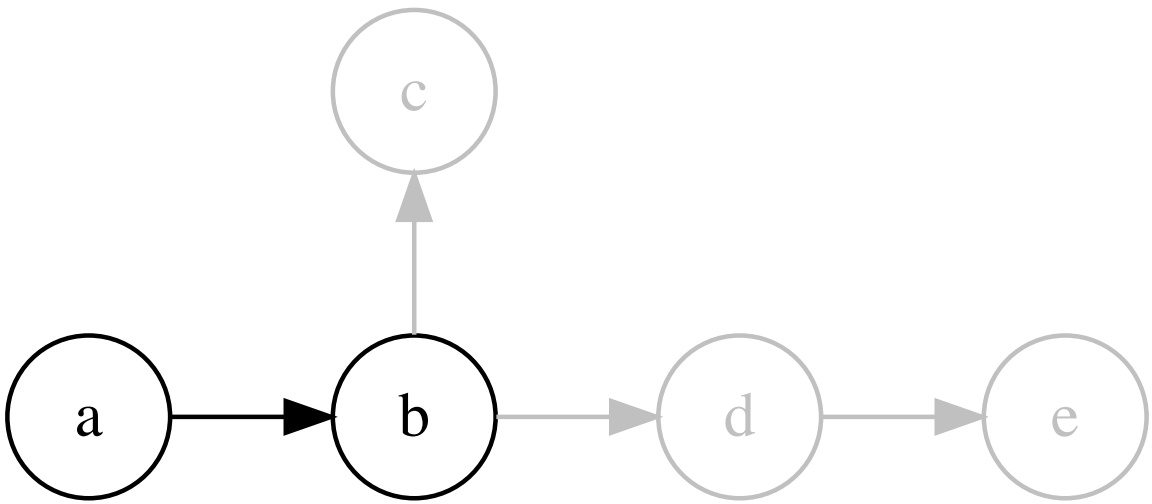
Canonical Example with Three Checkpoints

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	**observe** e
`stdout.print(c);`	



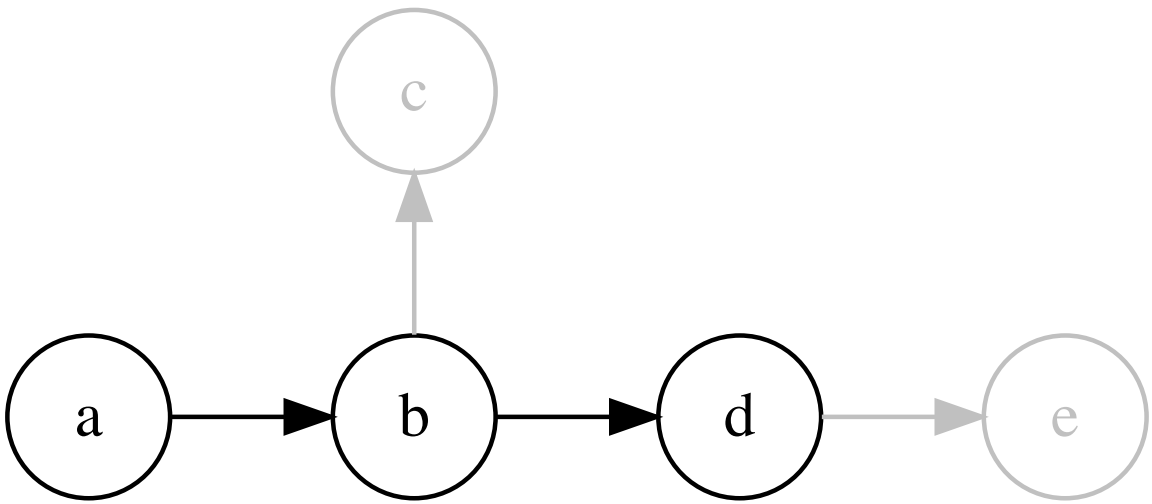
Canonical Example with Three Checkpoints

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	**observe** e
`stdout.print(c);`	



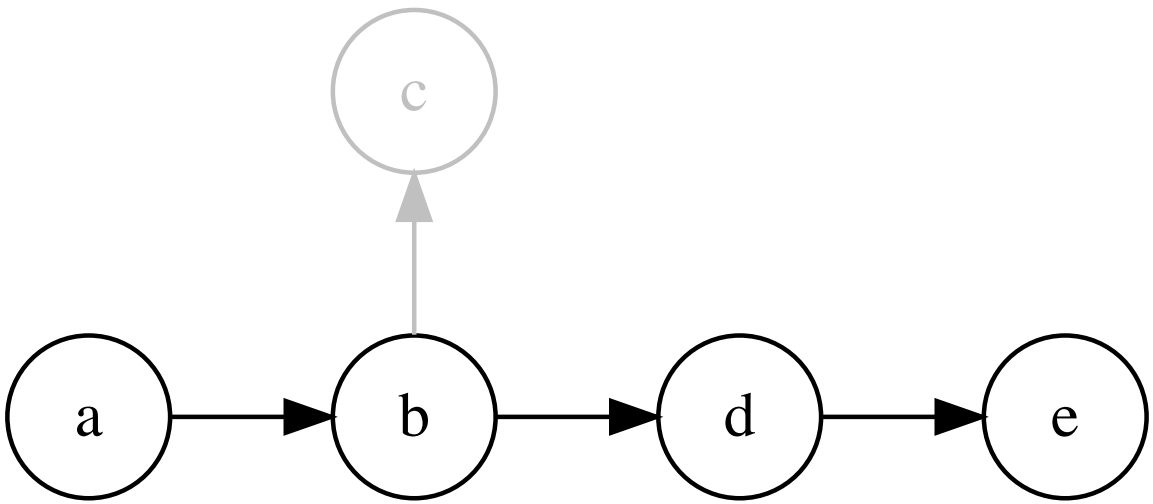
Canonical Example with Three Checkpoints

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	**observe** e
`stdout.print(c);`	



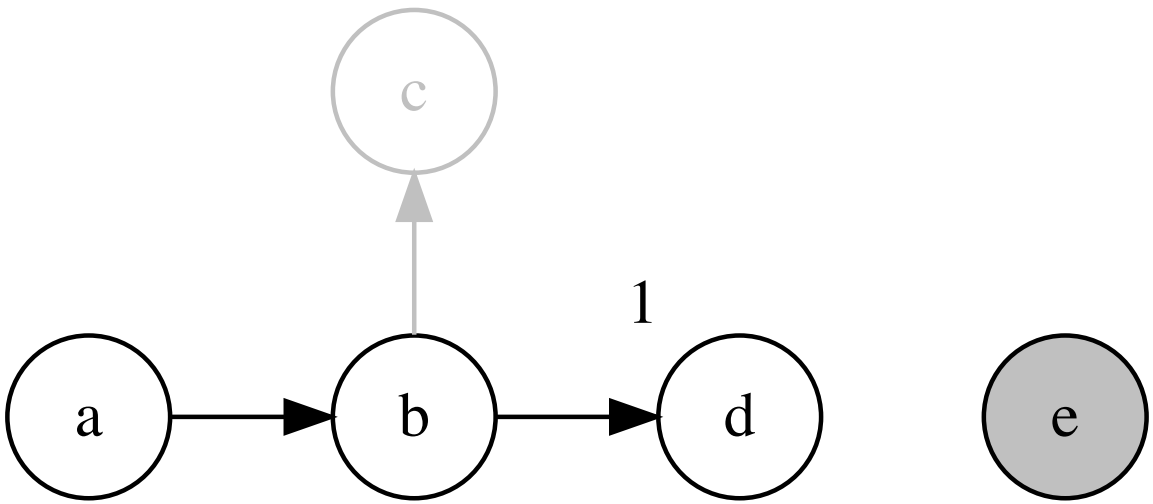
Canonical Example with Three Checkpoints

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	**observe** e
`stdout.print(c);`	



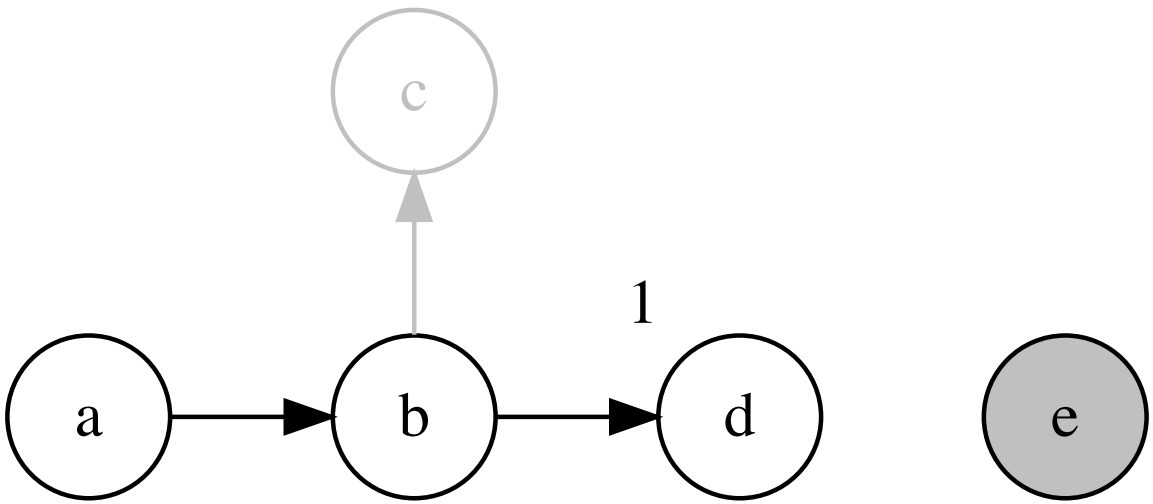
Canonical Example with Three Checkpoints

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	**observe** `e`
`stdout.print(c);`	



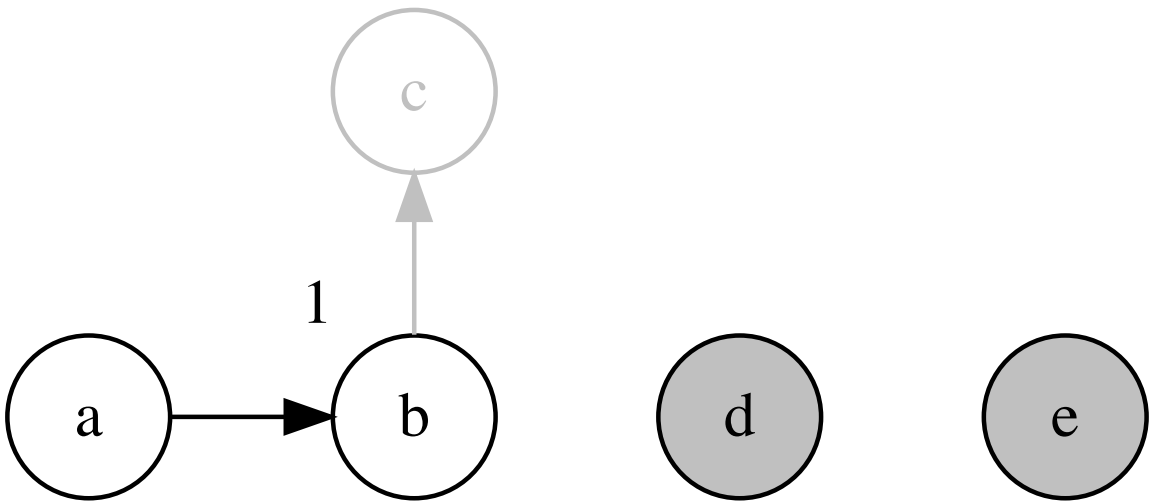
Canonical Example with Three Checkpoints

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	**value** `c`



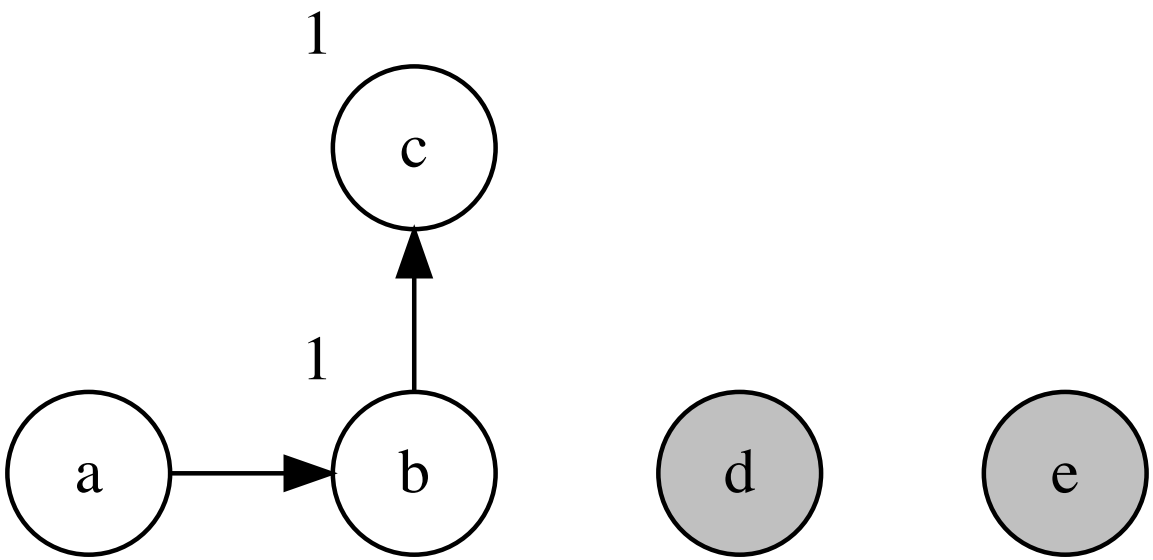
Canonical Example with Three Checkpoints

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	**value** `c`



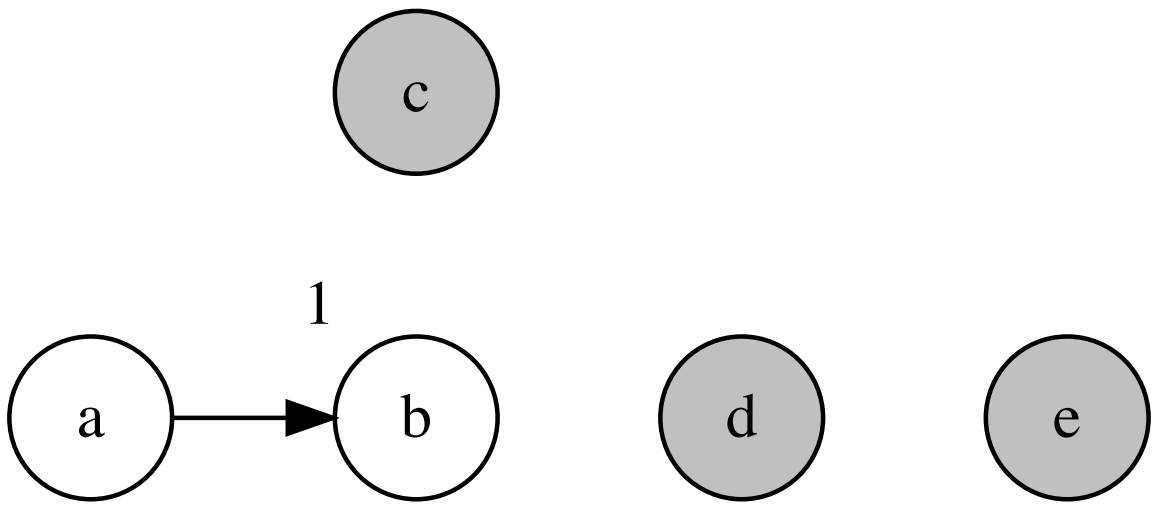
Canonical Example with Three Checkpoints

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	**value** `c`



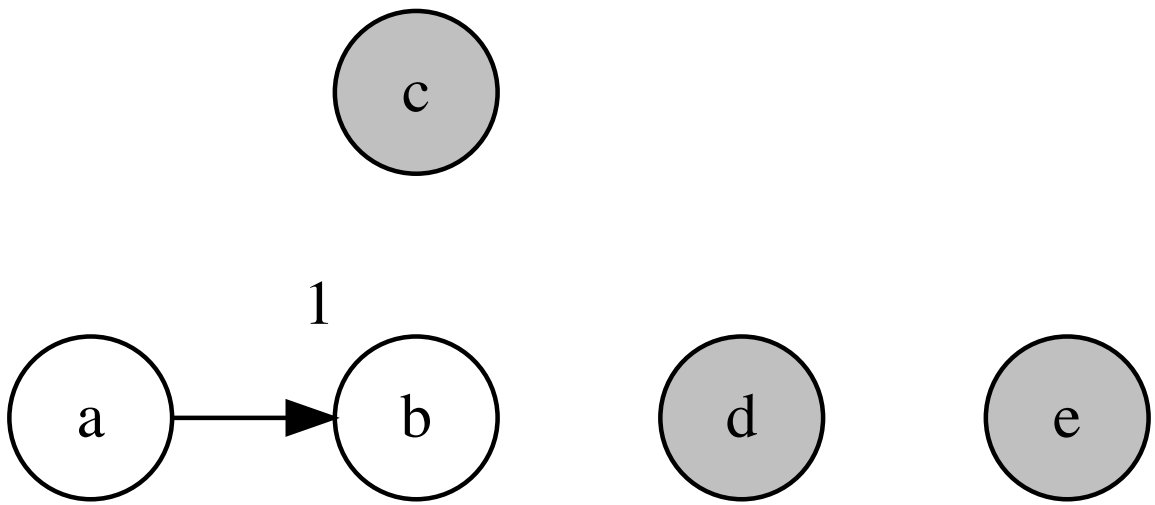
Canonical Example with Three Checkpoints

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	**value** `c`



Canonical Example with Three Checkpoints

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



Invariant conditions

1. The graph is a forest of disjoint trees.
2. If a node is  marginalized

   then its parent is  marginalized
3. A node has at most one child that is  marginalized

These imply that the marginalized nodes form a path: from the root to a tip.

Invariant conditions

1. The graph is a forest of disjoint trees.
2. If a node is  marginalized

   then its parent is  marginalized
3. A node has at most one child that is  marginalized

These imply that the marginalized nodes form a path: from the root to a tip.

> We refer to this path as the **M-path**, and the deepest node on this path as the **terminal node**.

The rules

We should not **Sample** or **Observe** just any node.

We should only **Sample** or **Observe** a **terminal** node of the **M-path**.

The rules

We should not **Sample** or **Observe** just any node.

We should only **Sample** or **Observe** a **terminal** node of the **M-path**.

To ensure this, we define two new operations:

- **Prune** to shorten the M-path by backward sampling.
- **Graft** to extend the M-path by forward marginalization.

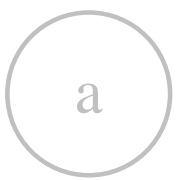
We can use these operations to turn any node into the terminal node before we **Sample** or **Observe** it.

Canonical Example Revisited

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	

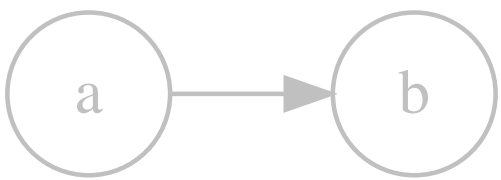
Canonical Example Revisited

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	**assume** a
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



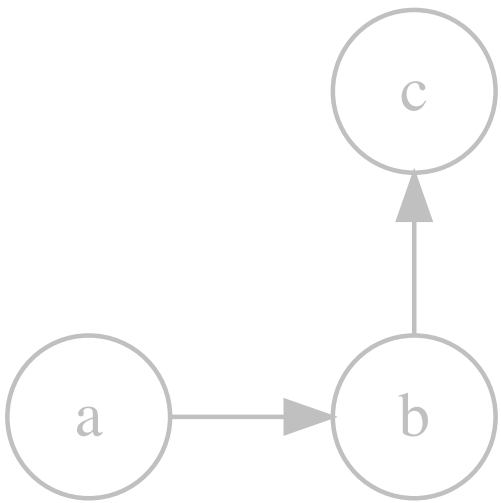
Canonical Example Revisited

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	**assume** b
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



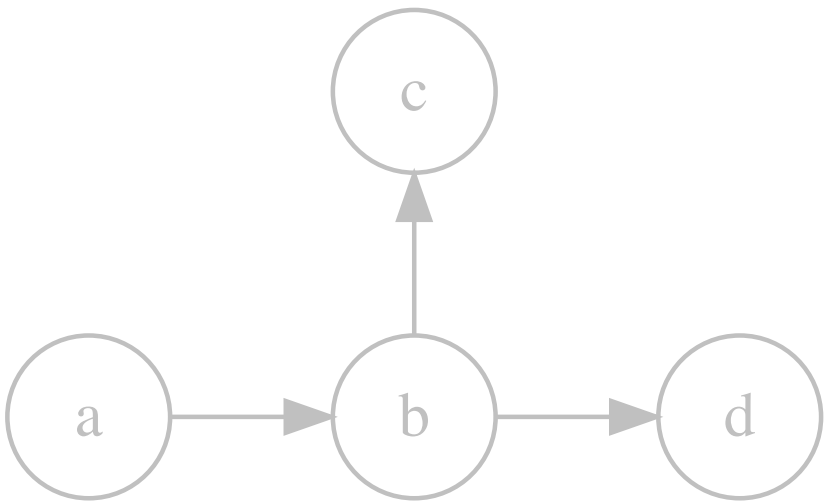
Canonical Example Revisited

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	**assume** `c`
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



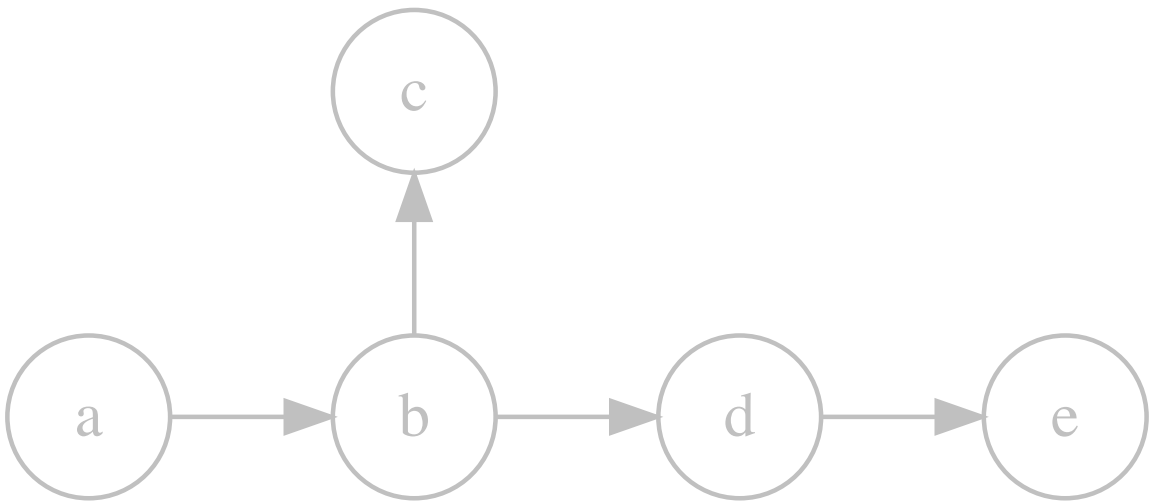
Canonical Example Revisited

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	**assume** d
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	



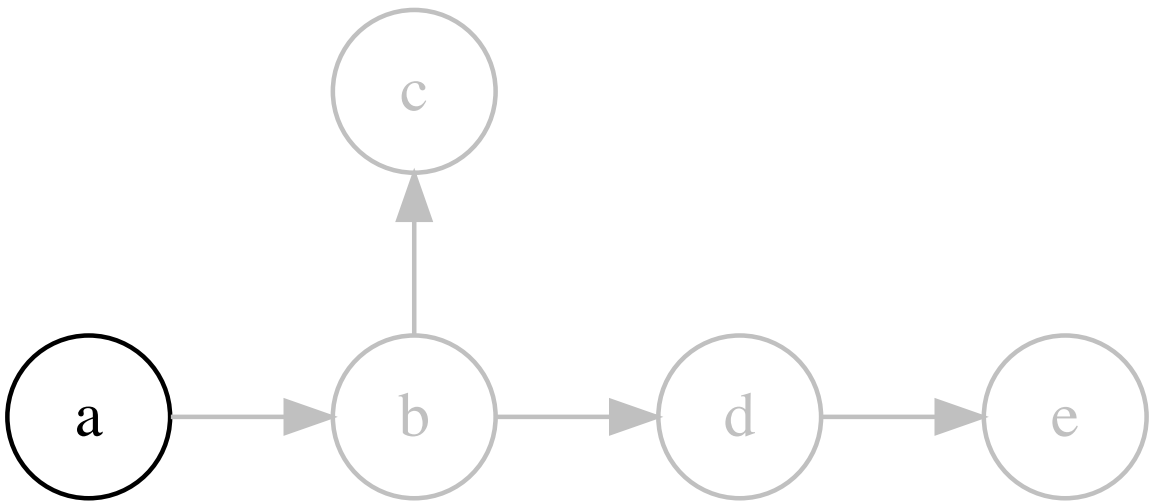
Canonical Example Revisited

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	**observe** `e`
`stdout.print(c);`	



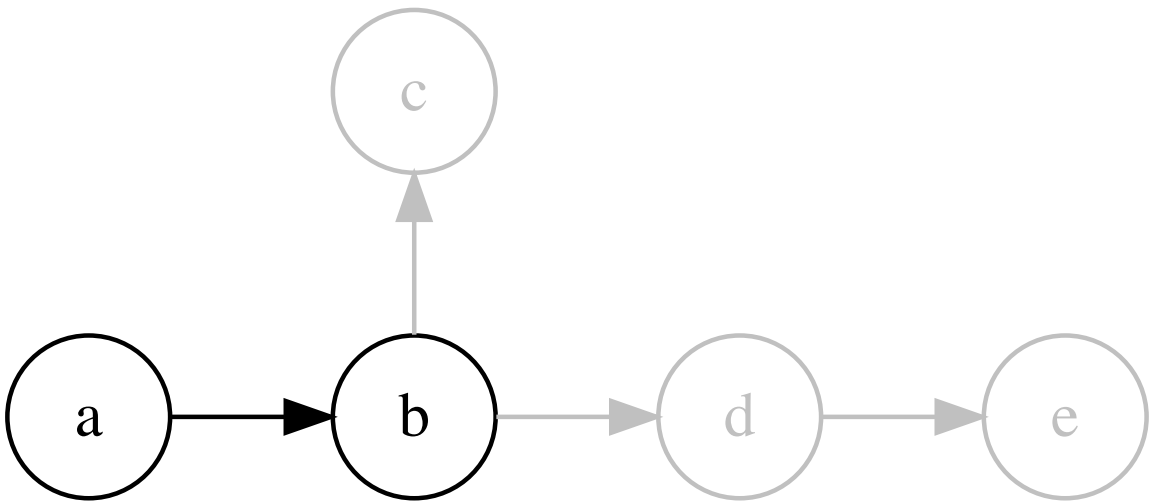
Canonical Example Revisited

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	**observe** `e`
`stdout.print(c);`	



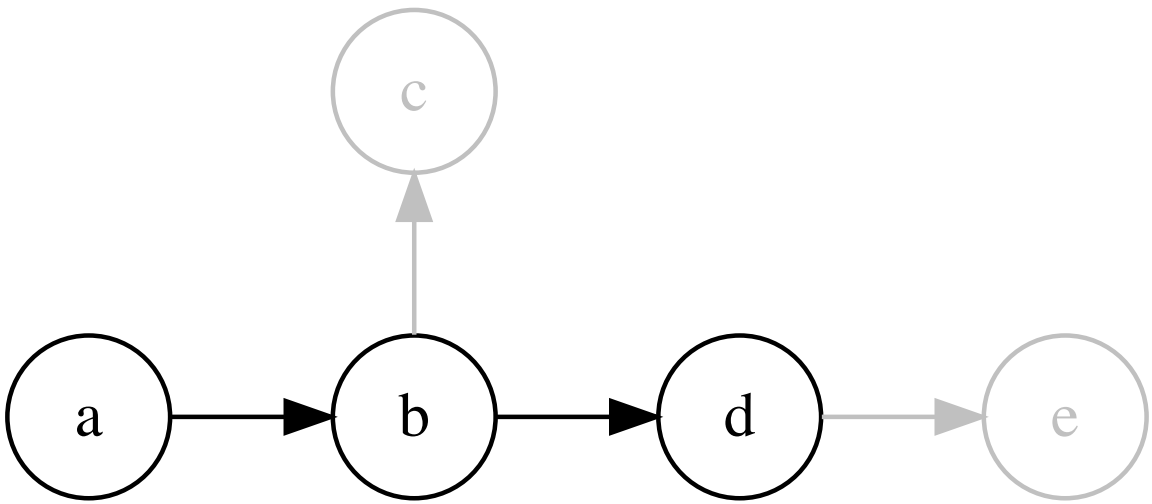
Canonical Example Revisited

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	**observe** `e`
`stdout.print(c);`	



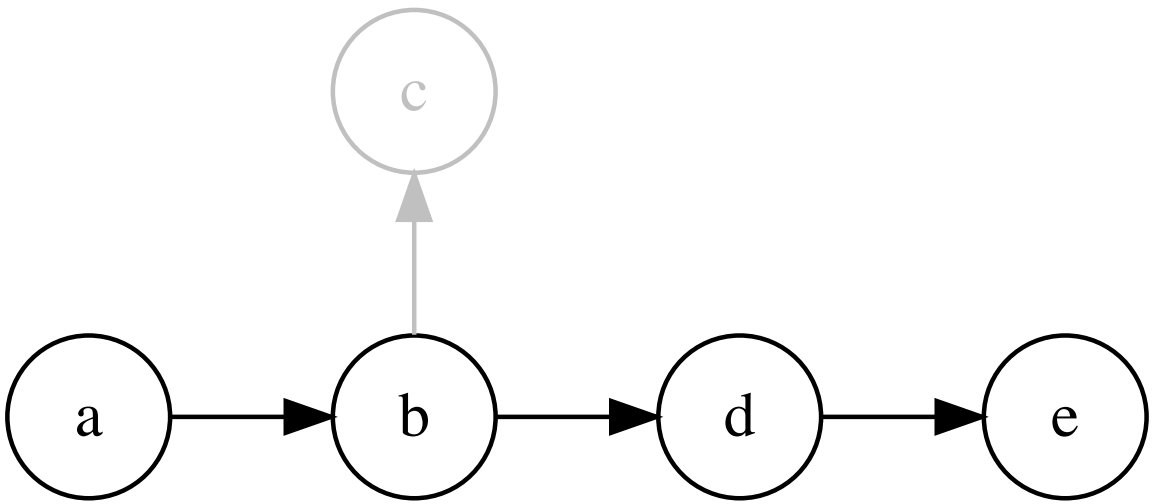
Canonical Example Revisited

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	**observe** e
`stdout.print(c);`	



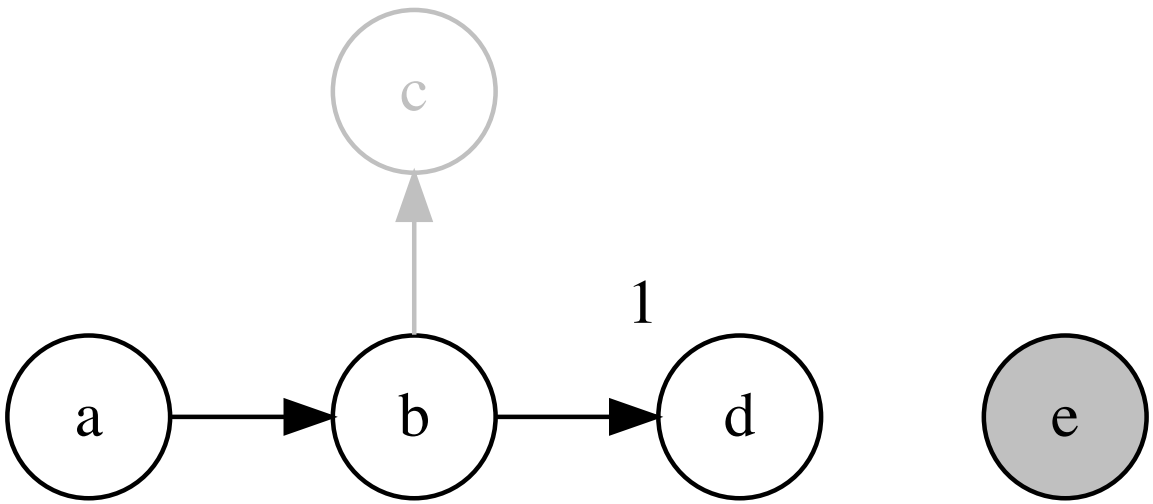
Canonical Example Revisited

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	**observe** e
`stdout.print(c);`	



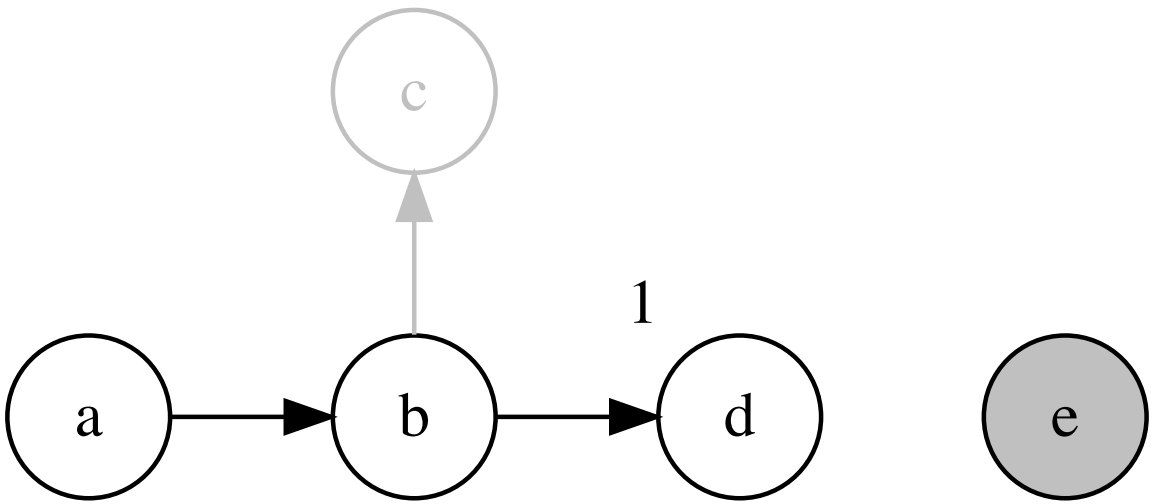
Canonical Example Revisited

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	**observe** e
`stdout.print(c);`	



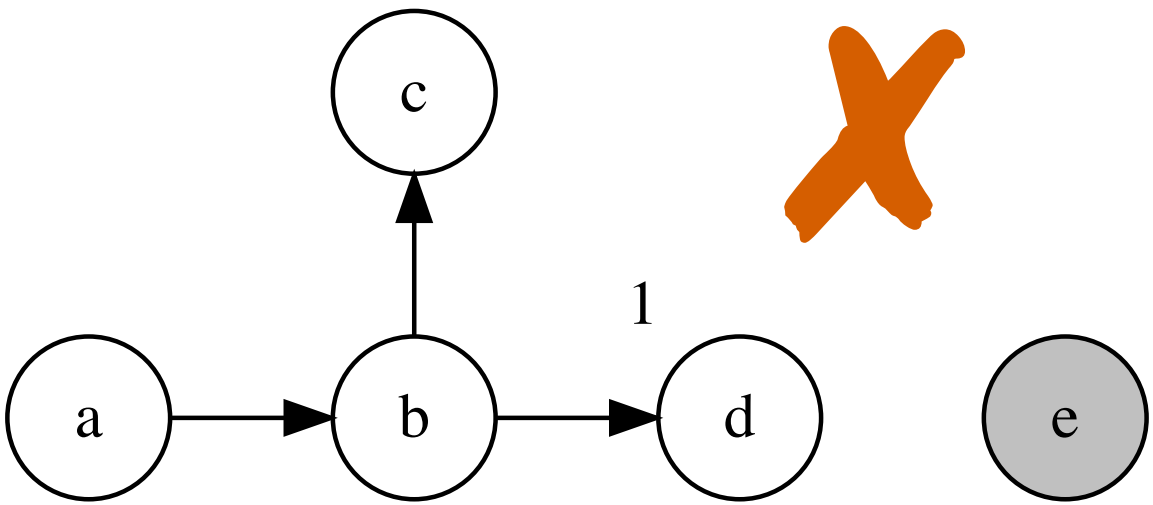
Canonical Example Revisited

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	**value** `c`



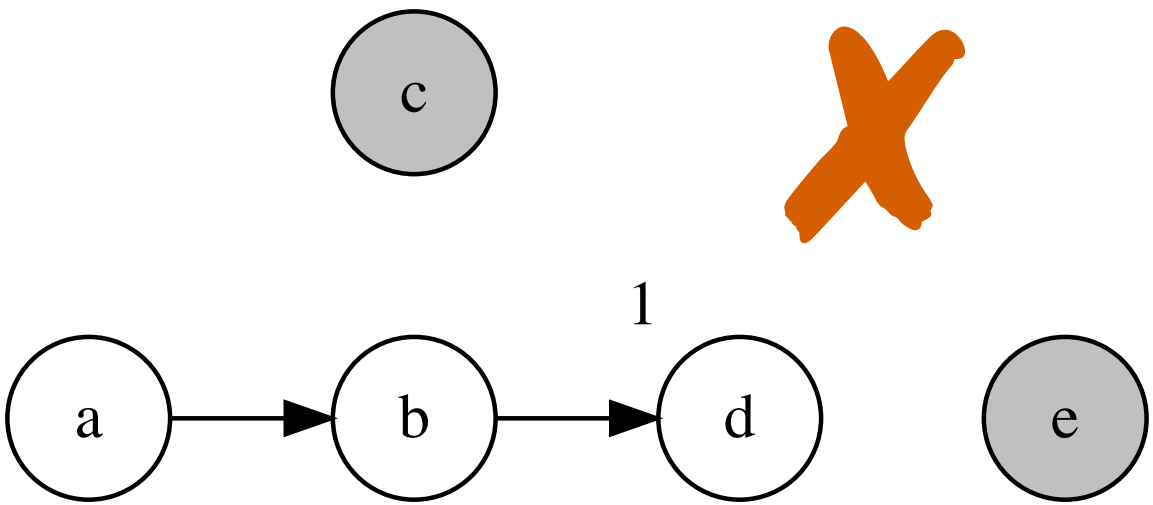
Canonical Example Revisited

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	**value** `c`



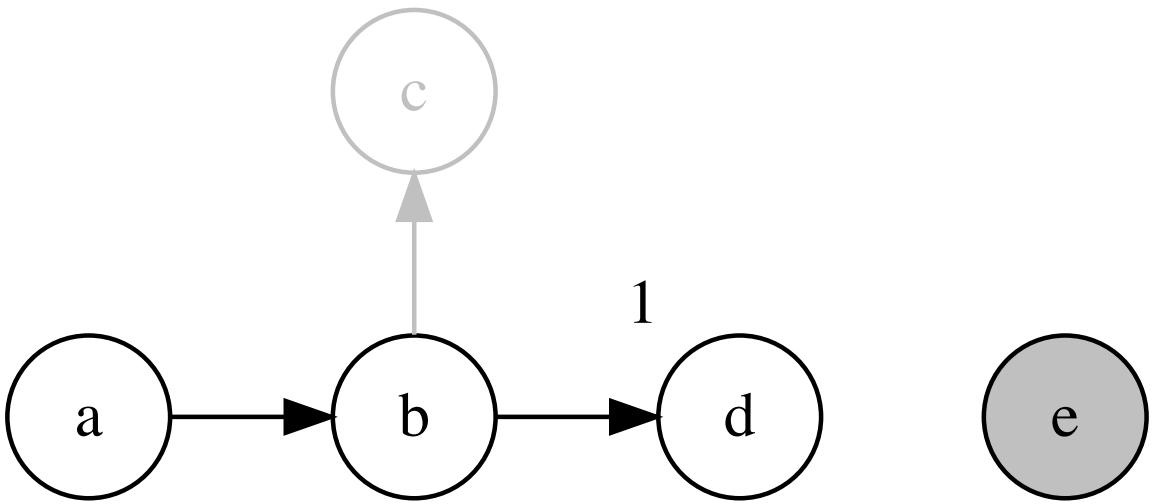
Canonical Example Revisited

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	**value** `c`



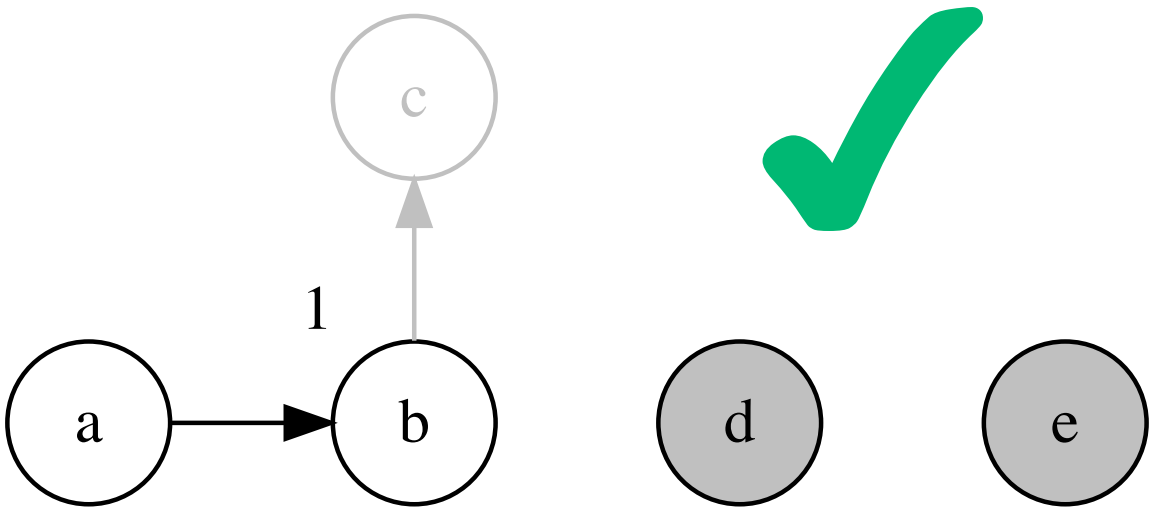
Canonical Example Revisited

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	**value** `c`



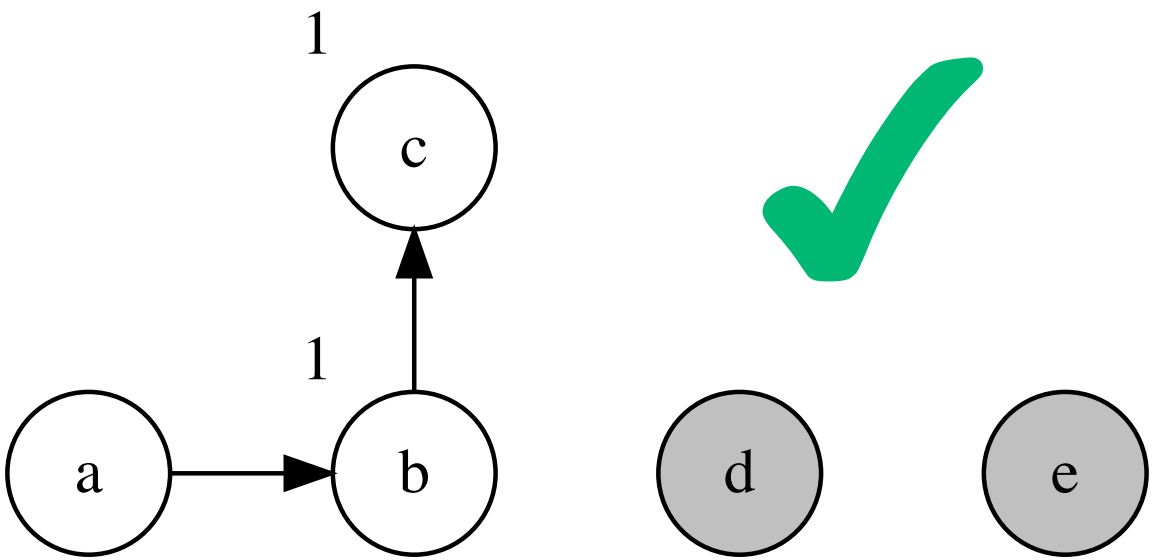
Canonical Example Revisited

Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	**value** `c`



Canonical Example Revisited

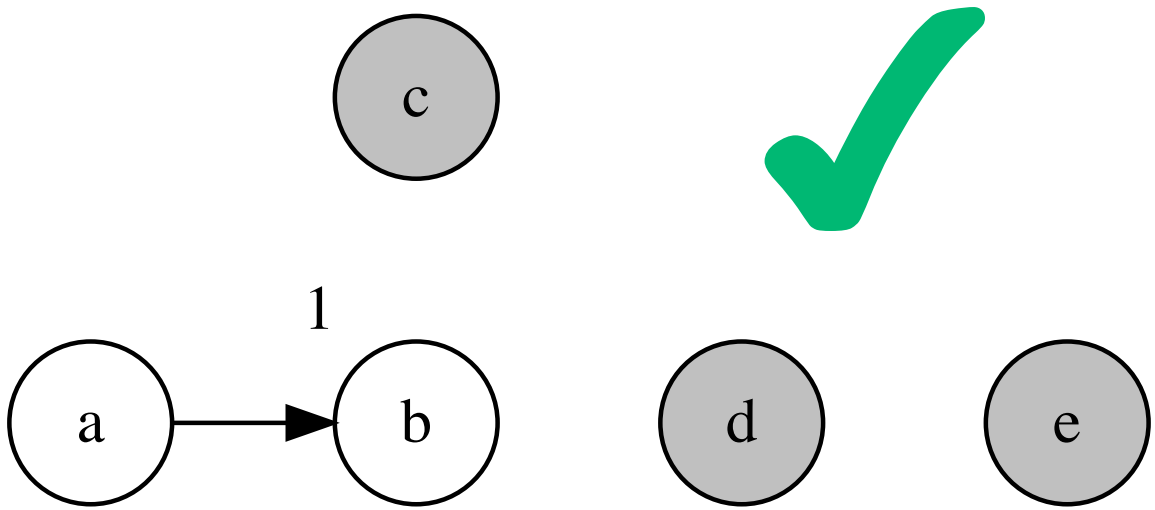
Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	
`b ~ Gaussian(a, 1.0);`	
`c ~ Gaussian(b, 1.0);`	
`d ~ Gaussian(b, 1.0);`	
`e ~ Gaussian(d, 1.0);`	
`stdout.print(c);`	**value** `c`



Canonical Example Revisited

Code	Checkpoint

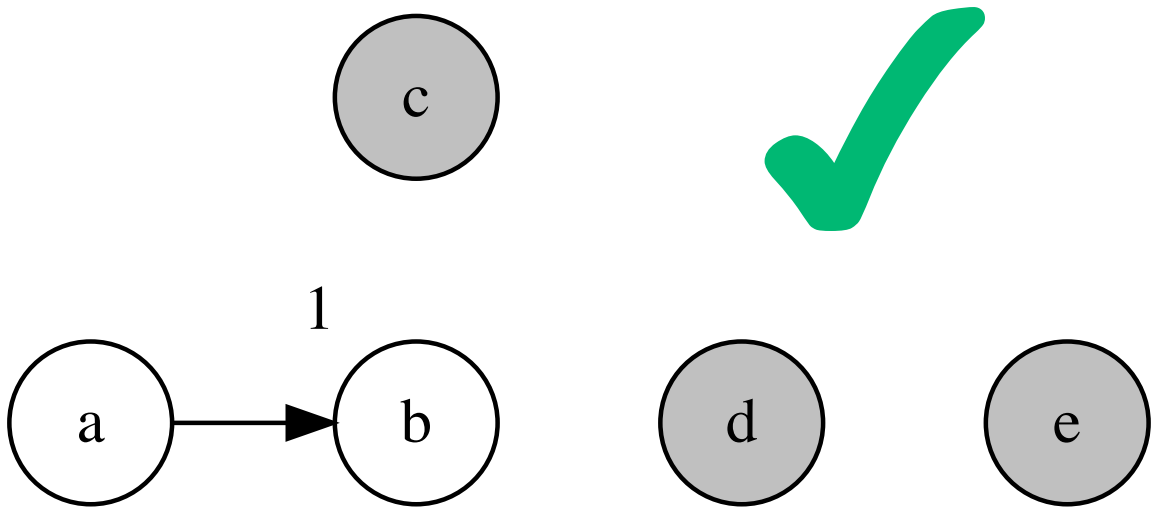
```
a ~ Gaussian(0.0, 1.0);
b ~ Gaussian(a, 1.0);
c ~ Gaussian(b, 1.0);
d ~ Gaussian(b, 1.0);
e ~ Gaussian(d, 1.0);
stdout.print(c);
```



Canonical Example Revisited

Code	Checkpoint

```
a ~ Gaussian(0.0, 1.0);
b ~ Gaussian(a, 1.0);
c ~ Gaussian(b, 1.0);
d ~ Gaussian(b, 1.0);
e ~ Gaussian(d, 1.0);
stdout.print(c);
```



Implementation

- Delayed sampling has been implemented in Anglican and Birch.
- Birch is an imperative, object-oriented, universal probabilistic programming language.
- It compiles to C++14.
- It is free and open source.
- See birch-lang.org.

Example #1

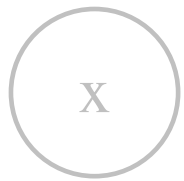
Code	Checkpoint

```
x ~ Gaussian(0.0, 1.0);
for (n in 1..N) {
  y[n] ~ Gaussian(x, 1.0);
}
stdout.print(x);
```

Example #1

Code	Checkpoint

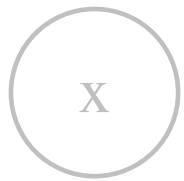
`x ~ Gaussian(0.0, 1.0);`
`for (n in 1..N) {`
`  y[n] ~ Gaussian(x, 1.0);`
`}`
`stdout.print(x);`
	assume x



Example #1

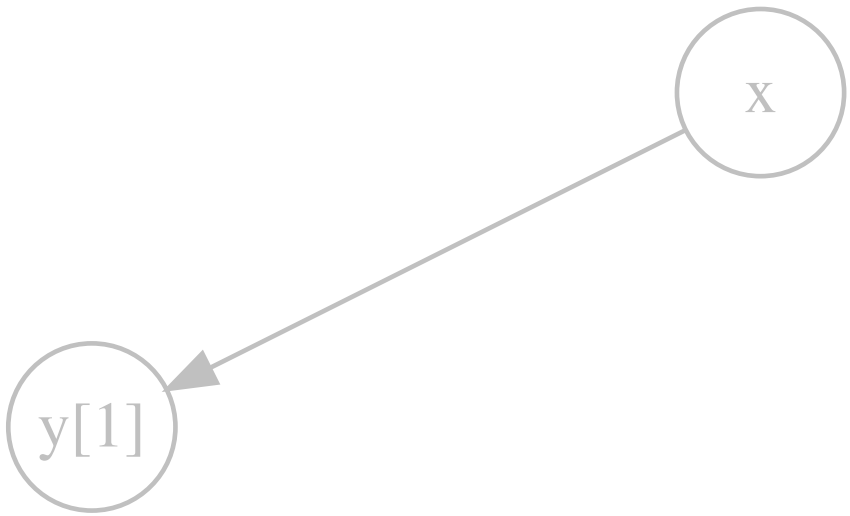
Code	Checkpoint

```
x ~ Gaussian(0.0, 1.0);
for (n in 1..N) {
  y[n] ~ Gaussian(x, 1.0);
}
stdout.print(x);
```



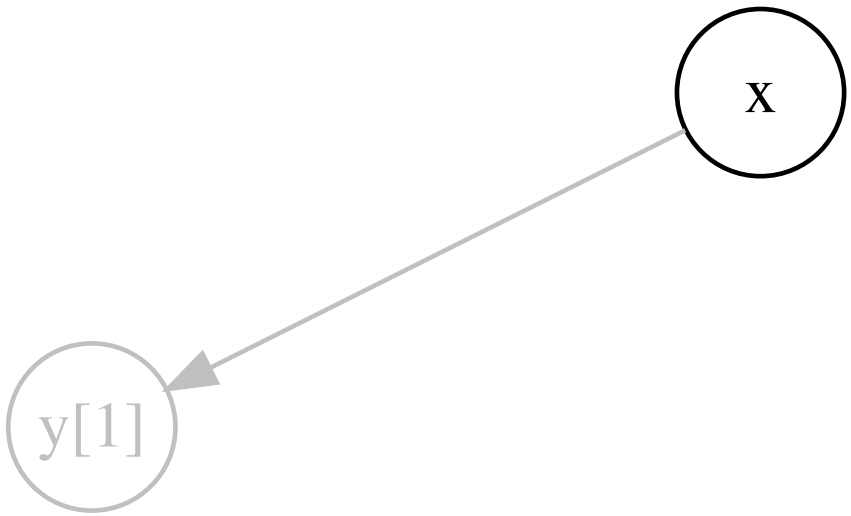
Example #1

Code	Checkpoint
`x ~ Gaussian(0.0, 1.0);`	
`for (n in 1..N) {`	
`  y[n] ~ Gaussian(x, 1.0);`	**observe** y[n]
`}`	
`stdout.print(x);`	



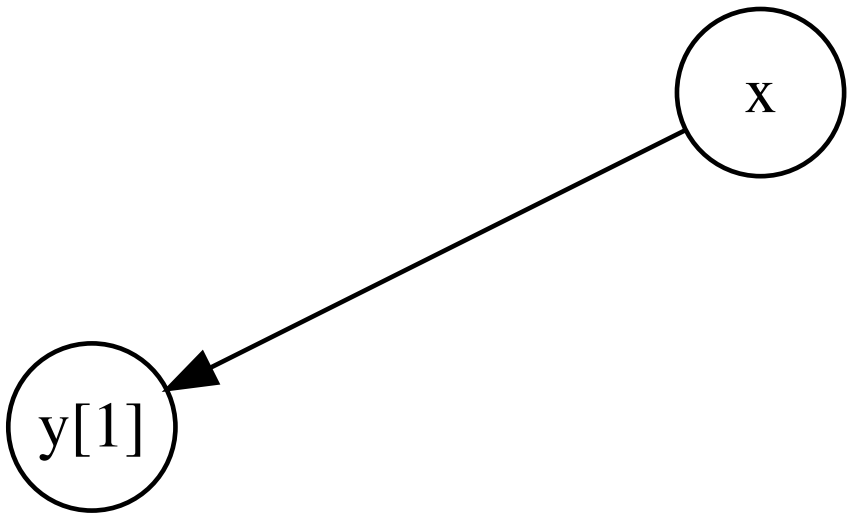
Example #1

Code	Checkpoint
`x ~ Gaussian(0.0, 1.0);`	
`for (n in 1..N) {`	
`  y[n] ~ Gaussian(x, 1.0);`	**observe** y[n]
`}`	
`stdout.print(x);`	



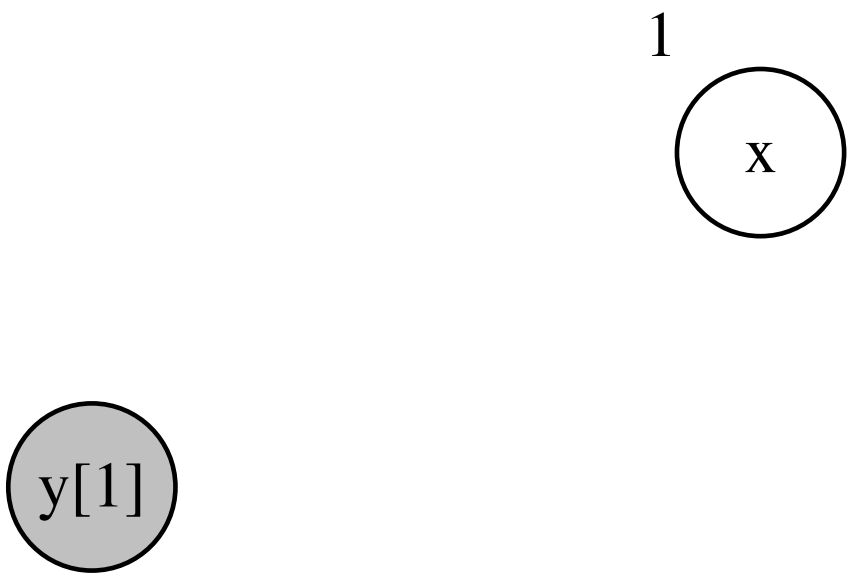
Example #1

Code	Checkpoint
`x ~ Gaussian(0.0, 1.0);`	
`for (n in 1..N) {`	
`  y[n] ~ Gaussian(x, 1.0);`	**observe** y[n]
`}`	
`stdout.print(x);`	



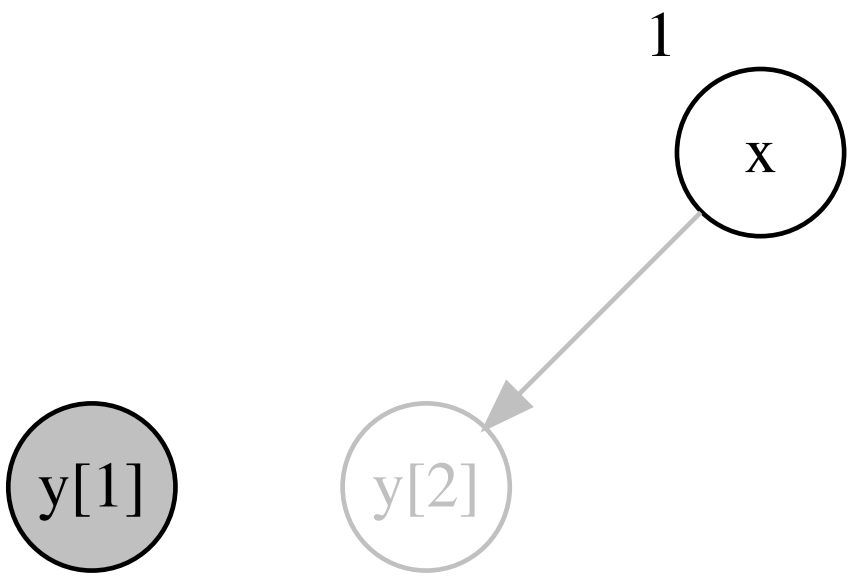
Example #1

Code	Checkpoint
`x ~ Gaussian(0.0, 1.0);`	
`for (n in 1..N) {`	
`  y[n] ~ Gaussian(x, 1.0);`	**observe** y[n]
`}`	
`stdout.print(x);`	



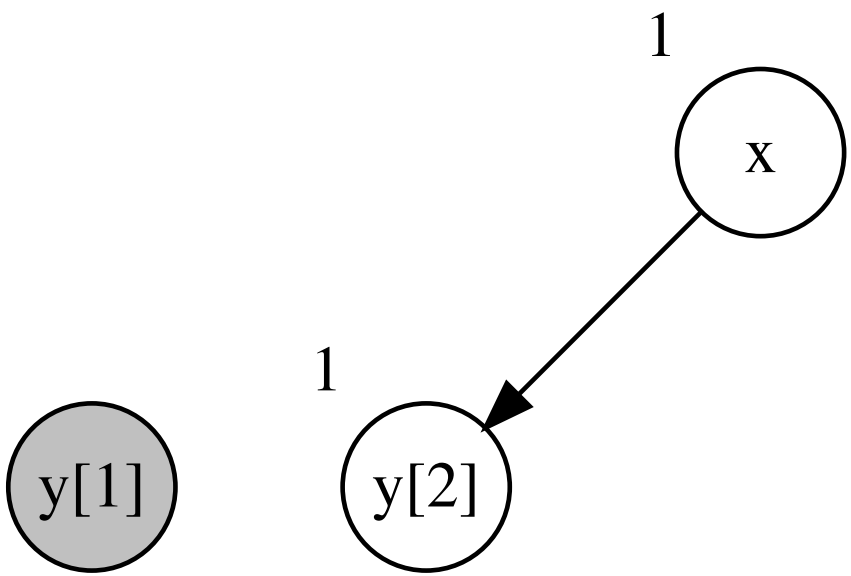
Example #1

Code	Checkpoint
`x ~ Gaussian(0.0, 1.0);`	
`for (n in 1..N) {`	
`  y[n] ~ Gaussian(x, 1.0);`	**observe** y[n]
`}`	
`stdout.print(x);`	



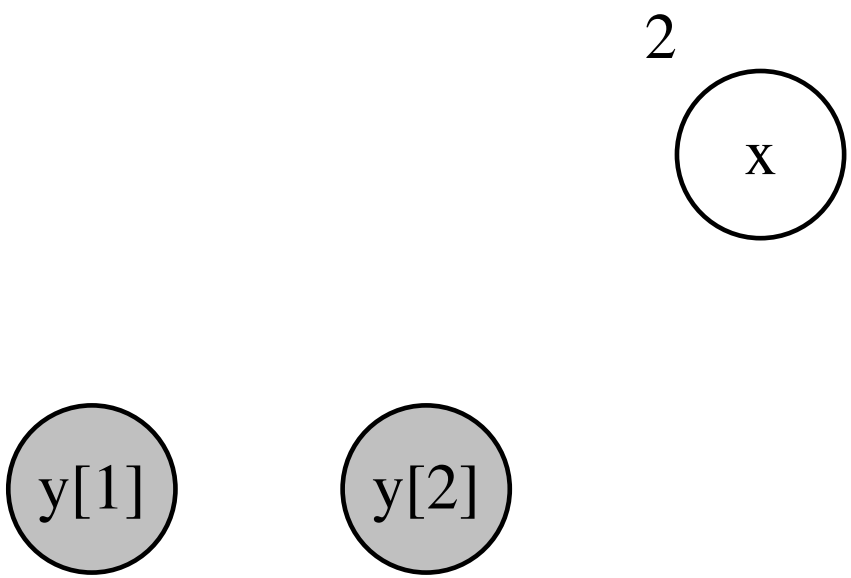
Example #1

Code	Checkpoint
`x ~ Gaussian(0.0, 1.0);`	
`for (n in 1..N) {`	
`  y[n] ~ Gaussian(x, 1.0);`	**observe** y[n]
`}`	
`stdout.print(x);`	



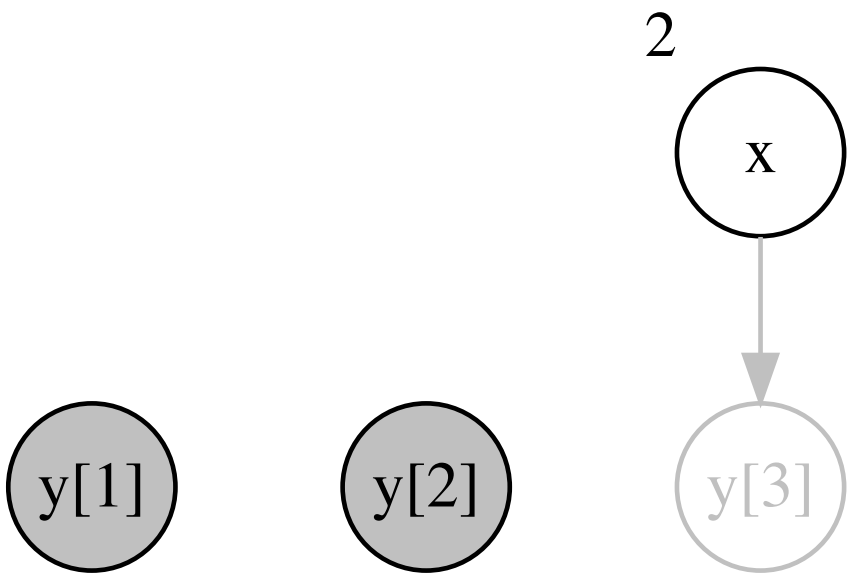
Example #1

Code	Checkpoint
`x ~ Gaussian(0.0, 1.0);`	
`for (n in 1..N) {`	
`  y[n] ~ Gaussian(x, 1.0);`	**observe** y[n]
`}`	
`stdout.print(x);`	



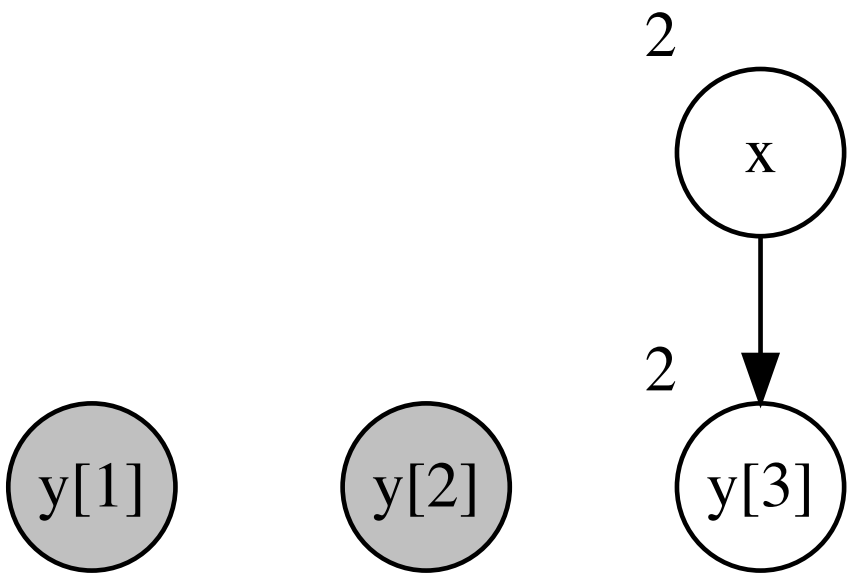
Example #1

Code	Checkpoint
`x ~ Gaussian(0.0, 1.0);`	
`for (n in 1..N) {`	
`  y[n] ~ Gaussian(x, 1.0);`	**observe** y[n]
`}`	
`stdout.print(x);`	



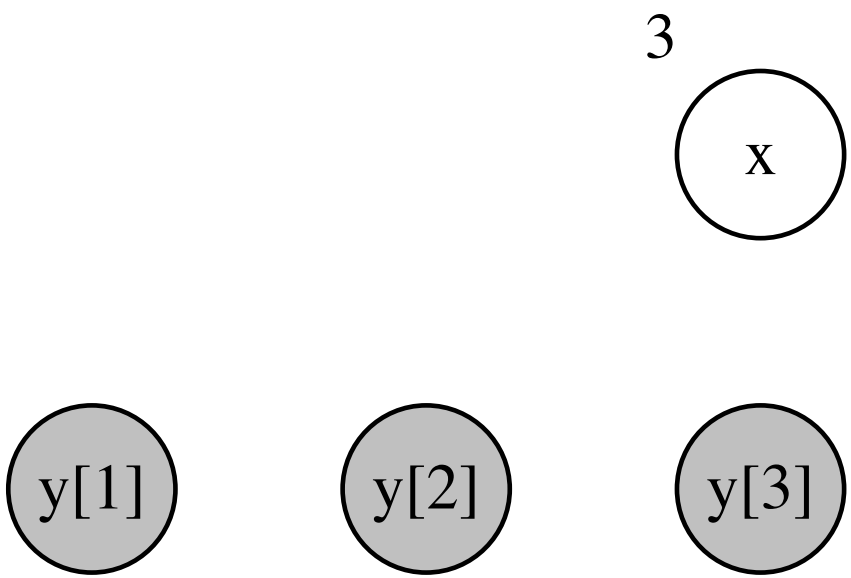
Example #1

Code	Checkpoint
`x ~ Gaussian(0.0, 1.0);`	
`for (n in 1..N) {`	
`  y[n] ~ Gaussian(x, 1.0);`	**observe** y[n]
`}`	
`stdout.print(x);`	



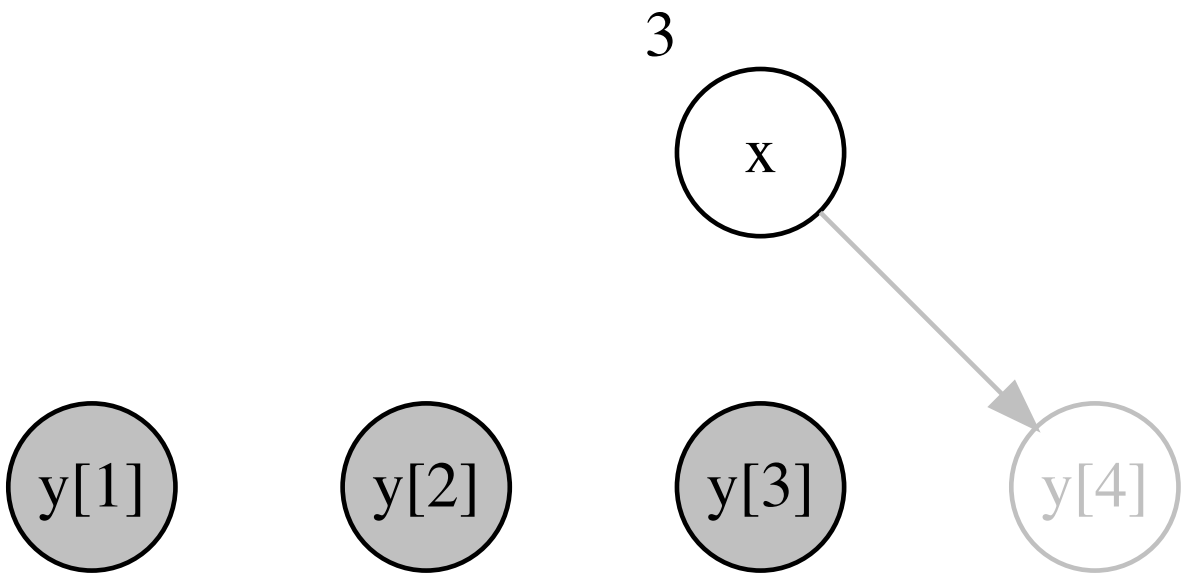
Example #1

Code	Checkpoint
`x ~ Gaussian(0.0, 1.0);`	
`for (n in 1..N) {`	
`  y[n] ~ Gaussian(x, 1.0);`	**observe** y[n]
`}`	
`stdout.print(x);`	



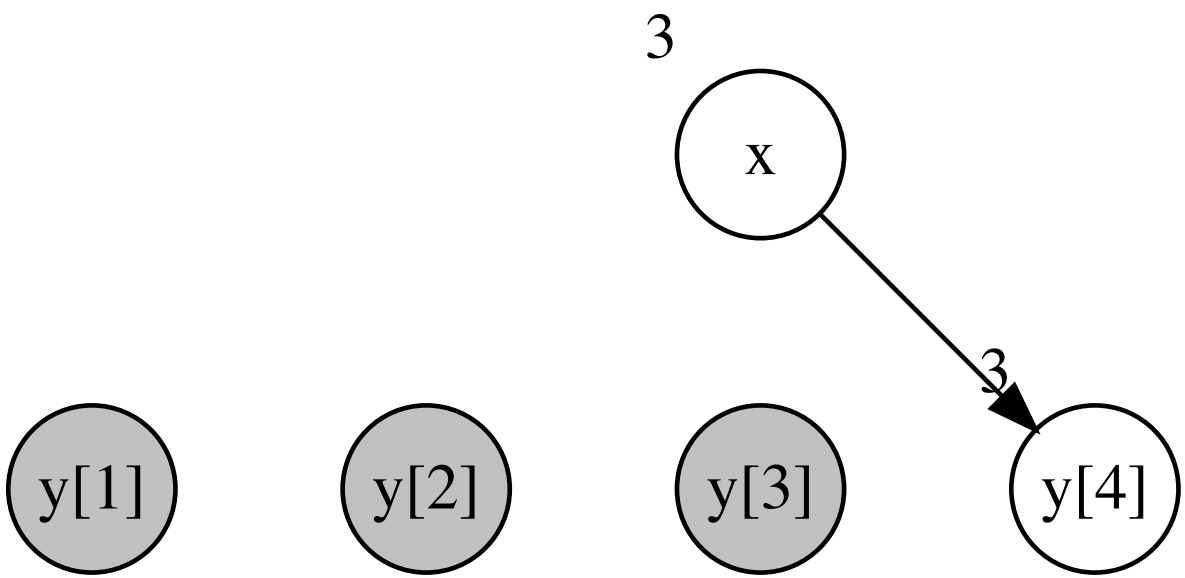
Example #1

Code	Checkpoint
`x ~ Gaussian(0.0, 1.0);`	
`for (n in 1..N) {`	
`  y[n] ~ Gaussian(x, 1.0);`	**observe** y[n]
`}`	
`stdout.print(x);`	



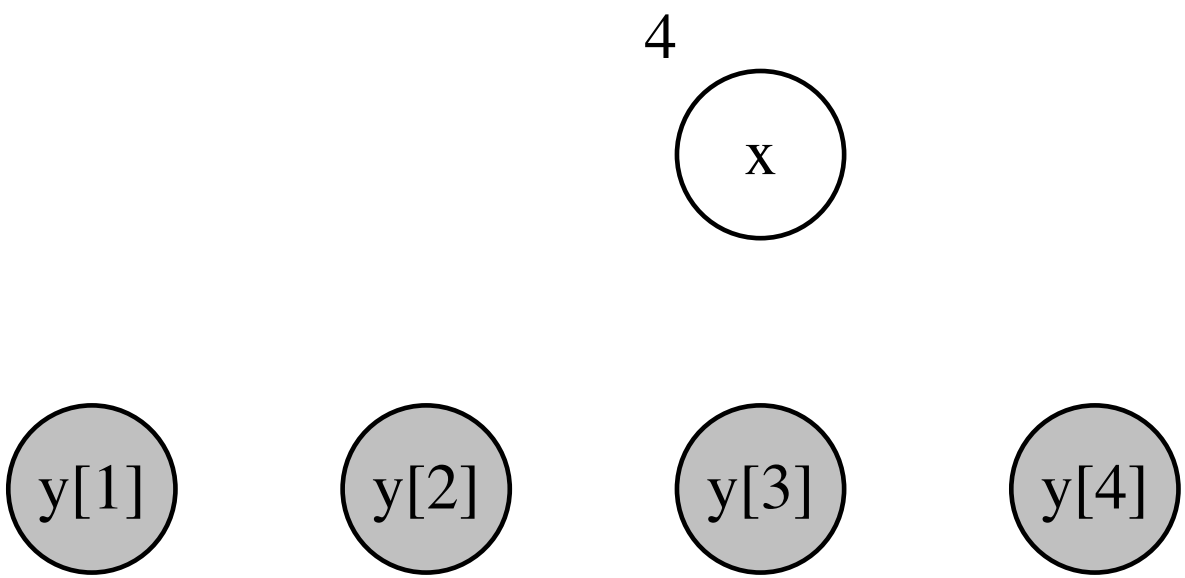
Example #1

Code	Checkpoint
`x ~ Gaussian(0.0, 1.0);`	
`for (n in 1..N) {`	
`  y[n] ~ Gaussian(x, 1.0);`	**observe** y[n]
`}`	
`stdout.print(x);`	



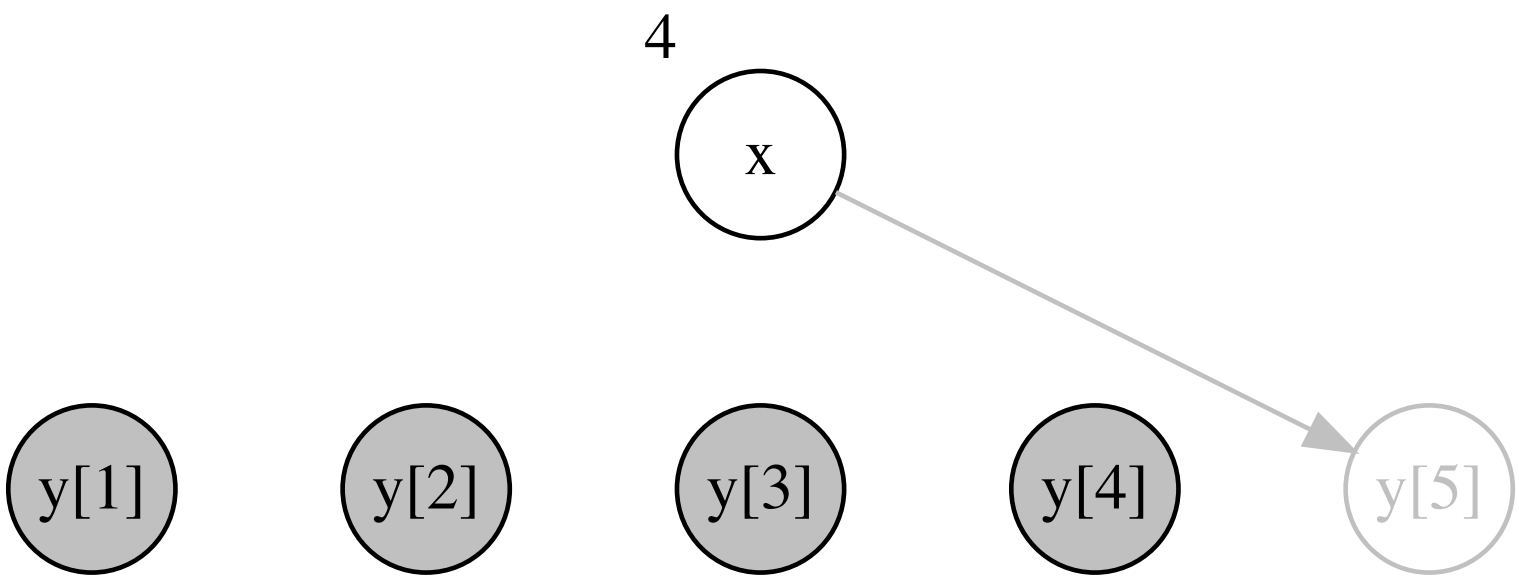
Example #1

Code	Checkpoint
`x ~ Gaussian(0.0, 1.0);`	
`for (n in 1..N) {`	
`  y[n] ~ Gaussian(x, 1.0);`	**observe** y[n]
`}`	
`stdout.print(x);`	



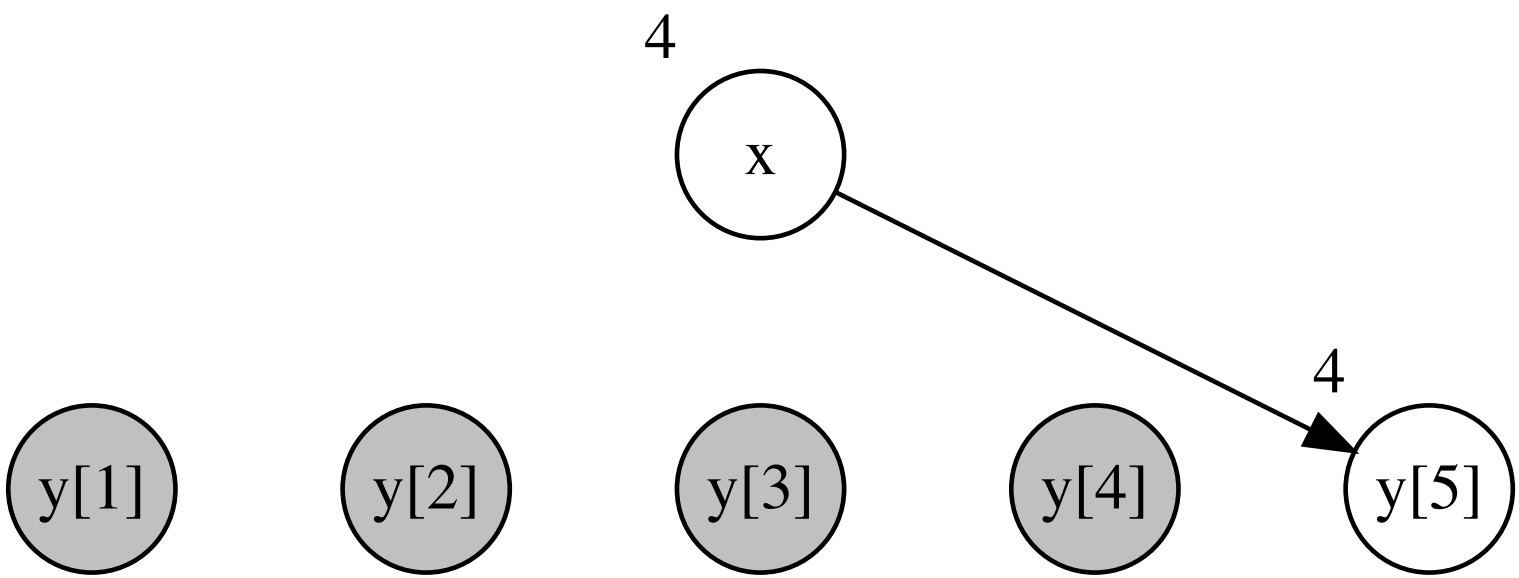
Example #1

Code	Checkpoint
`x ~ Gaussian(0.0, 1.0);`	
`for (n in 1..N) {`	
`  y[n] ~ Gaussian(x, 1.0);`	**observe** y[n]
`}`	
`stdout.print(x);`	



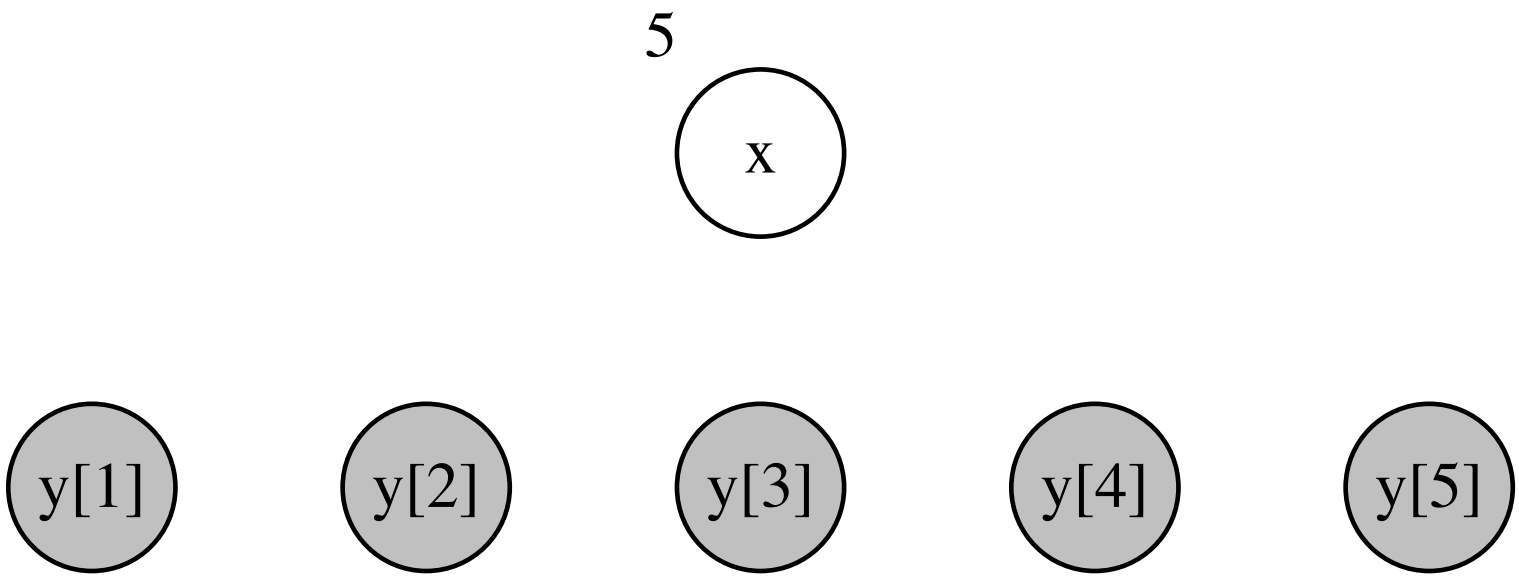
Example #1

Code	Checkpoint
`x ~ Gaussian(0.0, 1.0);`	
`for (n in 1..N) {`	
`  y[n] ~ Gaussian(x, 1.0);`	**observe** y[n]
`}`	
`stdout.print(x);`	



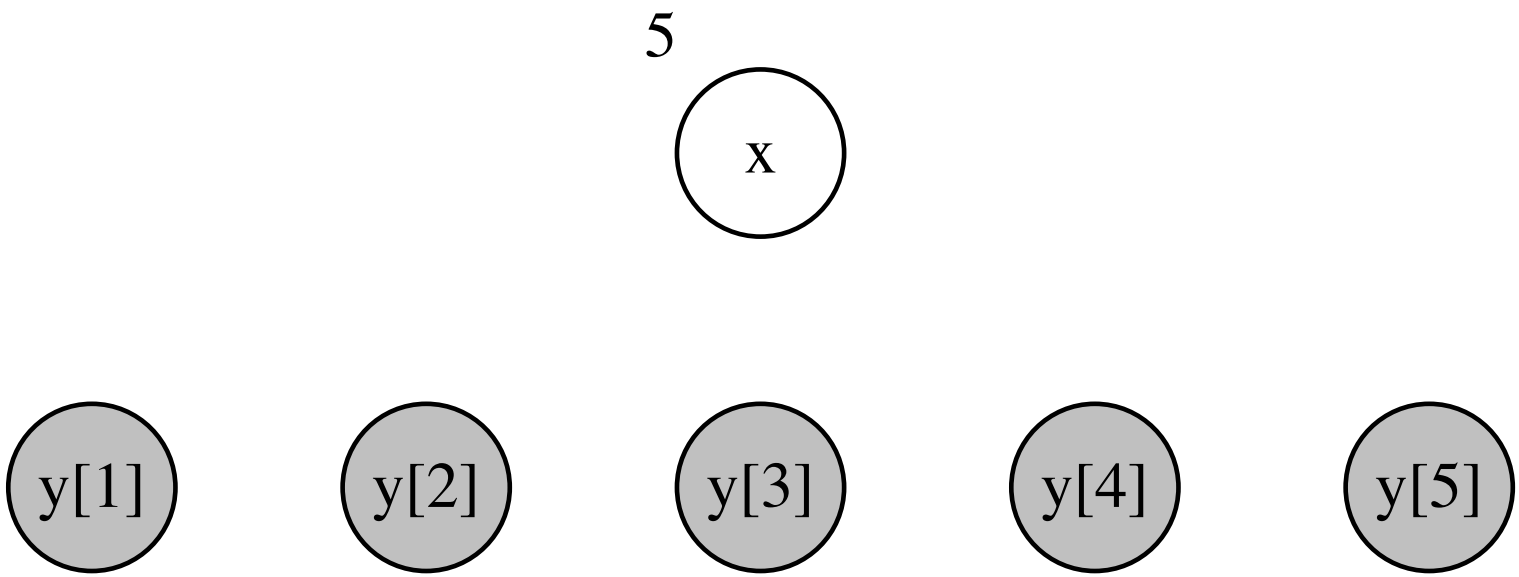
Example #1

Code	Checkpoint
`x ~ Gaussian(0.0, 1.0);`	
`for (n in 1..N) {`	
`  y[n] ~ Gaussian(x, 1.0);`	**observe** y[n]
`}`	
`stdout.print(x);`	



Example #1

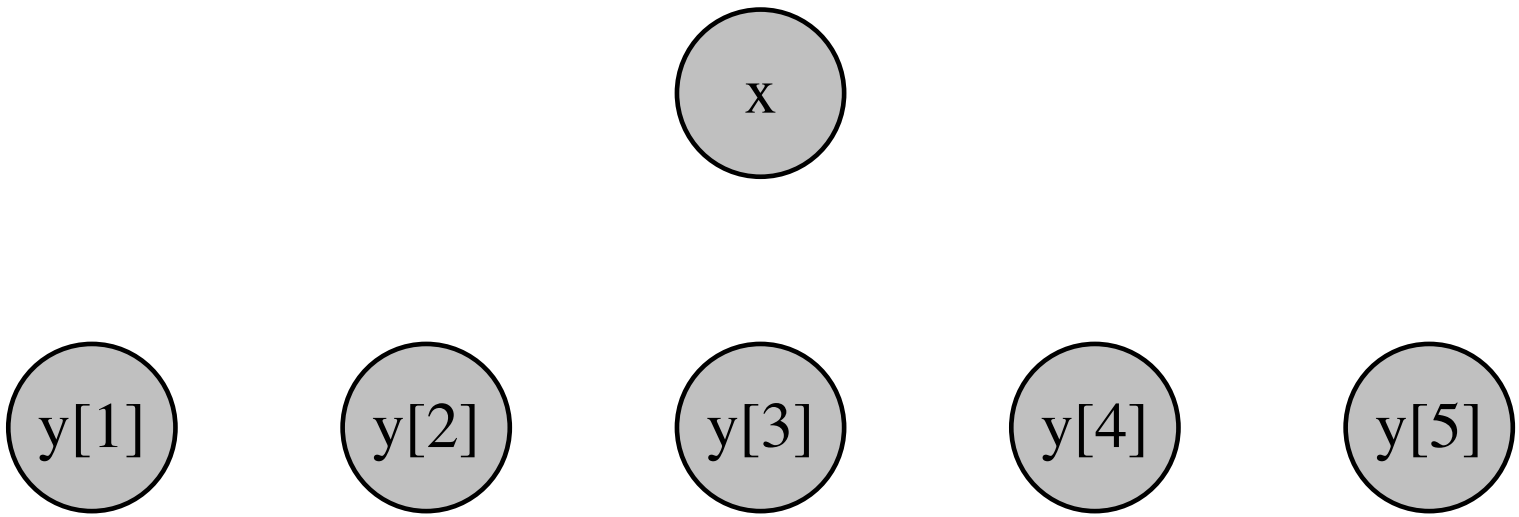
Code	Checkpoint
`x ~ Gaussian(0.0, 1.0);`	
`for (n in 1..N) {`	
`  y[n] ~ Gaussian(x, 1.0);`	
`}`	
`stdout.print(x);`	**value** x



Example #1

Code	Checkpoint

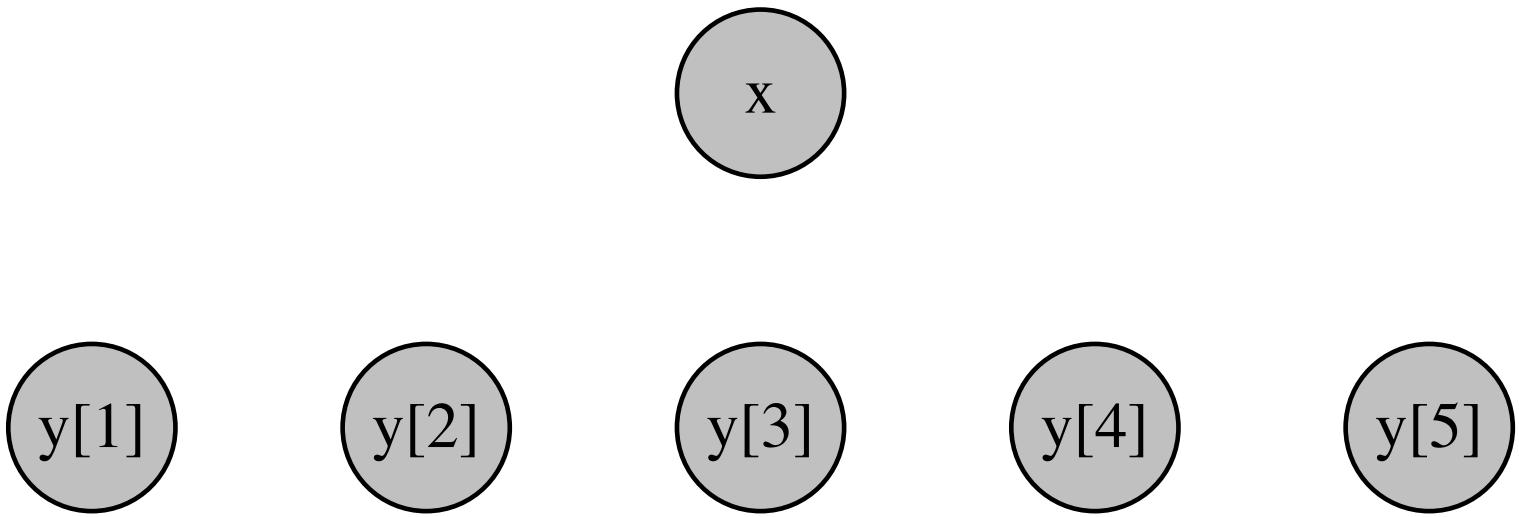
```
x ~ Gaussian(0.0, 1.0);
for (n in 1..N) {
  y[n] ~ Gaussian(x, 1.0);
}
stdout.print(x);
```



Example #1

Code	Checkpoint

```
x ~ Gaussian(0.0, 1.0);
for (n in 1..N) {
  y[n] ~ Gaussian(x, 1.0);
}
stdout.print(x);
```



Example #2

Code	Checkpoint

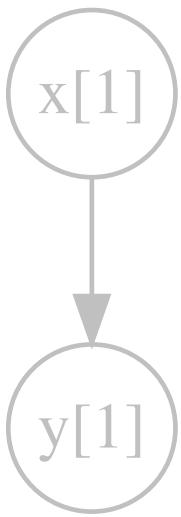
```
x[1] ~ Gaussian(0.0, 1.0);
y[1] ~ Gaussian(x[1], 1.0);
for (t in 2..T) {
  x[t] ~ Gaussian(a*x[t - 1], 1.0);
  y[t] ~ Gaussian(x[t], 1.0);
}
stdout.print(x[1]);
```

Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	**assume** x[1]
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	
`}`	
`stdout.print(x[1]);`	

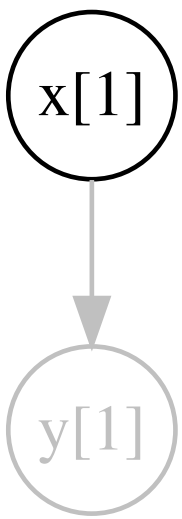
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	**observe** y[1]
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	
`}`	
`stdout.print(x[1]);`	



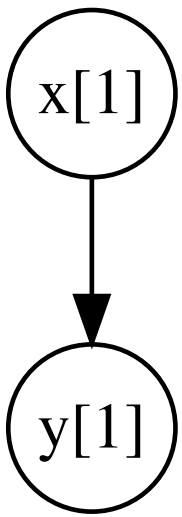
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	**observe** y[1]
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	
`}`	
`stdout.print(x[1]);`	



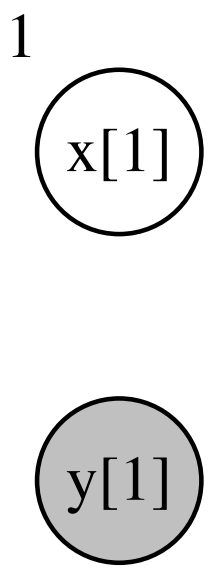
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	**observe** y[1]
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	
`}`	
`stdout.print(x[1]);`	



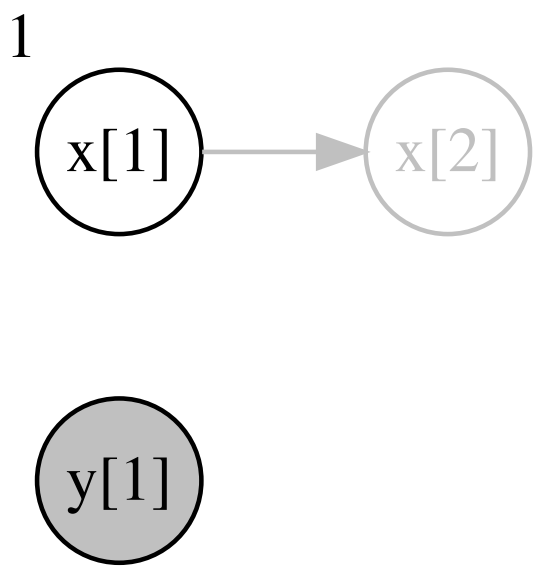
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	**observe** y[1]
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	
`}`	
`stdout.print(x[1]);`	



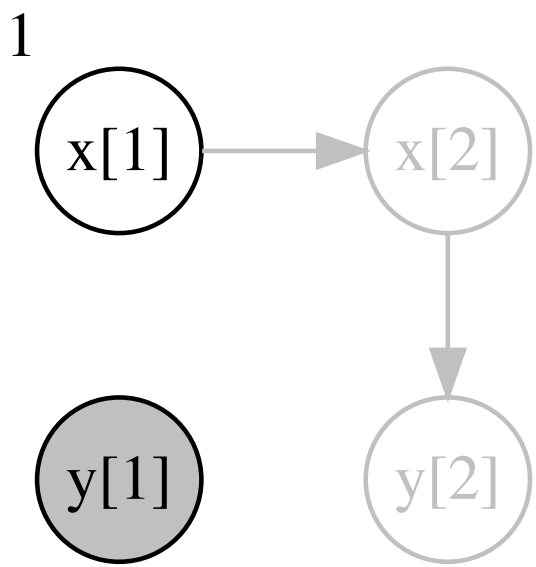
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	**assume** x[t]
`  y[t] ~ Gaussian(x[t], 1.0);`	
`}`	
`stdout.print(x[1]);`	



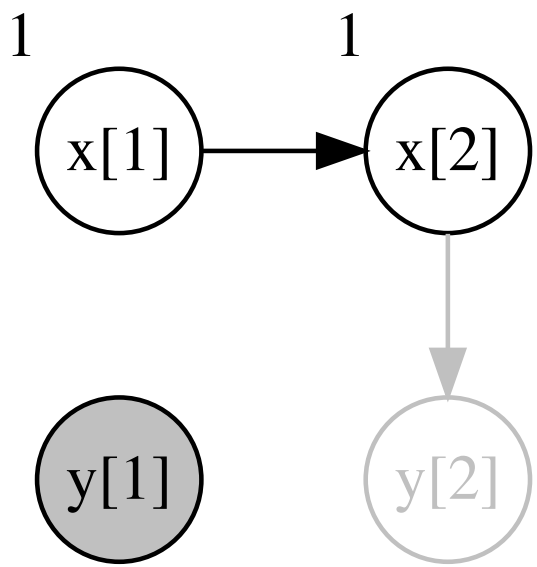
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	**observe** y[t]
`}`	
`stdout.print(x[1]);`	



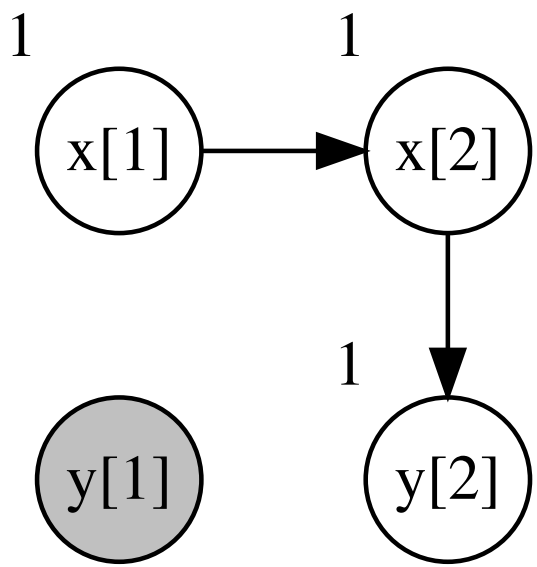
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	**observe** y[t]
`}`	
`stdout.print(x[1]);`	



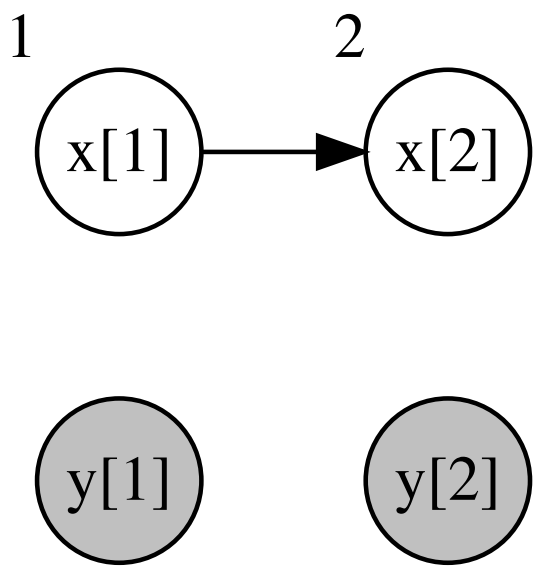
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	**observe** y[t]
`}`	
`stdout.print(x[1]);`	



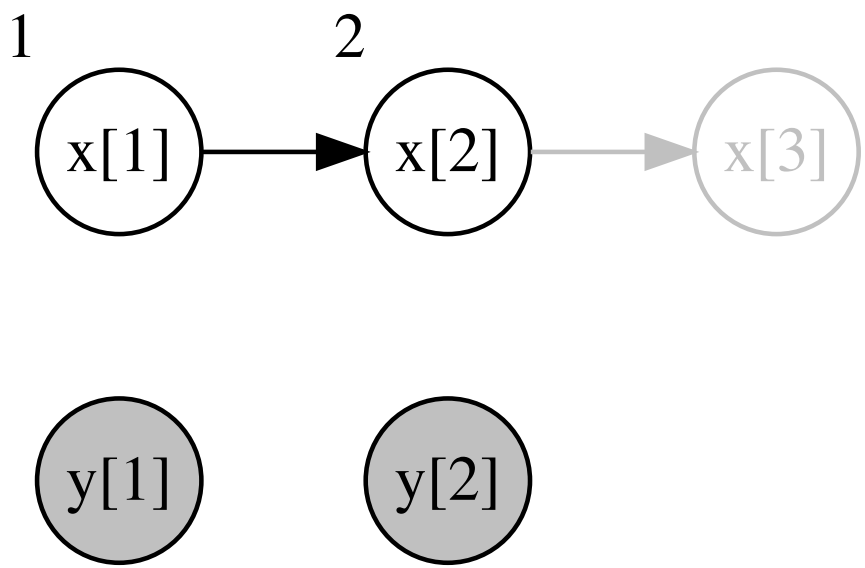
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	**observe** y[t]
`}`	
`stdout.print(x[1]);`	



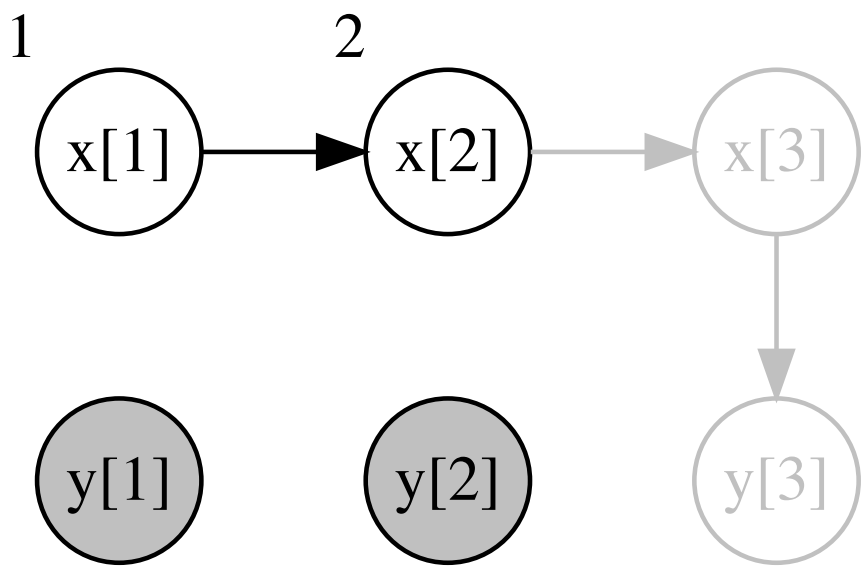
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	**assume** x[t]
`  y[t] ~ Gaussian(x[t], 1.0);`	
`}`	
`stdout.print(x[1]);`	



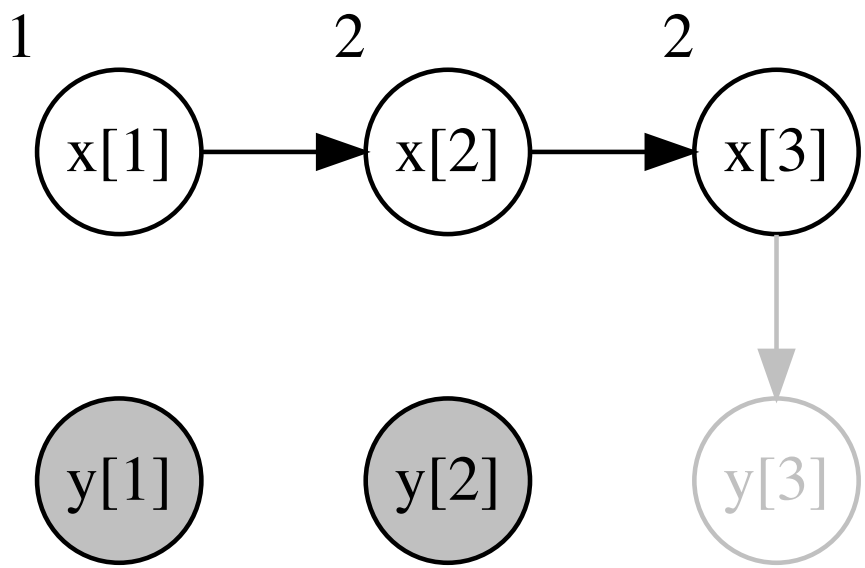
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	**observe** y[t]
`}`	
`stdout.print(x[1]);`	



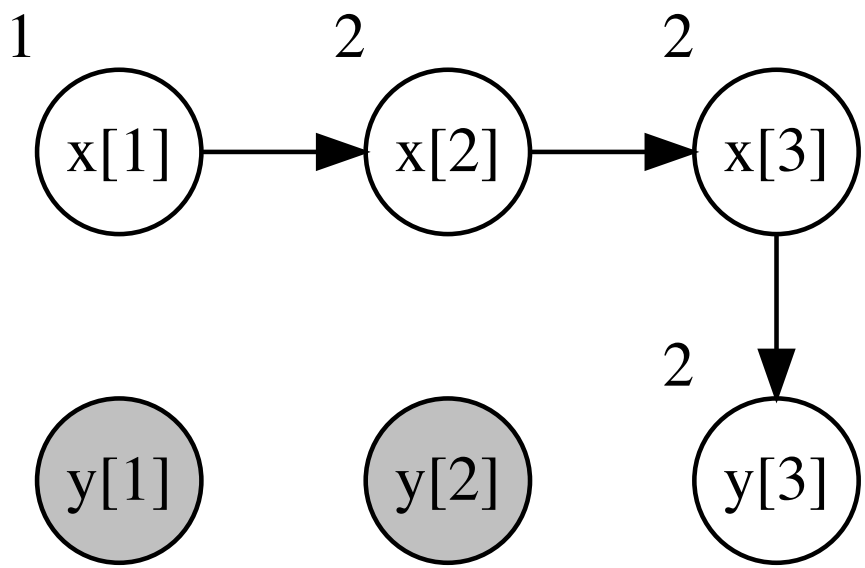
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	**observe** y[t]
`}`	
`stdout.print(x[1]);`	



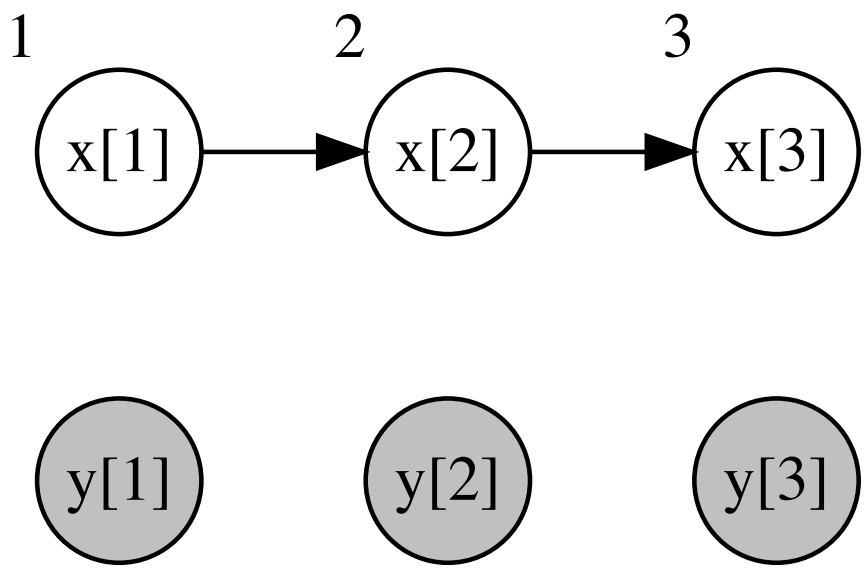
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	**observe** y[t]
`}`	
`stdout.print(x[1]);`	



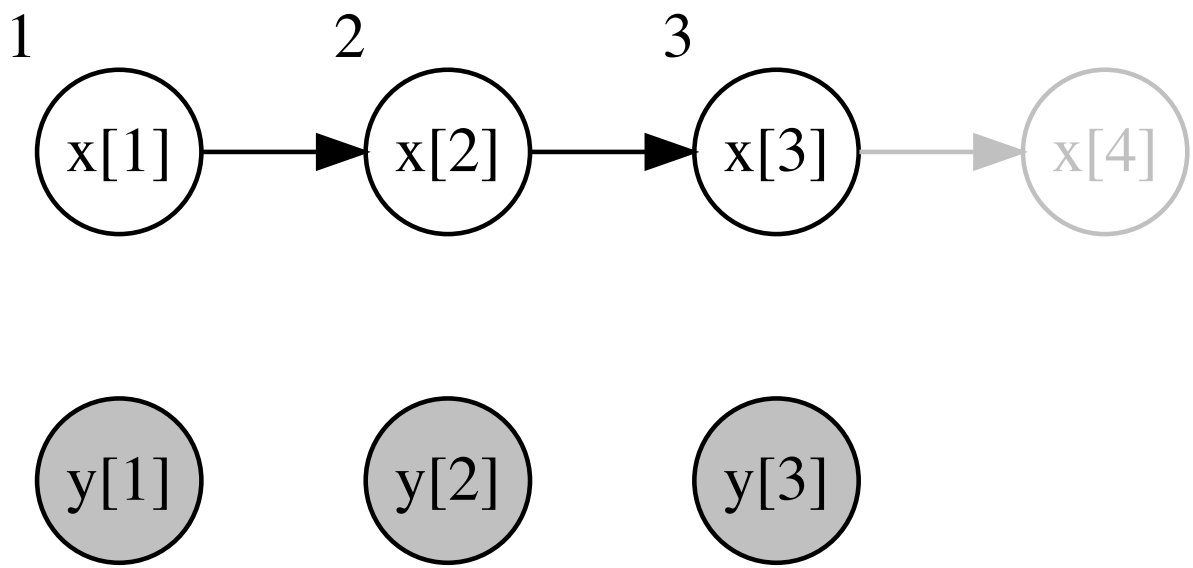
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	**observe** y[t]
`}`	
`stdout.print(x[1]);`	



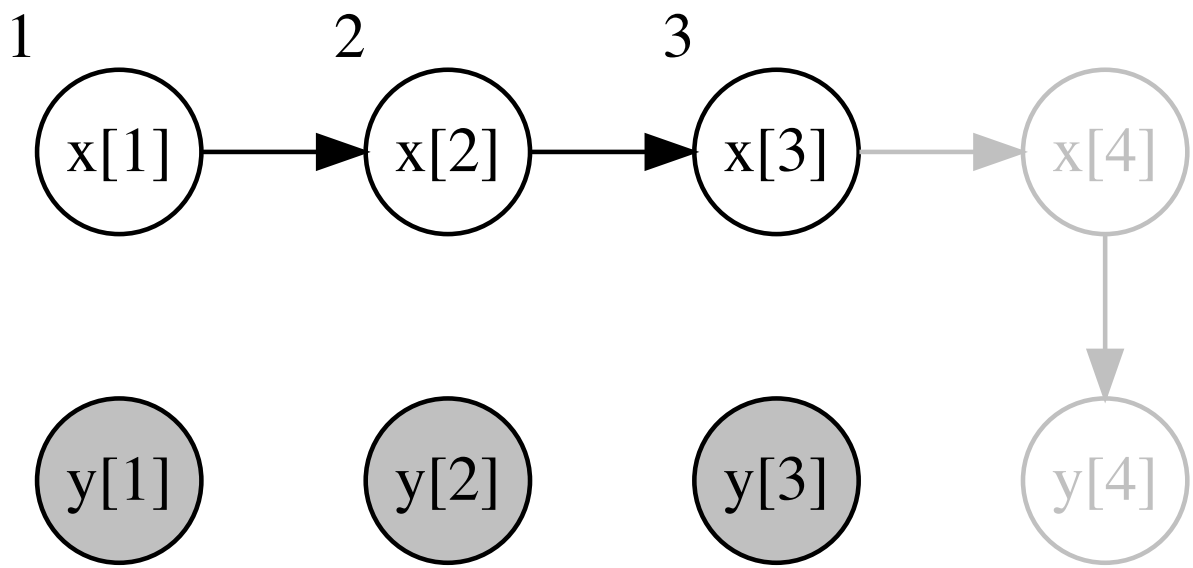
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	**assume** x[t]
`  y[t] ~ Gaussian(x[t], 1.0);`	
`}`	
`stdout.print(x[1]);`	



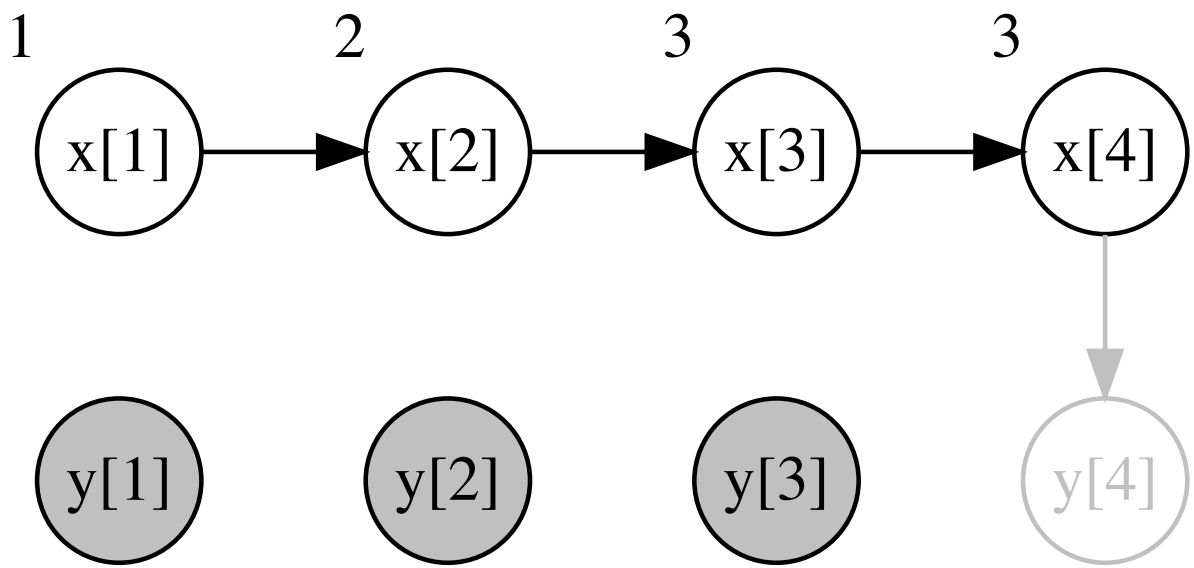
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	**observe** y[t]
`}`	
`stdout.print(x[1]);`	



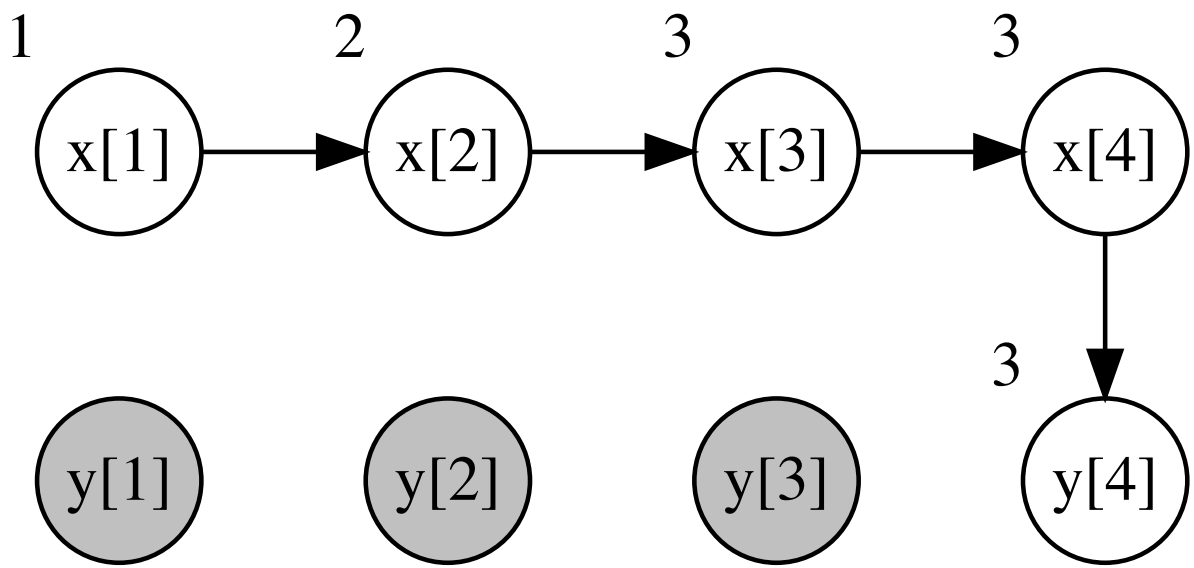
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	**observe** y[t]
`}`	
`stdout.print(x[1]);`	



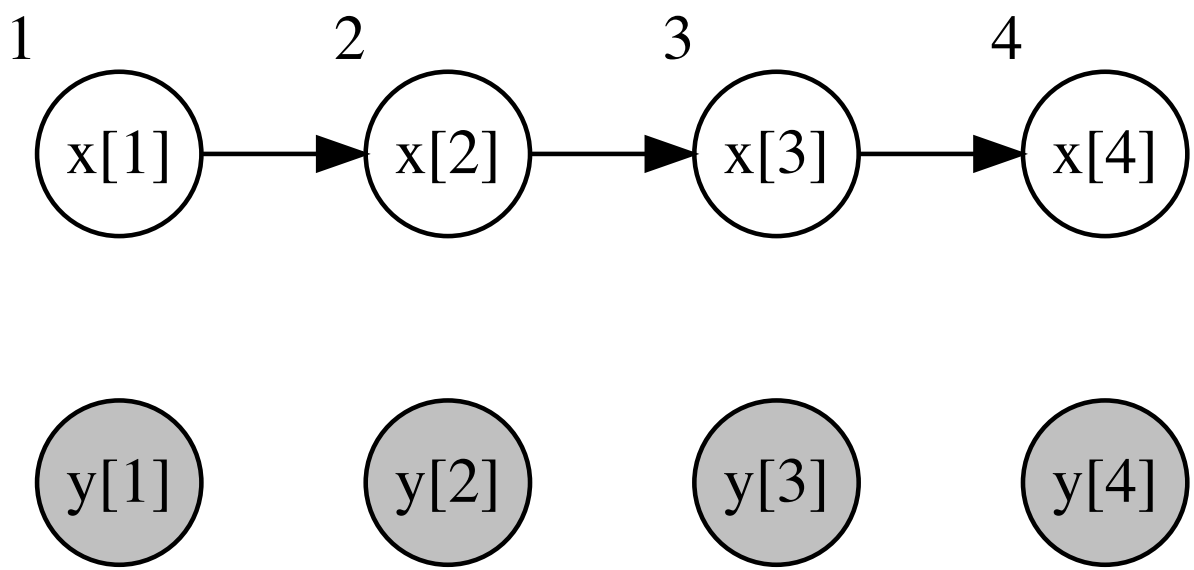
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	**observe** y[t]
`}`	
`stdout.print(x[1]);`	



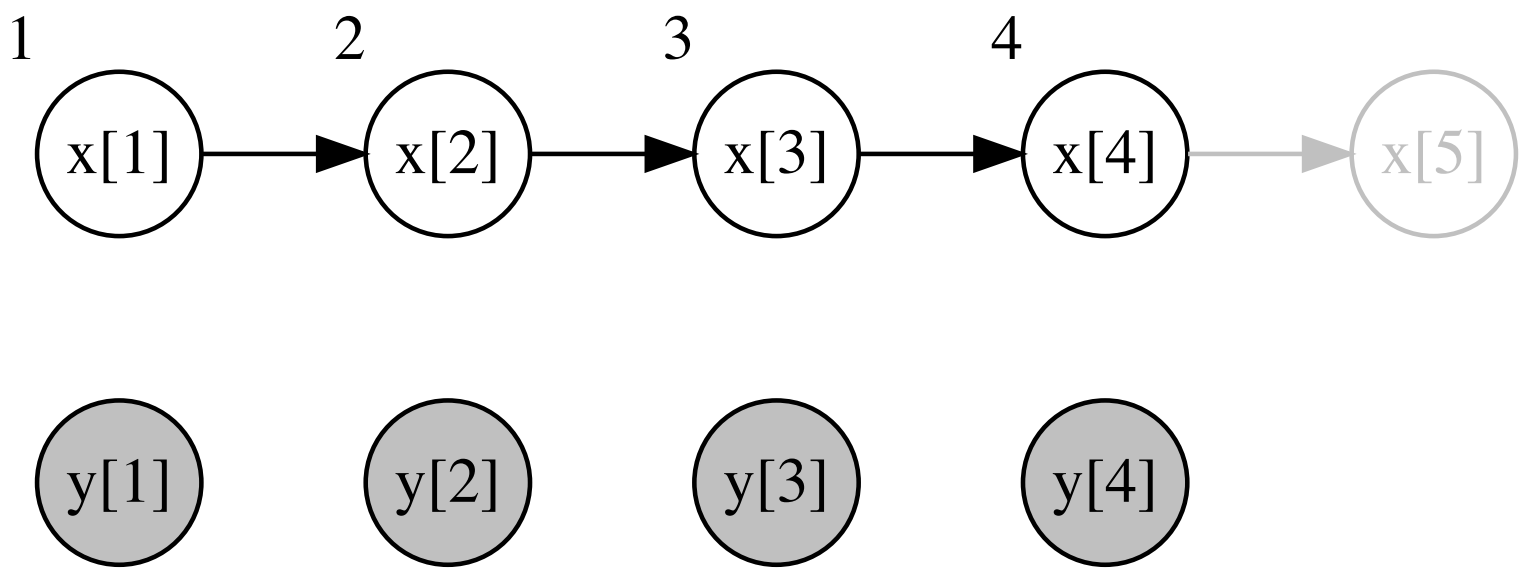
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	**observe** y[t]
`}`	
`stdout.print(x[1]);`	



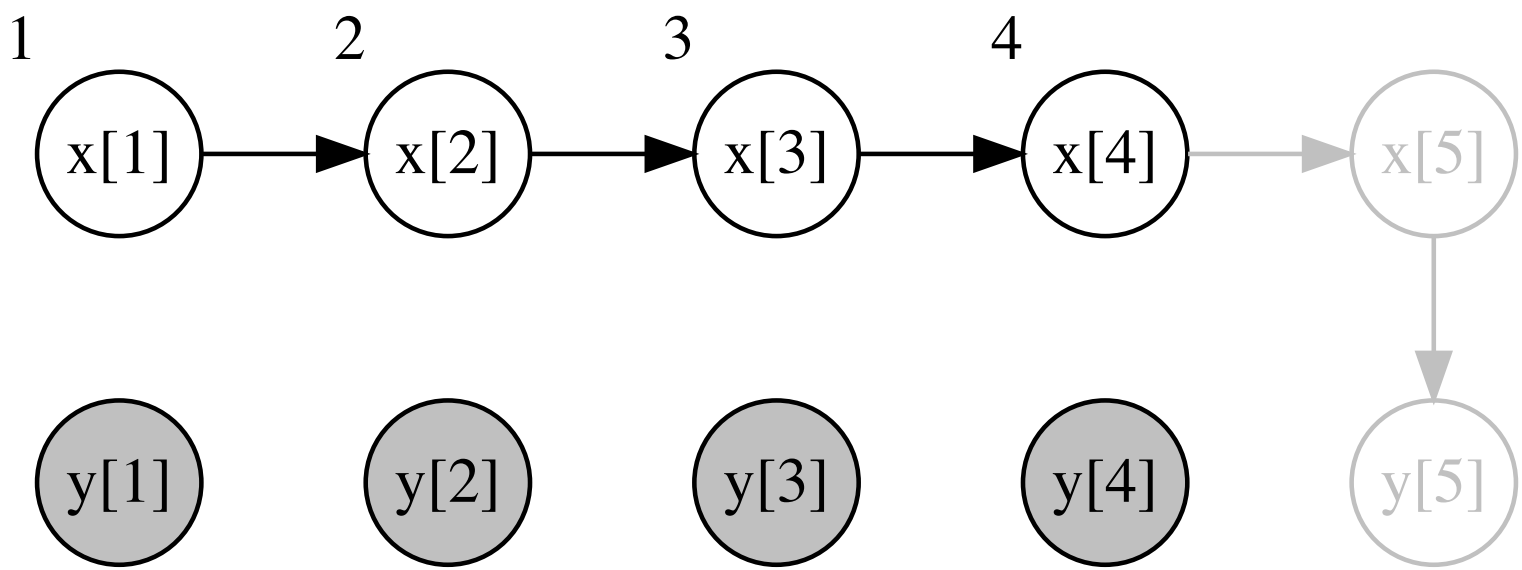
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	**assume** x[t]
`  y[t] ~ Gaussian(x[t], 1.0);`	
`}`	
`stdout.print(x[1]);`	



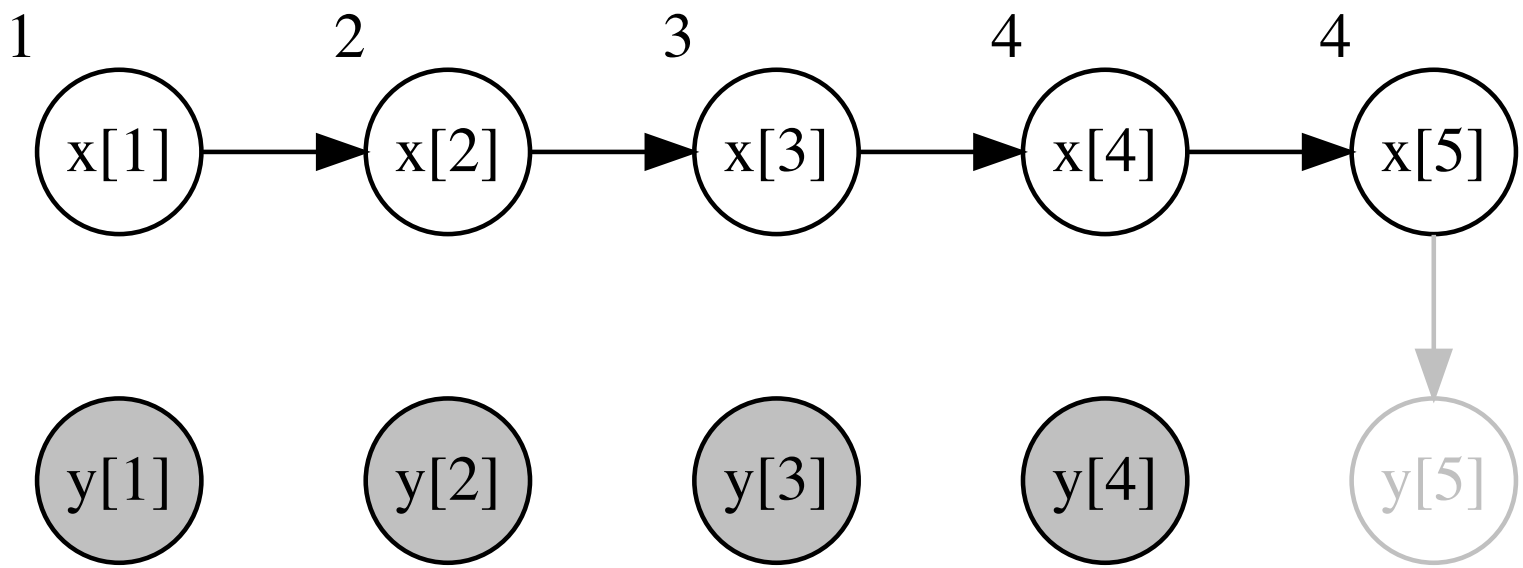
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	**observe** y[t]
`}`	
`stdout.print(x[1]);`	



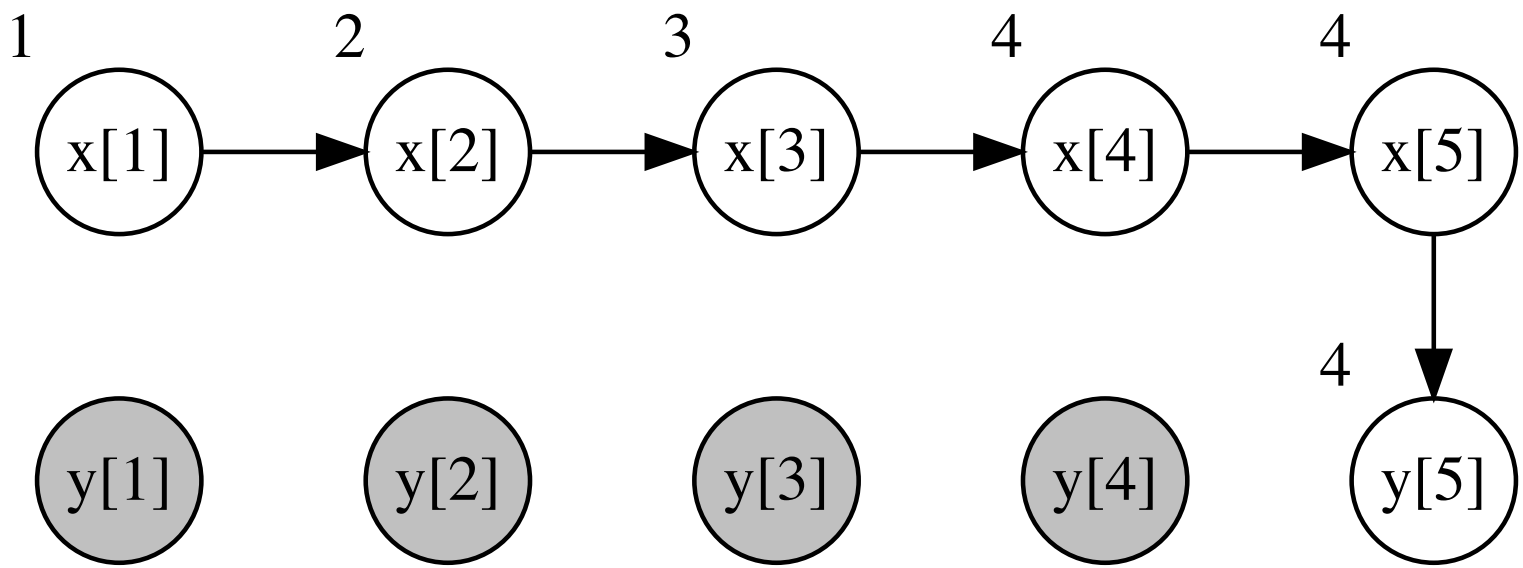
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	**observe** y[t]
`}`	
`stdout.print(x[1]);`	



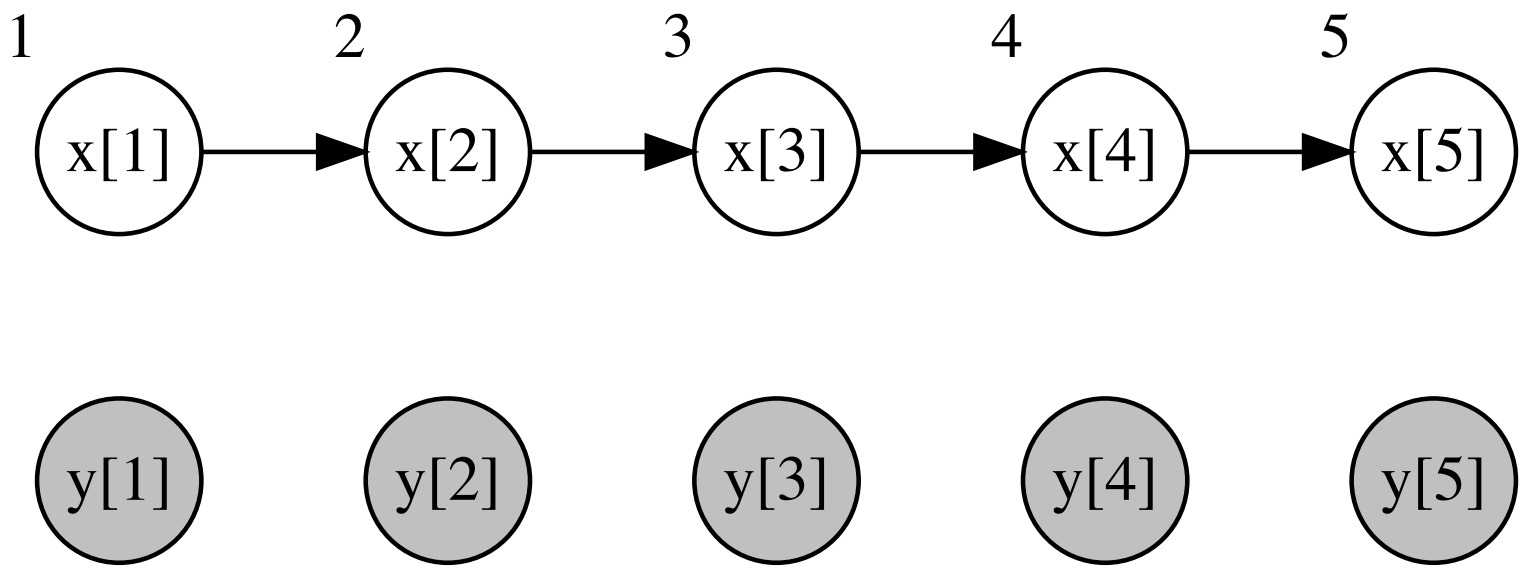
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	**observe** y[t]
`}`	
`stdout.print(x[1]);`	



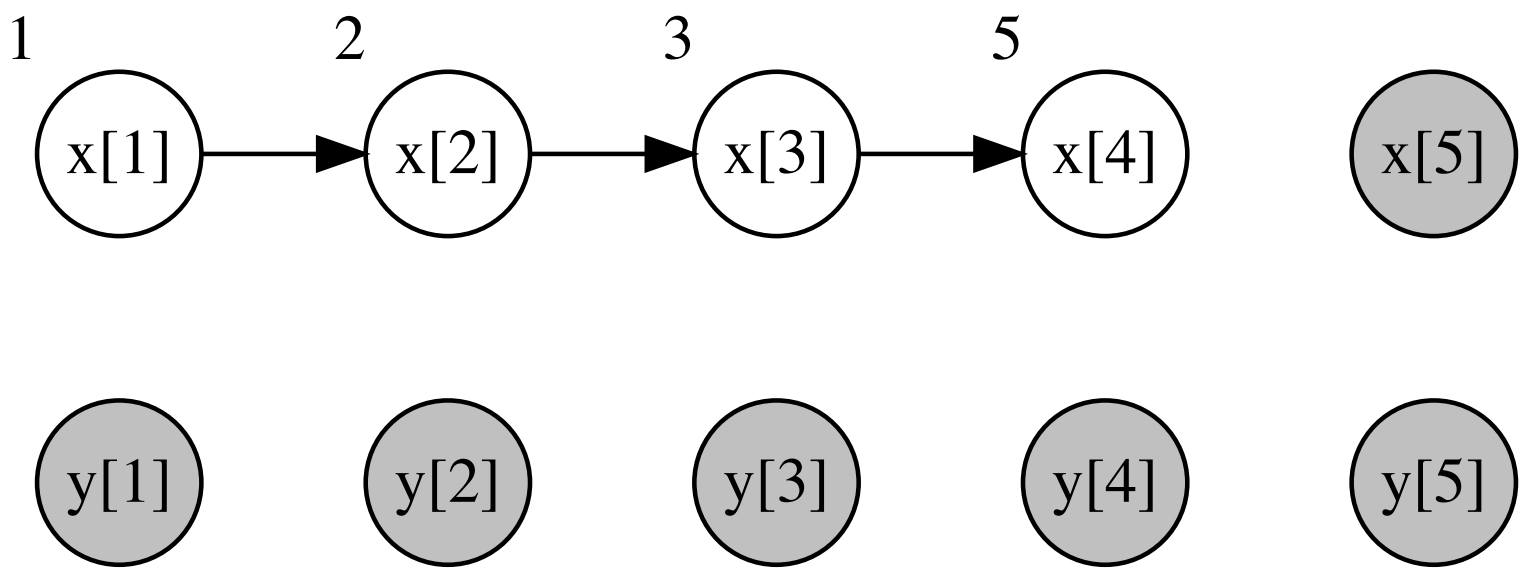
Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	**observe** y[t]
`}`	
`stdout.print(x[1]);`	



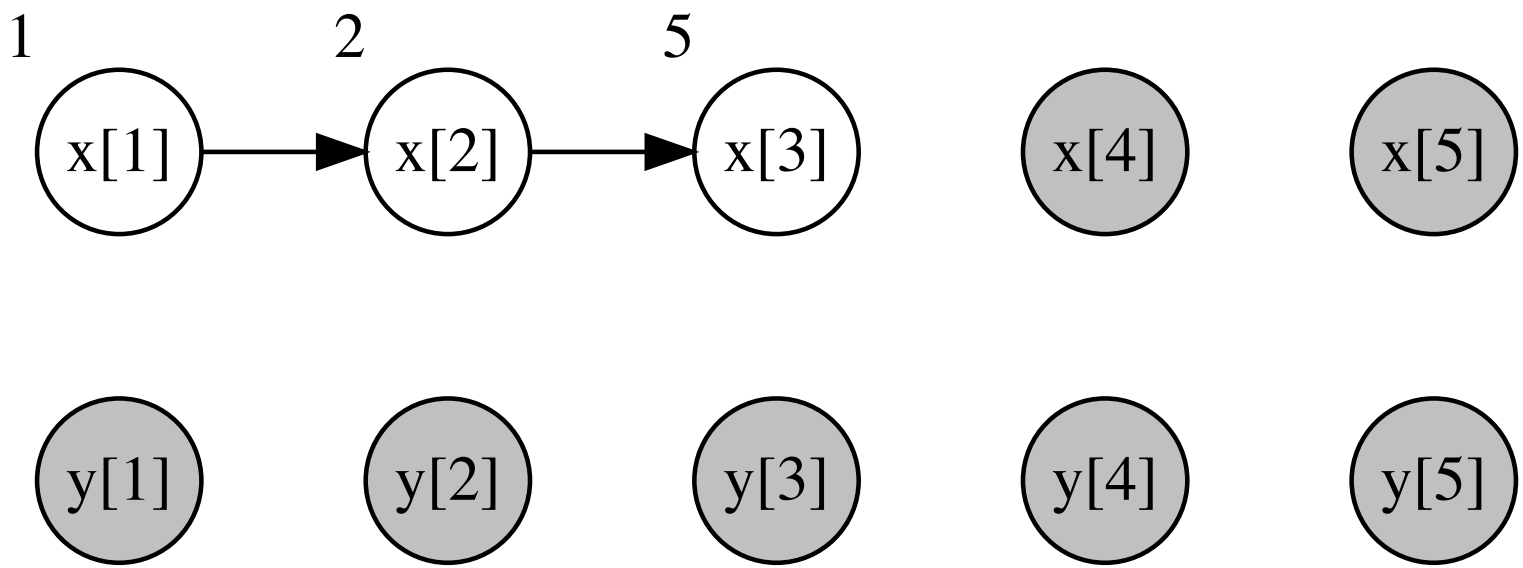
Example #2: Kalman Filter

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	
`}`	
`stdout.print(x[1]);`	**value** x[1]



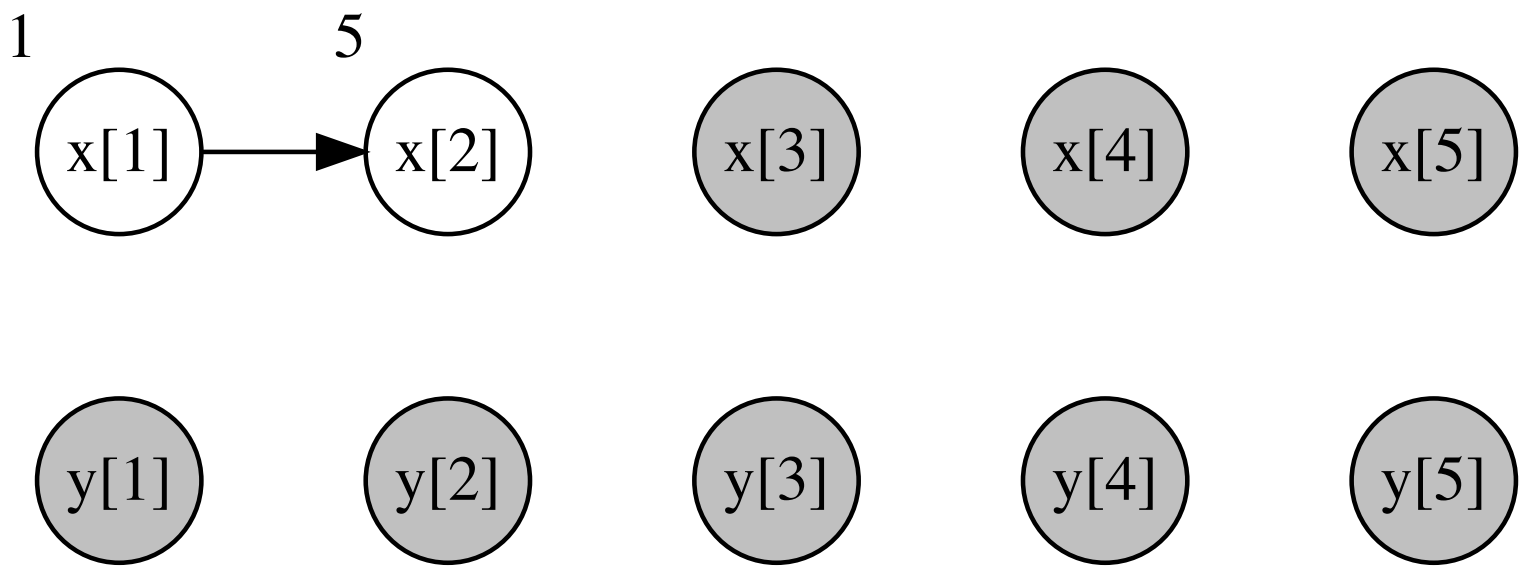
Example #2: Kalman Filter

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	
`}`	
`stdout.print(x[1]);`	**value** x[1]



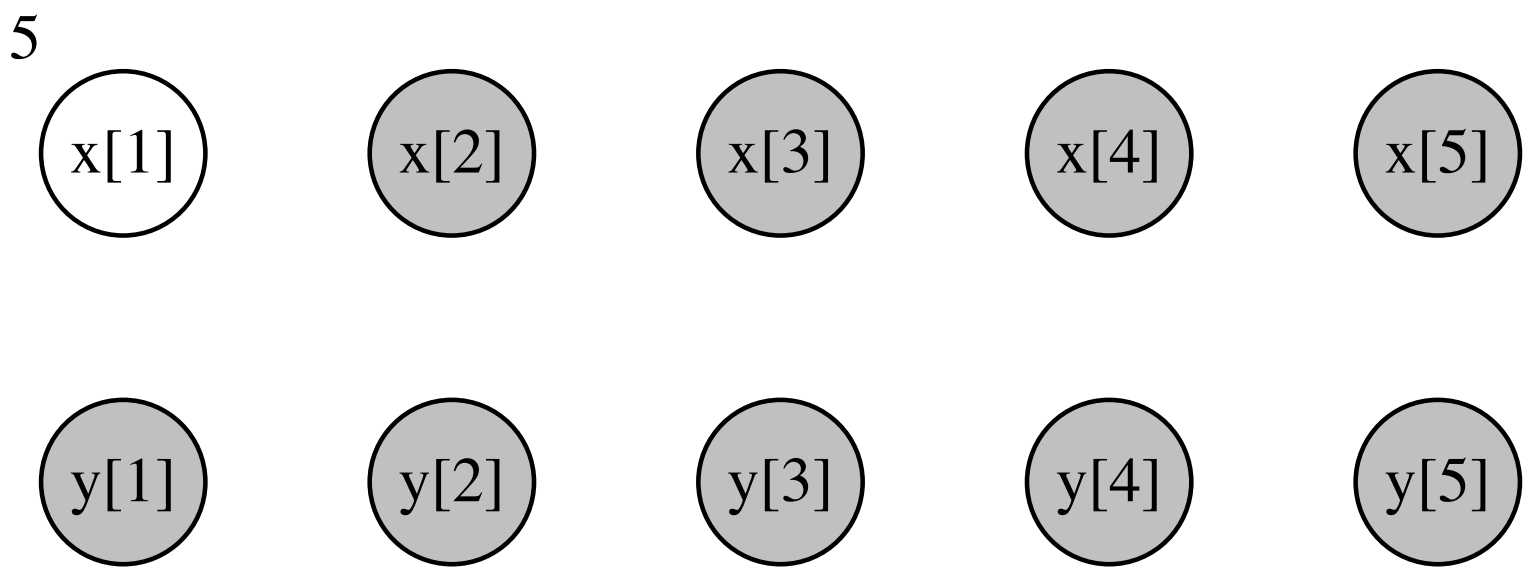
Example #2: Kalman Filter

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	
`}`	
`stdout.print(x[1]);`	**value** x[1]



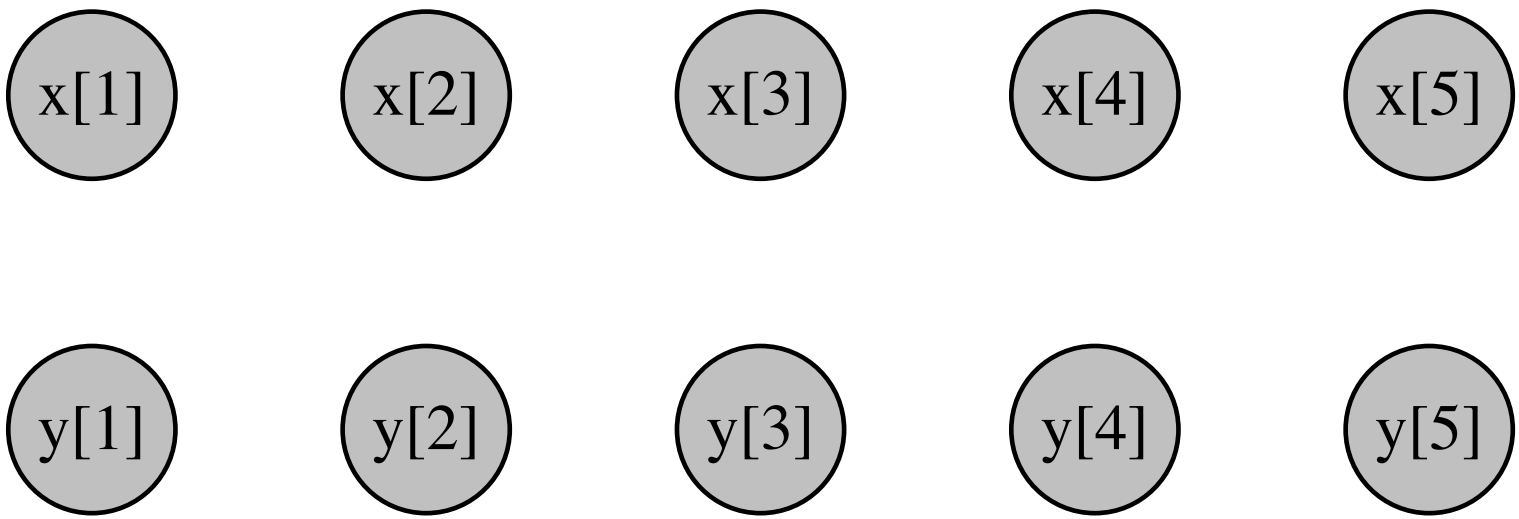
Example #2: Kalman Filter

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	
`}`	
`stdout.print(x[1]);`	**value** x[1]



Example #2: Kalman Filter

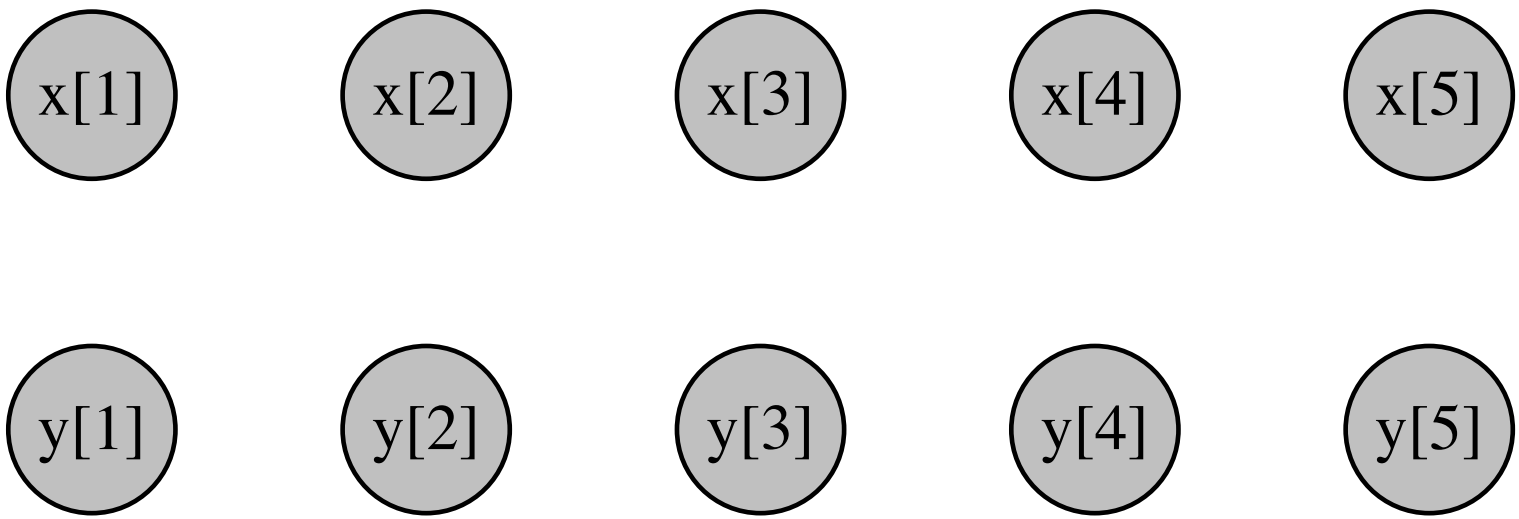
Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	
`y[1] ~ Gaussian(x[1], 1.0);`	
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	
`  y[t] ~ Gaussian(x[t], 1.0);`	
`}`	
`stdout.print(x[1]);`	**value** x[1]



Example #2: Kalman Filter

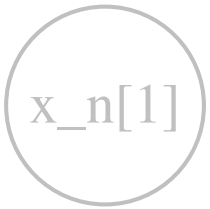
Code	Checkpoint

```
x[1] ~ Gaussian(0.0, 1.0);
y[1] ~ Gaussian(x[1], 1.0);
for (t in 2..T) {
  x[t] ~ Gaussian(a*x[t - 1], 1.0);
  y[t] ~ Gaussian(x[t], 1.0);
}
stdout.print(x[1]);
```

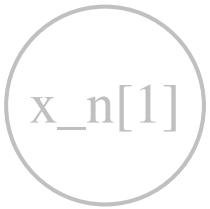


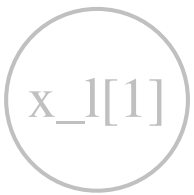
Example #3

Example #3

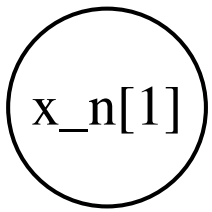


Example #3

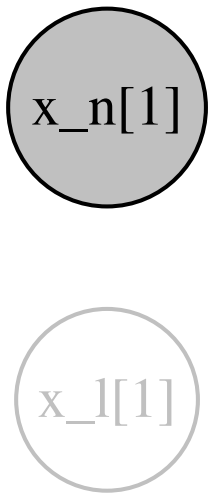




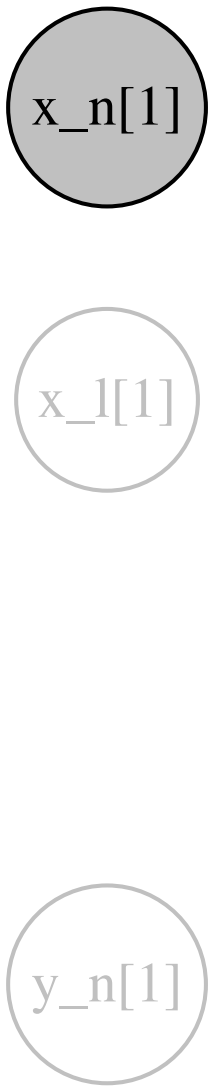
Example #3



Example #3



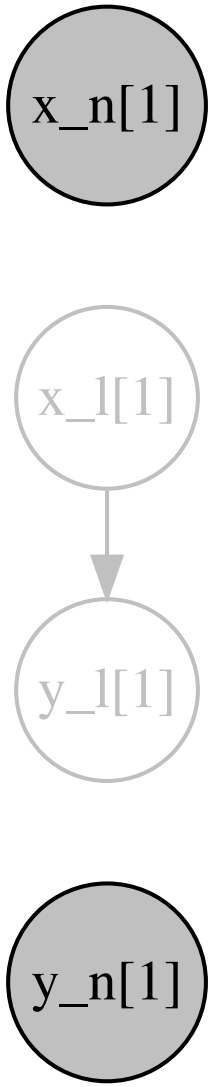
Example #3



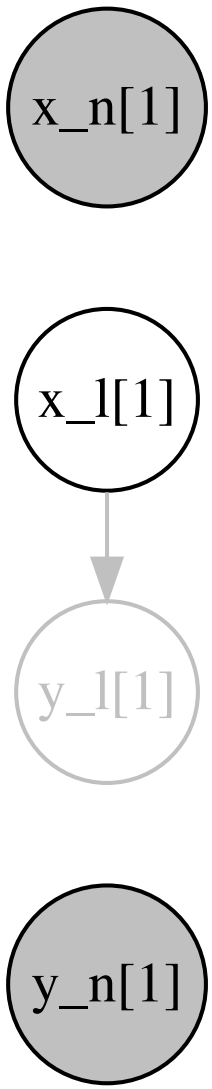
Example #3

Example #3

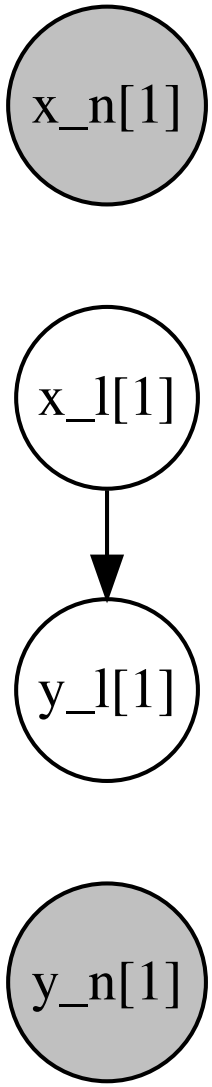
Example #3



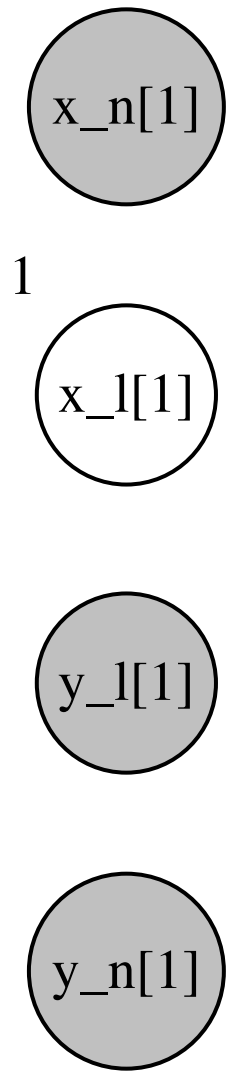
Example #3



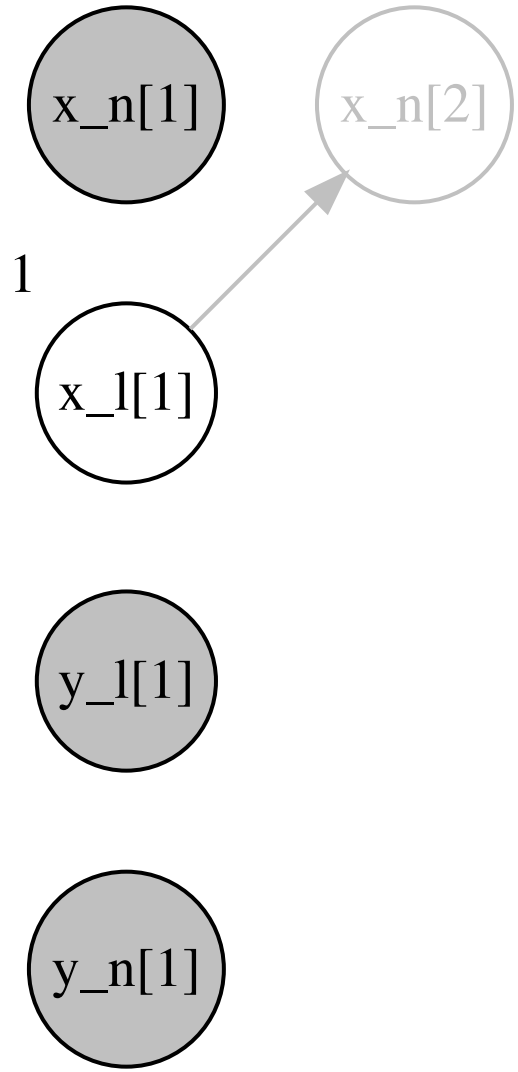
Example #3



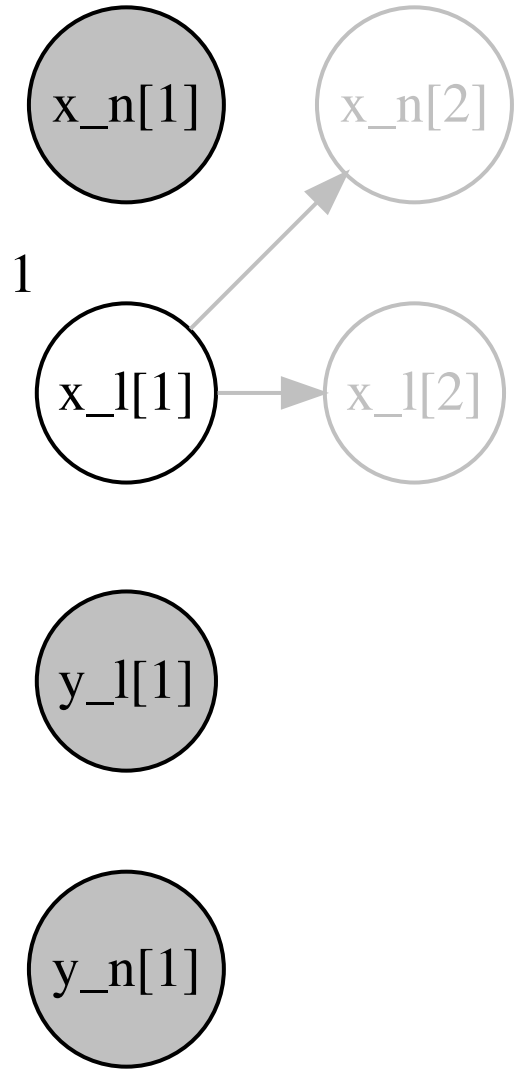
Example #3



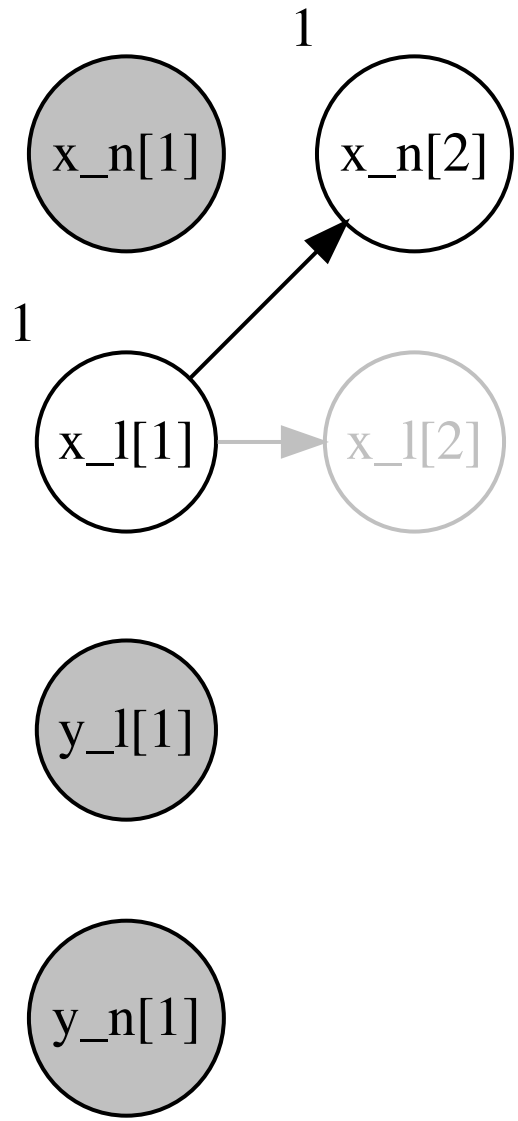
Example #3



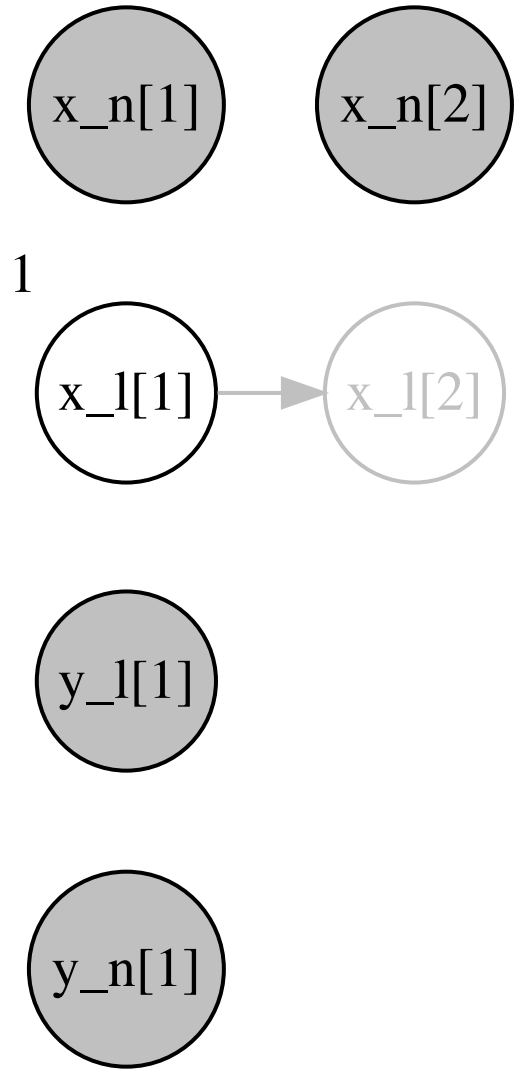
Example #3



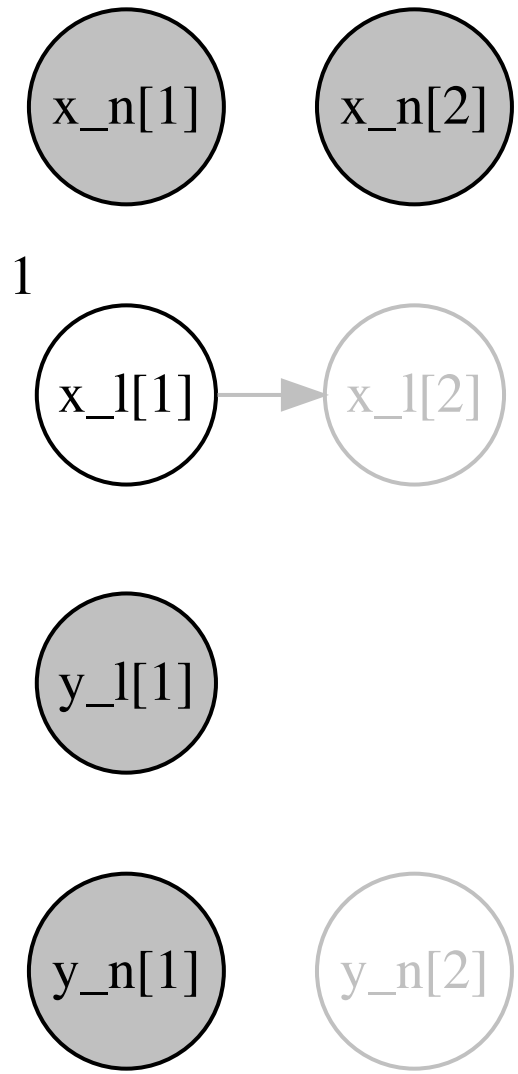
Example #3



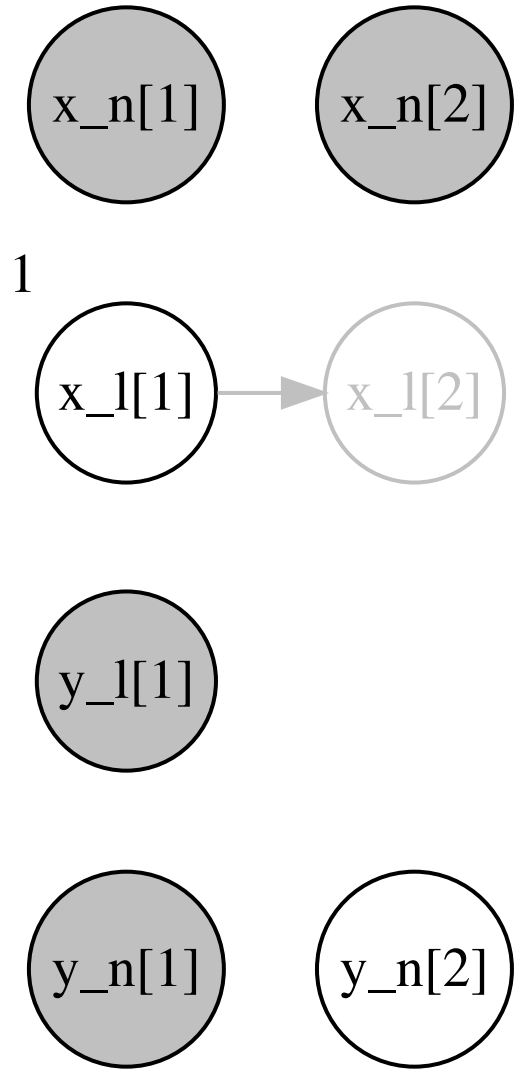
Example #3



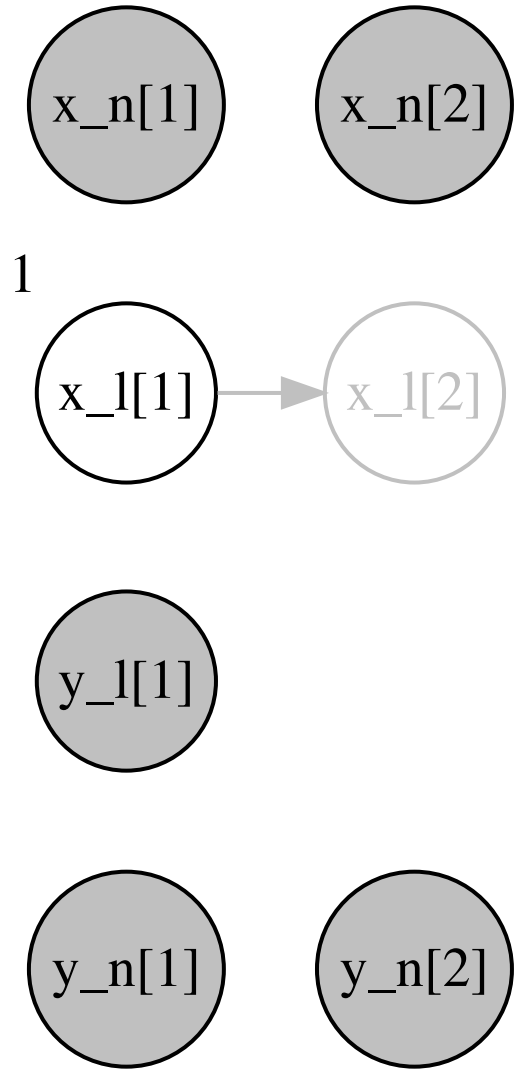
Example #3



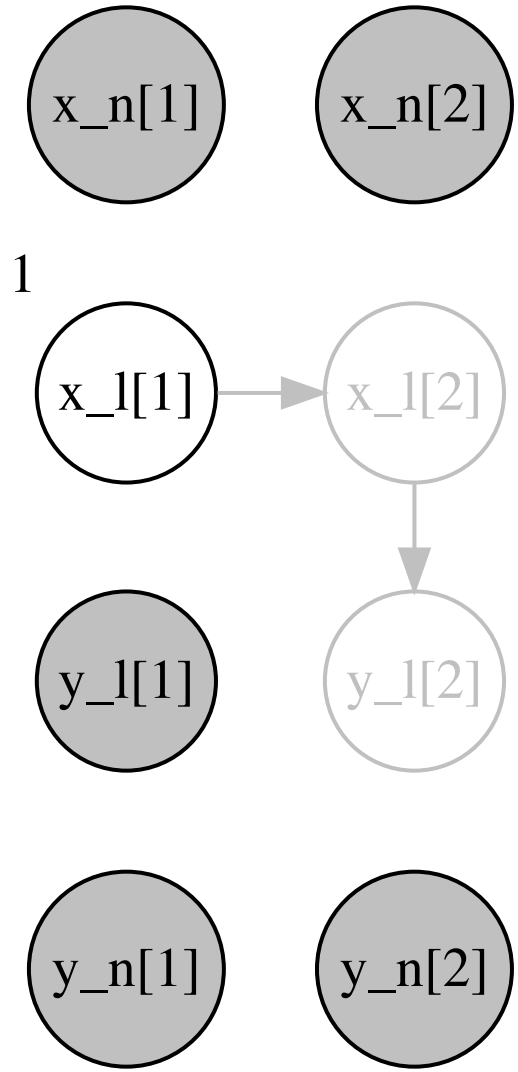
Example #3



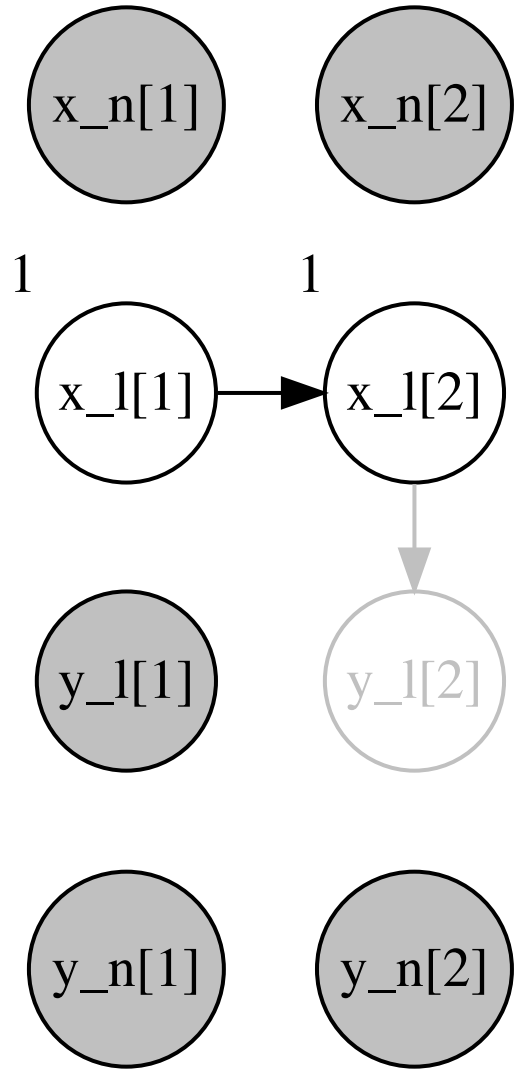
Example #3



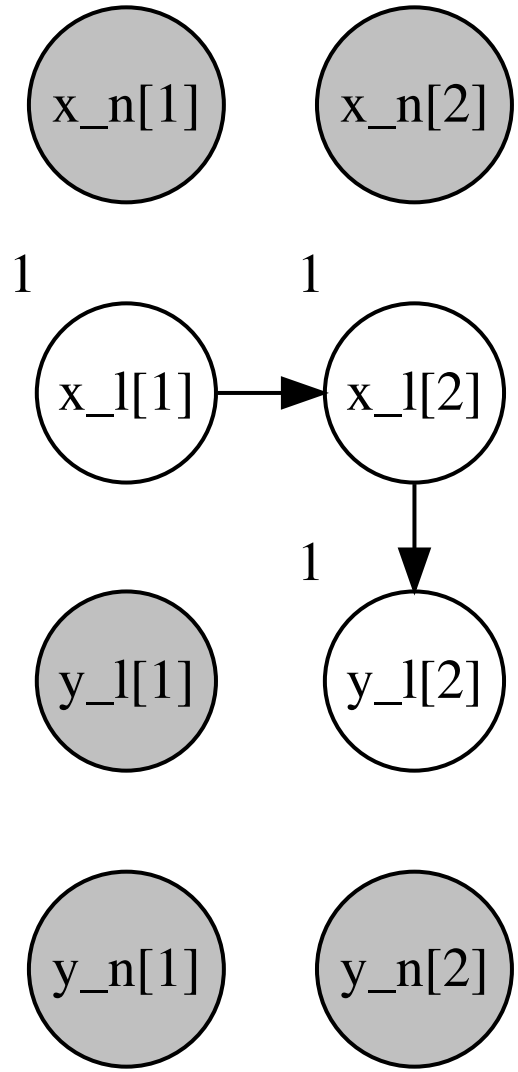
Example #3



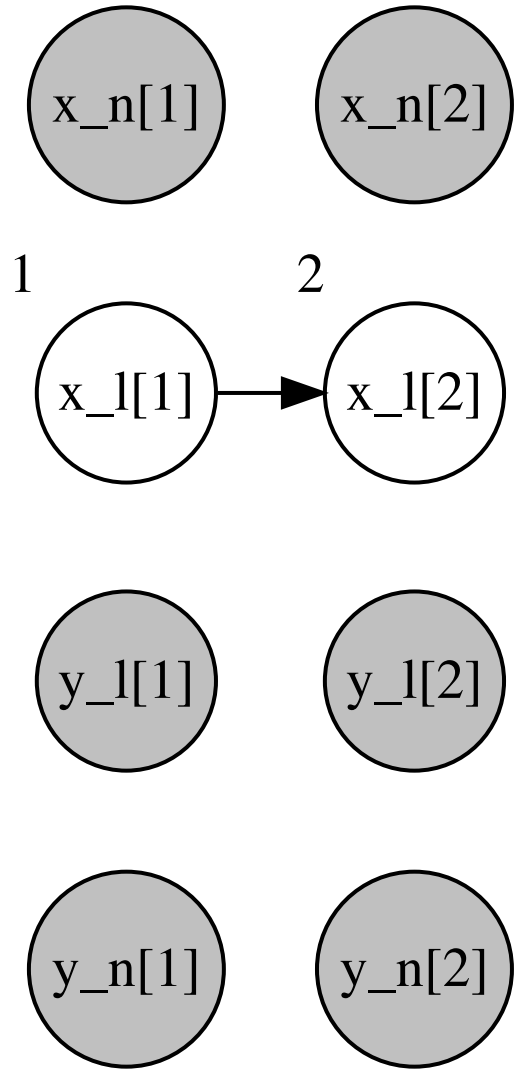
Example #3



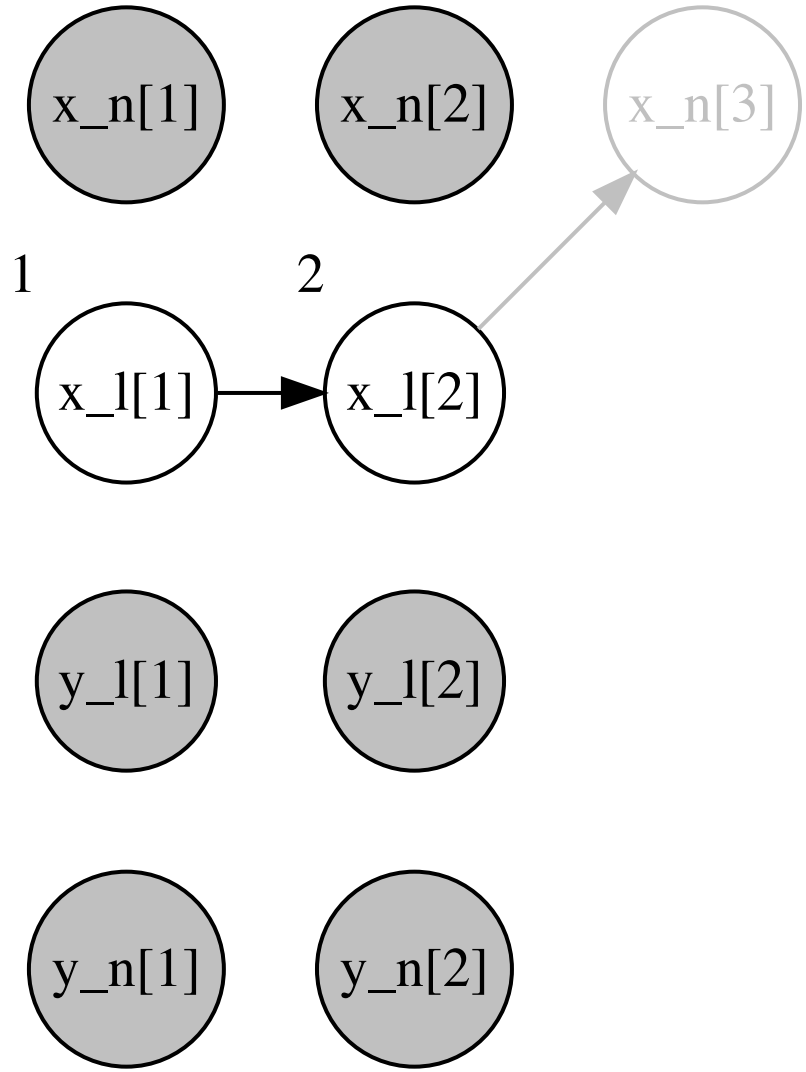
Example #3



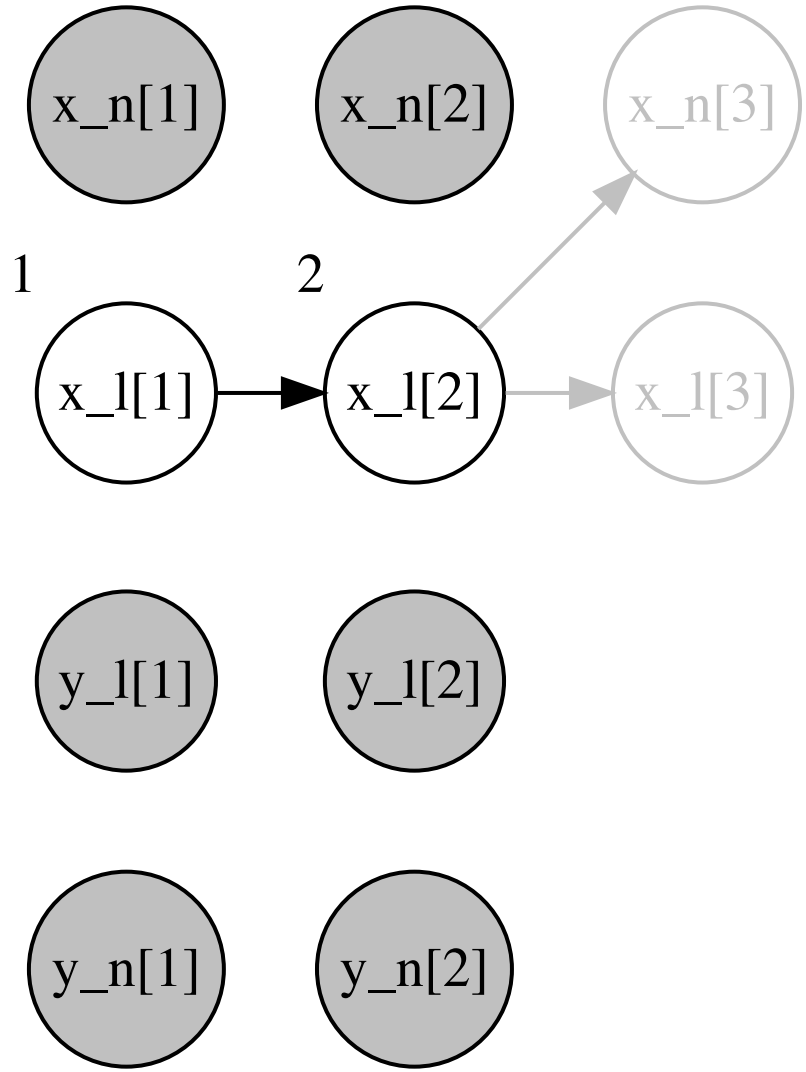
Example #3



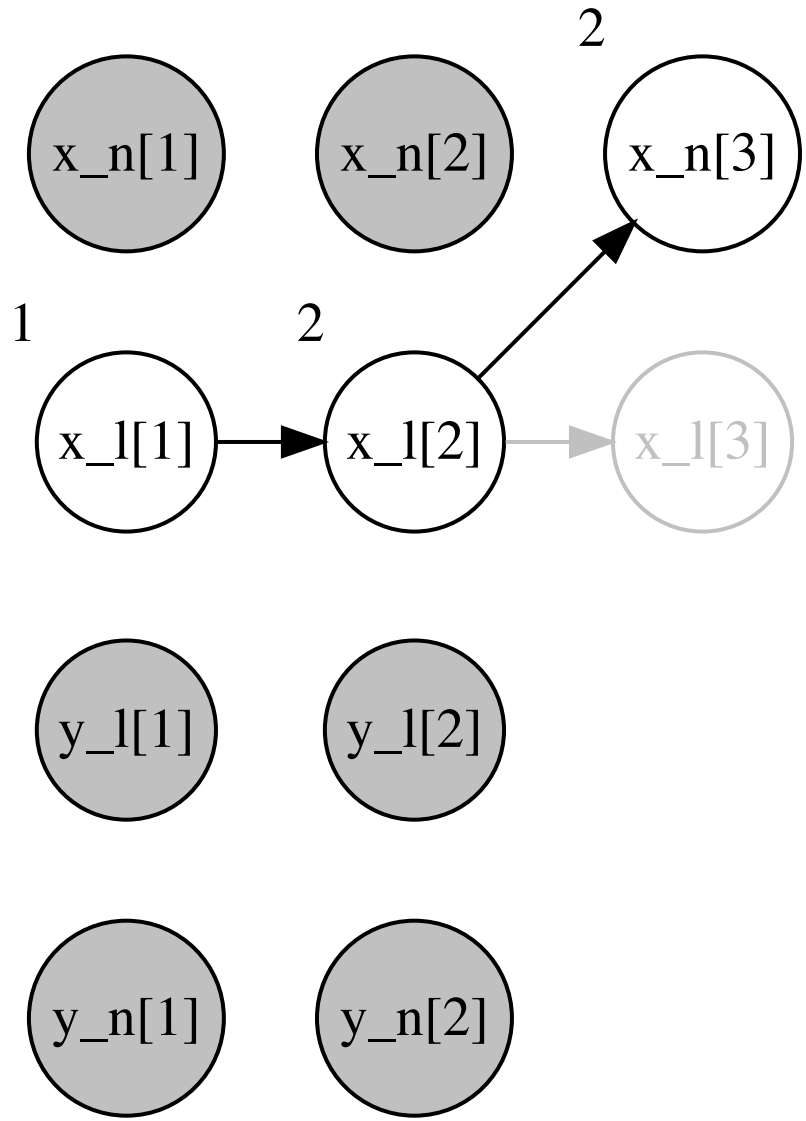
Example #3



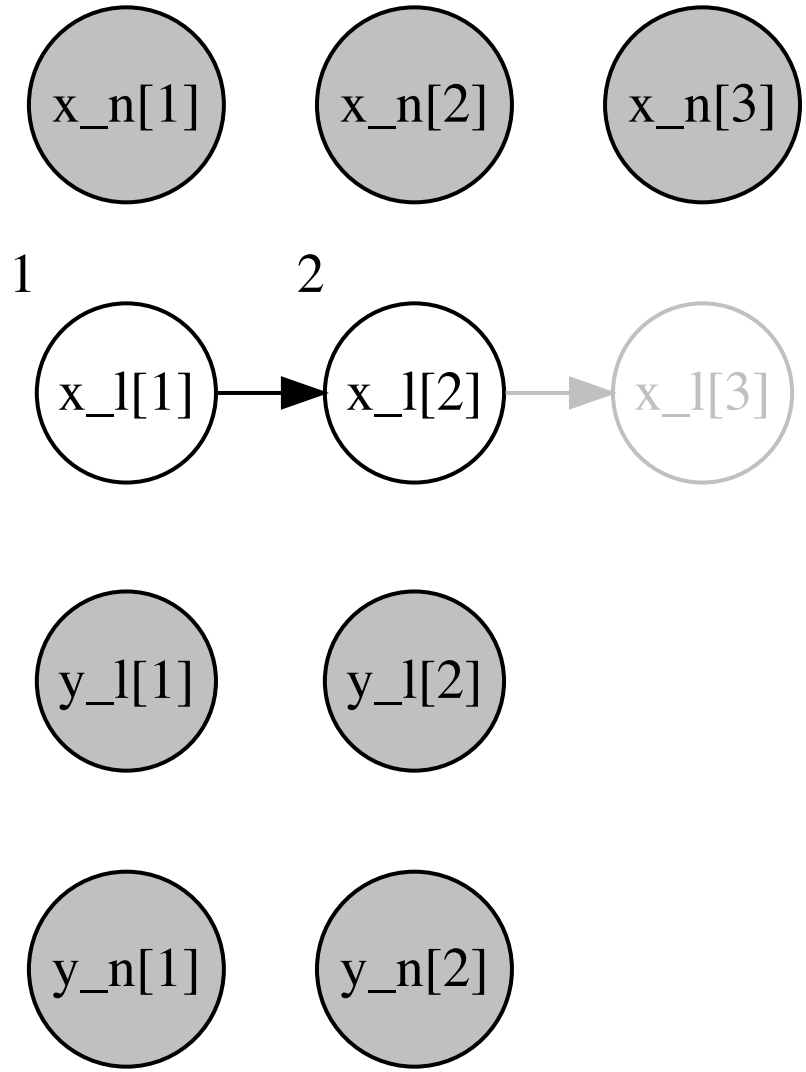
Example #3



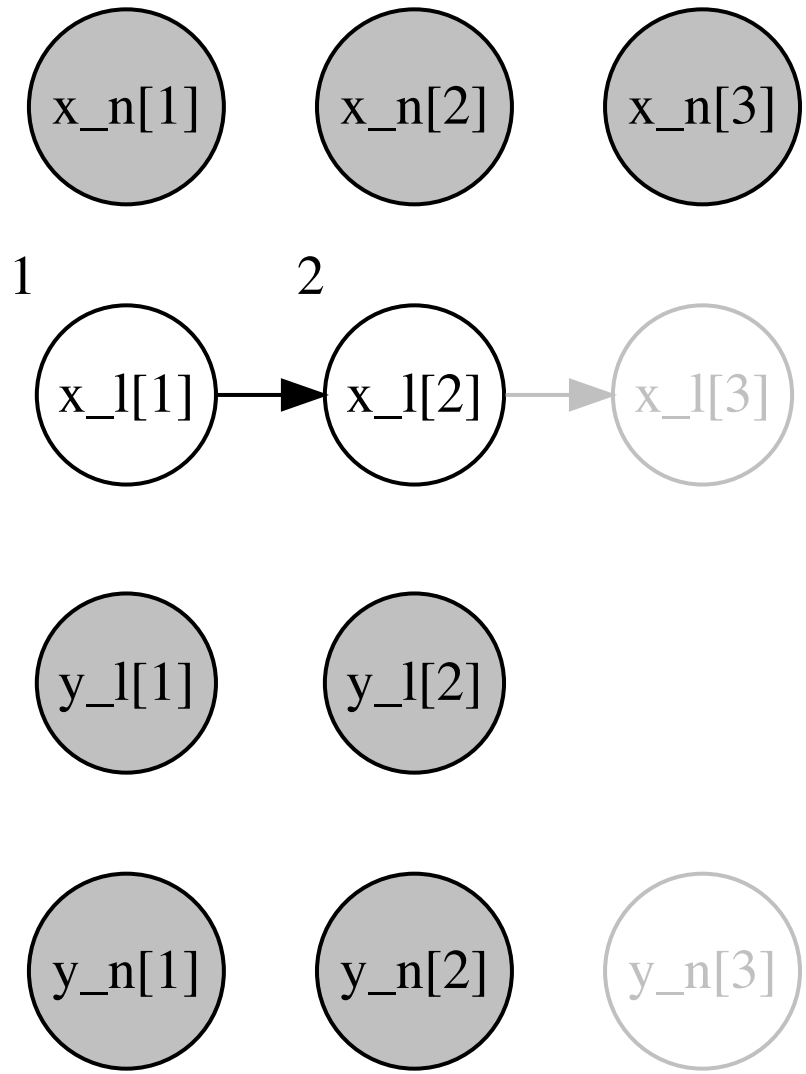
Example #3



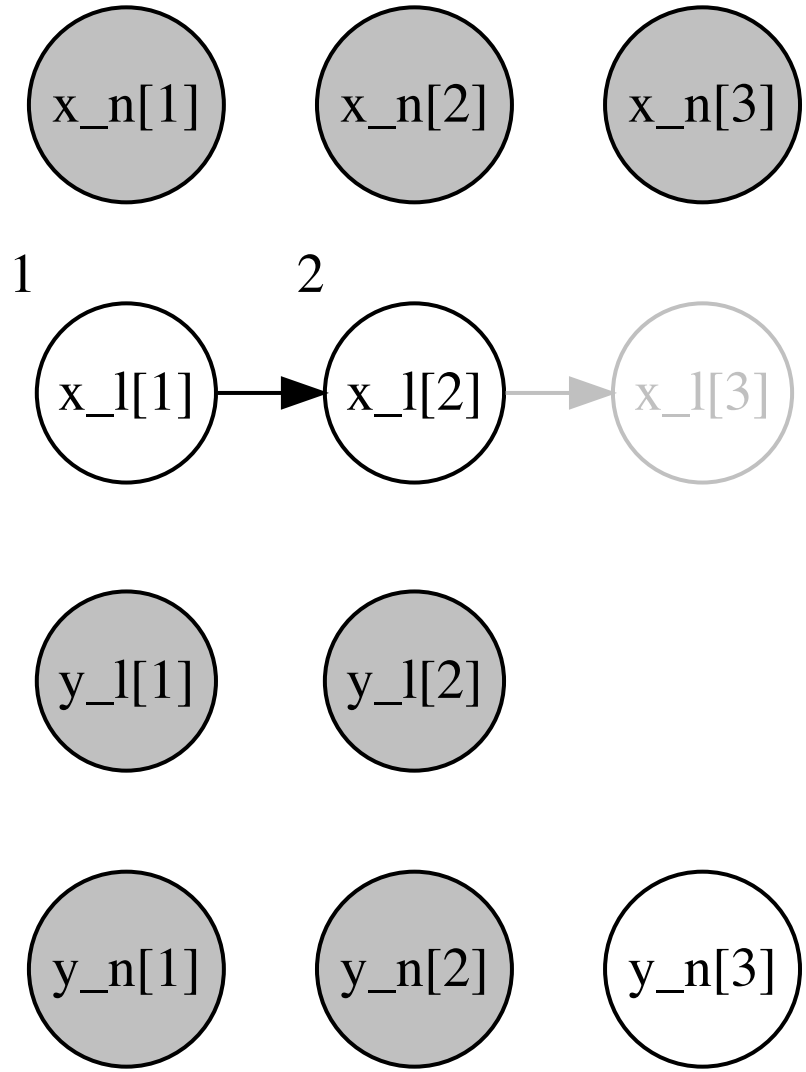
Example #3



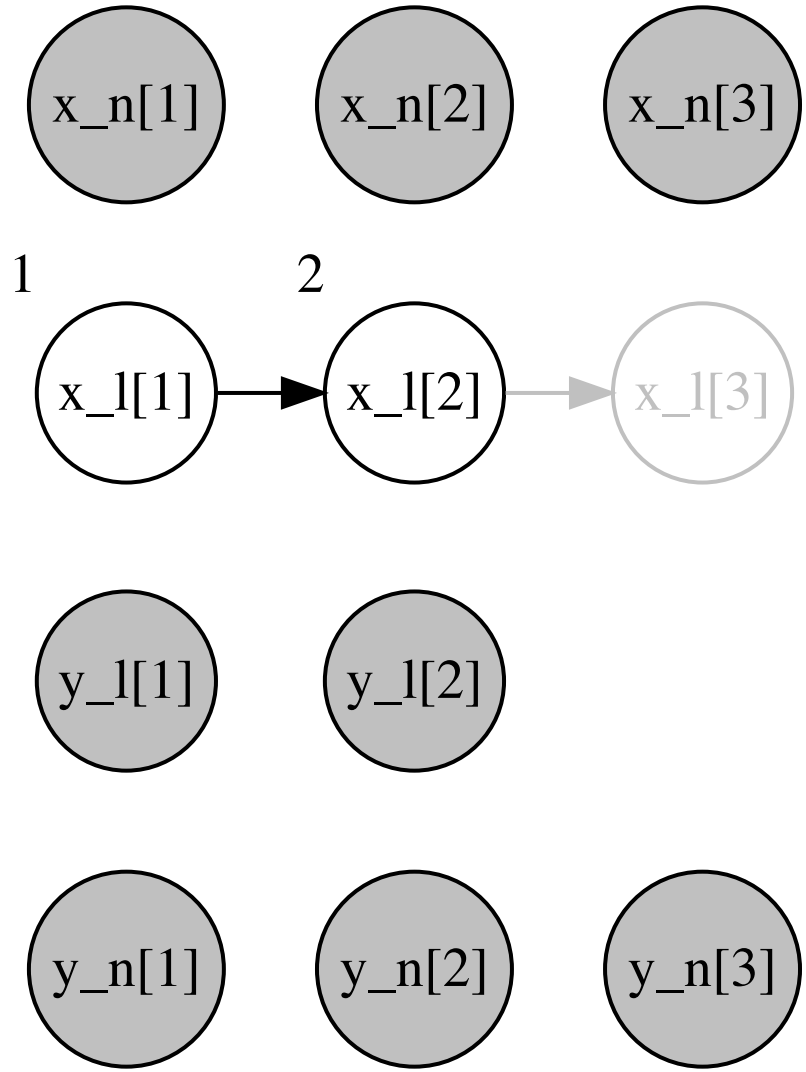
Example #3



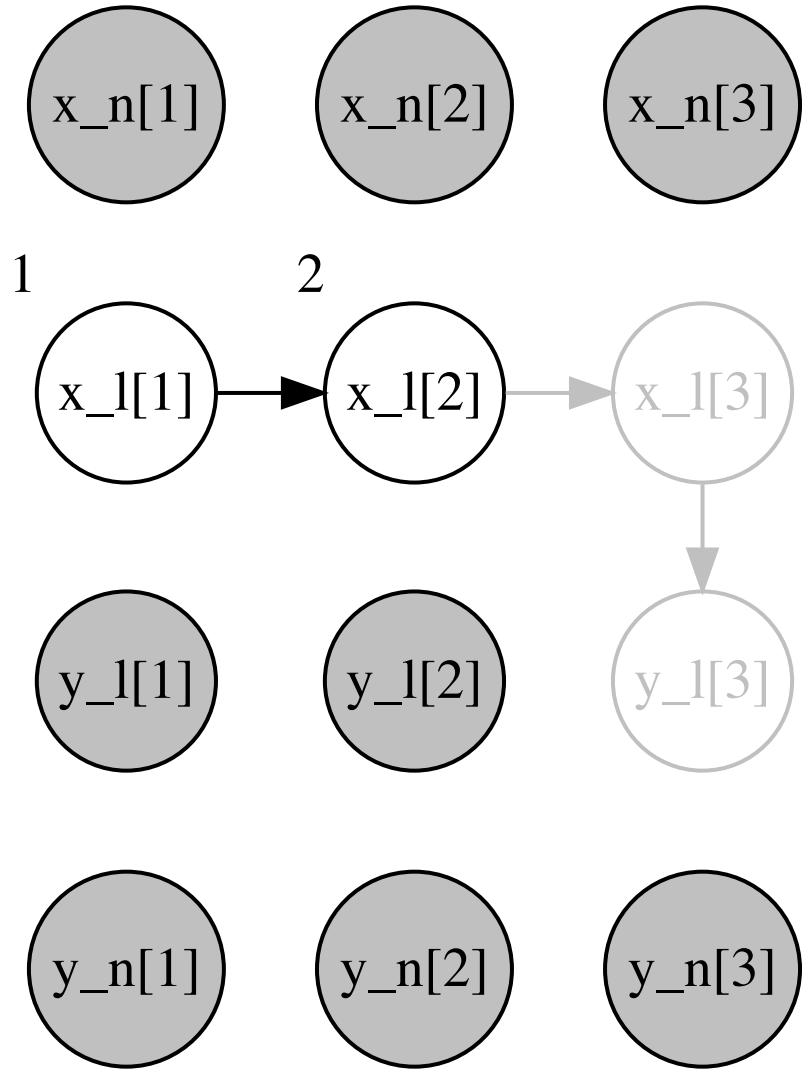
Example #3



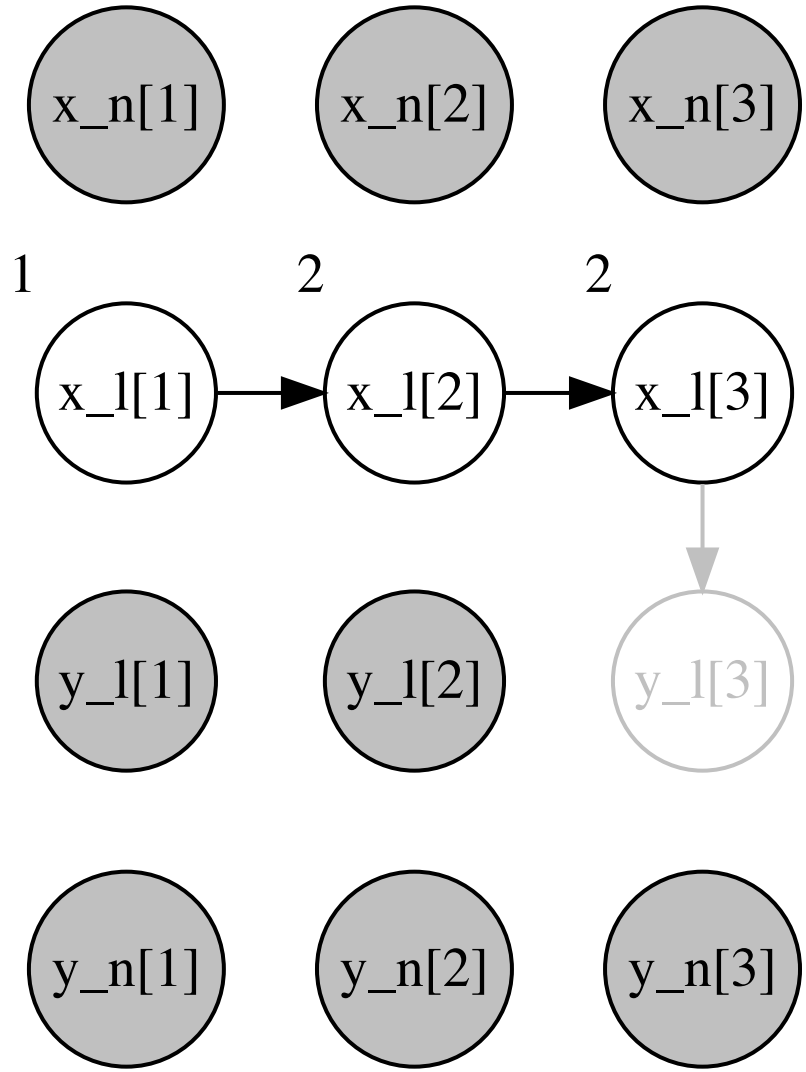
Example #3



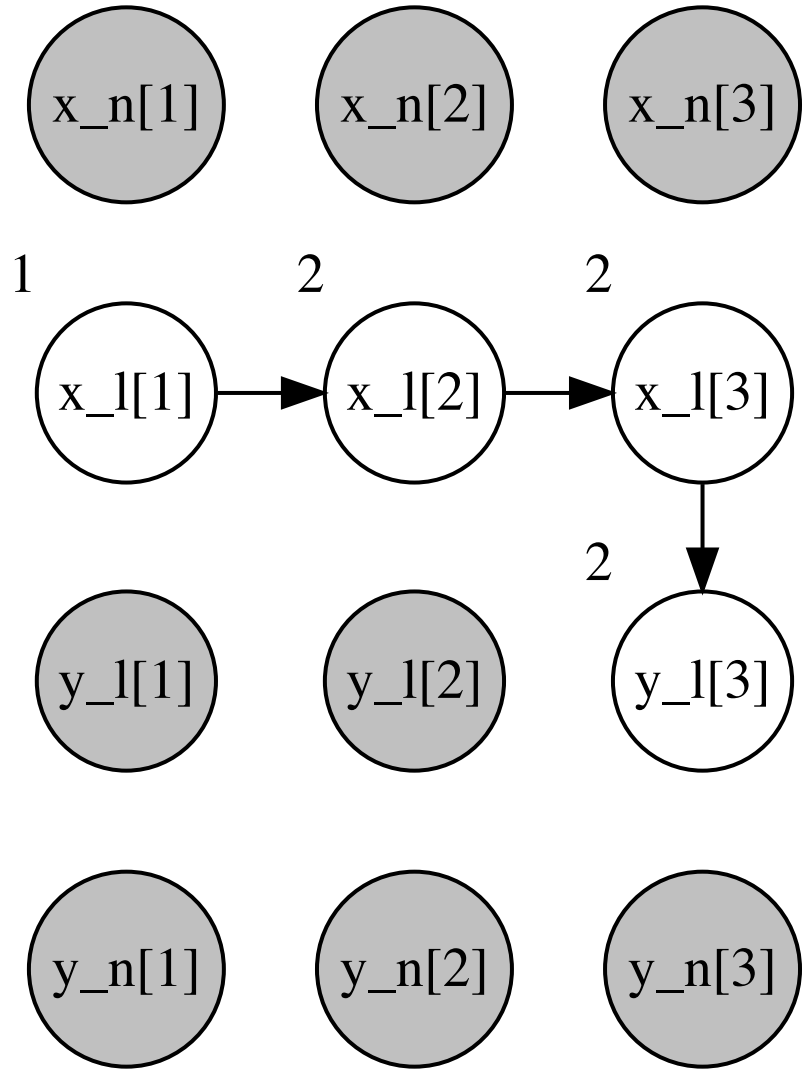
Example #3



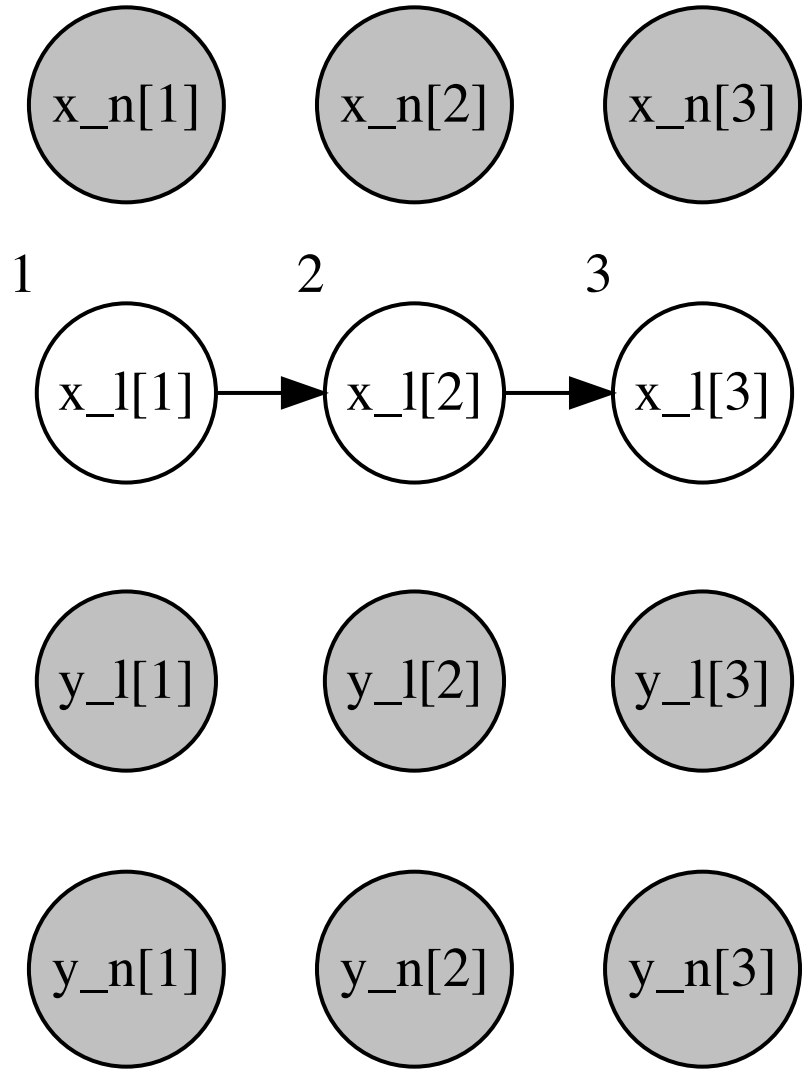
Example #3



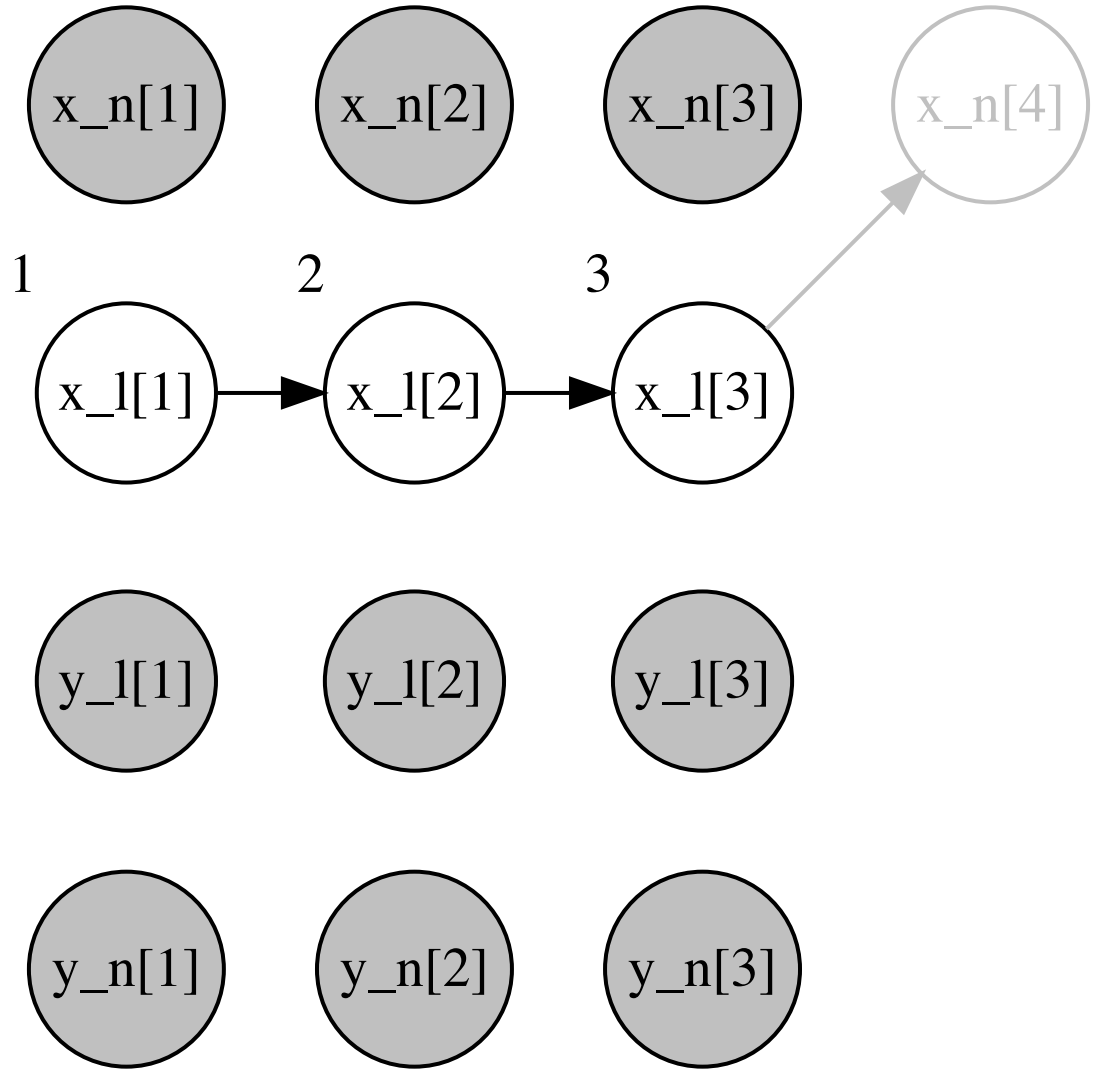
Example #3



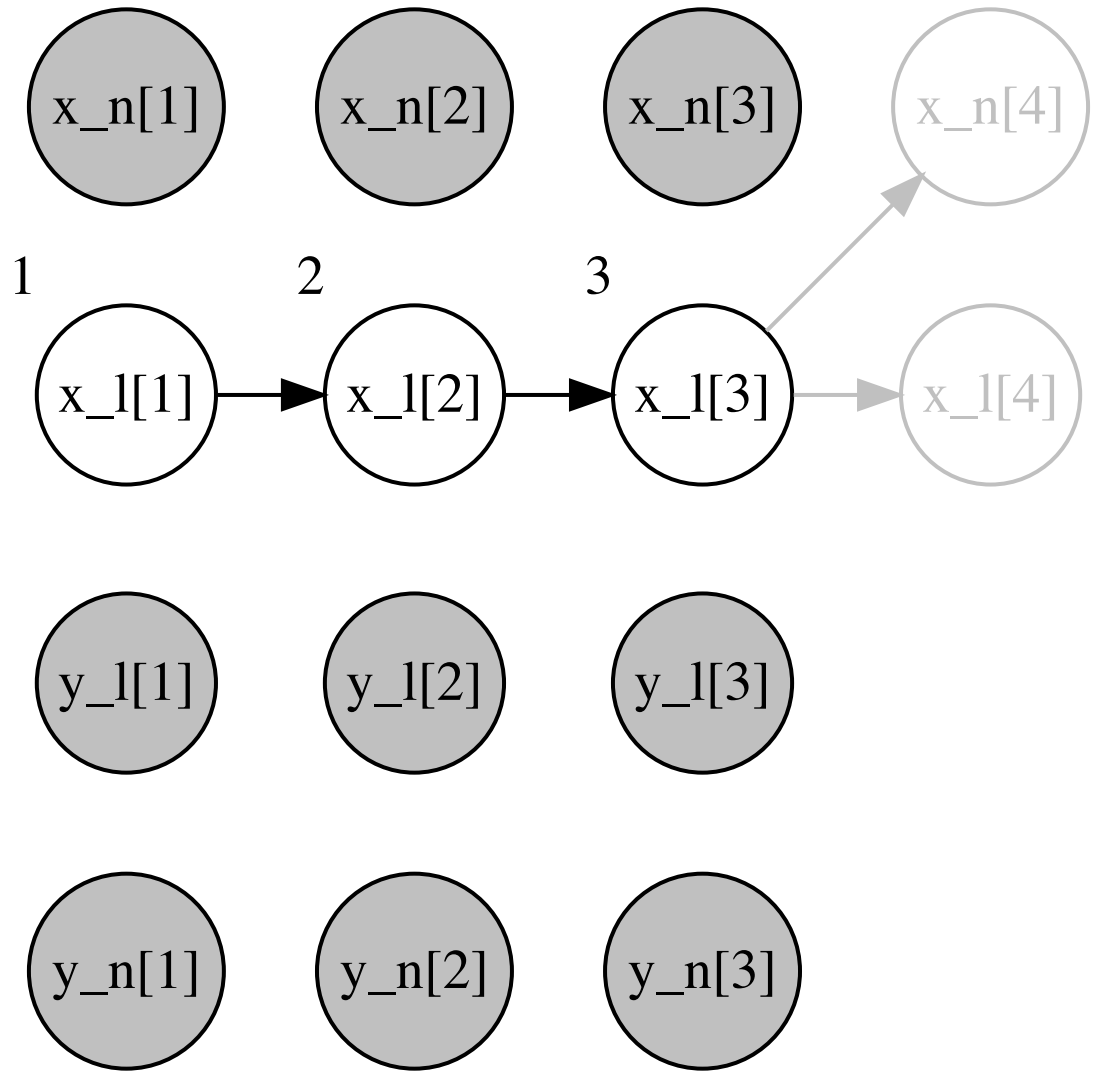
Example #3



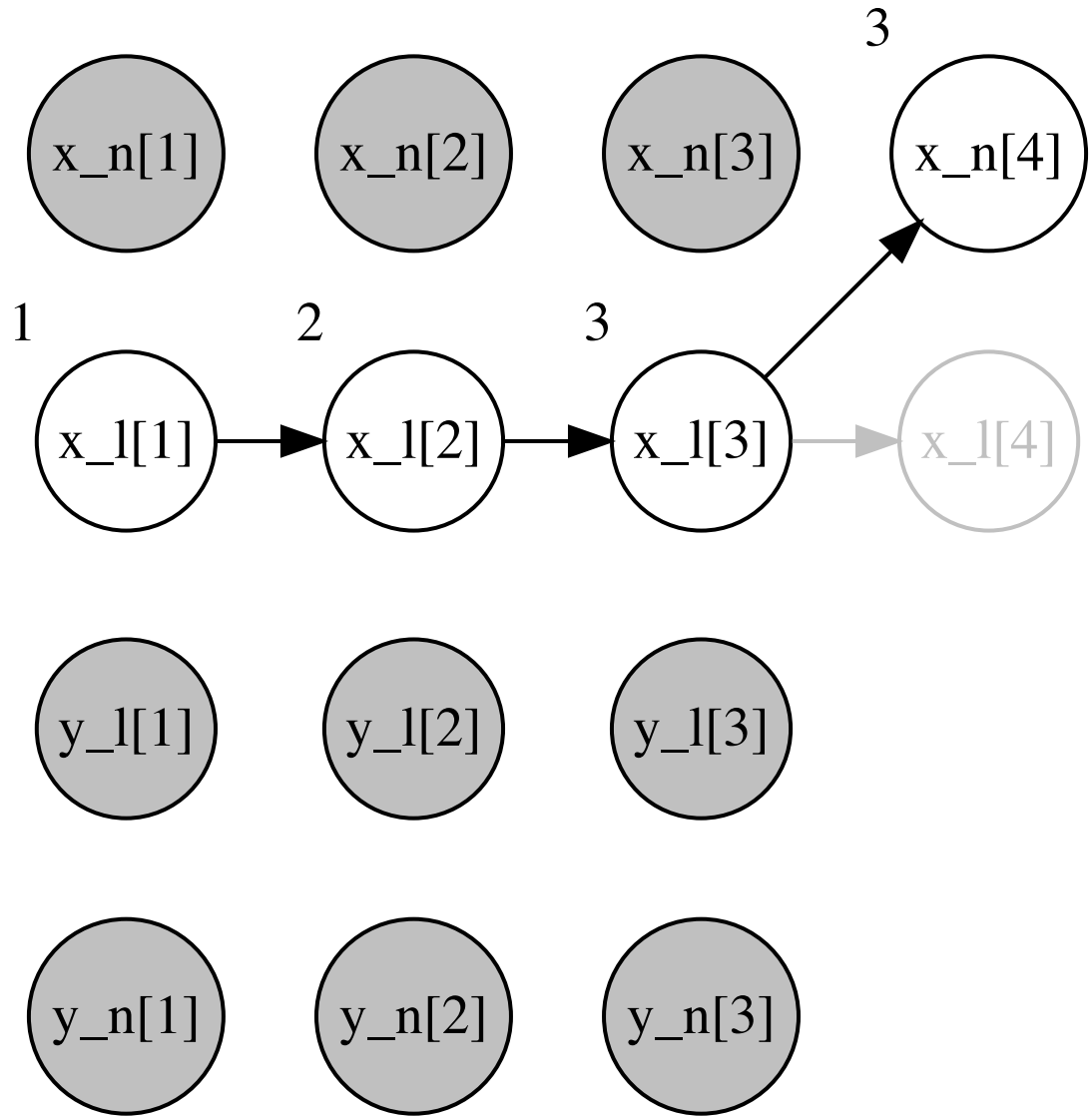
Example #3



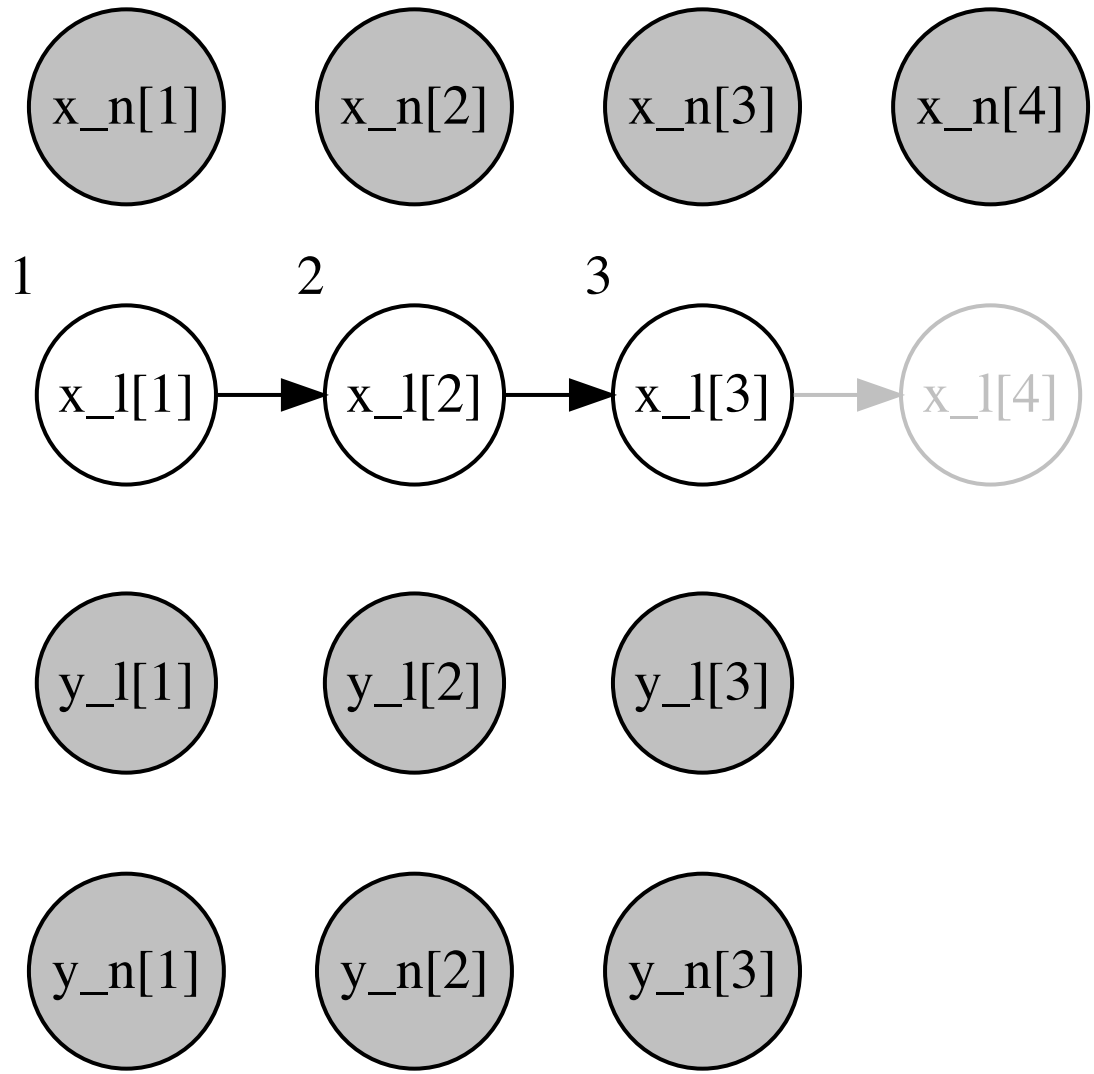
Example #3



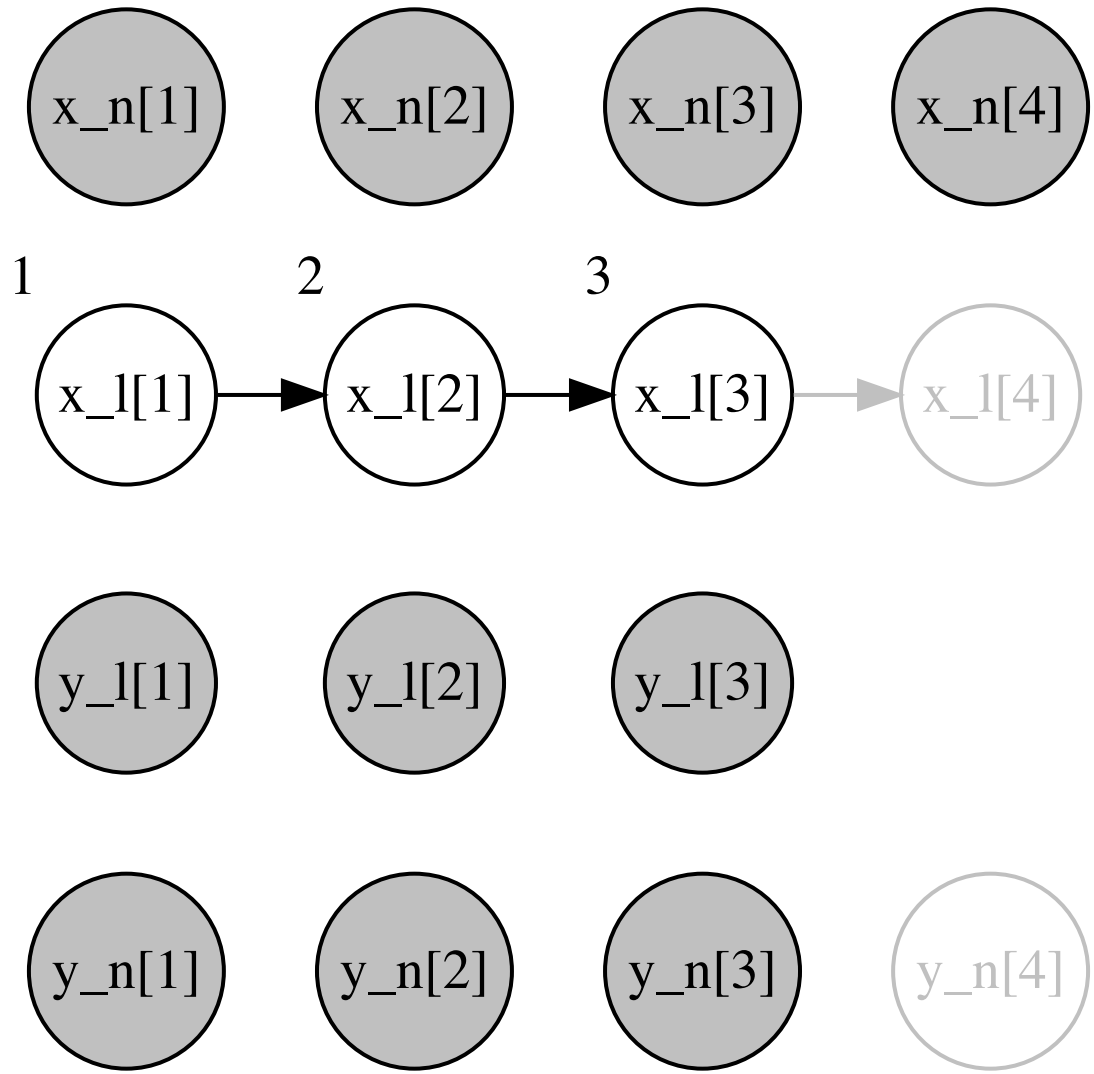
Example #3



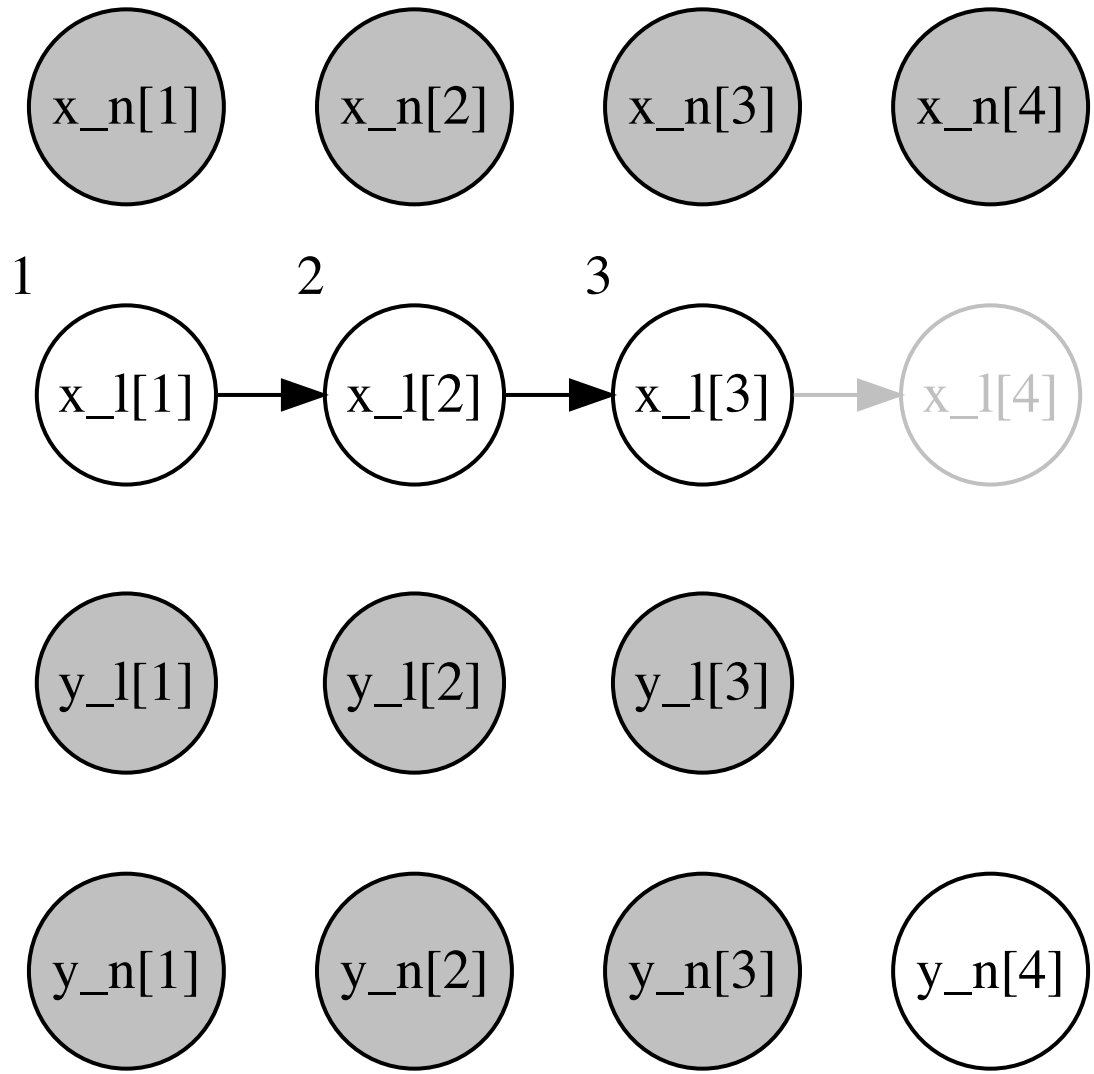
Example #3



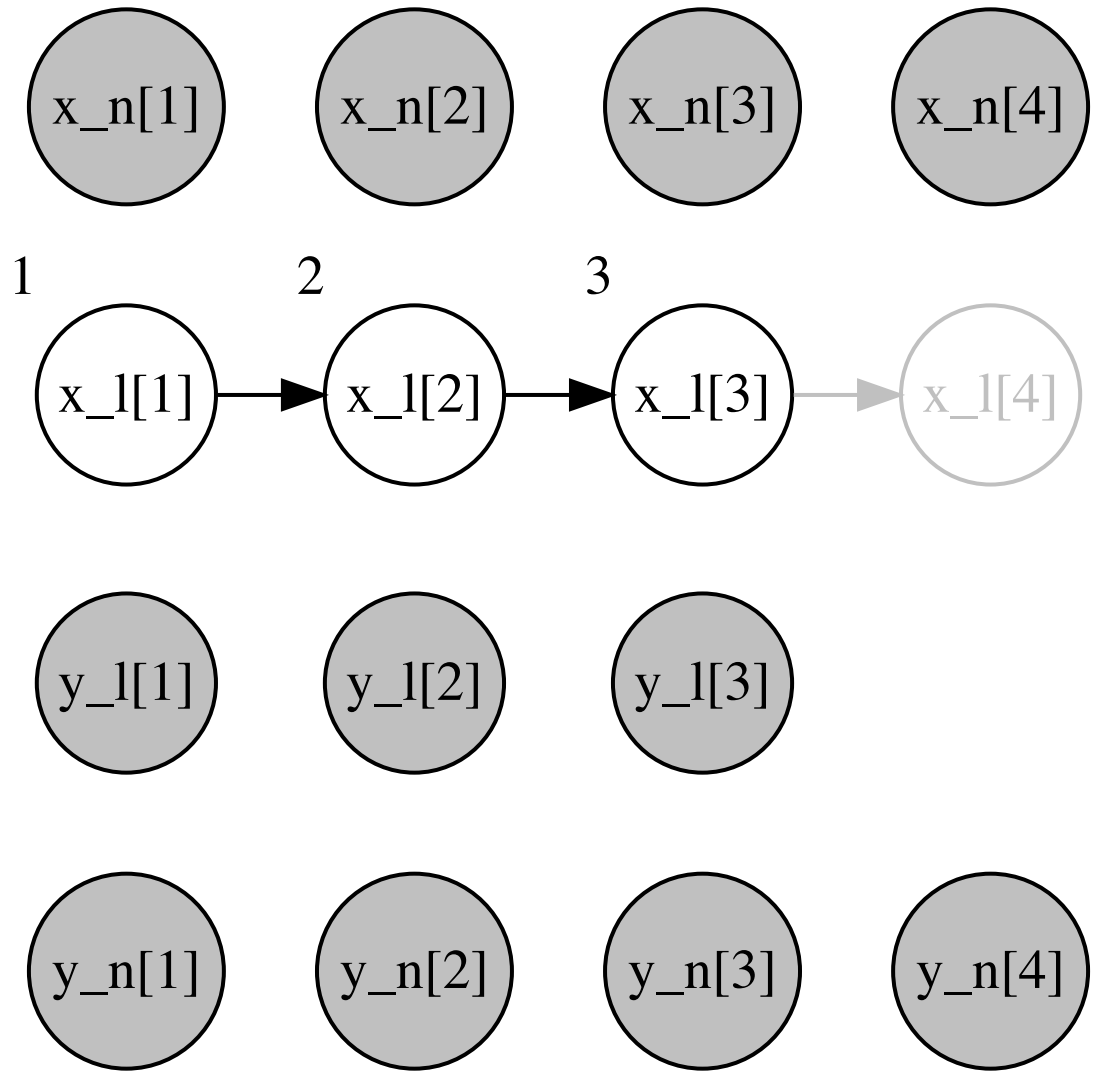
Example #3



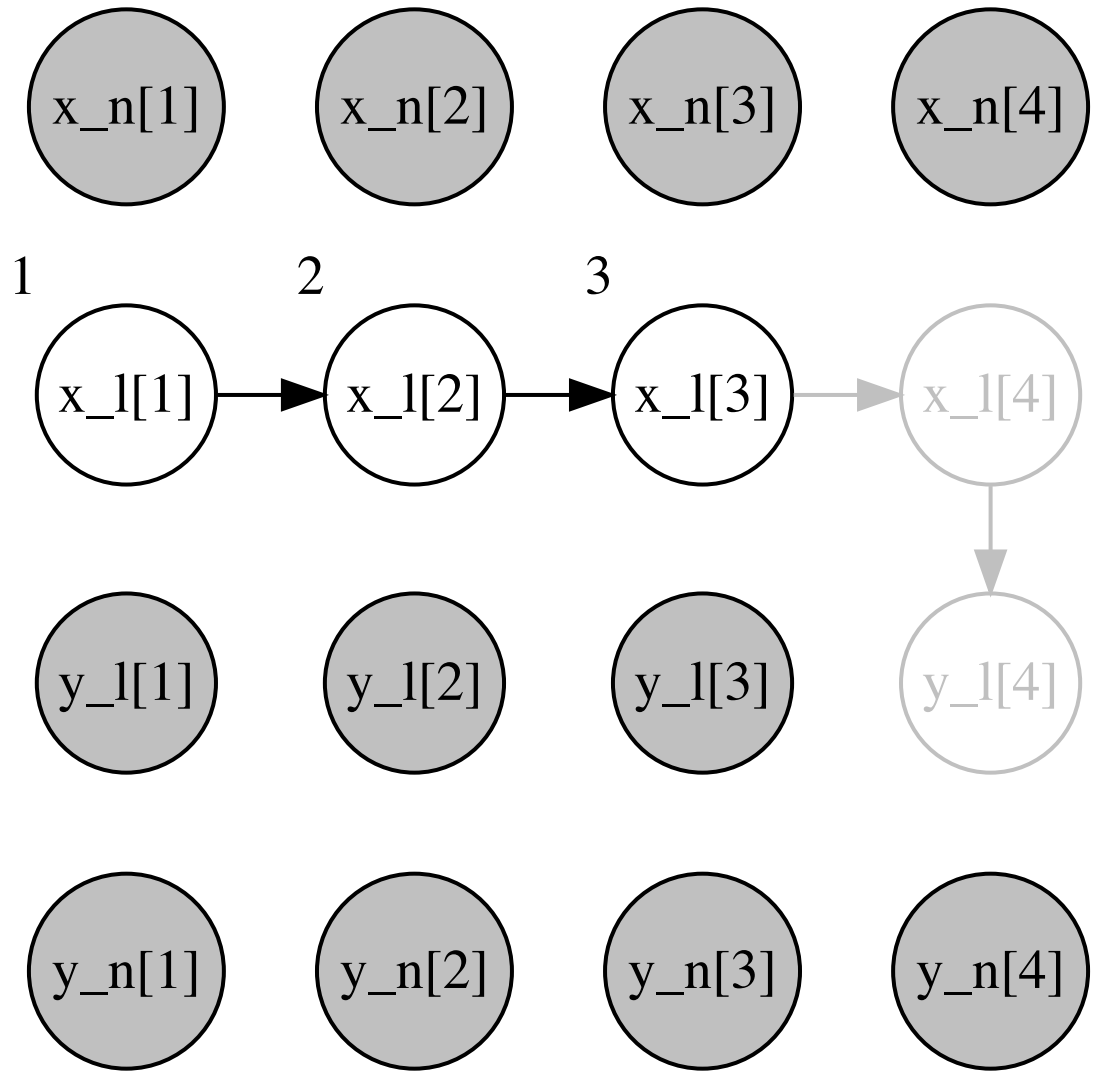
Example #3



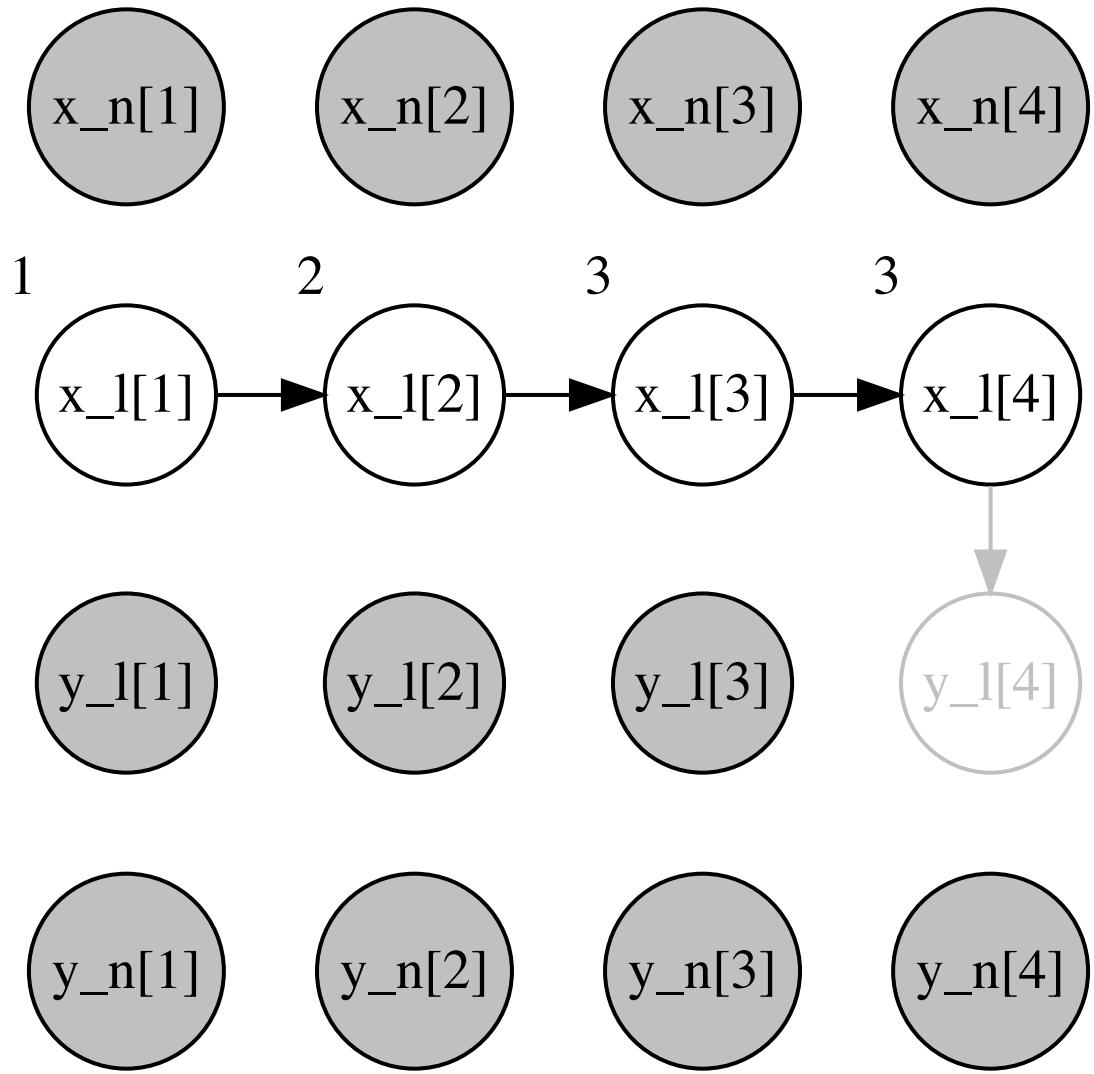
Example #3



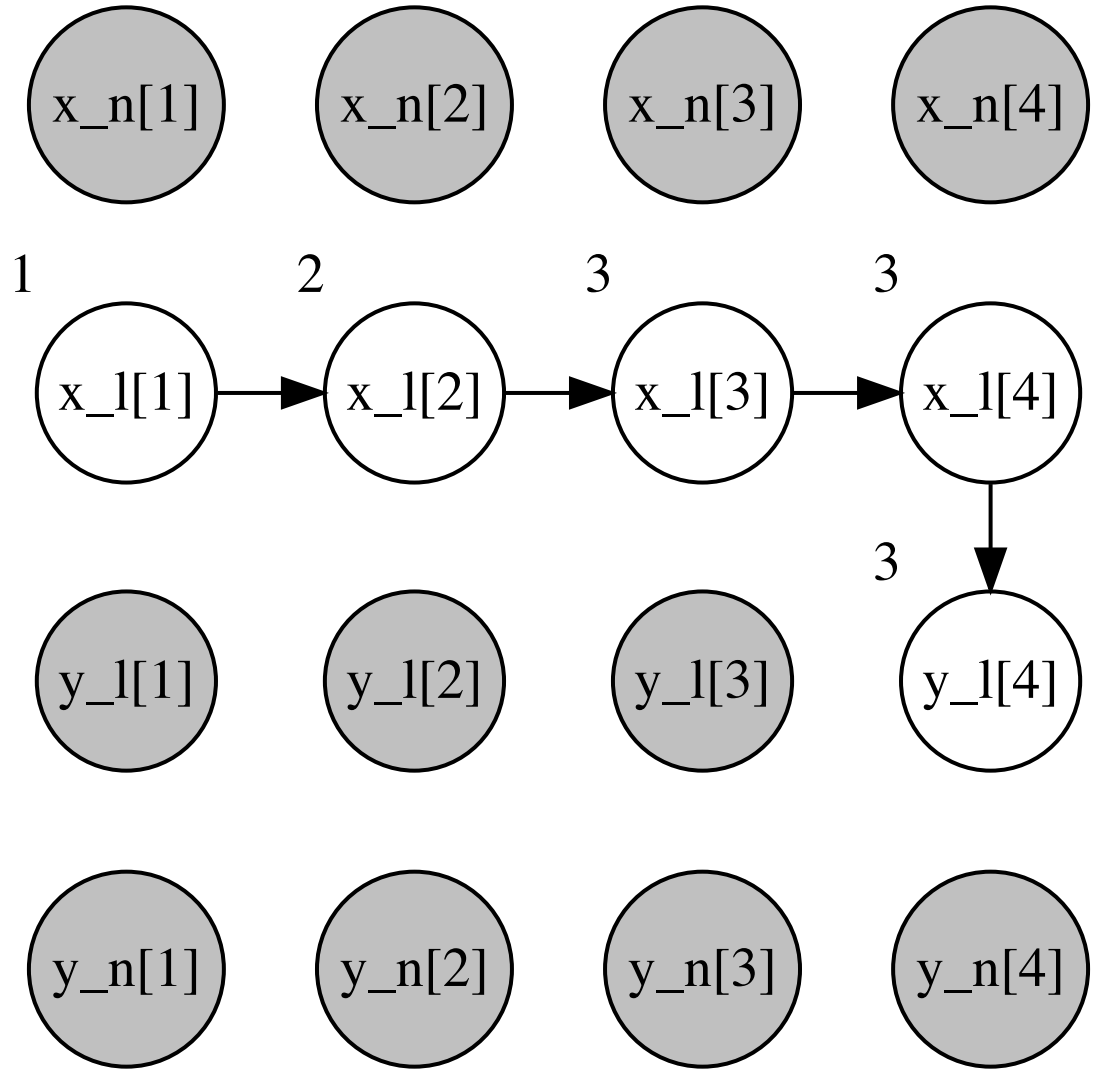
Example #3


x_n[1]
x_n[2]
x_n[3]
x_n[4]
1
2
3
x_l[1]
x_l[2]
x_l[3]
x_l[4]
y_l[1]
y_l[2]
y_l[3]
y_l[4]
y_n[1]
y_n[2]
y_n[3]
y_n[4]

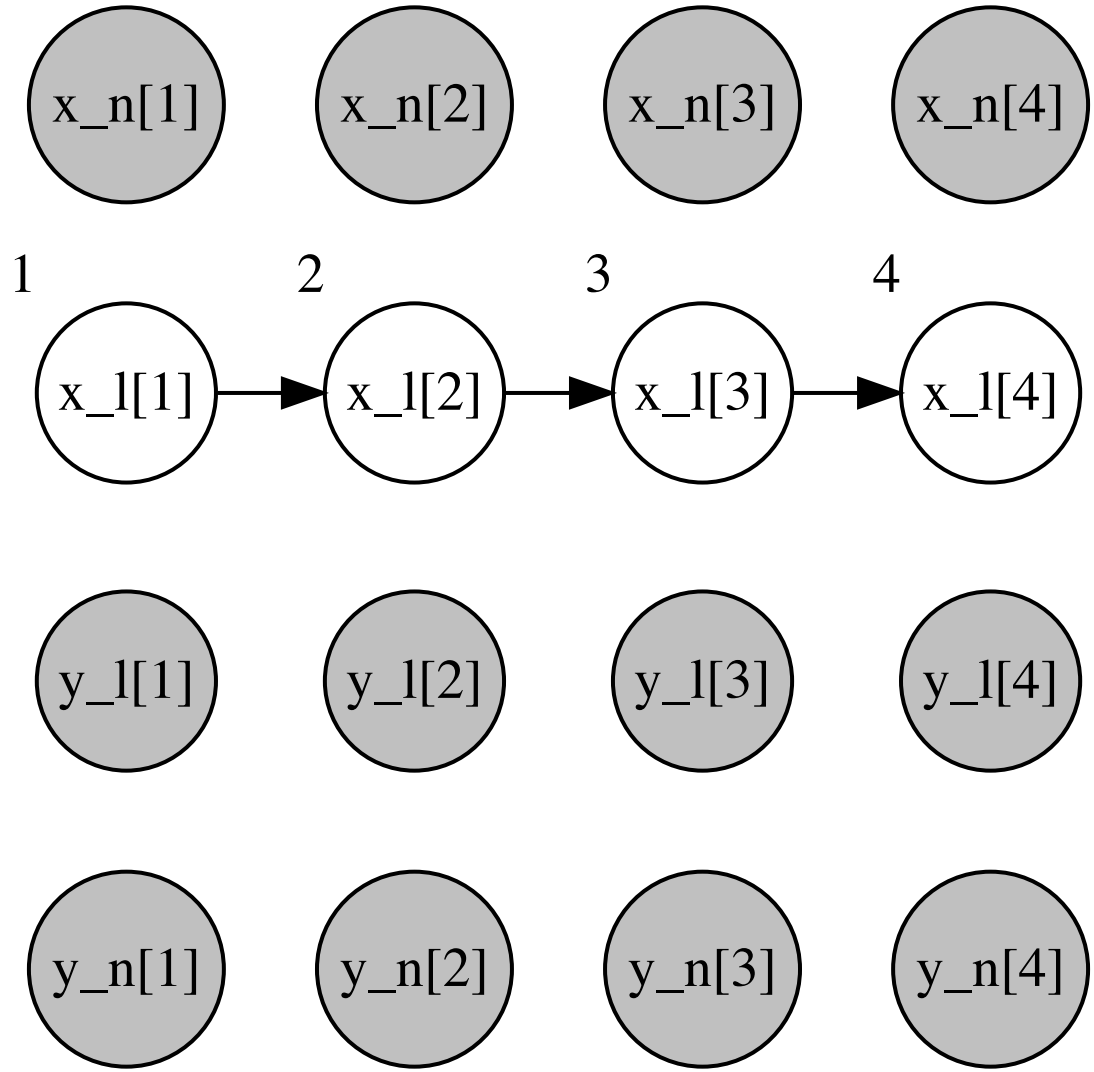
Example #3


x_n[1]
x_n[2]
x_n[3]
x_n[4]
1
2
3
3
x_l[1]
x_l[2]
x_l[3]
x_l[4]
y_l[1]
y_l[2]
y_l[3]
y_l[4]
y_n[1]
y_n[2]
y_n[3]
y_n[4]

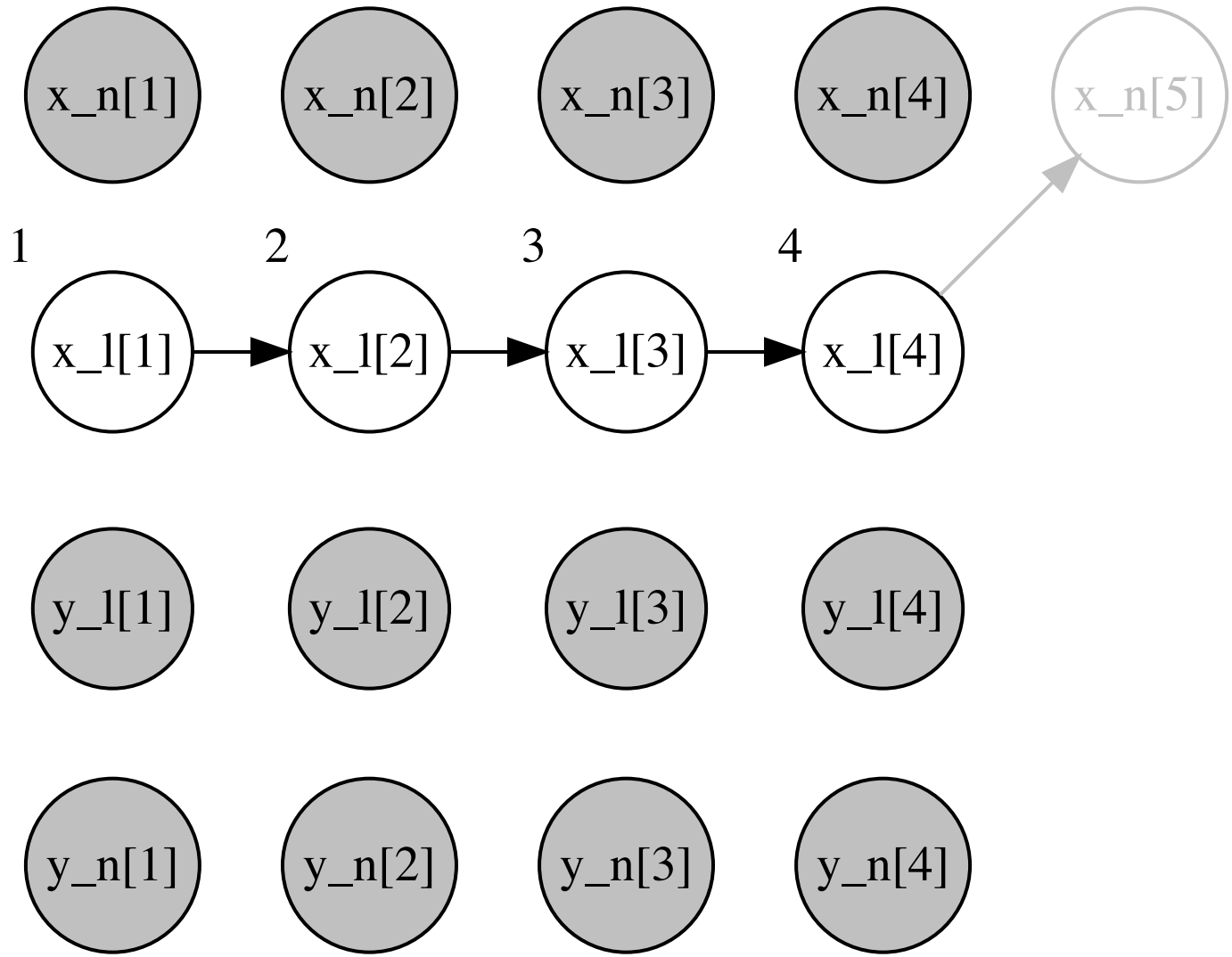
Example #3


x_n[1]
x_n[2]
x_n[3]
x_n[4]
1
2
3
3
x_l[1]
x_l[2]
x_l[3]
x_l[4]
3
y_l[1]
y_l[2]
y_l[3]
y_l[4]
y_n[1]
y_n[2]
y_n[3]
y_n[4]

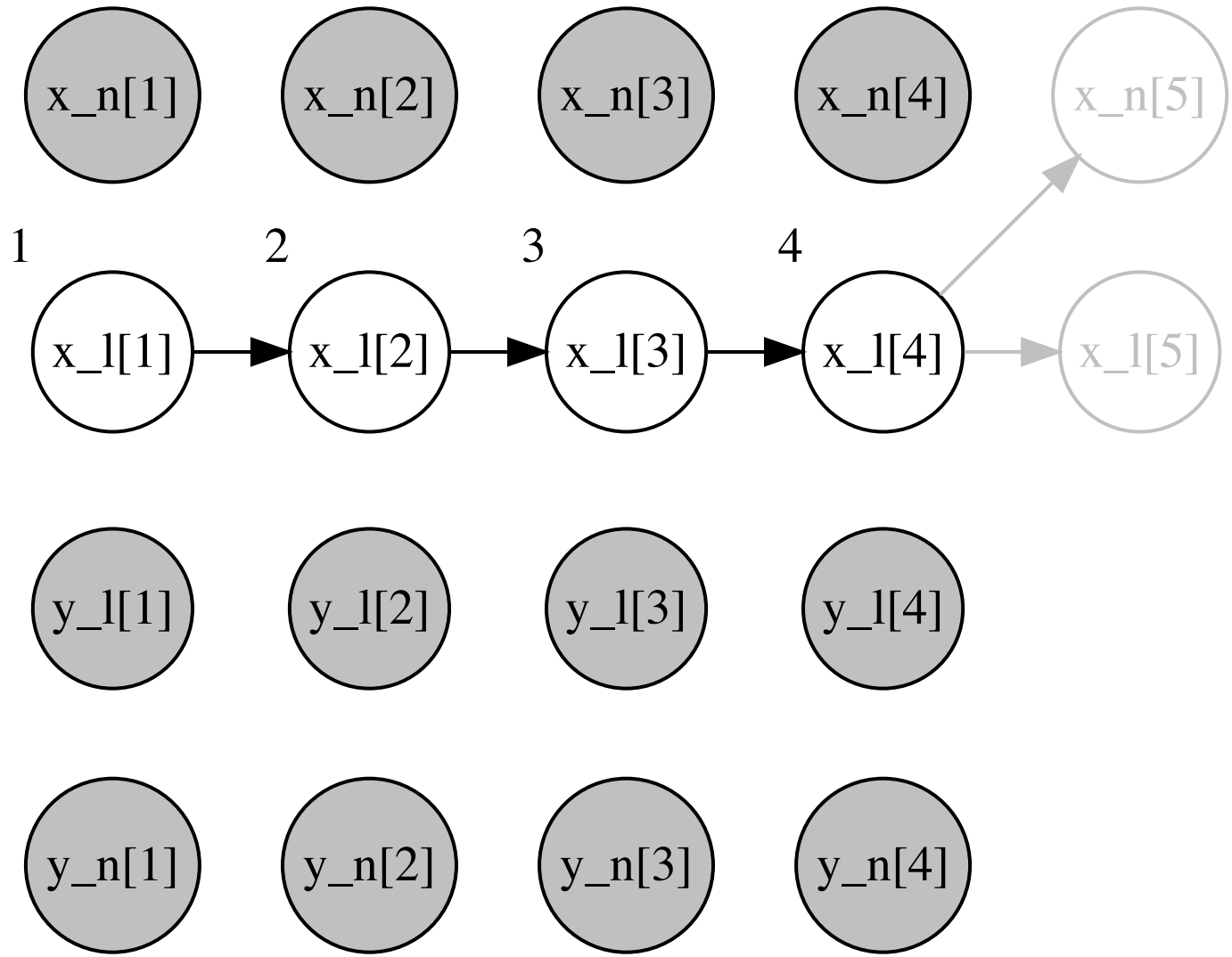
Example #3


x_n[1]
x_n[2]
x_n[3]
x_n[4]
1
2
3
4
x_l[1]
x_l[2]
x_l[3]
x_l[4]
y_l[1]
y_l[2]
y_l[3]
y_l[4]
y_n[1]
y_n[2]
y_n[3]
y_n[4]

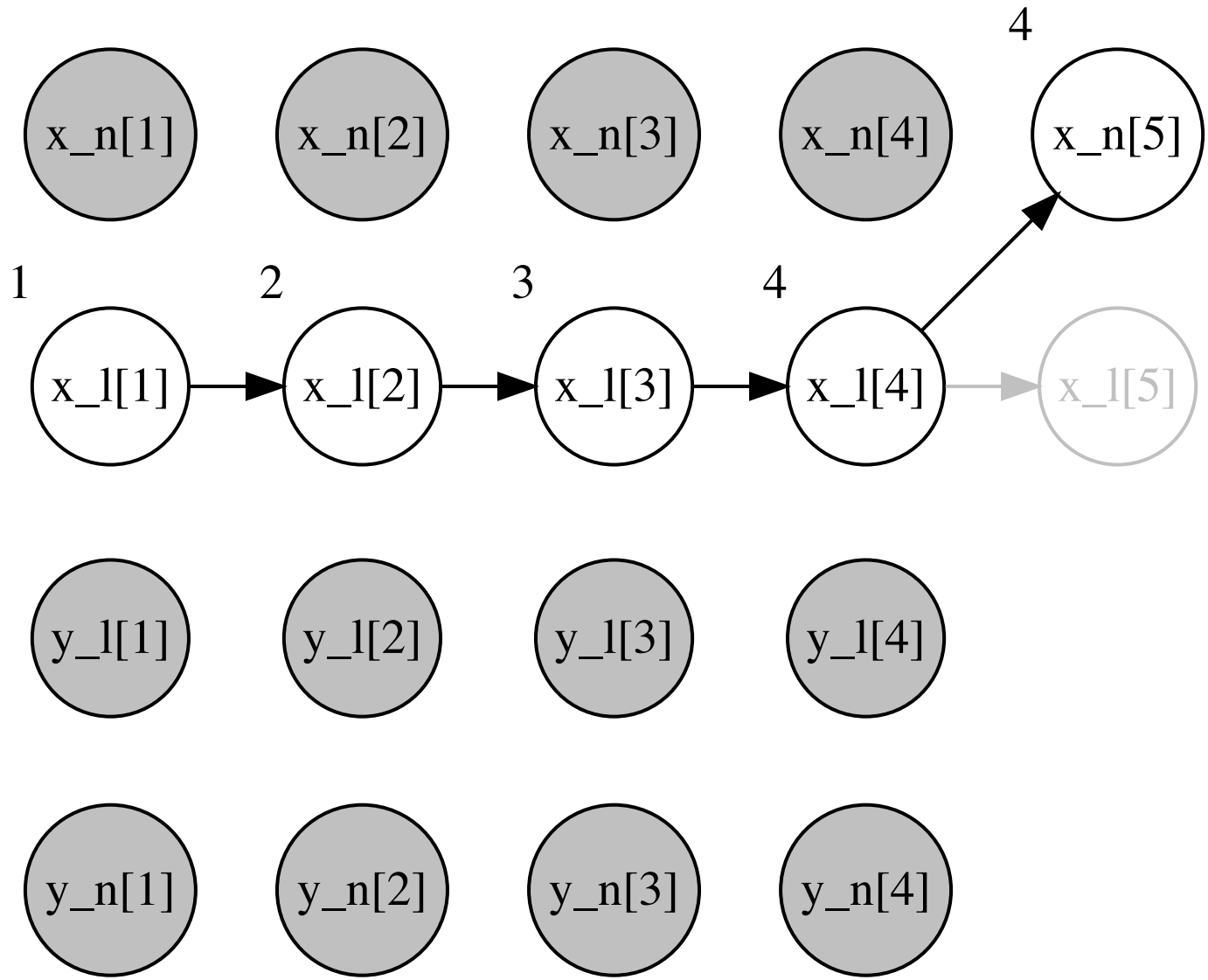
Example #3


x_n[1]
x_n[2]
x_n[3]
x_n[4]
x_n[5]
1
2
3
4
x_l[1]
x_l[2]
x_l[3]
x_l[4]
y_l[1]
y_l[2]
y_l[3]
y_l[4]
y_n[1]
y_n[2]
y_n[3]
y_n[4]

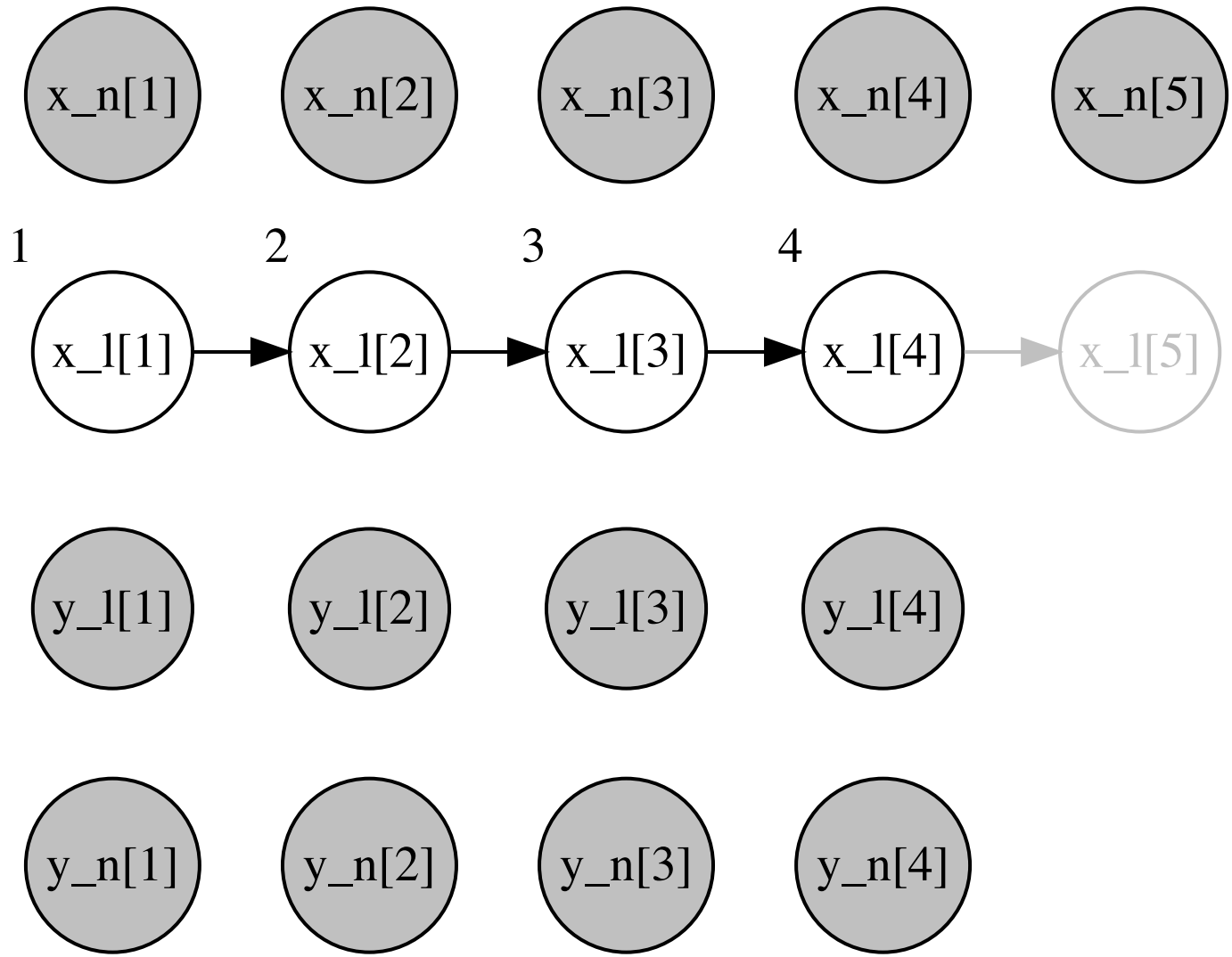
Example #3


x_n[1]
x_n[2]
x_n[3]
x_n[4]
x_n[5]
1
2
3
4
x_l[1]
x_l[2]
x_l[3]
x_l[4]
x_l[5]
y_l[1]
y_l[2]
y_l[3]
y_l[4]
y_n[1]
y_n[2]
y_n[3]
y_n[4]

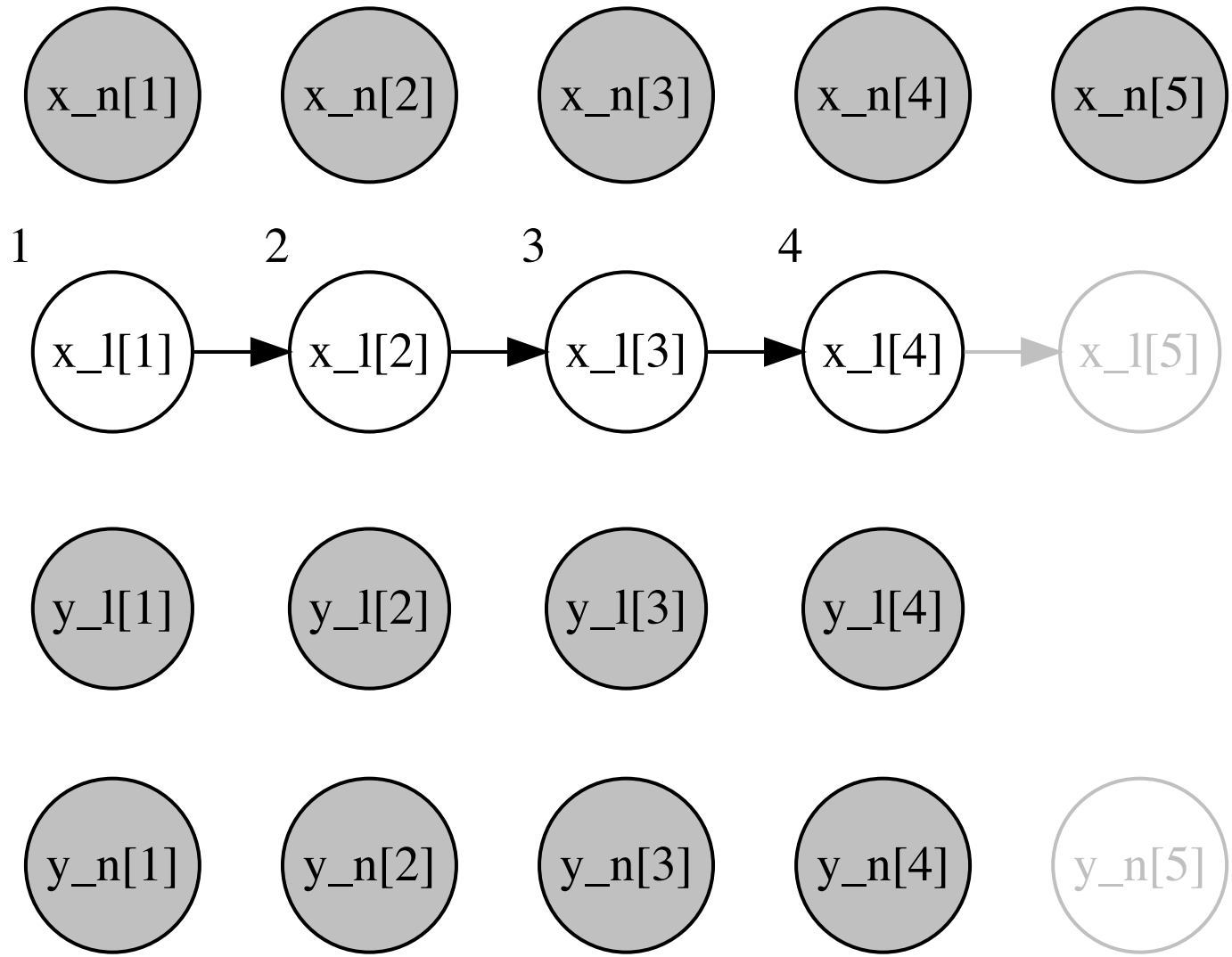
Example #3


4
x_n[1]
x_n[2]
x_n[3]
x_n[4]
x_n[5]
1
2
3
4
x_l[1]
x_l[2]
x_l[3]
x_l[4]
x_l[5]
y_l[1]
y_l[2]
y_l[3]
y_l[4]
y_n[1]
y_n[2]
y_n[3]
y_n[4]

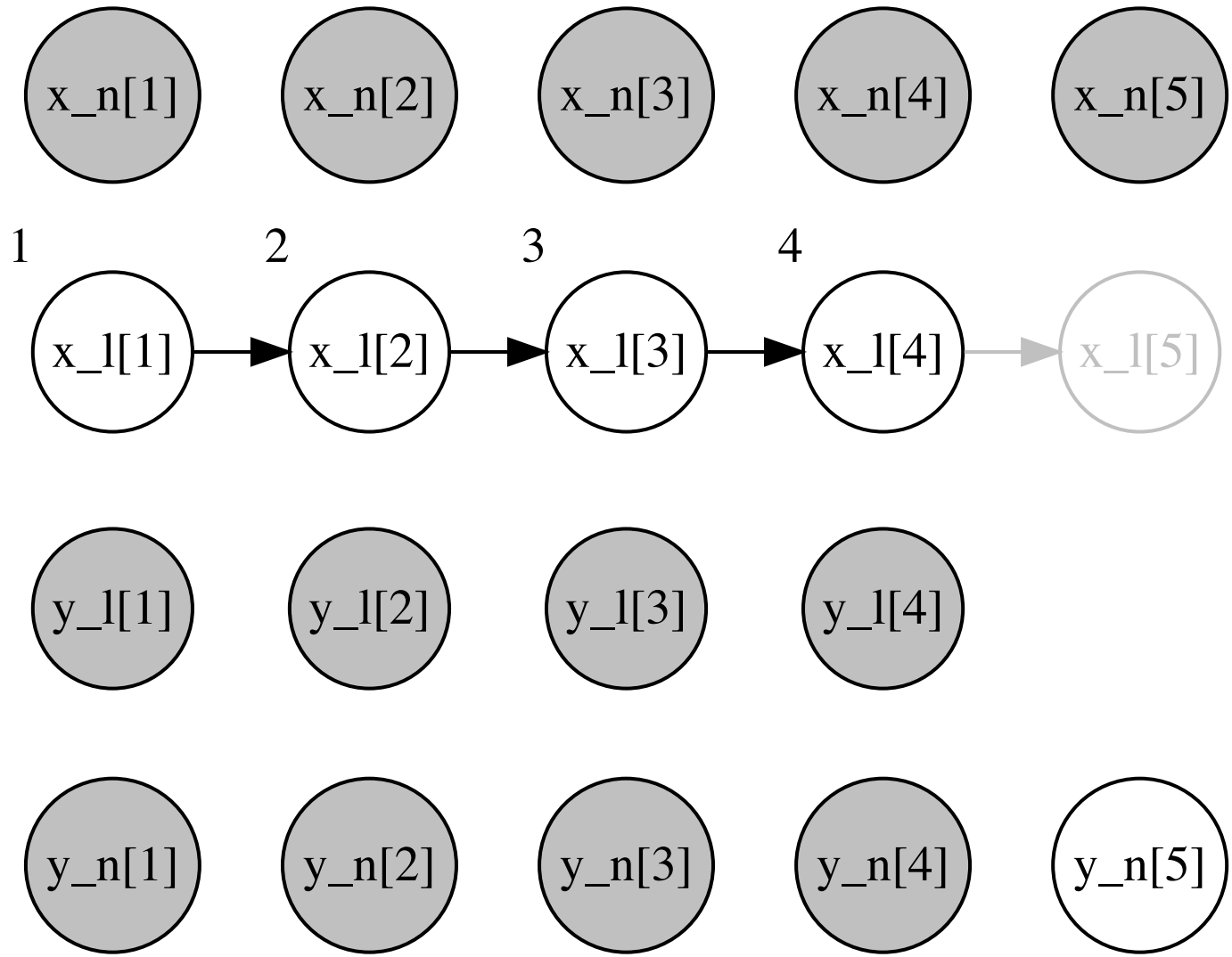
Example #3


x_n[1]
x_n[2]
x_n[3]
x_n[4]
x_n[5]
1
2
3
4
x_l[1]
x_l[2]
x_l[3]
x_l[4]
x_l[5]
y_l[1]
y_l[2]
y_l[3]
y_l[4]
y_n[1]
y_n[2]
y_n[3]
y_n[4]

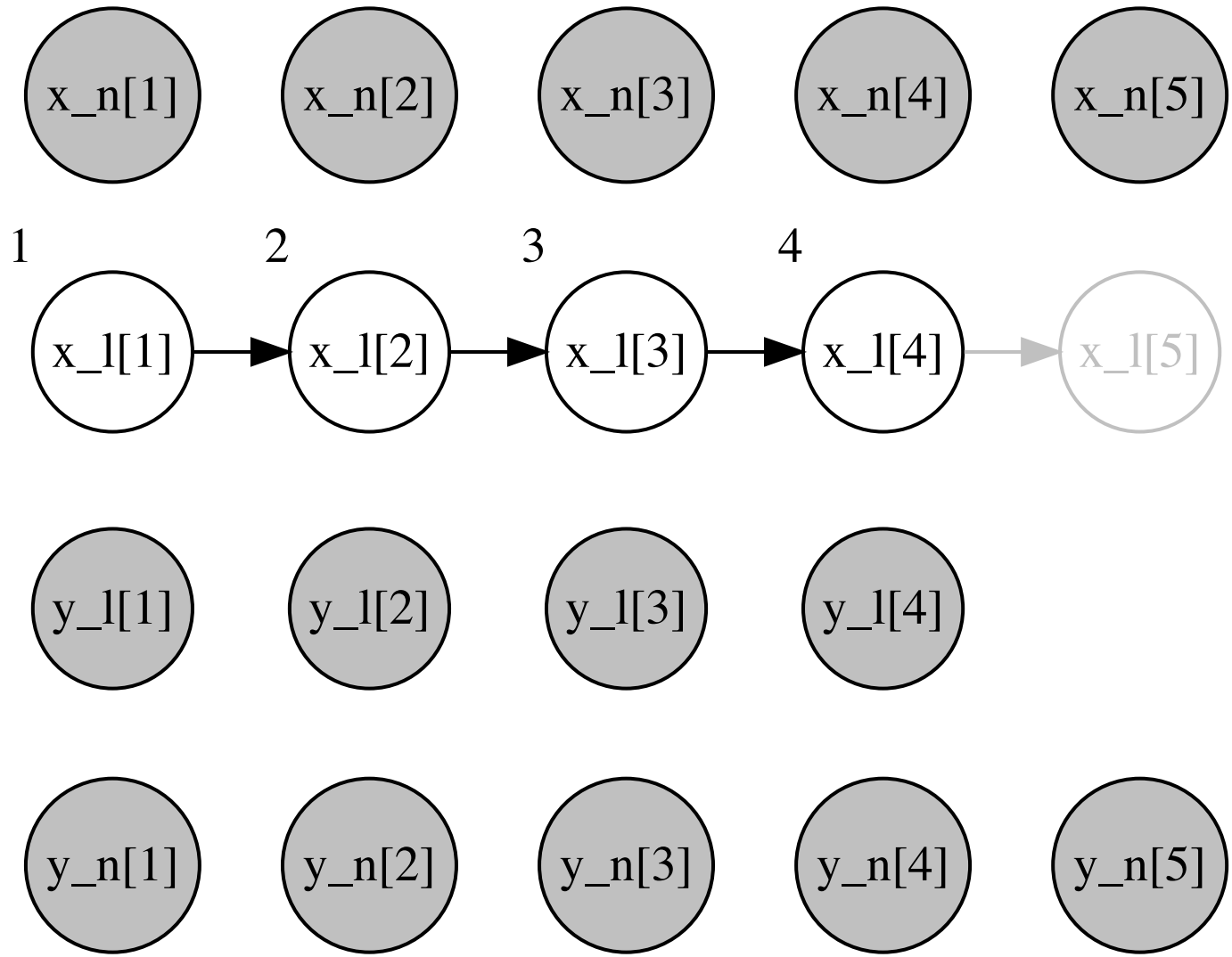
Example #3


x_n[1]
x_n[2]
x_n[3]
x_n[4]
x_n[5]
1
2
3
4
x_l[1]
x_l[2]
x_l[3]
x_l[4]
x_l[5]
y_l[1]
y_l[2]
y_l[3]
y_l[4]
y_n[1]
y_n[2]
y_n[3]
y_n[4]
y_n[5]

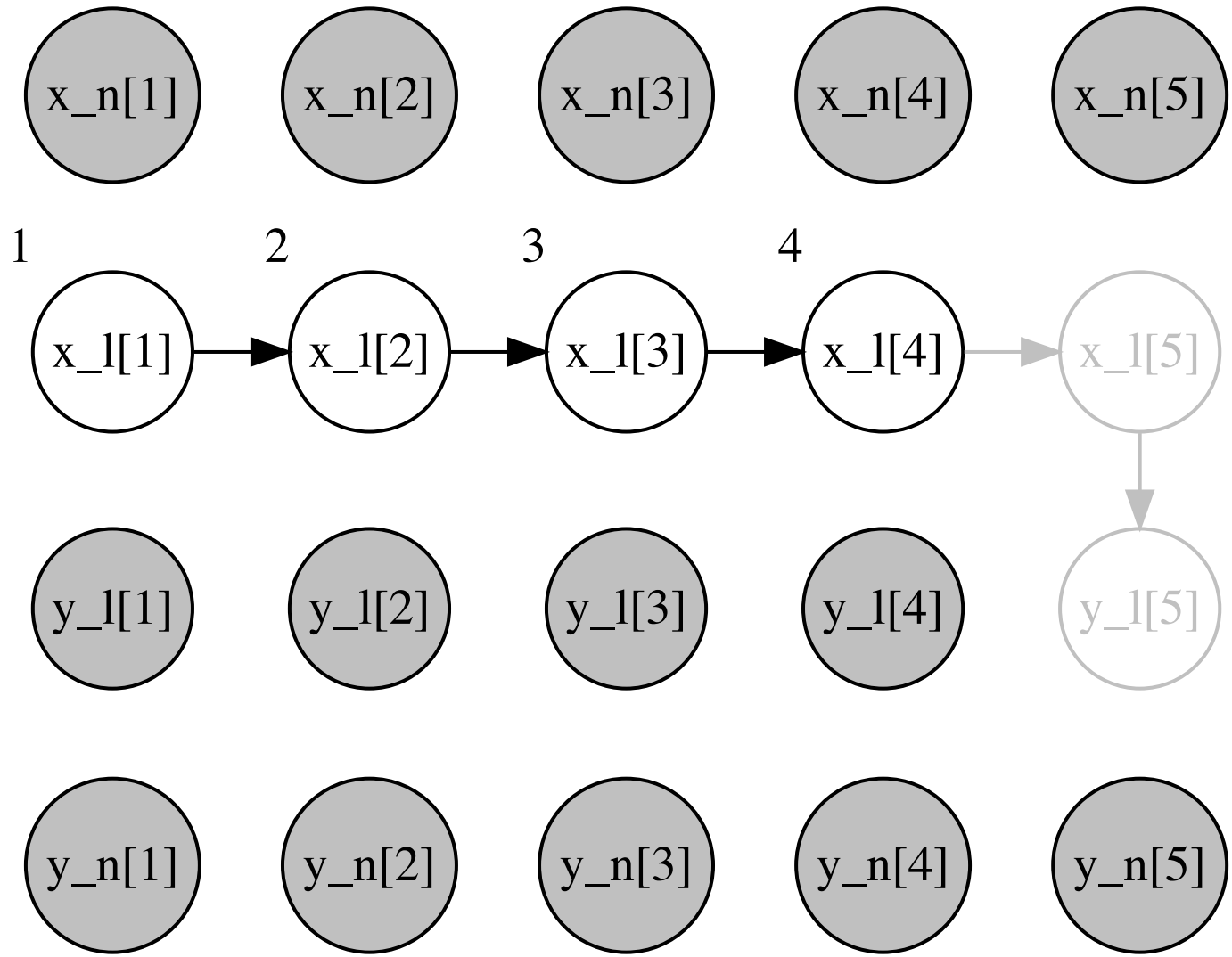
Example #3


x_n[1]
x_n[2]
x_n[3]
x_n[4]
x_n[5]
1
2
3
4
x_l[1]
x_l[2]
x_l[3]
x_l[4]
x_l[5]
y_l[1]
y_l[2]
y_l[3]
y_l[4]
y_n[1]
y_n[2]
y_n[3]
y_n[4]
y_n[5]

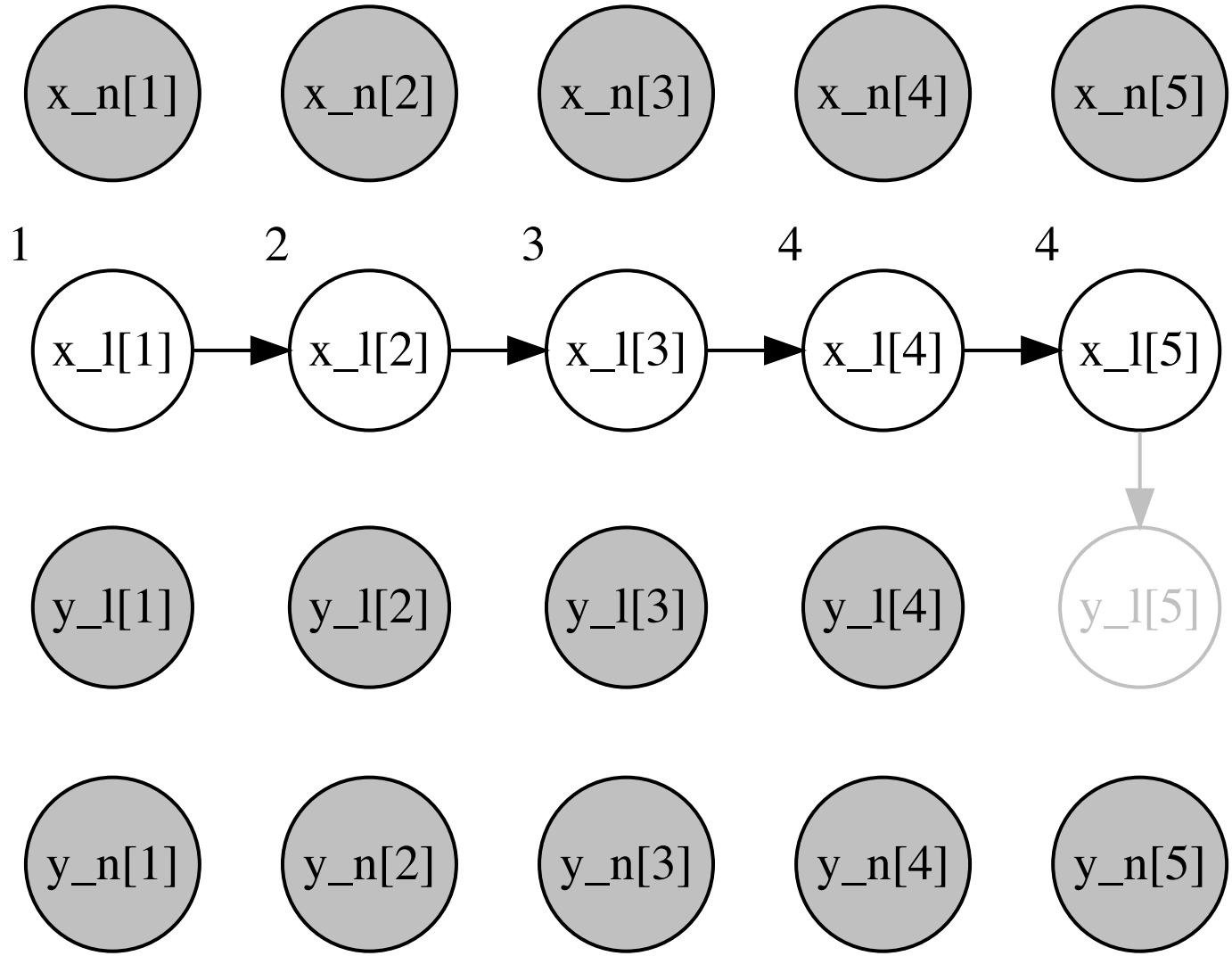
Example #3


x_n[1]
x_n[2]
x_n[3]
x_n[4]
x_n[5]
1
2
3
4
x_l[1]
x_l[2]
x_l[3]
x_l[4]
x_l[5]
y_l[1]
y_l[2]
y_l[3]
y_l[4]
y_n[1]
y_n[2]
y_n[3]
y_n[4]
y_n[5]

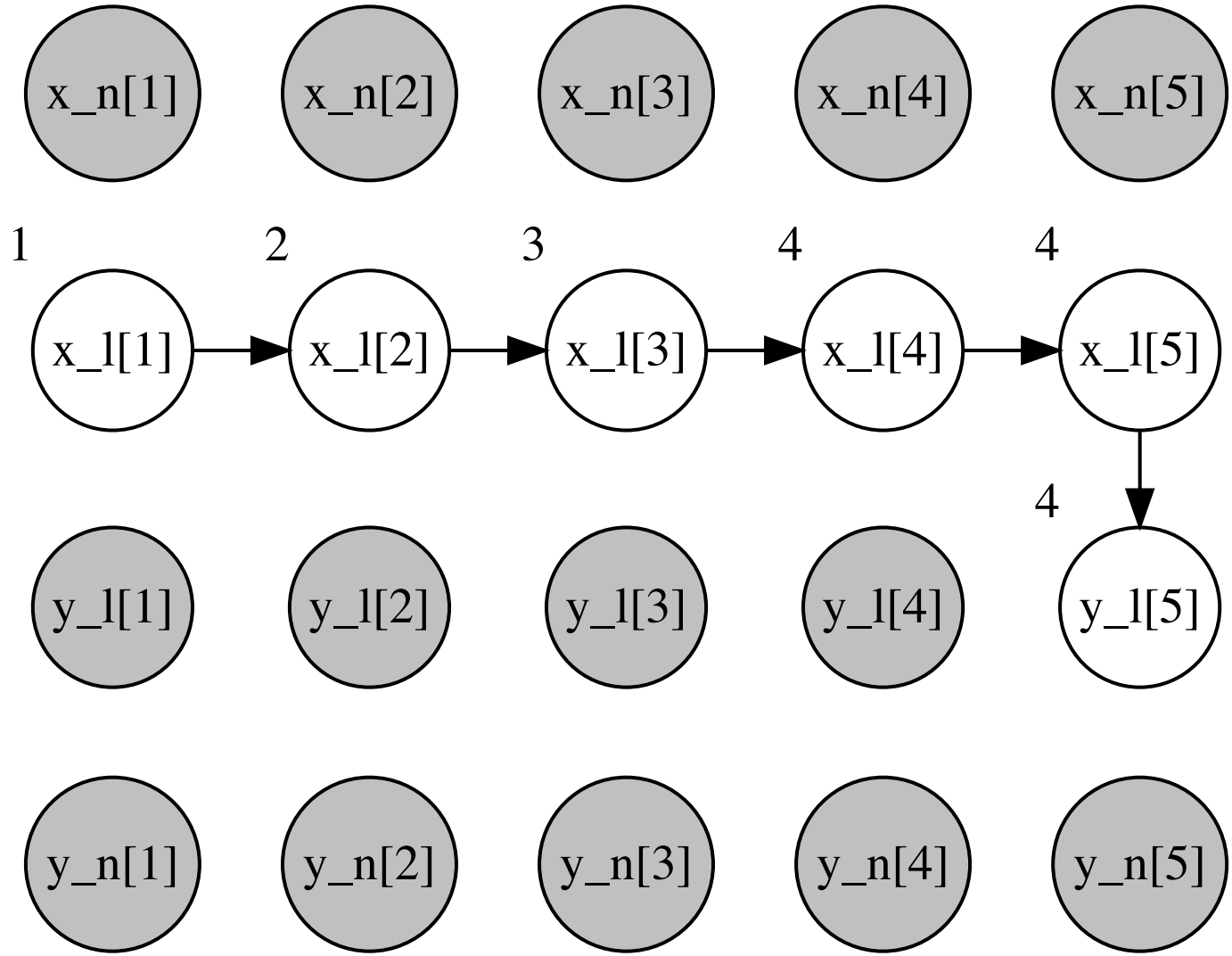
Example #3


x_n[1]
x_n[2]
x_n[3]
x_n[4]
x_n[5]
1
2
3
4
x_l[1]
x_l[2]
x_l[3]
x_l[4]
x_l[5]
y_l[1]
y_l[2]
y_l[3]
y_l[4]
y_l[5]
y_n[1]
y_n[2]
y_n[3]
y_n[4]
y_n[5]

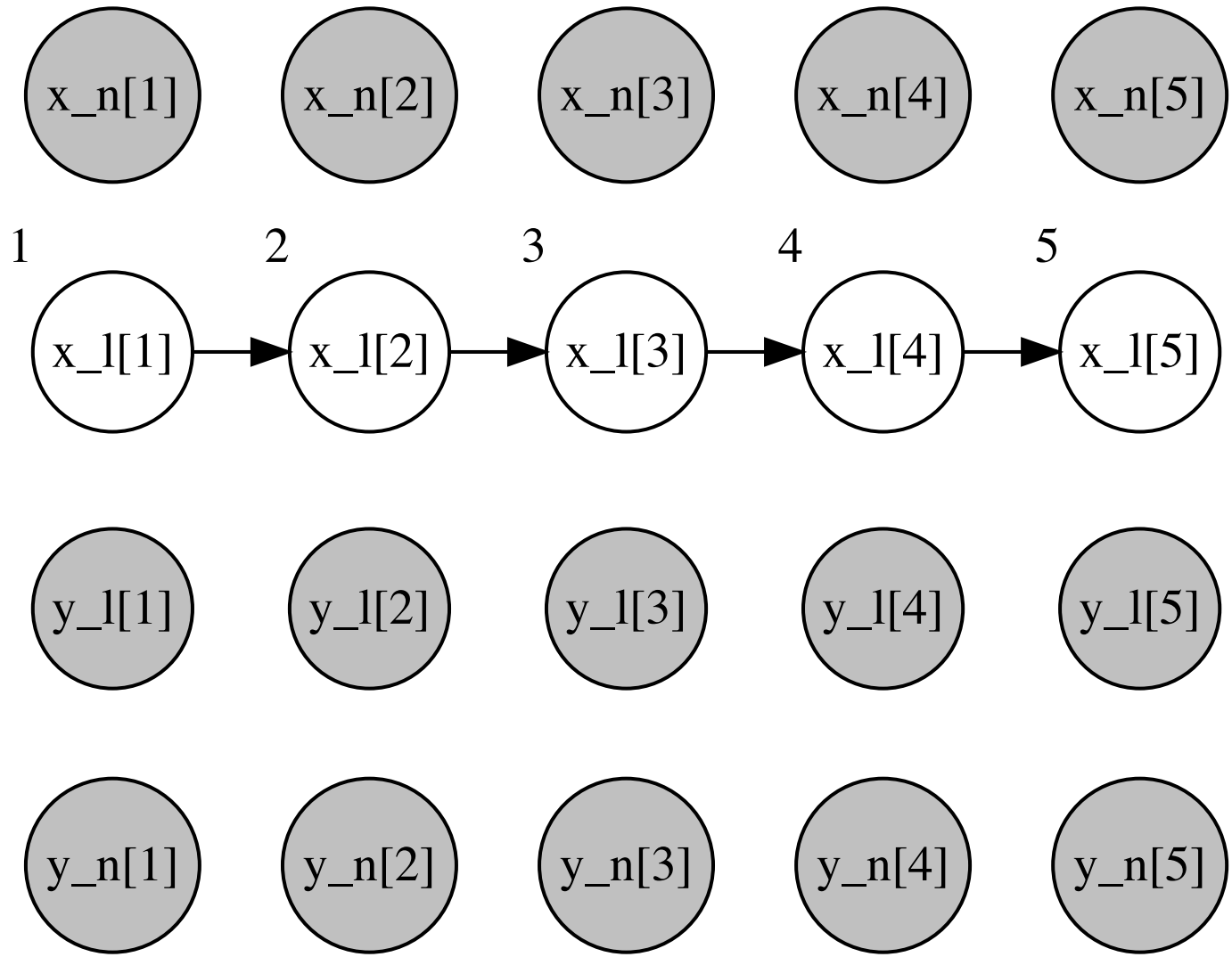
Example #3


x_n[1]
x_n[2]
x_n[3]
x_n[4]
x_n[5]
1
2
3
4
4
x_l[1]
x_l[2]
x_l[3]
x_l[4]
x_l[5]
y_l[1]
y_l[2]
y_l[3]
y_l[4]
y_l[5]
y_n[1]
y_n[2]
y_n[3]
y_n[4]
y_n[5]

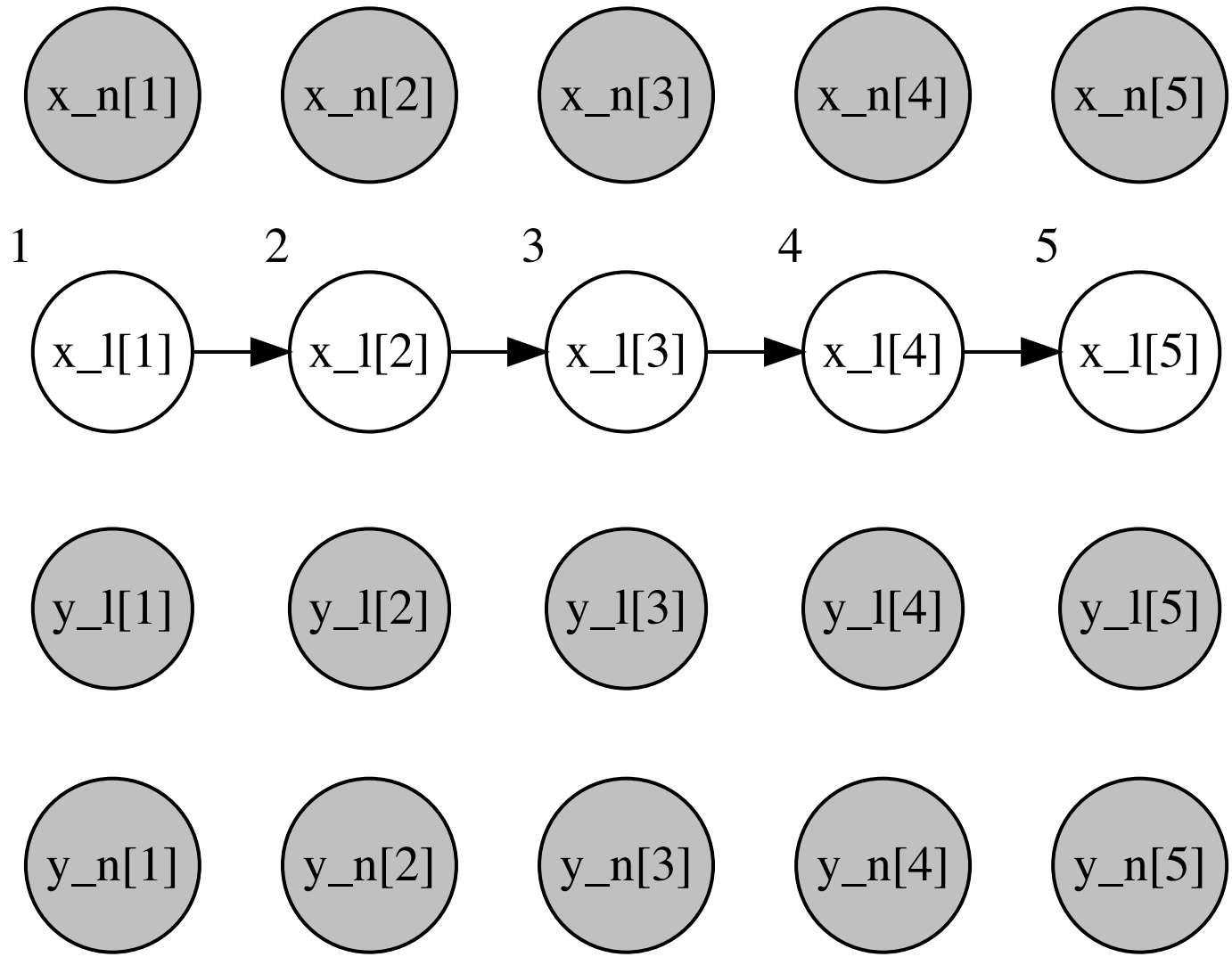
Example #3


x_n[1]
x_n[2]
x_n[3]
x_n[4]
x_n[5]
1
2
3
4
4
x_l[1]
x_l[2]
x_l[3]
x_l[4]
x_l[5]
4
y_l[1]
y_l[2]
y_l[3]
y_l[4]
y_l[5]
y_n[1]
y_n[2]
y_n[3]
y_n[4]
y_n[5]

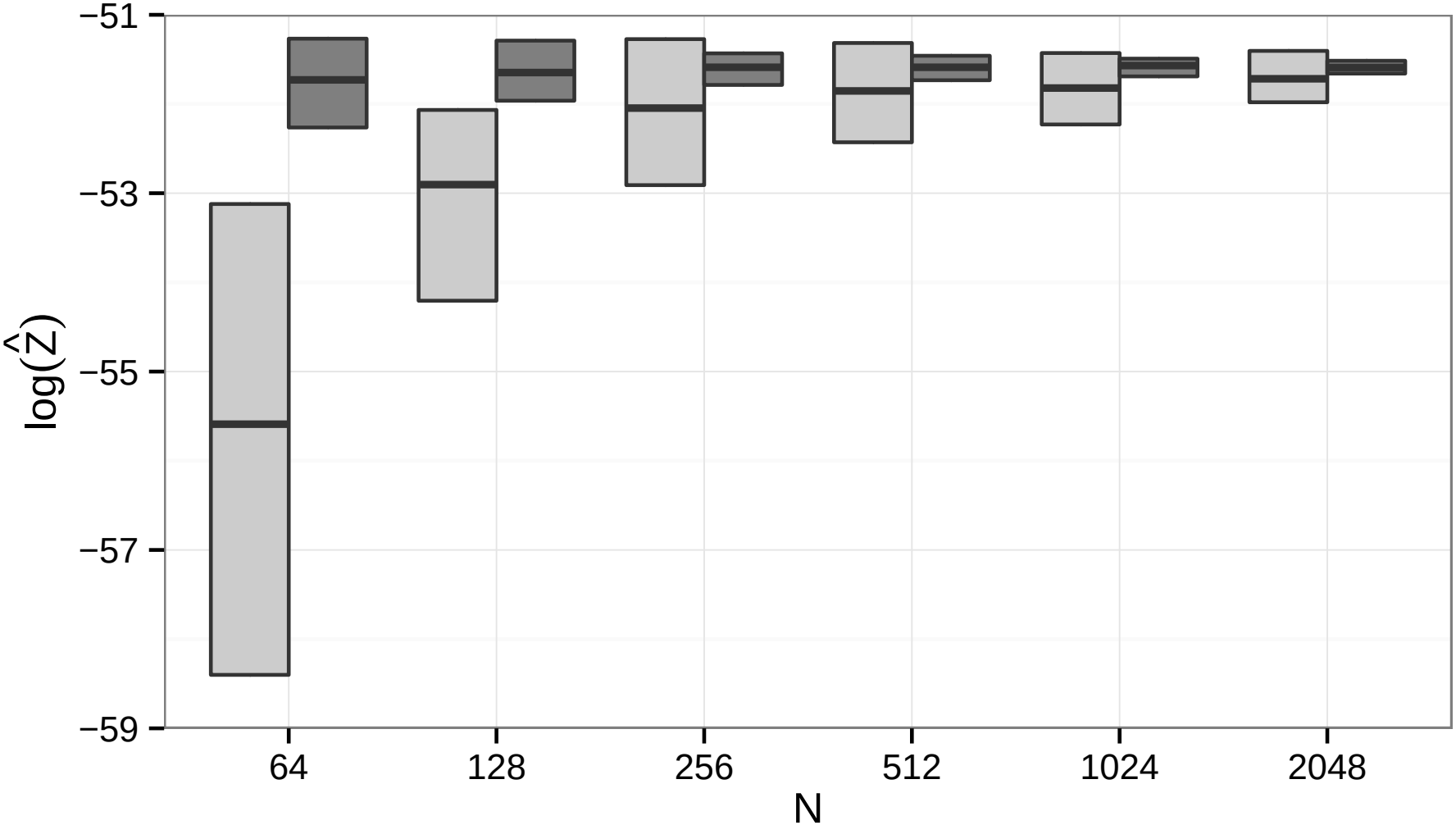
Example #3


x_n[1]
x_n[2]
x_n[3]
x_n[4]
x_n[5]
1
2
3
4
5
x_l[1]
x_l[2]
x_l[3]
x_l[4]
x_l[5]
y_l[1]
y_l[2]
y_l[3]
y_l[4]
y_l[5]
y_n[1]
y_n[2]
y_n[3]
y_n[4]
y_n[5]

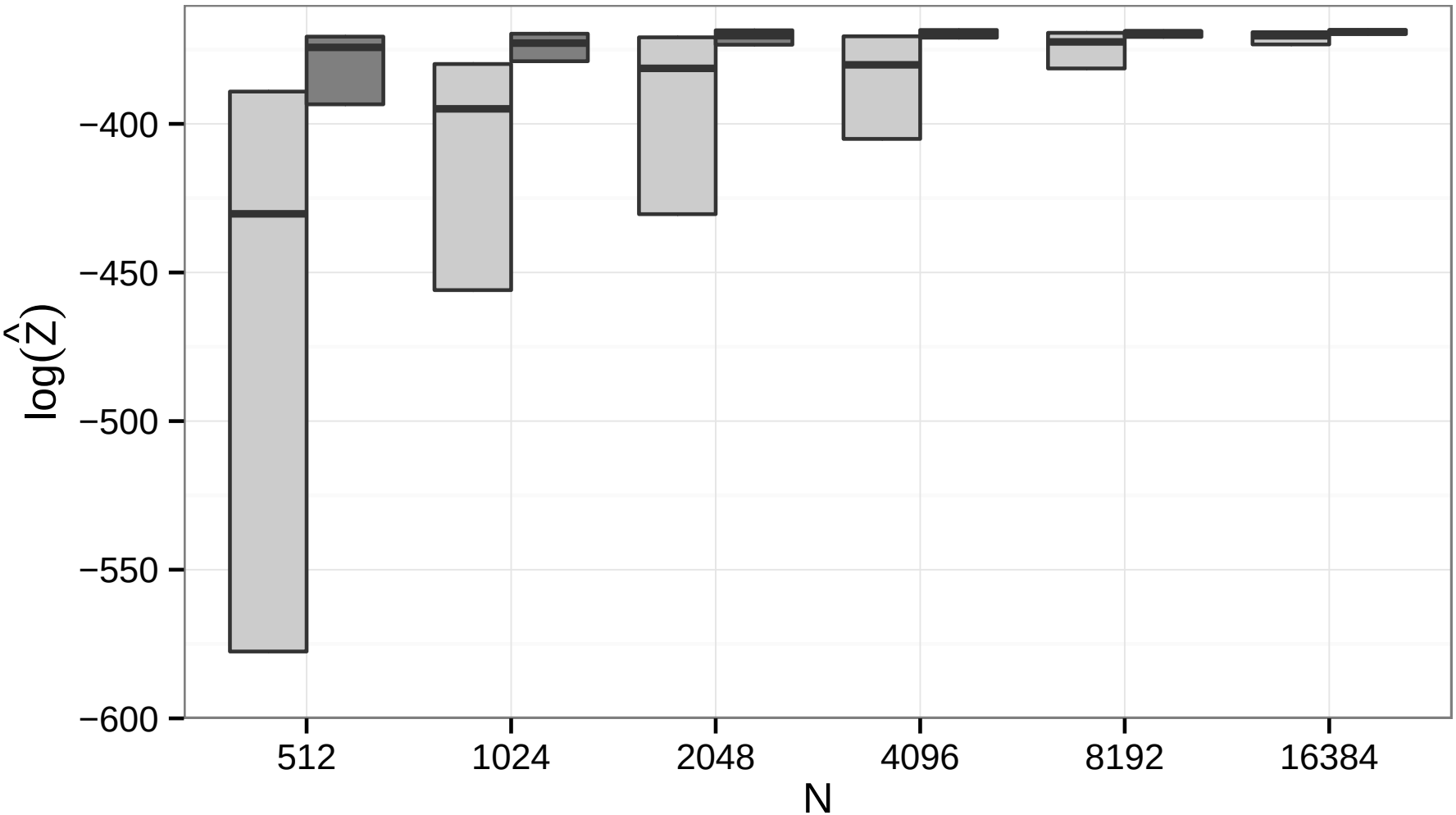
Example #3: Rao–Blackwellized Particle Filter



Example #3: Rao–Blackwellized Particle Filter



Example #4: Vector-Borne Disease Model



Limitation #1

The graph kept alongside the running probabilistic program must be a forest of disjoint trees.

- For more general graphs (e.g. with multivariate normals), multiple nodes can be grouped into "supernodes", where the supernodes form trees (c.f. the junction tree algorithm, belief propagation on factor graphs).
- Or, one or more analytical relationships can simply be ignored, so that some variables are sampled eagerly.
- But delayed sampling still produces correctly-weighted importance samples, it just misses an opportunity for variance reduction.

Limitation #2

Delayed sampling arbitrarily changes the order of random variables that are sampled, and their interleaving amongst random variables that are observed, but does not reorder random variables that are observed. This means that some analytical relationships are not visible.

- Reordering of observations is a possible extension, but requires more complex graph operations.
- The order of observations given by the programmer is likely to be a reasonable one.
- Again, delayed sampling still produces correctly-weighted importance samples, it just misses an opportunity for variance reduction.

Summary

- Delayed sampling can automate variance reduction techniques such as Rao–Blackwellization, locally-optimal proposals, and variable elimination.

- Mostly automatic, with little modification required to program code.

- It has been implemented in Anglican and Birch (birch-lang.org), a new universal probabilistic programming language.