# Graphical Models

## Graphical Models

Graphical models provide a means of depicting the dependencies among parameters in probabilistic models.

The “graphical” part of the name graphical models has to do with their basis on graphs, and not anything to do with drawing.

But it is convenient that graphical models can also be drawn, to clearly depict the structure of the model.

# Graphical Model Syntax

***Symbol***

$$\boxed{x}$$

***Type***

Constant Node

***Description***

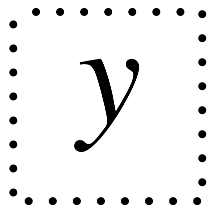
Constant nodes are like standard variables in a programming language. They are assigned fixed values.

***Rev Example***

```
x <- 2.3
```

# Graphical Model Syntax

## ***Symbol***



## ***Type***

Deterministic Node

## ***Description***

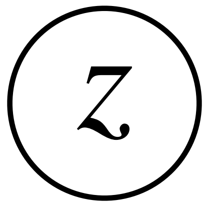
Deterministic nodes depend on the values of other nodes, but in a deterministic (fixed) way. They have no probability distribution intrinsically associated with them.

## ***Rev Example***

```
y  :=  2 * x
```

## Graphical Model Syntax

### *Symbol*



### *Type*

Stochastic Node

### *Description*

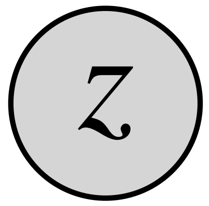
Stochastic nodes represent random variables and take on values according to some probability distribution. These distributions may have parameters defined in other nodes in the model graph. The type of distribution associated with a node is often not written explicitly when a model is drawn, but it must be specified.

### *Rev Example*

```
z  ~  dnBernoulli(0.5)
```

# Graphical Model Syntax

## *Symbol*



## *Type*

Clamped Stochastic Node

## *Description*

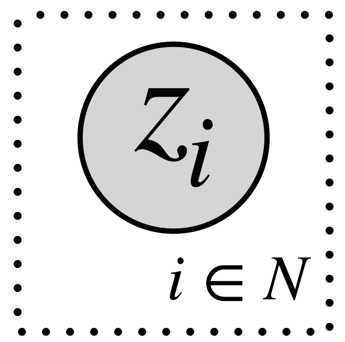
Data can be viewed as the observed outcome of a stochastic node. Clamping involves assigning a set of observations to an associated stochastic node. Clamping data allows the values of other parameters in the model to be inferred.

## *Rev Example*

```
z  ~  dnBernoulli(0.5)
      z.clamp(1)
```

# Graphical Model Syntax

## *Symbol*



## *Type*

Plate

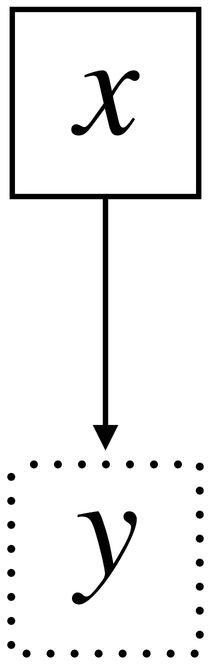
## *Description*

Plates simply represent repetition and make it easier to depict graphical models with a repetitive structure. For instance, the example above shows *N* instances of the clamped variable, *z*, with each instance indexed by *i*. Plates can be used with any type of node, but are commonly used with clamped nodes representing data.

## *Rev Example*

```
N <- 10
for (i in 1:N){
     z  ~  dnBernoulli(0.5)
     z.clamp(1)}
```

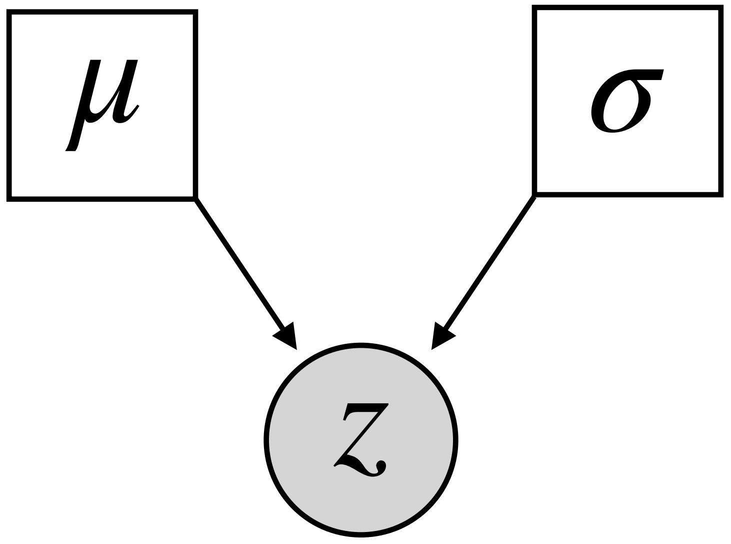
## Super Simple Models



```
x <- 2
y := x / 4
```

What's the structure of this model?
What would happen if x was set to 16?

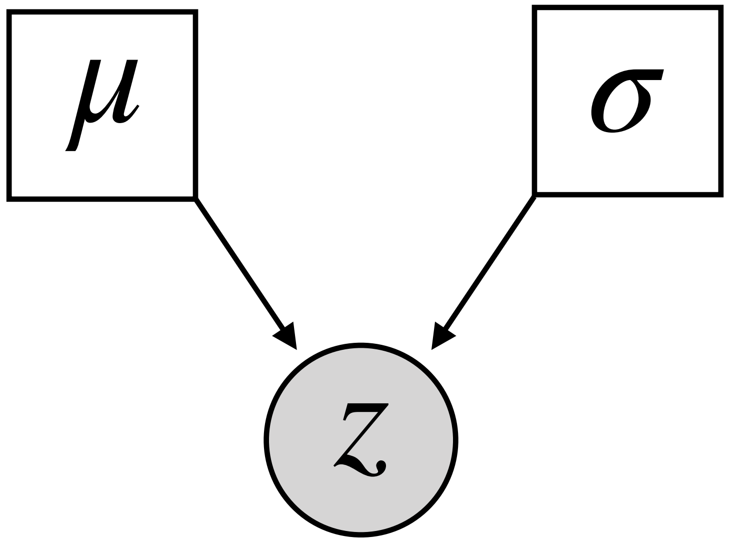
# Super Simple Models



```
m <- 10
s <- 1
z ~ dnNormal(m,s)
z.clamp(9.6)
```

What's the structure of this model?
What can we infer about this model?

# Super Simple Models



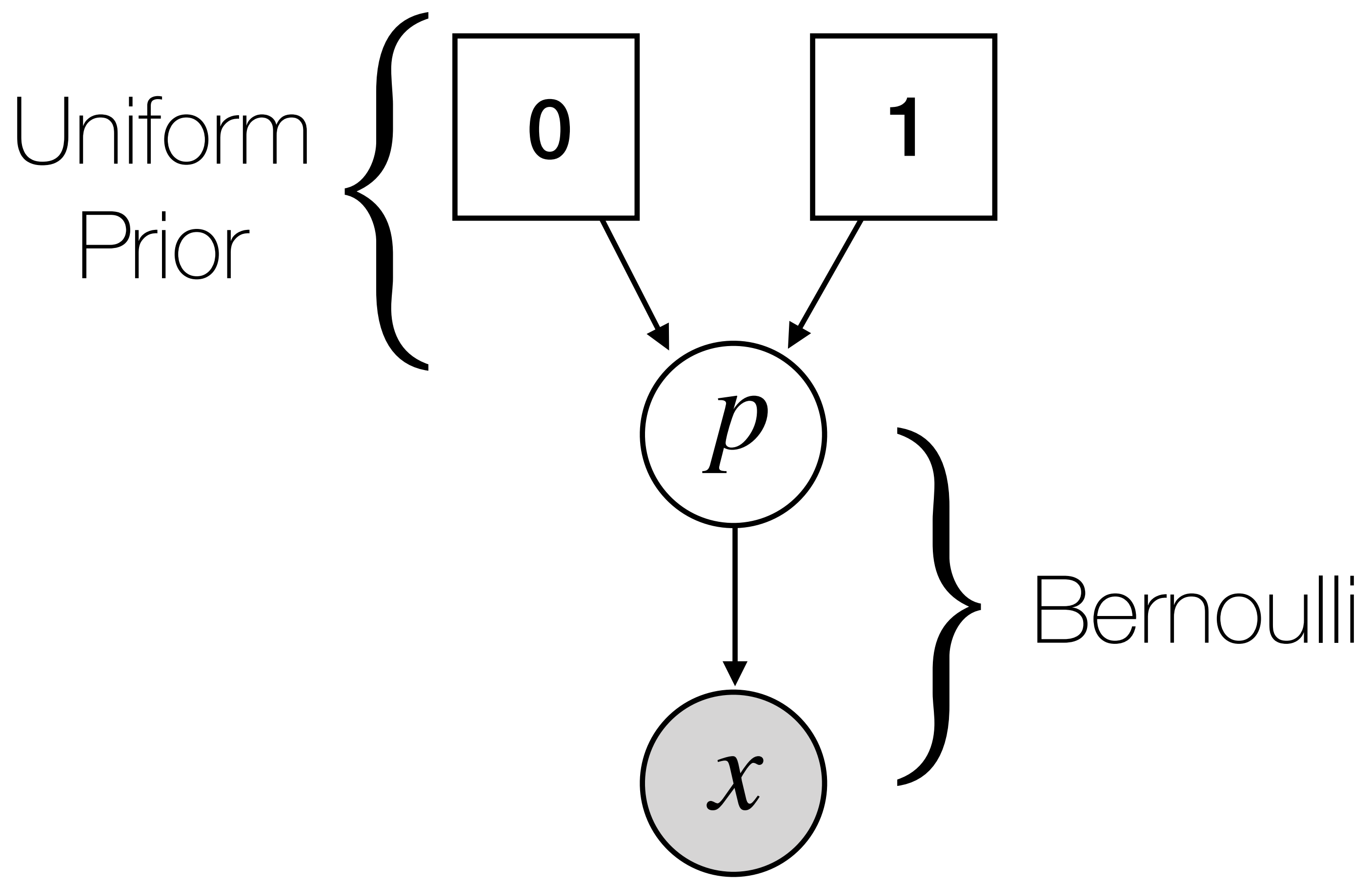
```
m <- 10
s <- 1
z ~ dnNormal(m,s)
z.clamp(9.6)
```

What's the structure of this model?

What can we infer about this model?

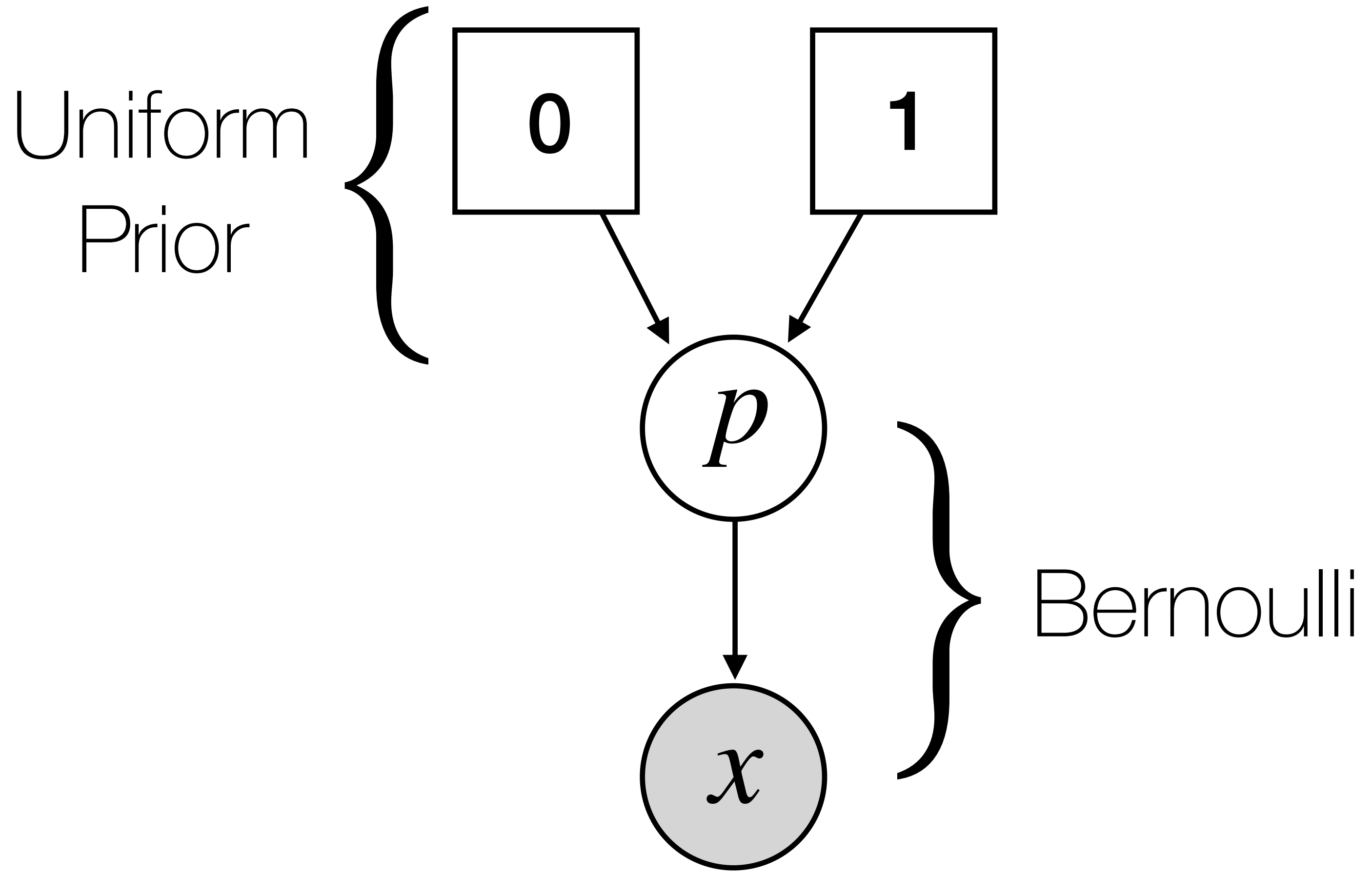
**Nothing to infer! Only nodes are constant or clamped.**

## Single Bernoulli Trial



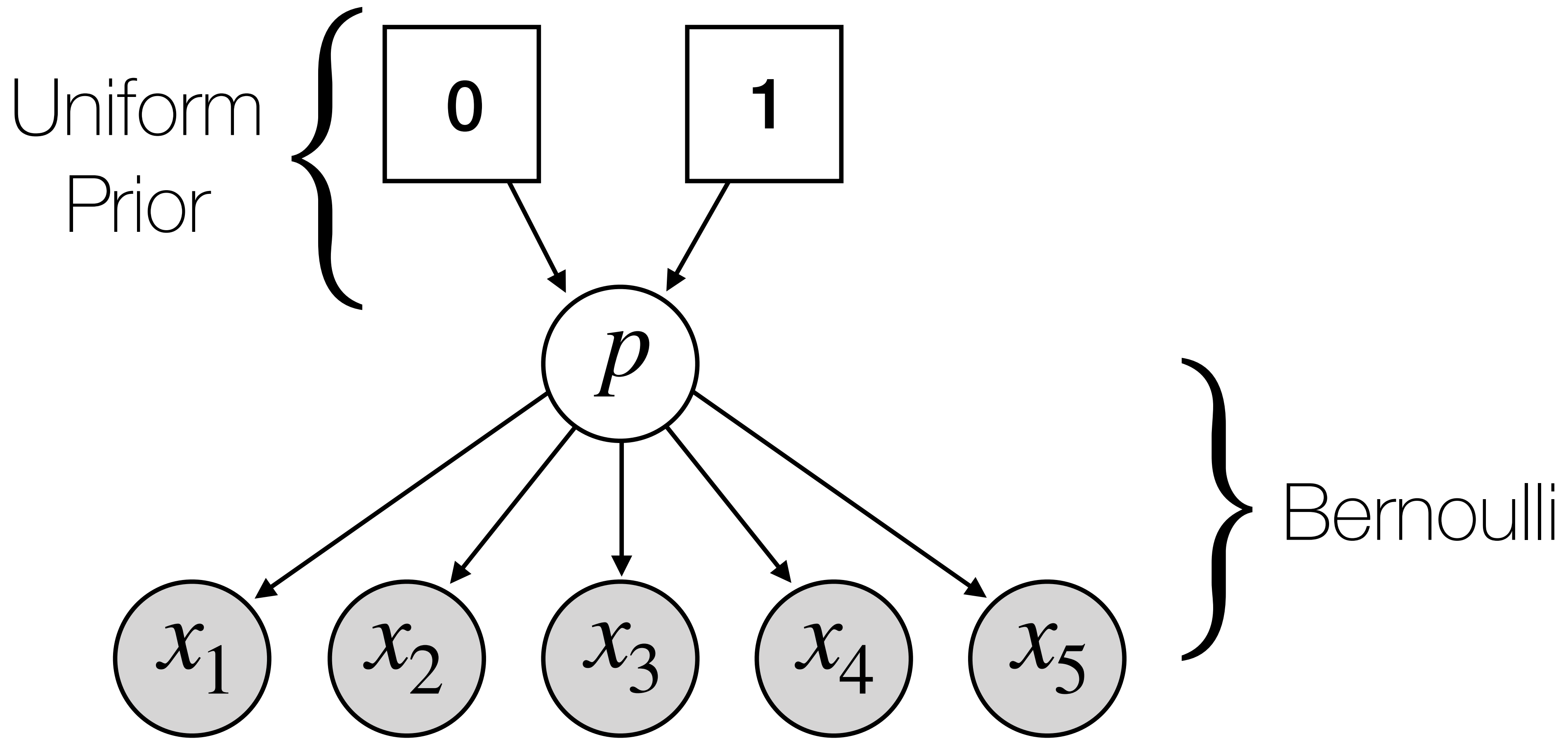
What's the structure of this model?
What can we infer about this model?
How much information will we have?

# Single Bernoulli Trial



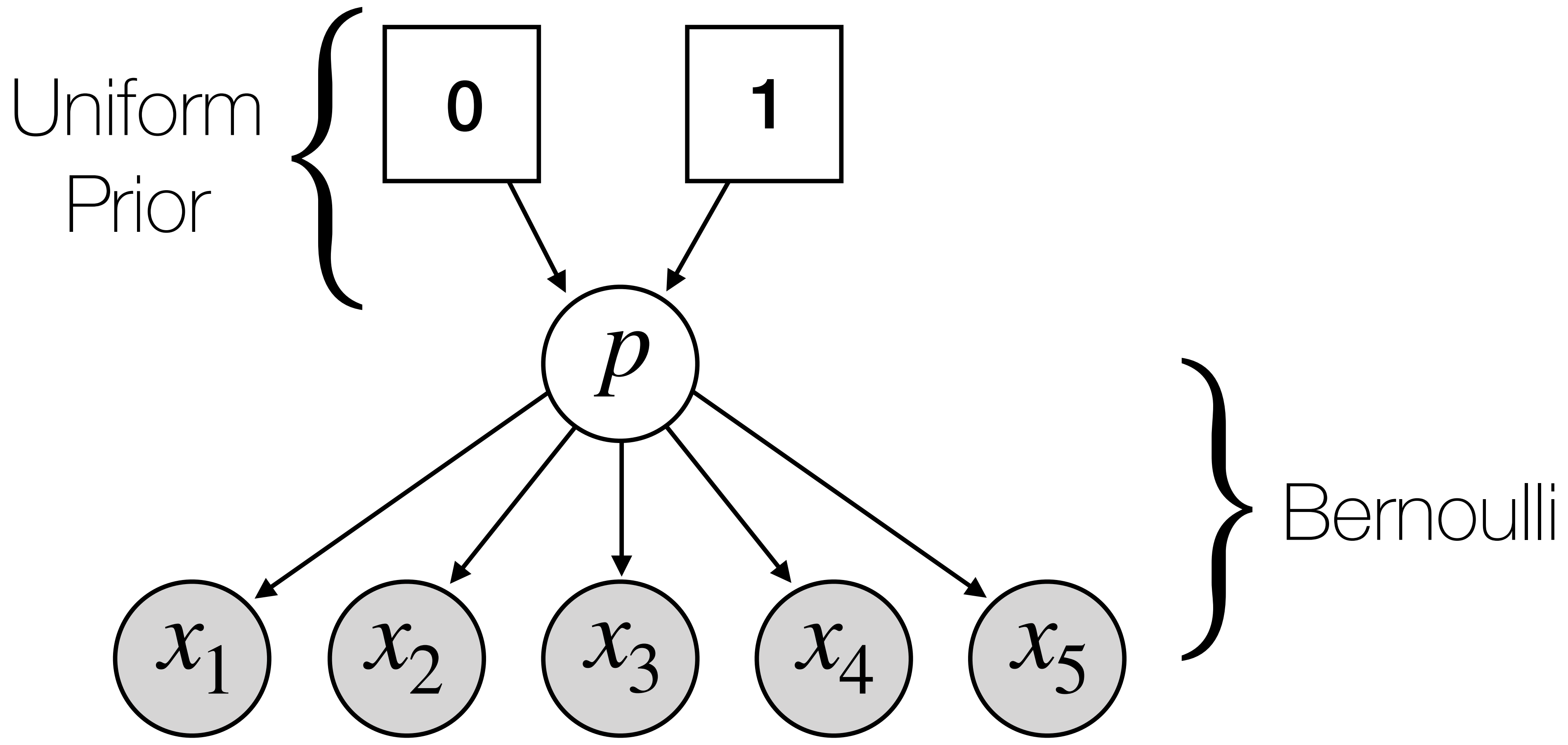
```
p ~ dnUnif(0,1)
x ~ dnBernoulli(p)
x.clamp(0)
```

## Series of Bernoullis with Linked *p*



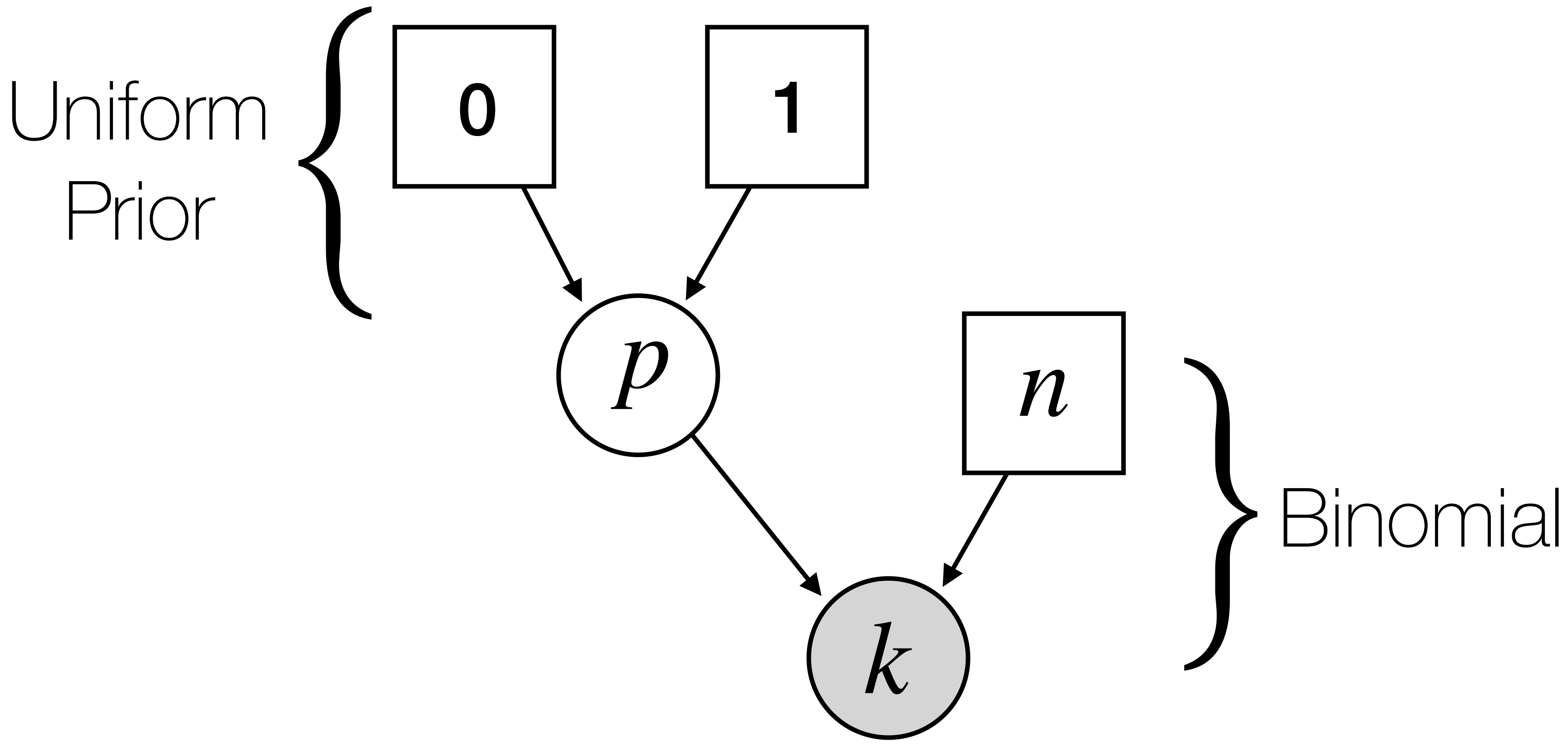
What can we infer about this model?
How much information will we have (relative to a single Bernoulli)?

# Series of Bernoullis with Linked *p*



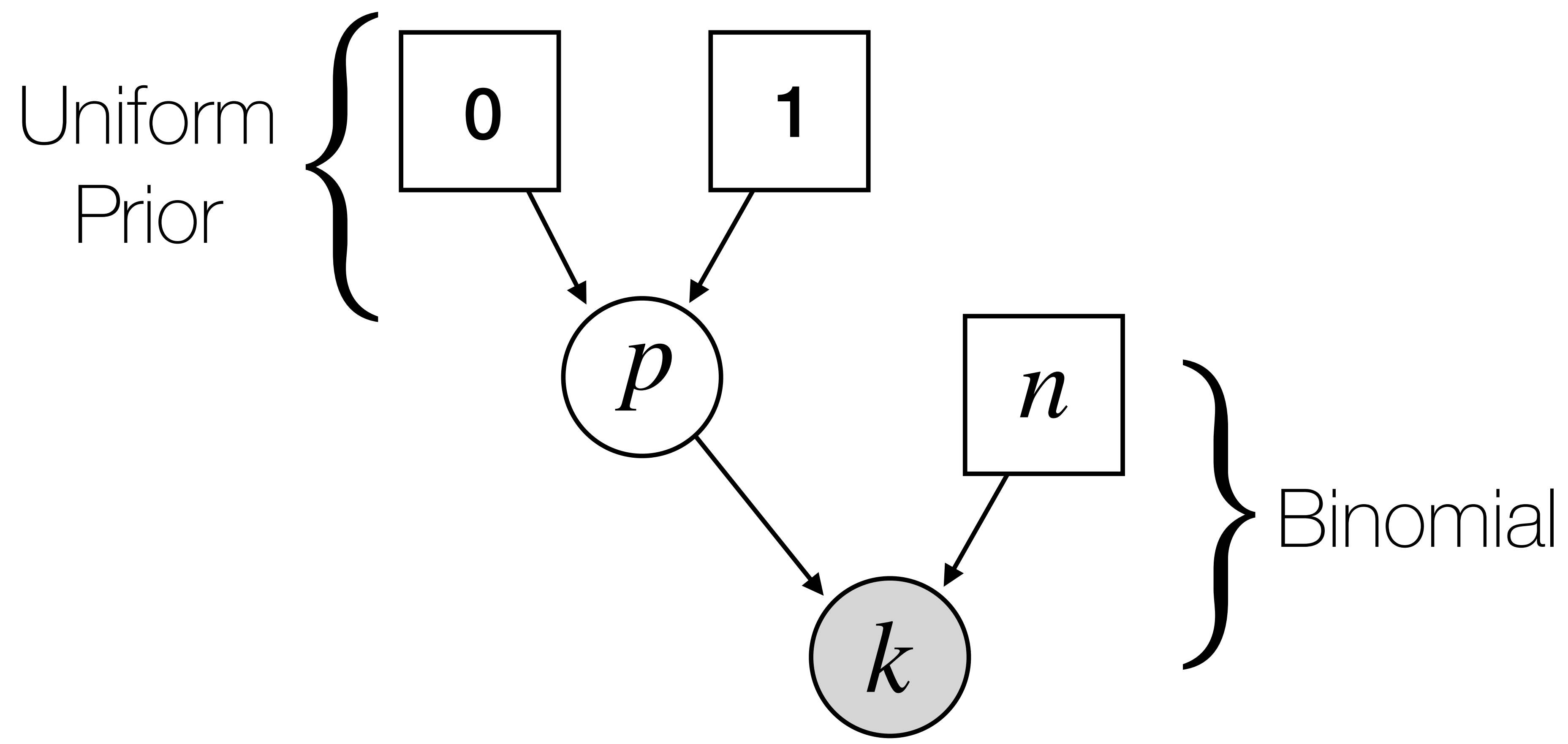
```
p ~ dnUnif(0,1)
for (i in 1:5){x[i] ~ dnBernoulli(p)}
x[1].clamp(0)
x[2].clamp(1)
x[3].clamp(1)
x[4].clamp(0)
x[5].clamp(1)
```

# Single Binomial



How does this model relate to the Bernoulli models?

# Single Binomial

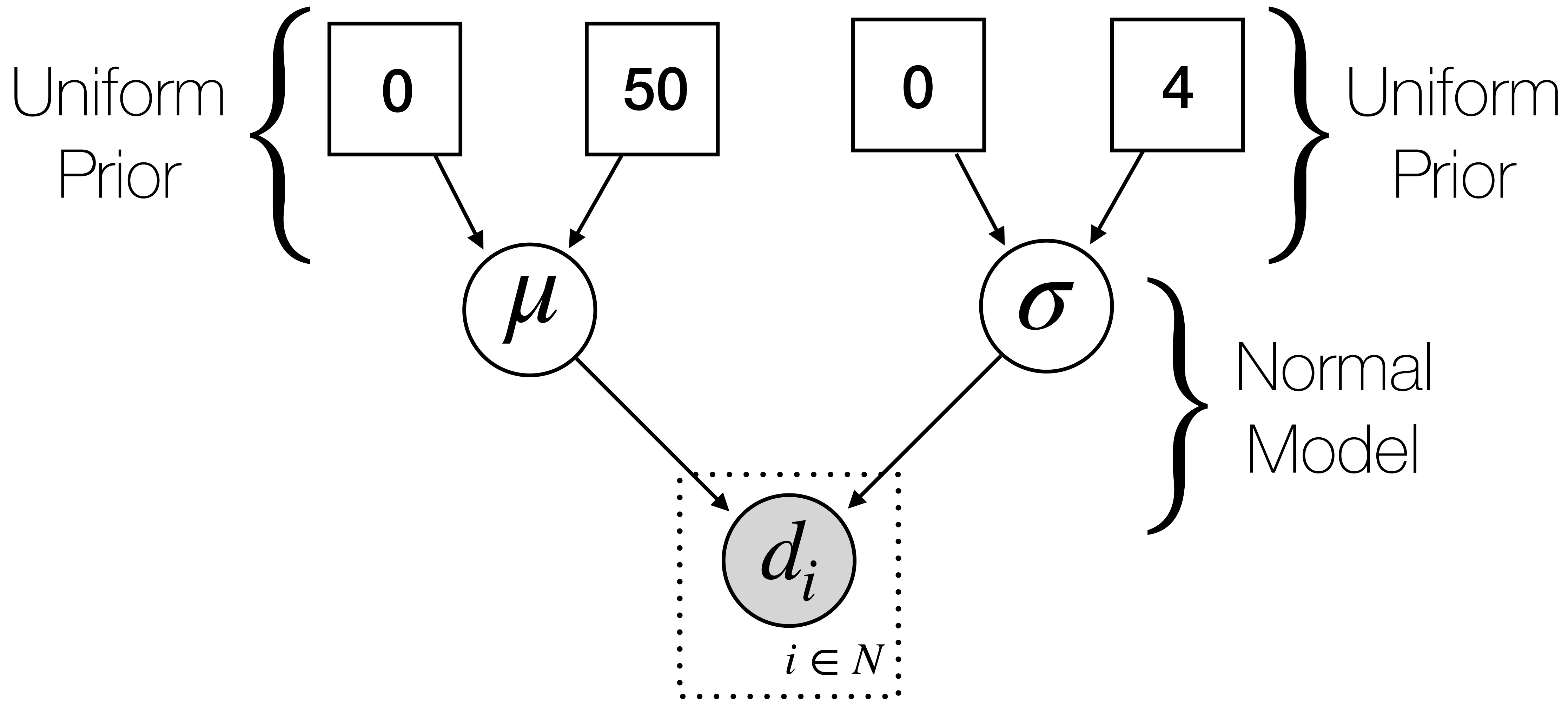


```
p ~ dnUnif(0,1)
n <- 5
k ~ dnBinomial(n,p)
k.clamp(3)
```

## Intro. to Graphical Models Jupyter Notebook (Part 1)

# Normal Model

(from last week)

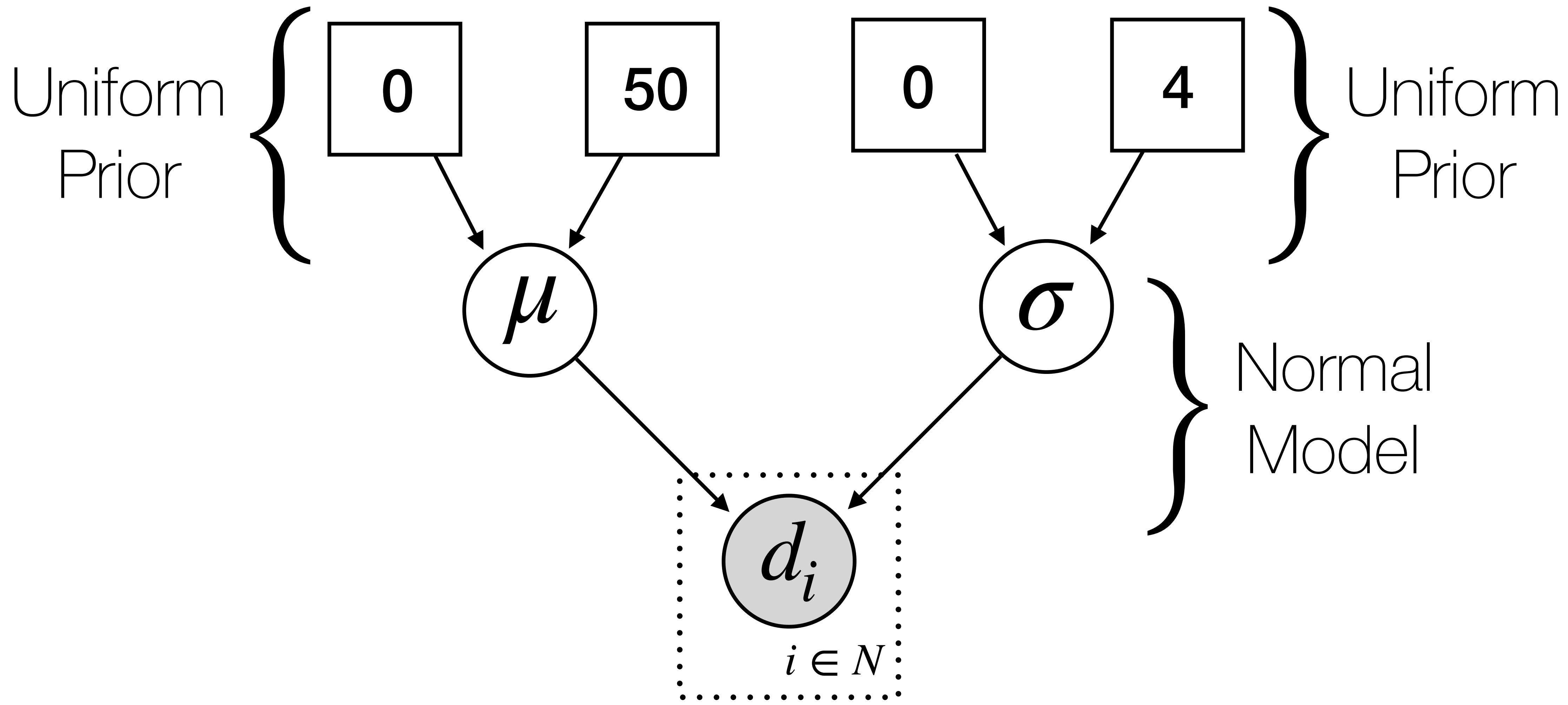


$d =$ observed data

$$d = \{10.1, 9.7, ..., 11.5\}$$

Where are the constant nodes? The deterministic nodes?
The stochastic nodes? The clamped stochastic nodes?

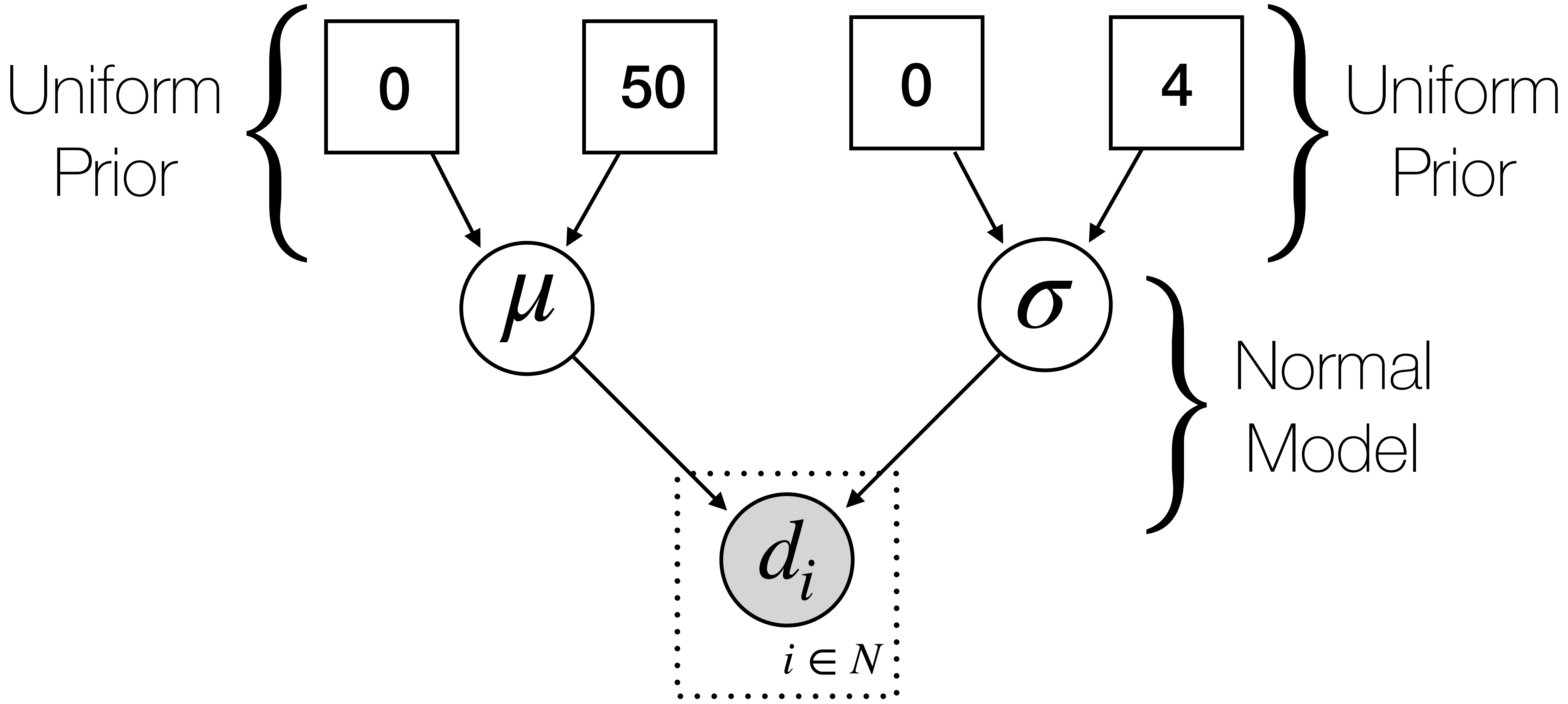
What parameters could we infer?



$d =$ observed data

$$d = \{10.1, 9.7, \ldots, 11.5\}$$

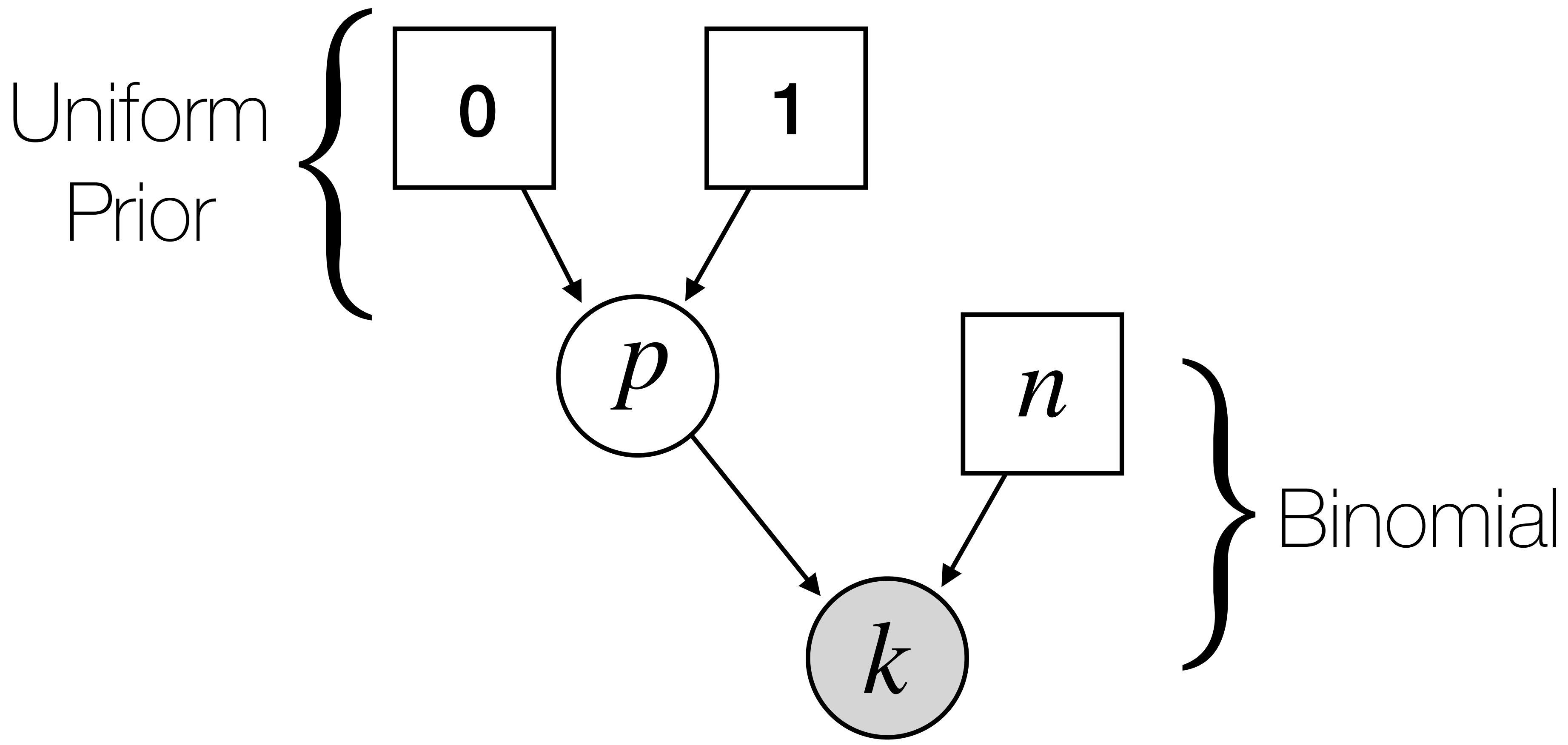
How would we write out this model in RevBayes?
Give it a try!
Start at the top and work your way down.
Make up your own data.



$d =$ observed data

$$d = \{10.1, 9.7, ..., 11.5\}$$

# Setting up MCMC in RevBayes

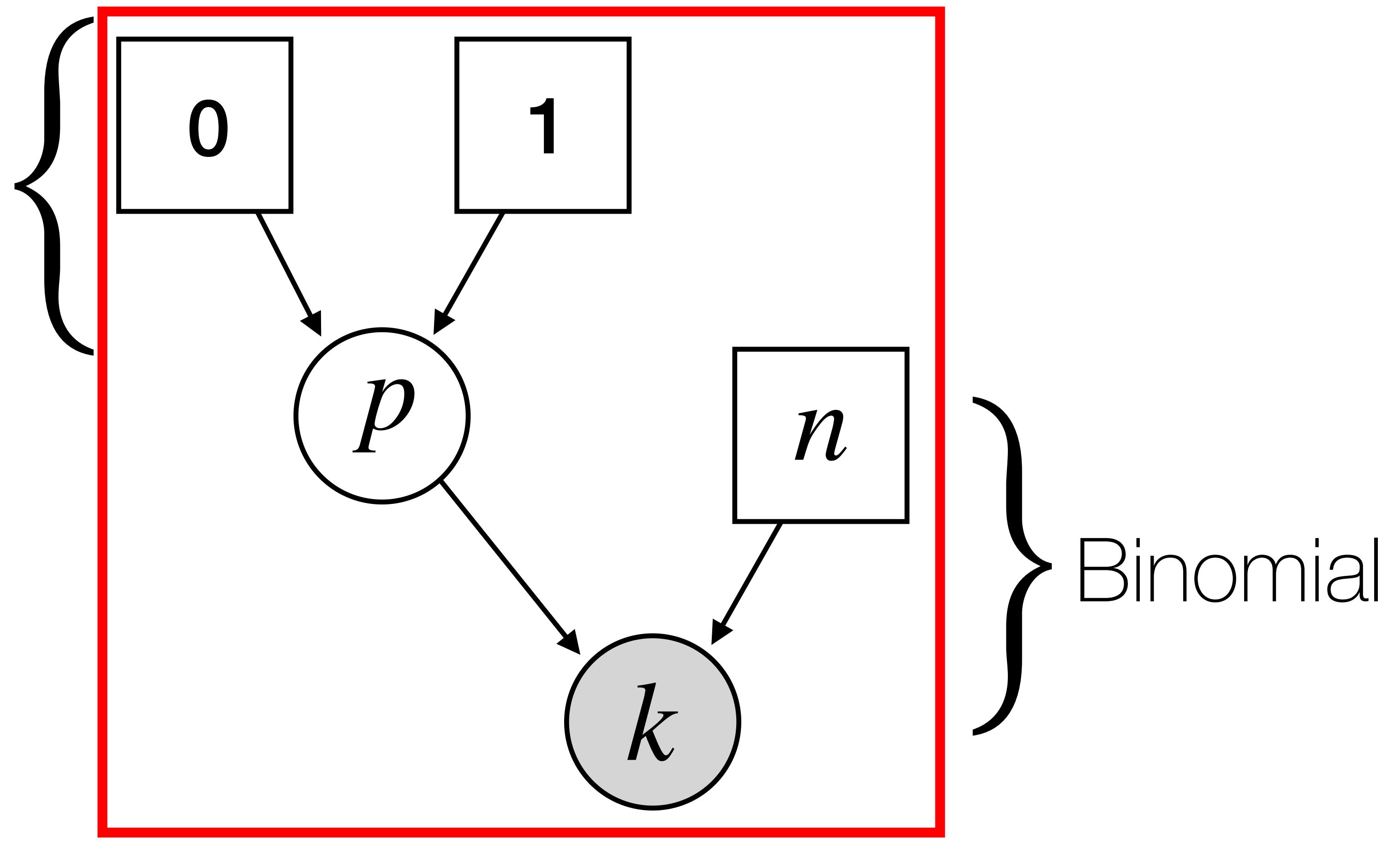


First, create a model object. You can pass any of the nodes to the constructor as a "handle".

```
myModel = model(n)
```

NOTE: We use the = assignment operator for "workspace" variables.

# Setting up MCMC in RevBayes

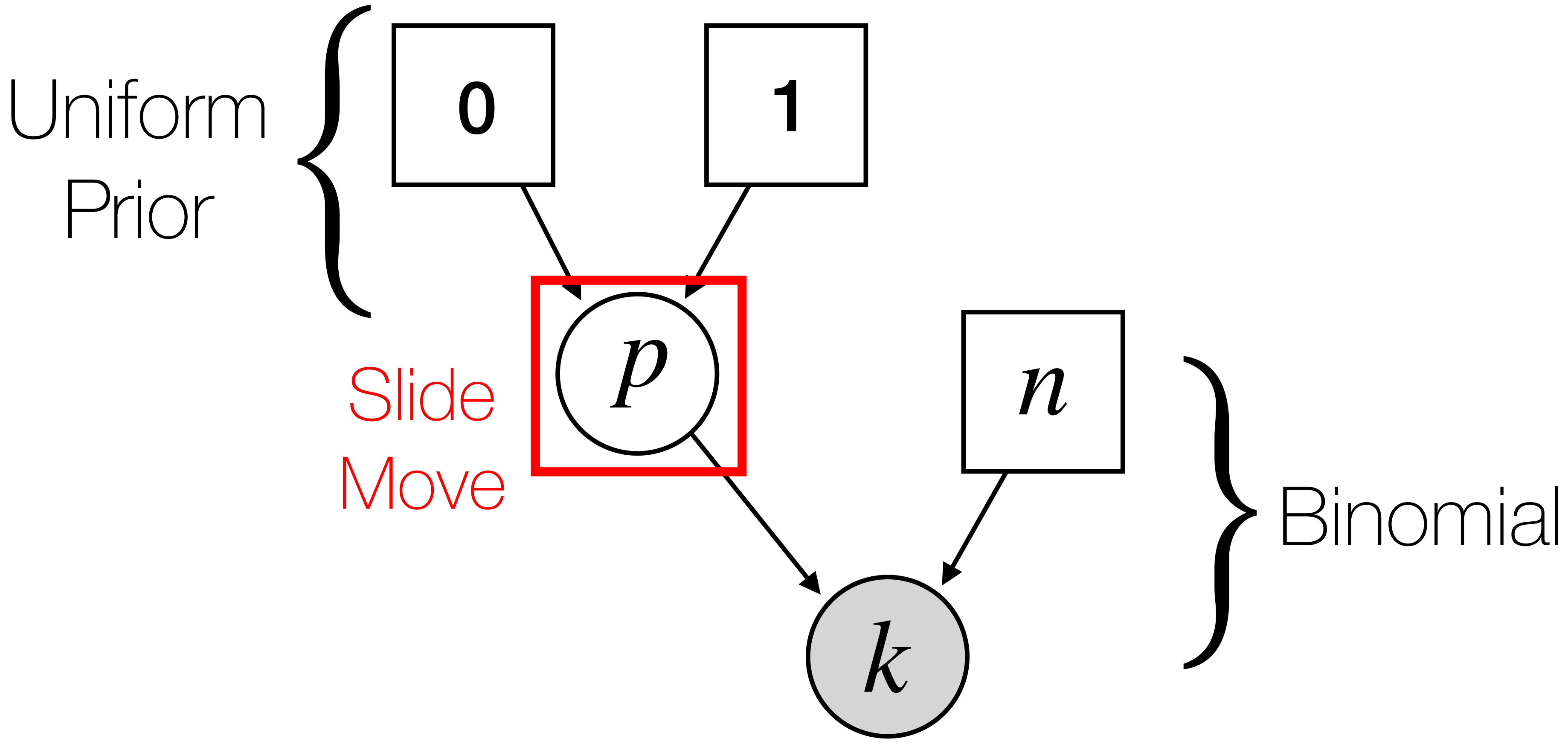


First, create a model object. You can pass any of the nodes to the constructor as a "handle".

```
myModel = model(n)
```

NOTE: We use the = assignment operator for "workspace" variables.

# Setting up MCMC in RevBayes

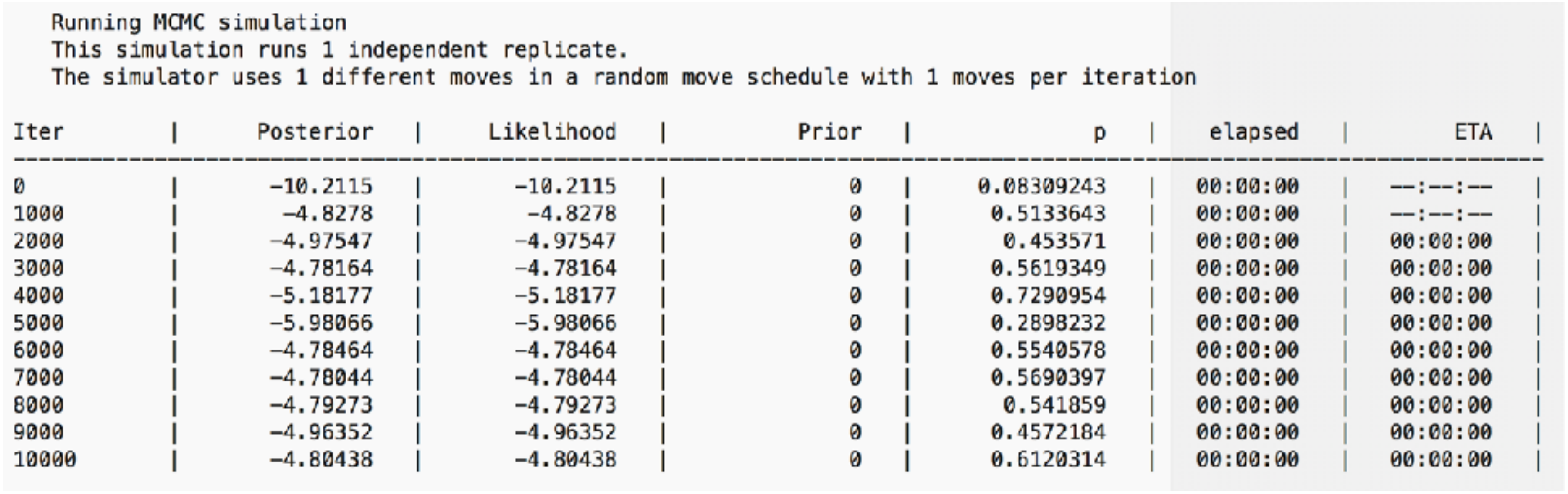


Next, we need to define a proposal distribution (move) for any parameters we are trying to infer.

```
moves[1] = mvSlide(p,delta=0.1,weight=1)
```

Many different move types are available in RevBayes.

# Setting up MCMC in RevBayes



```
   Running MCMC simulation
   This simulation runs 1 independent replicate.
   The simulator uses 1 different moves in a random move schedule with 1 moves per iteration

Iter          |      Posterior   |     Likelihood   |            Prior   |                p   |       elapsed   |         ETA   |
----------------------------------------------------------------------------------------------------------------------------------
0             |       -10.2115   |       -10.2115   |                0   |       0.08309243   |      00:00:00   |     --:--:--  |
1000          |        -4.8278   |        -4.8278   |                0   |        0.5133643   |      00:00:00   |     --:--:--  |
2000          |       -4.97547   |       -4.97547   |                0   |         0.453571   |      00:00:00   |     00:00:00  |
3000          |       -4.78164   |       -4.78164   |                0   |        0.5619349   |      00:00:00   |     00:00:00  |
4000          |       -5.18177   |       -5.18177   |                0   |        0.7290954   |      00:00:00   |     00:00:00  |
5000          |       -5.98066   |       -5.98066   |                0   |        0.2898232   |      00:00:00   |     00:00:00  |
6000          |       -4.78464   |       -4.78464   |                0   |        0.5540578   |      00:00:00   |     00:00:00  |
7000          |       -4.78044   |       -4.78044   |                0   |        0.5690397   |      00:00:00   |     00:00:00  |
8000          |       -4.79273   |       -4.79273   |                0   |         0.541859   |      00:00:00   |     00:00:00  |
9000          |       -4.96352   |       -4.96352   |                0   |        0.4572184   |      00:00:00   |     00:00:00  |
10000         |       -4.80438   |       -4.80438   |                0   |        0.6120314   |      00:00:00   |     00:00:00  |
```

As our MCMC runs, we need to keep track of our progress and sampled parameter values. To do that we use monitors.

```
monitors[1] = mnScreen(printgen=1000,p)
```

Next we create an MCMC object.

```
myMCMC = mcmc(myModel,moves,monitors)
```

Now, we start the MCMC!

```
myMCMC.run(10000)
```

## Intro. to Graphical Models Jupyter Notebook (Part 2)

From here on out, we'll be moving between running RevBayes in Jupyter and running it on the command line.

To make sure you have the command-line version in your PATH, open up Terminal and type

*rb*

Practice by setting up MCMC for the Normal model that you created.

Put these commands in text file with a “.rev” extension.

Run your MCMC.

Examine your MCMC output in Tracer.