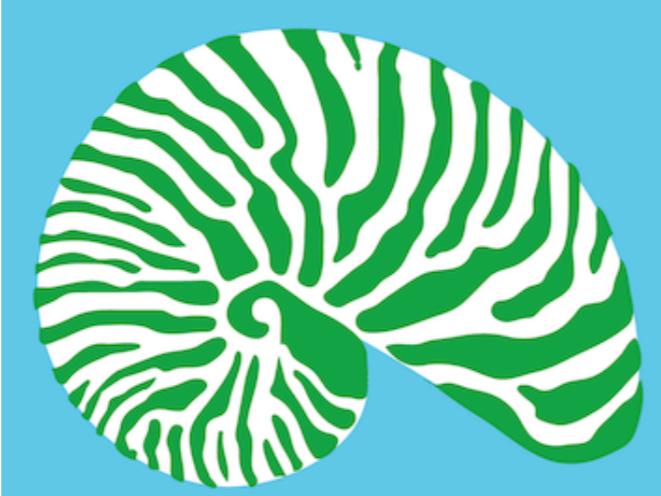


# **Please ask!**

<https://onlinequestions.org/>

18192



The Society of  
Systematic  
Biologists

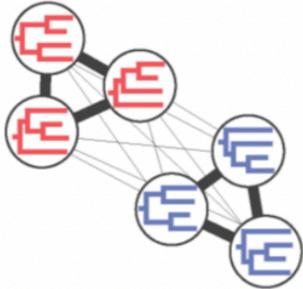
Breakout Meeting  
Jan. 9-11, 2026  
Baton Rouge, LA



The Importance of  
Natural History  
Collections

Featuring workshops, panels, lightning talks,  
museum tours, socials, and more.

<https://ssb2026.github.io/>



# What is CloudForest?

CloudForest is a portable, easy-to-deploy, and reproducible computational platform for phylogenomics. The goal of CloudForest is to serve as a bridge that allows researchers to span all steps of a phylogenomic analysis, in order to explore and better understand variation and structure in sets of phylogenetic trees. While the tools available in CloudForest have many applications, some use cases include:

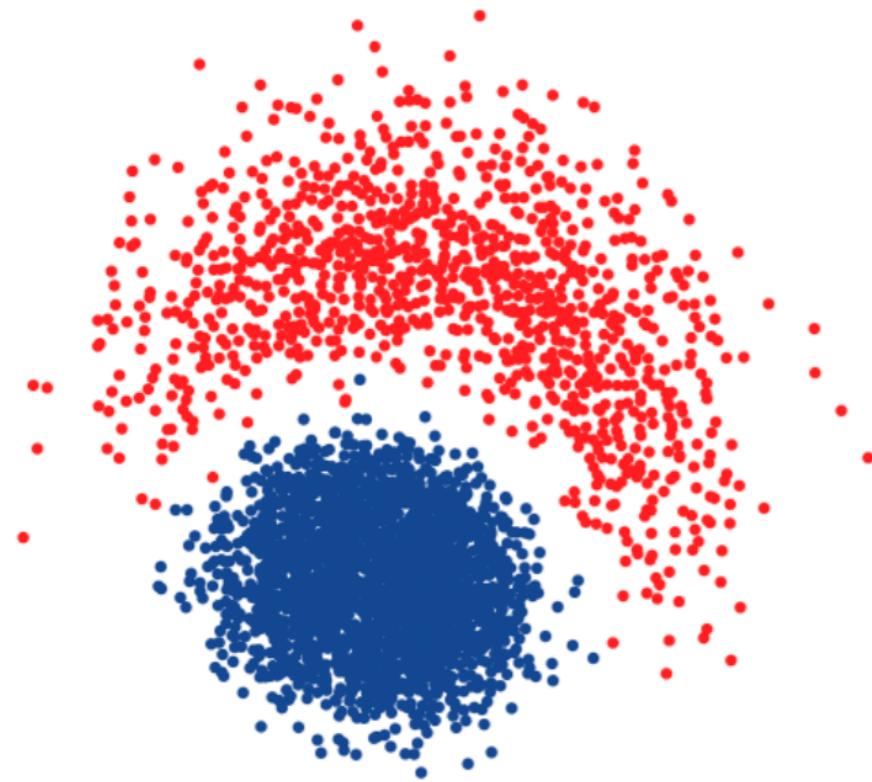
## Quick Links:

[Getting Started](#)

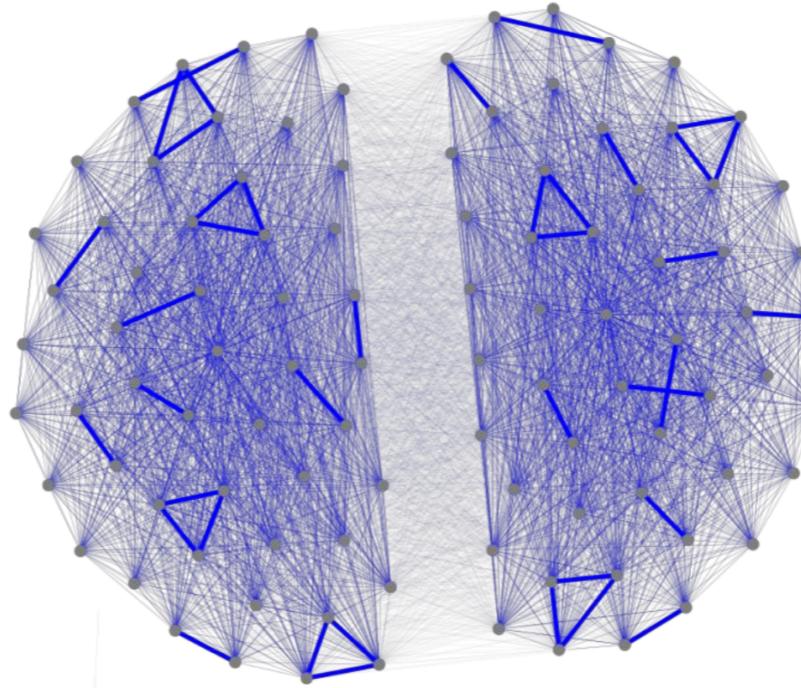
[Docker](#)

[CIPRES REST API](#)

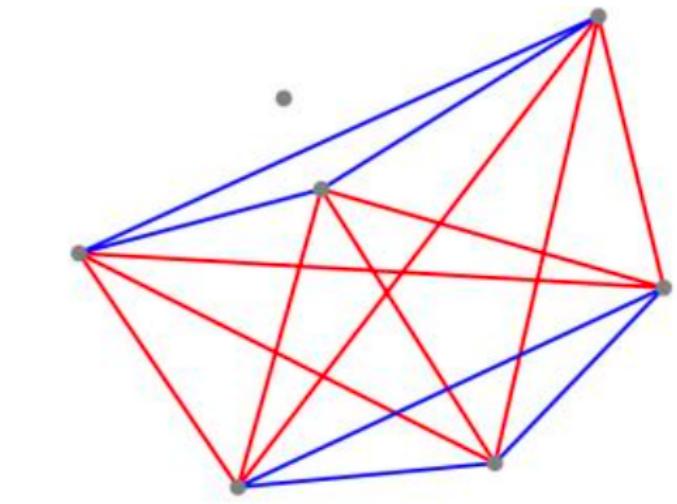
- Comparing the results of different phylogenomic analyses
- Exploring gene tree variation
- Identifying outlier genes
- Identifying species of hybrid origin
- Community phylogenetics and biogeography
- Sensitivity analyses (particularly for Bayesian inference)



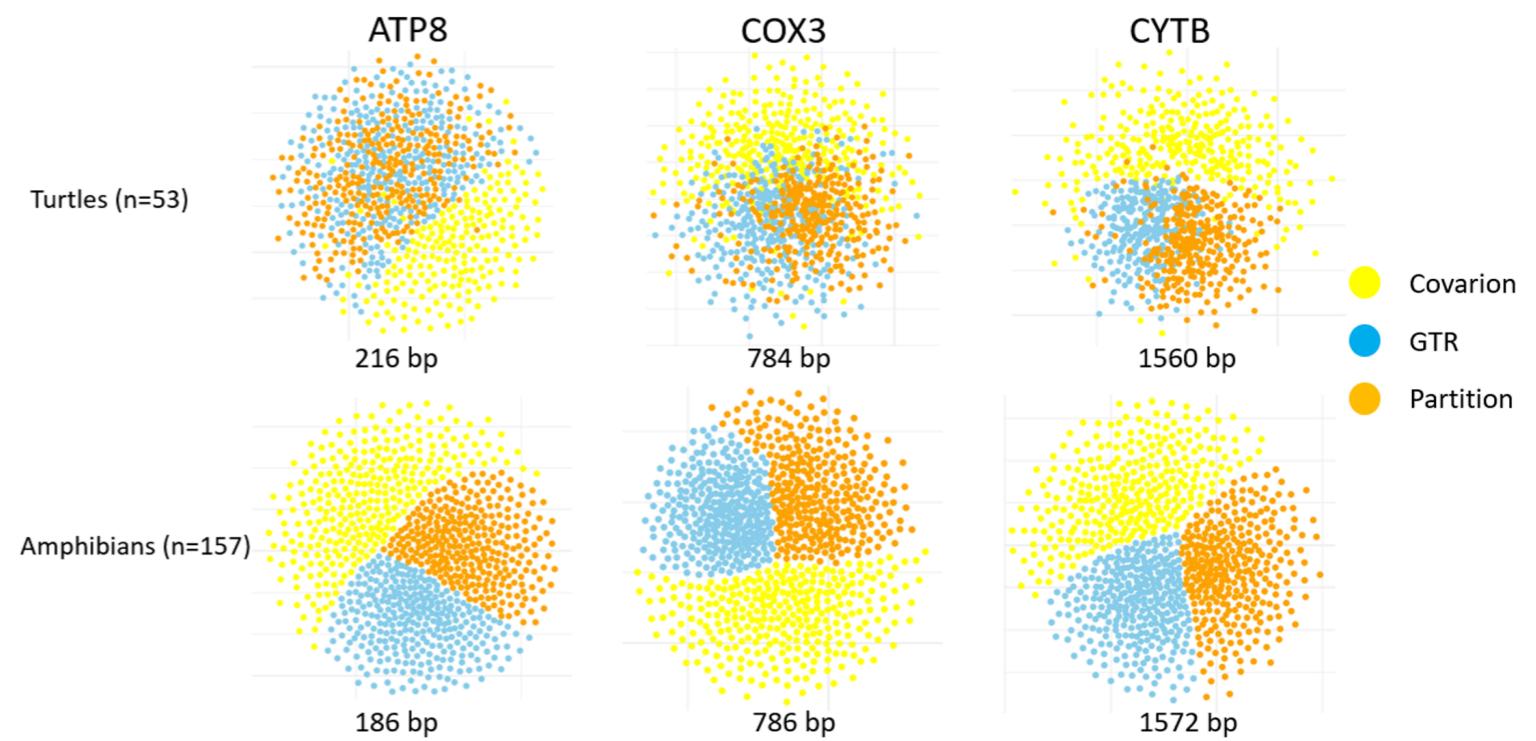
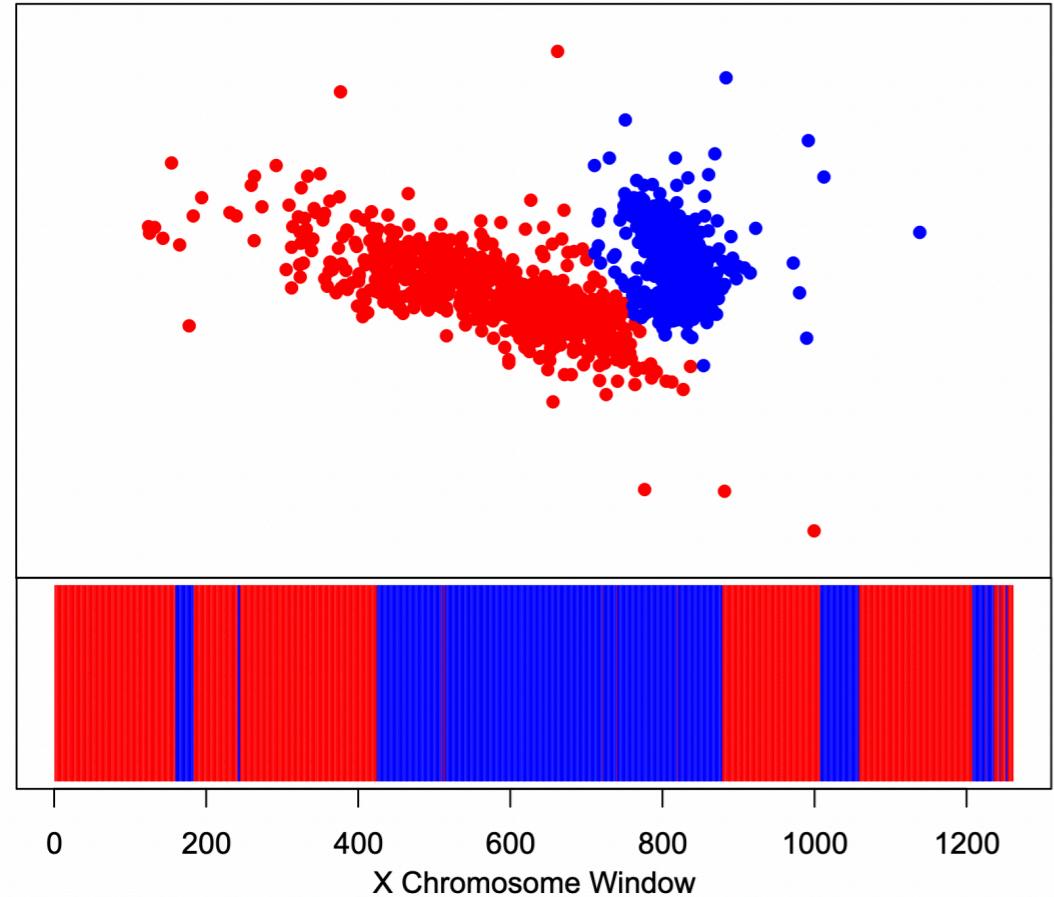
Visualizations of  
Tree Space



Networks of  
Trees



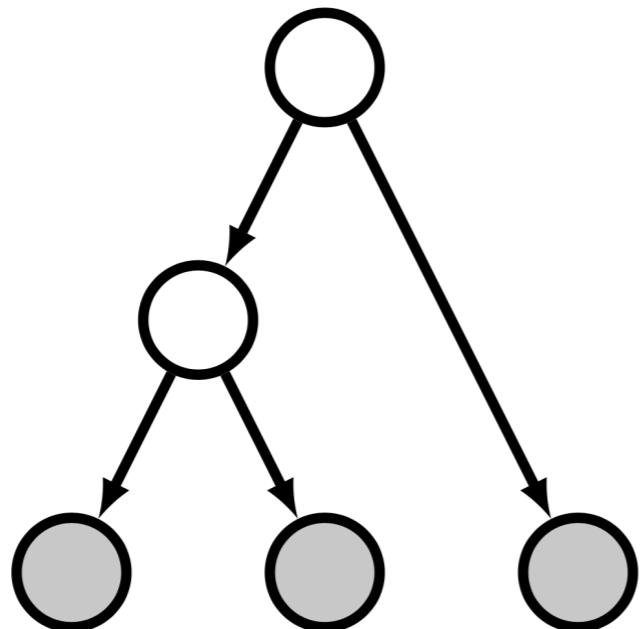
Networks of  
Bipartitions



# Structured Variation In Gene Trees

# Sensitivity Analyses

# Intro. to Graphical Models and RevBayes

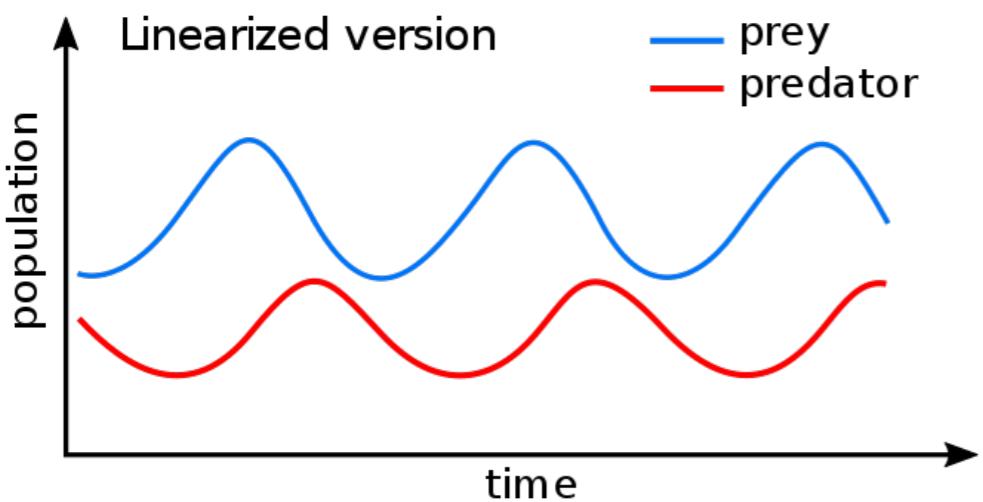


Jeremy M. Brown  
Louisiana State University

# What is a model?

Math

$$\frac{dx}{dt} = \alpha x - \beta xy,$$
$$\frac{dy}{dt} = \delta xy - \gamma y,$$



Biology

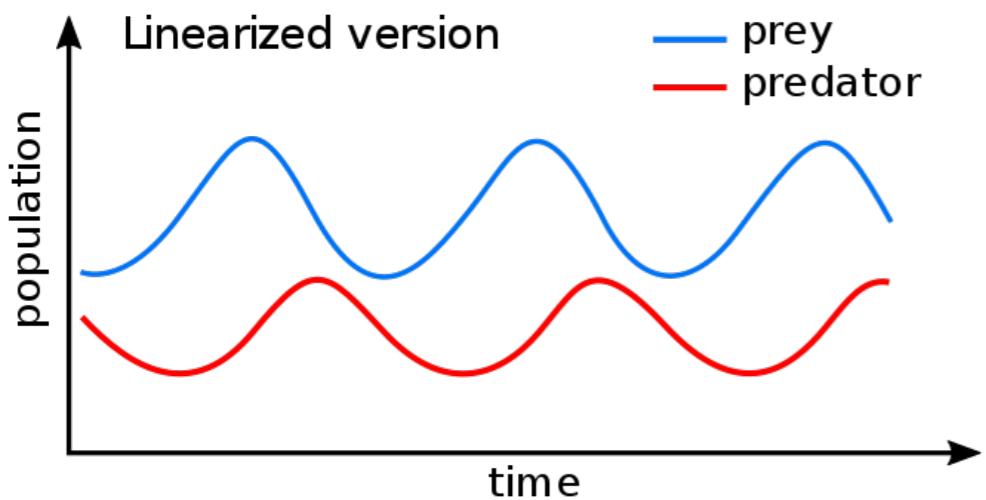
(something observed)



# What is a model?

Mathematical models try to explain important features of (biological) systems using (relatively simple) mathematical principles.

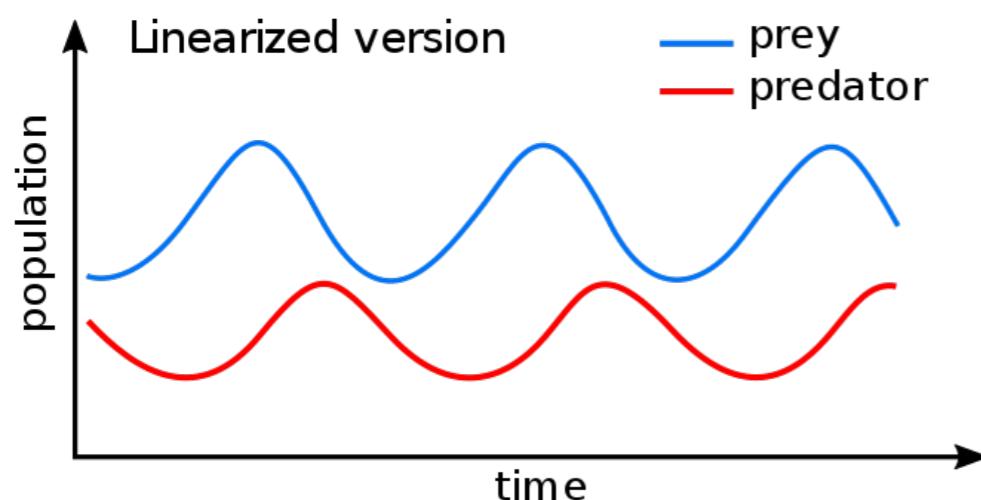
$$\frac{dx}{dt} = \alpha x - \beta xy,$$
$$\frac{dy}{dt} = \delta xy - \gamma y,$$



# Deterministic and Stochastic Models

$$\frac{dx}{dt} = \alpha x - \beta xy,$$

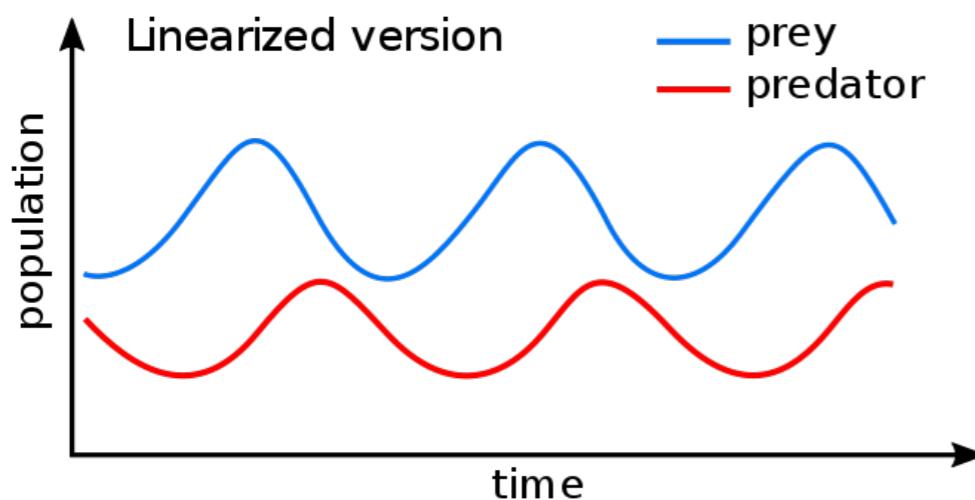
$$\frac{dy}{dt} = \delta xy - \gamma y,$$



# Deterministic and Stochastic Models

$$\frac{dx}{dt} = \alpha x - \beta xy,$$

$$\frac{dy}{dt} = \delta xy - \gamma y,$$



**NO randomness**

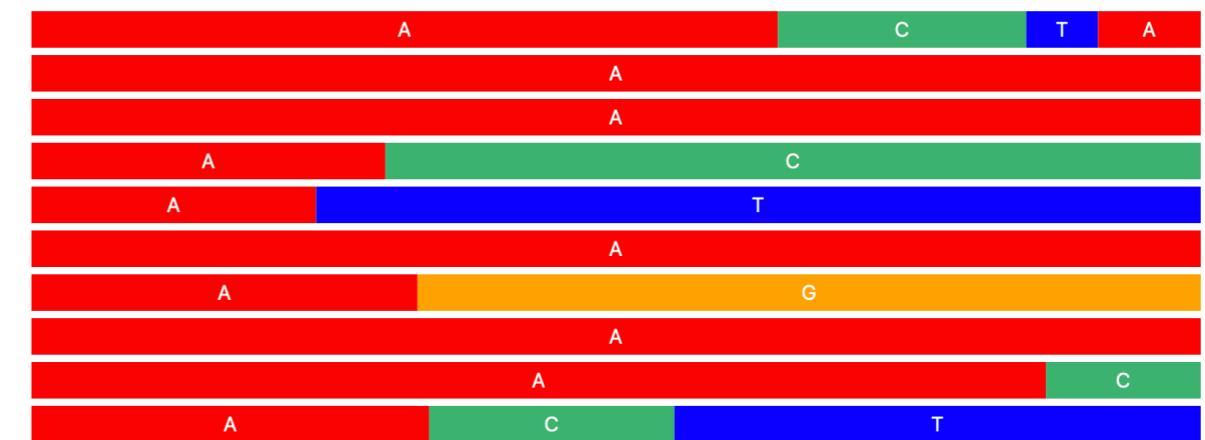
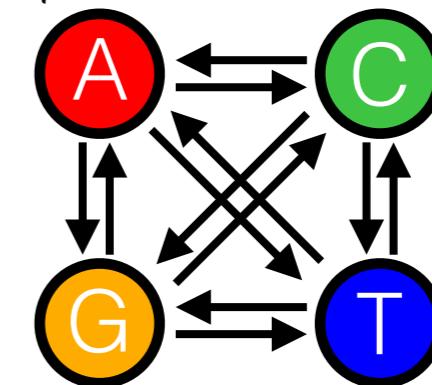
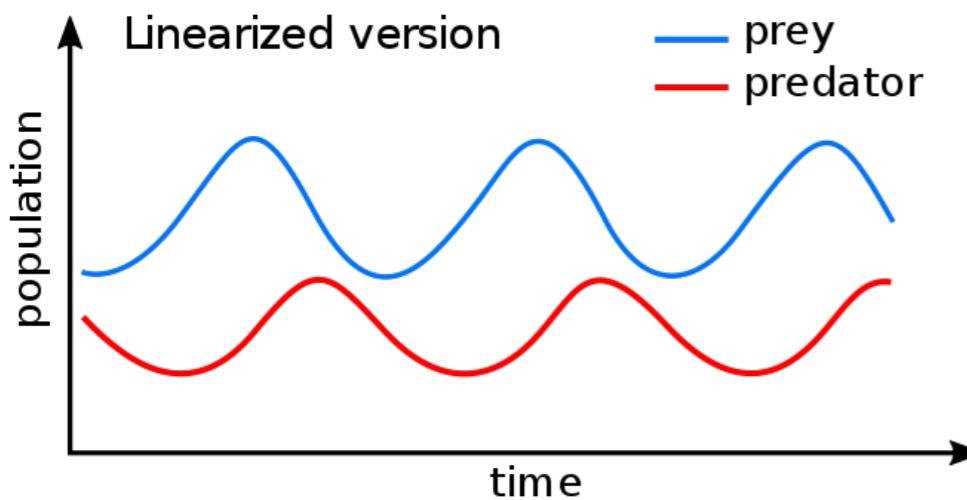
(no stochasticity)

(no probability statements)

# Deterministic and Stochastic Models



$$\frac{dx}{dt} = \alpha x - \beta xy,$$
$$\frac{dy}{dt} = \delta xy - \gamma y,$$



**NO randomness**

(no stochasticity)

(no probability statements)

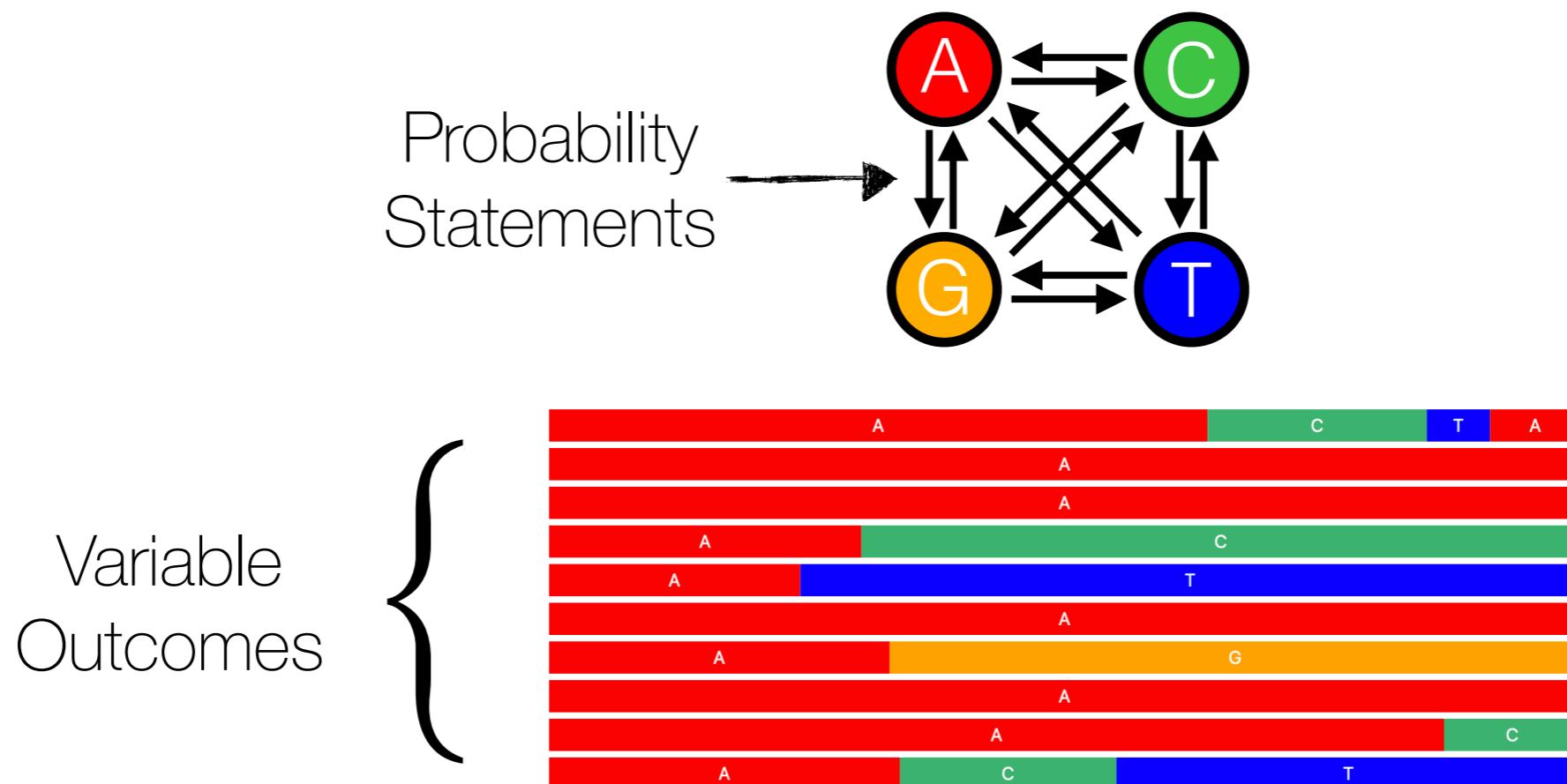
**Randomness!**

(stochasticity)

(probability statements)

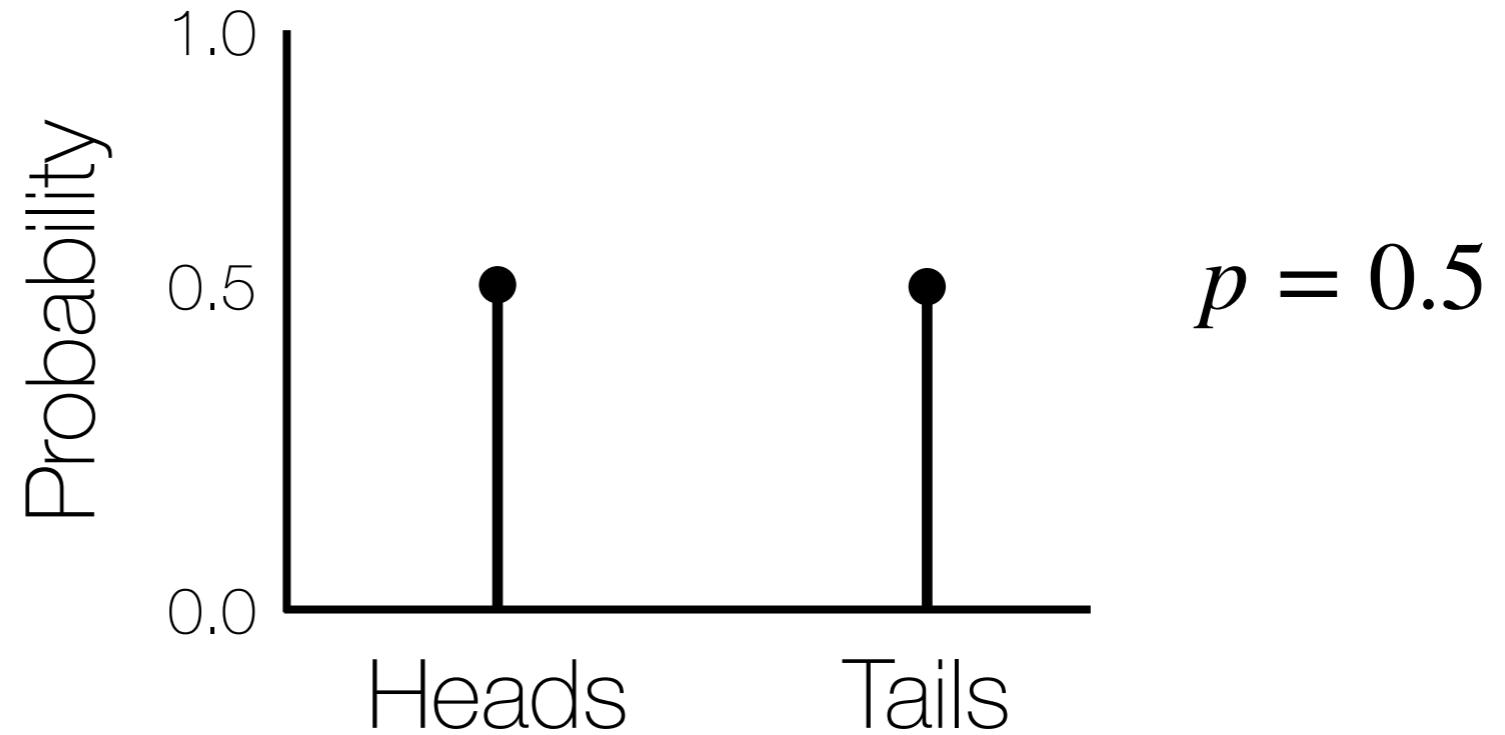
(variation in possible outcomes)

# Stochastic Models



**Randomness!**  
(stochasticity)  
(probability statements)  
(variation in possible outcomes)

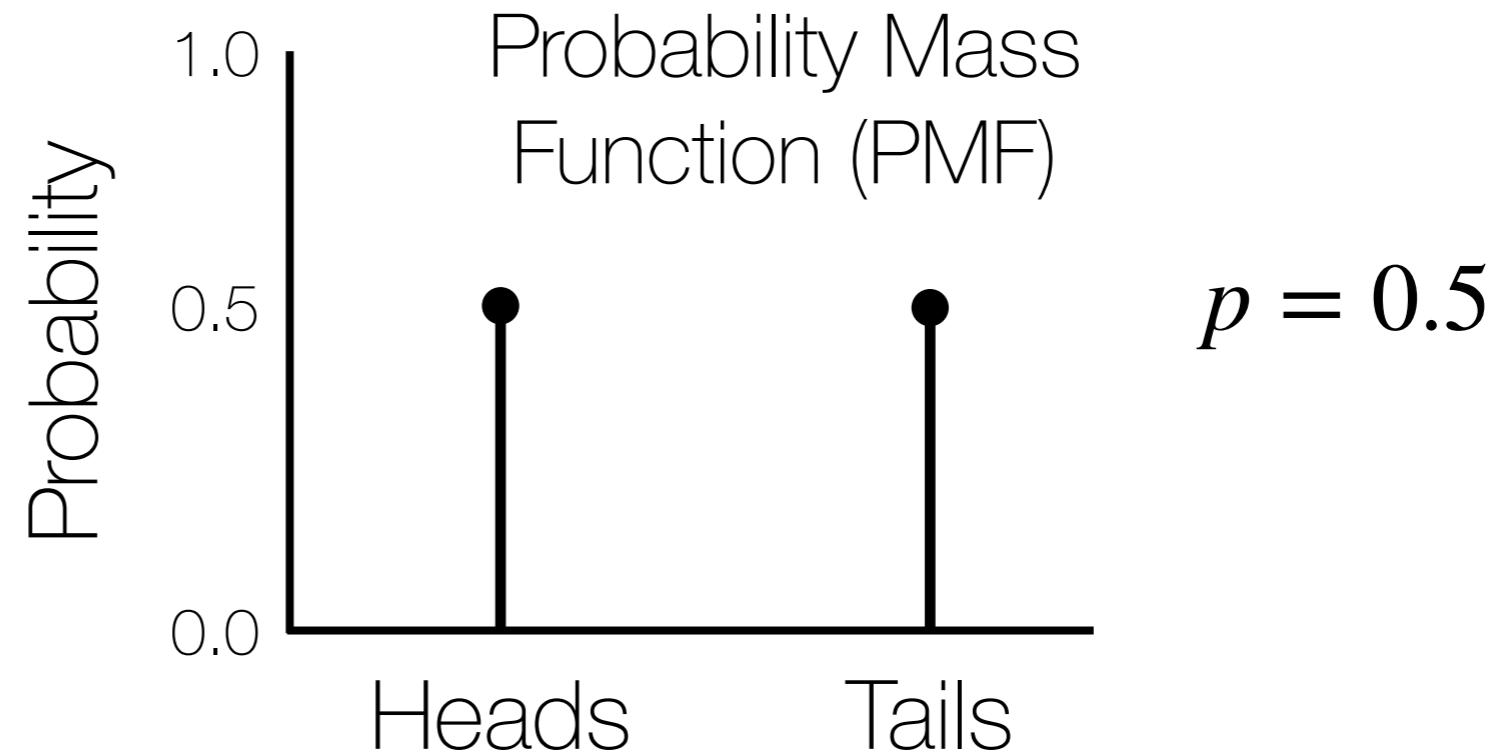
# A simple stochastic model



## Bernoulli

Describes the outcome of a single coin flip  
(or other binary 0/1 trial)

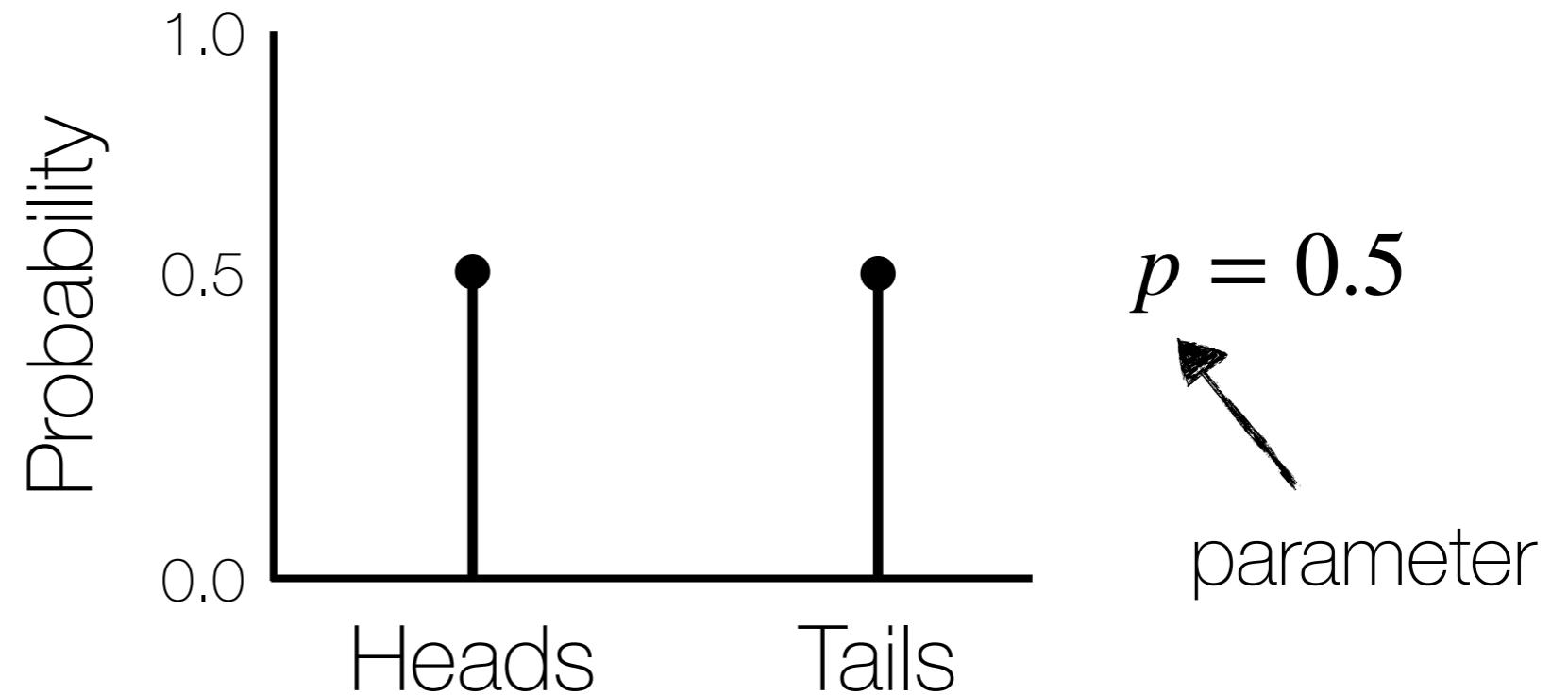
# A simple stochastic model



## Bernoulli

One type of *discrete* distribution

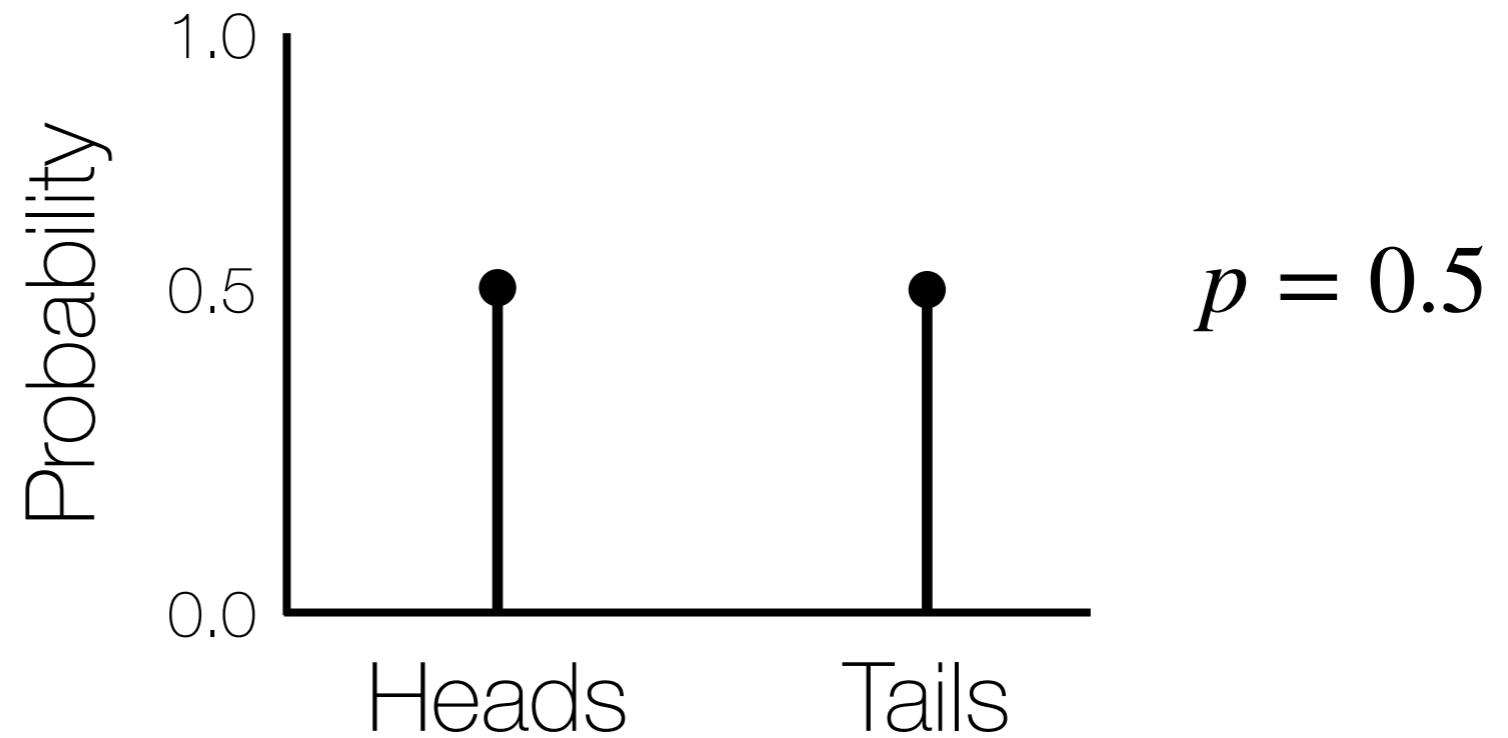
# A simple stochastic model



$$P(\text{Heads}) = p$$

$$P(\text{Tails}) = 1-p$$

# Let's build this model!



# RevBayes

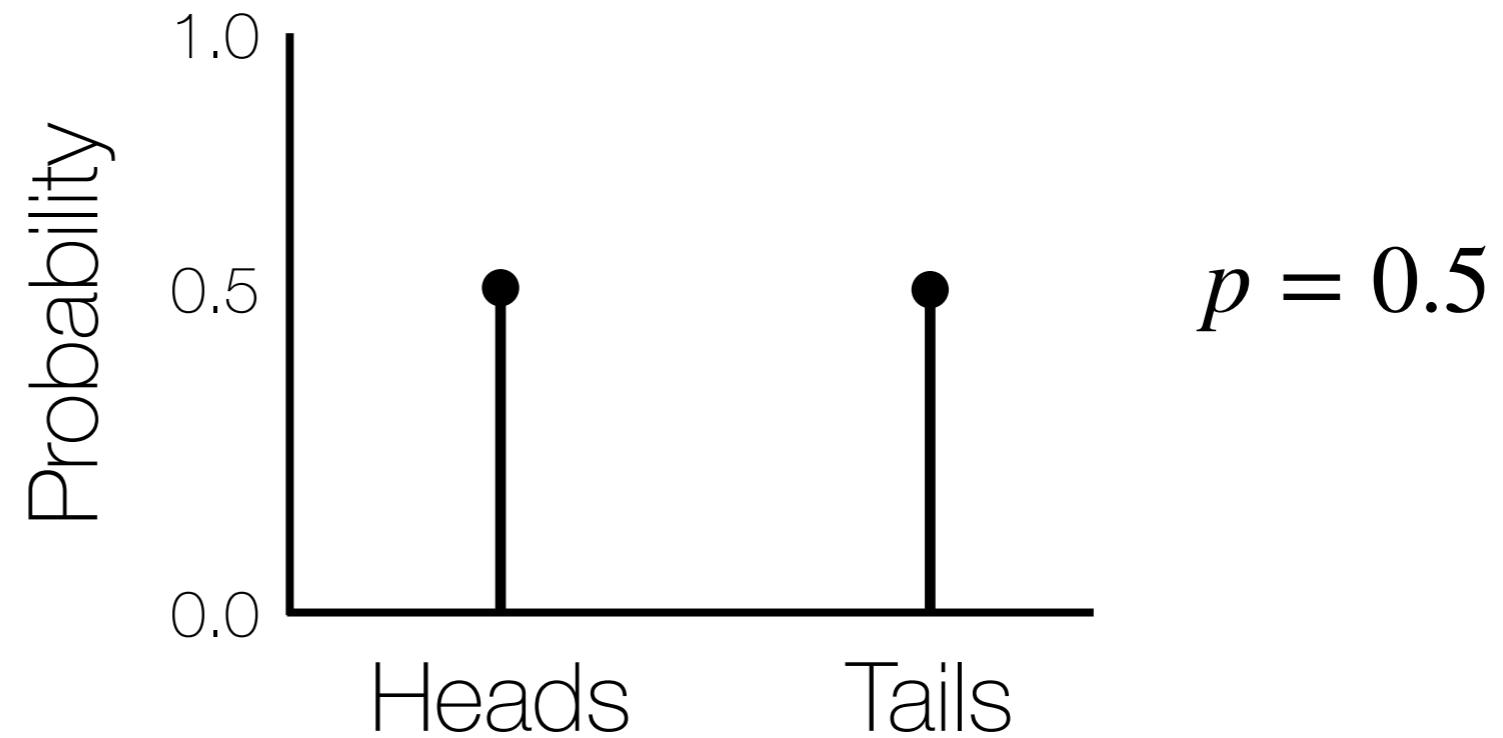


```
[moleuser@amphioxis-49-of-66:~$ rb
RevBayes version (1.1.1)
Build from tags/1.1.1 (rapture-588-gae00cc) on Sat Mar 19 00:48:15 UTC 2022
Visit the website www.RevBayes.com for more information about RevBayes.
RevBayes is free software released under the GPL license, version 3. Type 'license()' for details.
To quit RevBayes type 'quit()' or 'q()'.

> _
```

<https://revbayes.github.io>

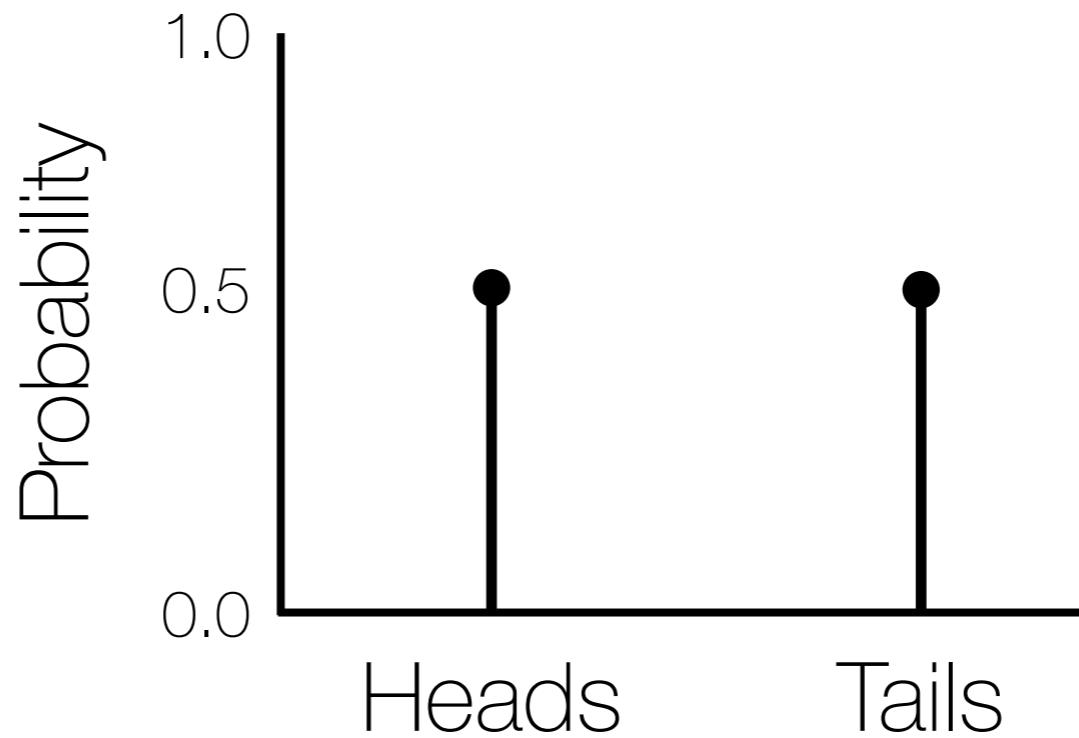
# Let's build this stochastic model!



$$z \sim \text{dnBernoulli}(0.5)$$

$z$

# Let's build this stochastic model!



Random  
Variable

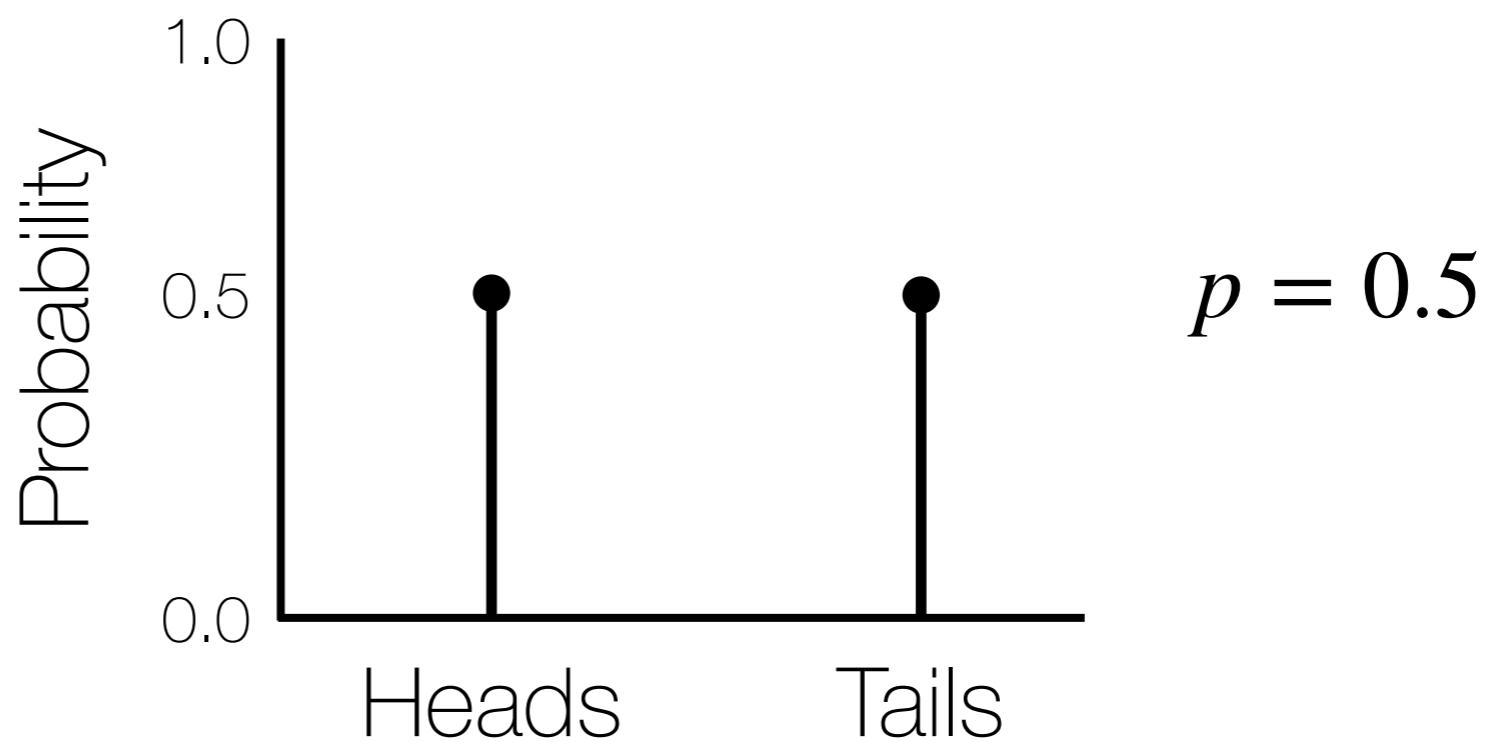


$$z \sim \text{dnBernoulli}(0.5)$$

$z$

“distributed as”

Rev language  
intended to feel a  
lot like R, but you  
can't create random  
variables in R.



```
> z ~ dnBernoulli(0.5)
[> z
  0
> z ~ dnBernoulli(0.5)
[> z
  0
> z ~ dnBernoulli(0.5)
[> z
  1
> z ~ dnBernoulli(0.5)
[> z
  1
> z ~ dnBernoulli(0.5)
[> z
  1
> z ~ dnBernoulli(0.5)
[> z
  0
```

When you create a random variable in RevBayes, it automatically initializes that variable by drawing a value from the corresponding probability distribution.

# The Role of Models

- (1) Can we explain the important features of the real world using relatively simple mathematical principles (**prediction**)?
- (2) Within this mathematical framework, can we learn about the processes that generated our data (**inference**)?

**Useful stochastic models can answer ‘yes’ to both.**

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

z

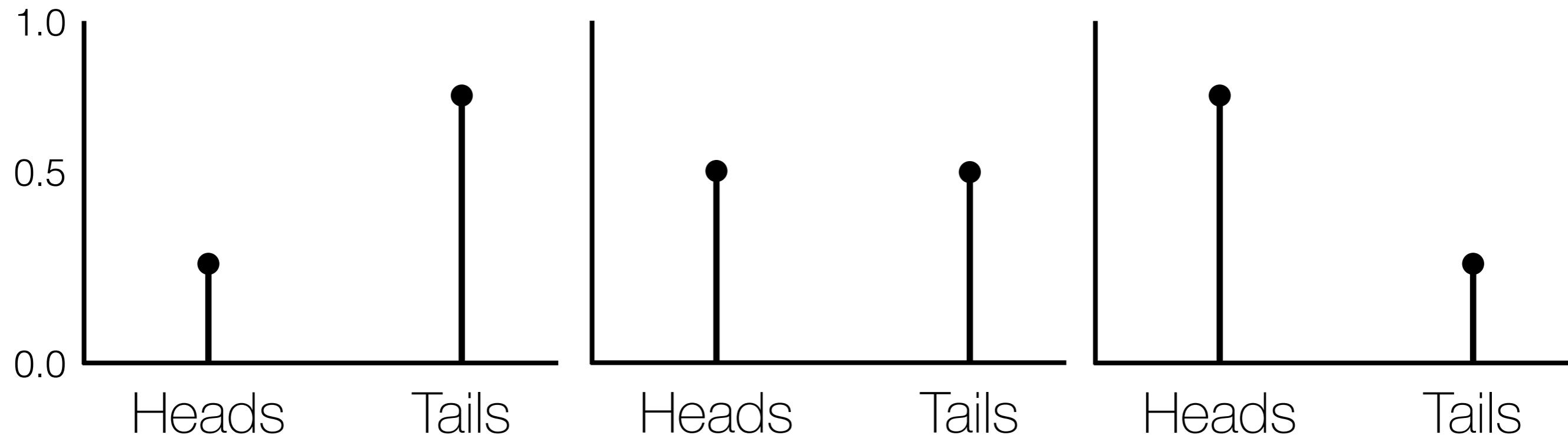
(1)   $p=0.5$   
(2)  No data

Different parameter values imply different probabilities for outcomes  
(different PMFs)

$p = 0.25$

$p = 0.5$

$p = 0.75$

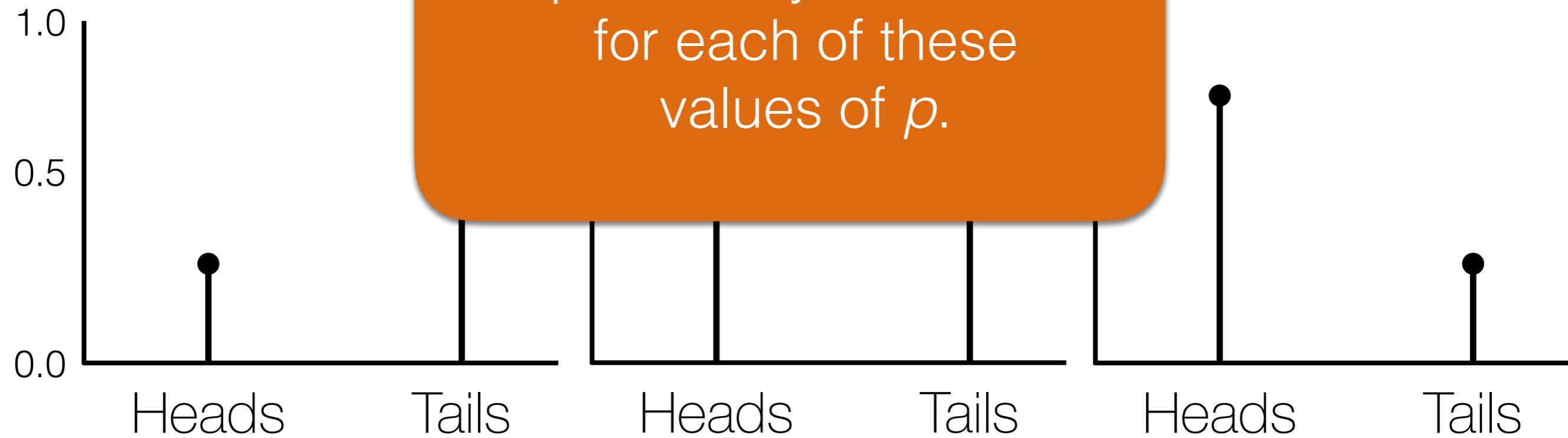


# Different parameter values imply different probabilities for outcomes (different PMFs)

$p = 0.25$

Let's "compute" the probability of Heads for each of these values of  $p$ .

$p = 0.75$



# Computing Likelihoods

```
p <- 0.5  
z ~ dnBernoulli(p)  
z.clamp(1)  
z.probability()
```

Here is where we attach our data (i.e., Heads) to our model. This is known as “clamping”.

Here we compute:

$$P(\text{Heads} \mid p = 0.5)$$

$$L(p = 0.5; \text{Heads})$$

```
> p <- 0.25  
> z ~ dnBernoulli(p)  
> z.clamp(1)  
> z.probability()  
0.25
```

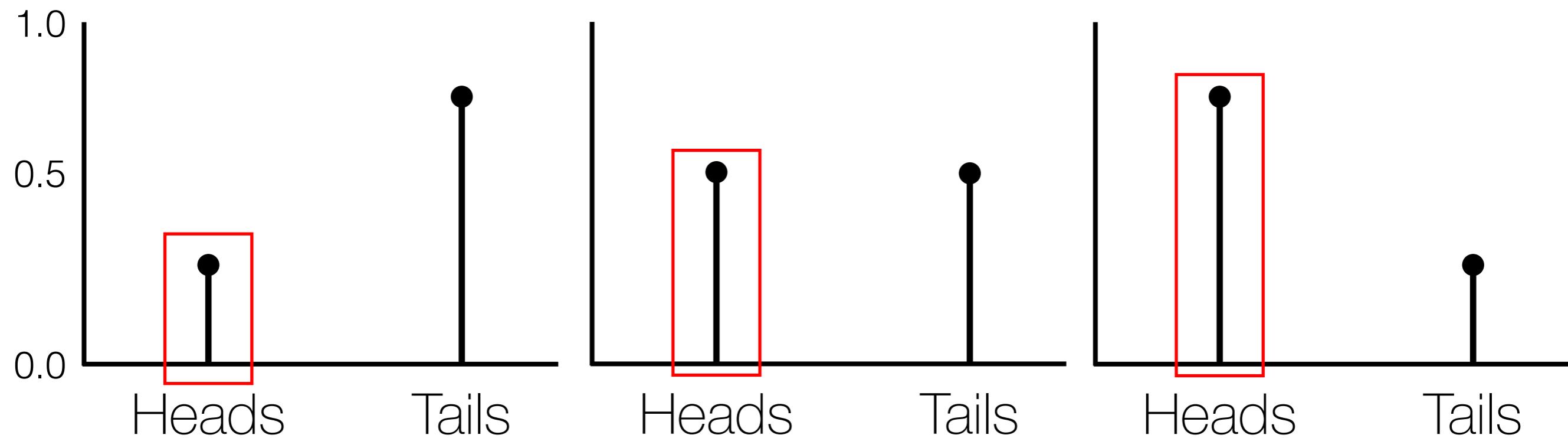
$$p = 0.25$$

```
> p <- 0.5  
> z ~ dnBernoulli(p)  
> z.clamp(1)  
> z.probability()  
0.5
```

$$p = 0.5$$

```
> p <- 0.75  
> z ~ dnBernoulli(p)  
> z.clamp(1)  
> z.probability()  
0.75
```

$$p = 0.75$$



```
> p <- 0.25  
> z ~ dnBernoulli(p)  
> z.clamp(1)  
> z.probability()  
0.25
```

```
> p <- 0.5  
> z ~ dnBernoulli(p)  
> z.clamp(1)
```

```
> p <- 0.75  
> z ~ dnBernoulli(p)  
> z.clamp(1)
```

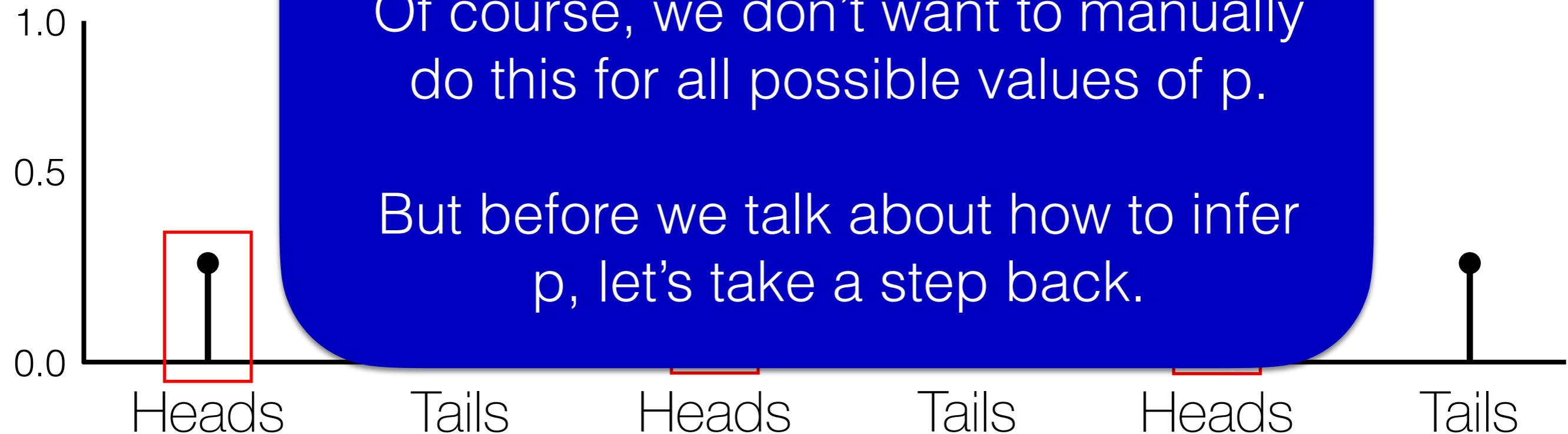
Being able to attach data to the model allows us to compare the likelihoods for different values of  $p$ !

Of course, we don't want to manually do this for all possible values of  $p$ .

But before we talk about how to infer  $p$ , let's take a step back.

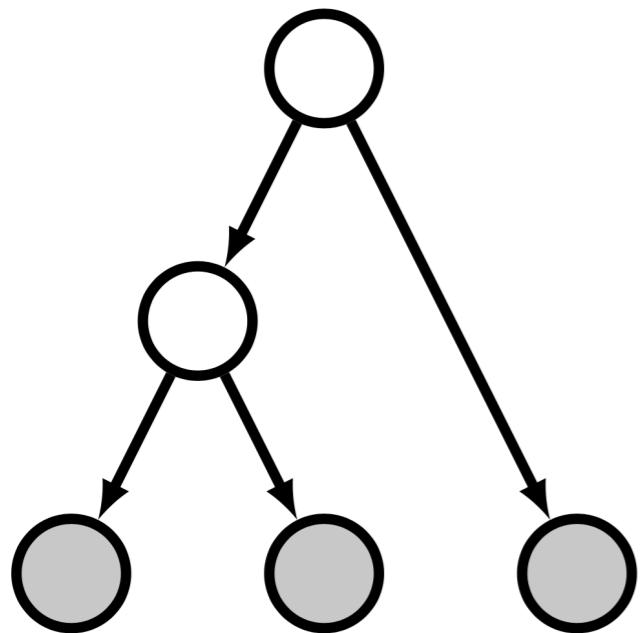
$p$

5



# Graphical Models

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

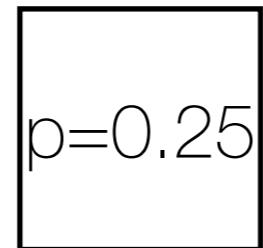


The “graphical” part of the name graphical models has to do with their basis on graphs, and not anything to do with drawing.

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

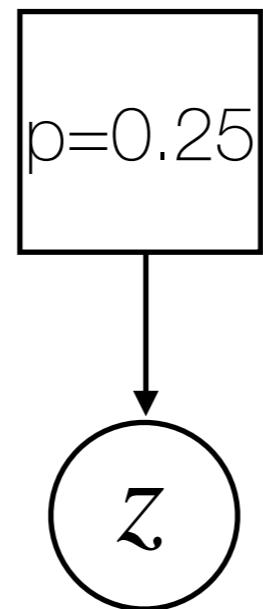
# Building a Graphical Model

$p$  is fixed at 0.25



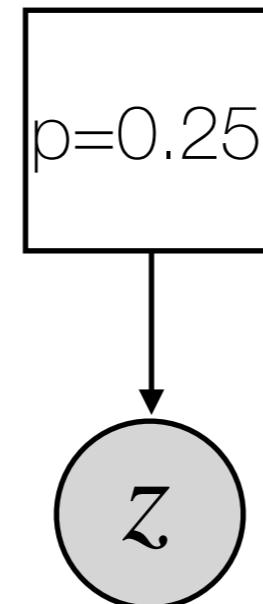
```
[> p <- 0.25
```

$z$  depends on  $p$   
 $z$  is a Bernoulli r.v.

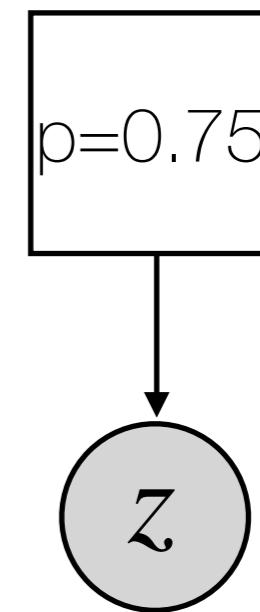
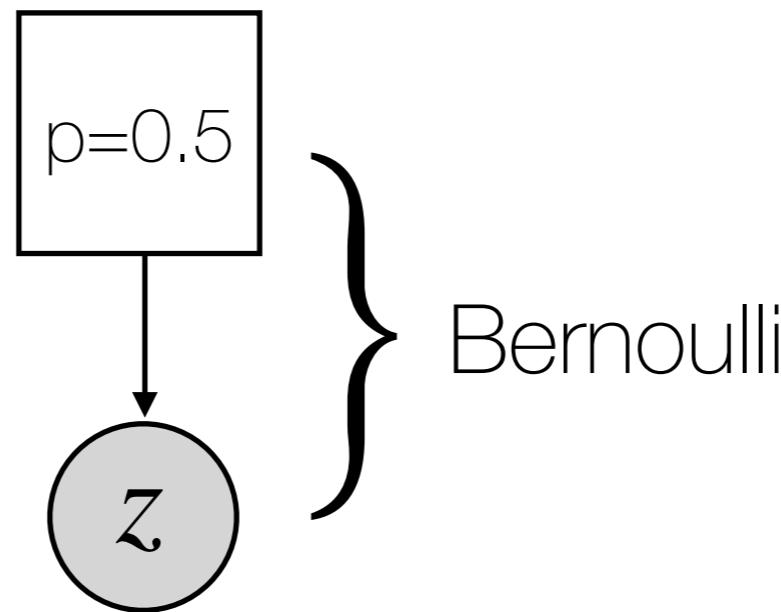
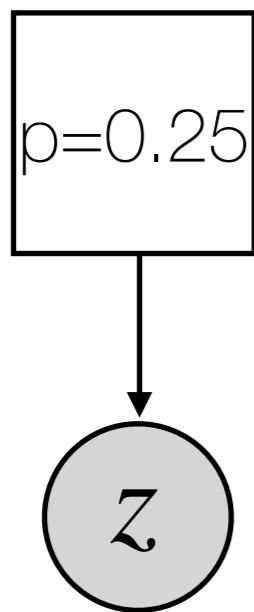


```
[> z ~ dnBernoulli(p)
```

We “observed” a  
value of 1 for  $z$ .



```
[> z.clamp(1)
```



```
> p <- 0.25  
> z ~ dnBernoulli(p)  
> z.clamp(1)  
> z.probability()  
 0.25
```

```
> p <- 0.5  
> z ~ dnBernoulli(p)  
> z.clamp(1)  
> z.probability()  
 0.5
```

```
> p <- 0.75  
> z ~ dnBernoulli(p)  
> z.clamp(1)  
> z.probability()  
 0.75
```

# What are the possible building blocks for graphical models?



# RevBayes Syntax

```
sqrt(16)
```

```
abs(-11)
```

## functions and arguments

```
choose(6,3)
```

```
ls()
```

```
max([2.1,5.5,7.8])
```

```
test <- v(5,9,12)  
test  
test[2]
```

## vectors

```
test[1] <- 5  
test[2] <- 9  
test[3] <- 12  
test
```

```
for (i in 1:5){  
    print(i)  
}
```

## for loops

```
for (i in 1:3){  
    print(test[i])  
}
```

<https://revbayes.github.io/tutorials/intro/rev.html>

<https://revbayes.github.io/documentation/>

# Graphical Model Syntax

## *Symbol*



## *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*

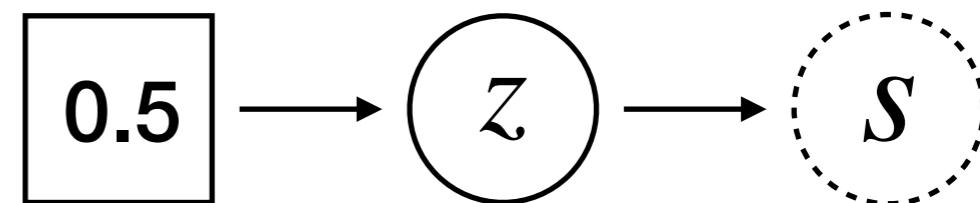
Conditional Dependency

*Description*

Arrows indicate conditional dependencies and are directional. The node (variable) that the arrow points to depends on the value of the node at the other end of the arrow. For instance, if  $p$  points to  $Z$ , then we say that  $Z$  is dependent on  $p$ .

*Rev Example*

```
p <- 0.5  
z ~ dnBernoulli(p)  
s := z^3
```



# Graphical Model Syntax

## *Symbol*



## *Type*

Plate

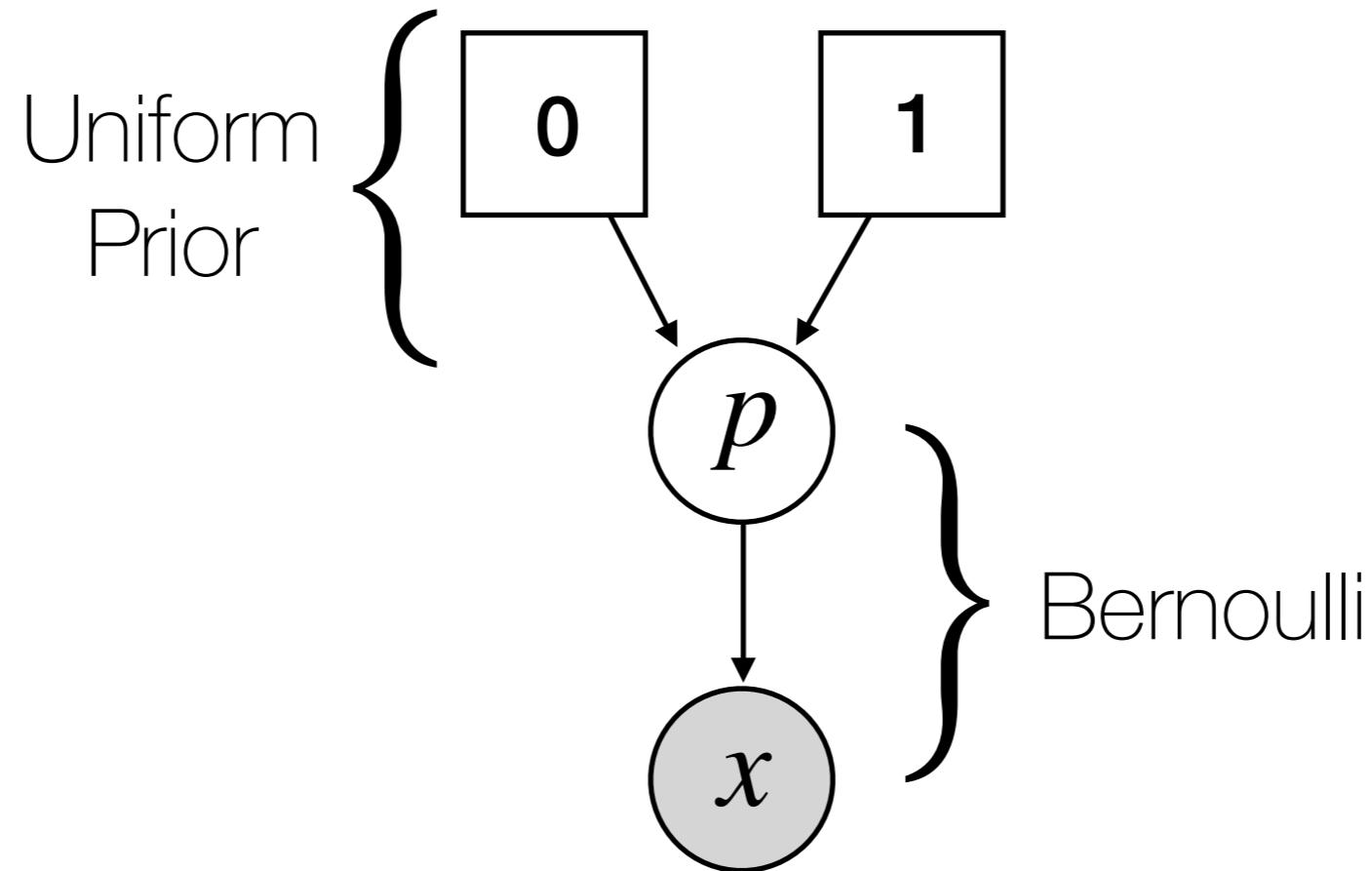
## *Description*

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*

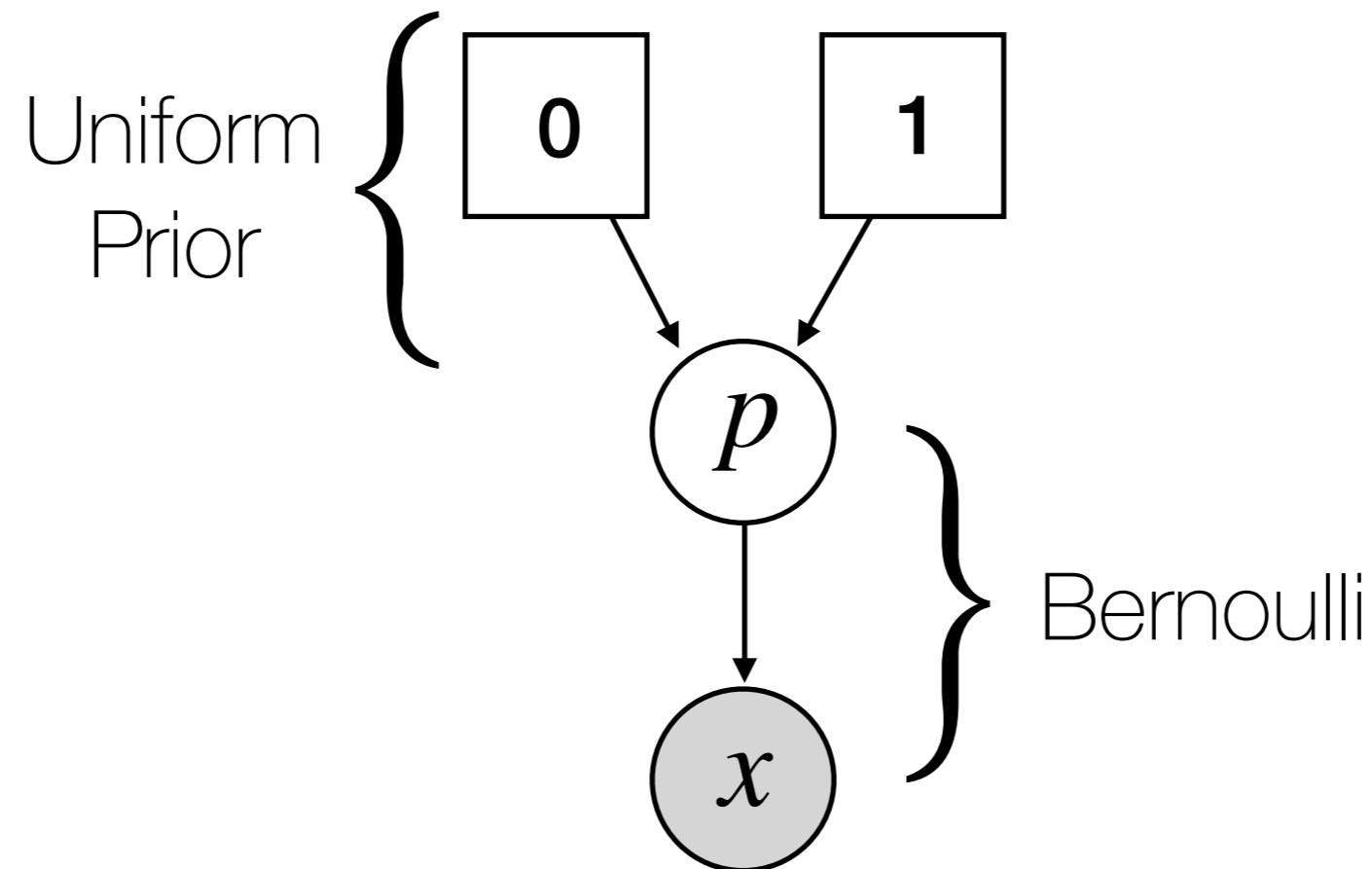
```
N <- 10
for (i in 1:N){
    z[i] ~ dnBernoulli(0.5)
    z[i].clamp(1)
}
```

# Single Bernoulli Trial (Bayesian)



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

# Single Bernoulli Trial (Bayesian)

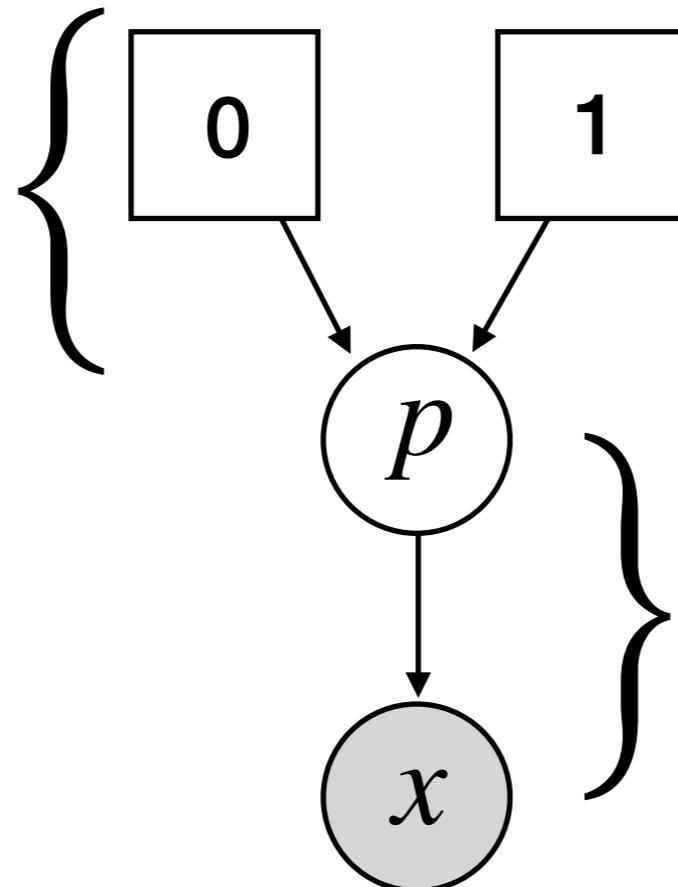


```
p ~ dnUnif(0,1)  
x ~ dnBernoulli(p)  
x.clamp(0)
```

# Single Bernoulli Trial (Bayesian)

$P(p)$   
Prior

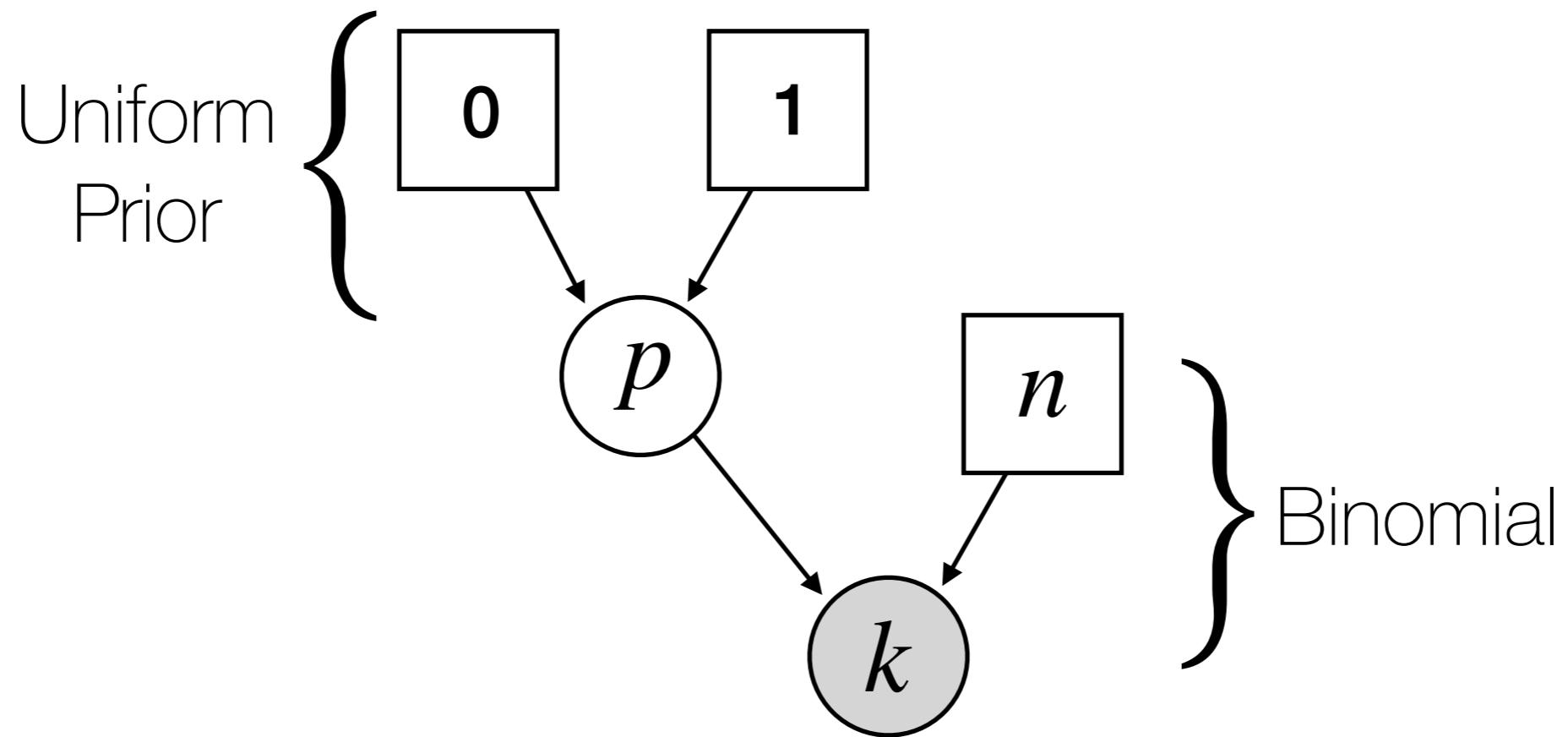
Uniform  
Prior



$P(D | p)$   
Likelihood

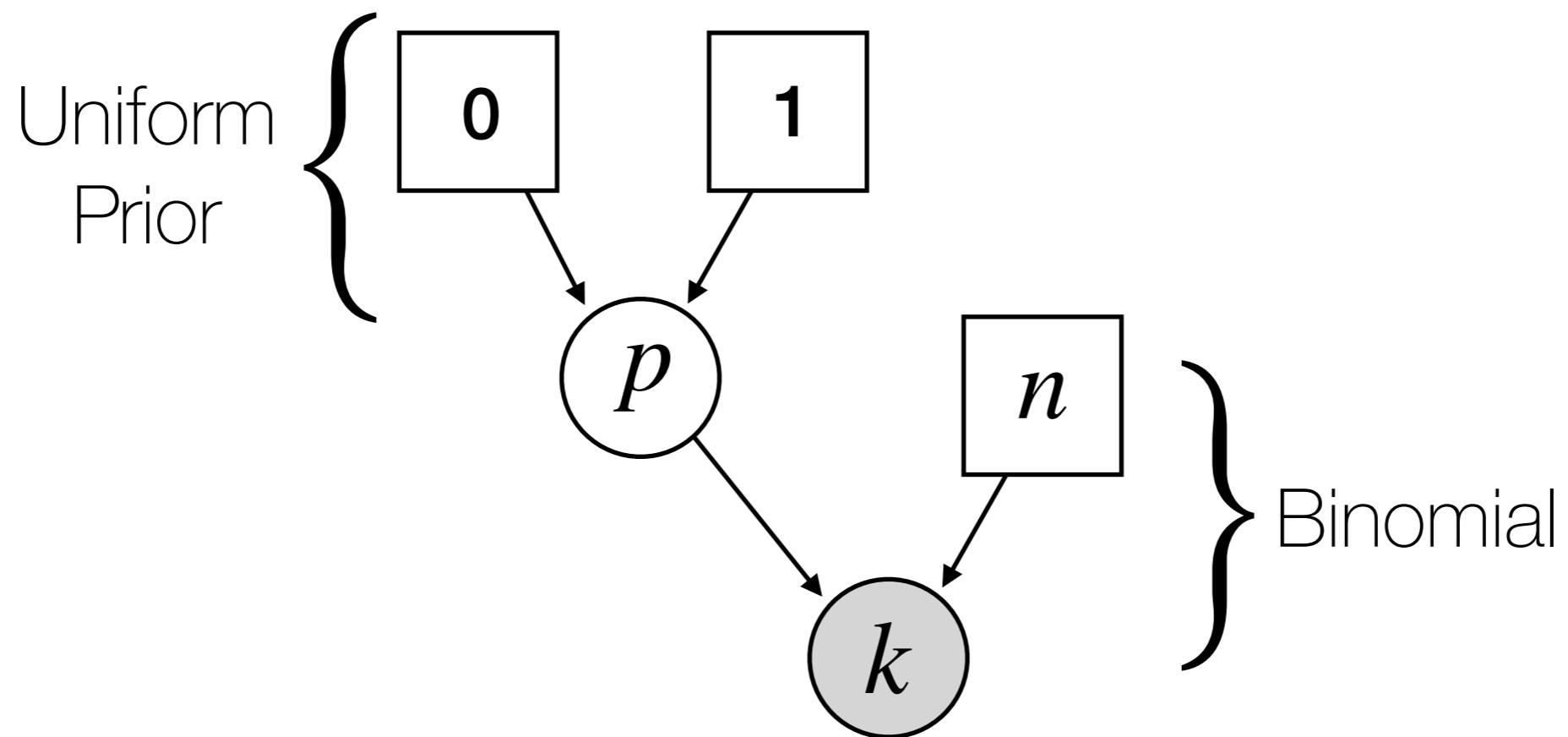
```
p ~ dnUnif(0,1)
x ~ dnBernoulli(p)
x.clamp(0)
```

# Single Binomial (Bayesian)



How does this model relate to the Bernoulli models?

# Single Binomial (Bayesian)

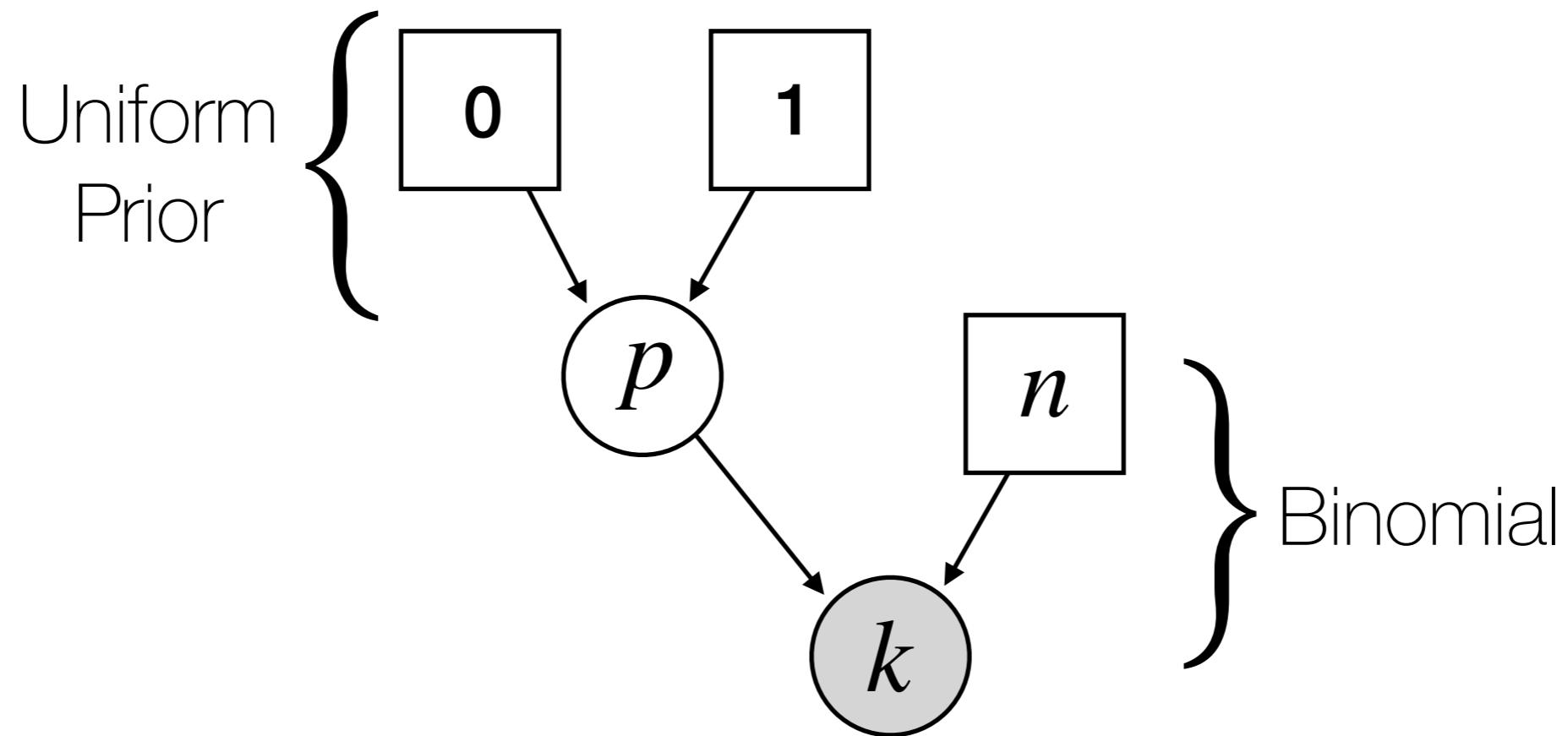


```
p ~ dnUnif(0,1)
```

```
n <- 5
```

```
k ~ dnBinomial(n,p)  
k.clamp(3)
```

# Setting up MCMC in RevBayes

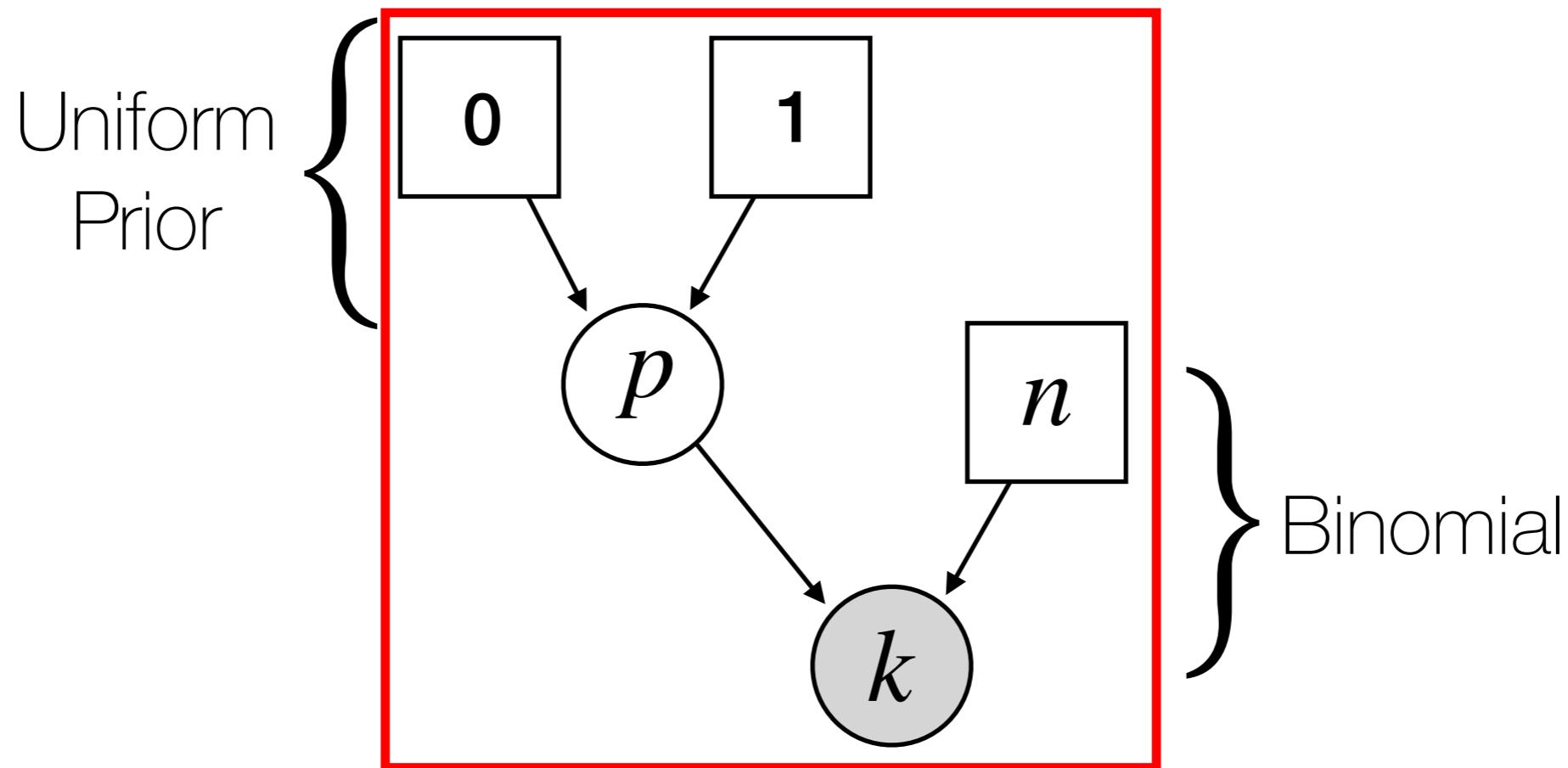


First, create a model object. You can pass any of the nodes to the constructor as a “handle”.

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

NOTE: We use the `=` assignment operator for “workspace” variables.

# Setting up MCMC in RevBayes

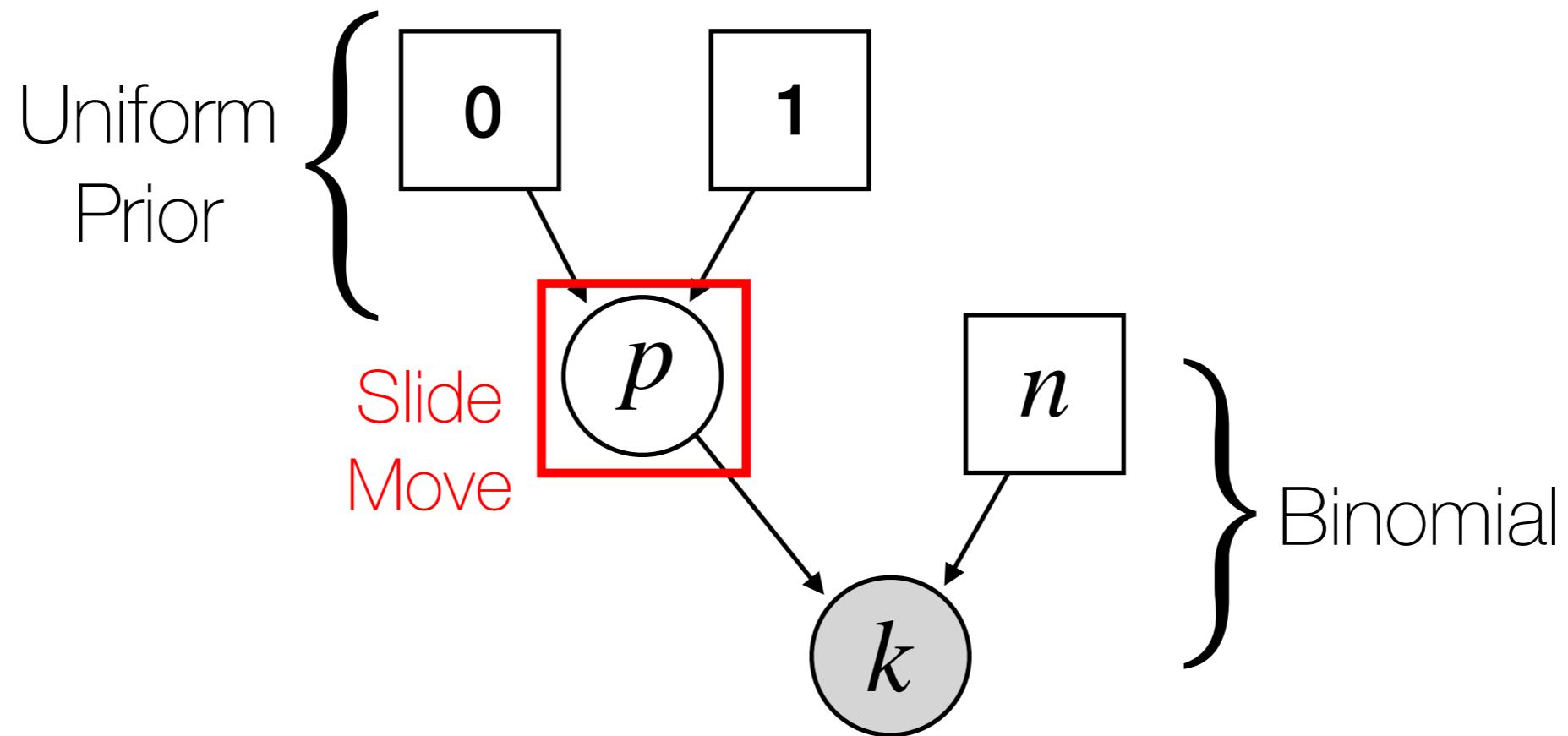


First, create a model object. You can pass any of the nodes to the constructor as a “handle”.

```
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 = VectorMoves()  
moves.append( mvSlide(p,delta=0.1,weight=1) )
```

Many different move types are available in RevBayes.

# Setting up MCMC in RevBayes

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

```
monitors = VectorMonitors()
monitors.append( mnScreen(printgen=1000,p) )
monitors.append( mnModel(filename="myMCMC.log", printgen=10) )
```

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	

# Setting up MCMC in RevBayes

Next we create an MCMC object.

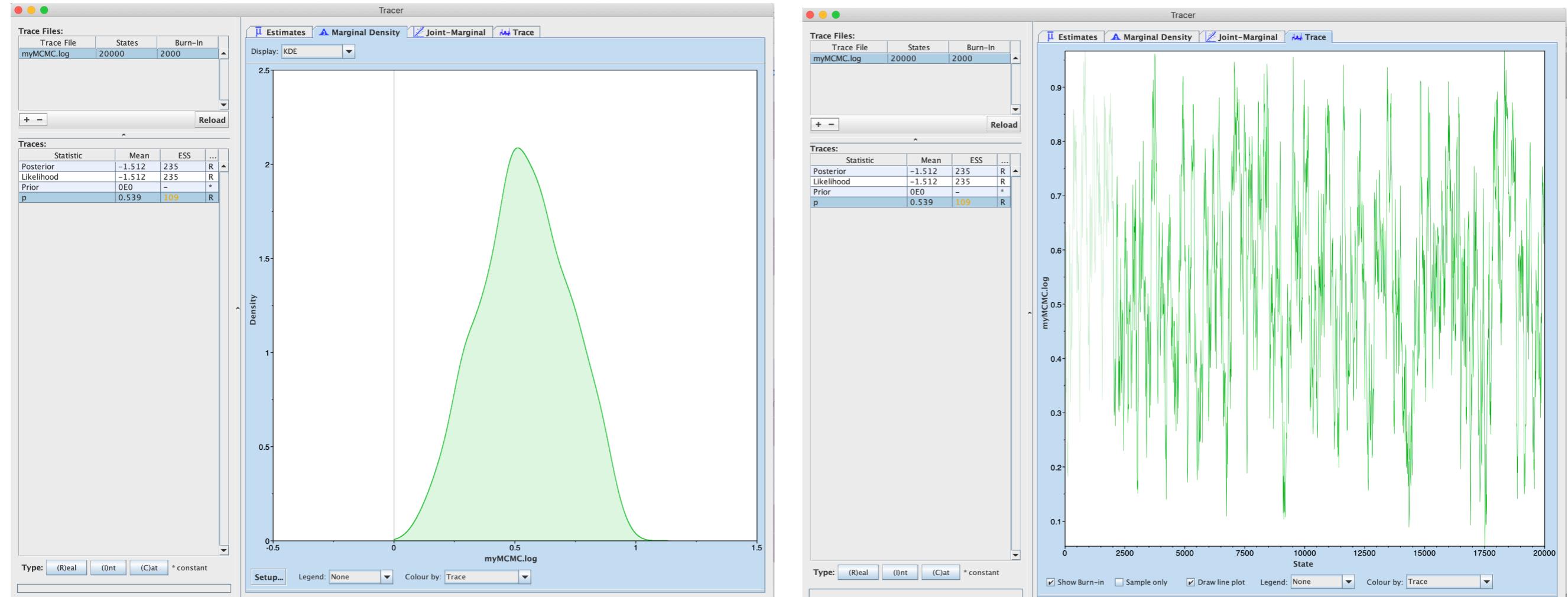
```
myMCMC = mcmc(myModel,moves,monitors)
```

Now, we start the MCMC!

```
myMCMC.run(20000)
```

**Wait** - what does this 20,000 mean?

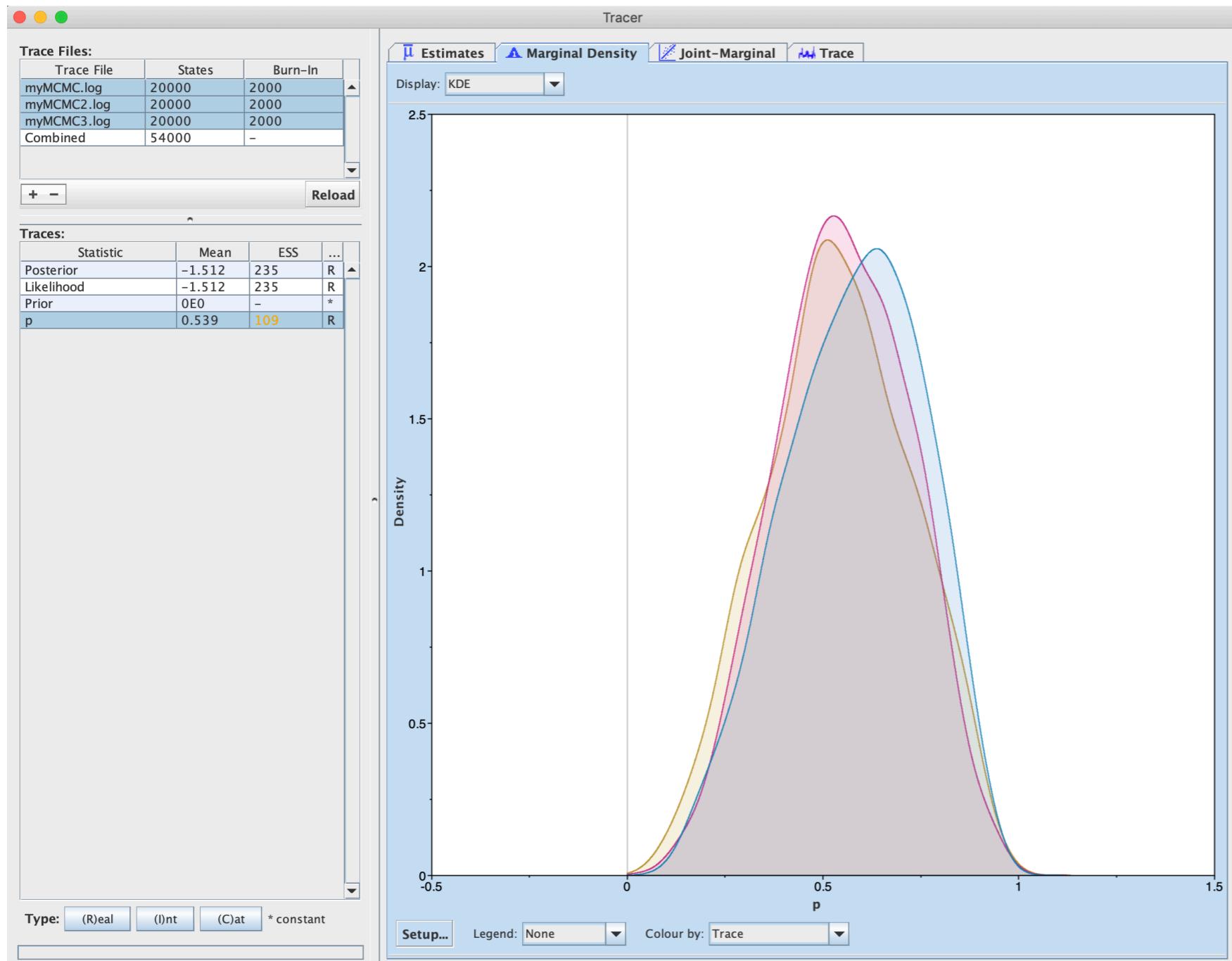
# Tracer



Marginal Distribution

Trace Plot

# Assessing Convergence



Traces:

Statistic	Mean	ESS	
Posterior	-1.512	235	R
Likelihood	-1.512	235	R
Prior	0E0	-	*
p	0.539	109	R

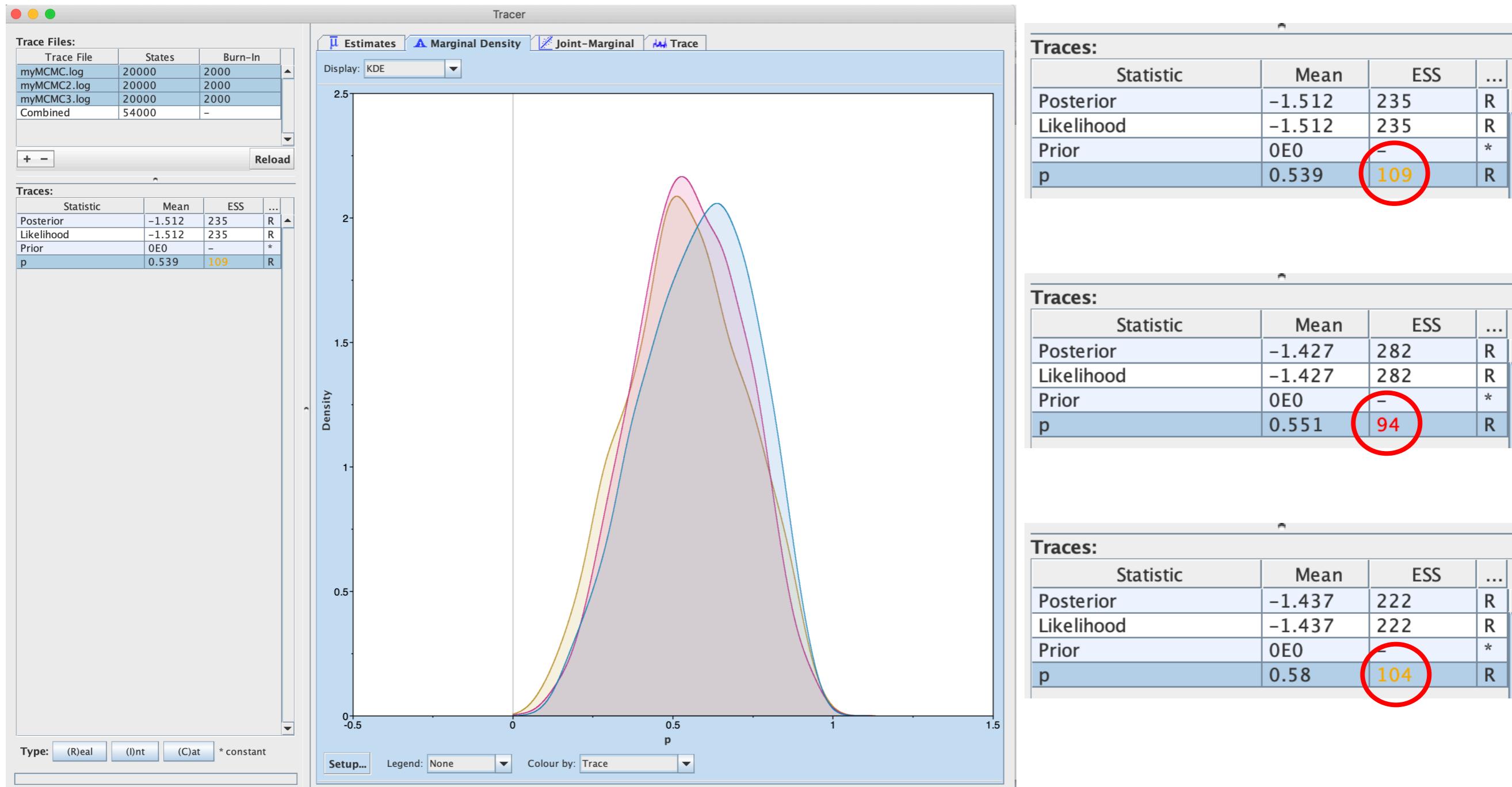
Traces:

Statistic	Mean	ESS	
Posterior	-1.427	282	R
Likelihood	-1.427	282	R
Prior	0E0	-	*
p	0.551	94	R

Traces:

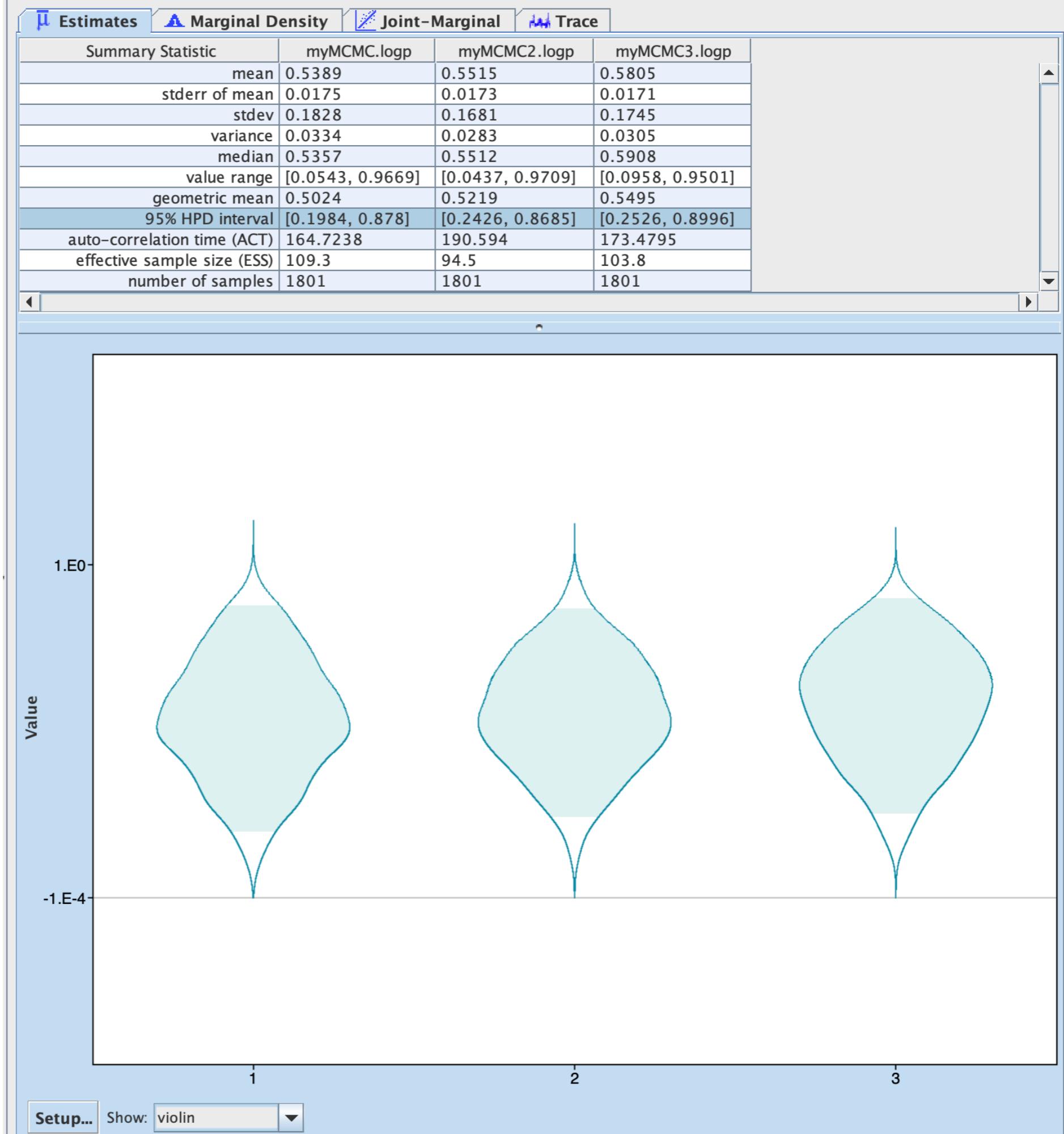
Statistic	Mean	ESS	
Posterior	-1.437	222	R
Likelihood	-1.437	222	R
Prior	0E0	-	*
p	0.58	104	R

# Assessing Convergence



How many samples did we draw during our MCMC analyses?  
Jeremy M. Brown

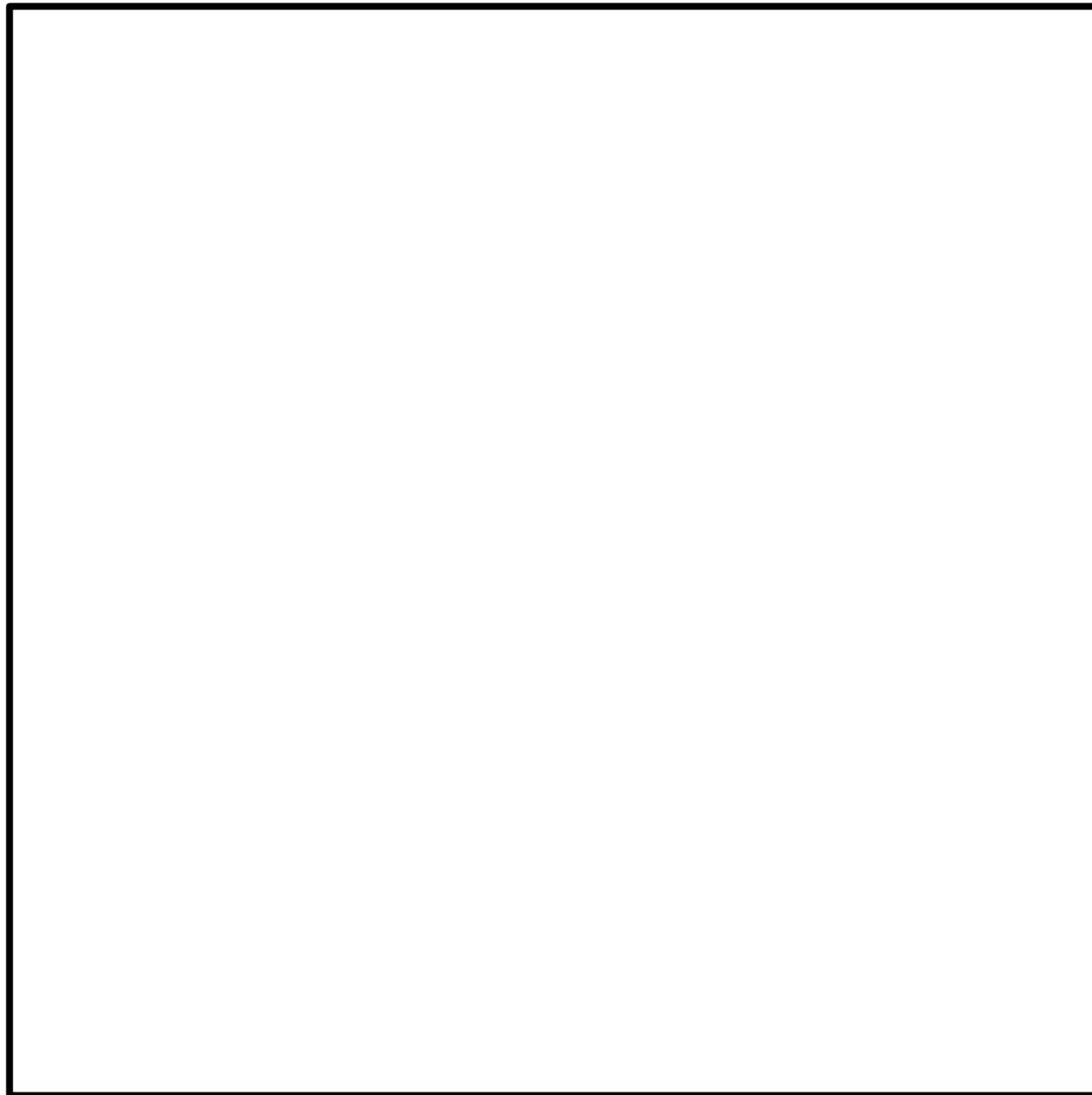
# 95% Highest Posterior Density (HPD) Intervals



# Phylogenetic Graphical Models

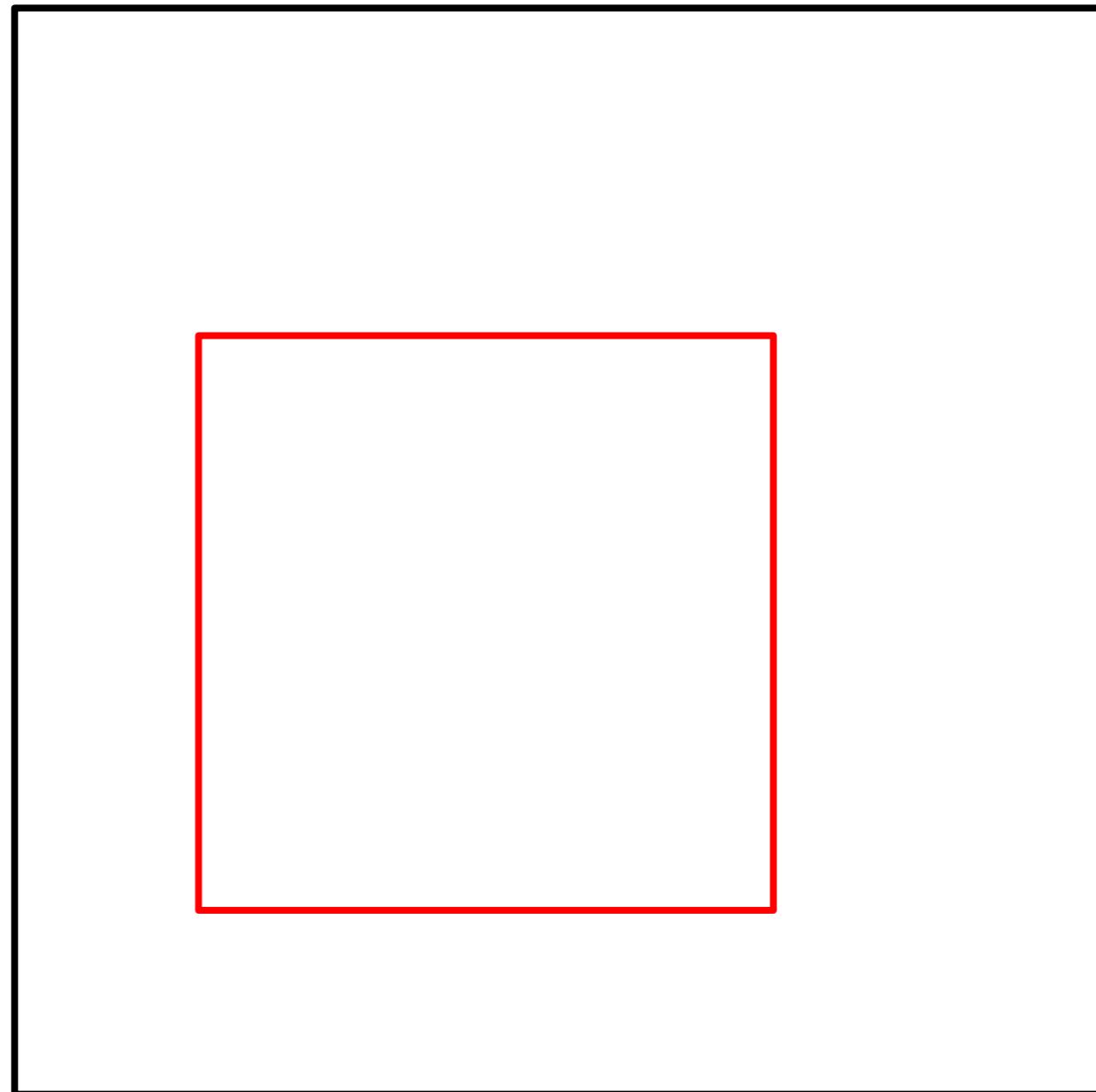
# Model Space (conceptual)

All  
Possible  
Ways  
Genes  
Could  
Evolve



# Model Space (conceptual)

All  
Possible  
Ways  
Genes  
Could  
Evolve

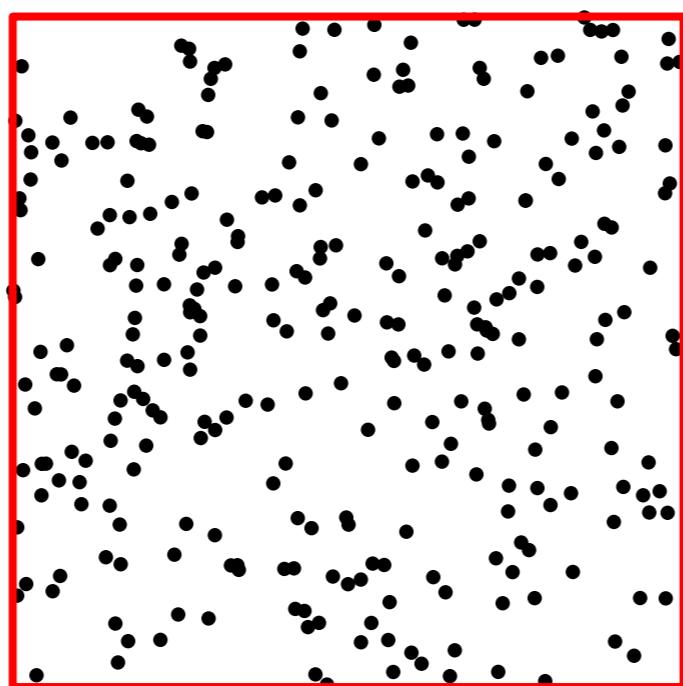


The  
Types of  
Evolution  
That Are  
Built Into  
Many  
Programs

# Our Hope

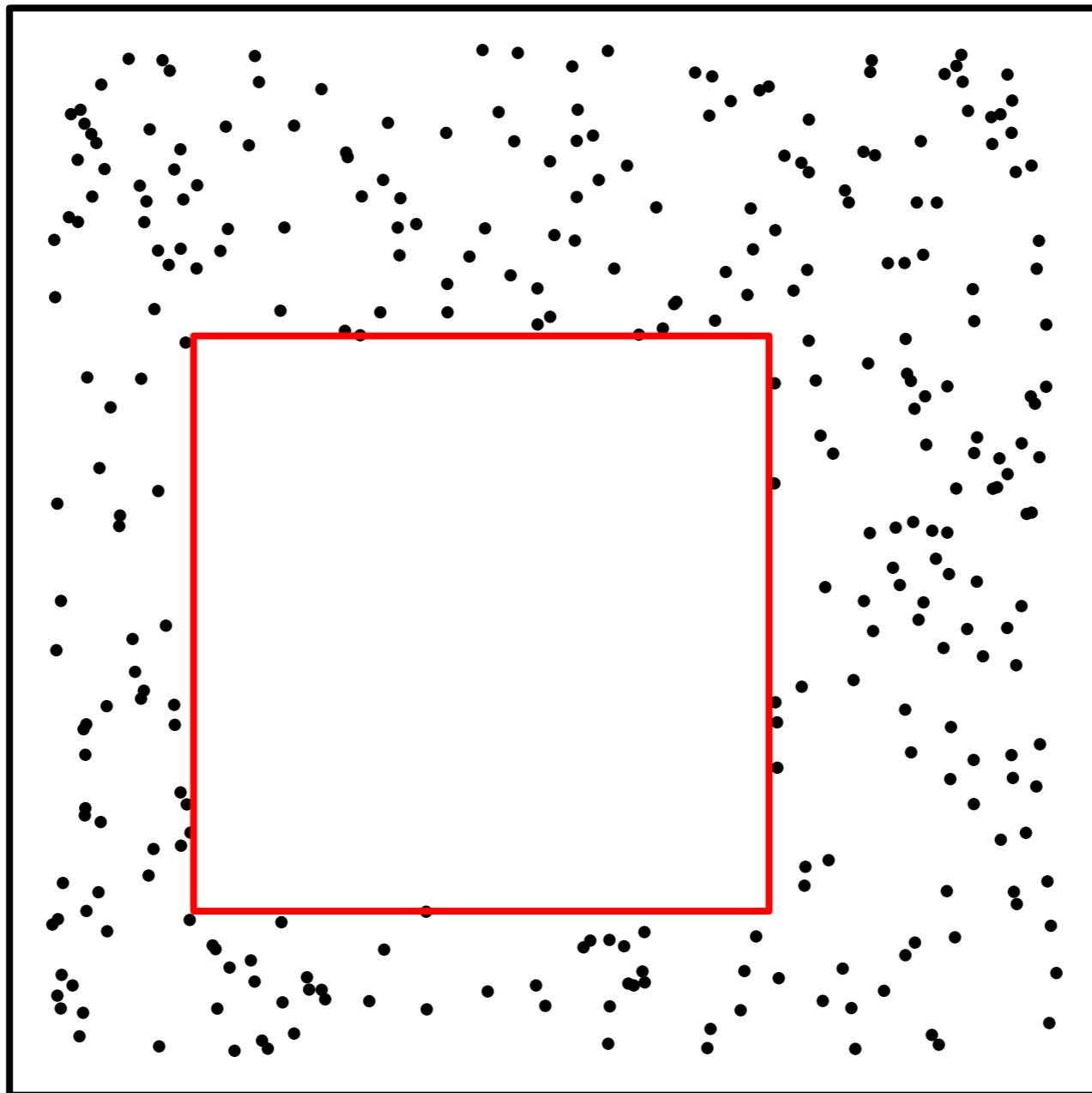
All  
Possible  
Ways  
Genes  
Could  
Evolve

The  
Types of  
Evolution  
That Are  
Built Into  
Many  
Programs



# Our Fears

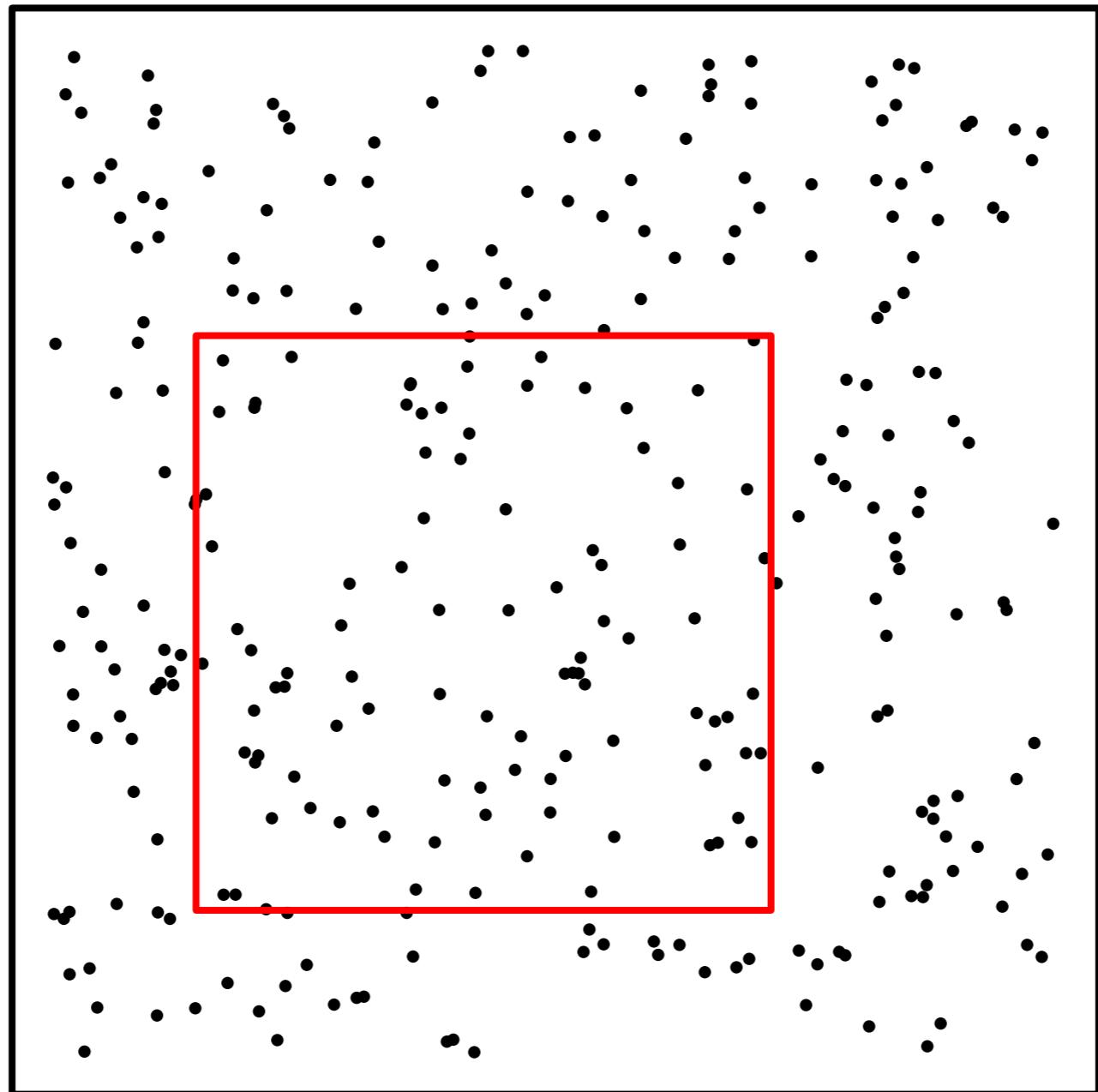
All  
Possible  
Ways  
Genes  
Could  
Evolve



The  
Types of  
Evolution  
That Are  
Built Into  
Many  
Programs

# The Probable Truth

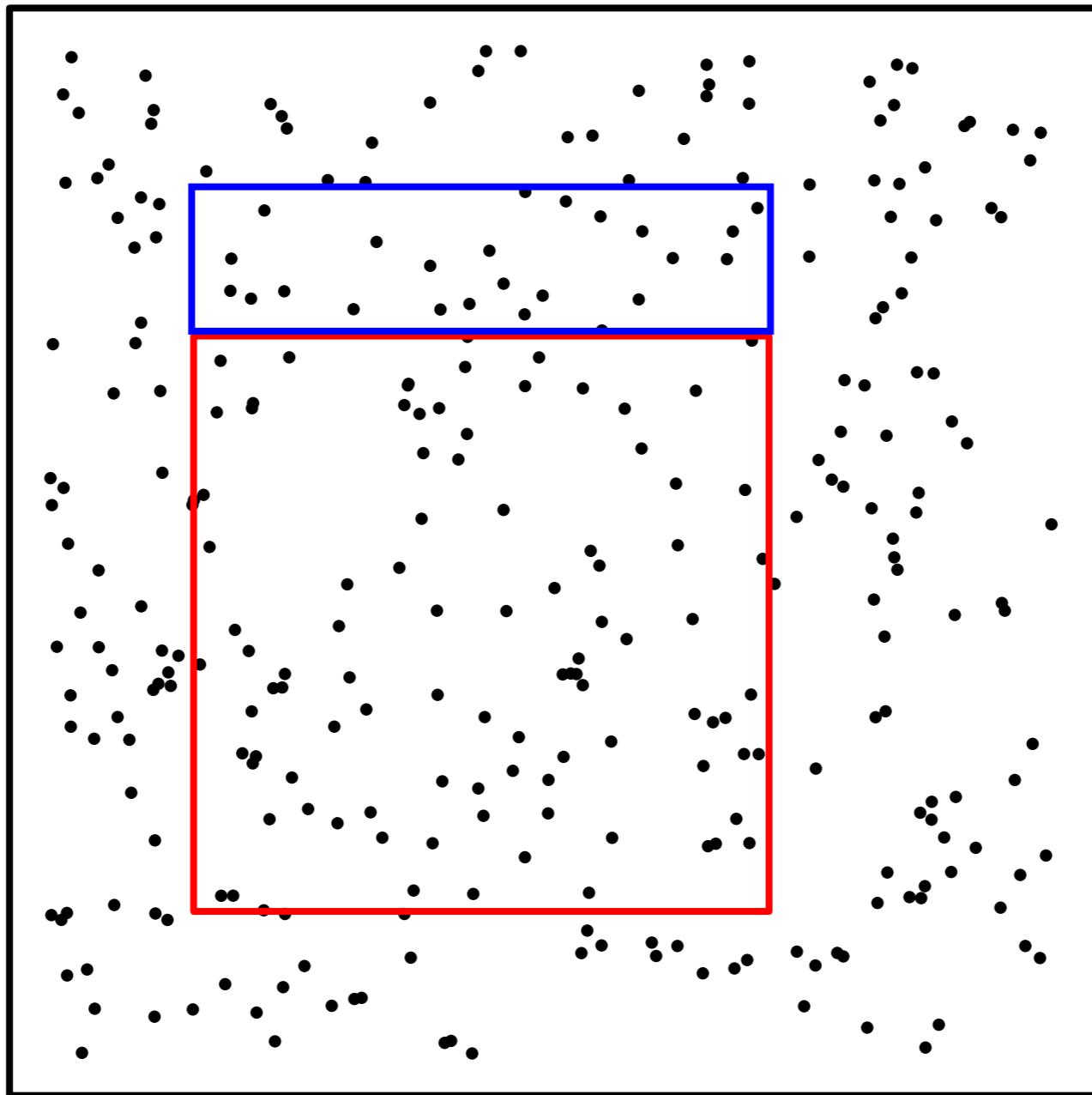
All  
Possible  
Ways  
Genes  
Could  
Evolve



The  
Types of  
Evolution  
That Are  
Built Into  
Many  
Programs

# The Probable Truth

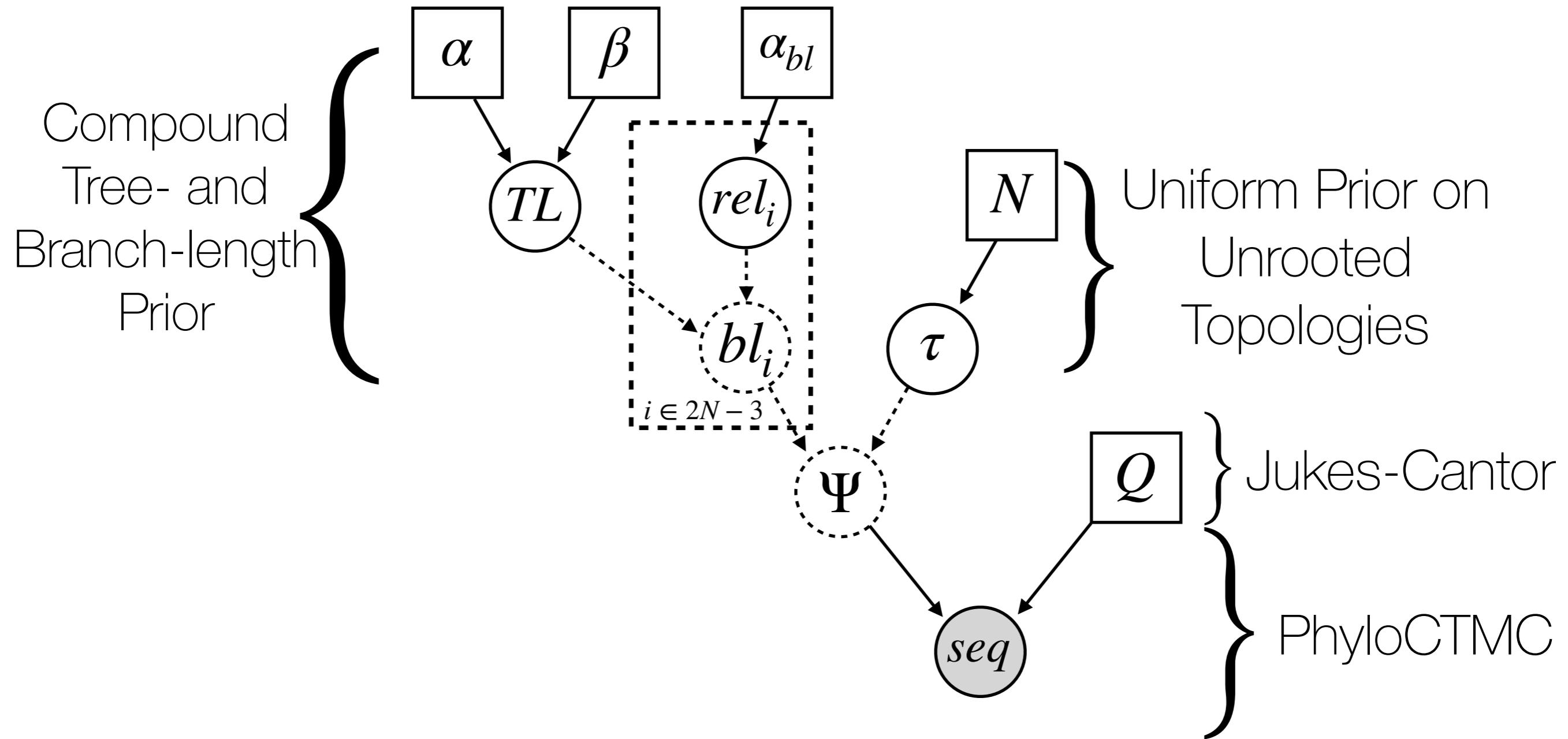
All  
Possible  
Ways  
Genes  
Could  
Evolve



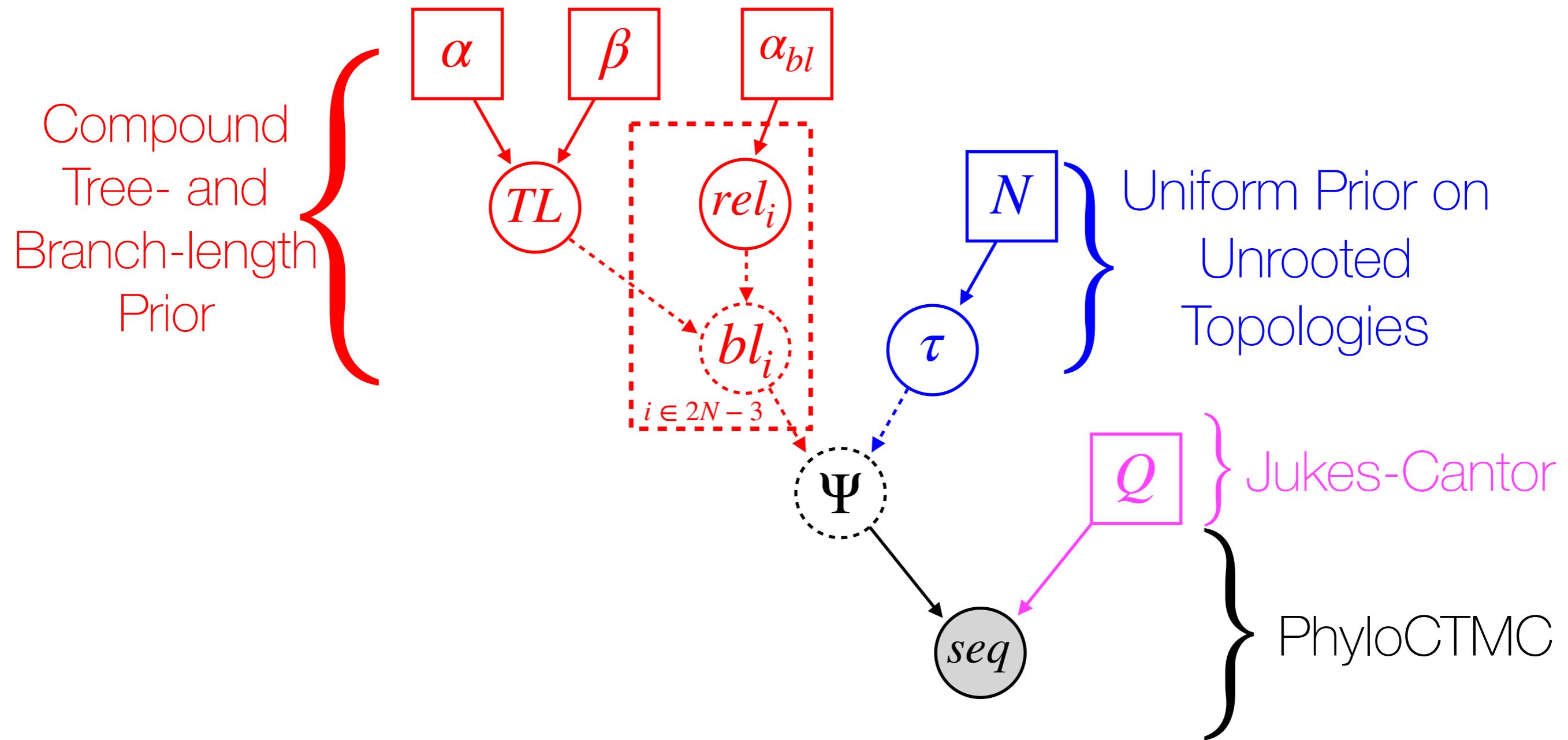
The  
Types of  
Evolution  
That Are  
Built Into  
Many  
Programs

Types of Evolution that You'd Like to Try!

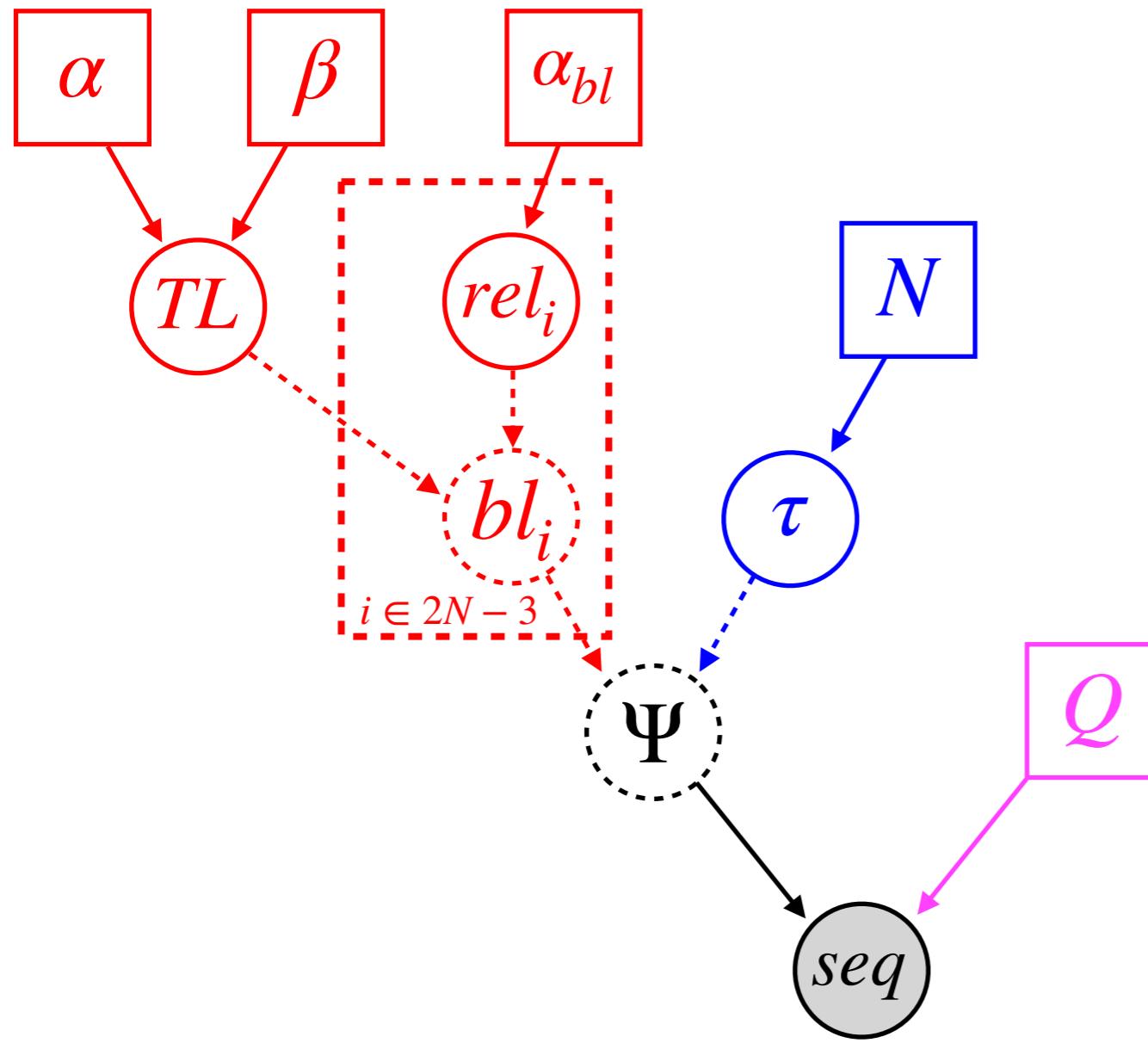
# Jukes-Cantor



# Jukes-Cantor



# Jukes-Cantor



```
data = readDiscreteCharacterData("myData.nex")
```

```
taxa <- data.taxa()
```

```
n_taxa <- data.ntaxa()
```

```
n_branches <- 2 * n_taxa - 3
```

```
topology ~ dnUniformTopology(taxa)
```

```
alpha <- 2
```

```
beta <- 4
```

```
TL ~ dnGamma(alpha,beta)
```

```
alpha_bl <- 1.0
```

```
rel_branch_lengths ~ dnDirichlet( rep(alpha_bl,n_branches) )
```

```
br_lens := rel_branch_lengths * TL
```

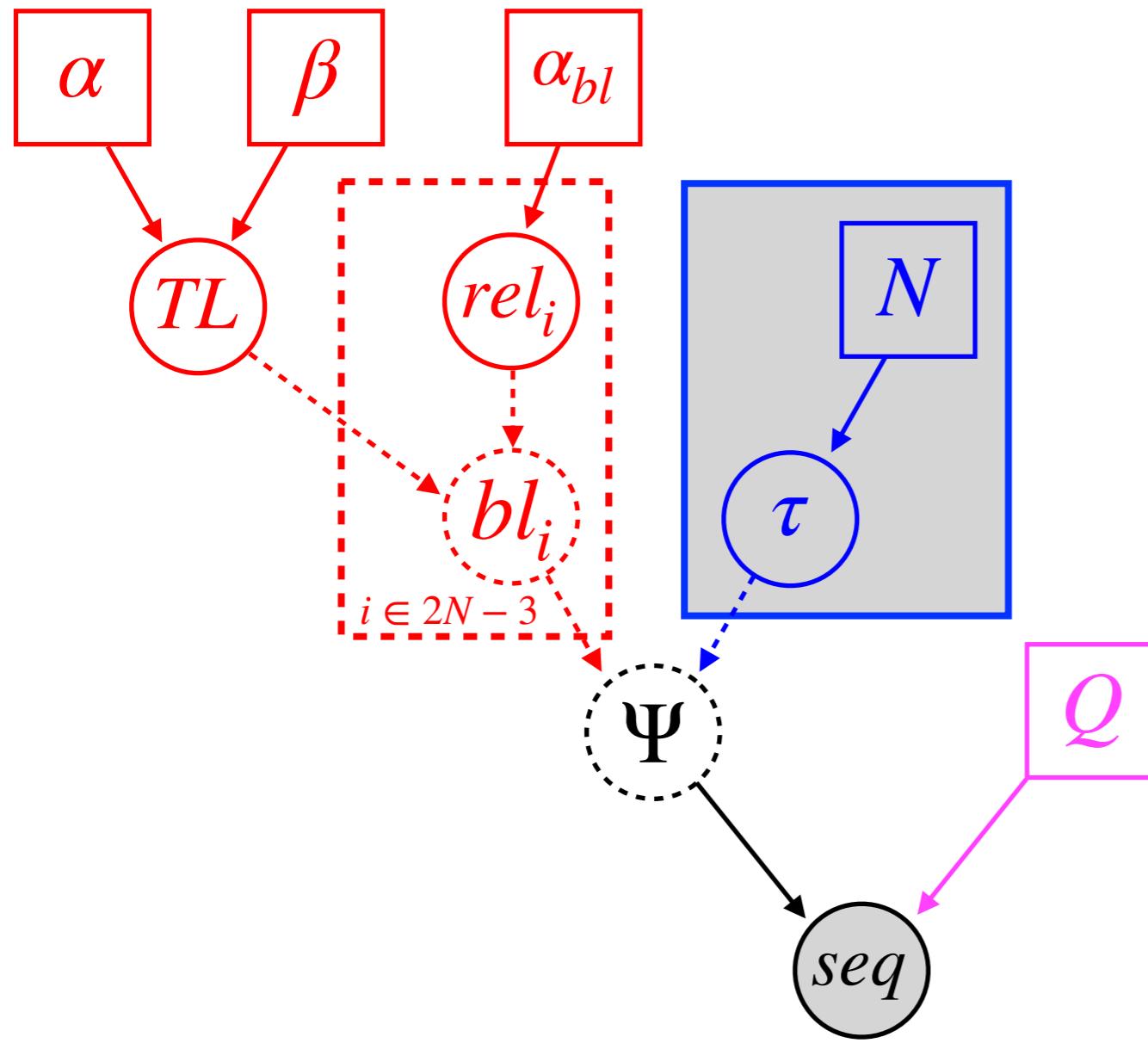
```
Q <- fnJC(4)
```

```
psi := treeAssembly(topology, br_lens)
```

```
seq ~ dnPhyloCTMC(tree=psi,Q=Q,type="DNA")
```

```
seq.clamp(data)
```

# Jukes-Cantor



```

data = readDiscreteCharacterData("myData.nex")
taxa <- data.taxa()
n_taxa <- data.ntaxa()
n_branches <- 2 * n_taxa - 3

topology ~ dnUniformTopology(taxa)
  
```

alpha <- 2  
beta <- 4

TL ~ dnGamma(alpha,beta)

alpha\_bl <- 1.0

rel\_branch\_lengths ~ dnDirichlet( rep(alpha\_bl,n\_branches) )

br\_lens := rel\_branch\_lengths \* TL

Q <- fnJC(4)

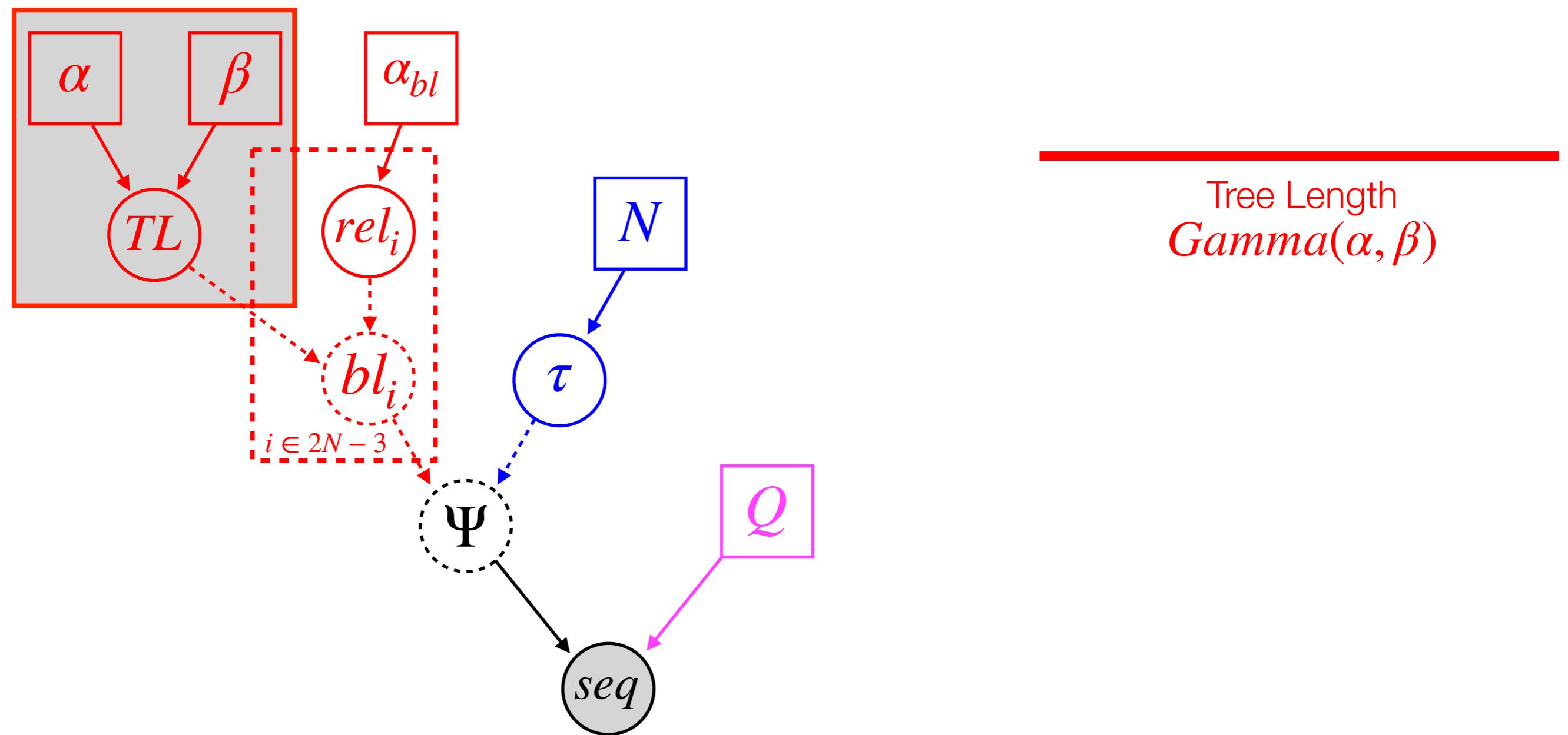
psi := treeAssembly(topology, br\_lens)

seq ~ dnPhyloCTMC(tree=psi,Q=Q,type="DNA")

seq.clamp(data)

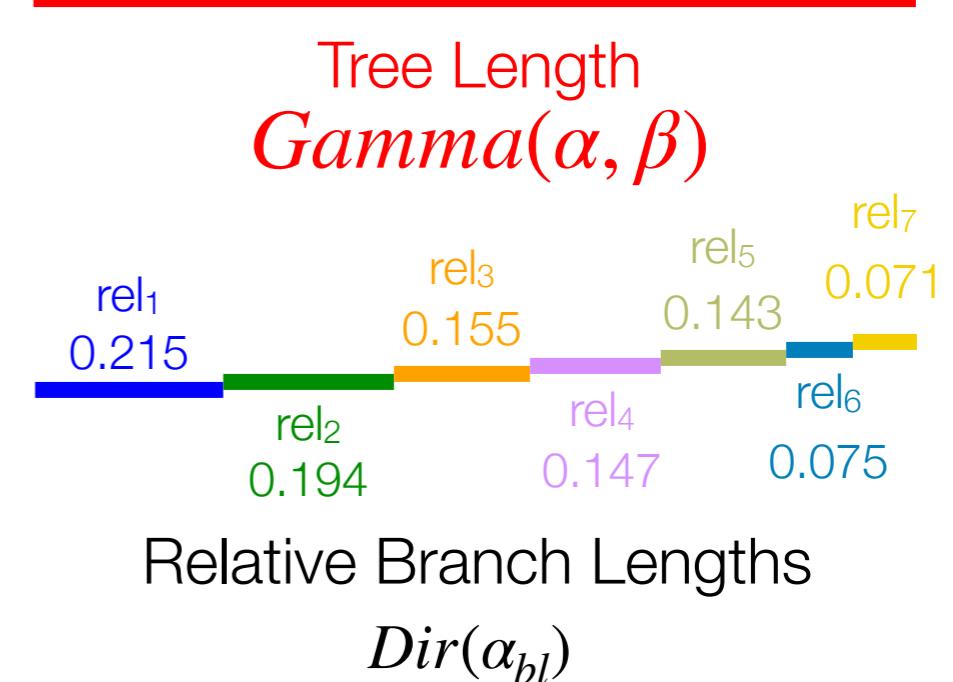
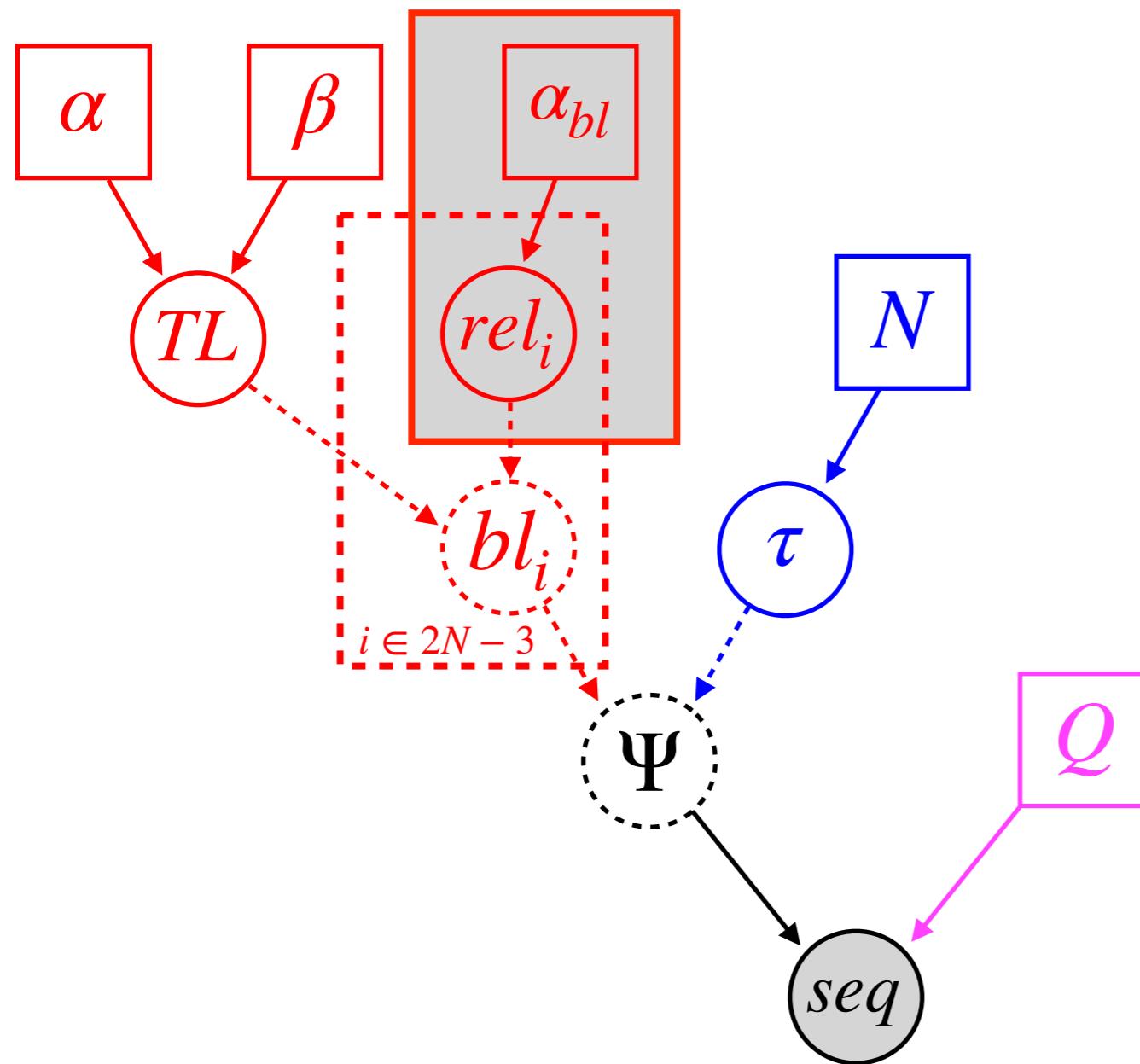
# Jukes-Cantor

(Compound Dirichlet tree- and branch-length prior)



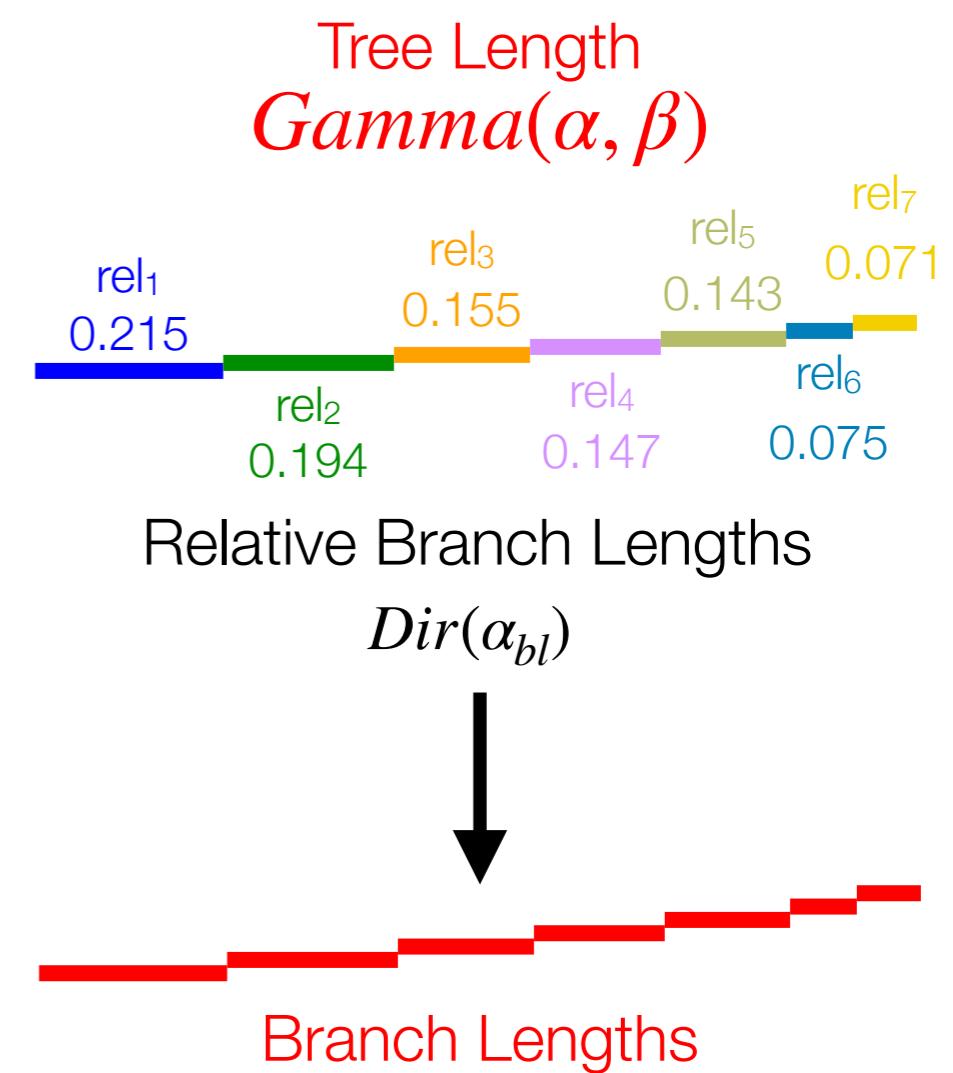
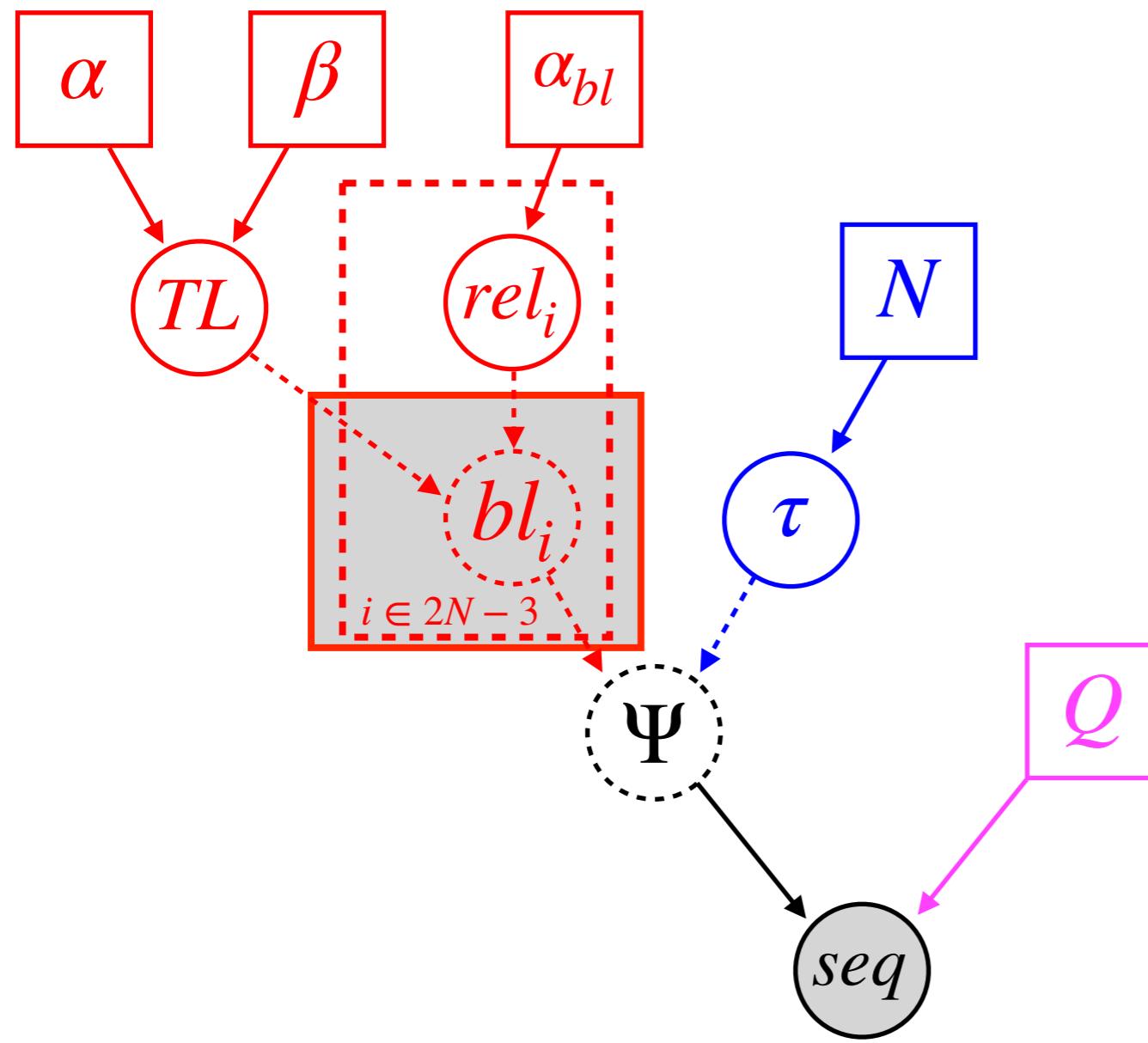
# Jukes-Cantor

(Compound Dirichlet tree- and branch-length prior)

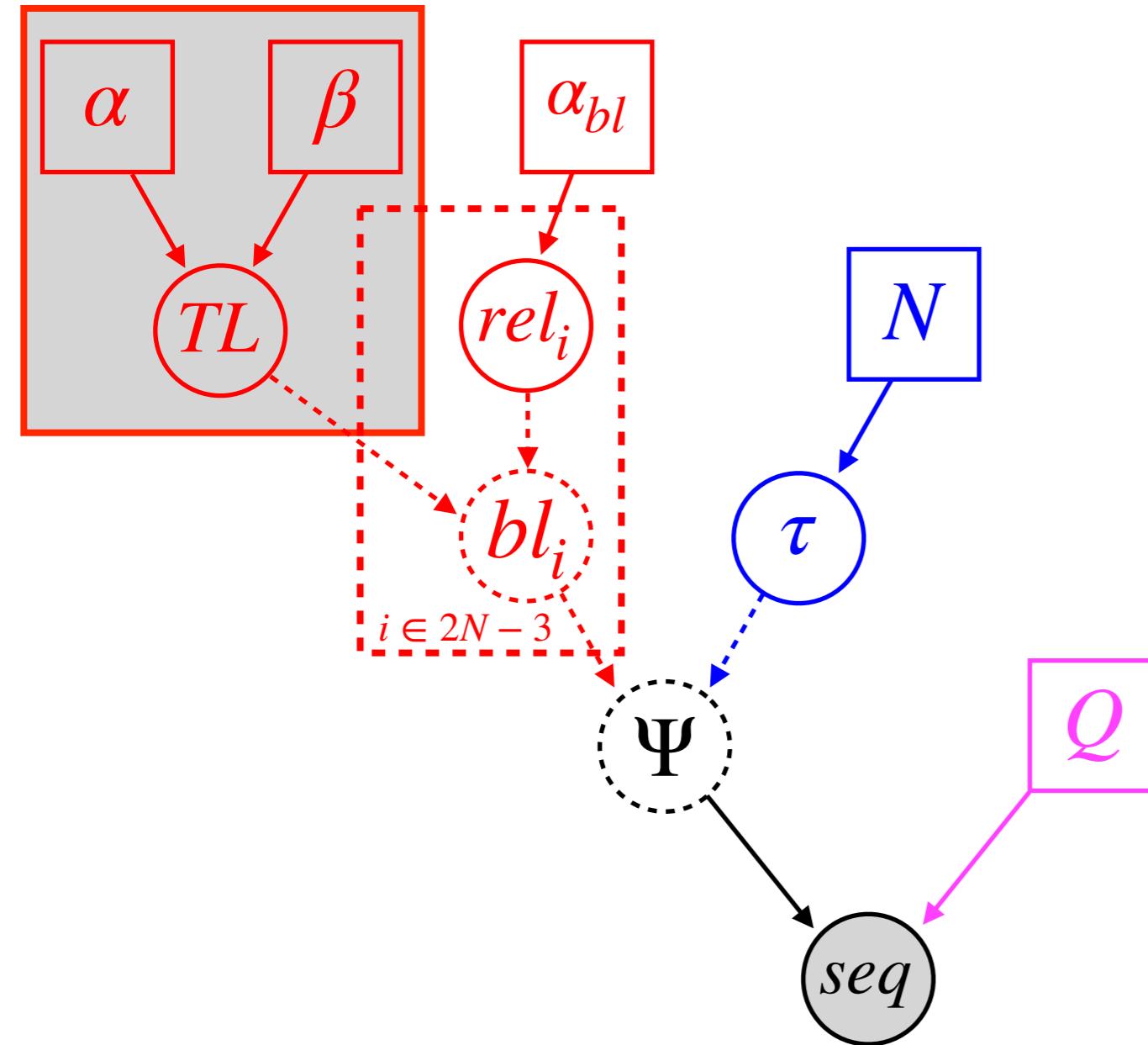


# Jukes-Cantor

(Compound Dirichlet tree- and branch-length prior)



# Jukes-Cantor



```

data = readDiscreteCharacterData("myData.nex")
taxa <- data.taxa()
n_taxa <- data.ntaxa()
n_branches <- 2 * n_taxa - 3

```

```
topology ~ dnUniformTopology(taxa)
```

```

alpha <- 2
beta <- 4
TL ~ dnGamma(alpha,beta)

```

```
alpha_bl <- 1.0
```

```
rel_branch_lengths ~ dnDirichlet(rep(alpha_bl,n_branches))
```

```
br_lens := rel_branch_lengths * TL
```

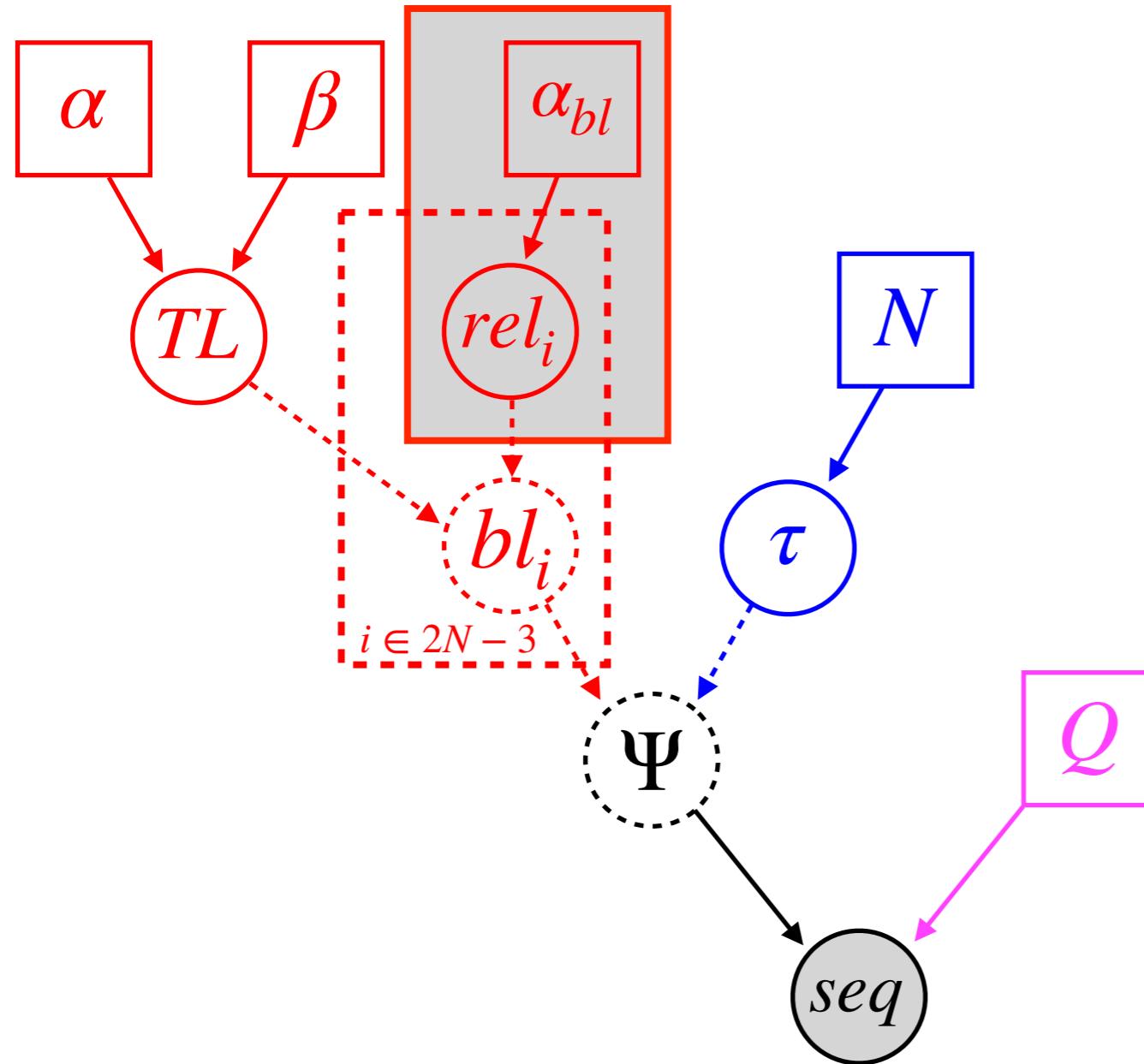
```
Q <- fnJC(4)
```

```
psi := treeAssembly(topology, br_lens)
```

```
seq ~ dnPhyloCTMC(tree=psi,Q=Q,type="DNA")
```

```
seq.clamp(data)
```

# Jukes-Cantor



```
data = readDiscreteCharacterData("myData.nex")
```

```
taxa <- data.taxa()
```

```
n_taxa <- data.ntaxa()
```

```
n_branches <- 2 * n_taxa - 3
```

```
topology ~ dnUniformTopology(taxa)
```

```
alpha <- 2
```

```
beta <- 4
```

```
TL ~ dnGamma(alpha,beta)
```

```
alpha_bl <- 1.0
```

```
rel_branch_lengths ~ dnDirichlet(rep(alpha_bl,n_branches))
```

```
br_lens := rel_branch_lengths * TL
```

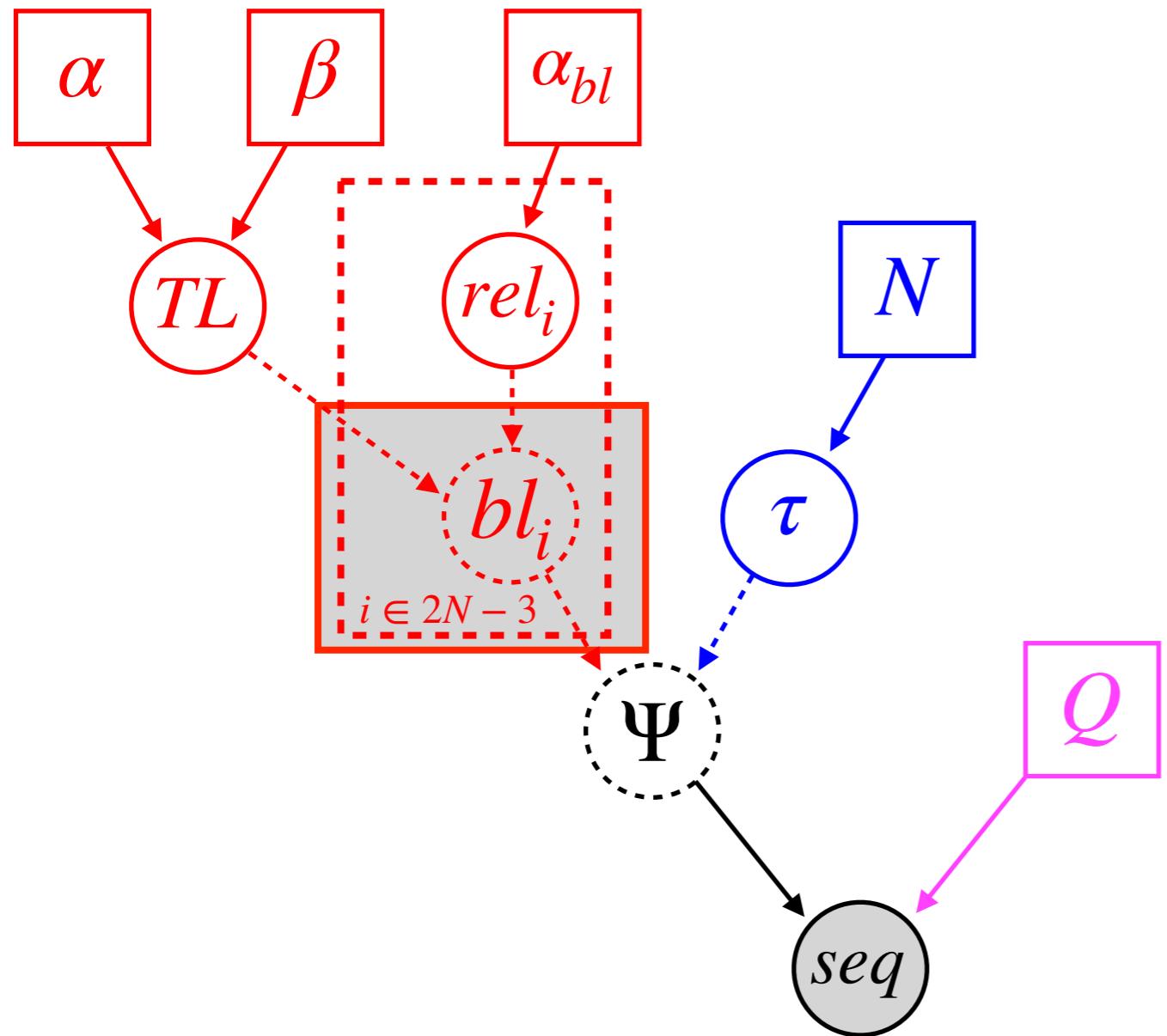
```
Q <- fnJC(4)
```

```
psi := treeAssembly(topology, br_lens)
```

```
seq ~ dnPhyloCTMC(tree=psi,Q=Q,type="DNA")
```

```
seq.clamp(data)
```

# Jukes-Cantor



```
data = readDiscreteCharacterData("myData.nex")
```

```
taxa <- data.taxa()
```

```
n_taxa <- data.ntaxa()
```

```
n_branches <- 2 * n_taxa - 3
```

```
topology ~ dnUniformTopology(taxa)
```

```
alpha <- 2
```

```
beta <- 4
```

```
TL ~ dnGamma(alpha,beta)
```

```
alpha_bl <- 1.0
```

```
rel_branch_lengths ~ dnDirichlet(rep(alpha_bl,n_branches))
```

```
br_lens := rel_branch_lengths * TL
```

```
Q <- fnJC(4)
```

```
psi := treeAssembly(topology, br_lens)
```

```
seq ~ dnPhyloCTMC(tree=psi,Q=Q,type="DNA")
```

```
seq.clamp(data)
```

# Jukes-Cantor

```
data = readDiscreteCharacterData("myData.nex")
taxa <- data.taxa()
n_taxa <- data.ntaxa()
```

$\alpha$

## ▼ Alternative branch-length priors

Some studies, e.g. [Brown et al. \(2010\)](#), [Rannala et al. \(2012\)](#), have criticized the exponential prior distribution for branch lengths because it induces a gamma-distributed tree-length and the mean of this gamma distribution grows with the number of taxa. As an alternative, we can instead use a specific gamma prior distribution (or any other distribution defined on a positive real variable) for the tree length, and then use a Dirichlet prior distribution to break the tree length into the corresponding branch lengths ([Zhang et al. 2012](#)).

First, specify a prior distribution on the tree length with your desired mean. For example, we use a gamma distribution as our prior on the tree length.

```
TL ~ dnGamma(2,4)
moves.append( mvScale(TL) )
```

Now we create a random variable for the relative branch lengths.

```
rel_branch_lengths ~ dnDirichlet( rep(1.0,num_branches) )
moves.append( mvBetaSimplex(rel_branch_lengths, weight=num_branches) )
moves.append( mvDirichletSimplex(rel_branch_lengths, weight=num_branches/10.0) )
```

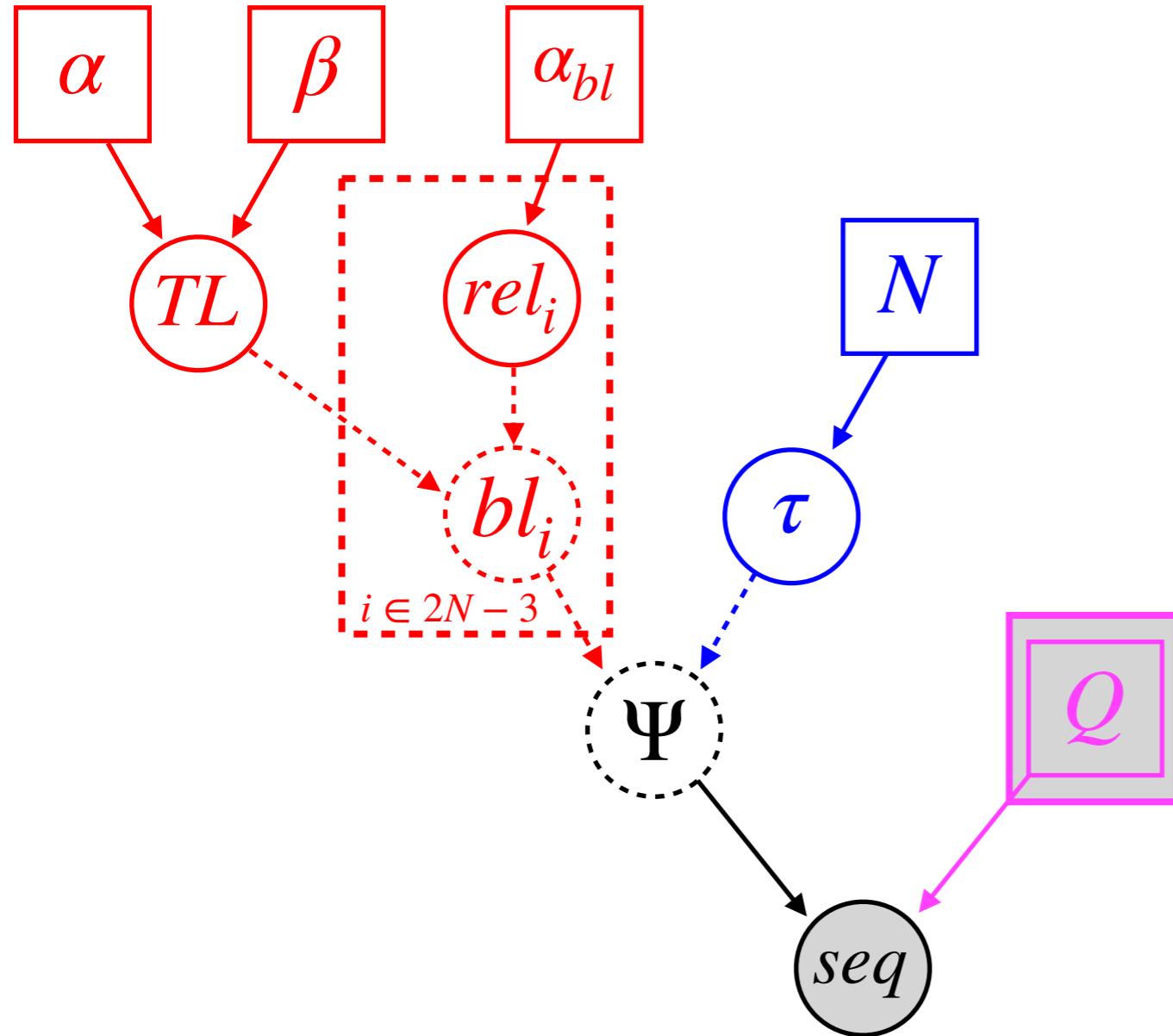
Finally, transform the relative branch lengths into actual branch lengths

```
br_lens := rel_branch_lengths * TL
```

```
seq ~ dnPhyloCTMC(tree=psi,Q=Q,type="DNA")
```

```
seq.clamp(data)
```

# Jukes-Cantor



```
data = readDiscreteCharacterData("myData.nex")
```

```
taxa <- data.taxa()
```

```
n_taxa <- data.ntaxa()
```

```
n_branches <- 2 * n_taxa - 3
```

```
topology ~ dnUniformTopology(taxa)
```

```
alpha <- 2
```

```
beta <- 4
```

```
TL ~ dnGamma(alpha,beta)
```

```
alpha_bl <- 1.0
```

```
rel_branch_lengths ~ dnDirichlet(rep(alpha_bl,n_branches))
```

```
br_lens := rel_branch_lengths * TL
```

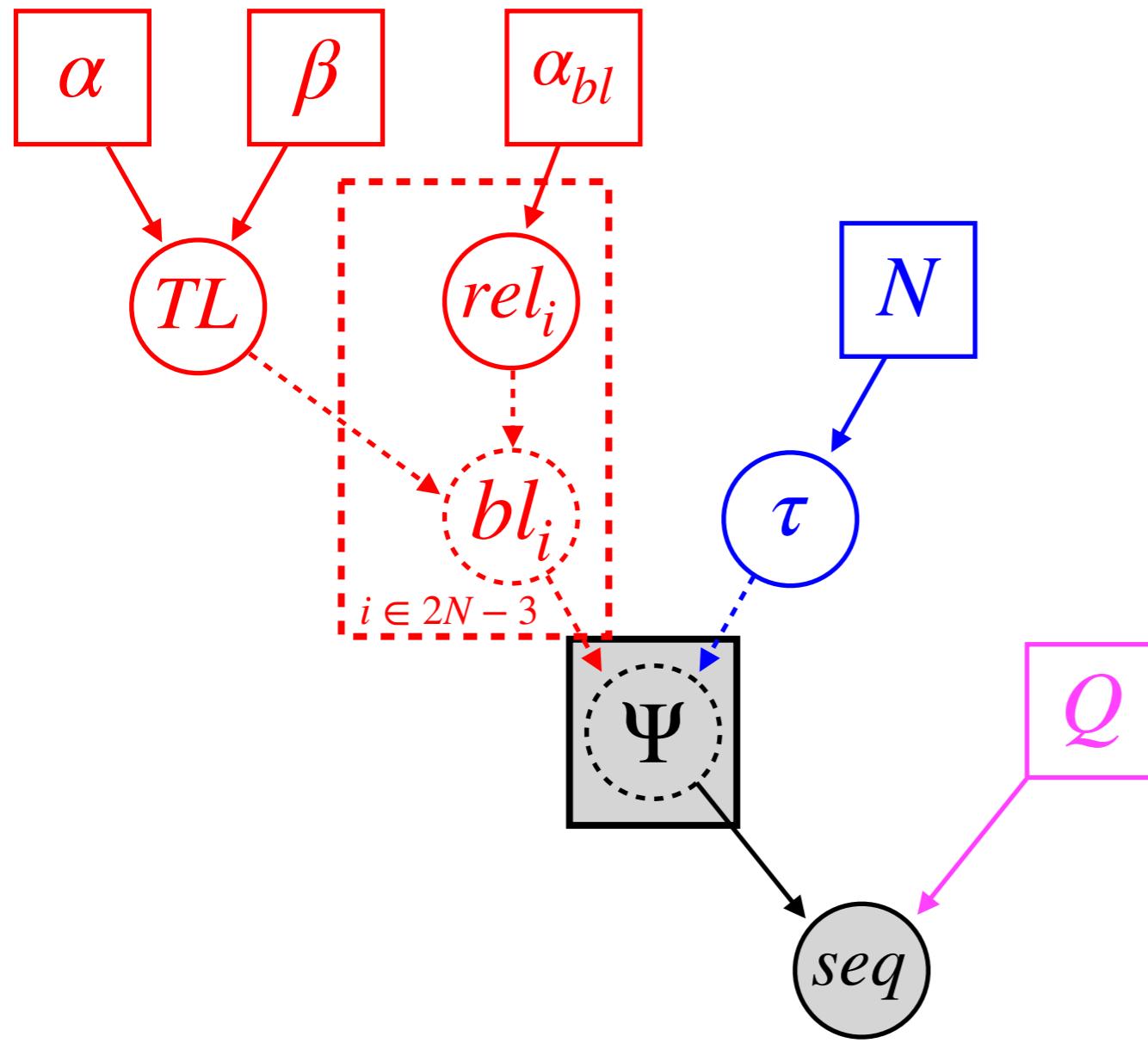
```
Q <- fnJC(4)
```

```
psi := treeAssembly(topology, br_lens)
```

```
seq ~ dnPhyloCTMC(tree=psi,Q=Q,type="DNA")
```

```
seq.clamp(data)
```

# Jukes-Cantor



```

data = readDiscreteCharacterData("myData.nex")
taxa <- data.taxa()
n_taxa <- data.ntaxa()
n_branches <- 2 * n_taxa - 3
  
```

```
topology ~ dnUniformTopology(taxa)
```

```

alpha <- 2
beta <- 4
  
```

```
TL ~ dnGamma(alpha,beta)
```

```
alpha_bl <- 1.0
```

```
rel_branch_lengths ~ dnDirichlet(rep(alpha_bl,n_branches))
```

```
br_lens := rel_branch_lengths * TL
```

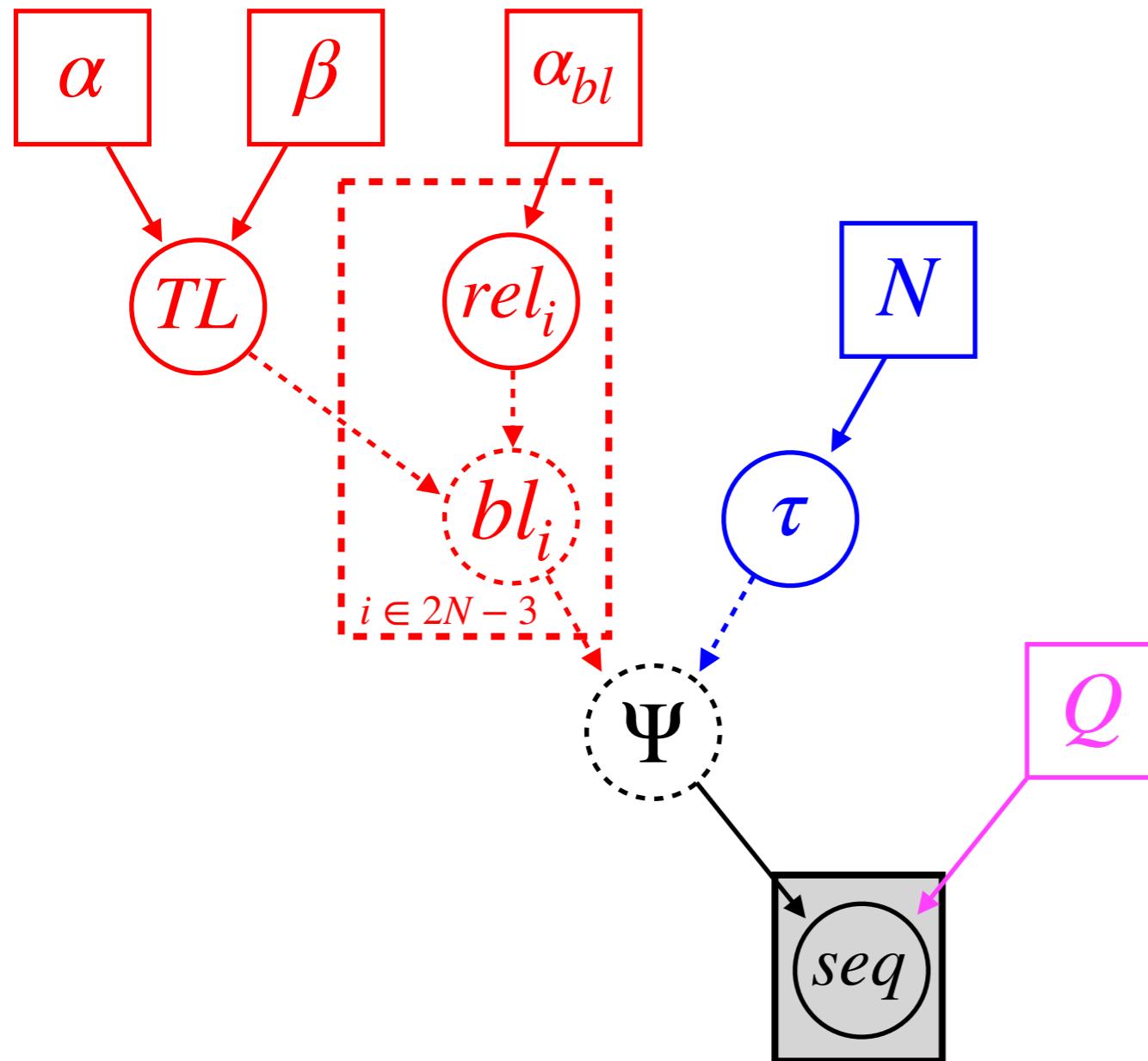
```
Q <- fnJC(4)
```

```
psi := treeAssembly(topology, br_lens)
```

```
seq ~ dnPhyloCTMC(tree=psi,Q=Q,type="DNA")
```

```
seq.clamp(data)
```

# Jukes-Cantor



```

data = readDiscreteCharacterData("myData.nex")
taxa <- data.taxa()
n_taxa <- data.ntaxa()
n_branches <- 2 * n_taxa - 3

topology ~ dnUniformTopology(taxa)

alpha <- 2
beta <- 4

TL ~ dnGamma(alpha,beta)

alpha_bl <- 1.0

rel_branch_lengths ~ dnDirichlet(rep(alpha_bl,n_branches))

br_lens := rel_branch_lengths * TL

Q <- fnJC(4)

psi := treeAssembly(topology, br_lens)

seq ~ dnPhyloCTMC(tree=psi,Q=Q,type="DNA")

seq.clamp(data)
  
```

# Jukes-Cantor

$TL$

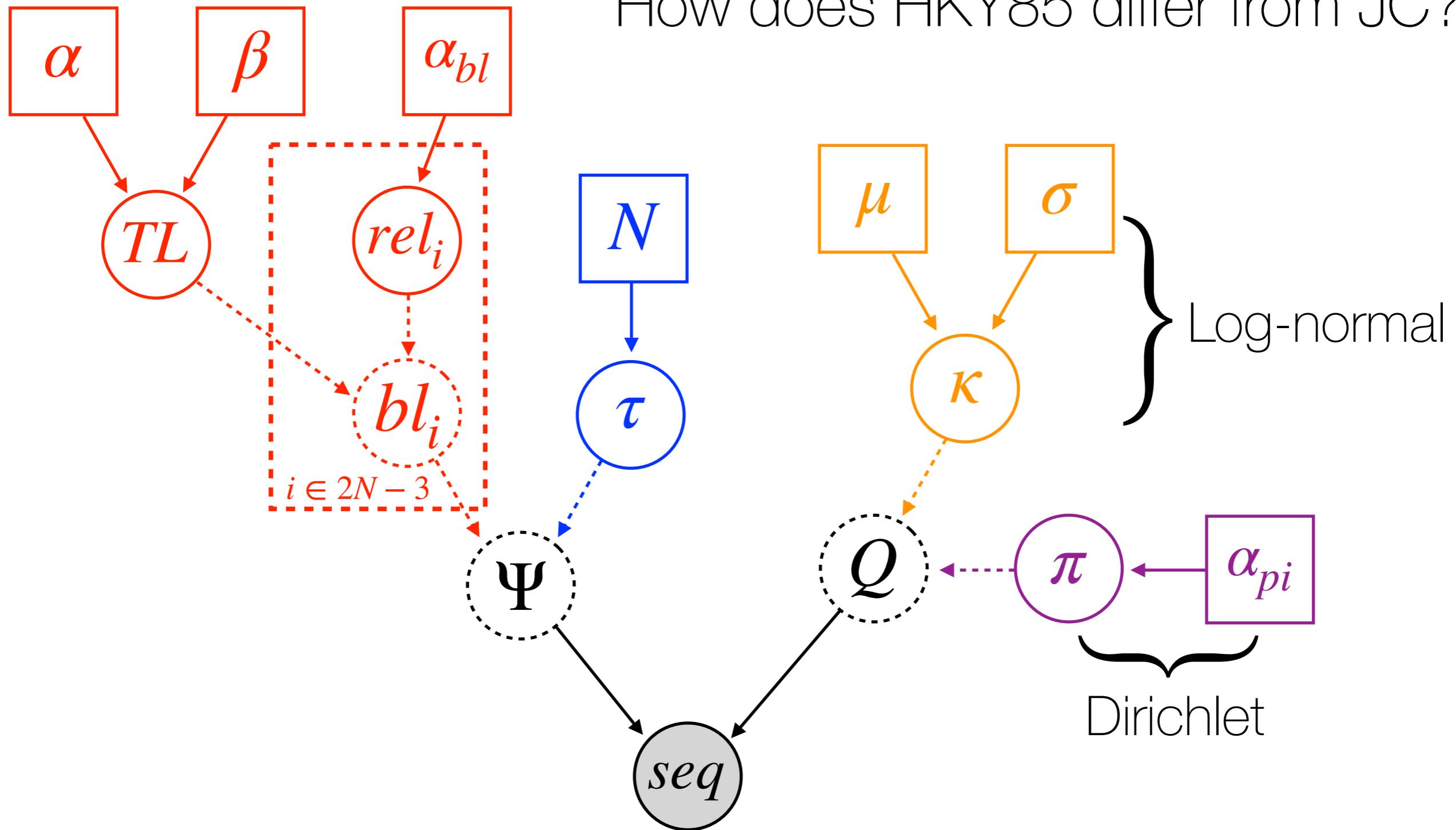
$rel_i$

$\tau$

Inferring these parameters  
under Jukes-Cantor.

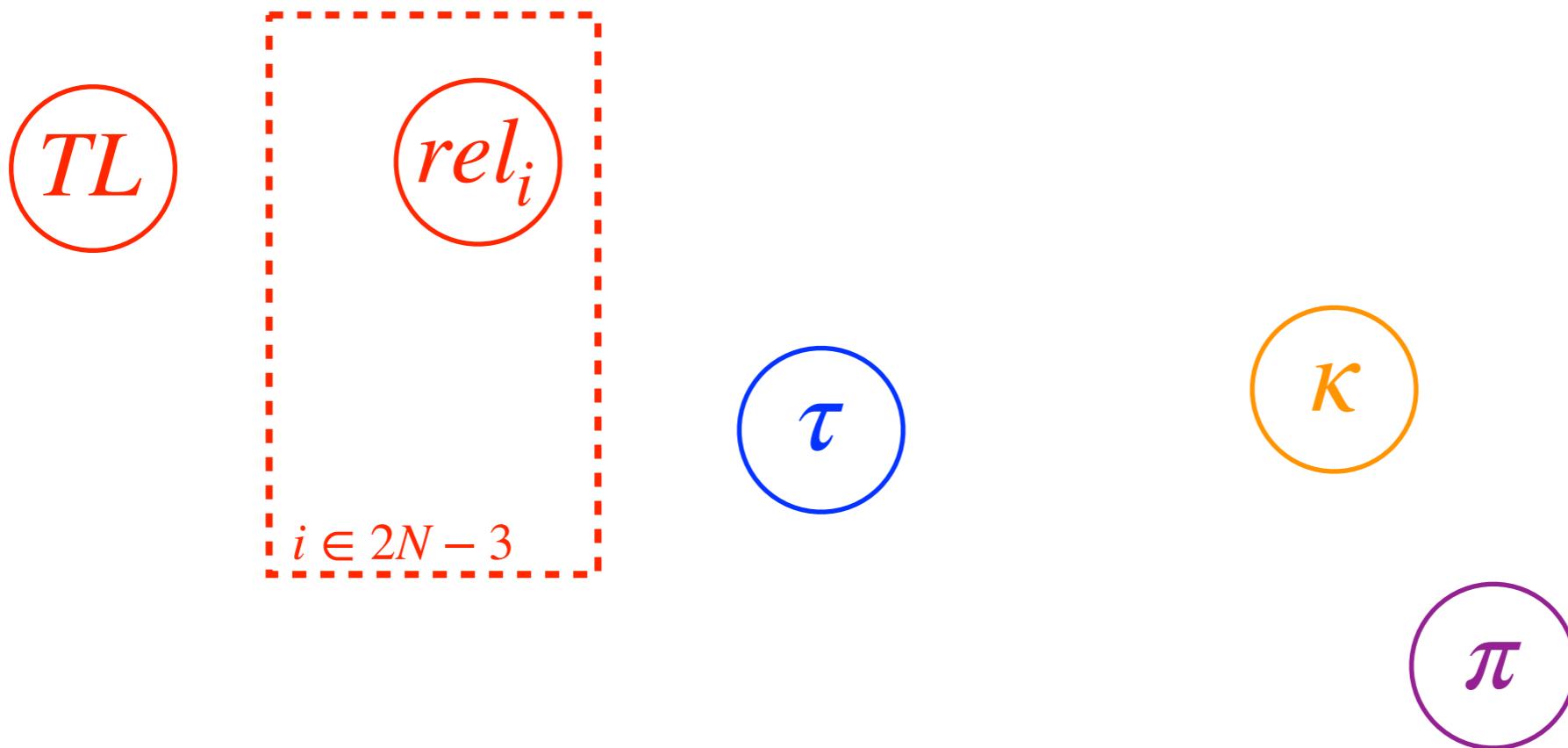
# HKY85

How does HKY85 differ from JC?



# HKY85

How does HKY85 differ from JC?



# HKY85

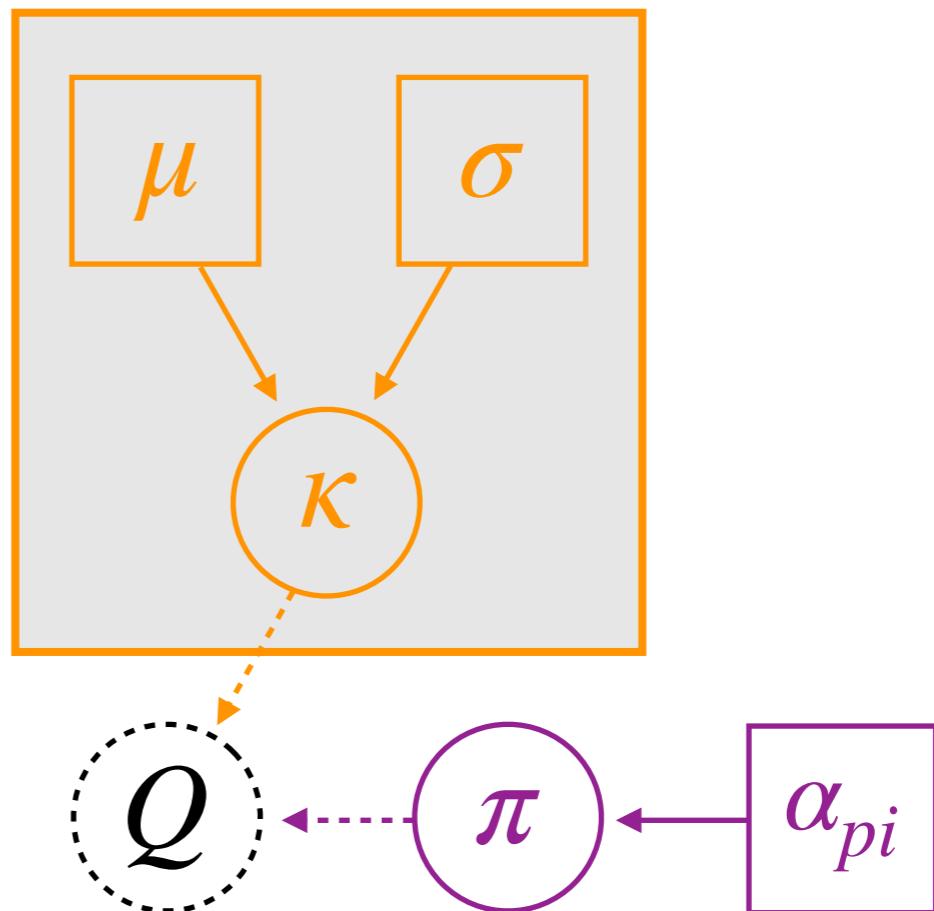
How does HKY85 differ from JC?

## Transition-Transversion Rate Ratio

```
mu <- 0  
sigma <- 1  
kappa ~ dnLognormal(mu,sigma)
```

```
alpha_pi <- 1.0  
pi ~ dnDirichlet( rep(alpha_pi,4) )
```

```
Q := fnHKY(kappa,pi)
```



# HKY85

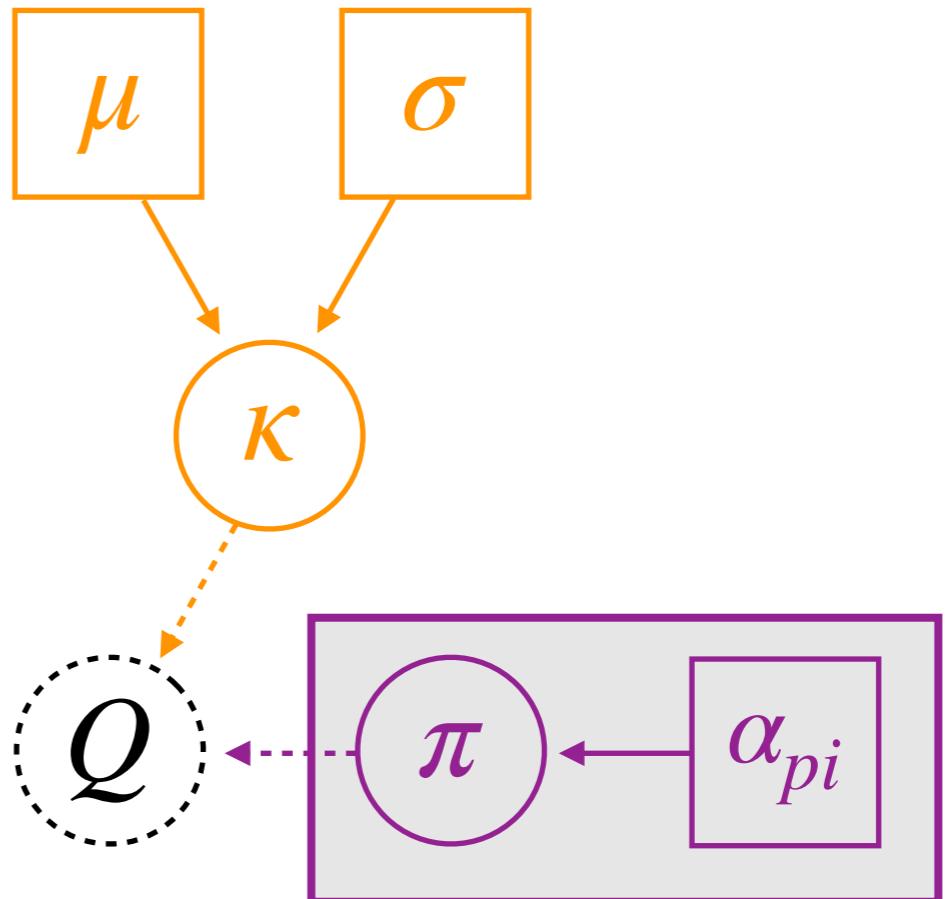
How does HKY85 differ from JC?

```
mu <- 0  
sigma <- 1  
kappa ~ dnLognormal(mu,sigma)
```

```
alpha_pi <- 1.0  
pi ~ dnDirichlet( rep(alpha_pi,4) )
```

## Variable Base Frequencies

```
Q := fnHKY(kappa,pi)
```



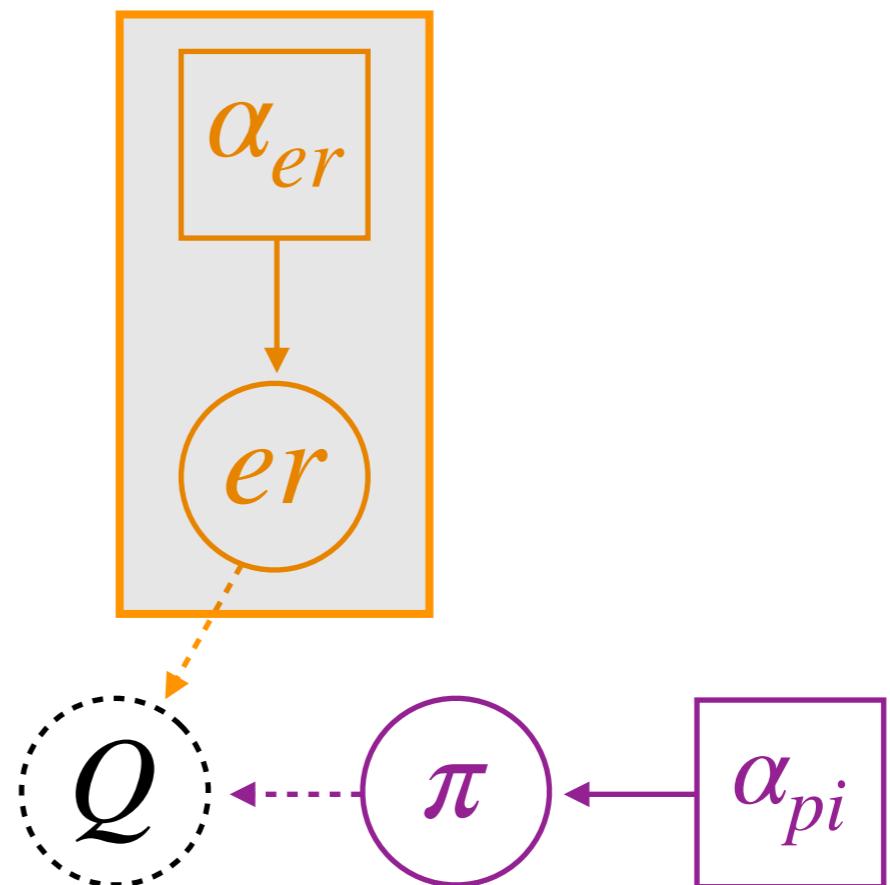
# General Time Reversible (GTR)

## Variable Exchangeabilities

```
alpha_er <- 1.0  
er ~ dnDirichlet( rep(alpha_er,6) )
```

```
alpha_pi <- 1.0  
pi ~ dnDirichlet( rep(alpha_pi,4) )
```

```
Q := fnGTR(er,pi)
```



# Things to Remember

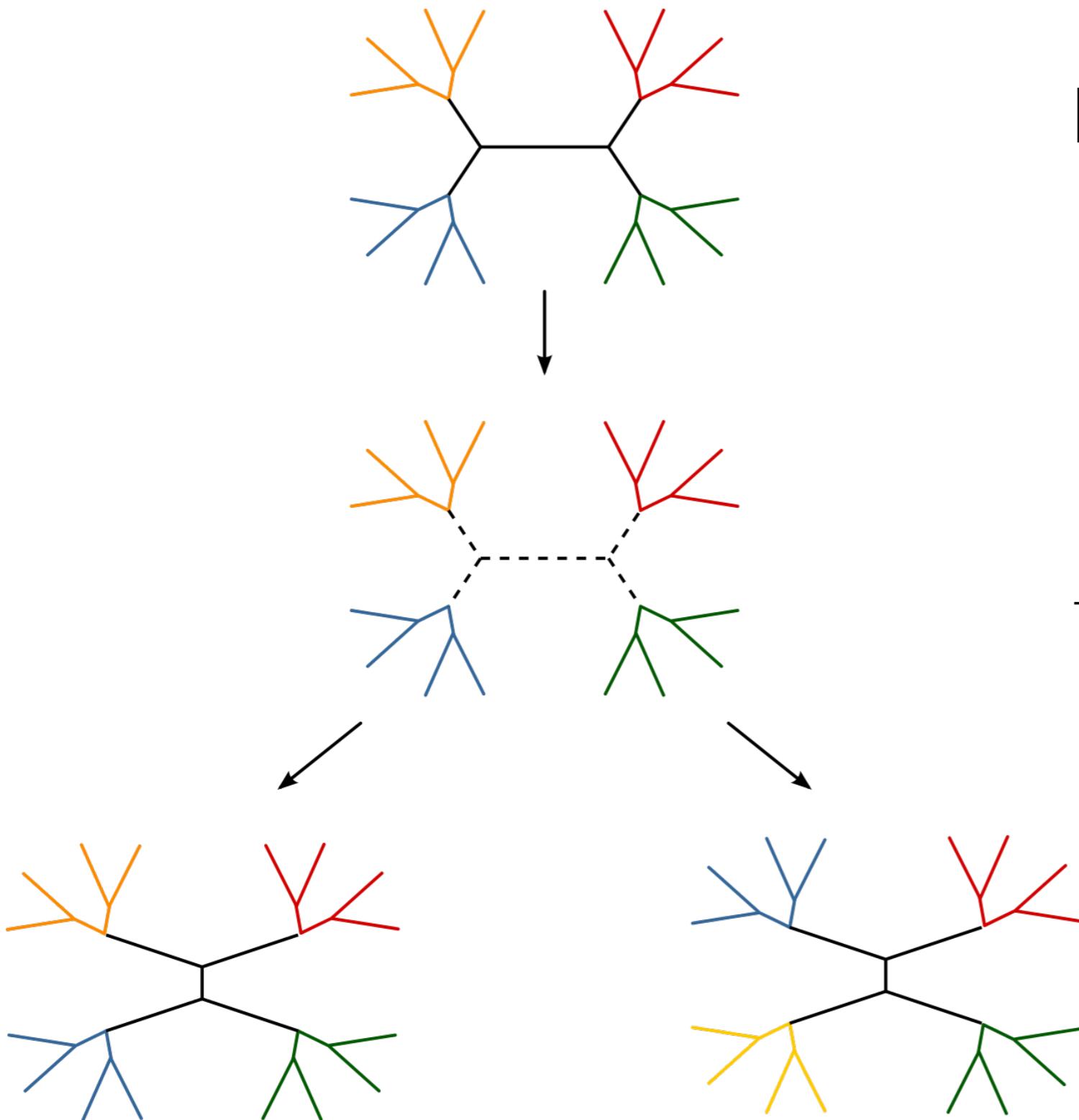
As we add more stochastic nodes, **remember to assign moves** to all of them!

Vectors modeled with a Dirichlet should have a joint proposal mechanism, since they need to sum to 1. Here are a couple:

mvBetaSimplex  
mvDirichletSimplex

You'll often want to run analyses on the same dataset with multiple models. Start by saving your simplest model into a text file (.rev). Then, copy and paste the code into a file for the next model and adjust as necessary. Keep going as you work your way to more complex models.

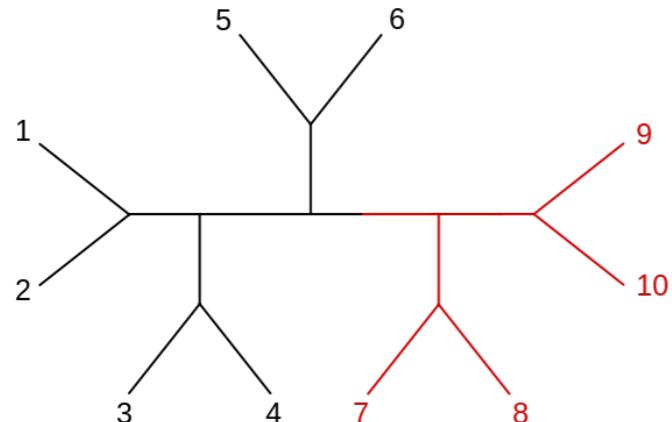
# Topology Moves



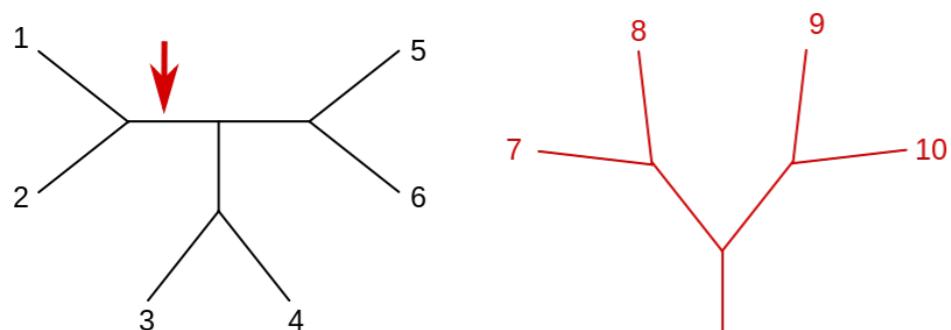
Nearest-Neighbor  
Interchange  
(NNI)

The most minor  
topological move.

# Topology Moves

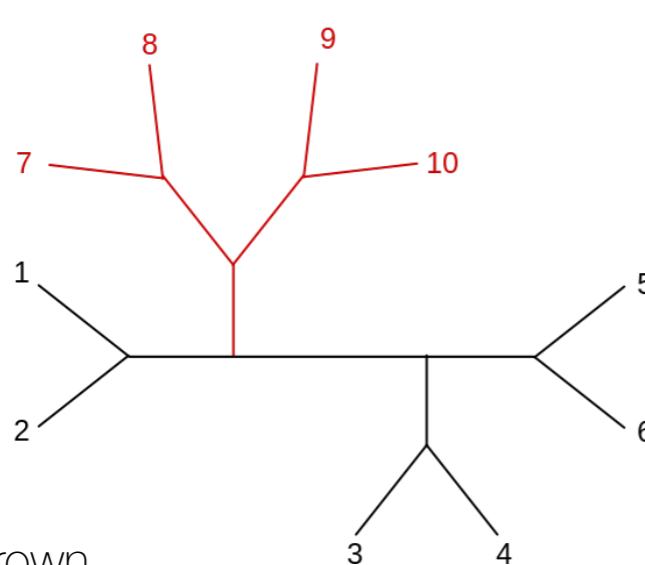


Subtree Prune and Regraft  
(SPR)



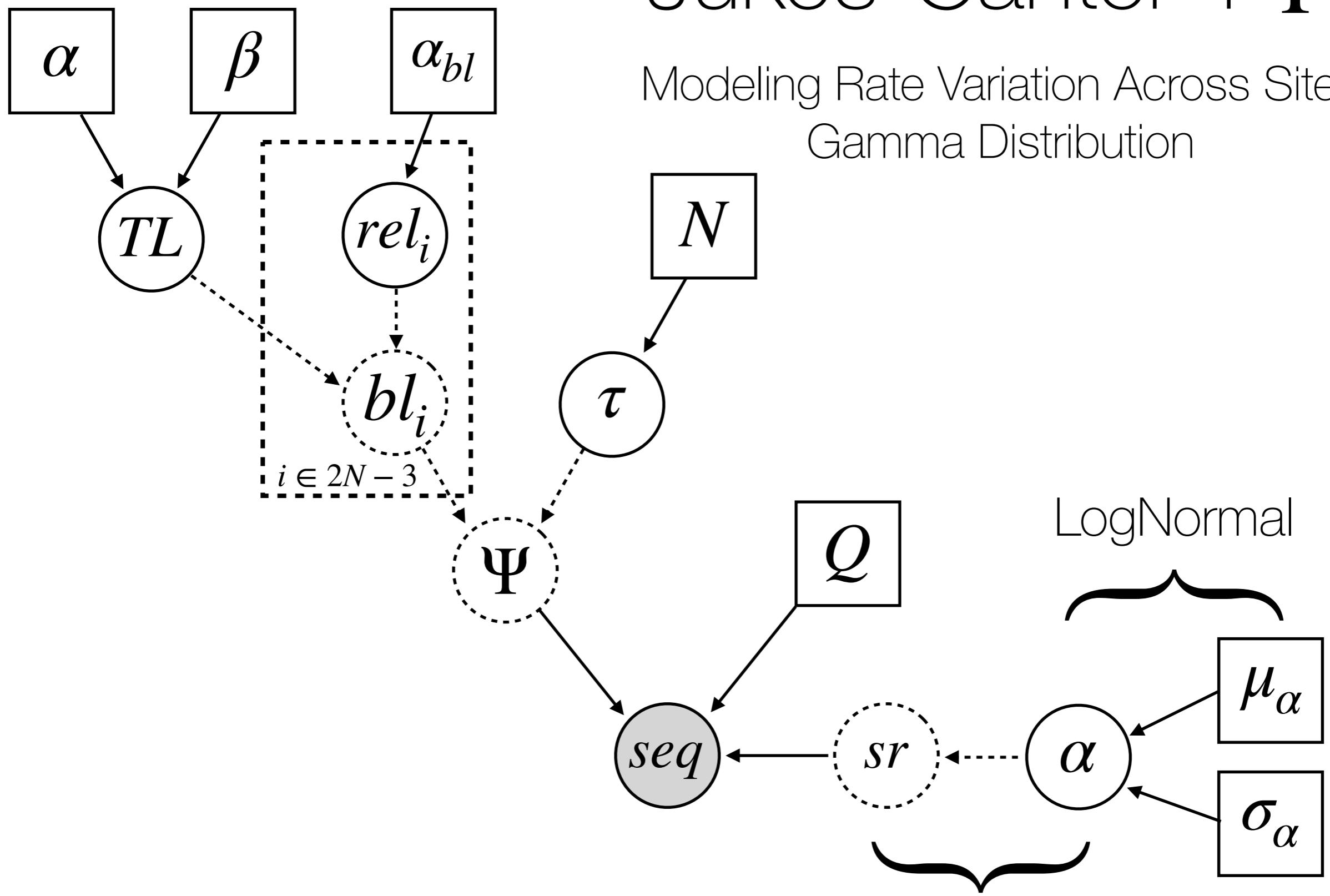
Intermediate topological move.

Often a good balance between exploring new tree space without disrupting things too much.



# Jukes-Cantor + $\Gamma$

Modeling Rate Variation Across Sites  
Gamma Distribution



Discretized Gamma

# Jukes-Cantor + $\Gamma$

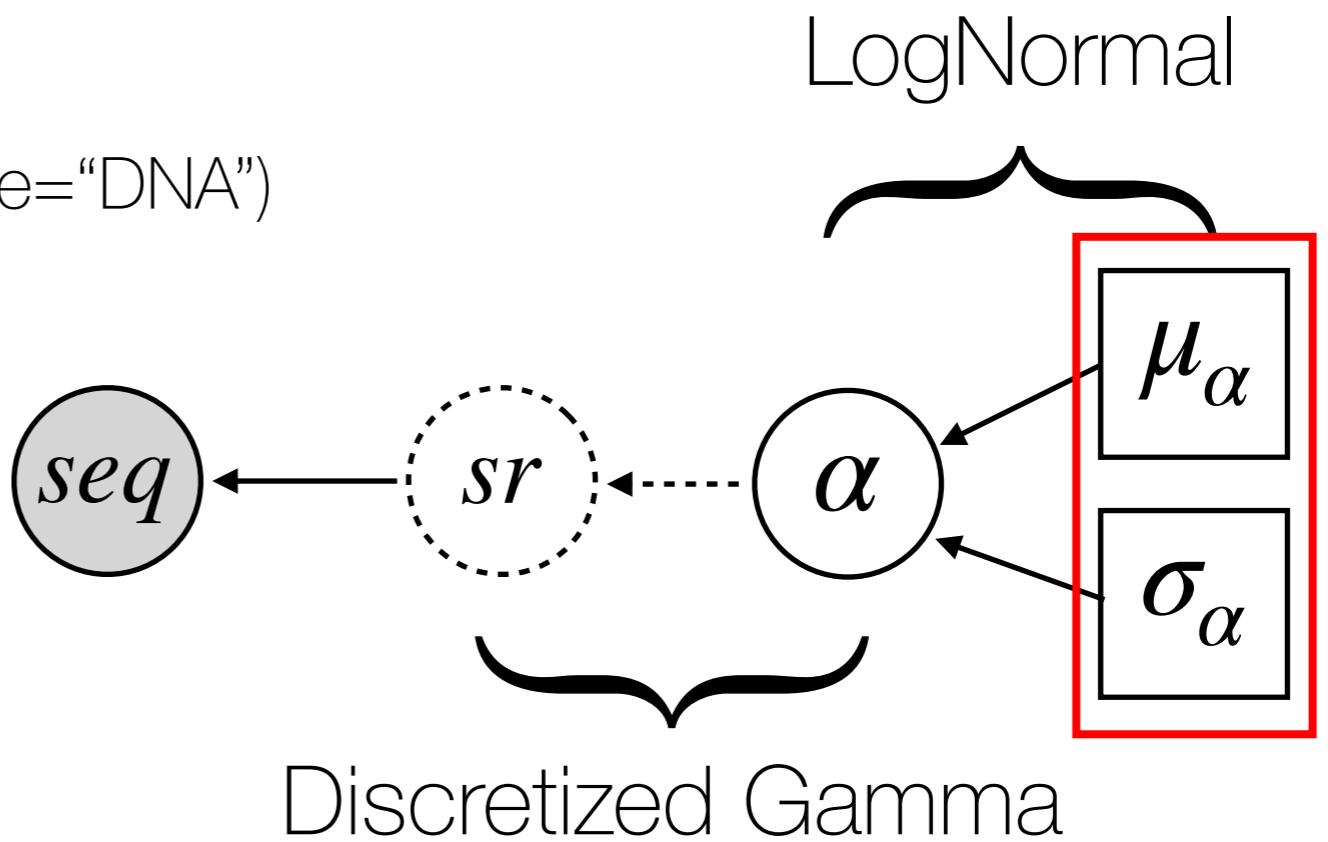
Modeling Rate Variation Across Sites  
Gamma Distribution

```
mu_a <- ln(5.0)  
sigma_a <- 0.587405
```

```
alpha ~ dnLognormal( mu_a, sigma_a )
```

```
sr := fnDiscretizeGamma( alpha, alpha, 4, false )
```

```
seq ~ dnPhyloCTMC(tree=psi,Q=Q,  
siteRates=sr,type="DNA")
```



# Jukes-Cantor + $\Gamma$

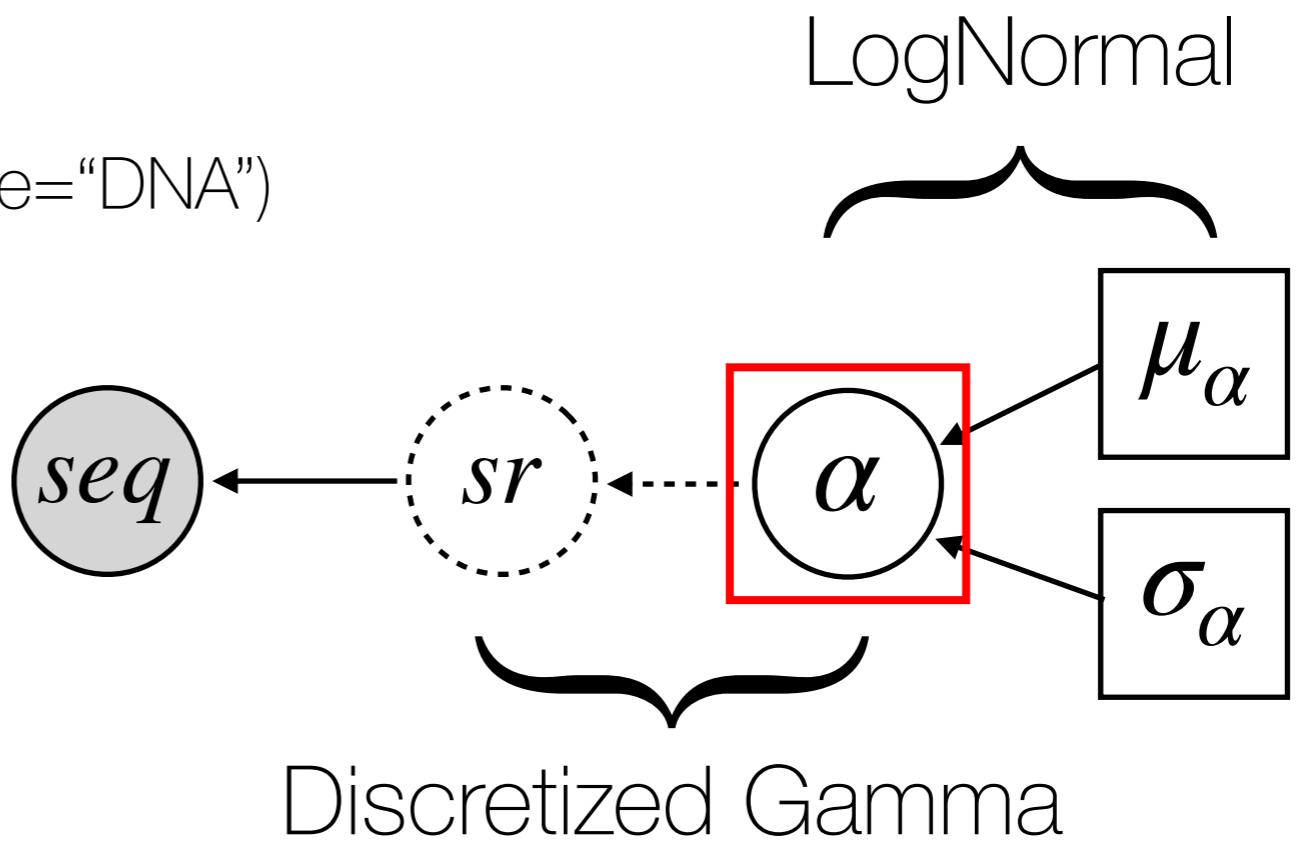
Modeling Rate Variation Across Sites  
Gamma Distribution

```
mu_a <- ln(5.0)
sigma_a <- 0.587405
```

```
alpha ~ dnLognormal( mu_a, sigma_a )
```

```
sr := fnDiscretizeGamma( alpha, alpha, 4, false )
```

```
seq ~ dnPhyloCTMC(tree=psi,Q=Q,
                     siteRates=sr,type="DNA")
```



# Jukes-Cantor + $\Gamma$

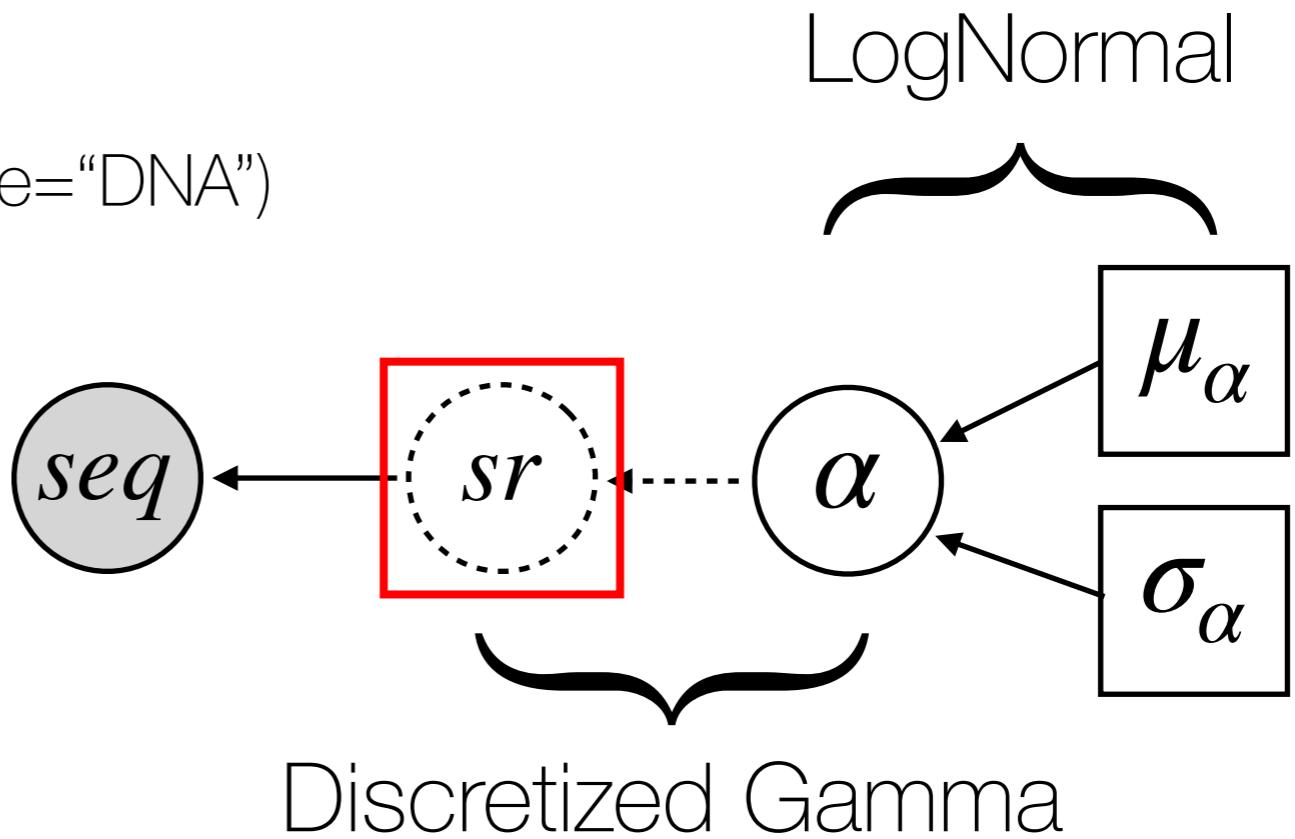
Modeling Rate Variation Across Sites  
Gamma Distribution

```
mu_a <- ln(5.0)
sigma_a <- 0.587405
```

```
alpha ~ dnLognormal( mu_a, sigma_a )
```

```
sr := fnDiscretizeGamma( alpha, alpha, 4, false )
```

```
seq ~ dnPhyloCTMC(tree=psi,Q=Q,
                     siteRates=sr,type="DNA")
```



# Jukes-Cantor + $\Gamma$

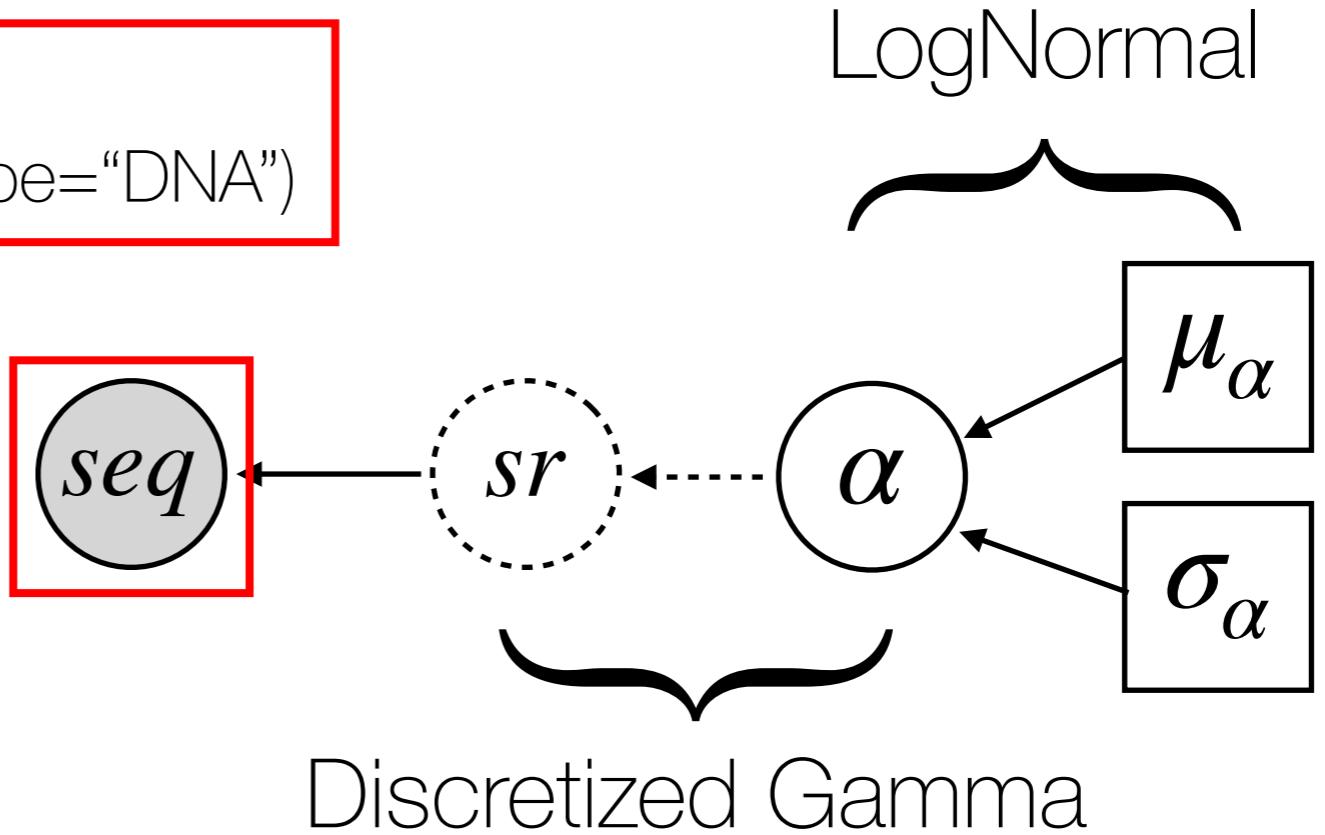
Modeling Rate Variation Across Sites  
Gamma Distribution

```
mu_a <- ln(5.0)
sigma_a <- 0.587405
```

```
alpha ~ dnLognormal( mu_a, sigma_a )
```

```
sr := fnDiscretizeGamma( alpha, alpha, 4, false )
```

```
seq ~ dnPhyloCTMC(tree=psi,Q=Q,
                     siteRates=sr,type="DNA")
```



# Jukes-Cantor + $\Gamma$

Modeling Rate Variation Across Sites  
Gamma Distribution

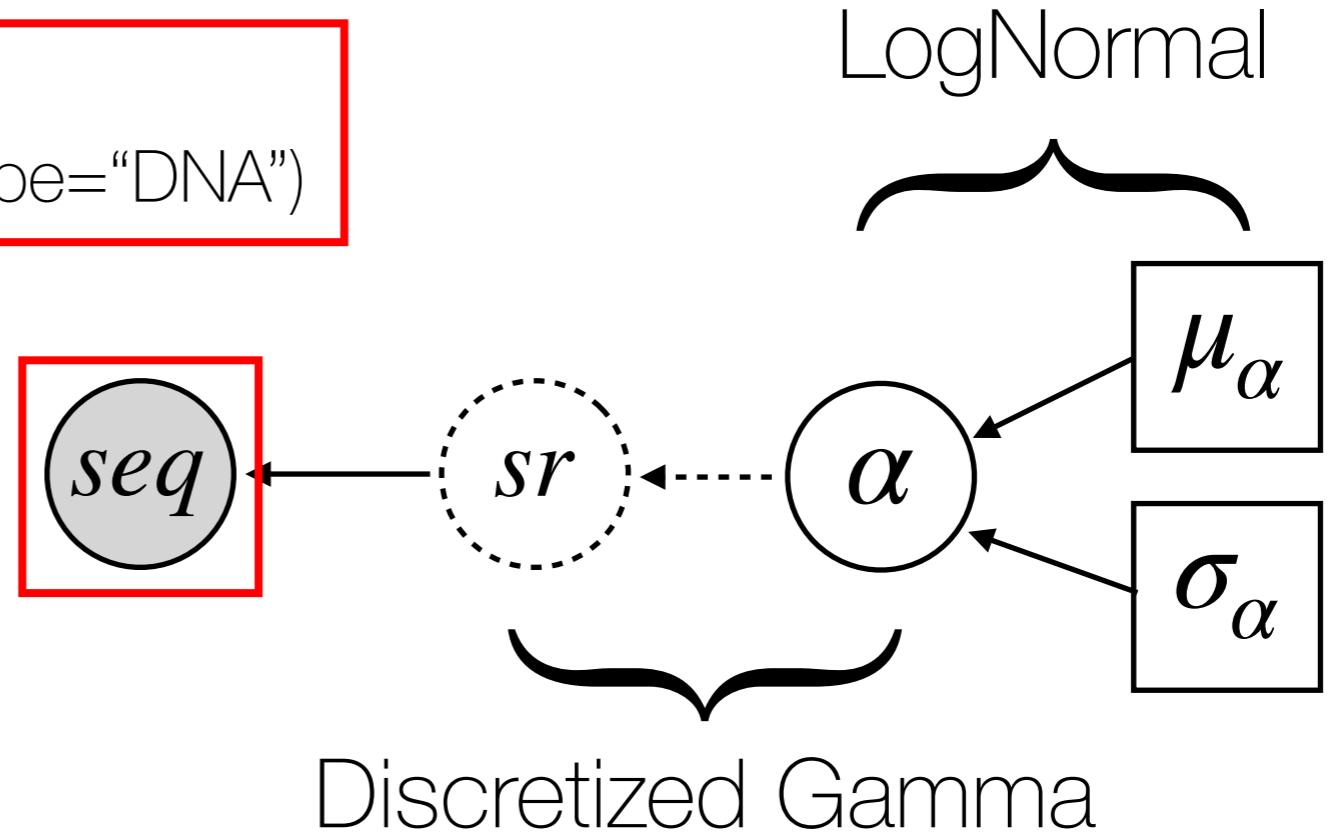
```
mu_a <- ln(5.0)
sigma_a <- 0.587405
```

```
alpha ~ dnLognormal( mu_a, sigma_a )
```

```
sr := fnDiscretizeGamma( alpha, alpha, 4, false )
```

```
seq ~ dnPhyloCTMC(tree=psi,Q=Q,
                     siteRates=sr,type="DNA")
```

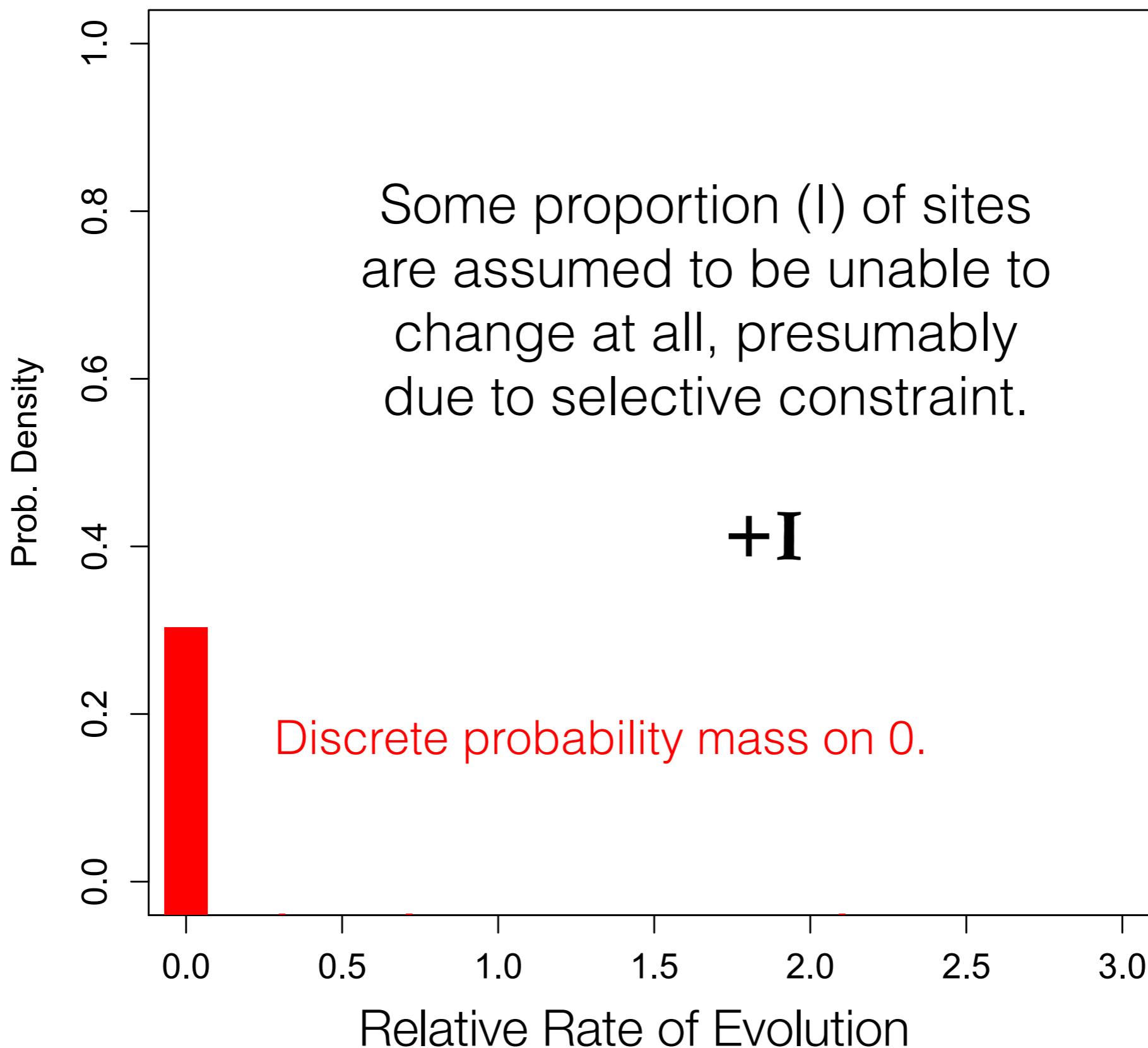
Can add  $+ \Gamma$  to any  
“standard” model of  
sequence evolution.

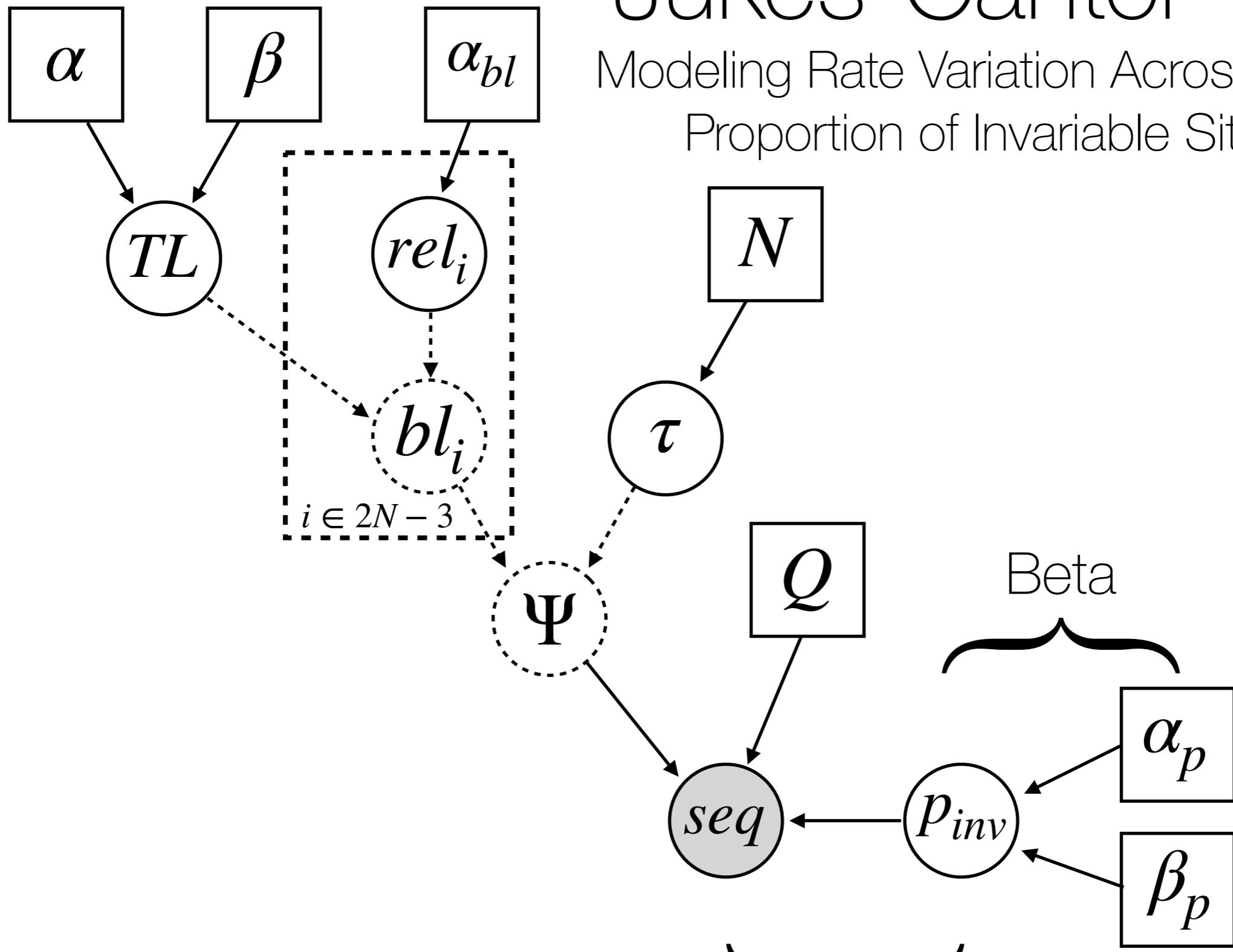


Discretized Gamma

# Modeling Rate Variation Across Sites

## Proportion of Invariable Sites





# Jukes-Cantor + I

Modeling Rate Variation Across Sites  
Proportion of Invariable Sites

Mixture Distribution

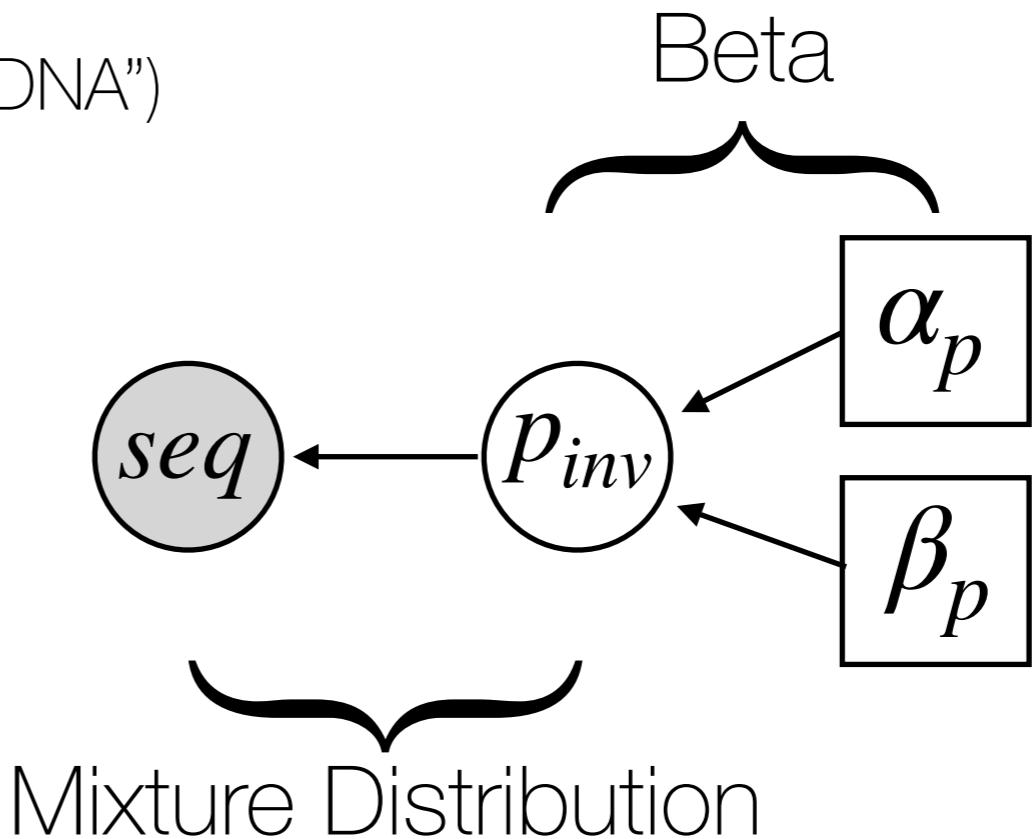
# Jukes-Cantor + I

Modeling Rate Variation Across Sites  
Proportion of Invariable Sites

```
alpha_p <- 1.0  
beta_p <- 1.0
```

```
p_inv ~ dnBeta( alpha_p, beta_p )
```

```
seq ~ dnPhyloCTMC(tree=psi, Q=Q,  
plnv=p_inv, type="DNA")
```



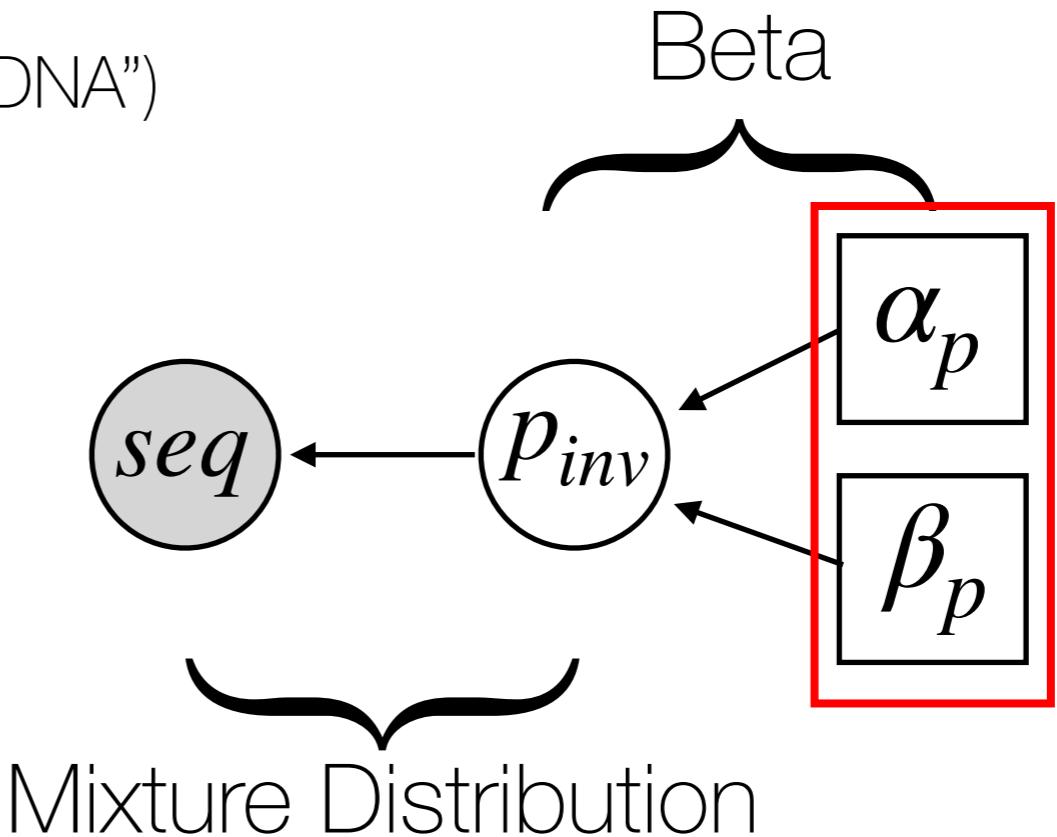
# Jukes-Cantor + I

Modeling Rate Variation Across Sites  
Proportion of Invariable Sites

```
alpha_p <- 1.0  
beta_p <- 1.0
```

```
p_inv ~ dnBeta( alpha_p, beta_p )
```

```
seq ~ dnPhyloCTMC(tree=psi, Q=Q,  
plnv=p_inv, type="DNA")
```



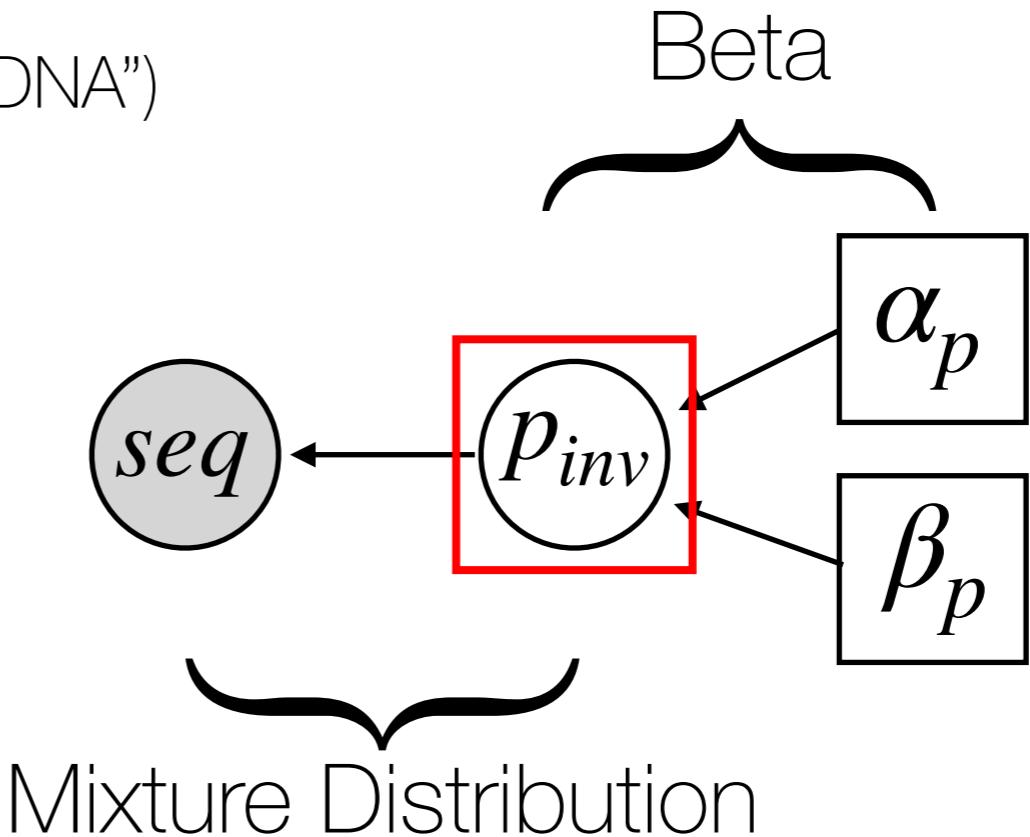
# Jukes-Cantor + I

Modeling Rate Variation Across Sites  
Proportion of Invariable Sites

```
alpha_p <- 1.0  
beta_p <- 1.0
```

```
p_inv ~ dnBeta( alpha_p, beta_p )
```

```
seq ~ dnPhyloCTMC(tree=psi, Q=Q,  
plnv=p_inv, type="DNA")
```



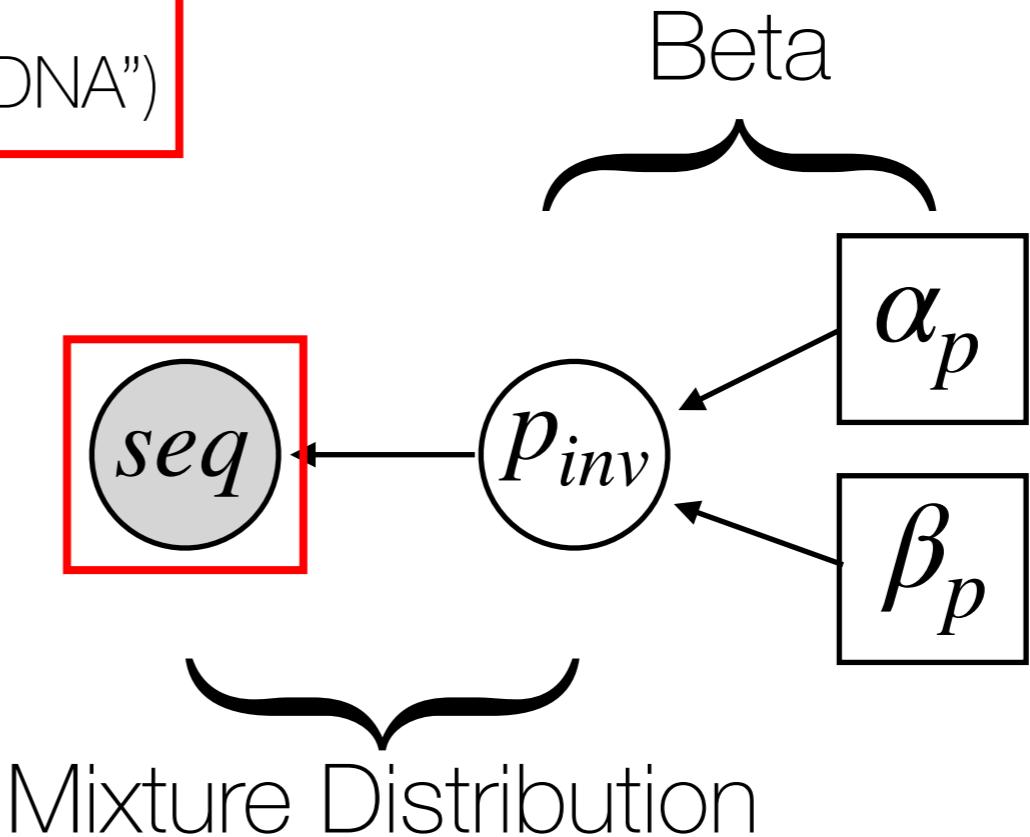
# Jukes-Cantor + I

Modeling Rate Variation Across Sites  
Proportion of Invariable Sites

```
alpha_p <- 1.0  
beta_p <- 1.0
```

```
p_inv ~ dnBeta( alpha_p, beta_p )
```

```
seq ~ dnPhyloCTMC(tree=psi, Q=Q,  
plnv=p_inv, type="DNA")
```



# Jukes-Cantor + I

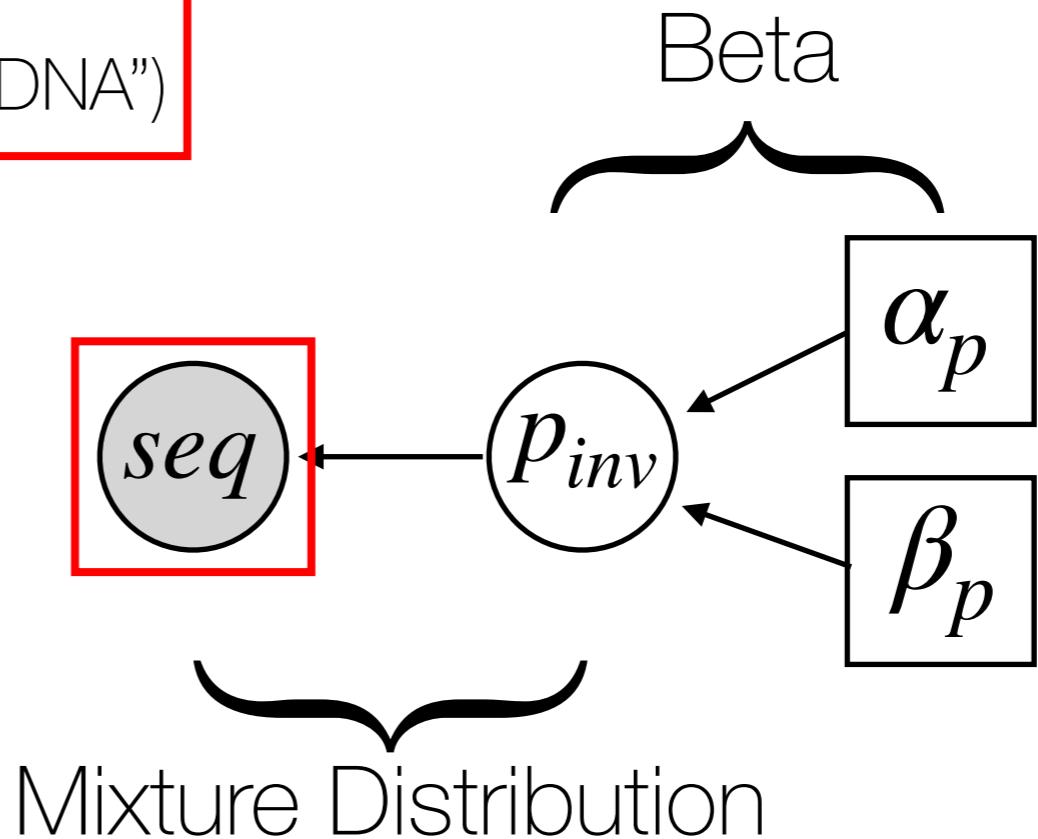
Modeling Rate Variation Across Sites  
Proportion of Invariable Sites

```
alpha_p <- 1.0  
beta_p <- 1.0
```

```
p_inv ~ dnBeta( alpha_p, beta_p )
```

```
seq ~ dnPhyloCTMC(tree=psi, Q=Q,  
plnv=p_inv, type="DNA")
```

Can also add + I to  
any “standard” model  
of sequence evolution.



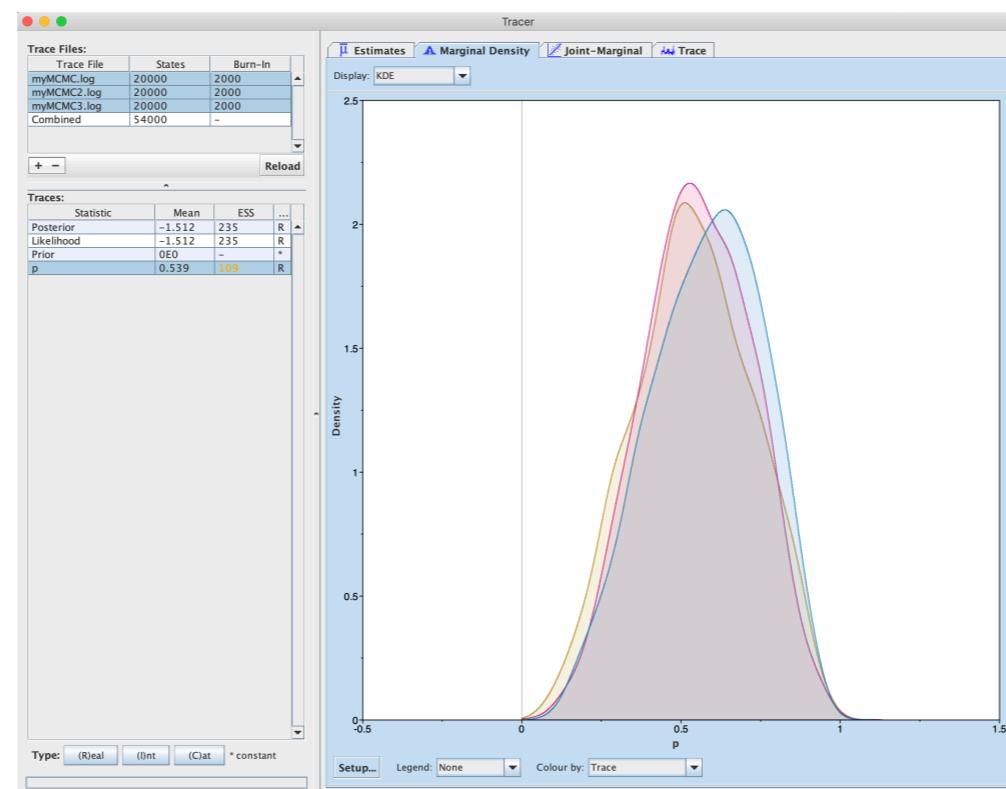
# Summarizing Phylogenetic MCMC Output

For scalar (numerical) parameters, we often report:

Mean

Median

95% Highest Posterior Density Interval



# Summarizing Phylogenetic MCMC Output

For trees, we often report:

Maximum A Posteriori (MAP) Tree  
("Best" Tree)

```
mapTree(run1Trees)
```

# Summarizing Phylogenetic MCMC Output

For trees, we often report:

Maximum A Posteriori (MAP) Tree  
("Best" Tree)

Majority-Rule Consensus Tree  
(Formed from all bipartitions with posterior > 0.5)

```
consensusTree(run1Trees,0.5)
```

# Summarizing Phylogenetic MCMC Output

For trees, we often report:

Maximum A Posteriori (MAP) Tree  
("Best" Tree)

Majority-Rule Consensus Tree  
(Formed from all bipartitions with posterior > 0.5)

Greedy Consensus Tree  
(Formed by ranking bipartitions and adding them until tree is fully resolved)

```
consensusTree(run1Trees,0.0)
```

# Summarizing Phylogenetic MCMC Output

For trees, we often report:

Maximum A Posteriori (MAP) Tree  
("Best" Tree)

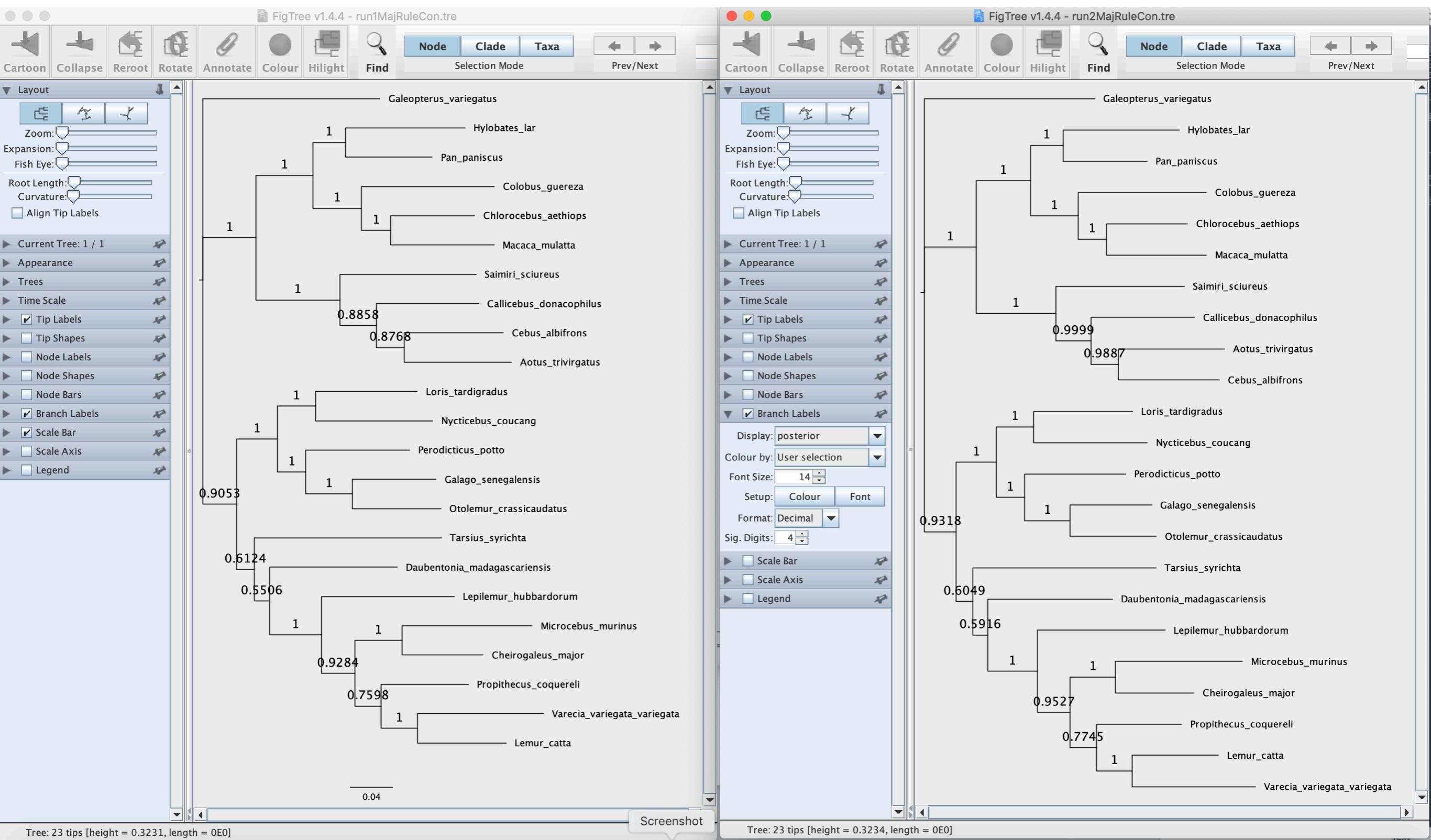
Majority-Rule Consensus Tree  
(Formed from all bipartitions with posterior > 0.5)

Greedy Consensus Tree  
(Formed by ranking bipartitions and adding them until tree is fully resolved)

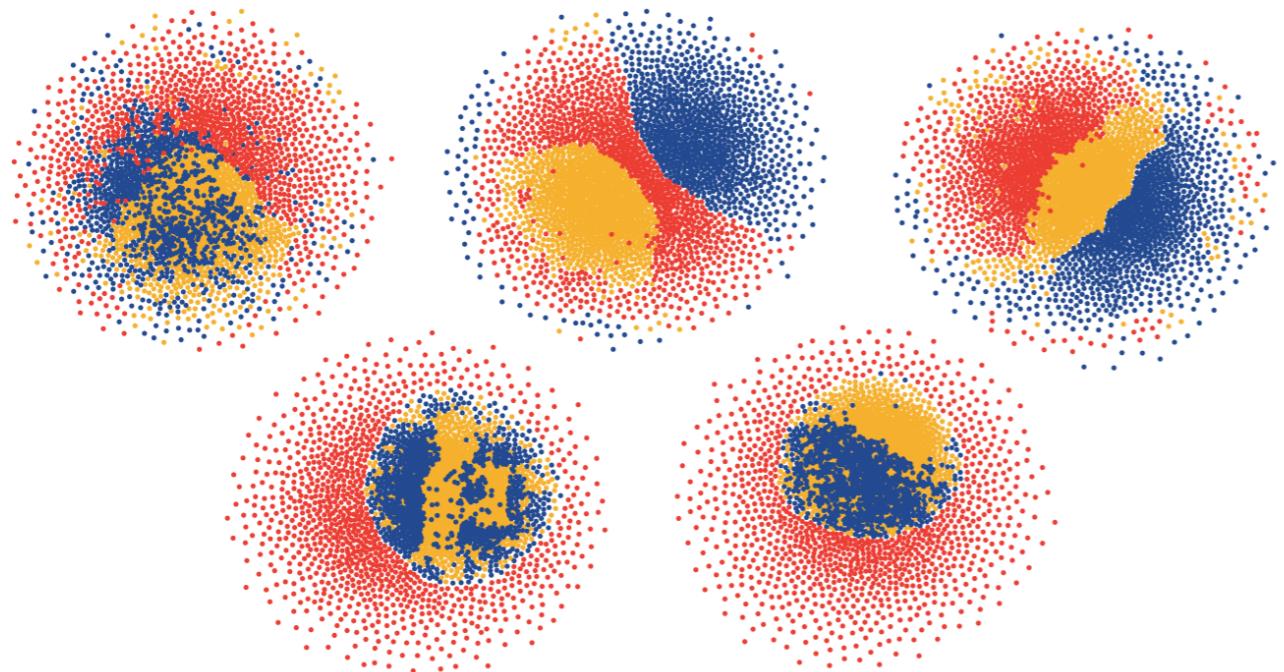
Maximum Clade Credibility Tree  
(Sampled tree whose product of bipartition probabilities is greatest)

```
mccTree(run1Trees)
```

# Assessing Topological Convergence



# Assessing Topological Convergence

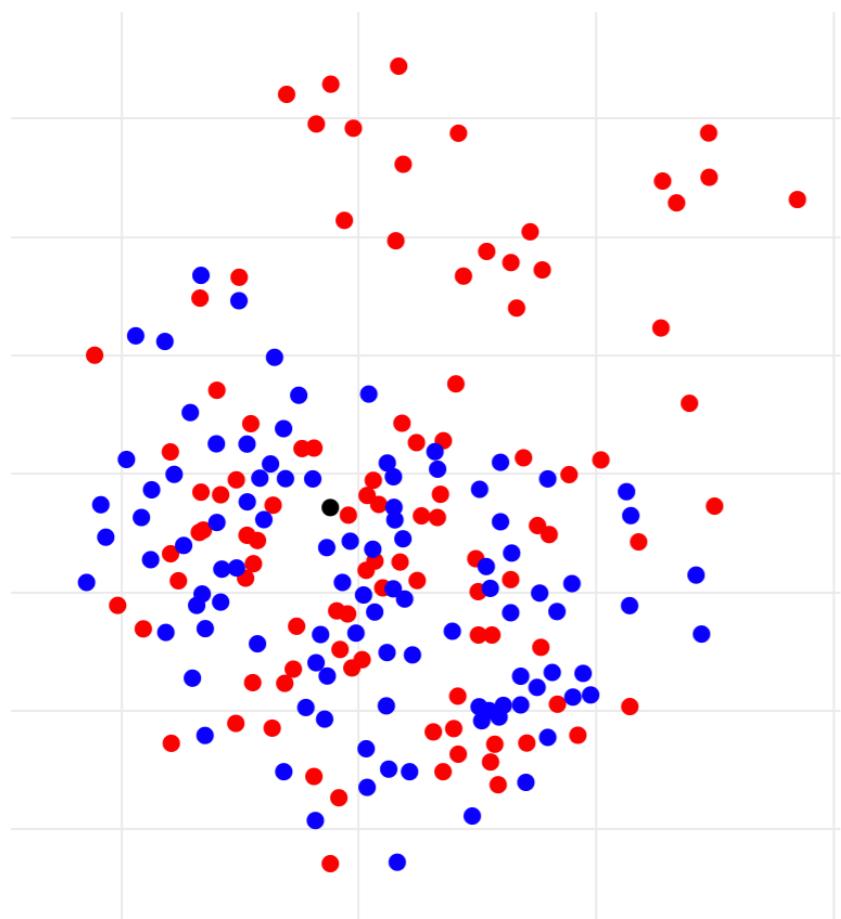


Visualizations of  
Tree Space

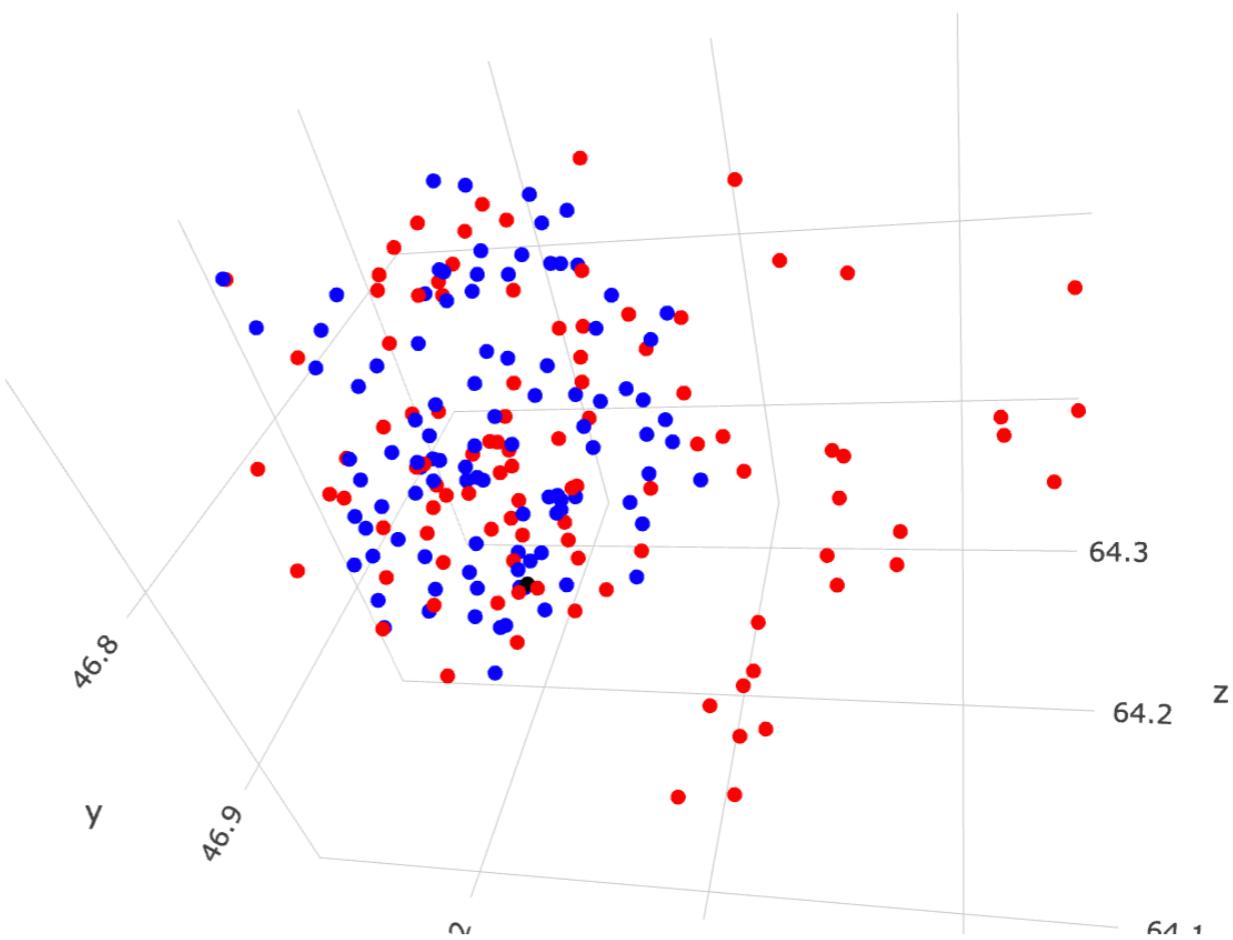
```
run1Trees = readTrees("run1.trees")
run2Trees = readTrees("run2.trees")
maxdiff([run1Trees,run2Trees])
```

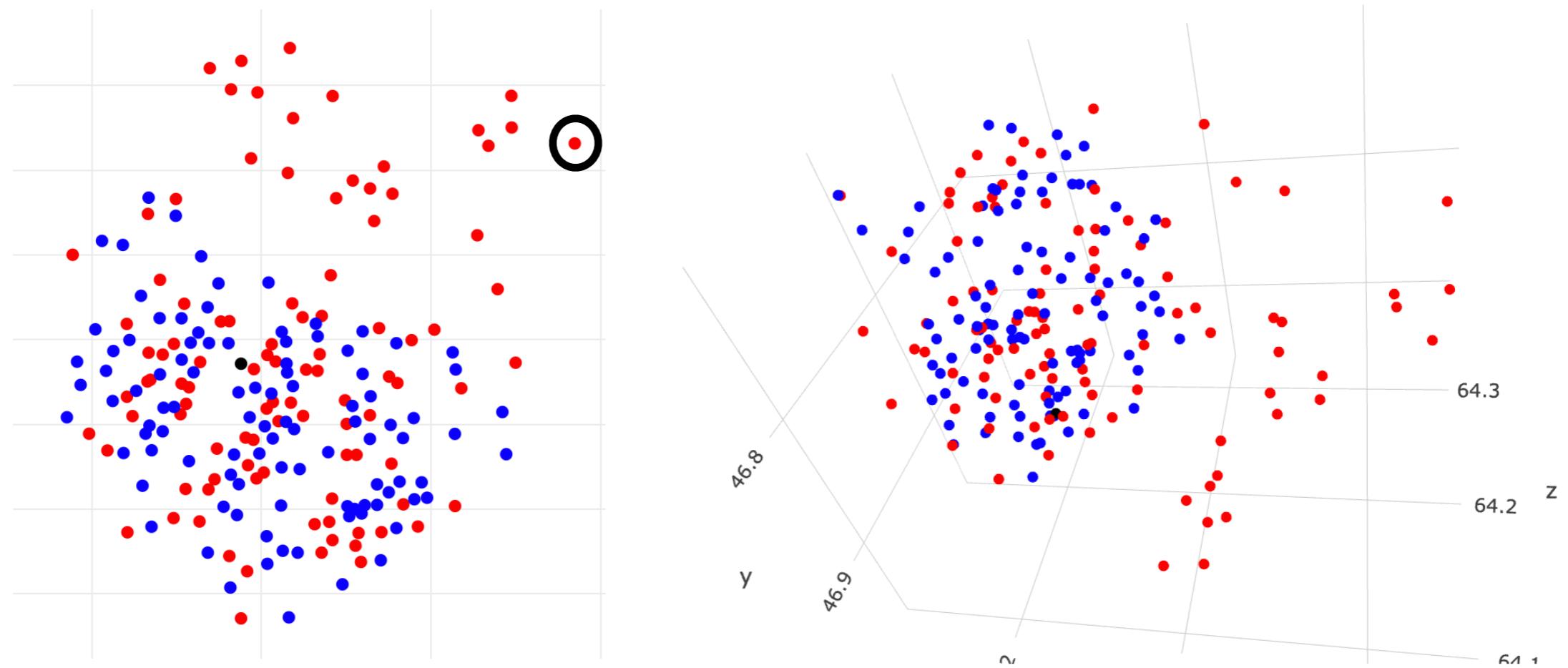
Finds the biggest difference in estimated marginal posterior probabilities for all bipartitions across different replicates.

2D

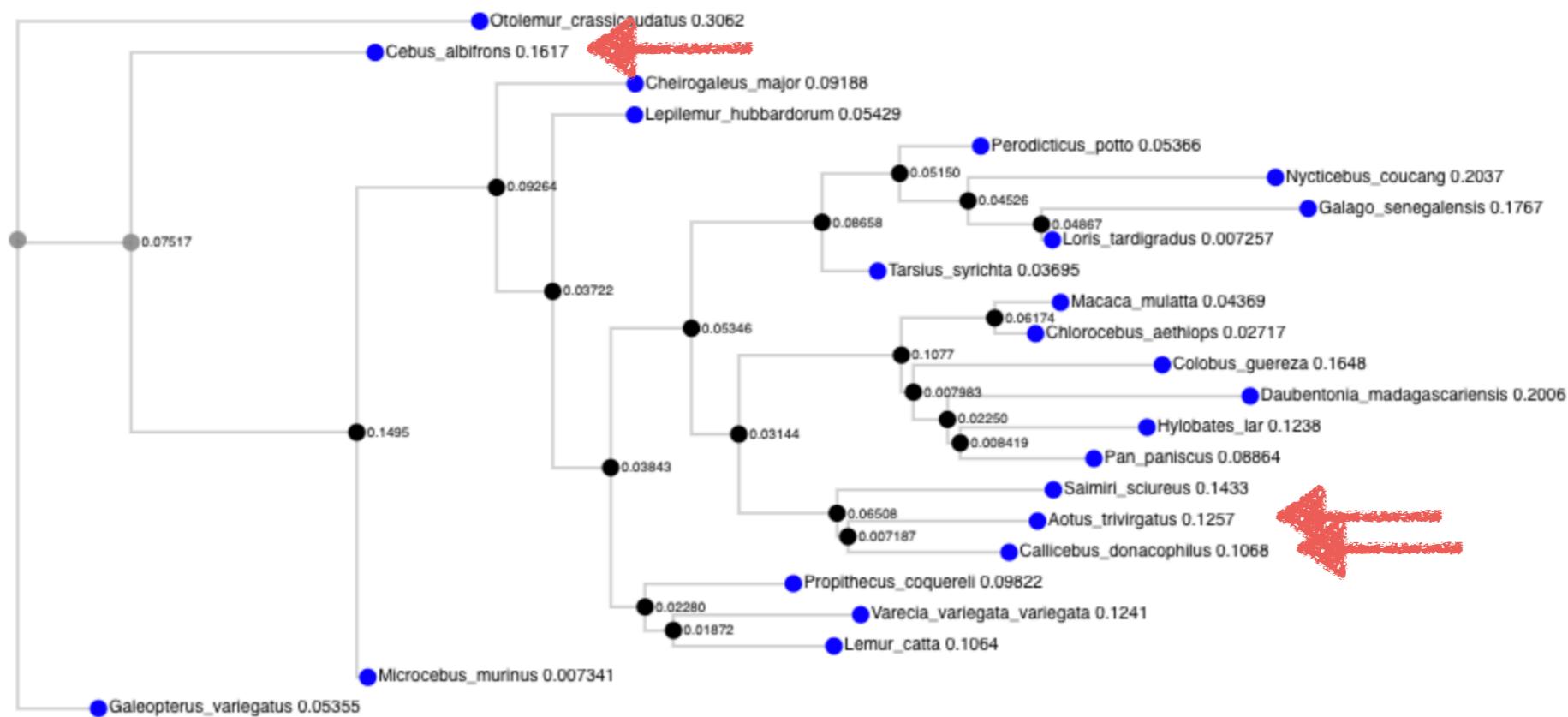


3D





Tree 10



# RevBayes Tutorial Setup

After logging on to your VM...

```
cd ~/moledata/revbayes/genetree  
rb
```

# RevBayes CTMC Tutorial

<https://revbayes.github.io/tutorials/ctmc/>

I strongly encourage you to try building at least one Rev script yourself for this tutorial, rather than simply running those that are already available.

Also note that in the scripts that are already written, the default number of replicate runs is 1. You should set nruns to be at least 2, so that you can assess convergence.

# RevBayes CTMC Tutorial

<https://revbayes.github.io/tutorials/ctmc/>

Look at Table 1 in the tutorial above.

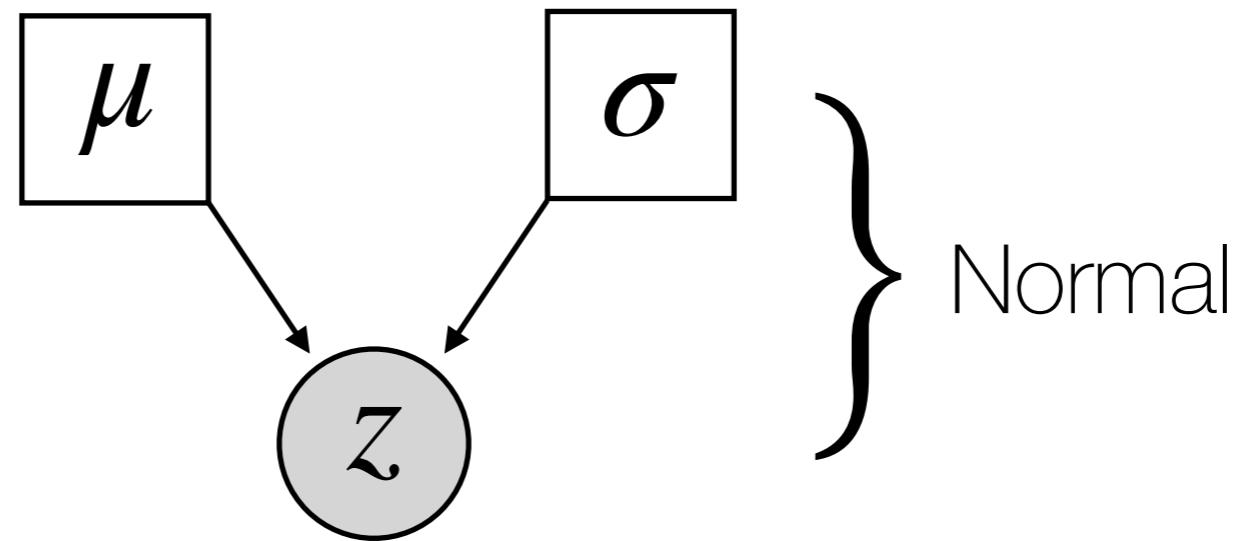
Using the model descriptions in the table, use the same principles we've been practicing to set up each of these models in RevBayes.

Fill out Table 2 for as many models as you can.

```
rb mcmc_JC.Rev
```

# End

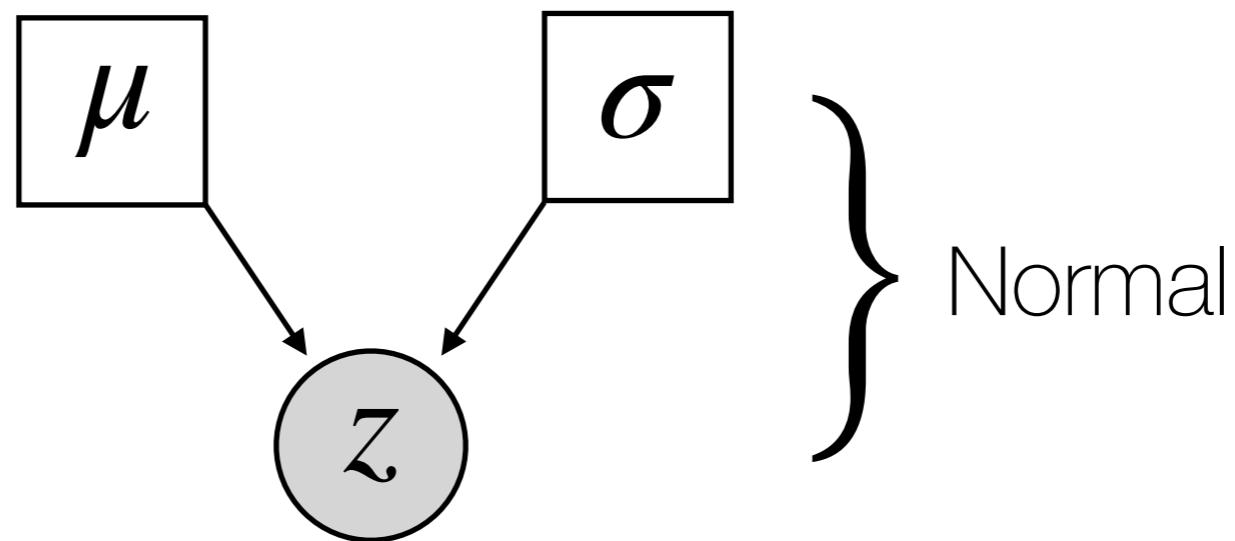
# Super Simple Models



```
m <- 10  
s <- 0.1  
z ~ dnNormal(m,s)  
z.clamp(10.01)
```

What can we infer about this model?

# Super Simple Models

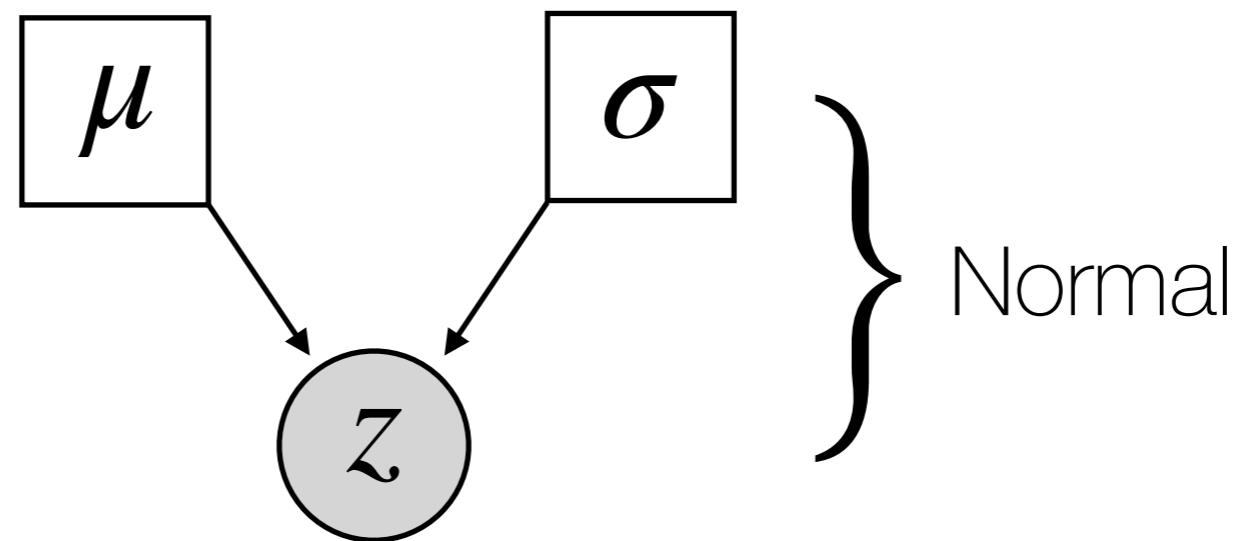


```
m <- 10  
s <- 0.1  
z ~ dnNormal(m,s)  
z.clamp(10.01)
```

What can we infer about this model?

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

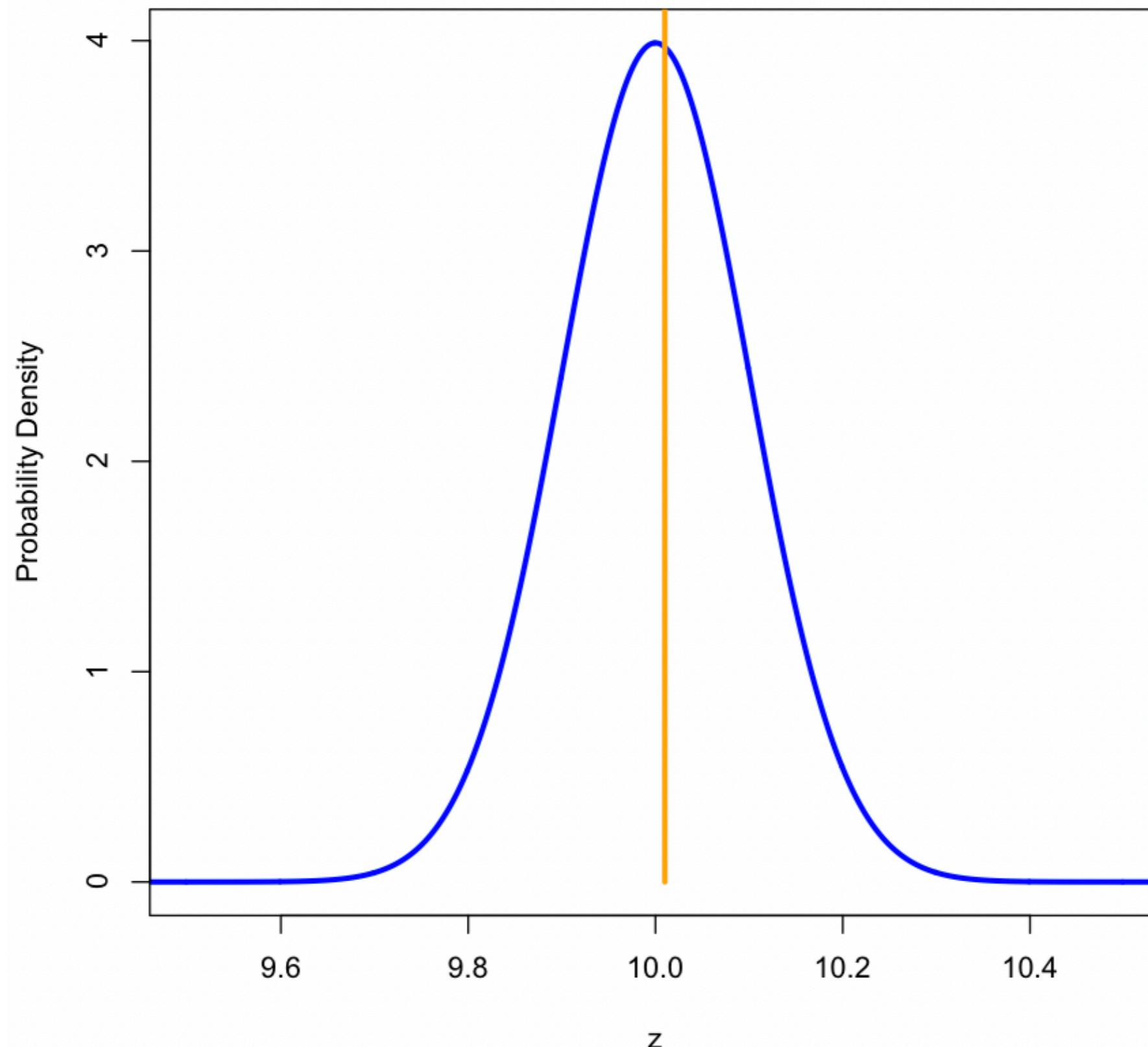
# Super Simple Models



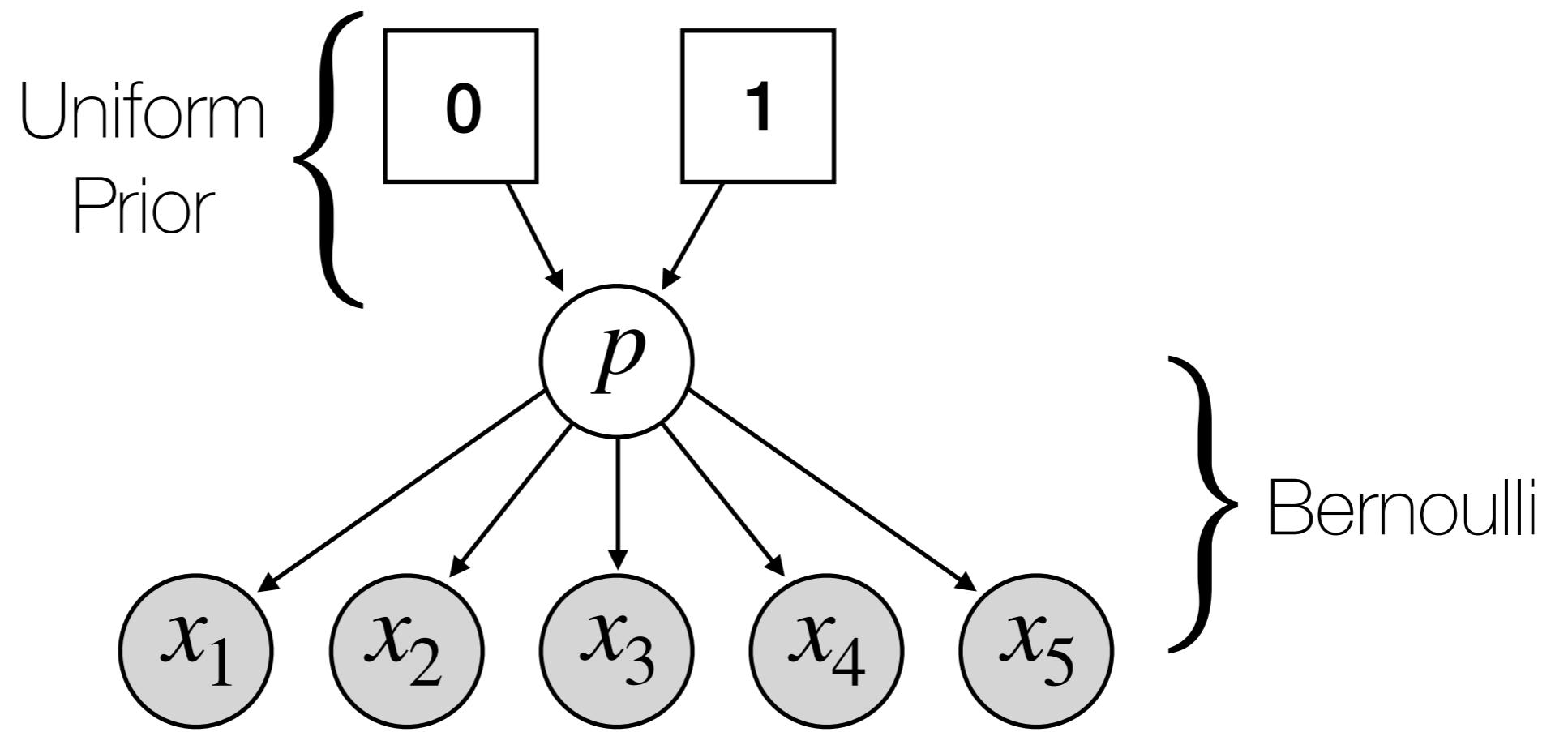
```
m <- 10  
s <- 0.1  
z ~ dnNormal(m,s)  
z.clamp(10.01)
```

What's the probability of our "observed" value?

# Probability Density Function (PDF)



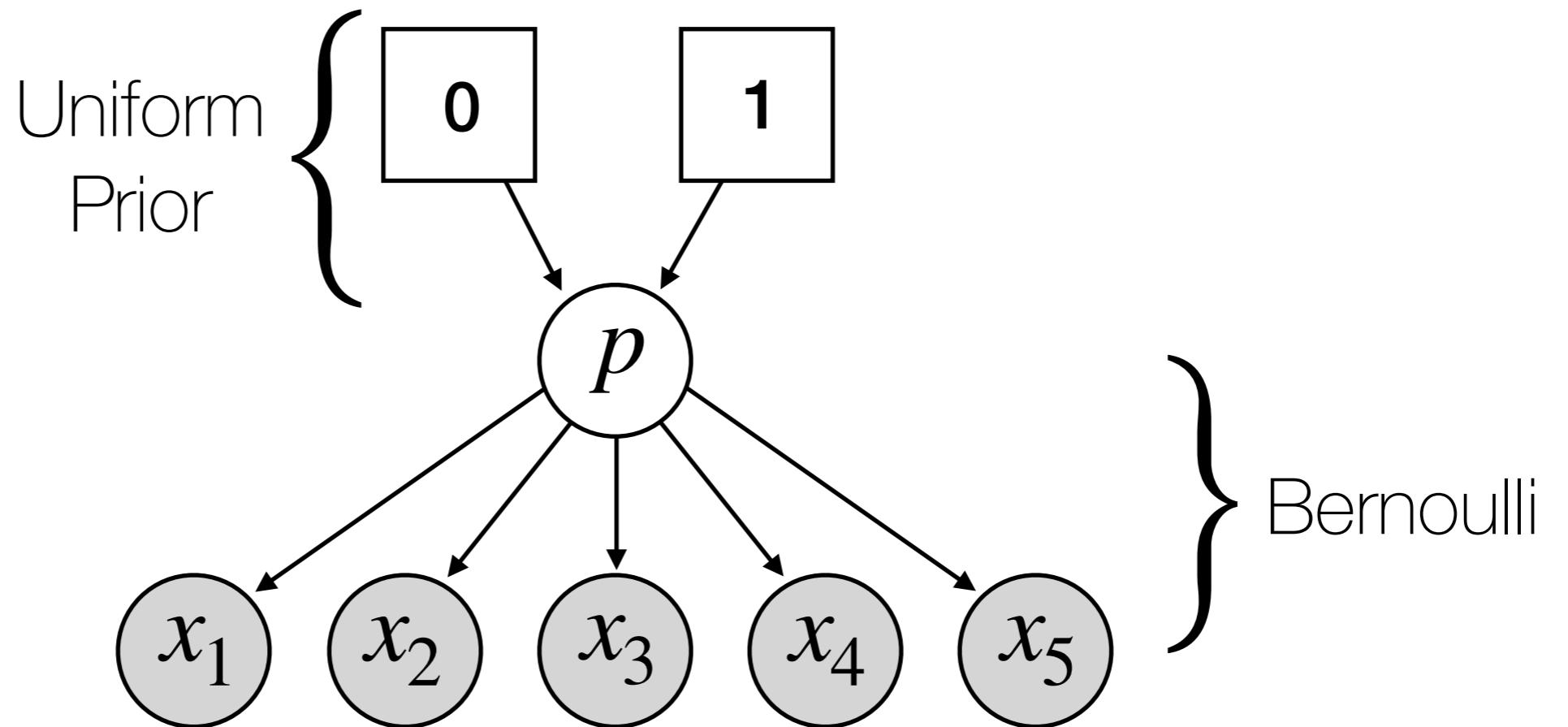
# Series of Bernoullis with Linked $p$



What can we infer about this model?

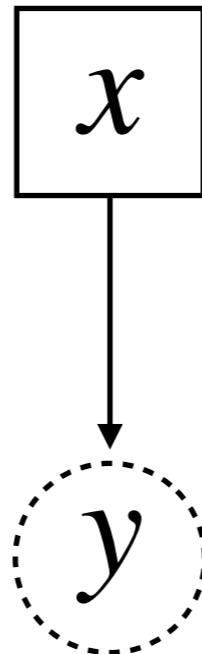
How much information will we have (relative to a single Bernoulli)?

# 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)
```

# Super Simple Models



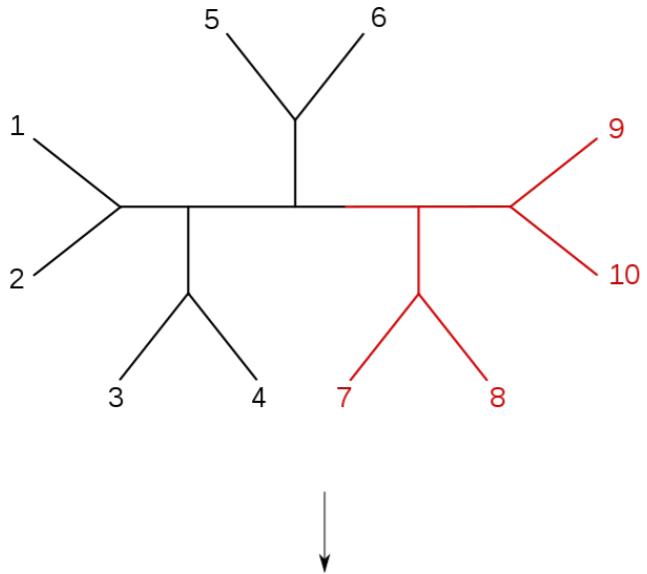
```
x <- 2  
y := x / 4
```

What's the structure of this model?

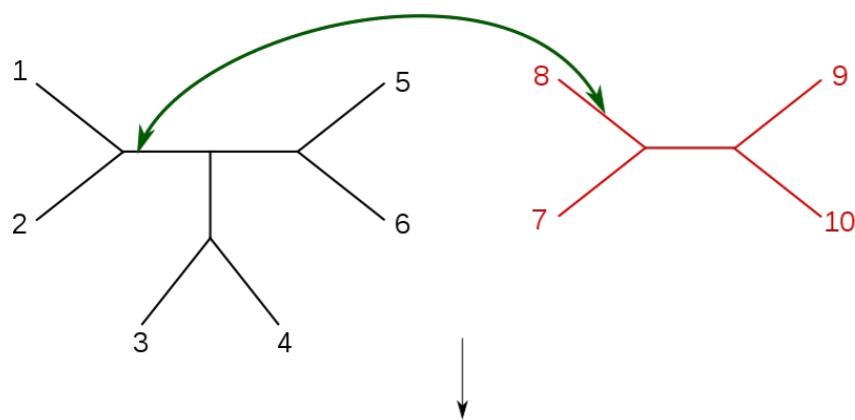
What would happen if  $x$  was set to 16?

**Try changing  $x$  without changing  $y$ , but then look at value of  $y$ .**

# Topology Moves



Tree Bisection and Reconnection  
(TBR)



Disruptive topological move.

