

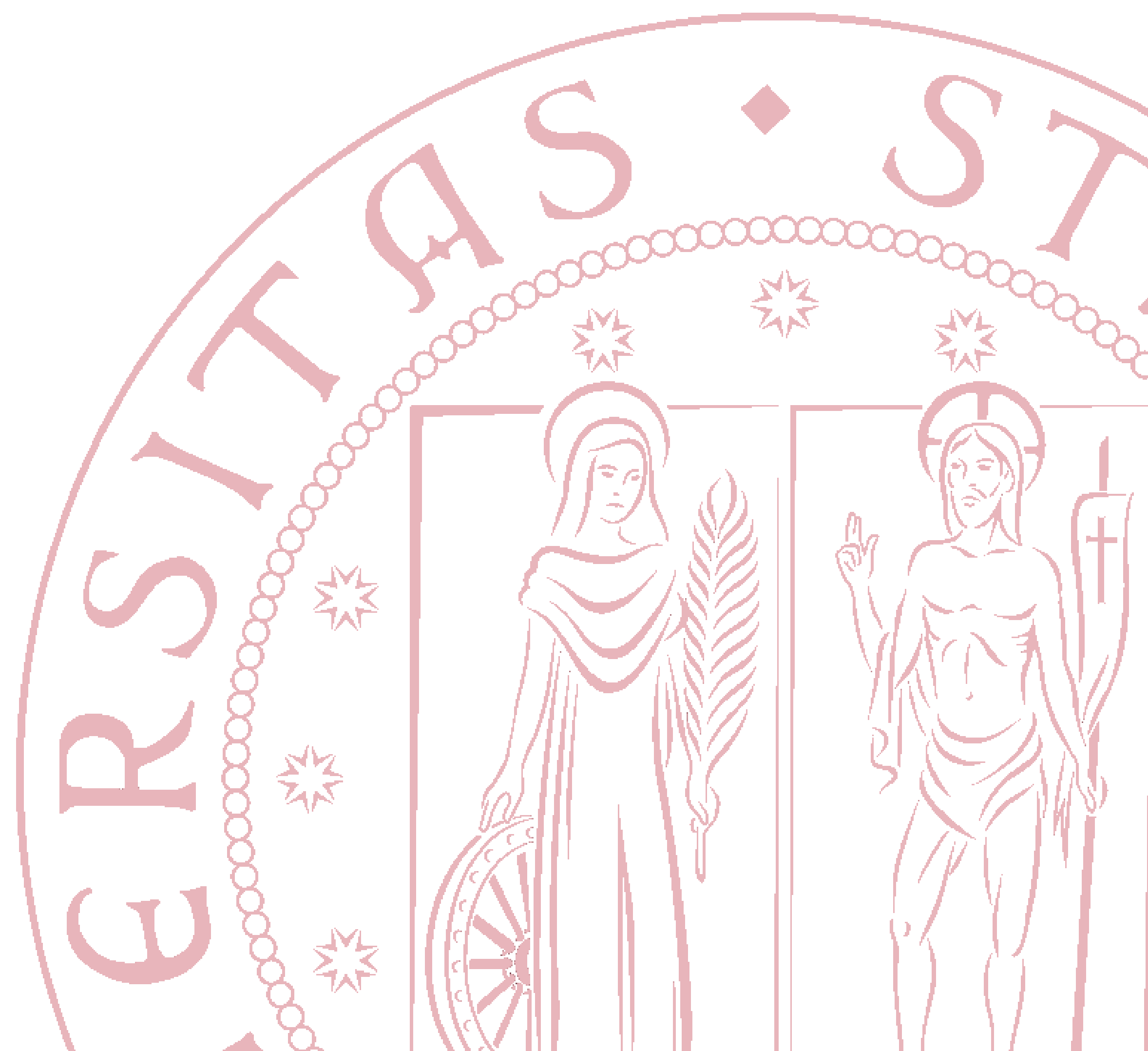
Approximate inference & Bayesian Networks (Part B)

Gloria Beraldo (gloria.beraldo@unipd.it)

Department of Information Engineering, University of Padova

Topics:

- Approximate inference
- Sampling from a probability distribution
- Sampling from a probability distribution: Sprinkler example
- Prior-Sampling
- Prior-Sampling: Sprinkler example
- Filtering: umbrella world example
- Pomegranate
- Pomegranate for umbrella world

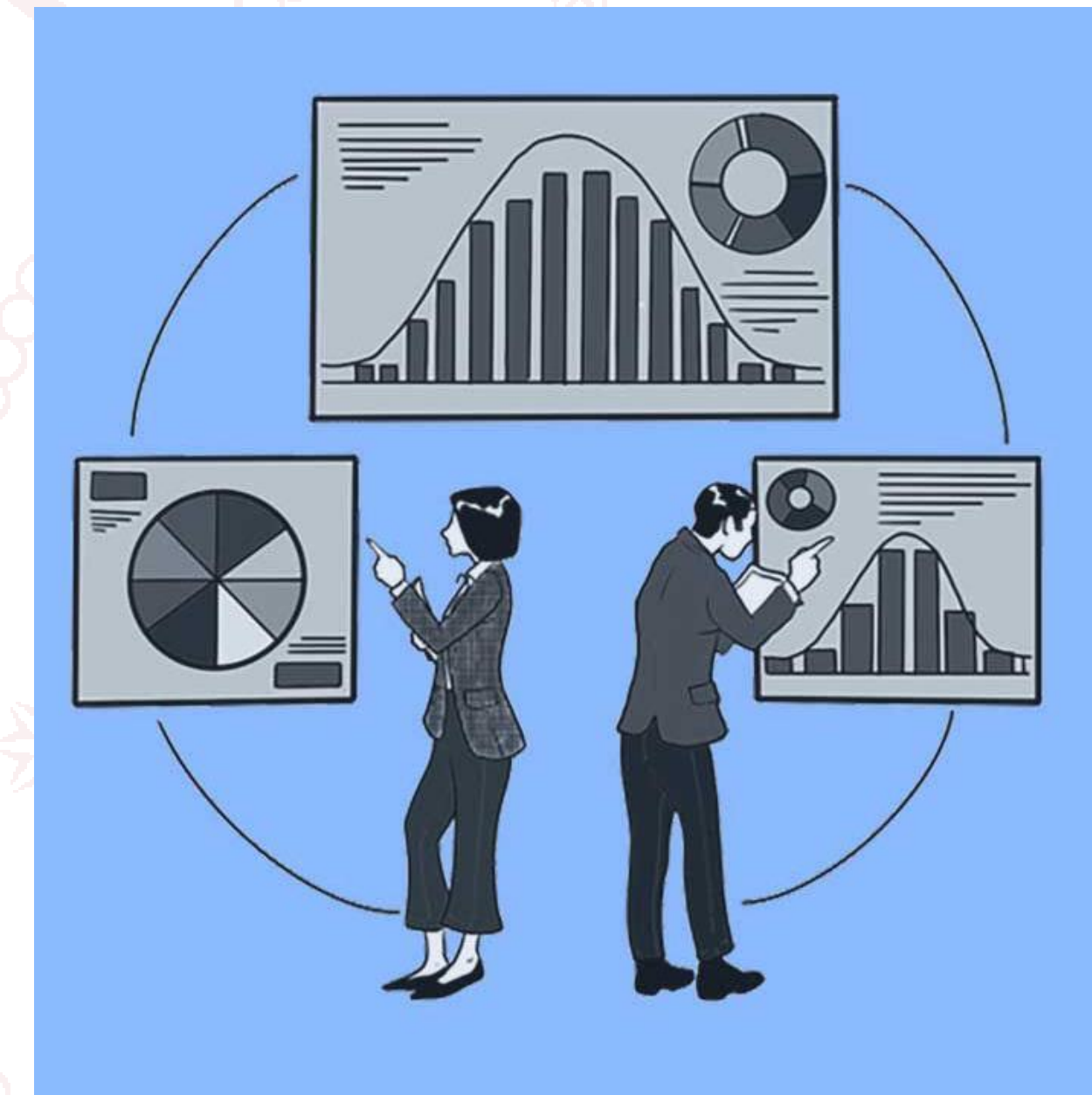


Sampling from a probability distribution

The methods seen last time (e.g. inference by enumeration) allow for exact inference

Approximate inference methods are therefore a viable alternative to give reasonable answers in case of large models

All the algorithms for **approximate inference** with Bayesian Networks require a **method for sampling** from a known probability distribution.



Sampling from a probability distribution

Let's see a possible implementation of such sampling method in case of **Boolean variables** and known **Conditional Probability Tables** (CPTs):

```
import numpy as np
import random as rnd

t, f = 0, 1

def samplegen(Pdist, Parents = []):
    assert len(Parents) < len(Pdist.shape)
    if rnd.random() < Pdist[t][tuple(Parents)]:
        return t
    return f
```

probability distribution with
the content of the CPT



Sampling from a probability distribution

Let's see a possible implementation of such sampling method in case of **Boolean variables** and known **Conditional Probability Tables** (CPTs):

```
import numpy as np
import random as rnd

t, f = 0, 1

def samplegen(Pdist, Parents = []):
    assert len(Parents) < len(Pdist.shape)
    if rnd.random() < Pdist[t][tuple(Parents)]:
        return t
    return f
```

if available, the values of the
Parents events



Sampling from a probability distribution

Let's see a possible implementation of such sampling method in case of **Boolean variables** and known **Conditional Probability Tables (CPTs)**:

```
import numpy as np
import random as rnd

t, f = 0, 1

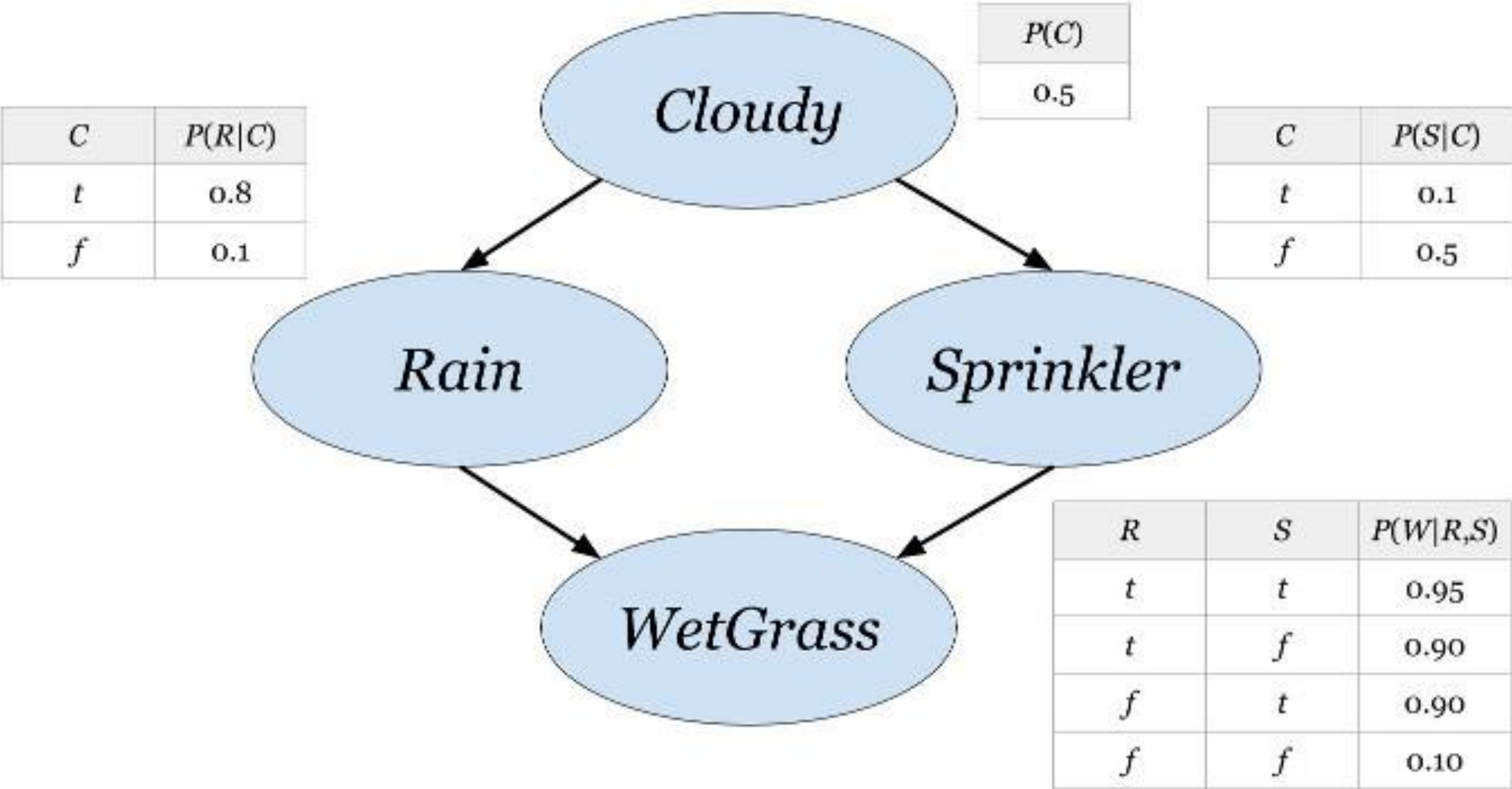
def samplegen(Pdist, Parents = []):
    assert len(Parents) < len(Pdist.shape)
    if rnd.random() < Pdist[t][tuple(Parents)]:
        return t
    return f
```

<https://docs.python.org/3/library/random.html>

random.random()
Return the next random
floating point number in
the range $0.0 \leq X < 1.0$

Sampling from a probability distribution: Sprinkler example

Let's consider the scenario in the figure, where the WetGrass can be caused a Sprinkler or the Rain, both of which depend on the Cloudy weather:



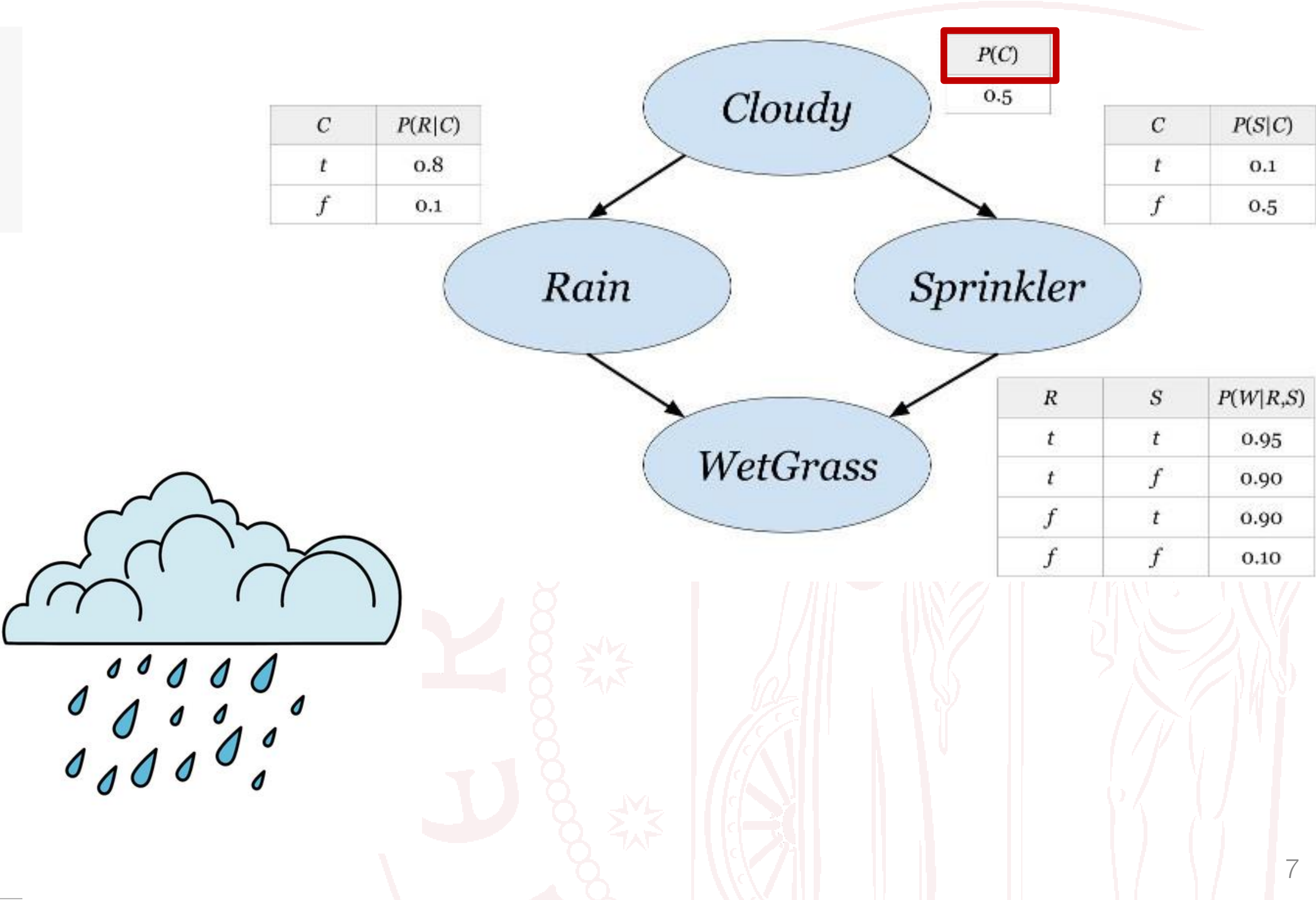
Sampling from a probability distribution: Sprinkler example

In particular, let's say we want to generate 20 samples from the Cloudy distribution $P(C)$ which has no parents:

```
P_C = np.array([0.5, 0.5])

for i in range(20):
    s = samplegen(P_C)
    print("C = ", 'TRUE' if s == t else 'FALSE')
```

C = FALSE
C = TRUE
C = FALSE
C = FALSE
C = TRUE
C = FALSE
C = FALSE
C = FALSE
C = TRUE
C = TRUE
C = FALSE
C = TRUE
C = FALSE
C = FALSE
C = FALSE
C = FALSE
C = FALSE
C = TRUE
C = TRUE



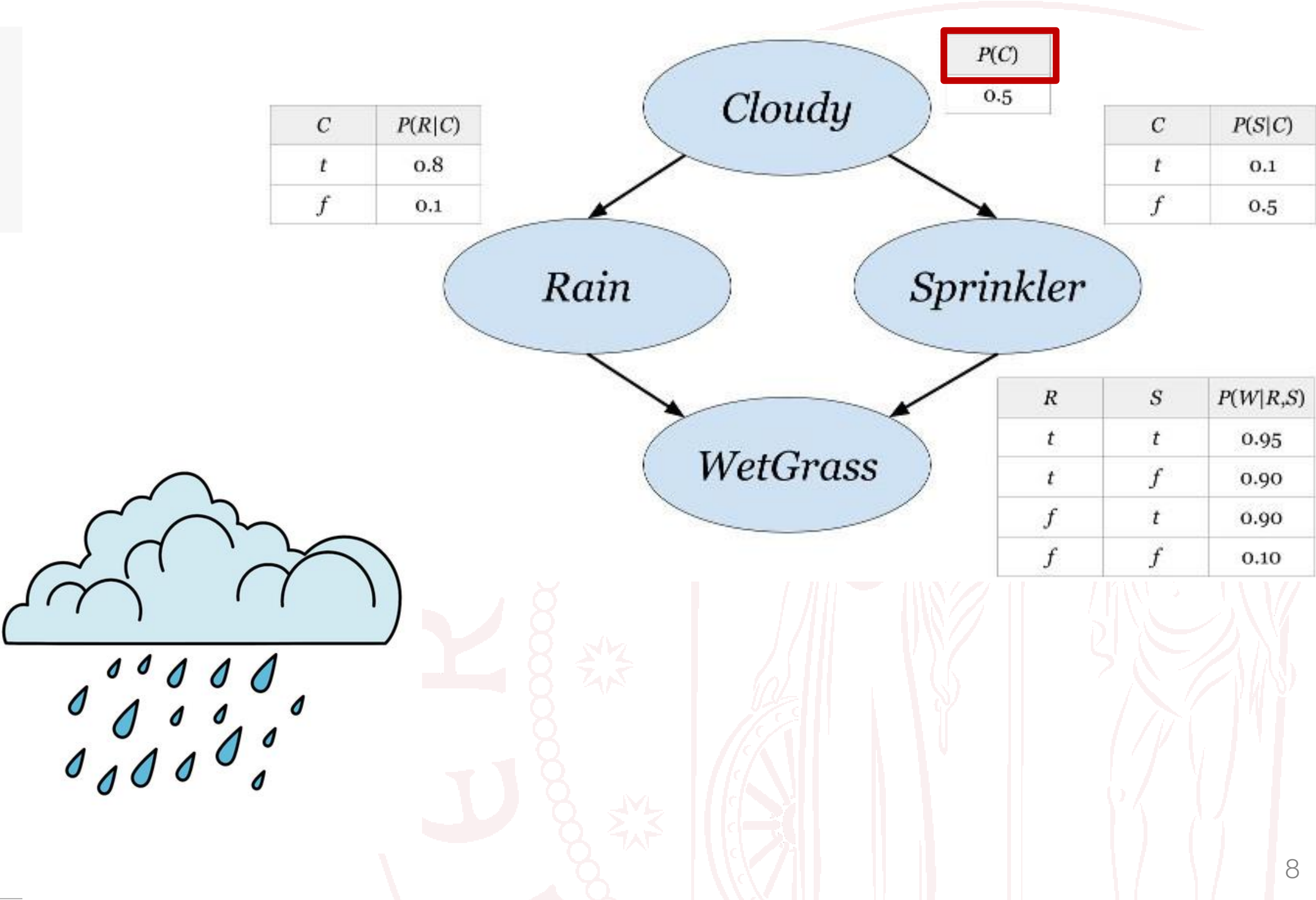
Sampling from a probability distribution: Sprinkler example

In particular, let's say we want to generate 20 samples from the Cloudy distribution $P(C)$ which has **no parents**:

```
P_C = np.array([0.5, 0.5])

for i in range(20):
    s = samplegen(P_C)
    print("C = ", 'TRUE' if s == t else 'FALSE')
```

C = FALSE
C = TRUE
C = FALSE
C = FALSE
C = TRUE
C = FALSE
C = FALSE
C = FALSE
C = TRUE
C = TRUE
C = FALSE
C = TRUE
C = FALSE
C = FALSE
C = FALSE
C = FALSE
C = FALSE
C = TRUE
C = TRUE



Sampling from a probability distribution: Sprinkler example

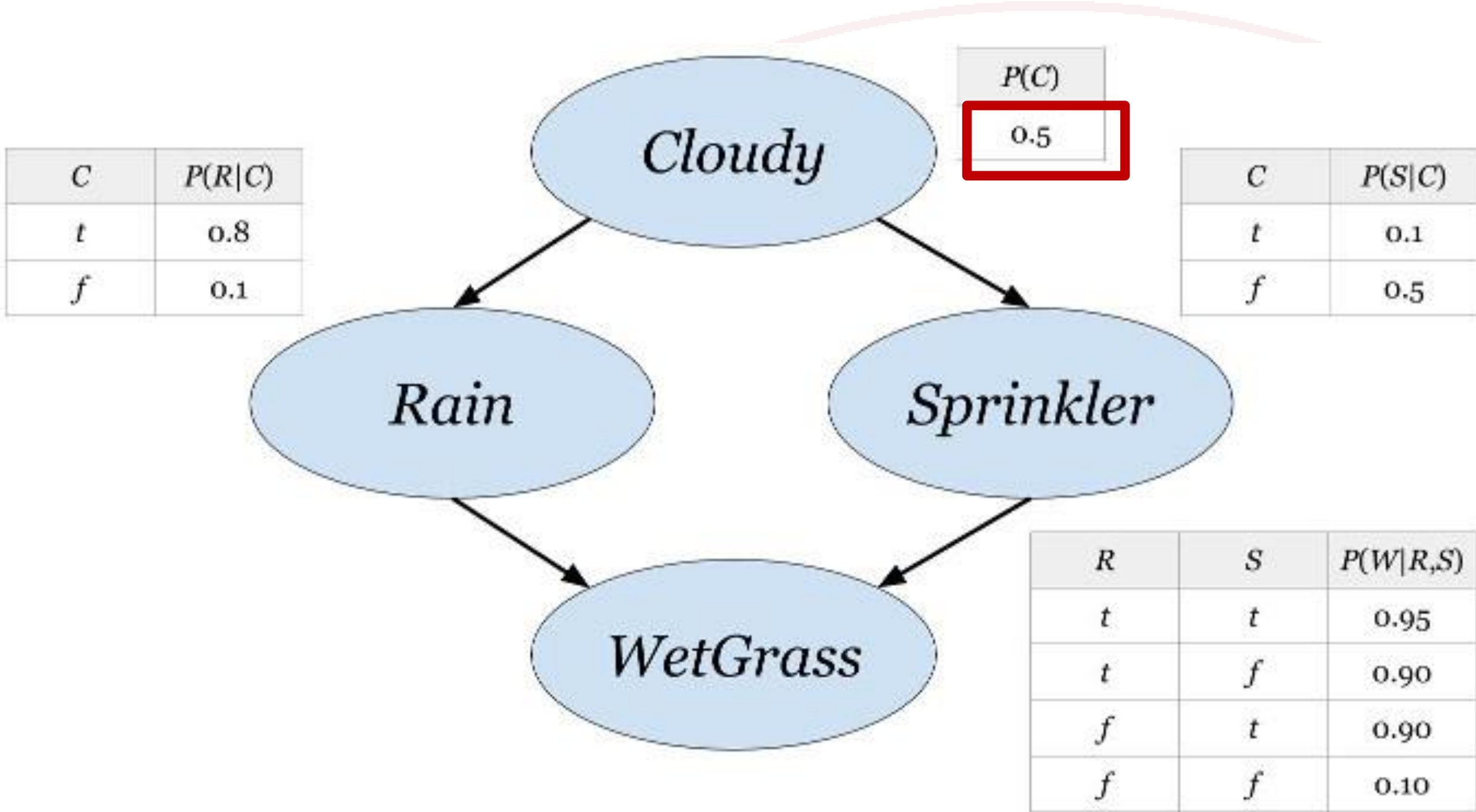
In particular, let's say we want to generate 20 samples from the Cloudy distribution $P(C)$ which has **no parents**:

```
P_C = np.array([0.5, 0.5])

for i in range(20):
    s = samplegen(P_C)
    print("C = ", 'TRUE' if s == t else 'FALSE')
```

C = FALSE
C = TRUE
C = FALSE
C = FALSE
C = TRUE
C = FALSE
C = FALSE
C = FALSE
C = TRUE
C = TRUE
C = FALSE
C = TRUE
C = FALSE
C = FALSE
C = FALSE
C = FALSE
C = FALSE
C = TRUE
C = TRUE

As expected, there are approximately 50% true and 50% false outcomes.



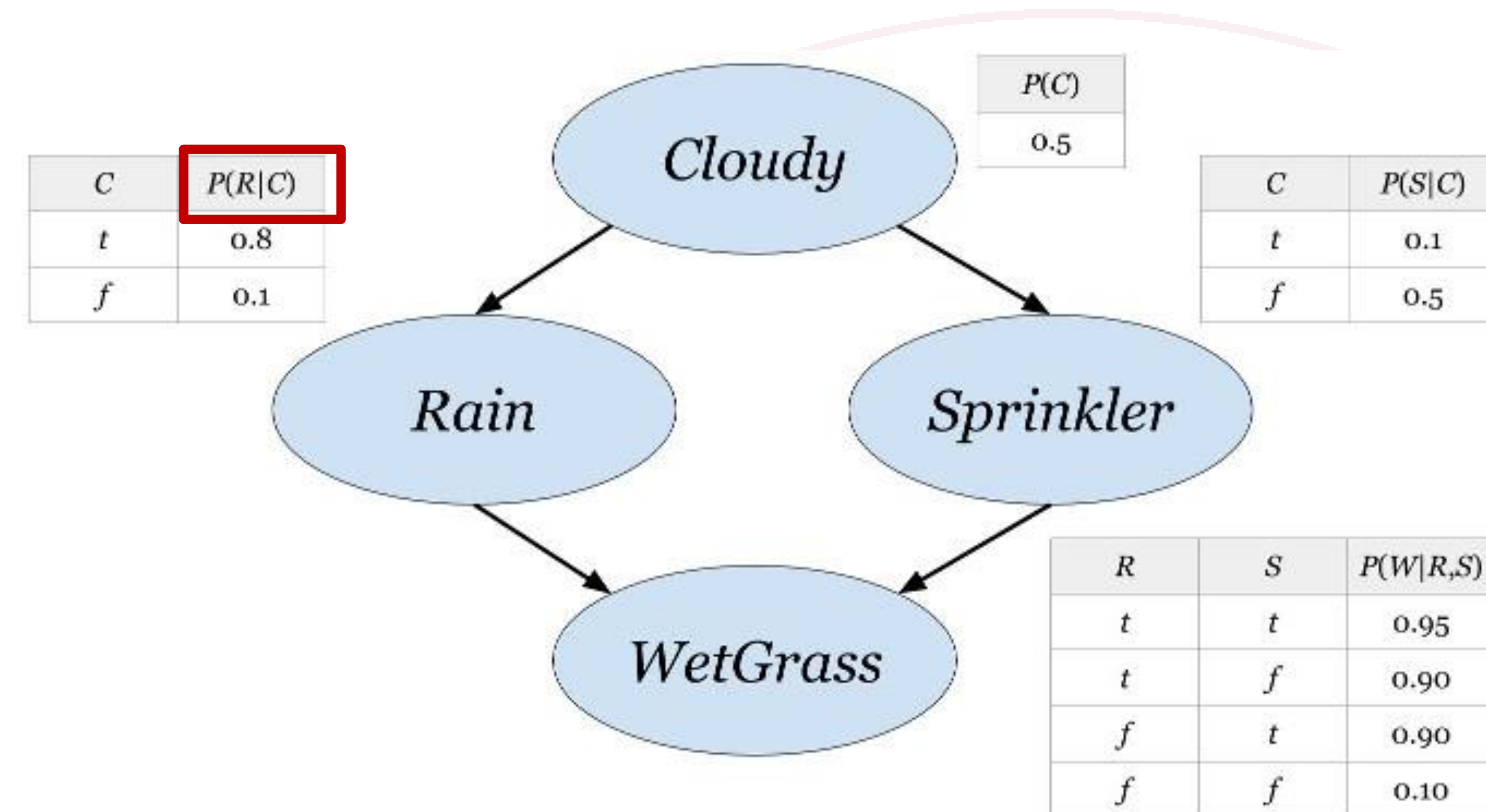
Sampling from a probability distribution: Sprinkler example

Let's sample from a **conditional distribution**, for example $P(R|\neg c)$

```
P_R_C = np.array([[0.8, 0.1],[0.2, 0.9]])

for i in range(20):
    s = samplegen(P_R_C, [f])
    print("R = ", 'TRUE' if s == t else 'FALSE')
```

R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = TRUE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE



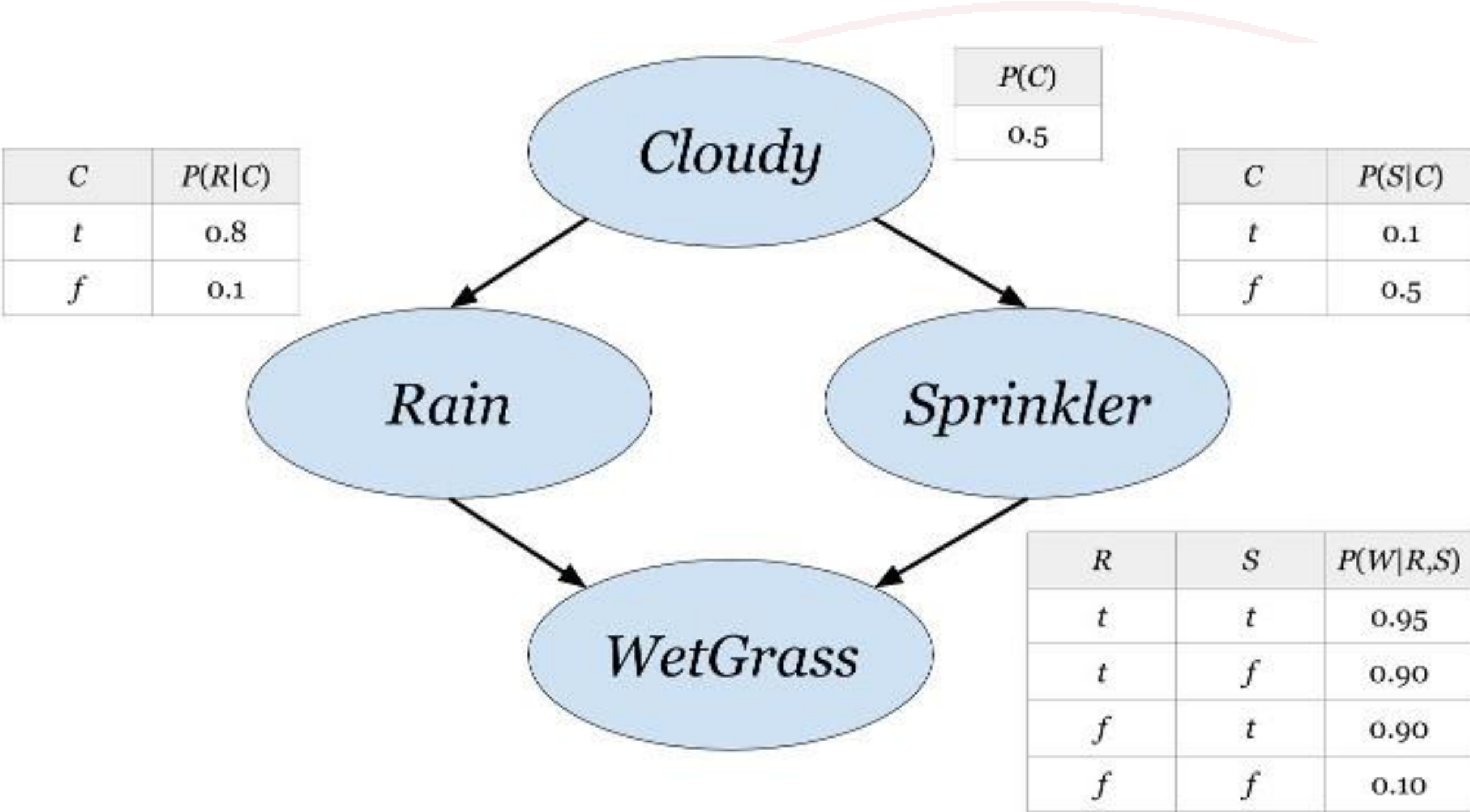
Sampling from a probability distribution: Sprinkler example

Let's sample from a **conditional distribution**, for example $P(R \mid \neg C)$

```
P_R_C = np.array([[0.8, 0.1],[0.2, 0.9]])

for i in range(20):
    s = samplegen(P_R_C, [f])
    print("R = ", 'TRUE' if s == t else 'FALSE')
```

R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = TRUE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE
R = FALSE



Sampling from a probability distribution: Sprinkler example

Let's sample from a **conditional distribution**, for example $P(R \mid \neg C)$

```
P_R_C = np.array([[0.8, 0.1],[0.2, 0.9]])

for i in range(20):
    s = samplegen(P_R_C, [f])
    print("R = ", 'TRUE' if s == t else 'FALSE')
```

R = FALSE

R = FALSE

R = FALSE

R = FALSE

R = FALSE

R = FALSE

R = FALSE

R = FALSE

R = FALSE

R = TRUE

R = FALSE

R = FALSE

R = FALSE

R = FALSE

R = FALSE

R = FALSE

R = FALSE

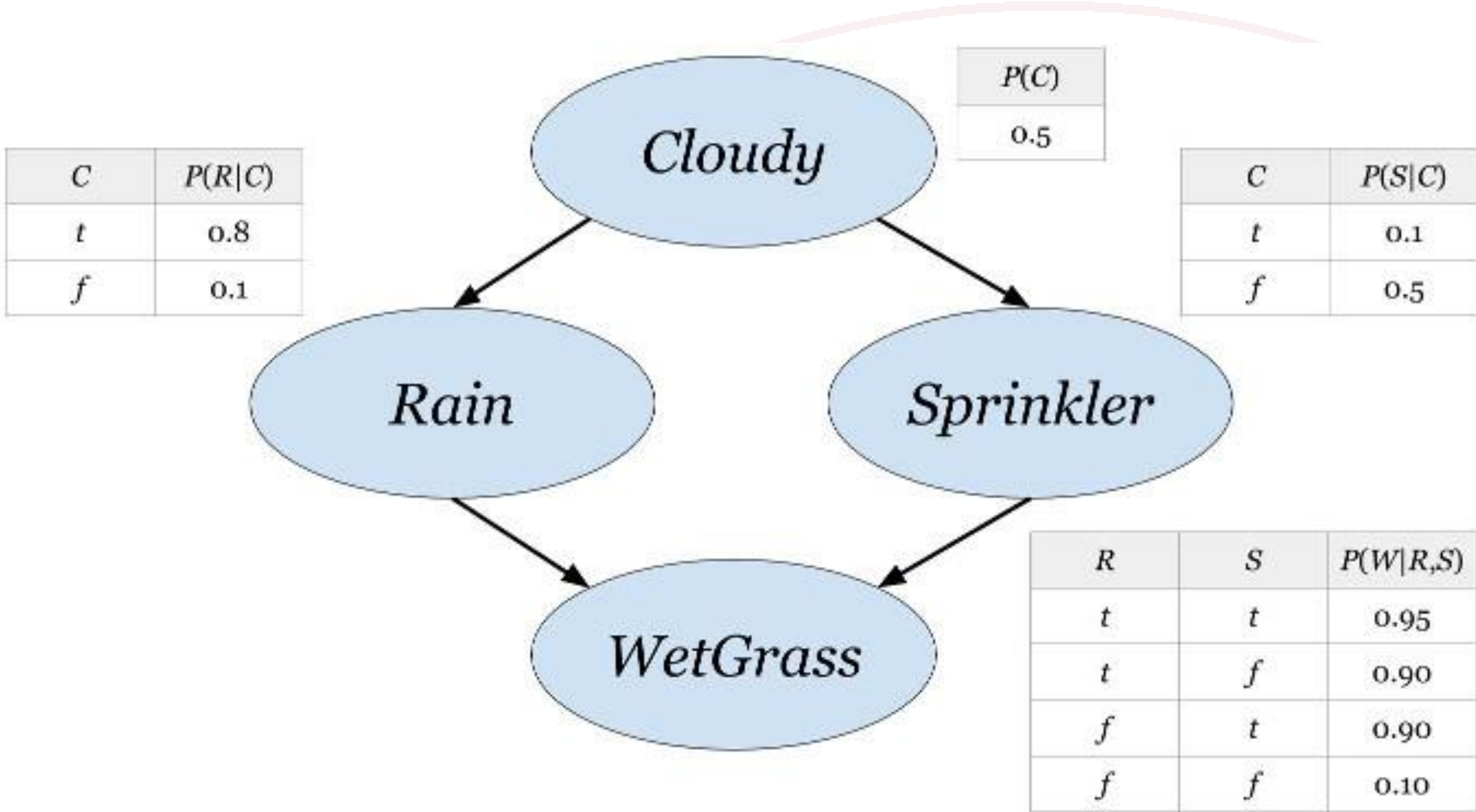
R = FALSE

R = FALSE

R = FALSE

In this case there are many more false than true outcomes because

$$P(R \mid \neg C) = \langle 0.1, 0.9 \rangle$$



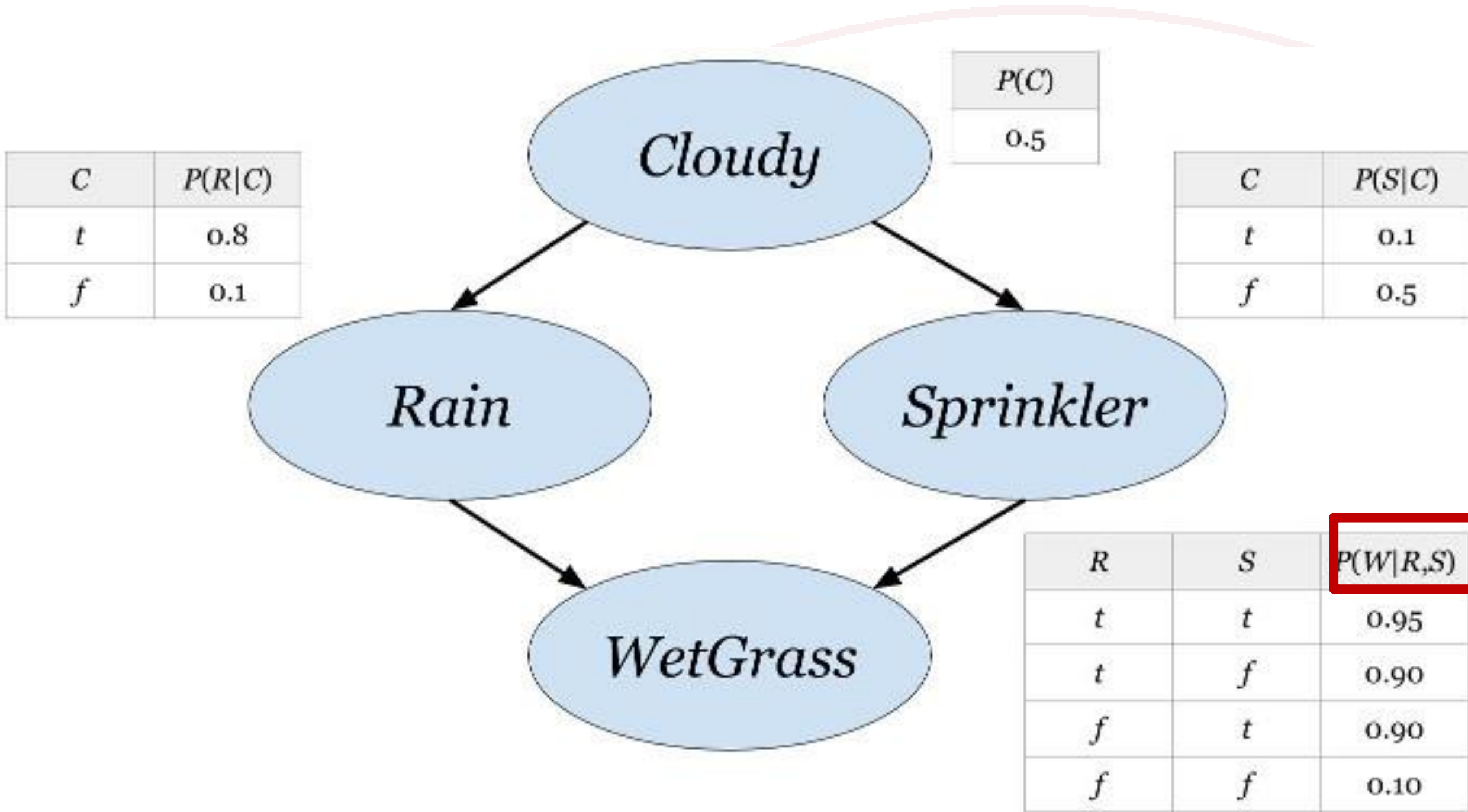
Sampling from a probability distribution: Sprinkler example

Finally, let's sample $P(W|\neg s, r) = \langle 0.9, 0.1 \rangle$

```
P_W_SR = np.array([[[0.95, 0.9],[0.9, 0.1]], [[0.05, 0.1],[0.1, 0.9]]])

for i in range(20):
    s = samplegen(P_W_SR, [f, t])
    print("W = ", 'TRUE' if s == t else 'FALSE')
```

W = FALSE
W = TRUE
W = TRUE
W = TRUE
W = TRUE
W = FALSE
W = TRUE
W = TRUE
W = FALSE
W = TRUE
W = TRUE
W = TRUE
W = TRUE
W = TRUE
W = TRUE
W = TRUE
W = TRUE
W = TRUE
W = TRUE
W = TRUE



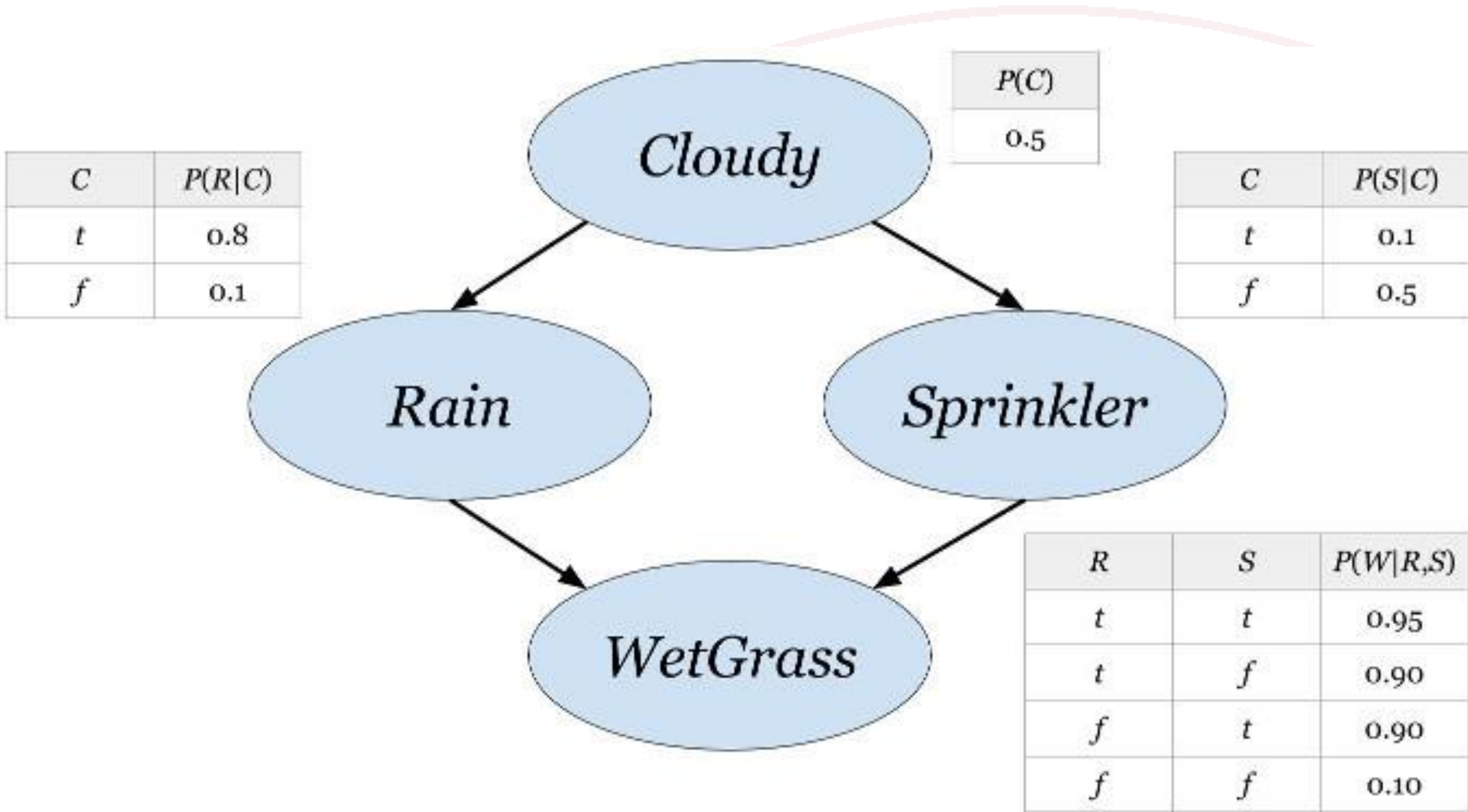
Sampling from a probability distribution: Sprinkler example

Finally, let's sample $P(W|\neg s, r) = \langle 0.9, 0.1 \rangle$

```
P_W_SR = np.array([[[0.95, 0.9],[0.9, 0.1]], [[0.05, 0.1],[0.1, 0.9]]])

for i in range(20):
    s = samplegen(P_W_SR, [f, t])
    print("W = ", 'TRUE' if s == t else 'FALSE')
```

W = FALSE
W = TRUE
W = TRUE
W = TRUE
W = TRUE
W = FALSE
W = TRUE
W = TRUE
W = FALSE
W = TRUE
W = TRUE
W = TRUE
W = TRUE
W = TRUE
W = TRUE
W = TRUE
W = TRUE
W = TRUE
W = TRUE
W = TRUE



Sampling from a probability distribution: Sprinkler example

Finally, let's sample $P(W|\neg s, r) = \langle 0.9, 0.1 \rangle$

```
P_W_SR = np.array([[[0.95, 0.9],[0.9, 0.1]], [[0.05, 0.1],[0.1, 0.9]]])

for i in range(20):
    s = samplegen(P_W_SR, [f, t])
    print("W = ", 'TRUE' if s == t else 'FALSE')
```

W = FALSE

W = TRUE

W = TRUE

W = TRUE

W = TRUE

W = FALSE

W = TRUE

W = TRUE

W = FALSE

W = TRUE

W = TRUE

W = TRUE

W = TRUE

W = TRUE

W = TRUE

W = TRUE

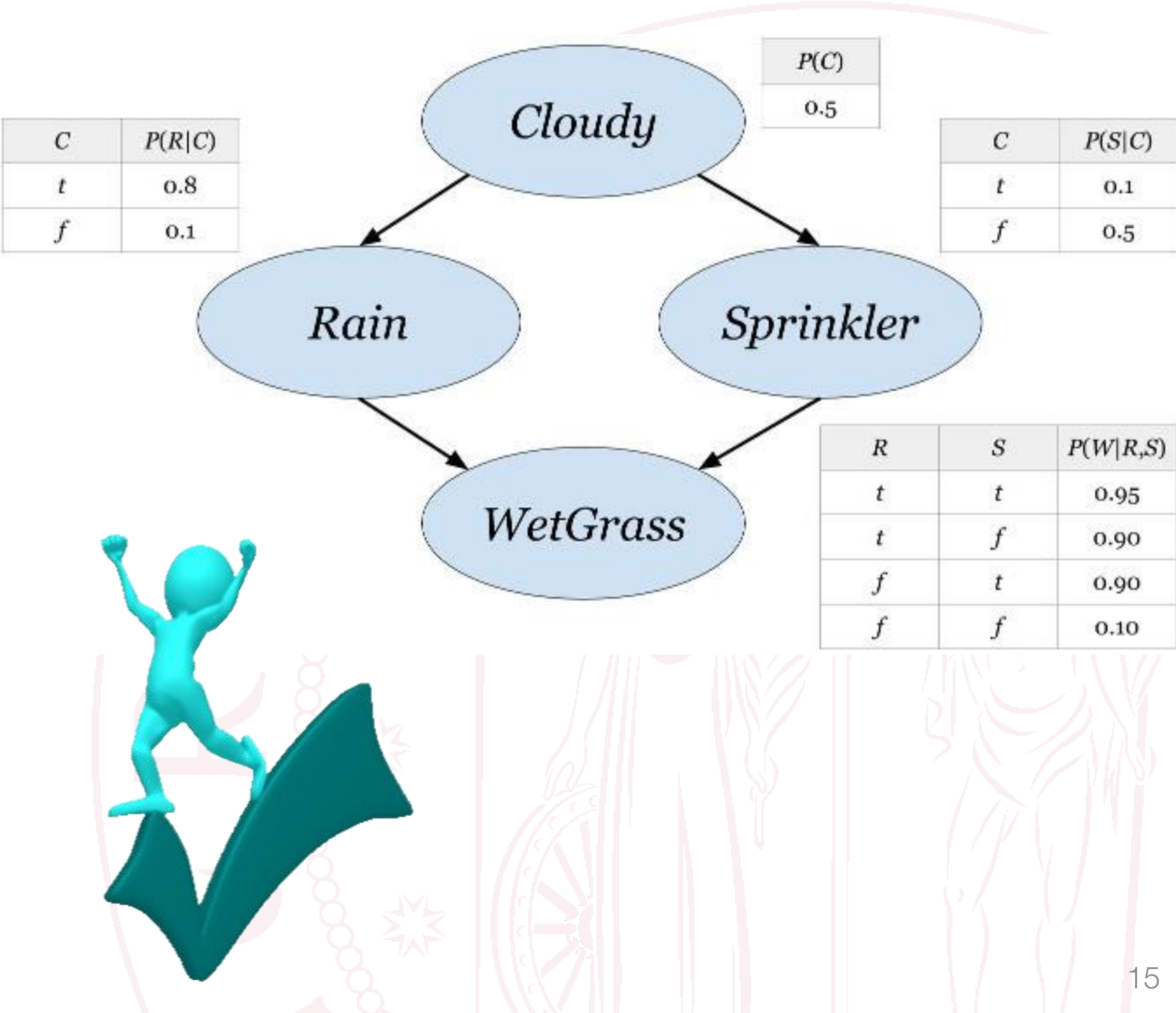
W = TRUE

W = TRUE

W = TRUE

W = TRUE

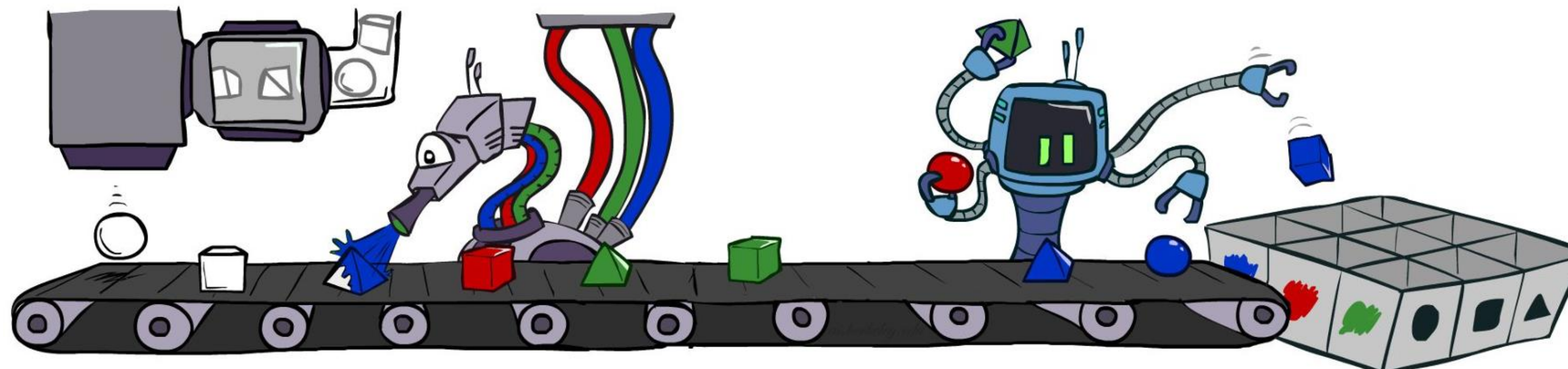
Even in this case, the samples are more or less as expected, about 90% true and 10% false.



Prior-Sampling

```
function PRIOR-SAMPLE( $bn$ ) returns an event sampled from the prior specified by  $bn$   
inputs:  $bn$ , a Bayesian network specifying joint distribution  $\mathbf{P}(X_1, \dots, X_n)$   
  
 $\mathbf{x} \leftarrow$  an event with  $n$  elements  
for each variable  $X_i$  in  $X_1, \dots, X_n$  do  
     $\mathbf{x}[i] \leftarrow$  a random sample from  $\mathbf{P}(X_i \mid \text{parents}(X_i))$   
return  $\mathbf{x}$ 
```

- For $i=1, 2, \dots, n$ (in topological order)
 - Sample X_i from $\mathbf{P}(X_i \mid \text{parents}(X_i))$
- Return (x_1, x_2, \dots, x_n)



Prior-Sampling: Sprinkler example

Let's implement now the Prior Sampling algorithm starting with some data structures to represent the Sprinkler network:

```
function PRIOR-SAMPLE( $bn$ ) returns an event sampled from the prior specified by  $bn$   
  inputs:  $bn$ , a Bayesian network specifying joint distribution  $\mathbf{P}(X_1, \dots, X_n)$   
  
   $\mathbf{x} \leftarrow$  an event with  $n$  elements  
  for each variable  $X_i$  in  $X_1, \dots, X_n$  do  
     $\mathbf{x}[i] \leftarrow$  a random sample from  $\mathbf{P}(X_i \mid \text{parents}(X_i))$   
  return  $\mathbf{x}$ 
```

```
# some of these distributions were already defined before, but we repeat them just in case
```

```
P_C = np.array([0.5, 0.5])
```

```
P_S_C = np.array([[0.1, 0.5], [0.9, 0.5]])
```

```
P_R_C = np.array([[0.8, 0.1], [0.2, 0.9]])
```

```
P_W_SR = np.array([[[0.95, 0.9], [0.9, 0.1]], [[0.05, 0.1], [0.1, 0.9]]])
```

```
# network variables...
```

```
var = ['C', 'S', 'R', 'W']
```

```
# their distributions...
```

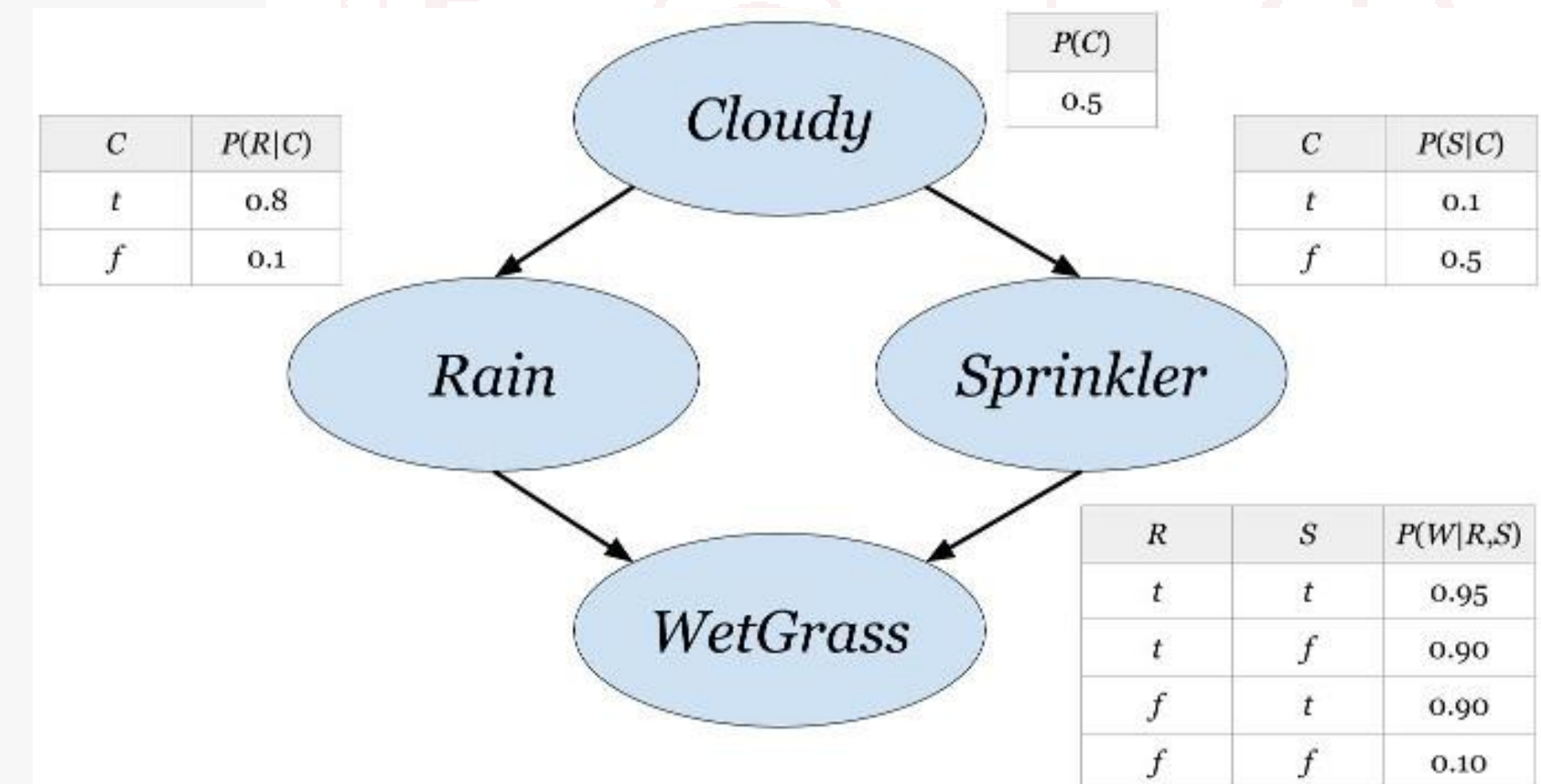
```
prd = {'C':P_C, 'S':P_S_C, 'R':P_R_C, 'W':P_W_SR}
```

```
# their parents...
```

```
par = {'C':[], 'S':['C'], 'R':['C'], 'W':['S', 'R']}
```

```
# and their initial values
```

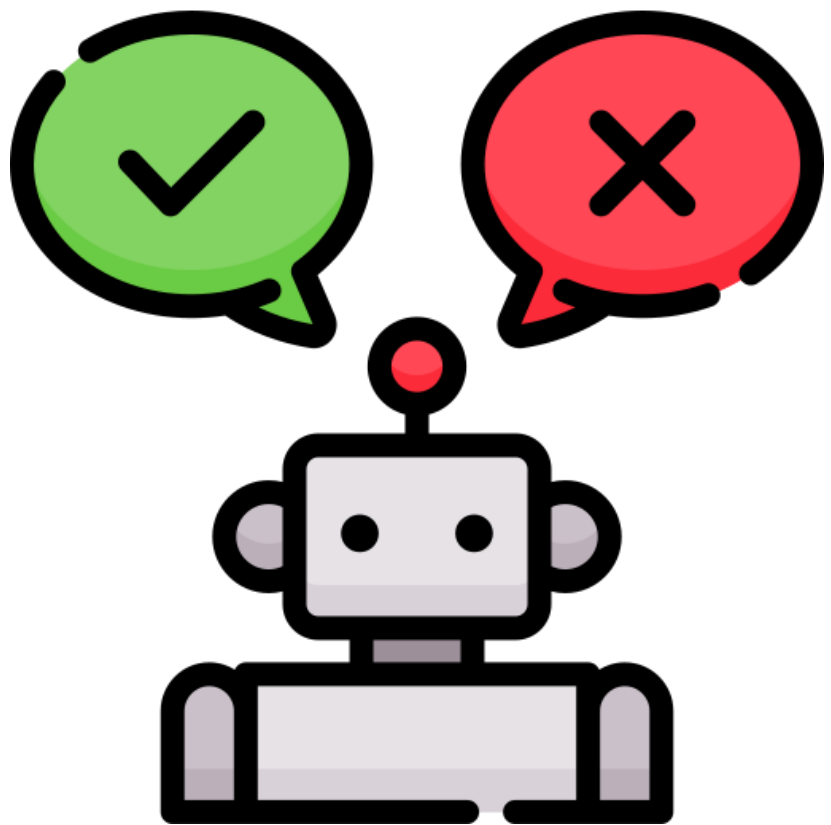
```
val = {'C':f, 'S':f, 'R':f, 'W':f}
```



Prior-Sampling: Sprinkler example

Let's define also a function to **retrieve the values** of the parents of a variable:

```
def parents(X):  
    return [val[i] for i in par[X]]
```



```
# some of these distributions were already defined before, but we repeat them just in case  
P_C = np.array([0.5, 0.5])  
P_S_C = np.array([[0.1, 0.5],[0.9, 0.5]])  
P_R_C = np.array([[0.8, 0.1],[0.2, 0.9]])  
P_W_SR = np.array([[[0.95, 0.9],[0.9, 0.1]], [[0.05, 0.1],[0.1, 0.9]]])  
  
# network variables...  
var = ['C','S','R','W']  
# their distributions...  
prd = {'C':P_C, 'S':P_S_C, 'R':P_R_C, 'W':P_W_SR}  
# their parents...  
par = {'C':[], 'S':['C'], 'R':['C'], 'W':['S','R']}  
# and their initial values  
val = {'C':f, 'S':f, 'R':f, 'W':f}
```

Prior-Sampling: Sprinkler example

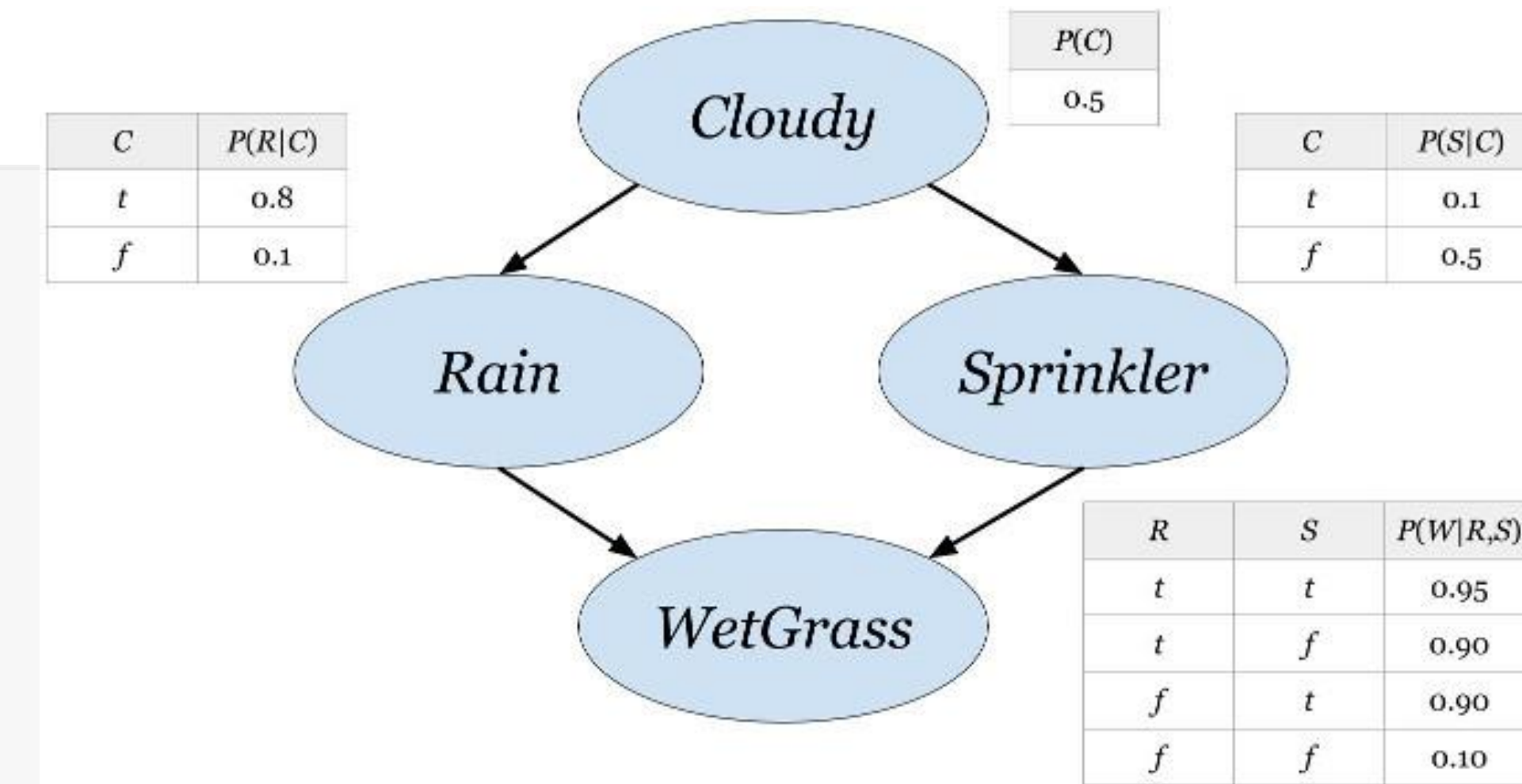
The following algorithm generates 1000 events from the Sprinkler network:

```
event = []

for n in range(1000):
    for x in var:
        val[x] = samplegen(prd[x], parents(x))
    event.append(['f' if val[x] else 't' for x in var])

print("First randomly generated event = ", event[0])
print("Number of randomly generated events = ", len(event))
```

```
First randomly generated event = ['t', 'f', 't', 't']
Number of randomly generated events = 1000
```



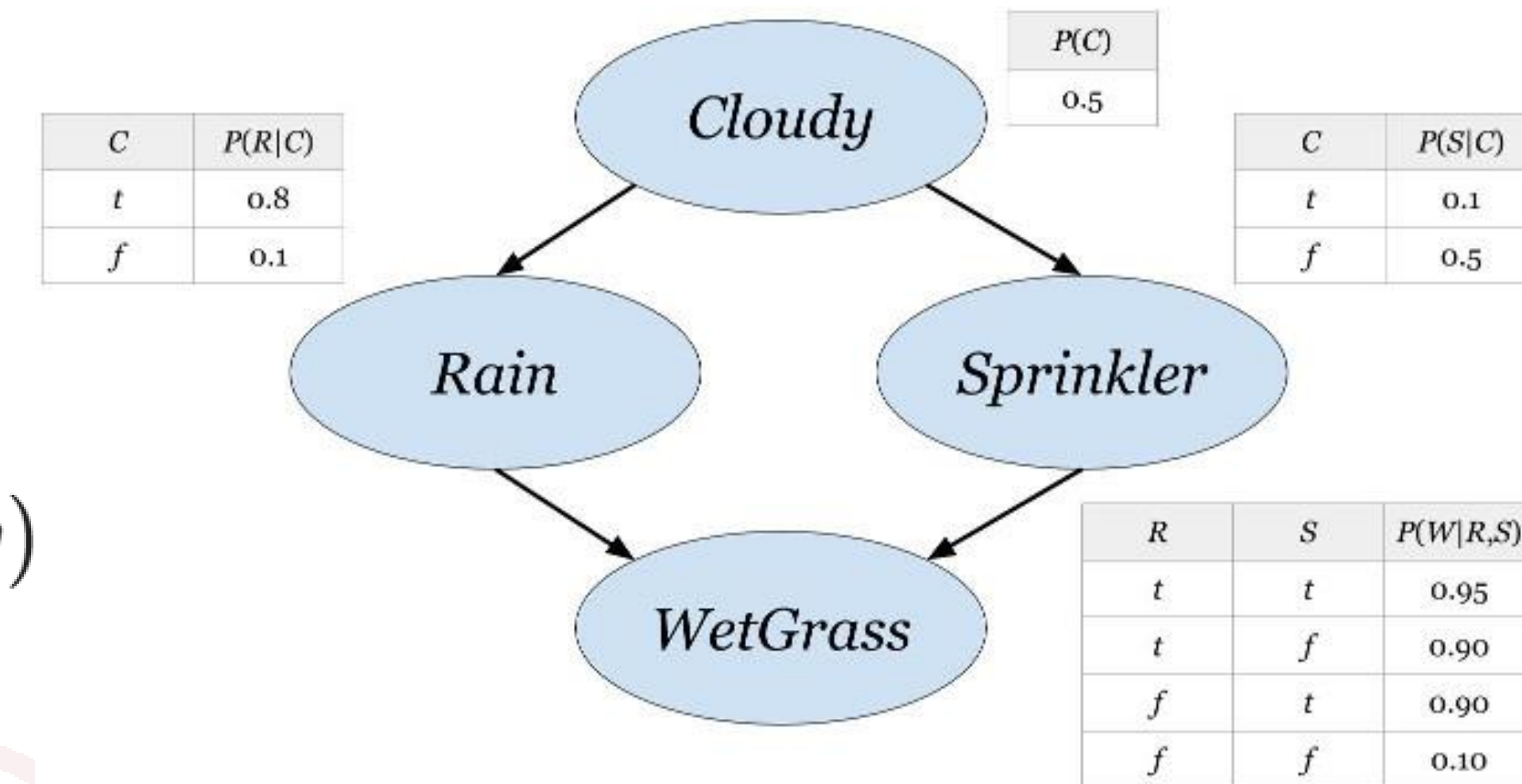
```
# network variables...
var = ['C','S','R','W']
# their distributions...
prd = {'C':P_C, 'S':P_S_C, 'R':P_R_C, 'W':P_W_SR}
# their parents...
par = {'C':[], 'S':['C'], 'R':['C'], 'W':['S','R']}
# and their initial values
val = {'C':f, 'S':f, 'R':f, 'W':f}
```

Prior-Sampling: Sprinkler example

Finally, we can compute the probability of any event by counting the number of times it was generated and normalising.

For example, we can verify

$$P(c, \neg s, r, w) = 0.328 \approx \frac{N_{PS}(c, \neg s, r, w)}{N} = \hat{P}(c, \neg s, r, w)$$



```
P_query = event.count(['t', 'f', 't', 't']) / len(event)
print("P(c,¬s,r,w) = ", P_query)
```

$P(c, \neg s, r, w) = 0.328$

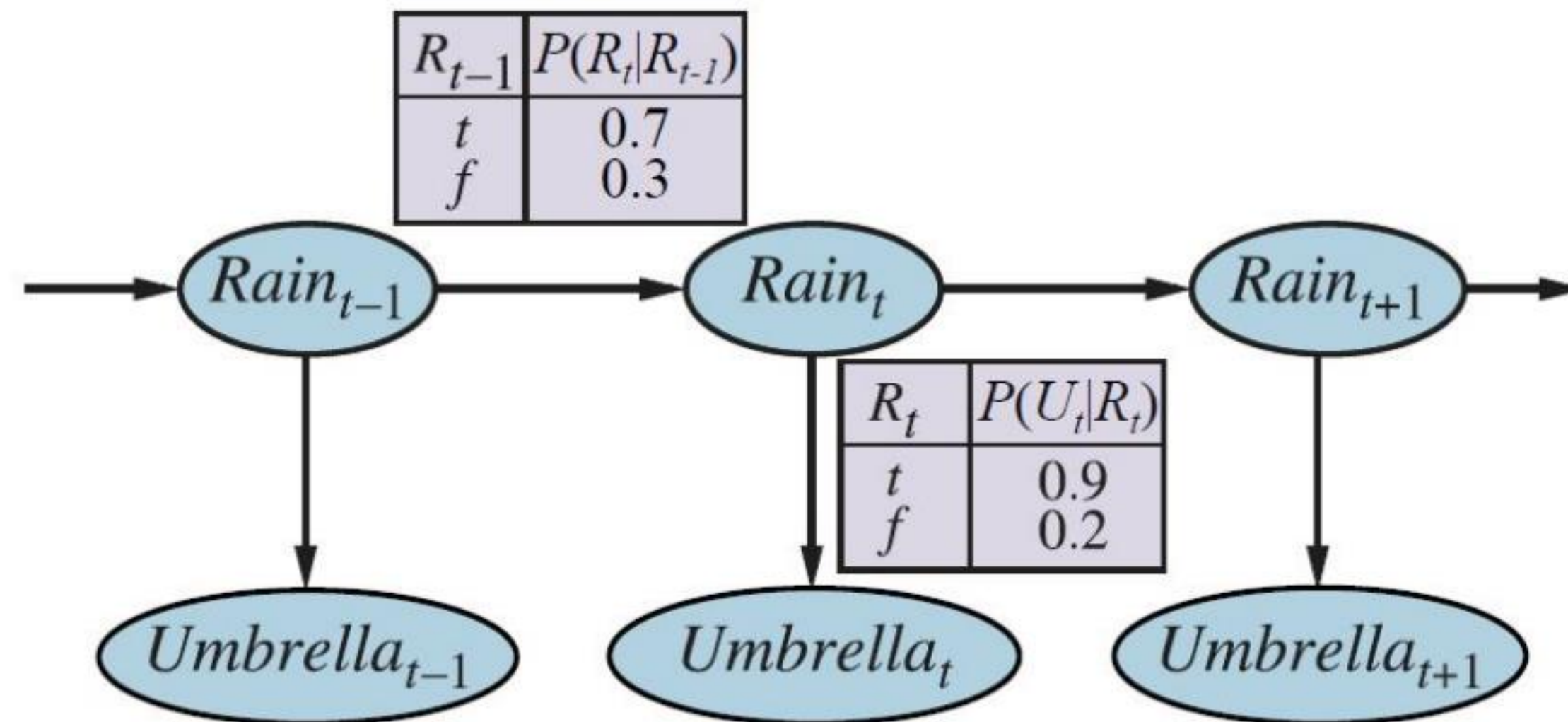
which is indeed very close to the exact probability!



Filtering: umbrella world example

Imagine to be imprisoned in a basement without windows, you only see whether the guard brings an umbrella or not

- First-order Markov process – the probability of rain is assumed to depend only on whether it rained the previous day



Bayesian network structure and conditional distributions describing the umbrella world: the transition model $P(Rain_t|Rain_{t-1})$ is
And the sensor model is $P(Umbrella_t|Rain_t)$

See the resolution in the excel

Pomegranate



is a Python package that implements fast and **flexible probabilistic models** ranging from individual probability distributions to compositional models such as **Bayesian networks** and hidden **Markov models**.

The core philosophy behind pomegranate is that all **probabilistic models can be viewed as a probability distribution** in that they all yield probability estimates for samples and can be **updated given samples and their associated weights**.

We need to install it using: `pip install pomegranate`

<https://pomegranate.readthedocs.io/en/latest/>

Pomegranate for umbrella world

Then you can run the following version of the umbrella model.

pomegranate **can only solve Bayesian networks (not Dynamic Bayesian Networks)**, so we have to unroll the whole example to the depth that we want.

```
!pip install pomegranate
from pomegranate import *

# Variables are RainN and UmbrellaN+1 for N = 0, 1, ...
# We have a prior for Rain0, two values 'y'es and 'n'o:
Rain0 = DiscreteDistribution({'y': 0.5, 'n': 0.5})

# Transition model
#
# Conditional distribution relating RainN and RainN+1. Notation for
# the conditional probability table is:
#
# [ 'RainN', 'RainN+1', <probability> ]
#
# for the conditional value P(Sprinkler|Cloudy). Note that we have to
# repeat the transition model for each pair of states
Rain1 = ConditionalProbabilityTable(
    [['y', 'y', 0.7],
     ['y', 'n', 0.3],
     ['n', 'y', 0.3],
     ['n', 'n', 0.7]], [Rain0])

Rain2 = ConditionalProbabilityTable(
    [['y', 'y', 0.7],
     ['y', 'n', 0.3],
     ['n', 'y', 0.3],
     ['n', 'n', 0.7]], [Rain1])
```

A discrete distribution, made up of characters and their probabilities, assuming that these probabilities will sum to 1.0.

<https://pomegranate.readthedocs.io/en/latest/Distributions.html?highlight=DiscreteDistribution#pomegranate.distributions.DiscreteDistribution>

Pomegranate for umbrella world

Then you can run the following version of the umbrella model.

pomegranate **can only solve Bayesian networks (not Dynamic Bayesian Networks)**, so we have to unroll the whole example to the depth that we want.

```
!pip install pomegranate
from pomegranate import *

# Variables are RainN and UmbrellaN+1 for N = 0, 1, ...
# We have a prior for Rain0, two values 'y'es and 'n'o:
Rain0 = DiscreteDistribution({'y': 0.5, 'n': 0.5})

# Transition model
#
# Conditional distribution relating RainN and RainN+1. Notation for
# the conditional probability table is:
#
# [ 'RainN', 'RainN+1', <probability>]
#
# for the conditional value P(Sprinkler|Cloudy). Note that we have to
# repeat the transition model for each pair of states
Rain1 = ConditionalProbabilityTable(
    [['y', 'y', 0.7],
     ['y', 'n', 0.3],
     ['n', 'y', 0.3],
     ['n', 'n', 0.7]], [Rain0])

Rain2 = ConditionalProbabilityTable(
    [['y', 'y', 0.7],
     ['y', 'n', 0.3],
     ['n', 'y', 0.3],
     ['n', 'n', 0.7]], [Rain1])
```

A conditional probability table, which is dependent on values from at least one previous distribution but up to as many as you want to encode for.

<https://pomegranate.readthedocs.io/en/latest/Distributions.html#pomegranate.distributions.ConditionalProbabilityTable>

Pomegranate for umbrella world

Then you can run the following version of the umbrella model.

pomegranate **can only solve Bayesian networks (not Dynamic Bayesian Networks)**, so we have to unroll the whole example to the depth that we want.

```
# Sensor model
#
# Conditional distribution relating Rain and Umbrella:
#
# [ 'Umbrella', 'Rain', <probability>]
#
# for the conditional value P(Sprinkler|Cloudy). Values for Umbrella are 'y'es and 'n'o.
# Again we have to enter the table for each day.
Umbrella1 = ConditionalProbabilityTable(
    [['y', 'y', 0.9],
     ['y', 'n', 0.1],
     ['n', 'y', 0.2],
     ['n', 'n', 0.8]], [Rain1])

Umbrella2 = ConditionalProbabilityTable(
    [['y', 'y', 0.9],
     ['y', 'n', 0.1],
     ['n', 'y', 0.2],
     ['n', 'n', 0.8]], [Rain2])
```

A conditional probability table, which is dependent on values from at least one previous distribution but up to as many as you want to encode for.

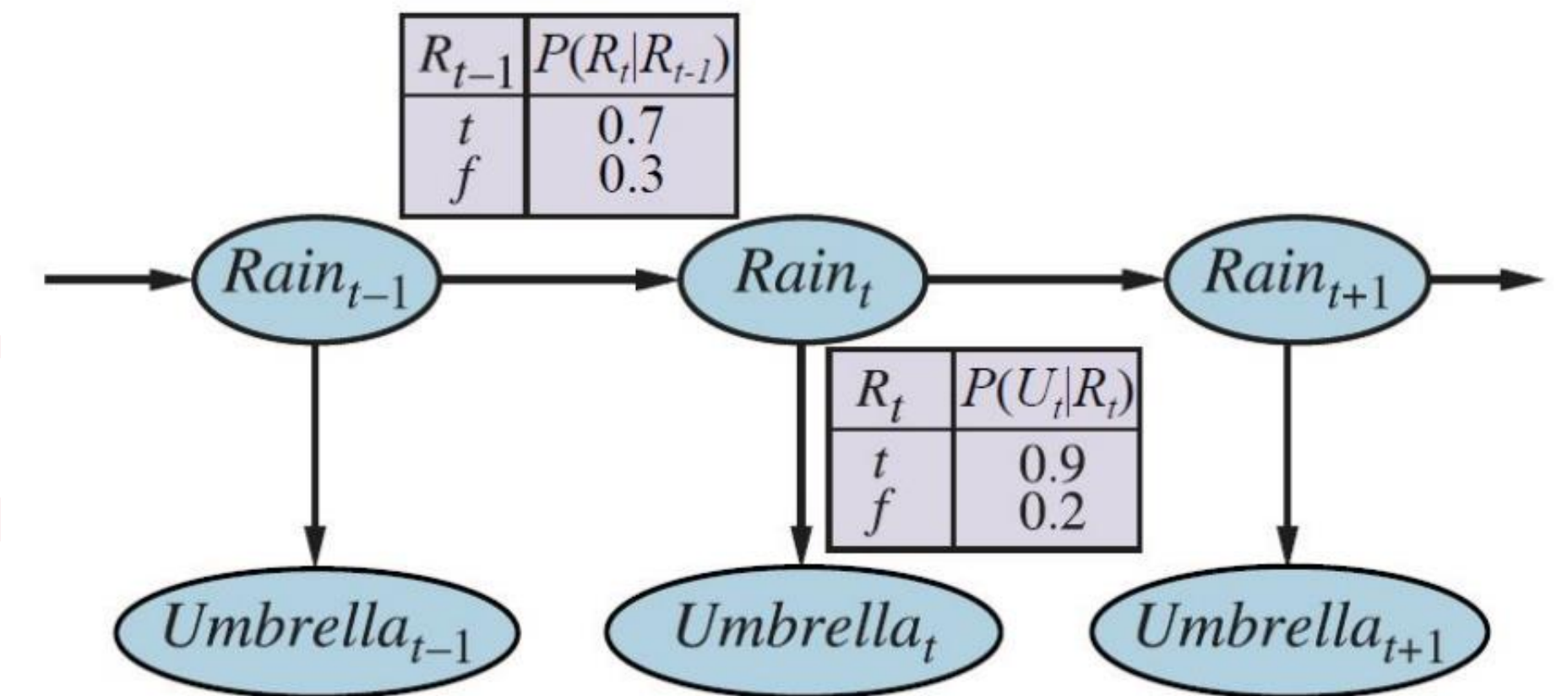
<https://pomegranate.readthedocs.io/en/latest/Distributions.html#pomegranate.distributions.ConditionalProbabilityTable>

Pomegranate for umbrella world

Then you can run the following version of the umbrella model.

pomegranate **can only solve Bayesian networks (not Dynamic Bayesian Networks)**, so we have to unroll the whole example to the depth that we want.

```
#  
# The whole network has five nodes:  
s1 = Node(Rain0, name="Rain0")  
s2 = Node(Rain1, name="Rain1")  
s3 = Node(Umbrella1, name="Umbrella1")  
s4 = Node(Rain2, name="Rain2")  
s5 = Node(Umbrella2, name="Umbrella2")  
# Create a network that includes nodes and edges between them:  
model = BayesianNetwork("Umbrella Network")  
model.add_states(s1, s2, s3, s4, s5)  
model.add_edge(s1, s2)  
model.add_edge(s2, s3)  
model.add_edge(s2, s4)  
model.add_edge(s4, s5)  
# Fix the model structure  
model.bake()
```



<https://pomegranate.readthedocs.io/en/latest/BayesianNetwork.html>

Pomegranate for umbrella world

Then you can run the following version of the umbrella model.

pomegranate **can only solve Bayesian networks (not Dynamic Bayesian Networks)**, so we have to unroll the whole example to the depth that we want.

```
#
# The whole network has five nodes:
s1 = Node(Rain0, name="Rain0")
s2 = Node(Rain1, name="Rain1")
s3 = Node(Umbrella1, name="Umbrella1")
s4 = Node(Rain2, name="Rain2")
s5 = Node(Umbrella2, name="Umbrella2")
# Create a network that includes nodes and edges between them:
model = BayesianNetwork("Umbrella Network")
model.add_states(s1, s2, s3, s4, s5)
model.add_edge(s1, s2)
model.add_edge(s2, s3)
model.add_edge(s2, s4)
model.add_edge(s4, s5)
# Fix the model structure
model.bake()
```

Finalize the topology of the model.
Assign a numerical index to every state
and create the underlying arrays
corresponding to the states and edges
between the states. **This method must
be called before any of the probability-
calculating methods.** This includes
converting conditional probability tables
into joint probability tables and creating a
list of both marginal and table nodes.

<https://pomegranate.readthedocs.io/en/latest/BayesianNetwork.html>

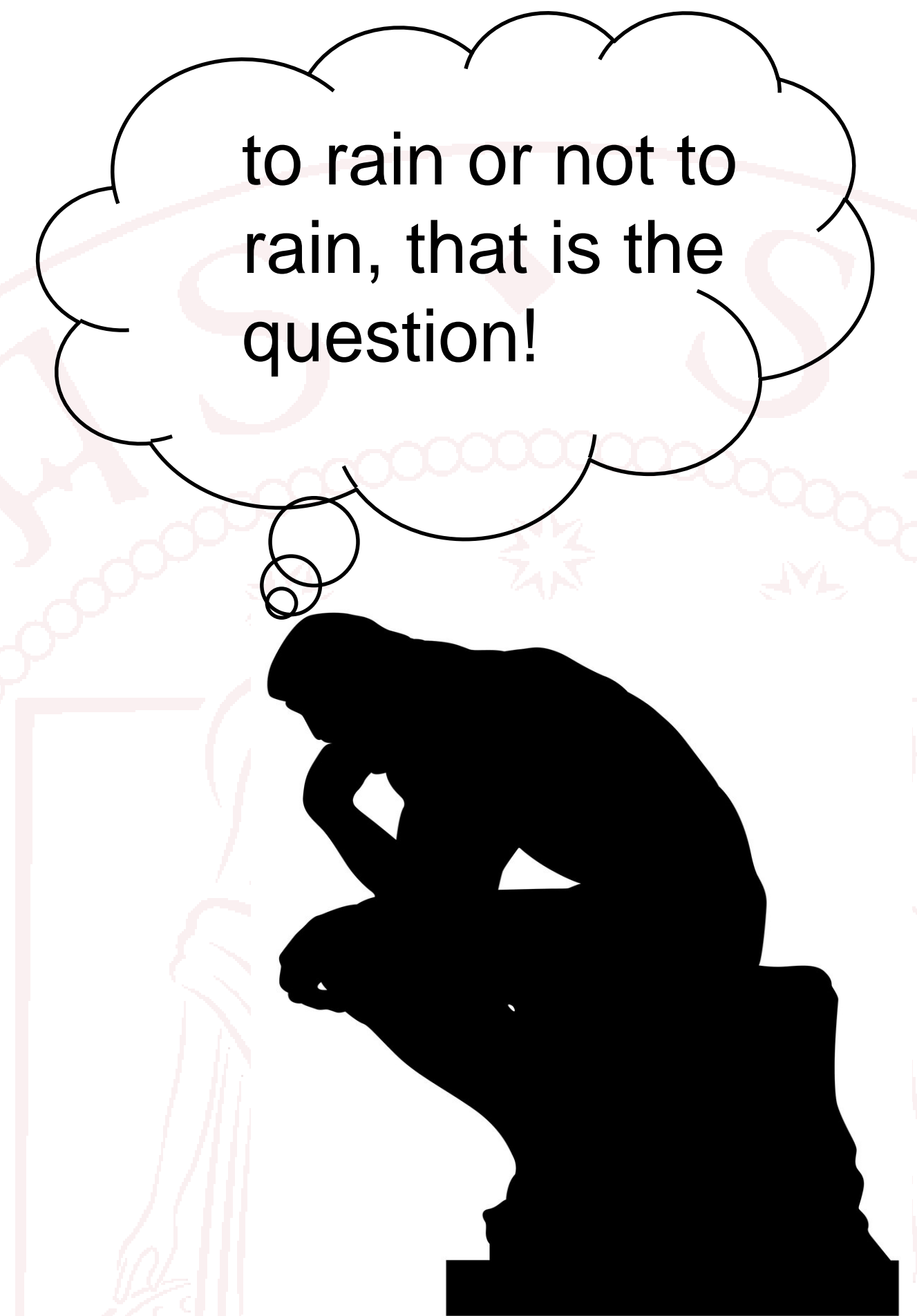
Pomegranate for umbrella world

Now that we have the model entered, we can ask it questions.
We can first ask it to predict the probability of rain on days 1 and 2:

```
# Do not instantiate any of the variables:  
scenario = [[None, None, None, None, None]]  
# Run the model  
results = model.predict_proba(scenario)  
# Ask for the probability of rain on Day 1:  
print(results[0][1].items())  
# And the probability of rain on Day 2:  
print(results[0][3].items())
```

```
((('y', 0.5000000000000000001), ('n', 0.4999999999999999999))  
((('y', 0.5000000000000000001), ('n', 0.4999999999999999999))
```

The reason that we ask for elements 1 and 3 of the datastructure results is because **they are elements 1 and 3 of model.add_states(s1, s2, s3, s4, s5)**



Pomegranate for umbrella world

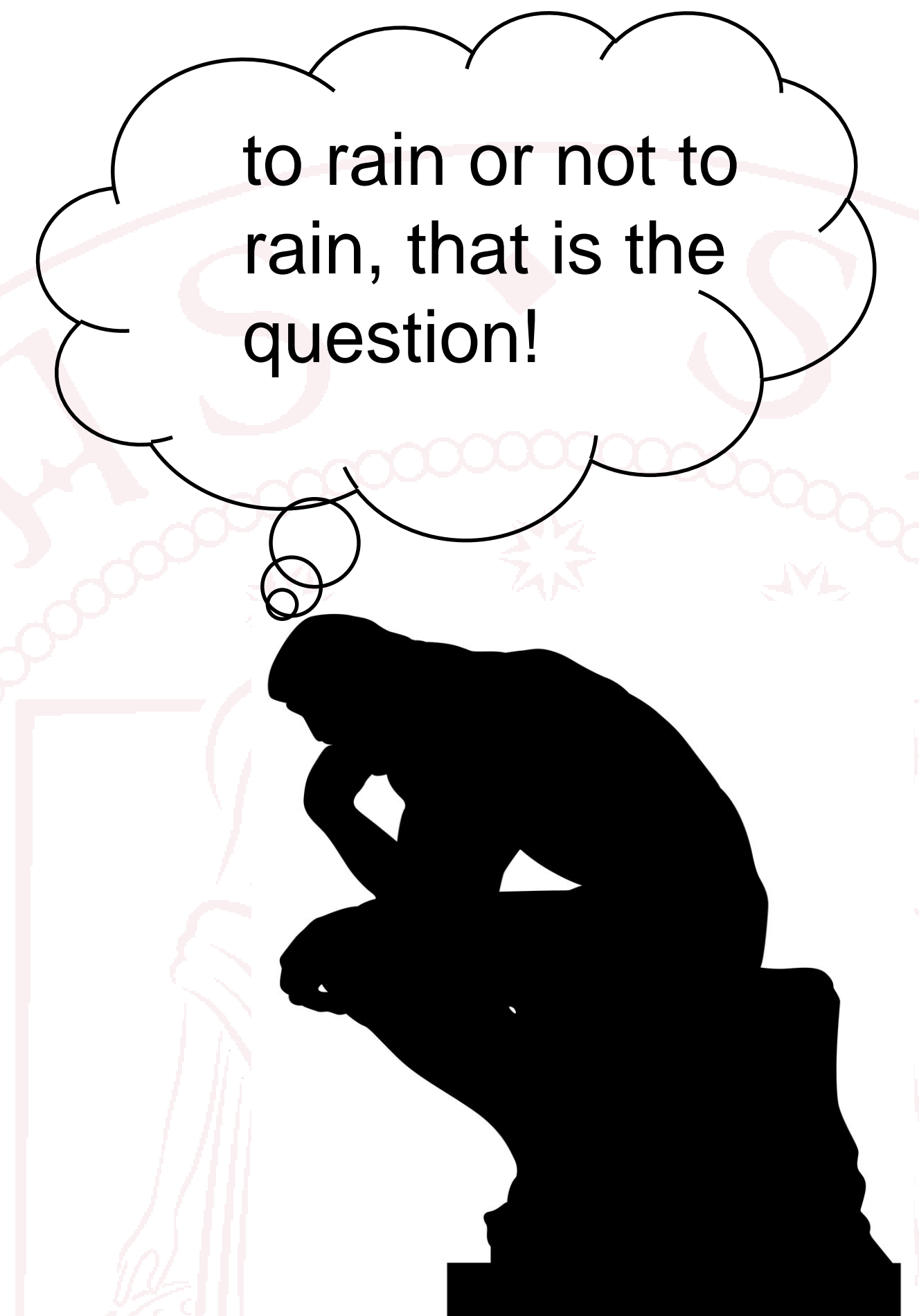
Now that we have the model entered, we can ask it questions.
We can first ask it to predict the probability of rain on days 1 and 2:

```
# Do not instantiate any of the variables:
scenario = [[None, None, None, None, None]]
# Run the model
results = model.predict_proba(scenario)
# Ask for the probability of rain on Day 1:
print(results[0][1].items())
# And the probability of rain on Day 2:
print(results[0][3].items())
```

```
((('y', 0.5000000000000000001), ('n', 0.4999999999999999999)))
((('y', 0.5000000000000000001), ('n', 0.4999999999999999999)))
```

So both Day 1 and Day 2 have a probability 0.5 of being rainy before we see any umbrellas.

<https://pomegranate.readthedocs.io/en/latest/BayesianNetwork.html>



Pomegranate for umbrella world

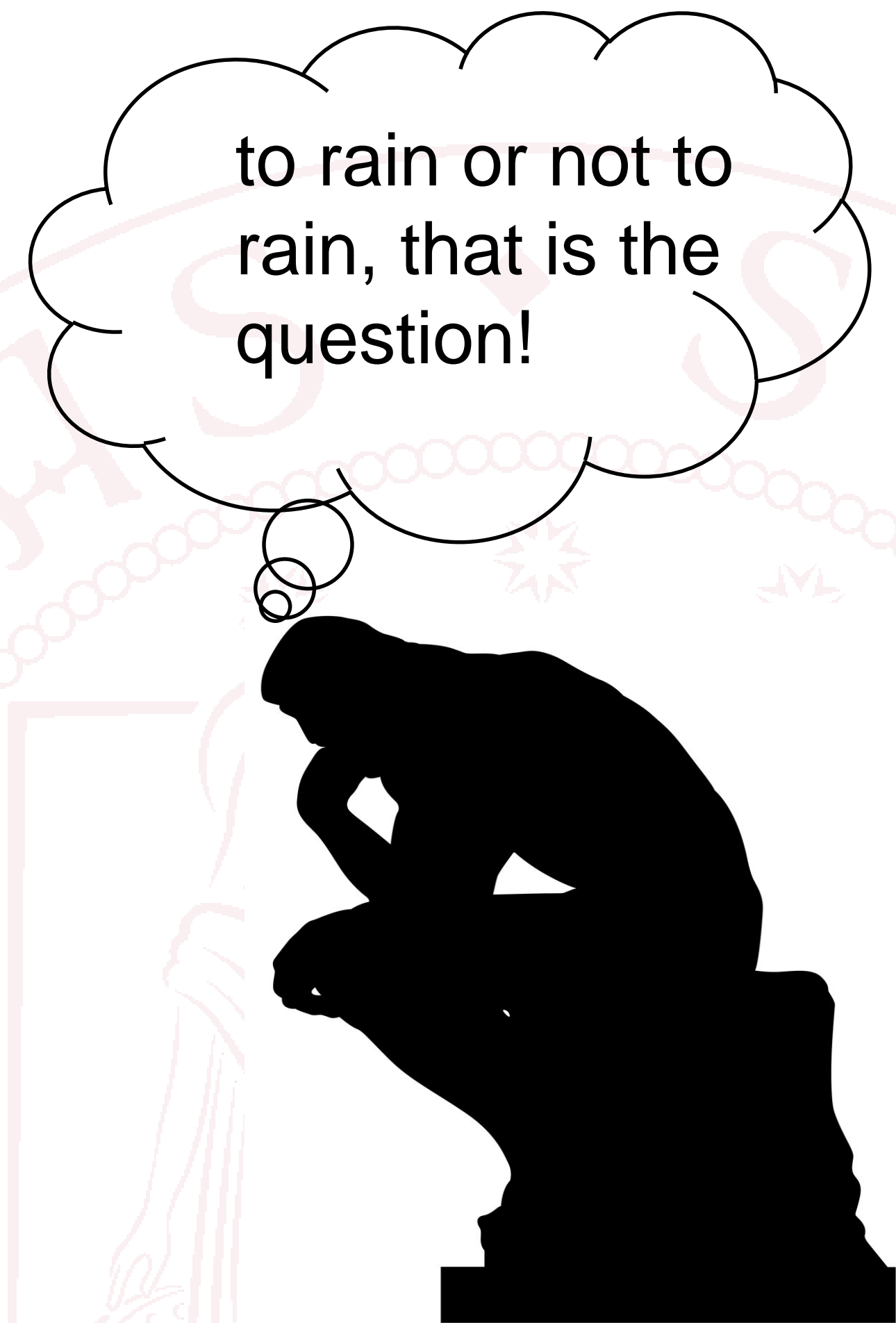
Now that we have the model entered, we can ask it questions.
We can first ask it to predict the probability of rain on days 1 and 2:

```
# Do not instantiate any of the variables:
scenario = [[None, None, None, None, None]]
# Run the model
results = model.predict_proba(scenario)
# Ask for the probability of rain on Day 1:
print(results[0][1].items())
# And the probability of rain on Day 2:
print(results[0][3].items())
```

```
((('y', 0.5000000000000000001), ('n', 0.4999999999999999999)))
((('y', 0.5000000000000000001), ('n', 0.4999999999999999999)))
```

In Bayesian probability terms, this tells us that we can't say anything about how likely it is to rain.

A binary variable with probability of 0.5 for both values is how we represent



<https://pomegranate.readthedocs.io/en/latest/BayesianNetwork.html>

Pomegranate for umbrella world

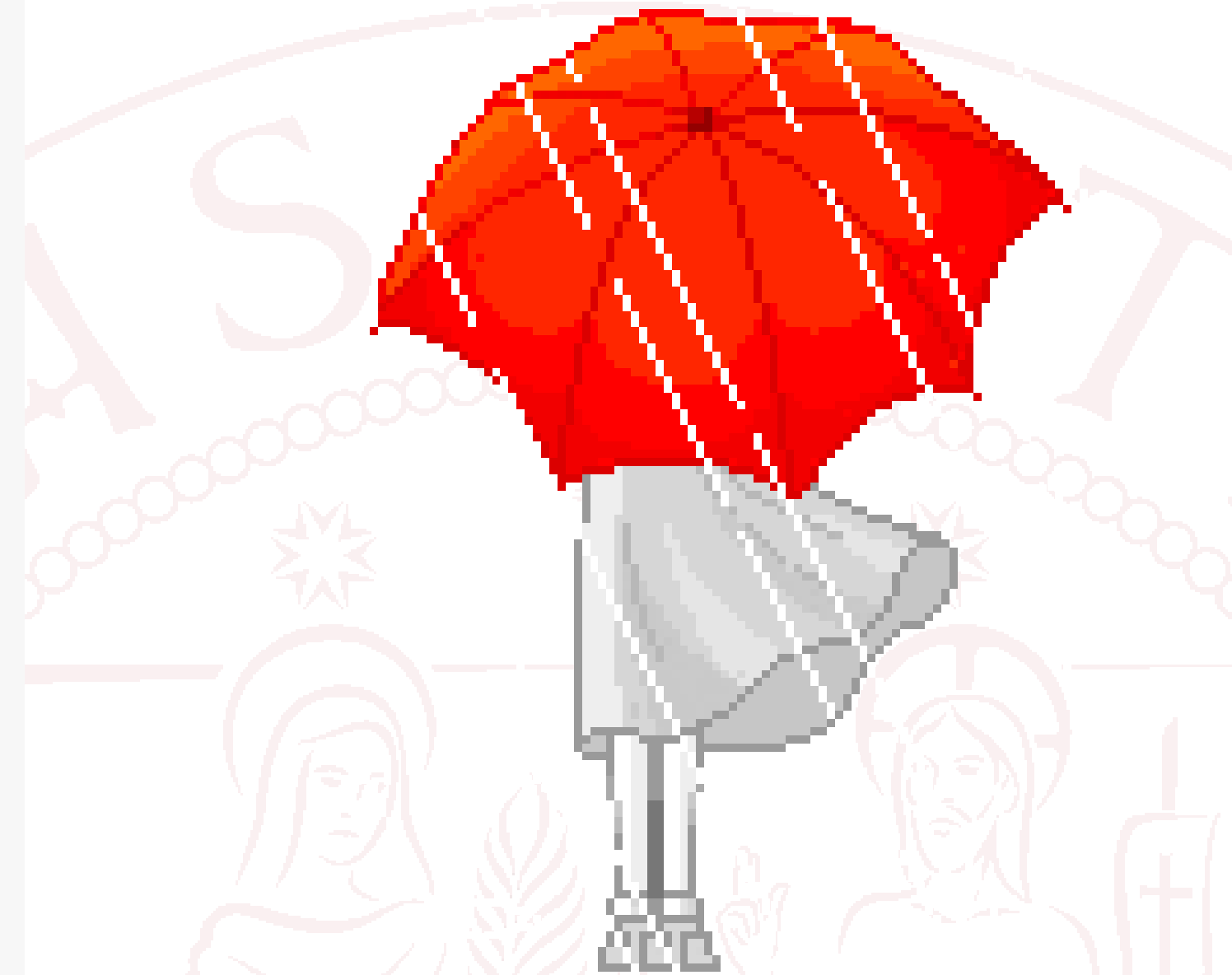
Now let's tell the model that we see an **umbrella on Day 1** and see what that gets us:

```
# Set Rain1 to 'y'
scenario = [[None, None, 'y', None, None]]
# Run the model
results = model.predict_proba(scenario)
# Ask for the probability of rain on Day 1:
print(results[0][1].items())
# And the probability of rain on Day 2:
print(results[0][3].items())
```

```
(( 'y', 0.8181818181818179), ('n', 0.1818181818181821))
(( 'y', 0.6272727272727271), ('n', 0.3727272727272729))
```

So it has filtered the probability of rain for Day 1,
and also predicted the probability for Day 2 as well.

<https://pomegranate.readthedocs.io/en/latest/BayesianNetwork.html>



Pomegranate for umbrella world

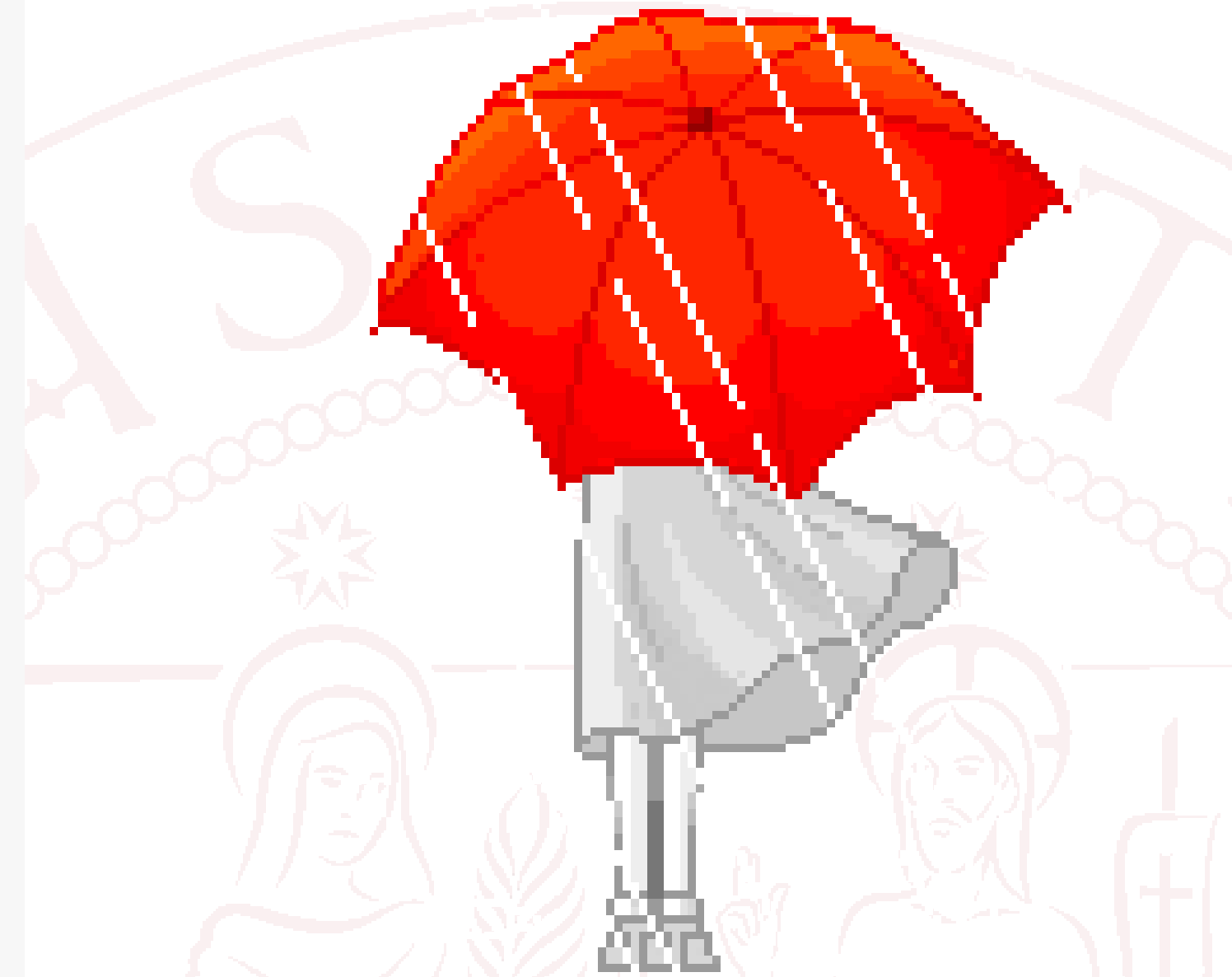
Now let's tell the model that we see an **umbrella on Day 1** and see what that gets us:

```
# Set Rain1 to 'y'
scenario = [[None, None, 'y', None, None]]
# Run the model
results = model.predict_proba(scenario)
# Ask for the probability of rain on Day 1:
print(results[0][1].items())
# And the probability of rain on Day 2:
print(results[0][3].items())
```

```
((('y', 0.8181818181818179), ('n', 0.1818181818181821)))
((('y', 0.6272727272727271), ('n', 0.3727272727272729)))
```

That is because pomegranate **propagates all updates** through the whole model/network.

<https://pomegranate.readthedocs.io/en/latest/BayesianNetwork.html>



Pomegranate for umbrella world

That is because pomegranate **propagates all updates** through the whole model/network.

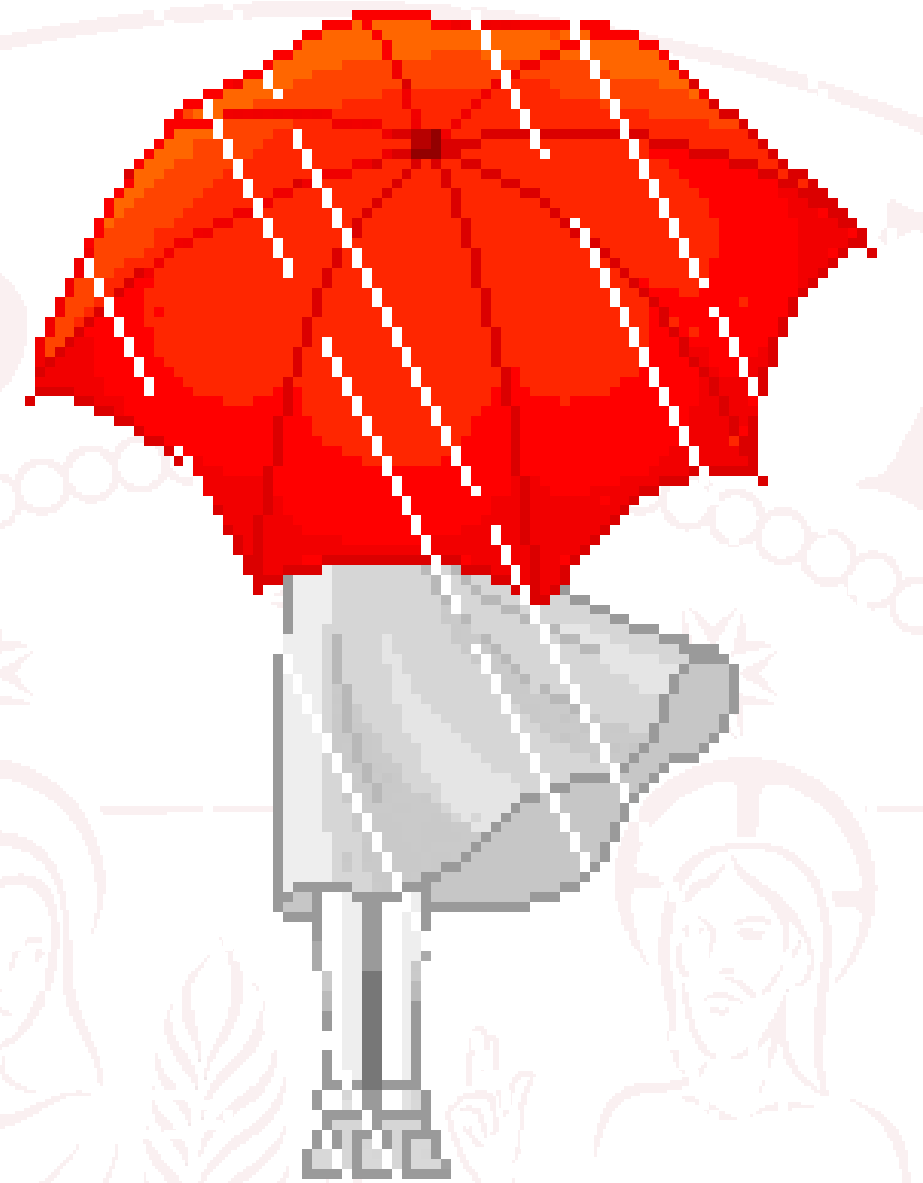
It has, for example also **computed the probability of rain on Day 0** (that it rained on Day 0 even though we said nothing about the rain that day):

```
# Ask for the probability of rain on Day 0:  
print(results[0][0].items())
```

```
(( 'y', 0.6272727272727271), ( 'n', 0.37272727272727273))
```

This is what we call the **smoothed probability** of rain on Day 0.

<https://pomegranate.readthedocs.io/en/latest/BayesianNetwork.html>



Pomegranate for umbrella world

Now let's tell pomegranate about rain on Day 3, so we need to add information about Day3:

```
# Conditional probability table
Rain3 = ConditionalProbabilityTable(
    [['y', 'y', 0.7],
     ['y', 'n', 0.3],
     ['n', 'y', 0.3],
     ['n', 'n', 0.7]], [Rain2])
```

```
# Node
s6 = Node(Rain3, name="Rain3")
# State
model.add_states(s6)
# Edge
model.add_edge(s4, s6)
# Fix the model structure
model.bake()
```

Note that we only call `model.bake()` once the last elements are entered.

Pomegranate for umbrella world

Now that we have the model entered, we can ask it questions.

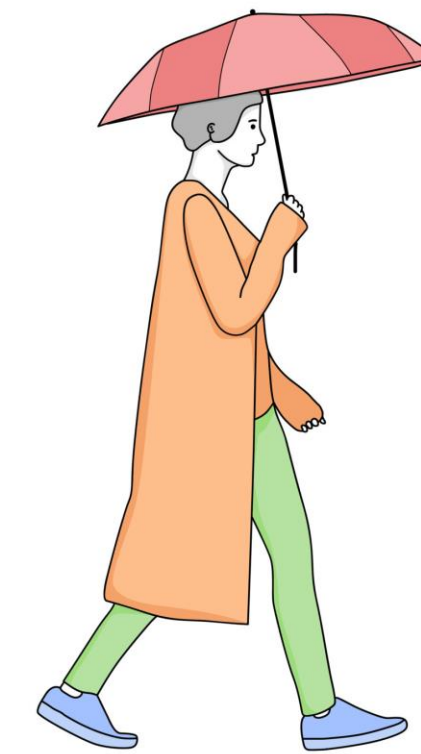
We tell the model that we saw Umbrellas on Days 1 and 2:

```
# Umbrellas on Day 1 and 2:
scenario = [[None, None, 'y', None, 'y', None]]
# Run the model
results = model.predict_proba(scenario)
# Ask for the probability of rain on Day 1:
print(results[0][1].items())
# And the probability of rain on Day 2:
print(results[0][3].items())

(('y', 0.8833570412517776), ('n', 0.11664295874822228))
(('y', 0.8833570412517776), ('n', 0.1166429587482225))
```



Day1



Day2

Pomegranate for umbrella world

Now that we have the model entered, we can ask it questions.

We tell the model that we saw Umbrellas on Days 1 and 2:

```
# Umbrellas on Day 1 and 2:
scenario = [[None, None, 'y', None, 'y', None]]
# Run the model
results = model.predict_proba(scenario)
# Ask for the probability of rain on Day 1:
print(results[0][1].items())
# And the probability of rain on Day 2:
print(results[0][3].items())

(('y', 0.8833570412517776), ('n', 0.11664295874822228))
(('y', 0.8833570412517776), ('n', 0.1166429587482225))
```



Note that we **didn't tell pomegranate to do smoothing**. As we saw before with Day 0, it (in effect) **always runs the backwards propagation** and gives us smoothed probabilities for all days before the latest piece of evidence.

I said "in effect" because pomegranate doesn't do the computation the way we studied.

It just computes the probability of **every hidden variable** given the evidence.

<https://pomegranate.readthedocs.io/en/latest/BayesianNetwork.html>

Questions

