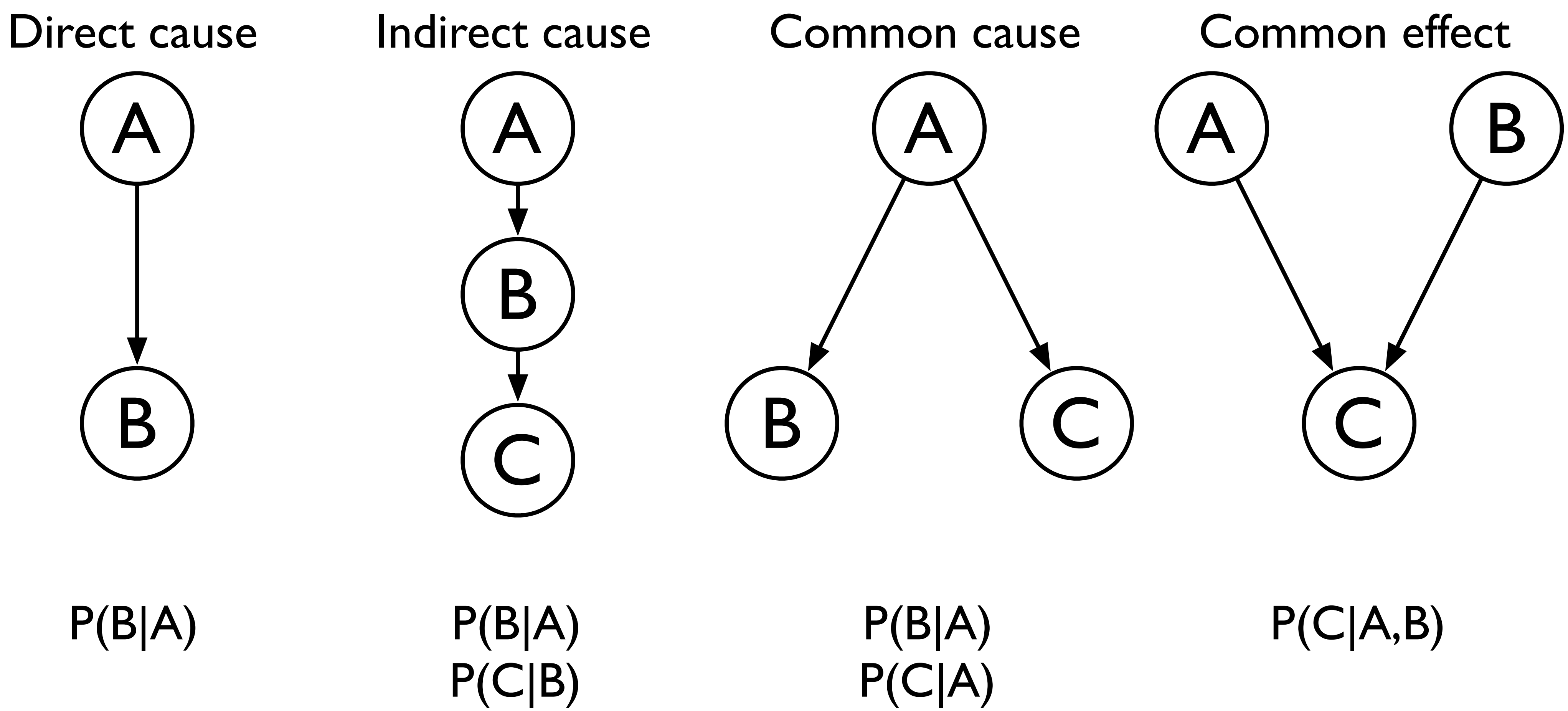
EECS 391
Intro to AI

Inference in Bayes Nets

L15 Tue Oct 31

Recap: Modeling causal relationships with belief networks

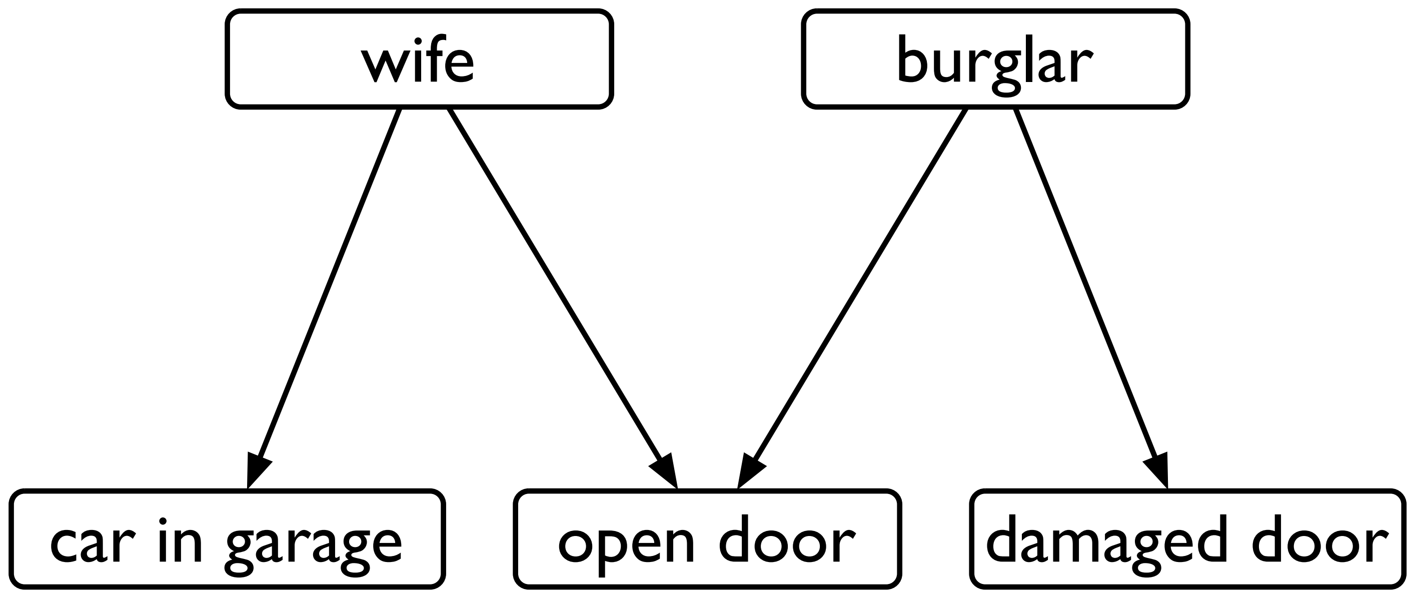


Recap (cont'd)

P(W)
0.05

W	B	P(O|W,B)
F	F	0.01
F	T	0.25
T	F	0.05
T	T	0.75

P(B)
0.001



W	P(C|W)
F	0.01
T	0.95

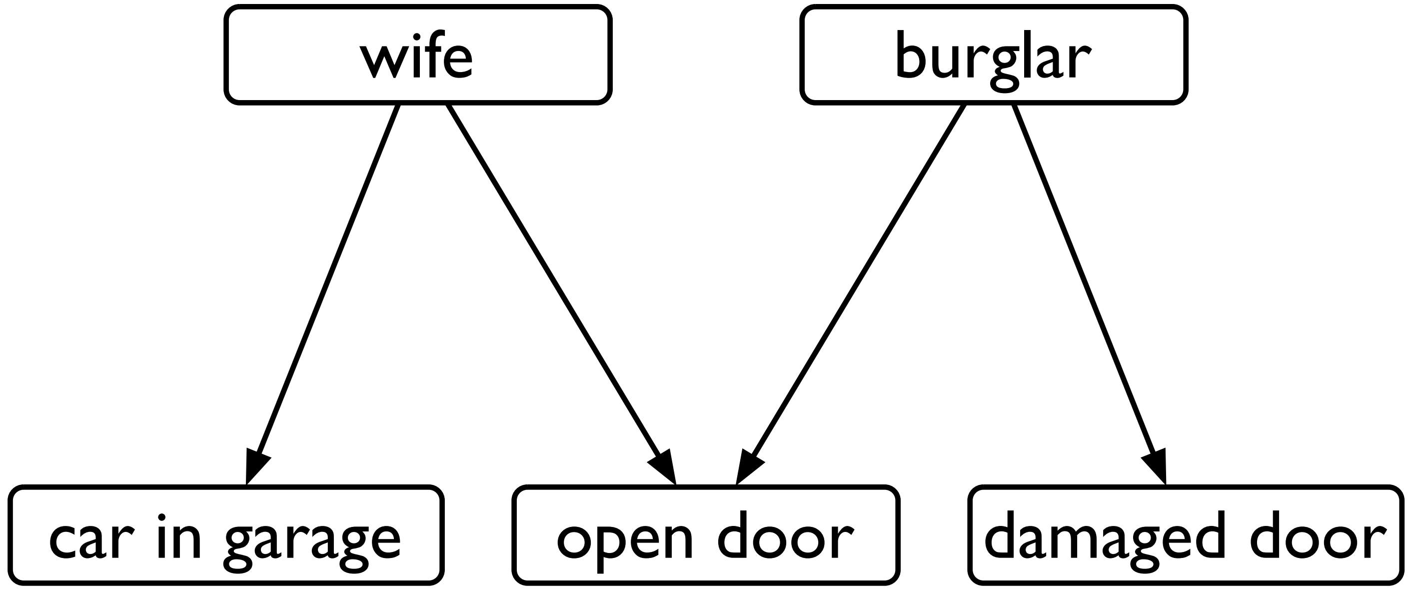
B	P(D|B)
F	0.001
T	0.5

- The structure of this model allows a simple expression for the joint probability

$$
\begin{aligned}
P(x_1, \ldots, x_n) &\equiv P(X_1 = x_1 \wedge \ldots \wedge X_n = x_n) \\
&= \prod_{i=1}^{n} P(x_i | \text{parents}(X_i))
\end{aligned}
$$

$$
\Rightarrow P(o, c, d, w, b) = P(c|w)P(o|w, b)P(d|b)P(w)P(b)
$$

Recap (cont'd): Noisy OR



- We could reduce the number of parameters with a Noisy-OR approximation

$$
\begin{aligned}
P(E_i|\text{par}(E_i)) &= P(E_i|C_1,\ldots,C_n) \\
&= 1 - \prod_i (1 - P(E_i|C_j)) \\
&= 1 - \prod_i (1 - f_{ij})
\end{aligned}
$$

W	B	P(O|W,B)
F	F	0.01
F	T	0.25
T	F	0.05
T	T	0.75

Recall how we calculated the joint probability on the burglar network:

- $P(o,w,\neg b,c,\neg d) = P(o|w,\neg b)P(c|w)P(\neg d|\neg b)P(w)P(\neg b)$

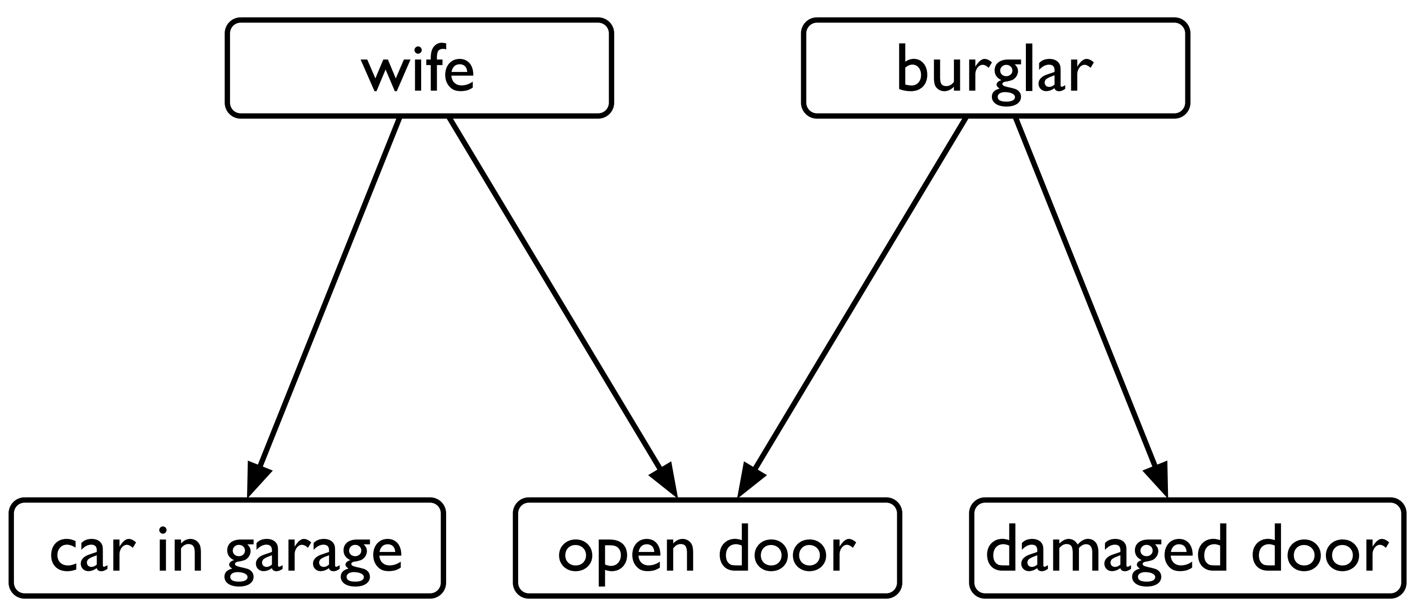
$$= \quad 0.05 \times 0.95 \times 0.999 \times 0.05 \times 0.999 = 0.0024$$

- How do we calculate $P(b|o)$, i.e. the probability of a burglar given we see the open door?
- This is not an entry in the joint distribution.

P(W)
0.05

W	B	P(O|W,B)
F	F	0.01
F	T	0.25
T	F	0.05
T	T	0.75

P(B)
0.001



W	P(C|W)
F	0.01
T	0.95

B	P(D|B)
F	0.001
T	0.5

Computing probabilities of propositions

- How do we compute P(o|b)?
  - Bayes' rule
  - marginalize joint distribution

Variable elimination on the burglary network

- As we mentioned in the last lecture, we could do straight summation:

$$
\begin{aligned}
p(b|o) &= \alpha\, p(o, w, b, c, d) \\
&= \alpha \sum_{w,c,d} p(o|w, b)p(c|w)p(d|b)p(w)p(b)
\end{aligned}
$$

- But: the number of terms in the sum is *exponential* in the non-evidence variables.
- This is bad, and we can do much better.
- We start by observing that we can pull out many terms from the summation.

Variable elimination

- When we've pulled out all the redundant terms we get:

$$p(b|o) = \alpha\, p(b) \sum_d p(d|b) \sum_w p(w)p(o|w,b) \sum_c p(c|w)$$

- We can also note the last term sums to one. In fact, every variable that is not an ancestor of a query variable or evidence variable is *irrelevant* to the query, so we get

$$p(b|o) = \alpha\, p(b) \sum_d p(d|b) \sum_w p(w)p(o|w,b)$$

  which contains far fewer terms: In general, *complexity is* ***linear*** *in the # of CPT entries.*

- This method is called *variable elimination*.
  - if # of parents is bounded, also linear in the number of nodes.
  - the expressions are evaluated in right-to-left order (bottom-up in the network)
  - intermediate results are stored
  - sums over each are done only for those expressions that depend on the variable

- Note: for multiply connected networks, variable elimination can have exponential complexity in the worst case.

Inference in Bayesian networks

- For queries in Bayesian networks, we divide variables into three classes:
  - evidence variables: $e = \{e_1, \ldots, e_m\}$ what you know
  - query variables: $x = \{x_1, \ldots, x_n\}$ what you want to know
  - non-evidence variables: $y = \{y_1, \ldots, y_l\}$ what you don't care about
- The complete set of variables in the network is $\{e \cup x \cup y\}$.
- Inferences in Bayesian networks consist of computing p(x|e), the posterior probability of the query given the evidence:

$$p(x|e) = \frac{p(x,e)}{p(e)} = \alpha\, p(x,e) = \alpha \sum_y p(x,e,y)$$

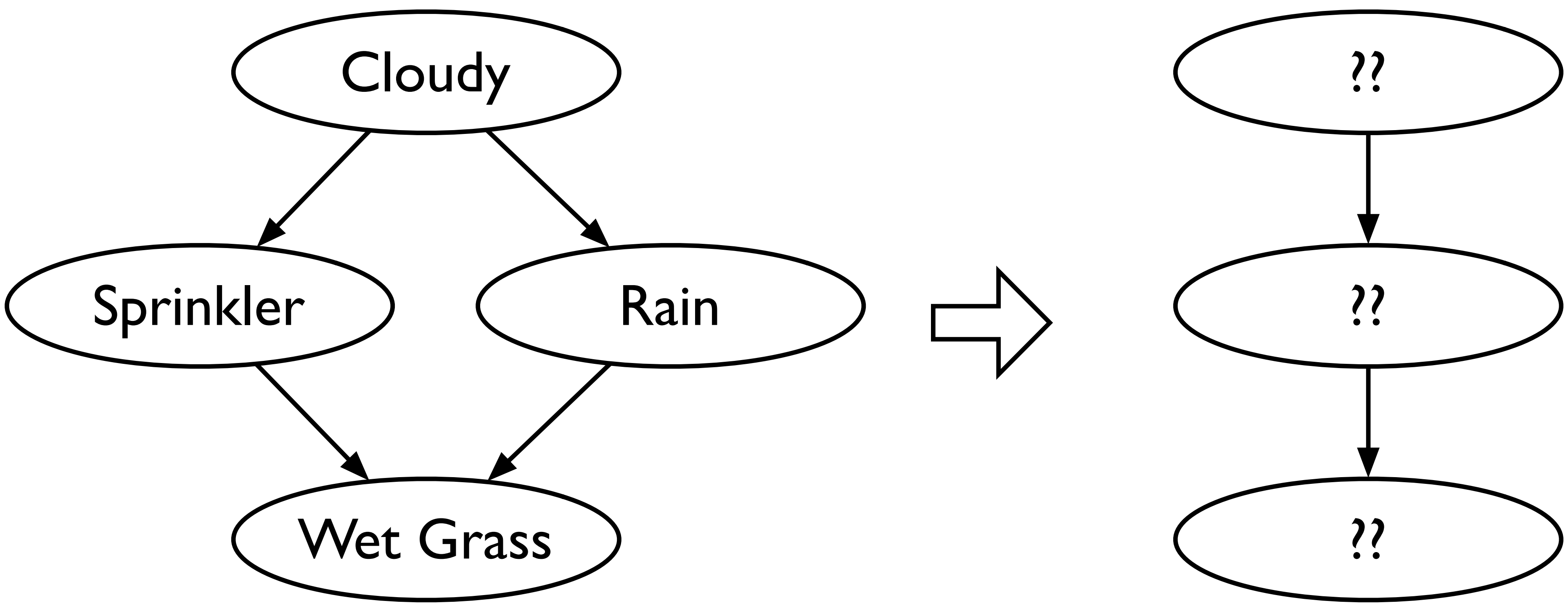
- This computes the marginal distribution p(x,e) by summing the joint over all values of y.
- Recall that the joint distribution is defined by the product of the conditional pdfs:

$$p(z) = \prod_{i=1} P(z_i|\text{parents}(z_i))$$

where the product is taken over all variables in the network.

Clustering algorithms

- Inference is efficient if you have a *polytree*, ie a singly connected network.
- But what if you don't?
- Idea: Convert a non-singly connected network to an equivalent singly connected network.

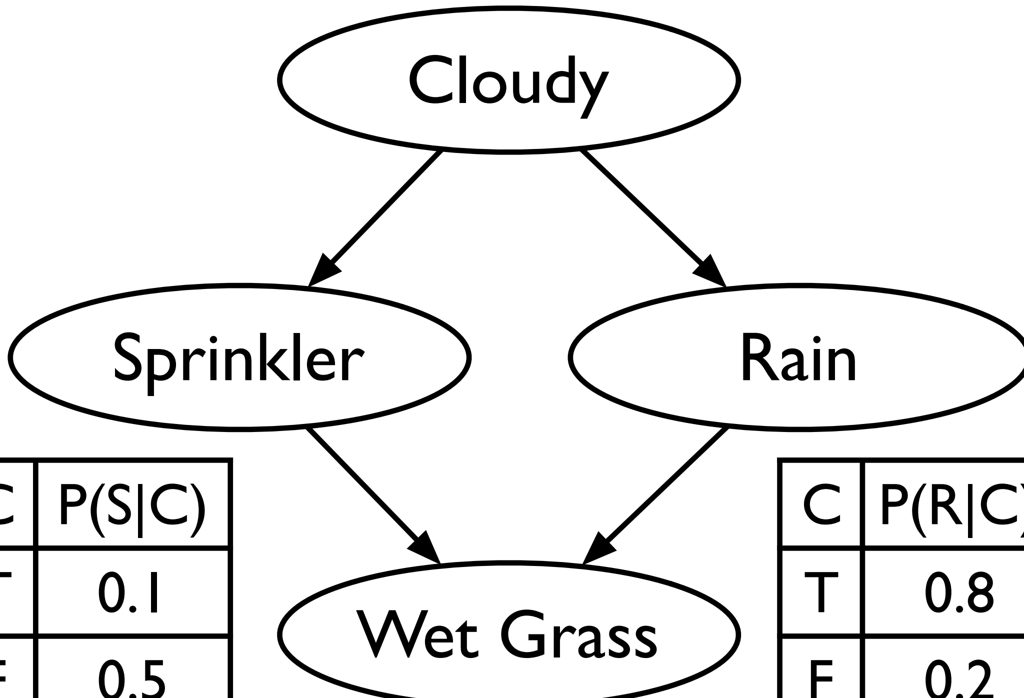


What should go into the nodes?

Clustering or join tree algorithms

Idea: merge multiply connected nodes into a single, higher-dimensional case.

P(C)
0.5

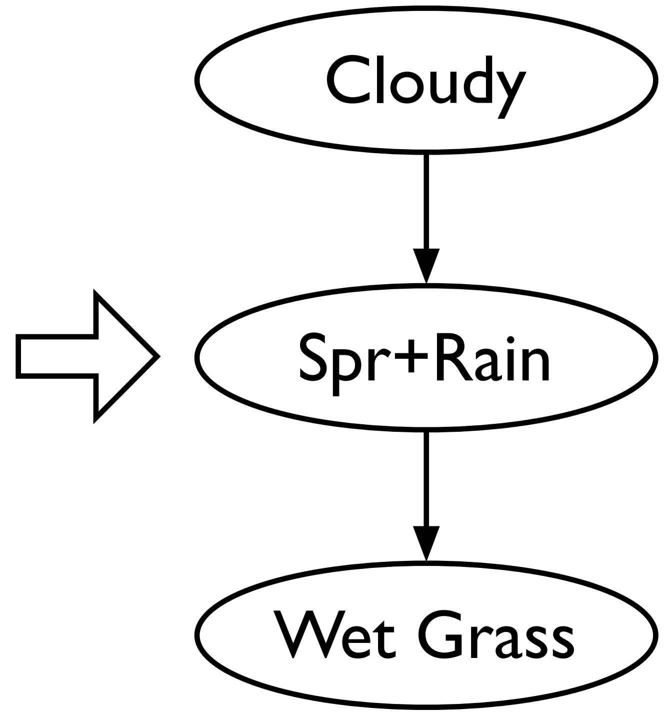


C	P(S|C)
T	0.1
F	0.5

C	P(R|C)
T	0.8
F	0.2

S	R	P(W|S,R)
T	T	0.99
T	F	0.9
F	T	0.9
F	F	0.01

P(C)
0.5



	P(S+R=x|C)			
C	TT	TF	FT	FF
T	0.08	0.02	0.72	0.18
F	0.1	0.4	0.1	0.4

S+R	P(W|S+R)
T+T	0.99
T+F	0.9
F+T	0.9
F+F	0.01

Can take exponential time to construct CPTs
But approximate algorithms usu. give reasonable solutions.

Another approach: Inference by *stochastic simulation*

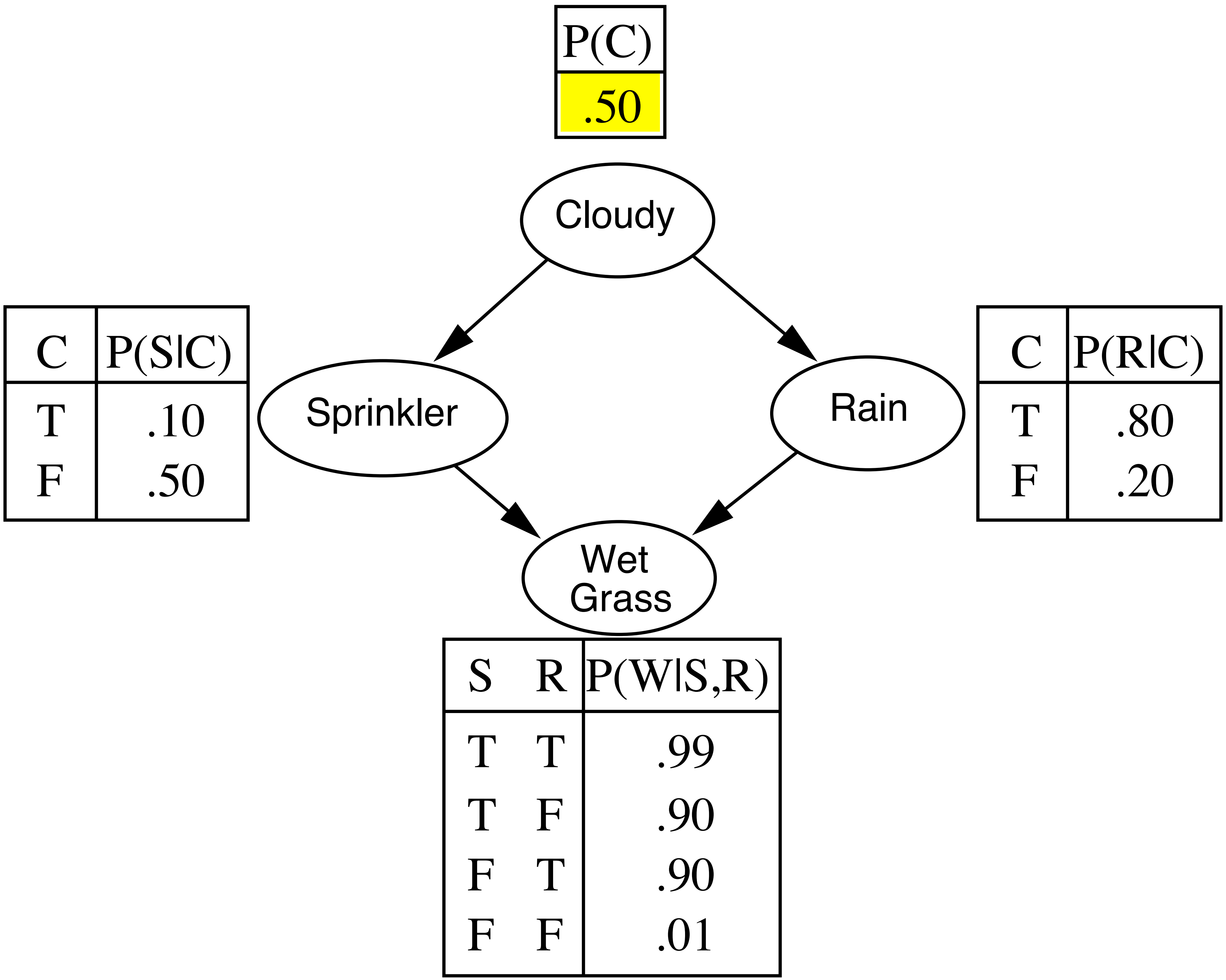
Basic idea:

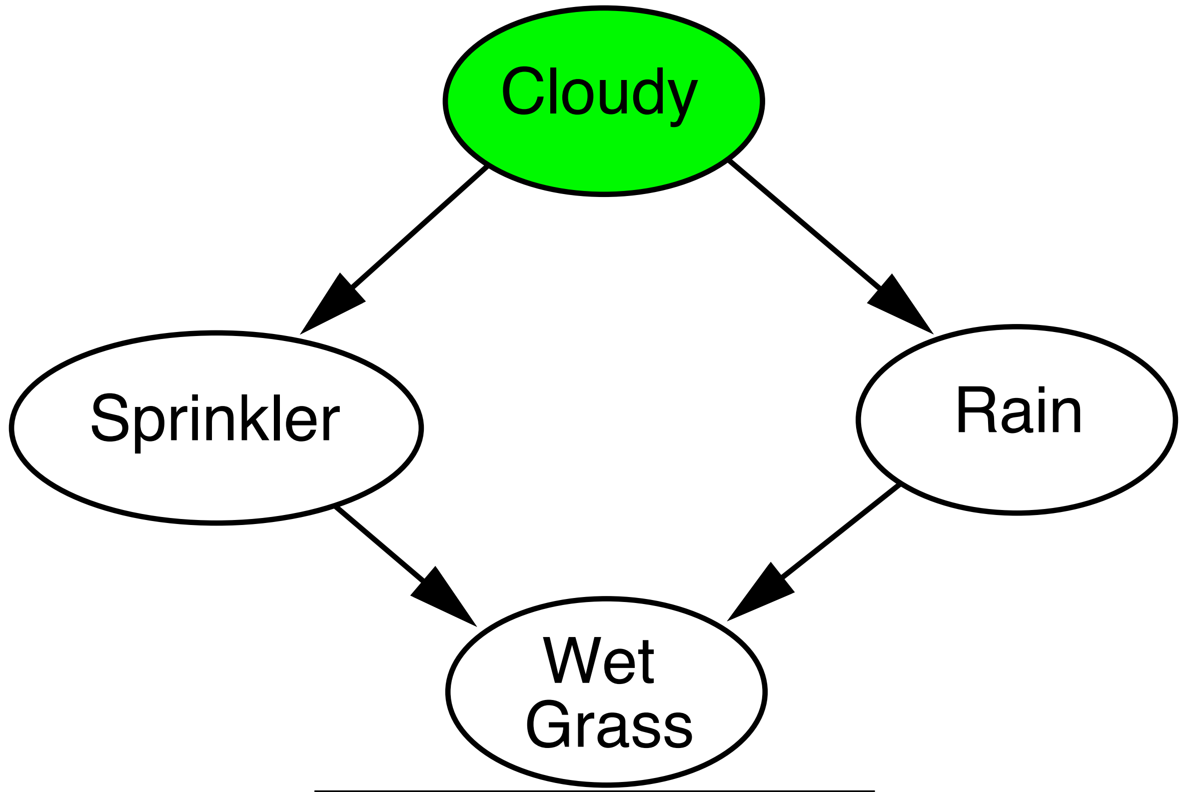
1. Draw N samples from a sampling distribution S
2. Compute an approximate posterior probability
3. Show this converges to the true probability

Sampling with no evidence (from the prior)

function PRIOR-SAMPLE(bn) **returns** an event sampled from bn
  inputs: bn, a belief network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$

  $\mathbf{x} \leftarrow$ an event with n elements
  for $i = 1$ **to** n **do**
    $x_i \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
      given the values of $Parents(X_i)$ in $\mathbf{x}$
  return $\mathbf{x}$


P(C)
.50
Cloudy
Sprinkler
Rain
Wet
Grass
C P(S|C)
T .10
F .50
C P(R|C)
T .80
F .20
S R P(W|S,R)
T T .99
T F .90
F T .90
F F .01

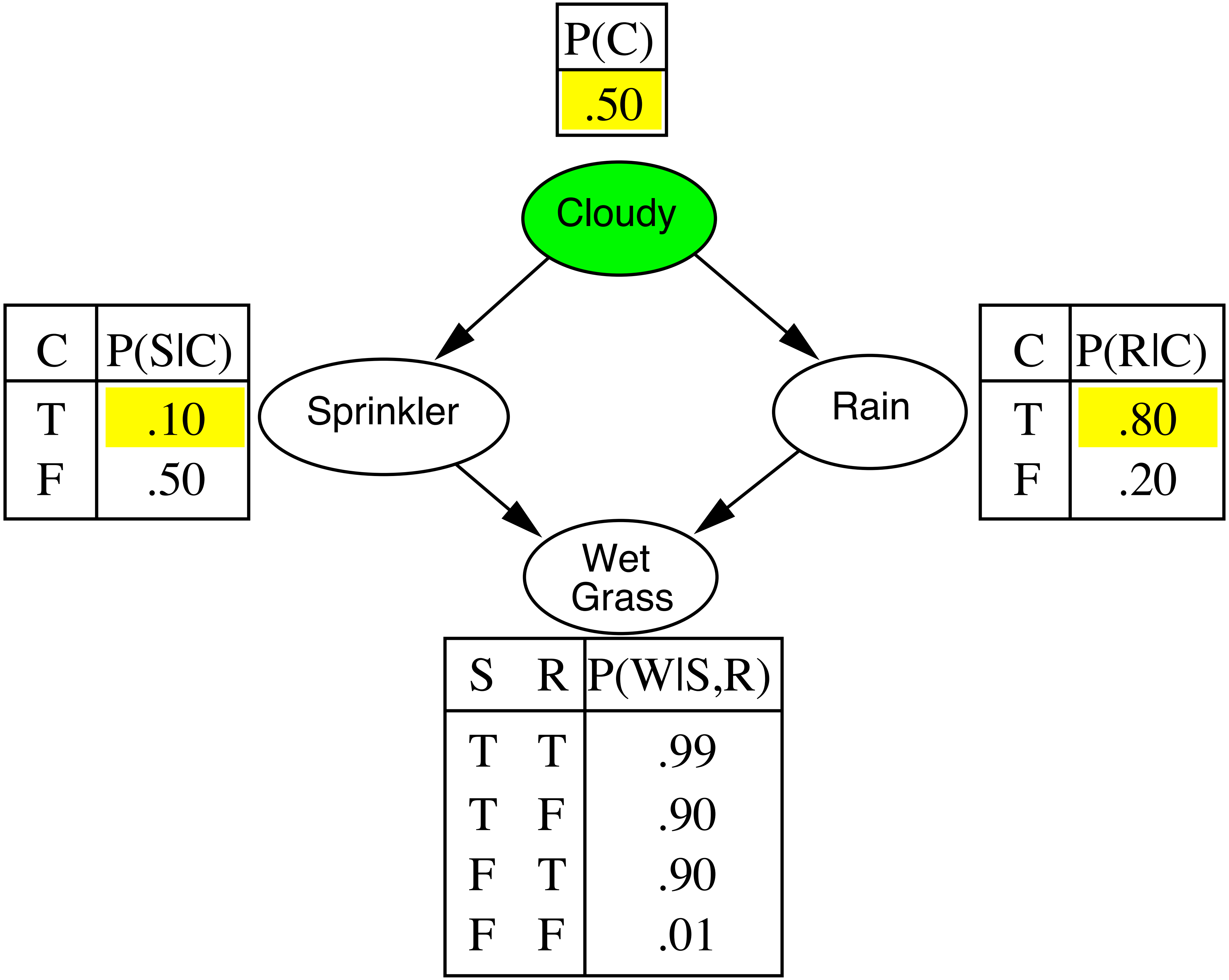


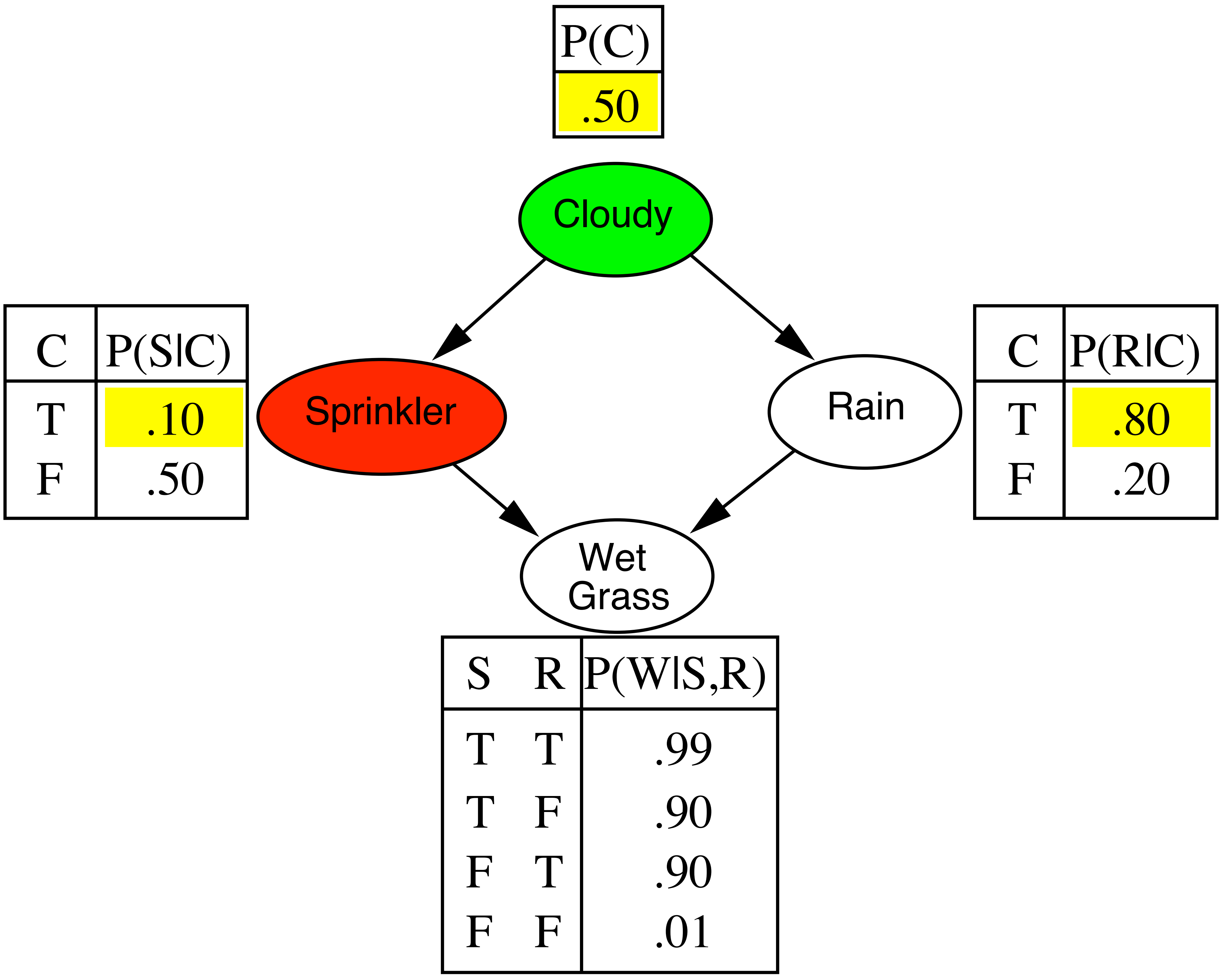
P(C)
.50

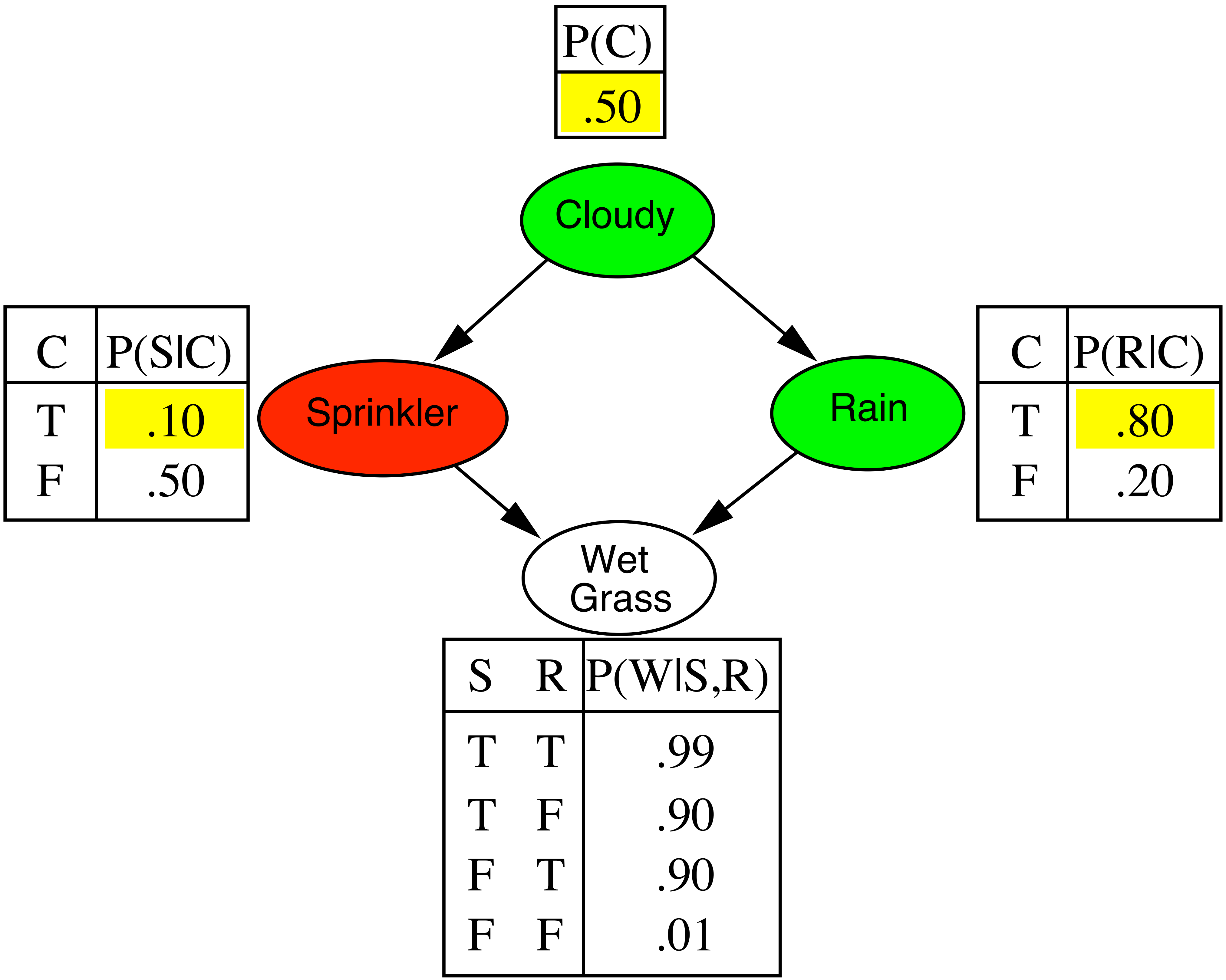
C	P(S|C)
T	.10
F	.50

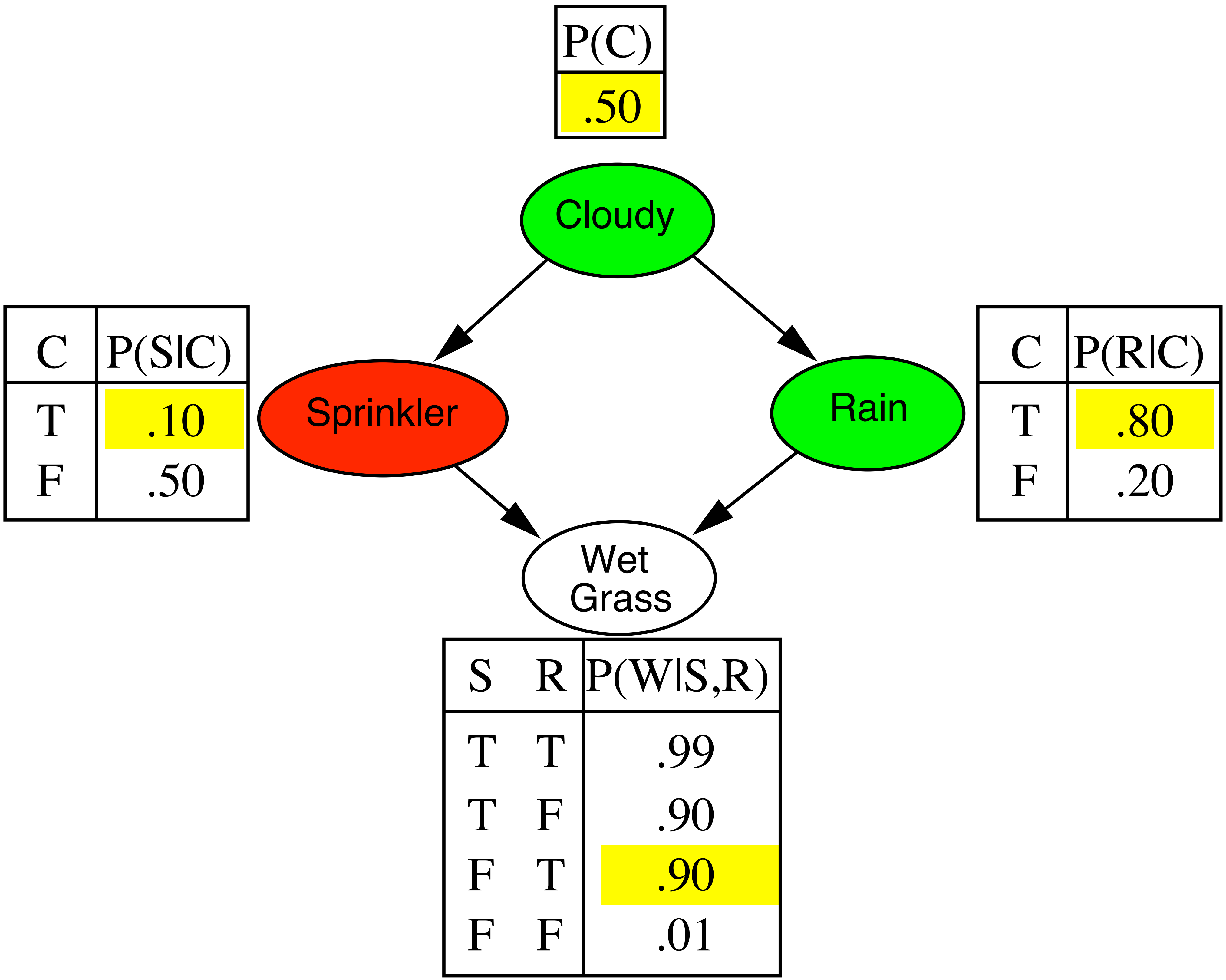
C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

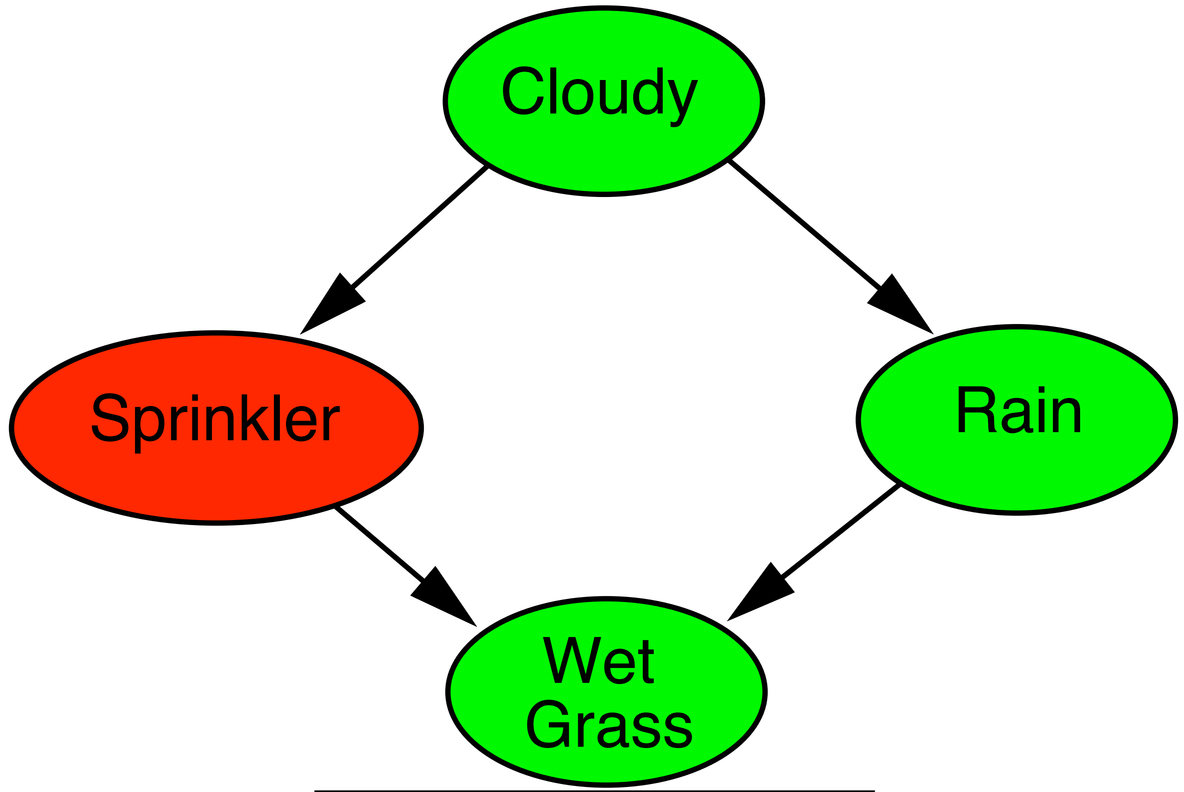

P(C)
.50
Cloudy
C P(S|C)
T .10
F .50
Sprinkler
Rain
C P(R|C)
T .80
F .20
Wet Grass
S R P(W|S,R)
T T .99
T F .90
F T .90
F F .01


P(C)
.50
Cloudy
C P(S|C)
T .10
F .50
Sprinkler
Rain
C P(R|C)
T .80
F .20
Wet Grass
S R P(W|S,R)
T T .99
T F .90
F T .90
F F .01


P(C)
.50
Cloudy
C P(S|C)
T .10
F .50
Sprinkler
Rain
C P(R|C)
T .80
F .20
Wet Grass
S R P(W|S,R)
T T .99
T F .90
F T .90
F F .01



P(C)
.50
Cloudy
C P(S|C)
T .10
F .50
Sprinkler
Rain
C P(R|C)
T .80
F .20
Wet Grass
S R P(W|S,R)
T T .99
T F .90
F T .90
F F .01



P(C)
.50

C	P(S|C)
T	.10
F	.50

C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

Keep sampling to estimate joint probabilities of interest.

What if we do have some evidence? Rejection sampling.

$\hat{\mathbf{P}}(X|\mathbf{e})$ estimated from samples agreeing with $\mathbf{e}$

function REJECTION-SAMPLING(X, $\mathbf{e}$, bn, N) **returns** an estimate of $P(X|\mathbf{e})$
  local variables: $\mathbf{N}$, a vector of counts over X, initially zero

  for $j = 1$ to N **do**
    $\mathbf{x} \leftarrow$ PRIOR-SAMPLE(bn)
    if $\mathbf{x}$ is consistent with $\mathbf{e}$ **then**
      $\mathbf{N}[x] \leftarrow \mathbf{N}[x]+1$ where x is the value of X in $\mathbf{x}$
  return NORMALIZE($\mathbf{N}[X]$)

E.g., estimate $\mathbf{P}(Rain|Sprinkler = true)$ using 100 samples
  27 samples have $Sprinkler = true$
    Of these, 8 have $Rain = true$ and 19 have $Rain = false$.

$\hat{\mathbf{P}}(Rain|Sprinkler = true) = \text{NORMALIZE}(\langle 8, 19\rangle) = \langle 0.296, 0.704\rangle$

Similar to a basic real-world empirical estimation procedure

Analysis of rejection sampling

$$
\begin{aligned}
\hat{\mathbf{P}}(X|\mathbf{e}) &= \alpha \mathbf{N}_{PS}(X,\mathbf{e}) && \text{(algorithm defn.)} \\
&= \mathbf{N}_{PS}(X,\mathbf{e})/N_{PS}(\mathbf{e}) && \text{(normalized by } N_{PS}(\mathbf{e})\text{)} \\
&\approx \mathbf{P}(X,\mathbf{e})/P(\mathbf{e}) && \text{(property of } \textsc{PriorSample}\text{)} \\
&= \mathbf{P}(X|\mathbf{e}) && \text{(defn. of conditional probability)}
\end{aligned}
$$

Hence rejection sampling returns consistent posterior estimates

Problem: hopelessly expensive if $P(\mathbf{e})$ is small

$P(\mathbf{e})$ drops off exponentially with number of evidence variables!

Approximate inference using Markov Chain Monte Carlo (MCMC)

"State" of network = current assignment to all variables.

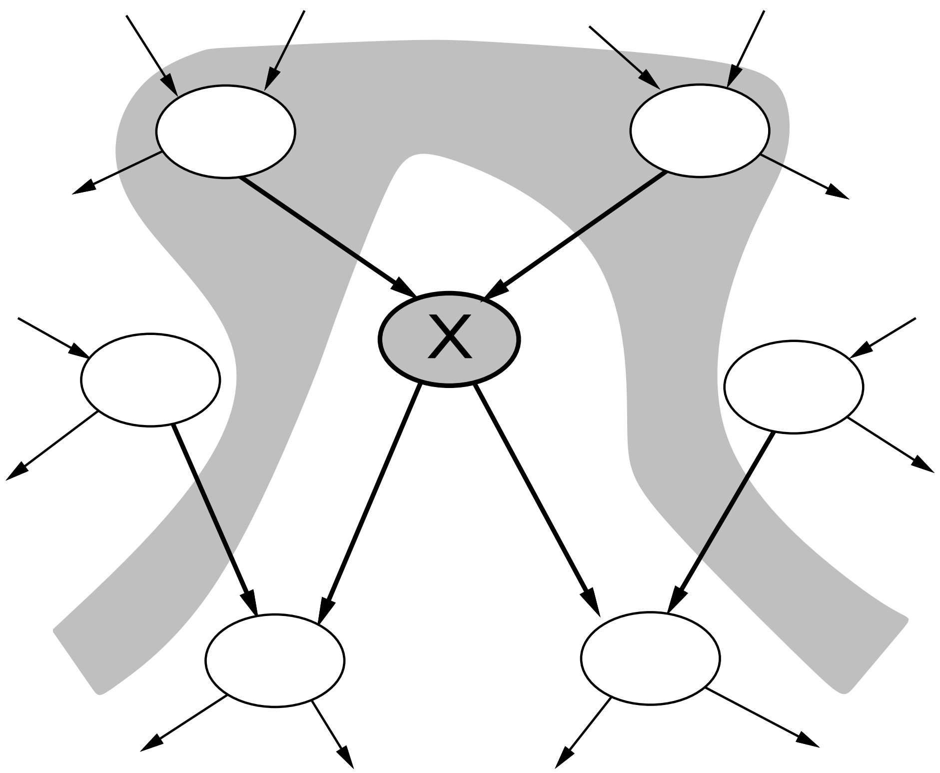
Generate next state by sampling one variable given Markov blanket
Sample each variable in turn, keeping evidence fixed

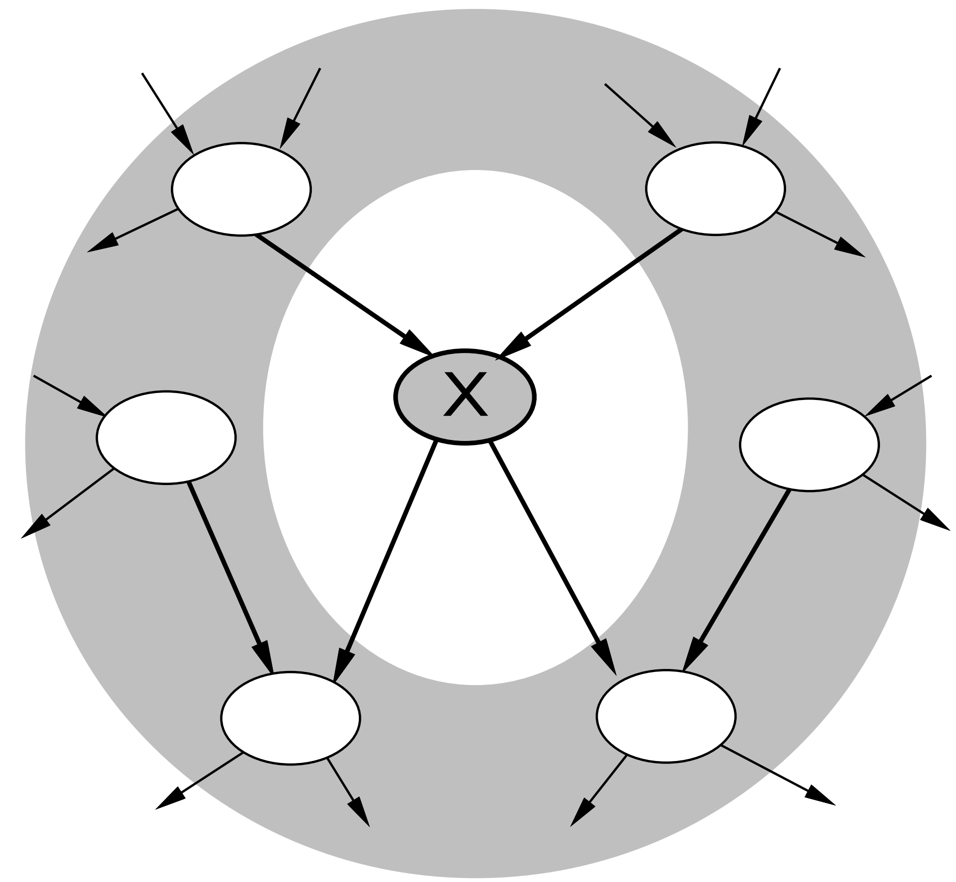
function MCMC-Ask($X, \mathbf{e}, bn, N$) **returns** an estimate of $P(X|\mathbf{e})$
  local variables: $\mathbf{N}[X]$, a vector of counts over X, initially zero
    $\mathbf{Z}$, the nonevidence variables in bn
    $\mathbf{x}$, the current state of the network, initially copied from $\mathbf{e}$

  initialize $\mathbf{x}$ with random values for the variables in $\mathbf{Y}$
  for $j = 1$ to N **do**
    for each Z_i in $\mathbf{Z}$ **do**
      sample the value of Z_i in $\mathbf{x}$ from $\mathbf{P}(Z_i|mb(Z_i))$
        given the values of $MB(Z_i)$ in $\mathbf{x}$
      $\mathbf{N}[x] \leftarrow \mathbf{N}[x] + 1$ where x is the value of X in $\mathbf{x}$
  return Normalize($\mathbf{N}[X]$)

Can also choose a variable to sample at random each time

The extent of dependencies in Bayesian networks



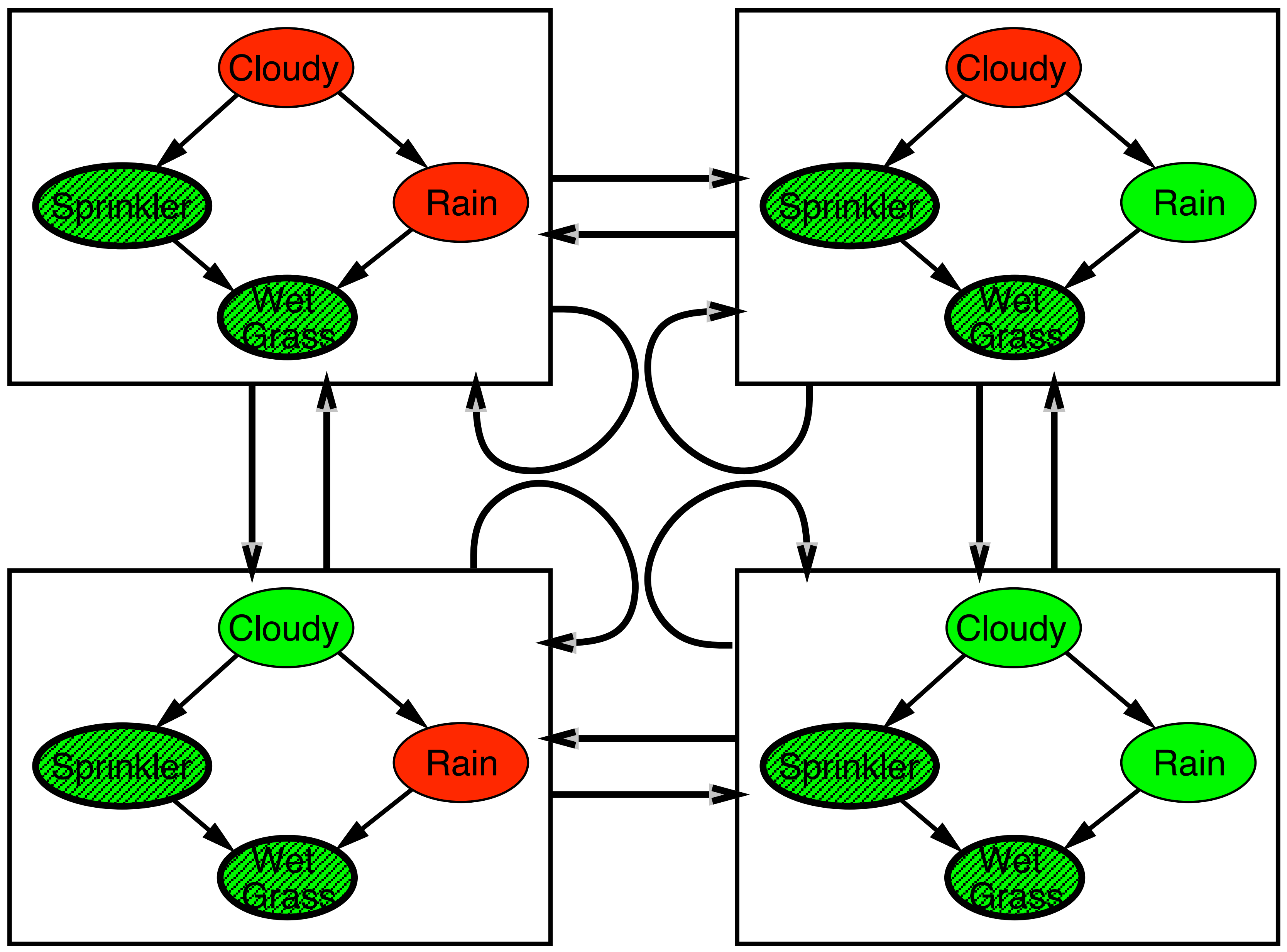


- A node X is conditionally independent of its non-descendants given its parents

- A node X is conditionally independent of all the other nodes in the network given its **Markov blanket**.

The Markov chain

With $Sprinkler = true, WetGrass = true$, there are four states:



Wander about for a while, average what you see

After obtaining the MCMC samples

Estimate $\mathbf{P}(Rain|Sprinkler = true, WetGrass = true)$

Sample $Cloudy$ or $Rain$ given its Markov blanket, repeat.
Count number of times $Rain$ is true and false in the samples.

E.g., visit 100 states
31 have $Rain = true$, 69 have $Rain = false$

$$\hat{\mathbf{P}}(Rain|Sprinkler = true, WetGrass = true)$$
$$= \textsc{Normalize}(\langle 31, 69 \rangle) = \langle 0.31, 0.69 \rangle$$

Theorem: chain approaches stationary distribution:
long-run fraction of time spent in each state is exactly proportional to its posterior probability

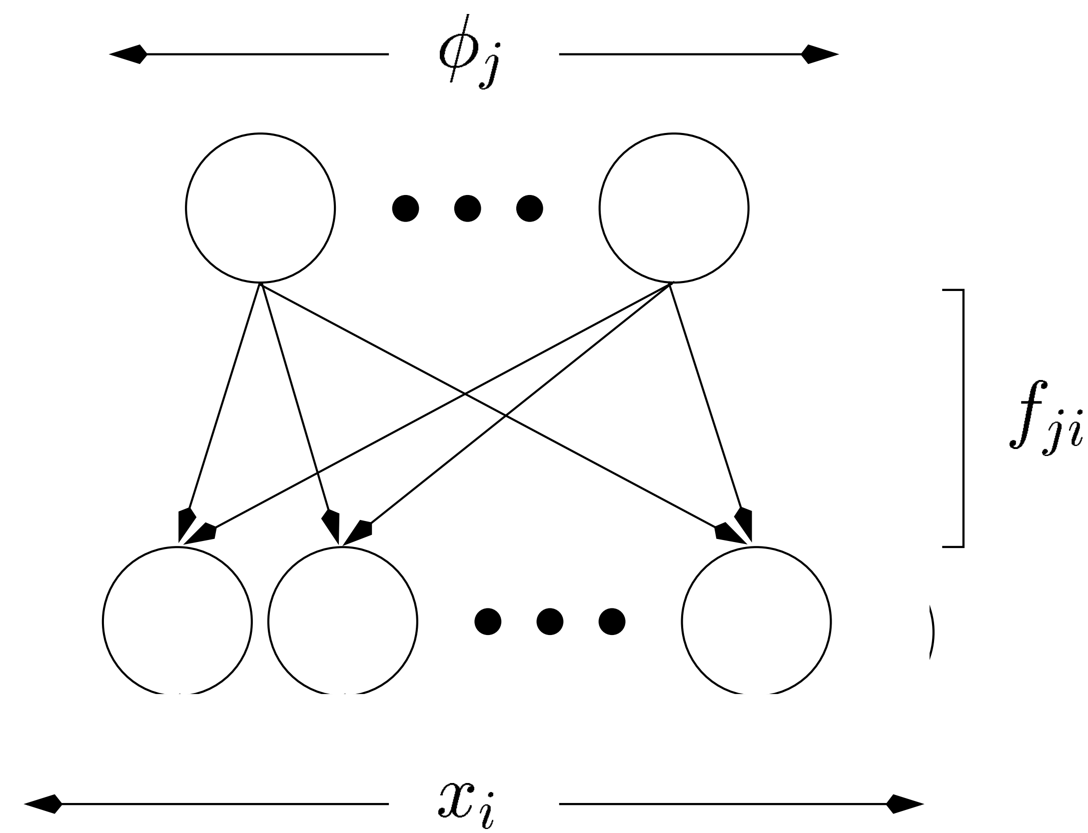
But:

1. Difficult to tell when samples have converged. Theorem only applies in limit, and it could take time to "settle in".
2. Can also be inefficient if each state depends on many other variables.

Gibbs sampling (back to the noisy-OR example)

- Model represents stochastic binary features.
- Each input x_i encodes the probability that the ith binary input feature is present.
- The set of features represented by ϕ_j is defined by weights f_{ij} which encode the probability that feature i is an instance of ϕ_j.
- Trick: It's easier to adapt weights in an unbounded space, so use the transformation:

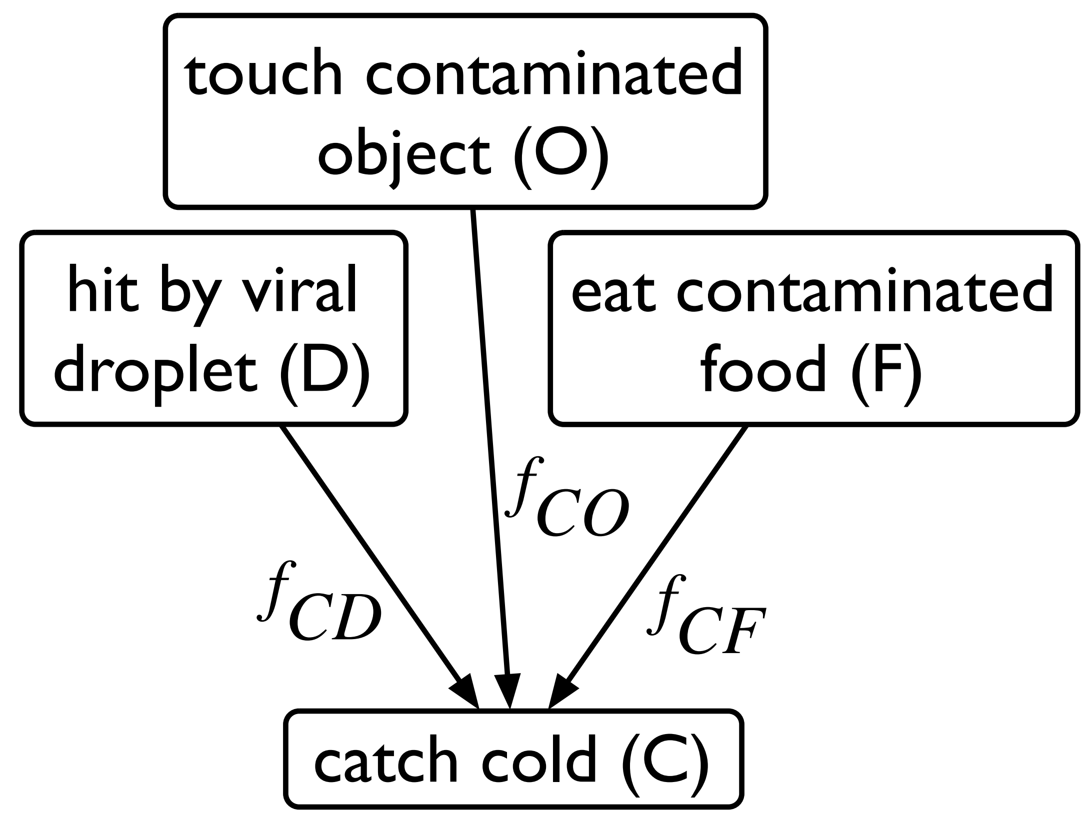
$$f = 1/(1 + \exp(-w))$$



Beyond tables: modeling causal relationships using Noisy-OR

- We assume each cause C_j can produce effect E_i with probability f_{ij}.
- The noisy-OR model assumes the parent causes of effect E_i contribute independently.
- The probability that none of them caused effect E_i is simply the product of the probabilities that each one *did not* cause E_i.
- The probability that any of them caused E_i is just one minus the above, i.e.

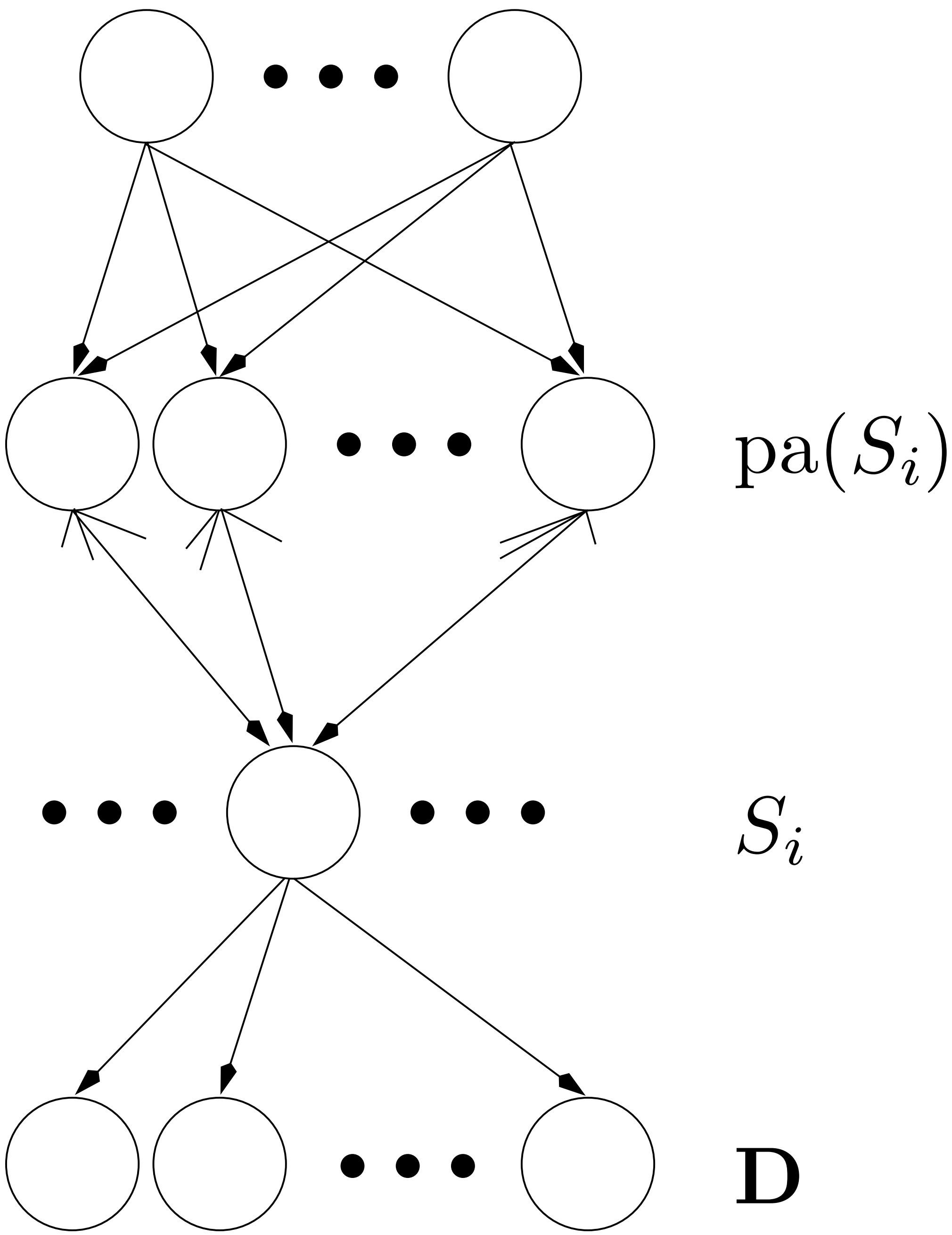
$$\begin{aligned} P(E_i|\mathrm{par}(E_i)) &= P(E_i|C_1, \ldots, C_n) \\ &= 1 - \prod_i (1 - P(E_i|C_j)) \\ &= 1 - \prod_i (1 - f_{ij}) \end{aligned}$$



$$P(C|D, O, F) = 1 - (1 - f_{CD})(1 - f_{CO})(1 - f_{CF})$$

Hierarchical Statistical Models

A Bayesian belief network:



The joint probability of binary states is

$$P(\mathbf{S}|\mathbf{W}) = \prod_i P(S_i|\mathrm{pa}(S_i), \mathbf{W})$$

The probability S_i depends only on its parents:

$$P(S_i|\mathrm{pa}(S_i), \mathbf{W}) = \begin{cases} h(\sum_j S_j w_{ji}) & \text{if } S_i = 1 \\ 1 - h(\sum_j S_j w_{ji}) & \text{if } S_i = 0 \end{cases}$$

The function h specifies how causes are combined, $h(u) = 1 - \exp(-u)$, $u > 0$.

Main points:

- hierarchical structure allows model to form high order representations
- upper states are priors for lower states
- weights encode higher order features

Inferring the best representation of the observed variables

- Given on the input **D**, the is no simple way to determine which states are the input's most likely causes.
  - Computing the most probable network state is an *inference* process
  - we want to find the explanation of the data with highest probability
  - this can be done efficiently with *Gibbs sampling*
- Gibbs sampling is another example of an MCMC method
- Key idea:

The samples are guaranteed to converge to the

true posterior probability distribution

Gibbs Sampling

Gibbs sampling is a way to select an ensemble of states that are representative of the posterior distribution $P(\mathbf{S}|\mathbf{D},\mathbf{W})$.

- Each state of the network is updated iteratively according to the probability of S_i given the remaining states.
- this conditional probability can be computed using (Neal, 1992)

$$P(S_i = a|S_j : j \neq i, \mathbf{W}) \propto P(S_i = a|\mathrm{pa}(S_i), \mathbf{W}) \prod_{j \in \mathrm{ch}(S_i)} P(S_j|\mathrm{pa}(S_j), S_i = a, \mathbf{W})$$

- limiting ensemble of states will be typical samples from $P(\mathbf{S}|\mathbf{D},\mathbf{W})$
- also works if any subset of states are fixed and the rest are sampled

The Gibbs sampling equations (derivation omitted)

The probability of S_i changing state given the remaining states is

$$P(S_i = 1 - S_i | S_j : j \neq i, \mathbf{W}) = \frac{1}{1 + \exp(-\Delta x_i)}$$

Δx_i indicates how much changing the state S_i changes the probability of the whole network state

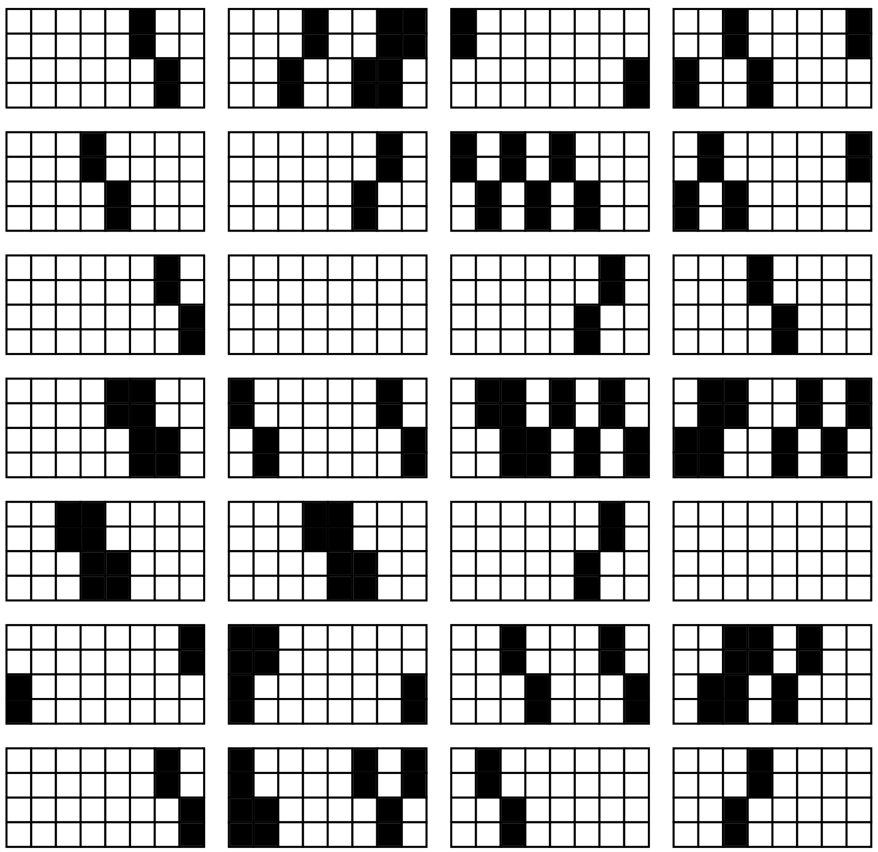
$$\begin{aligned}\Delta x_i = & \log h(u_i; 1 - S_i) - \log h(u_i; S_i) \\ & + \sum_{j \in \mathrm{ch}(S_i)} \log h(u_j + \delta_{ij}; S_j) - \log h(u_j; S_j)\end{aligned}$$

- u_i is the causal input to S_i, $u_i = \sum_k S_k w_{ki}$
- δ_j specifies the change in u_j for a change in S_i, $\delta_{ij} = +S_j w_{ij}$ if $S_i = 0$, or $-S_j w_{ij}$ if $S_i = 1$

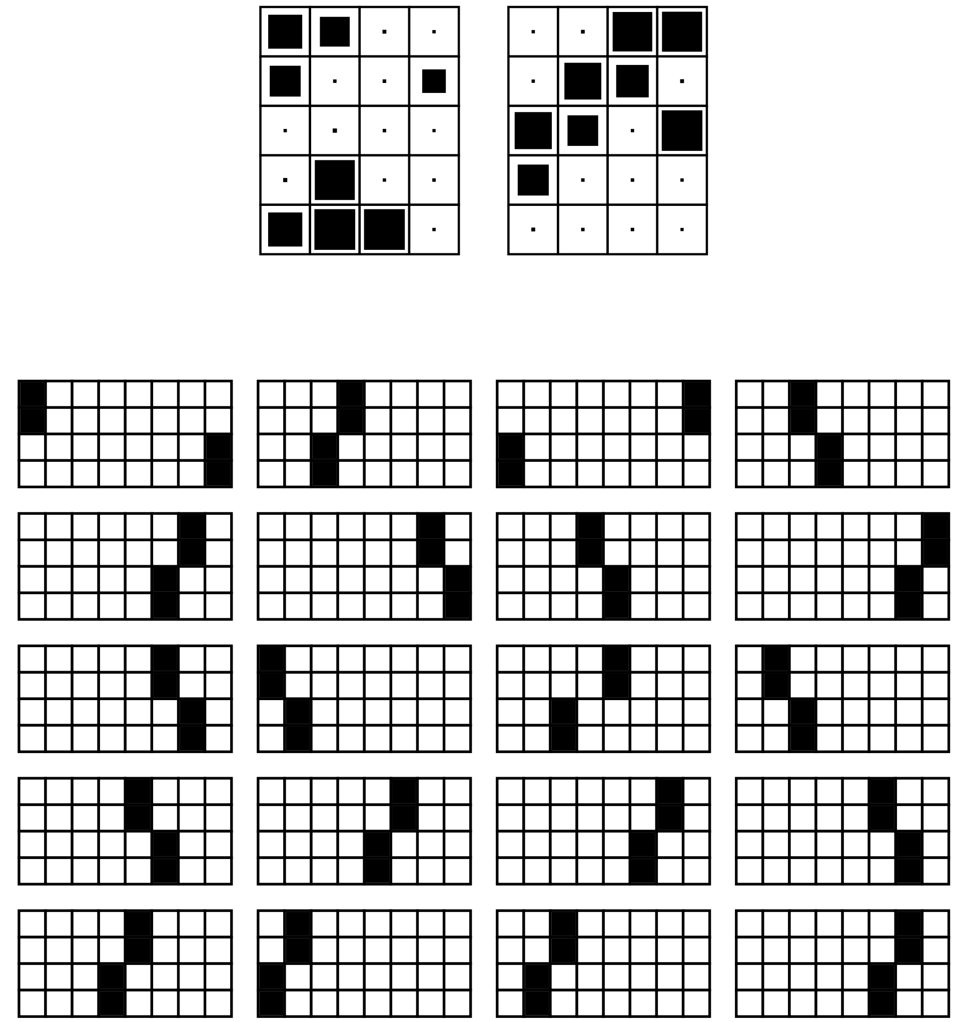
Interpretation of the Gibbs sampling equation

- The Gibbs equation can be interpreted as: *feedback* + $\sum$ *feedforward*
- *feed-back:* how consistent is Si with current causes?
- $\sum$ *feedforward:* how likely is Si a cause of its children
- feedback allows the lower-level units to use information only computable at higher levels
- feedback determines (disambiguates) the state when the feedforward input is ambiguous

The Shifter Problem

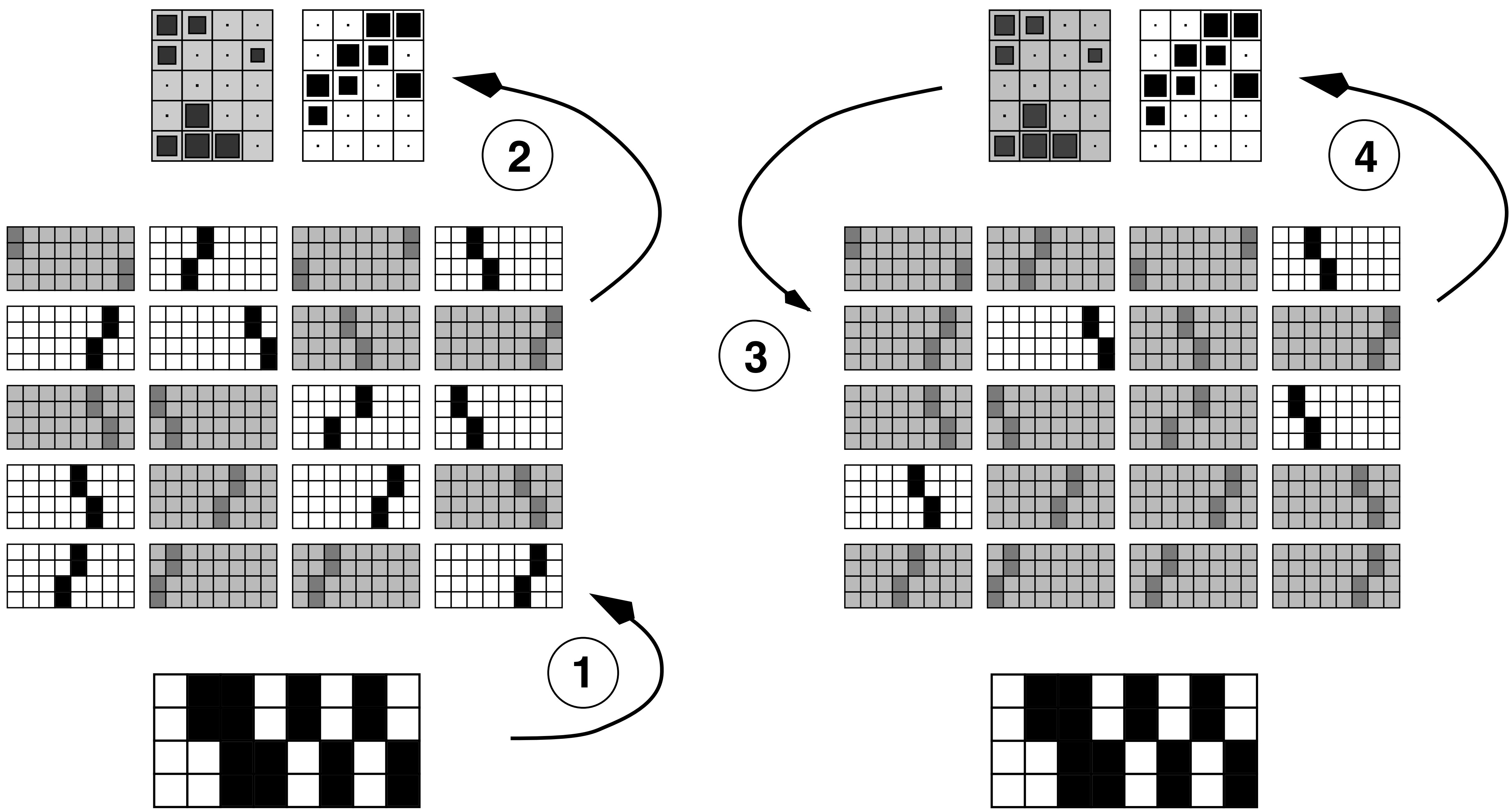


Shift patterns



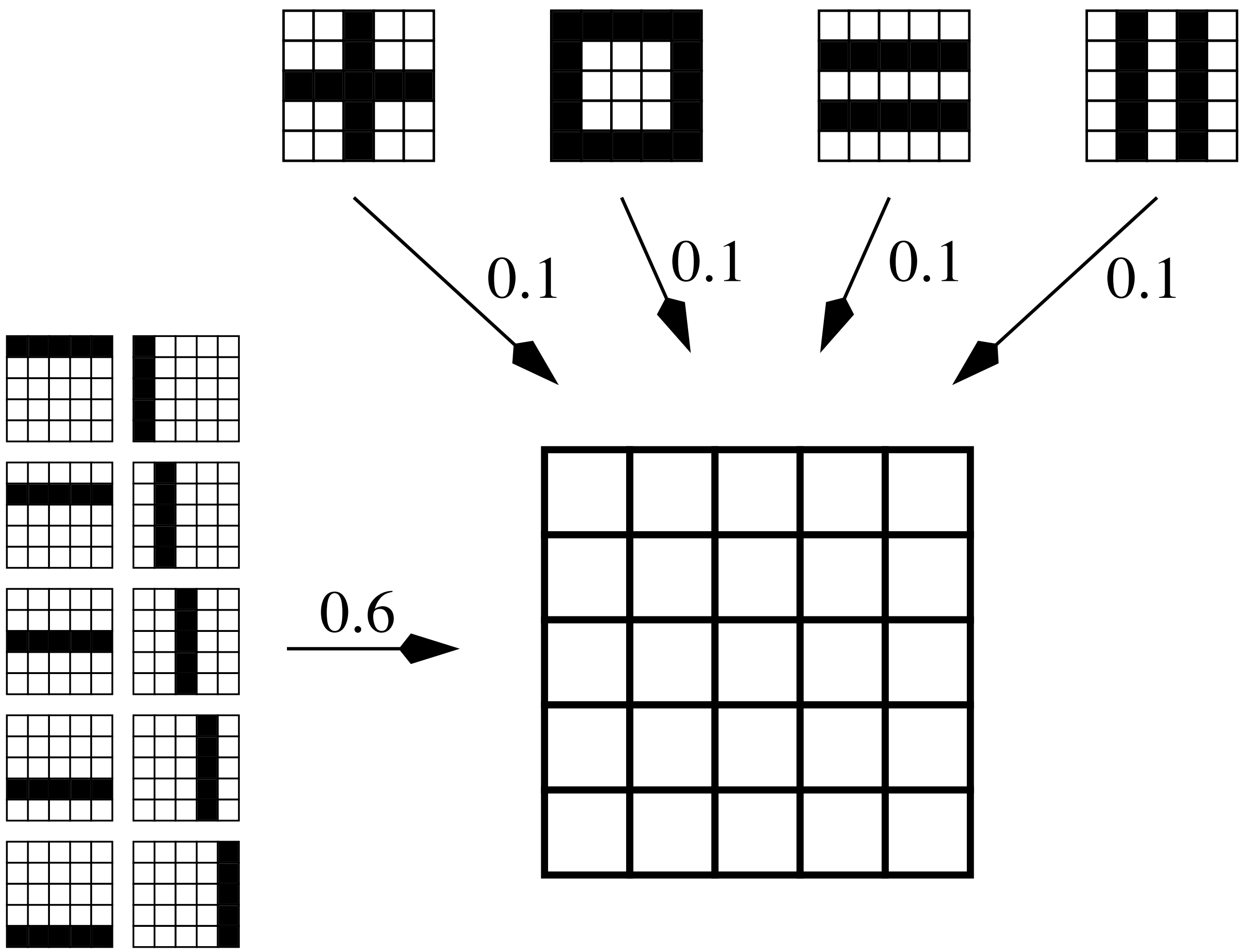
weights of a 32-20-2 network after learning

Gibbs sampling: feedback disambiguates lower-level states

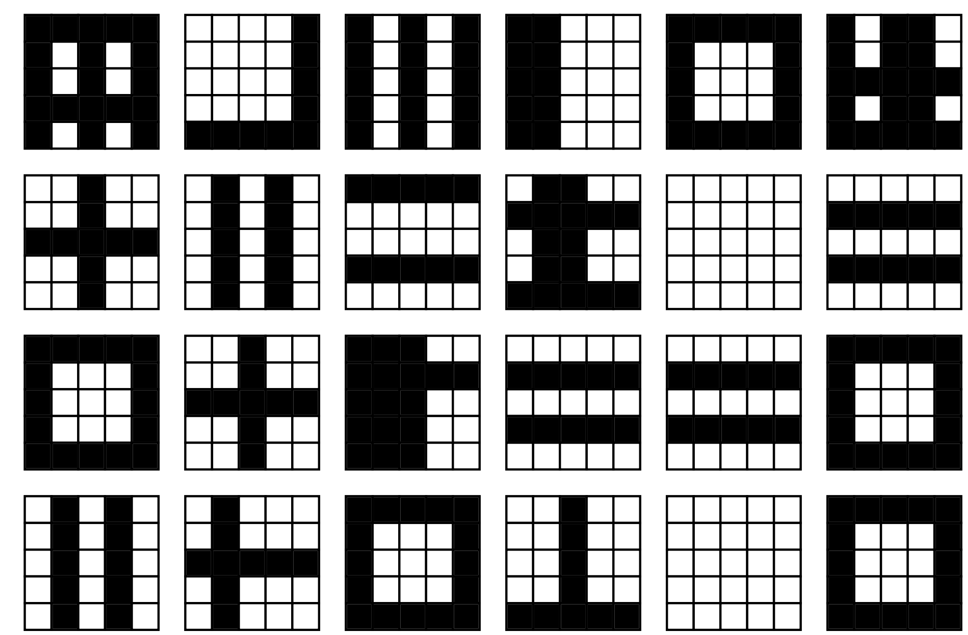


One the structure learned, the Gibbs updating convergences in two sweeps.

The higher-order lines problem



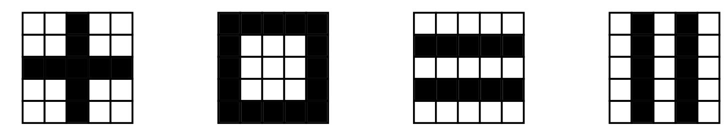
The true generative model



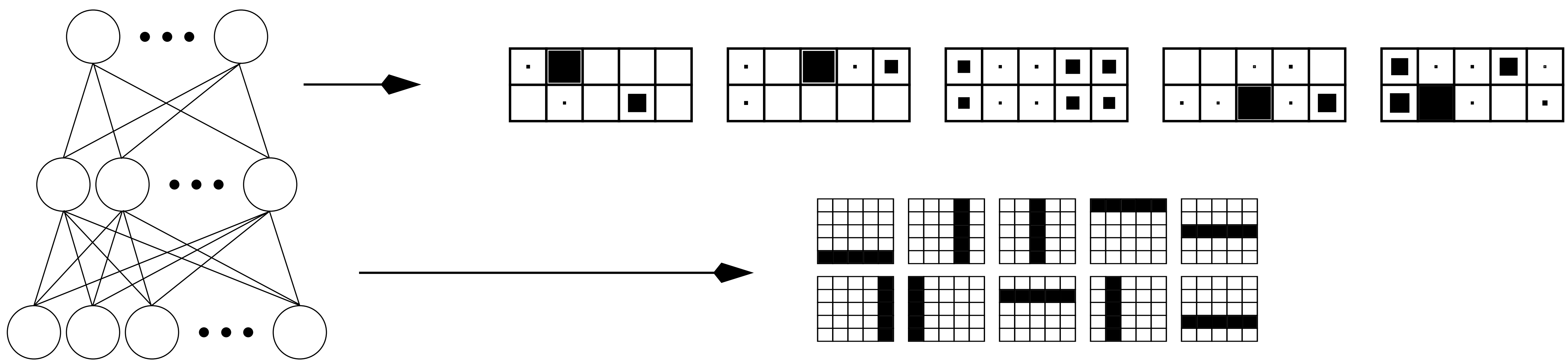
Patterns sampled from the model

Can we infer the structure of the network given only the patterns?

Weights in a 25-10-5 belief network after learning

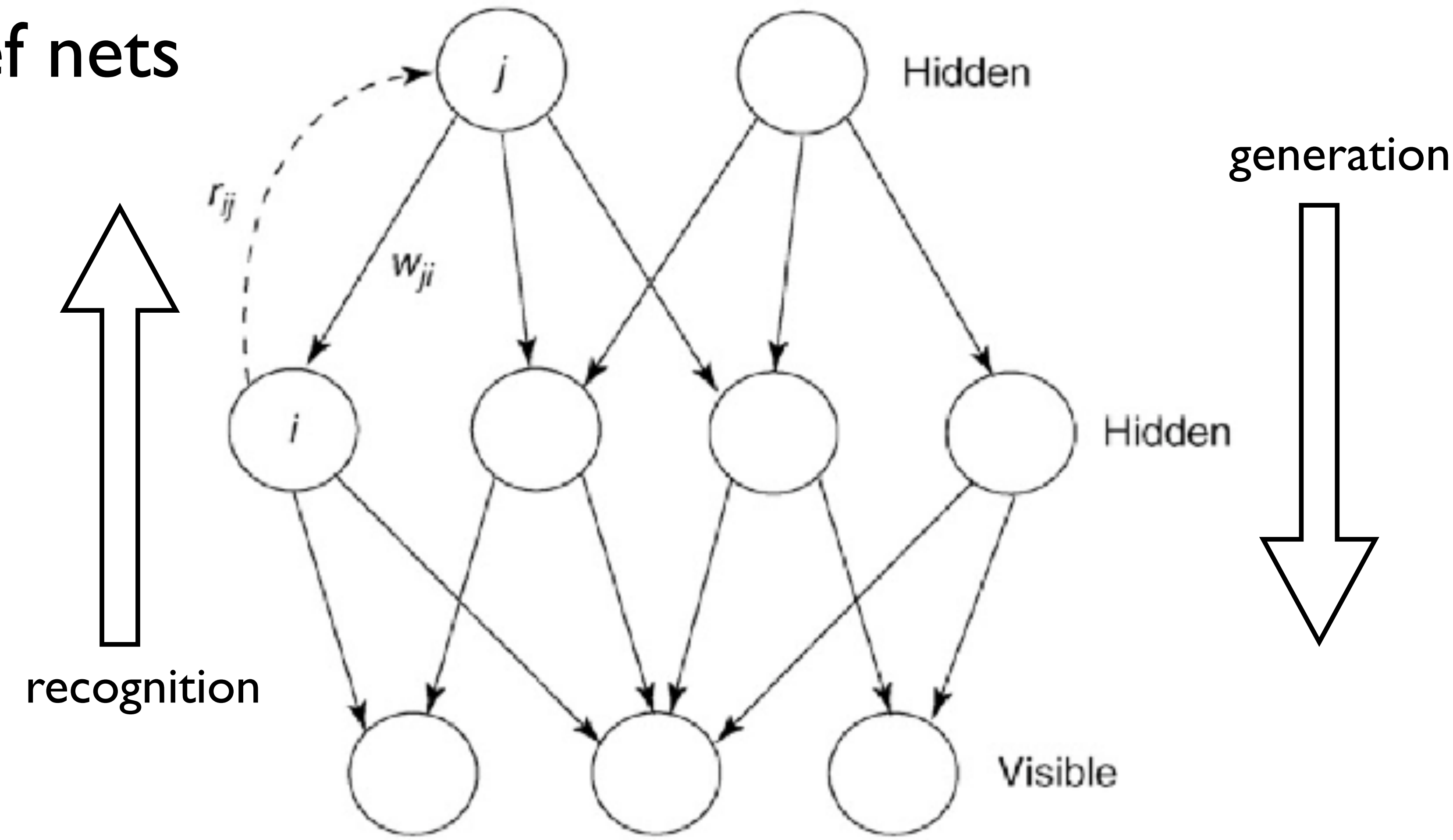


The second layer learns combinations of the first layer features



The first layer of weights learn that
patterns are combinations of lines.

Logistic belief nets



- generative model:

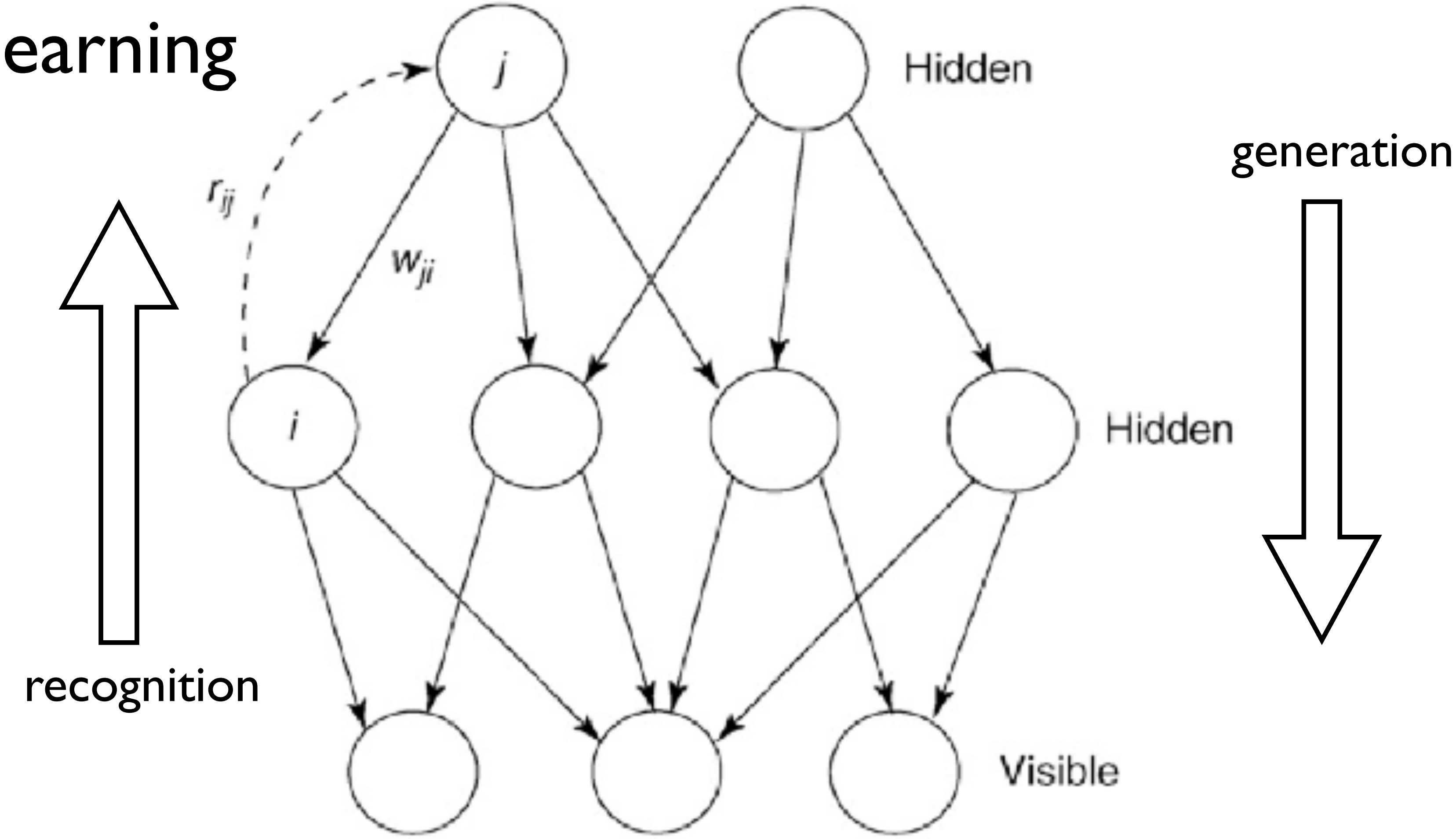
$$p(v_i = 1) = \sigma(b_i + \sum_j h_j w_{ij})$$

- $\sigma(x)$ is the logistic function:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

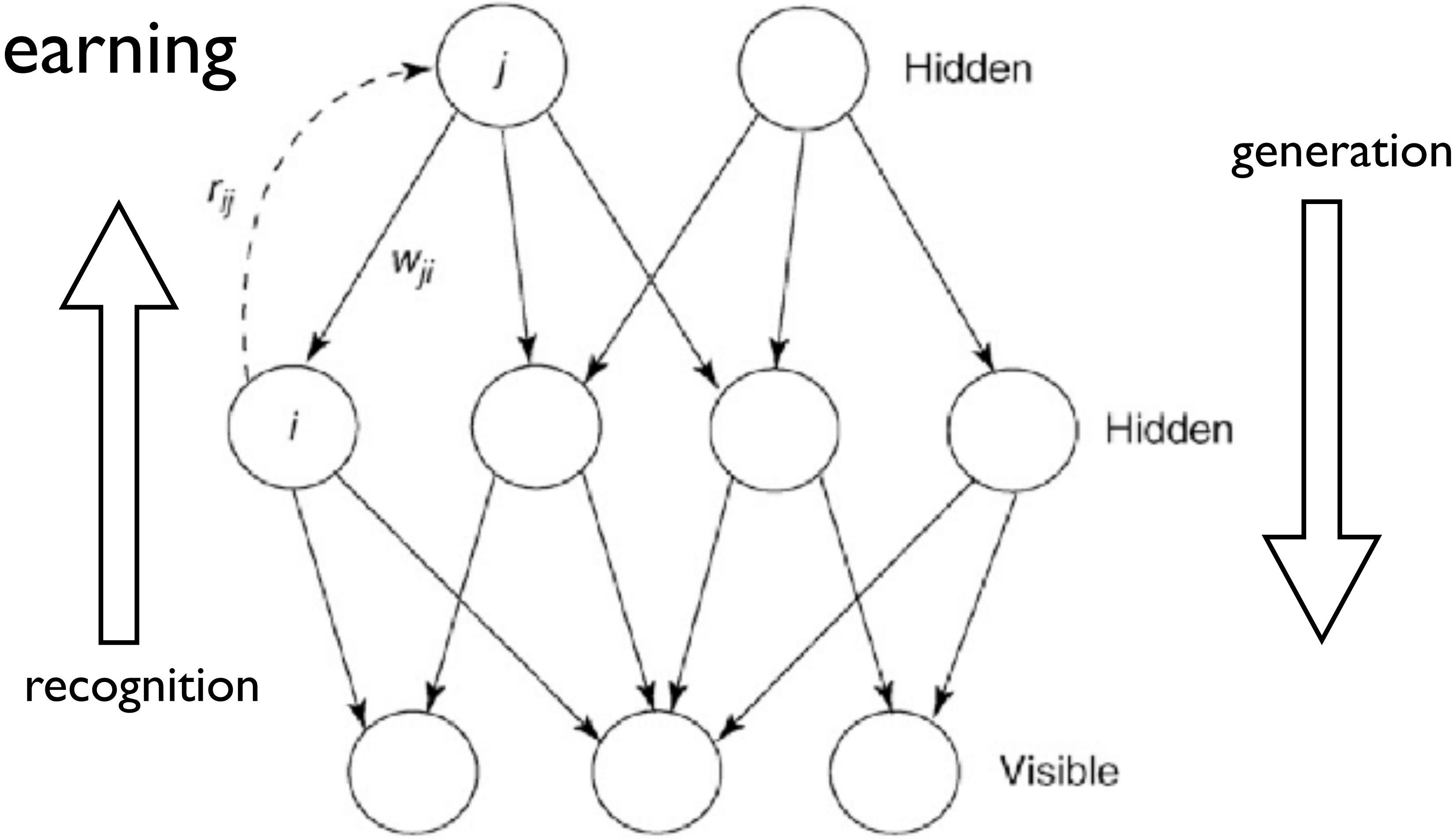
- with deep networks, it is possible to learn complex joint probability distributions

Wake-sleep learning



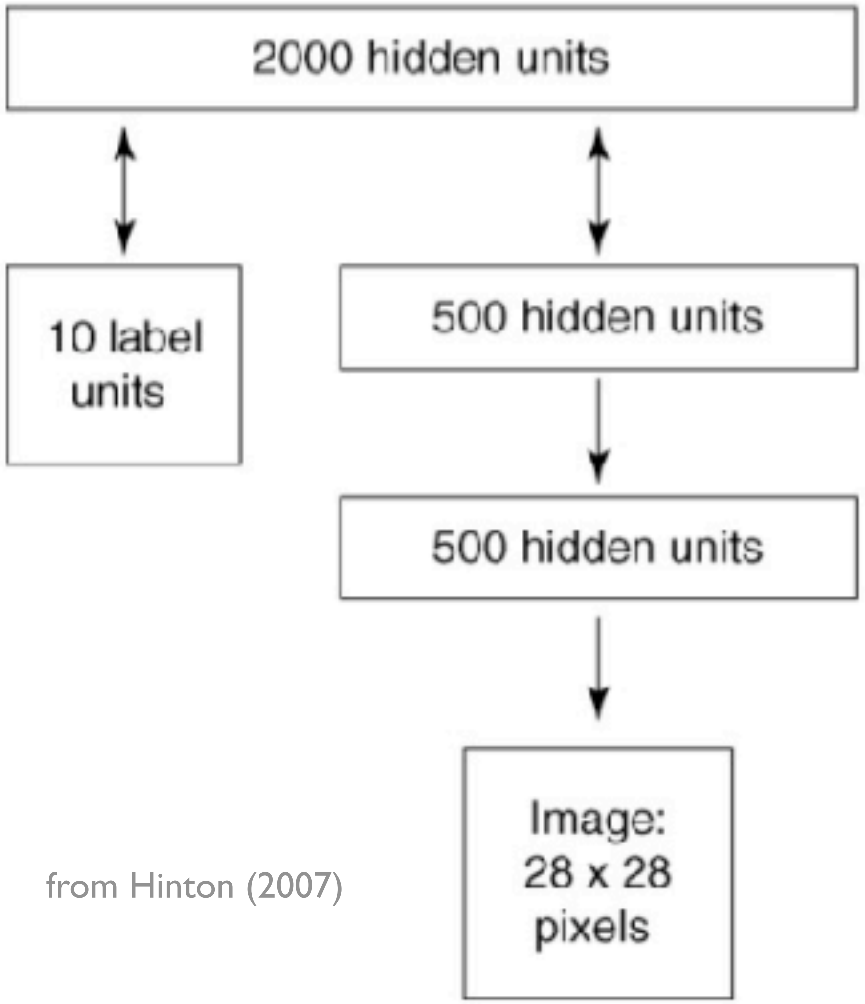
- For probabilistic models
  - top-down weights generate patterns from model distribution
  - bottom-up weights convey distribution of data-vectors
  - ideally the two distributions should match
- The "wake-sleep" algorithm adjusts the weights so that the distribution from recognition (wake) matches the distribution from generation (sleep)

Wake-sleep learning

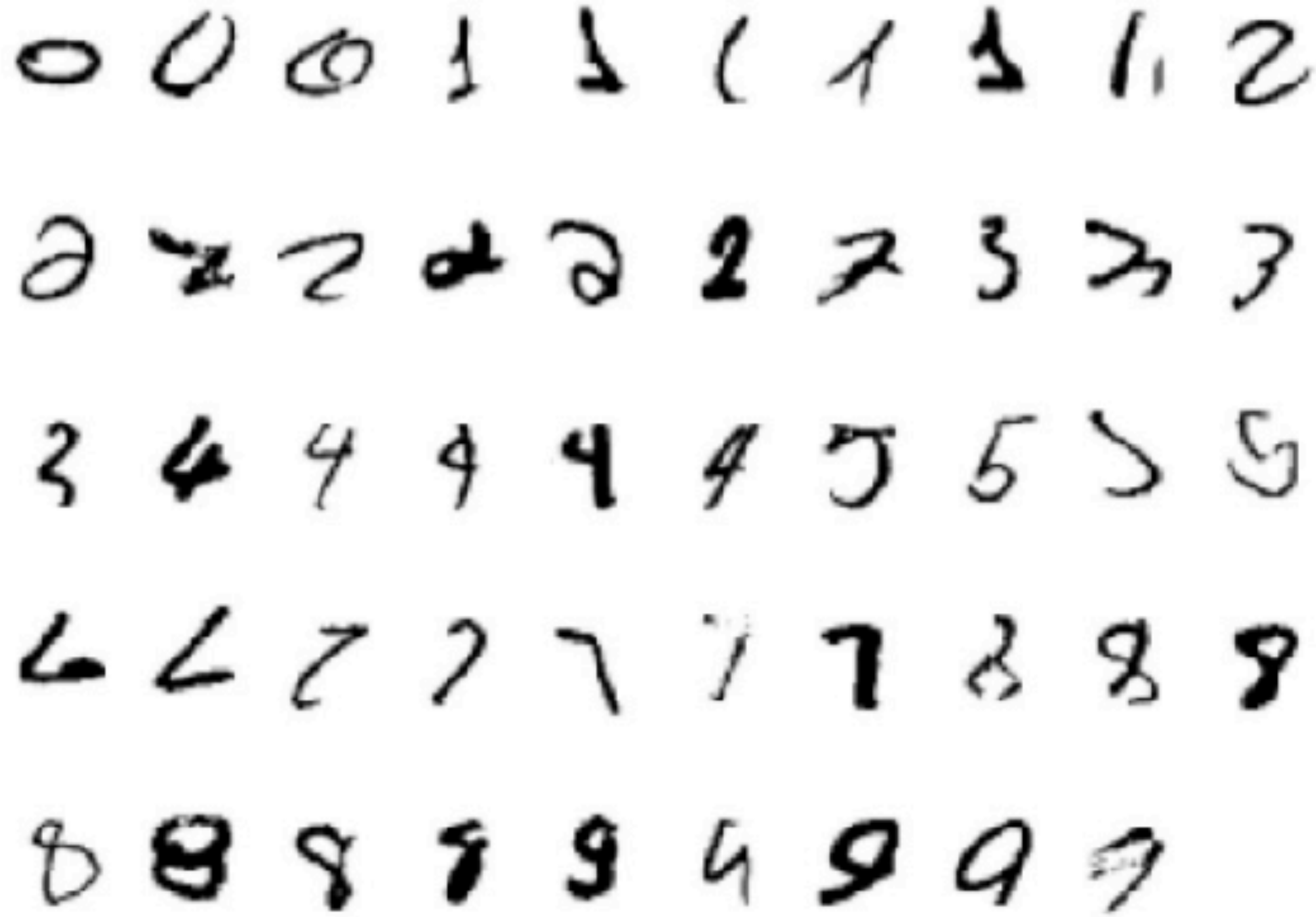


- For each digit in training set
  - bottom-up pass: use recognition weights to stochastically set hidden states
    $$p(h_j = 1) = \sigma(b_j + \sum_i v_i w_{ij})$$
  - adjust generative weights to improve how model generates training data:
    $$\Delta w_{ji} \propto h_j(h_i - \hat{h}_i)$$
  - $\hat{h}_i$ is the probability of activating state i given inferred states h_j

Generative model for hand-written digits

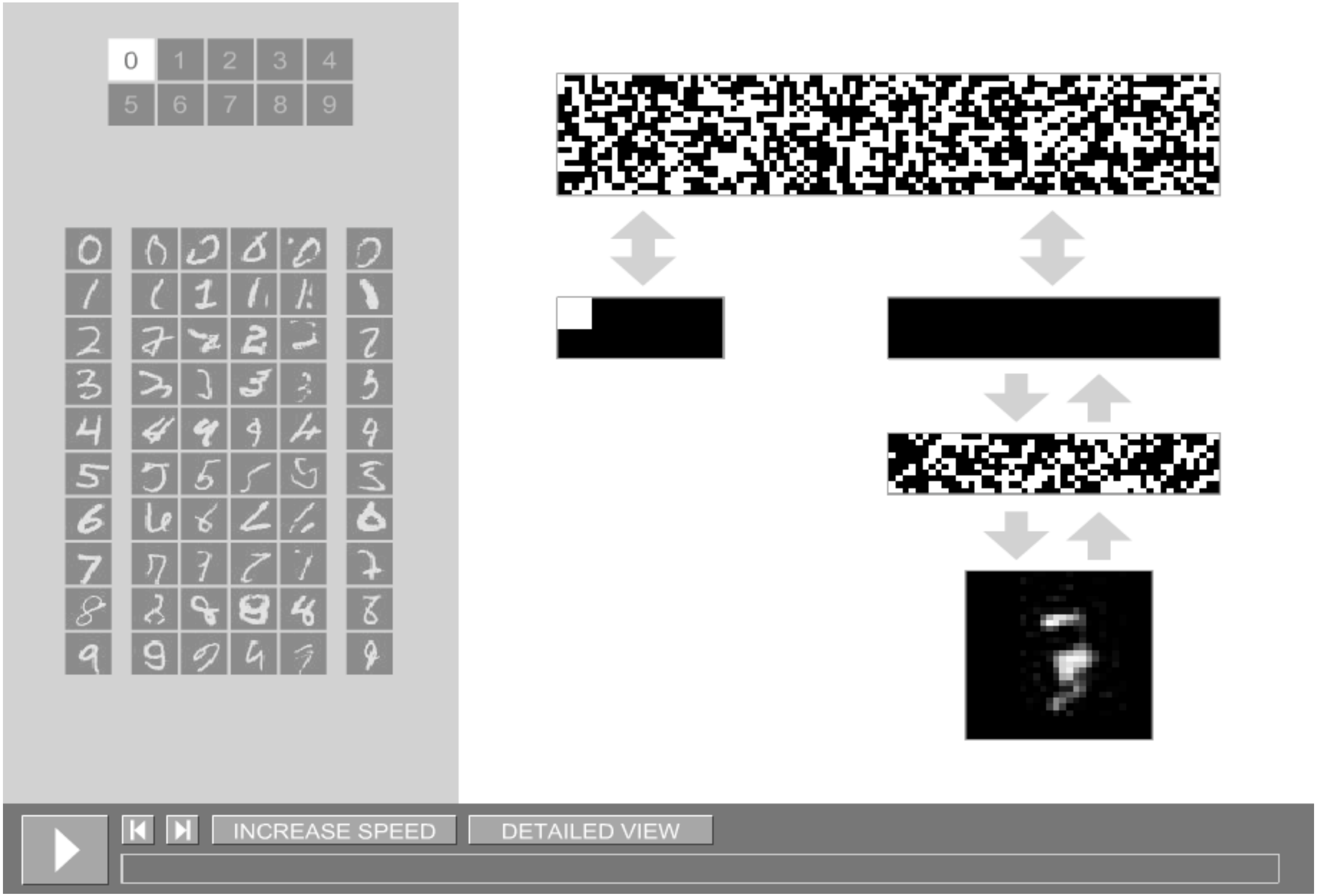


from Hinton (2007)



- Generation:
  - use alternating Gibbs sampling from top-level assoc. memory
  - use directed weights to stochastically generate pixel probs. from sampled binary of 500 hidden units
- Recognition:
  - Use bottom-up weights to produce binary activities in two lower layers
  - use alternating Gibbs sampling in the top two layers

Demo: deep-belief network model (Hinton)



www.cs.toronto.edu/~hinton/adi/index.htm