Artificial Intelligence
EECS 491

Stochastic Inference

Another approach: Inference by stochastic simulation

Basic idea:

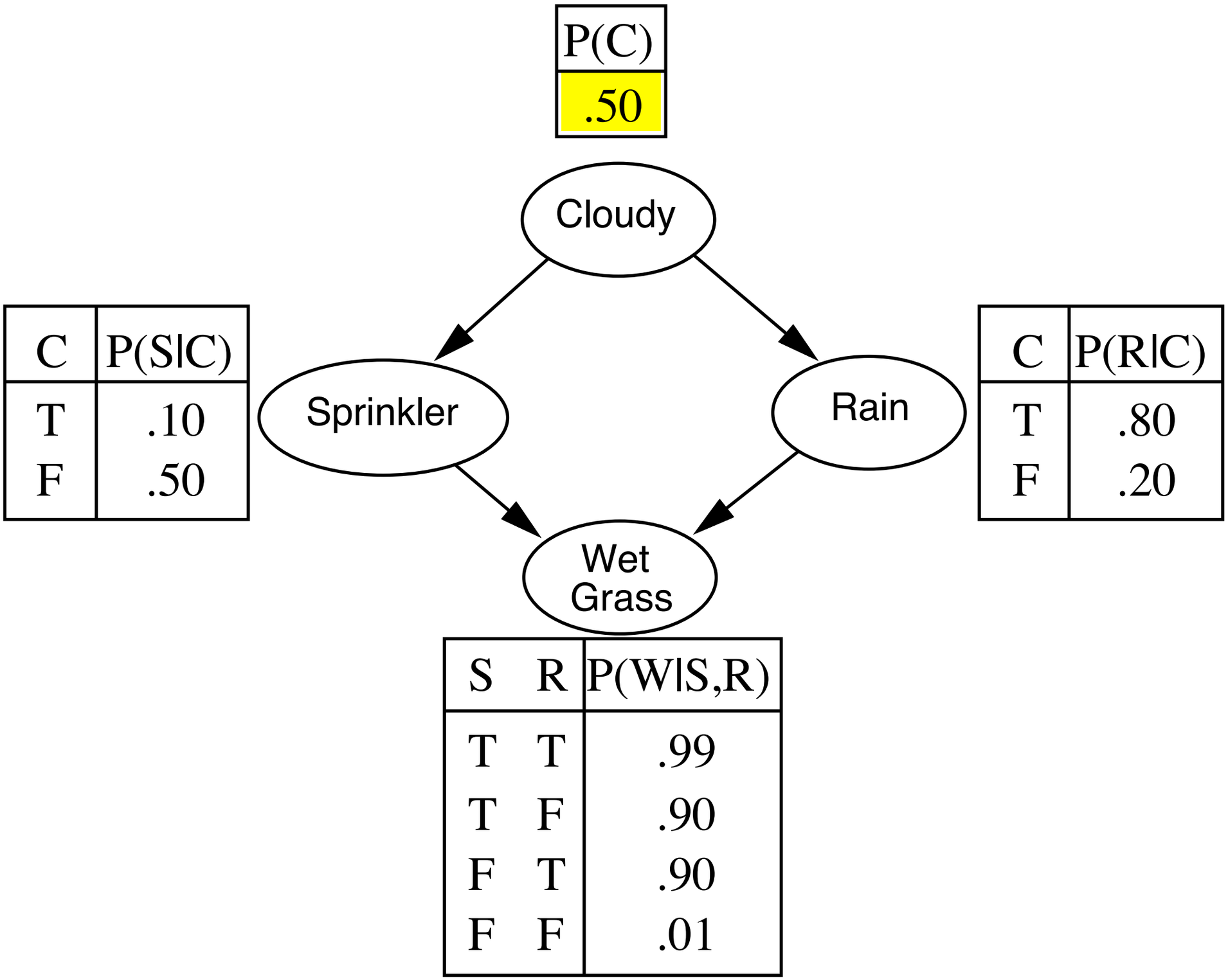
1. Draw N samples from a sampling distribution S
2. Compute an approximate posterior probability
3. Show this converges to the true probability

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 $\mathbf{x}$

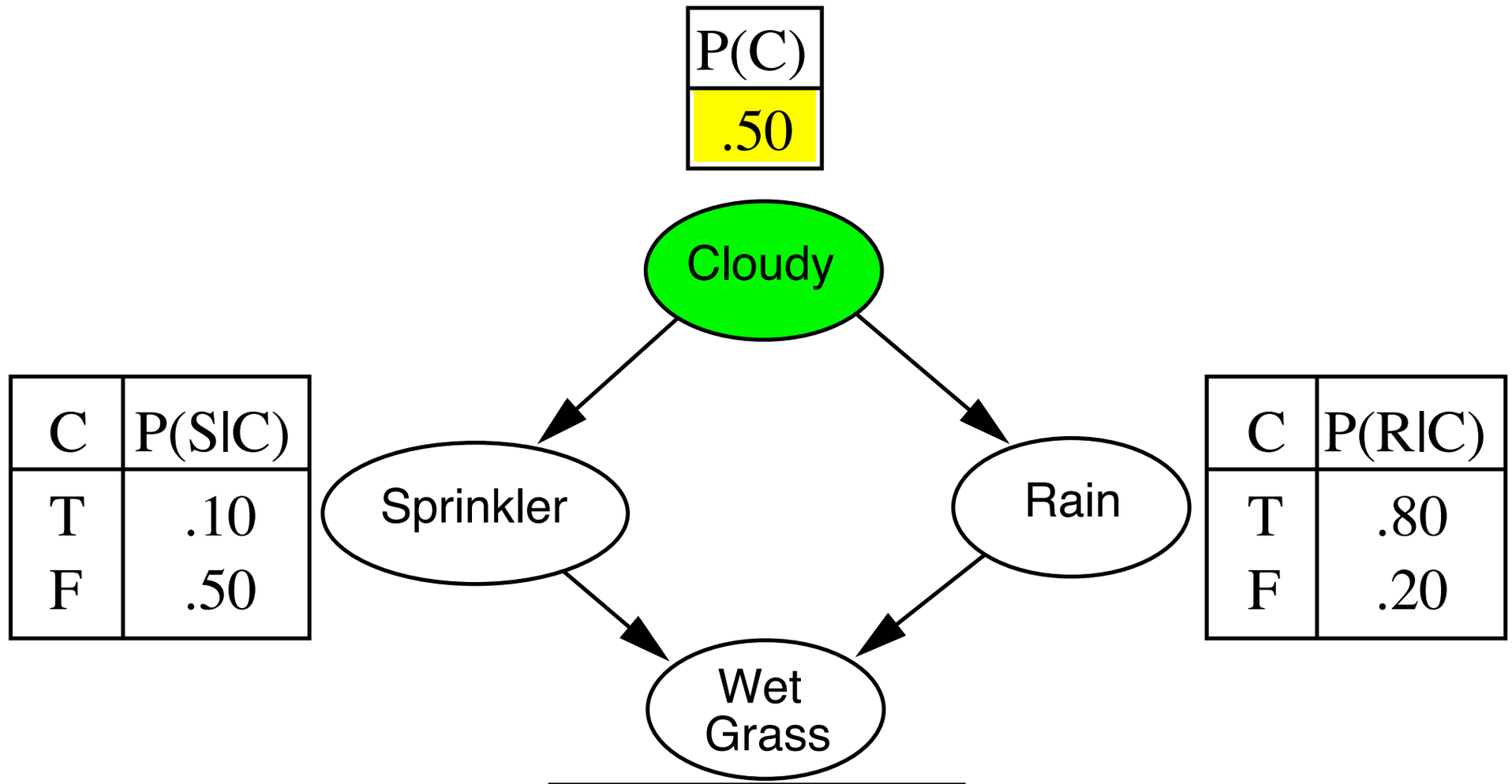
(the following slide series is from our textbook authors.)



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

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

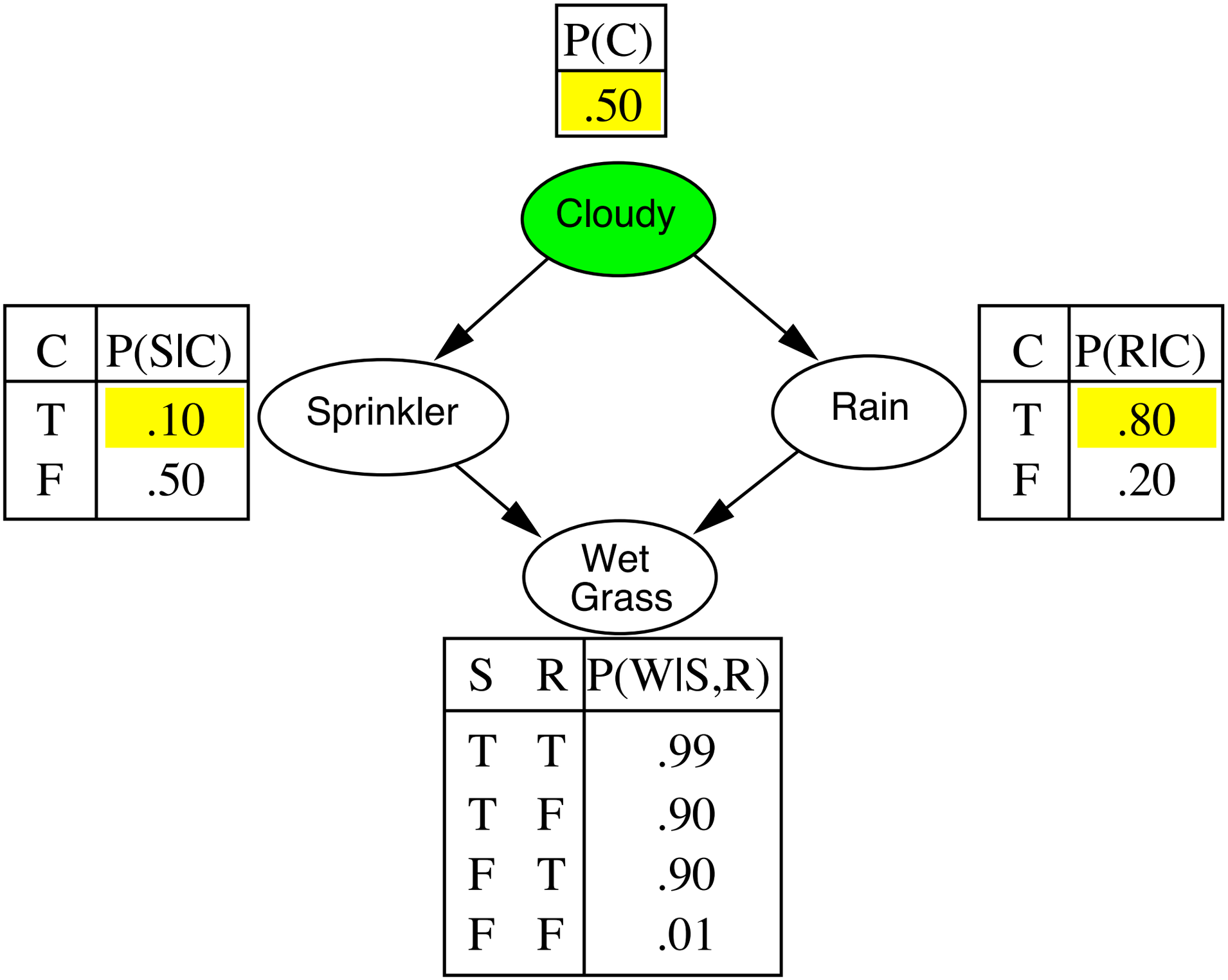
S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

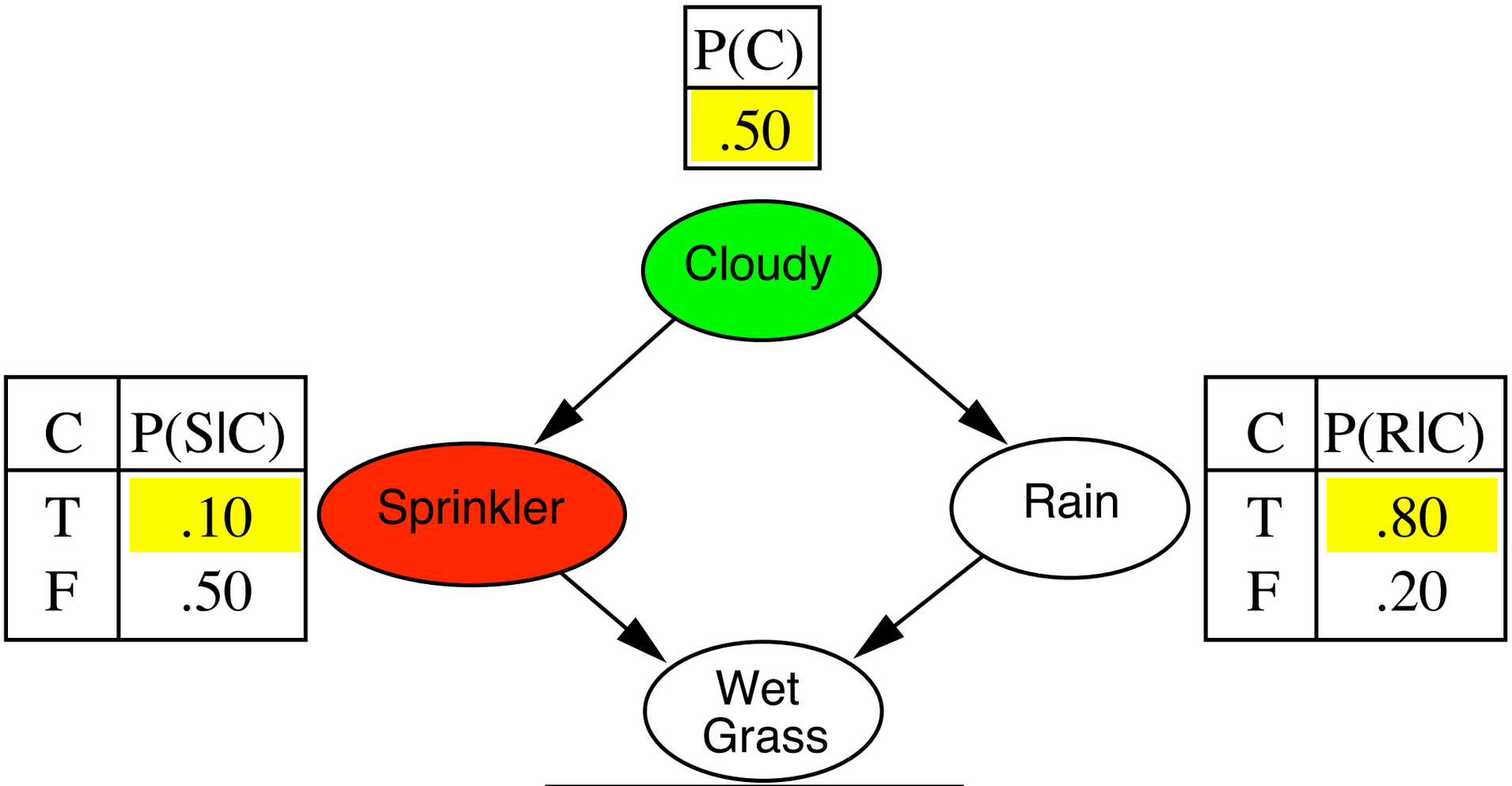


C	P(S|C)
T	.10
F	.50

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

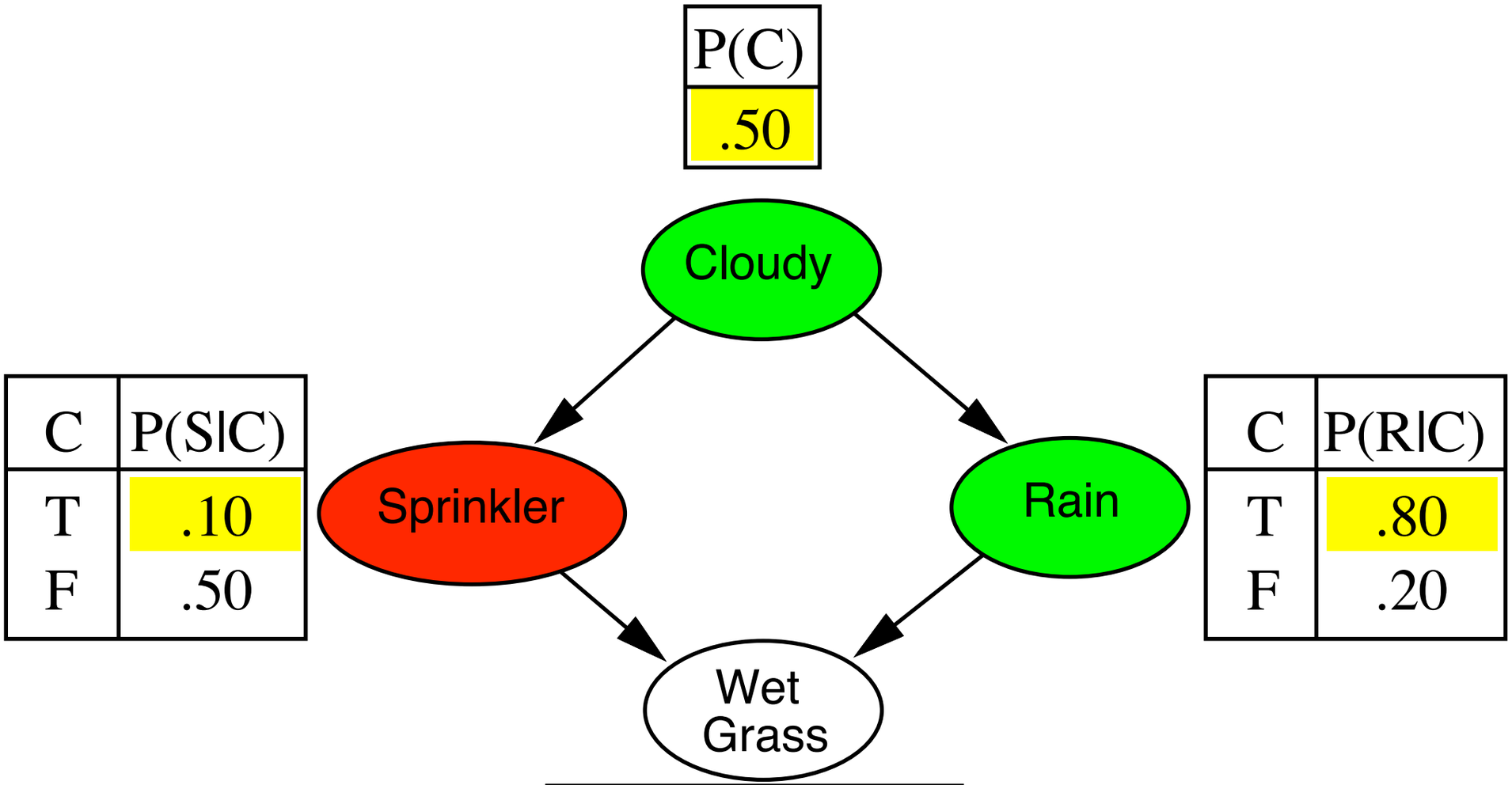


P(C)
.50

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

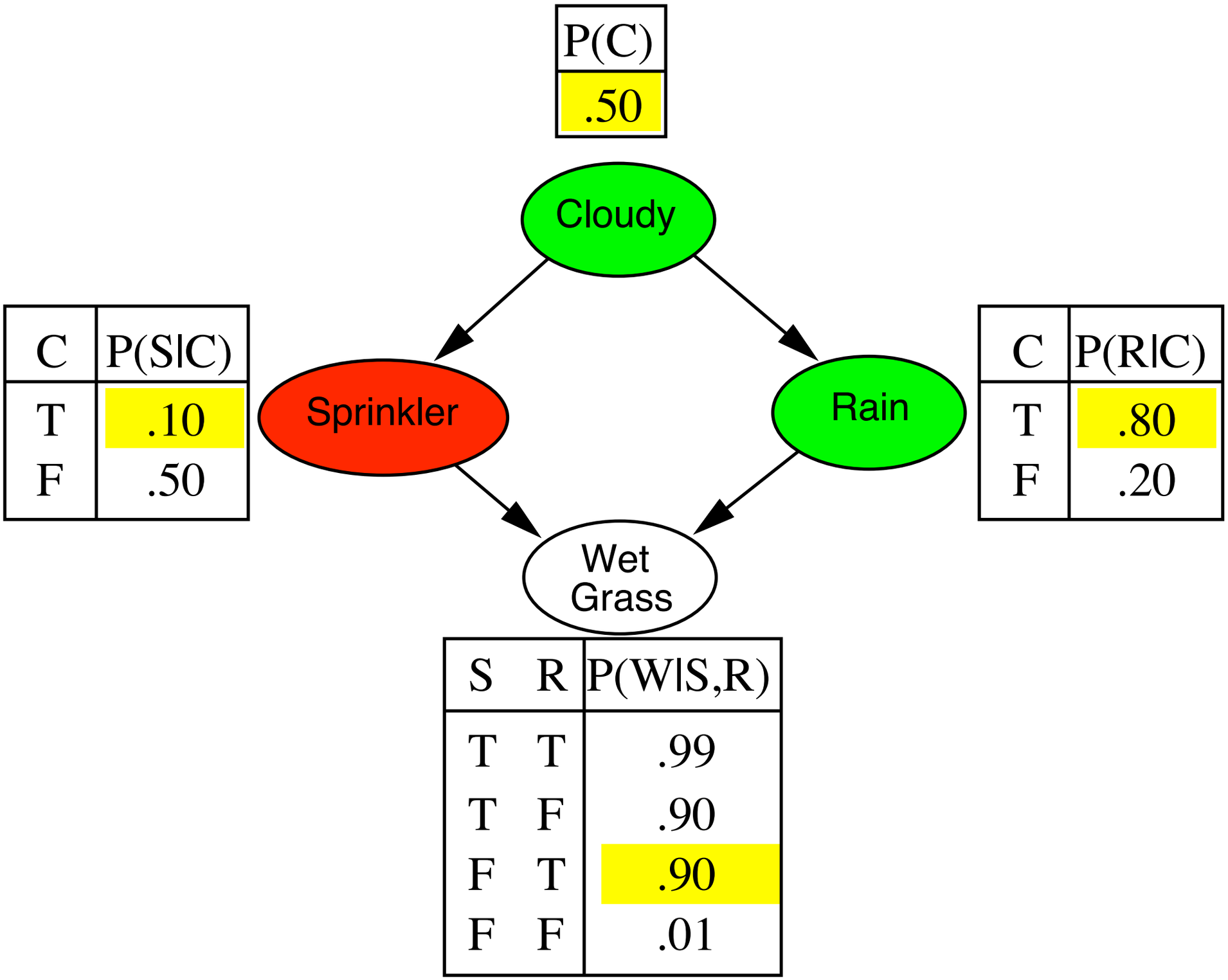


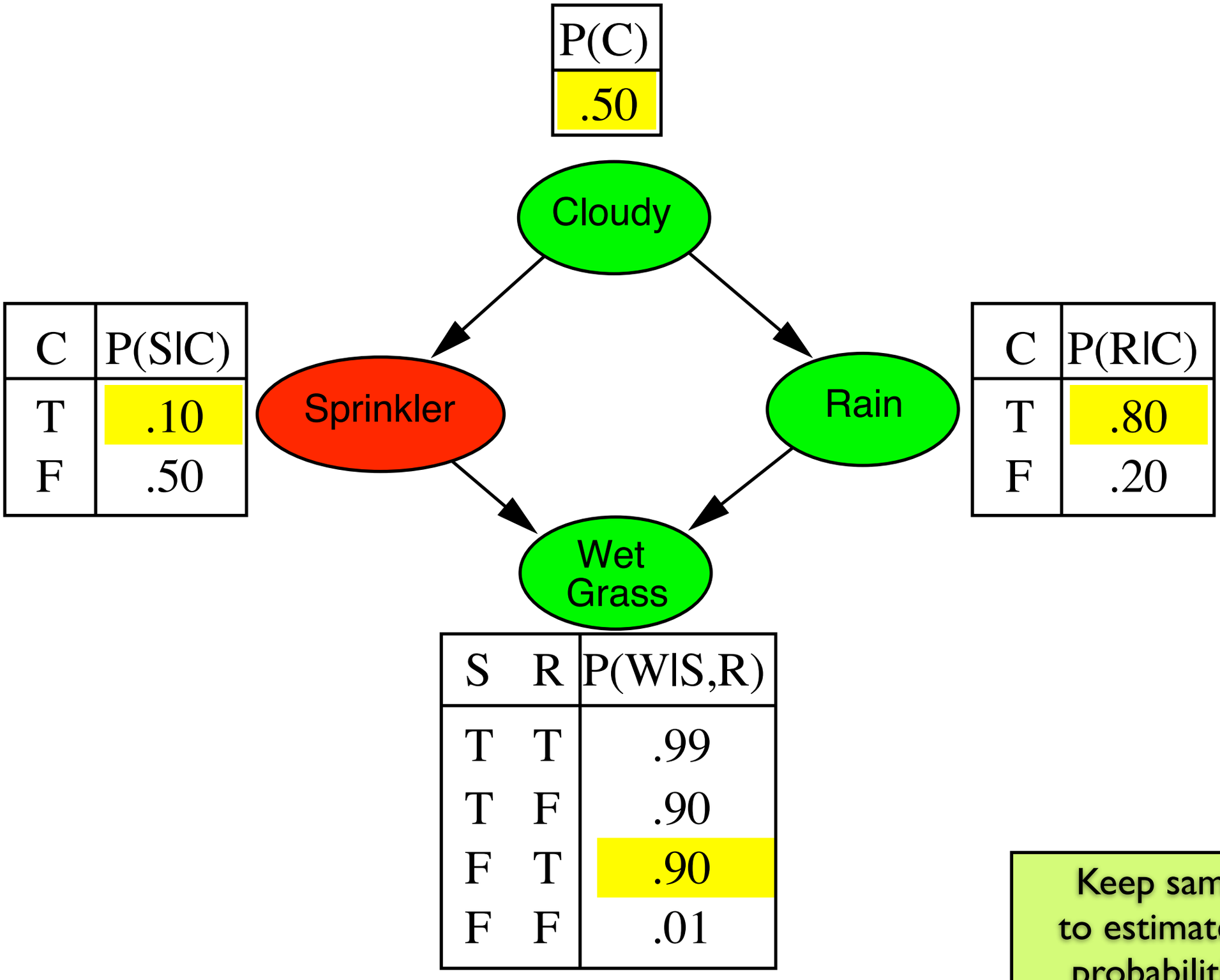
P(C)
.50

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



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

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

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.

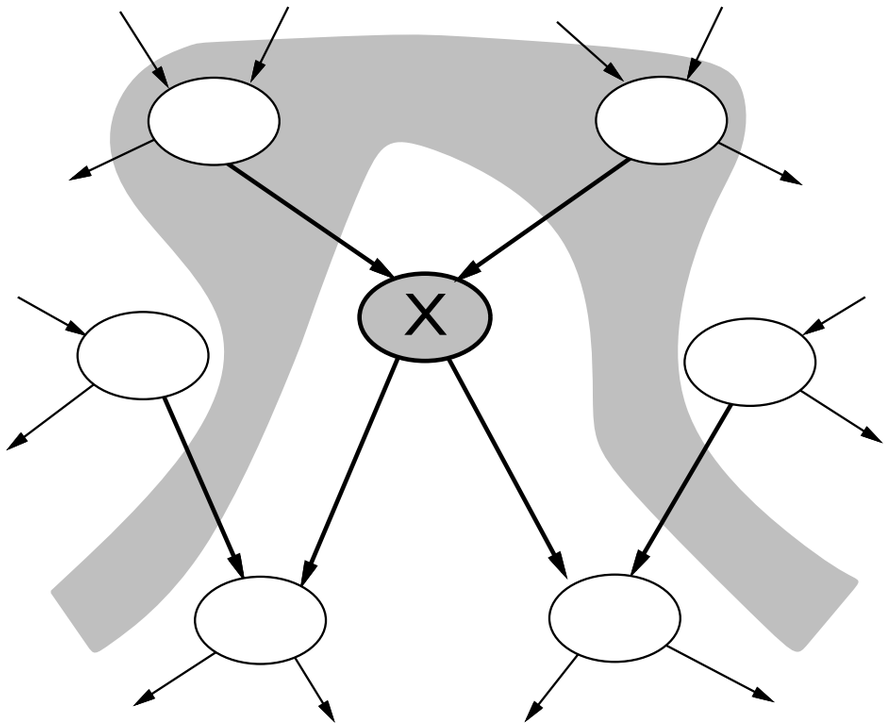
Generate next state by sampling one variable given Markov blanket
Sample each variable in turn, keeping evidence fixed

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 $\mathbf{e}$

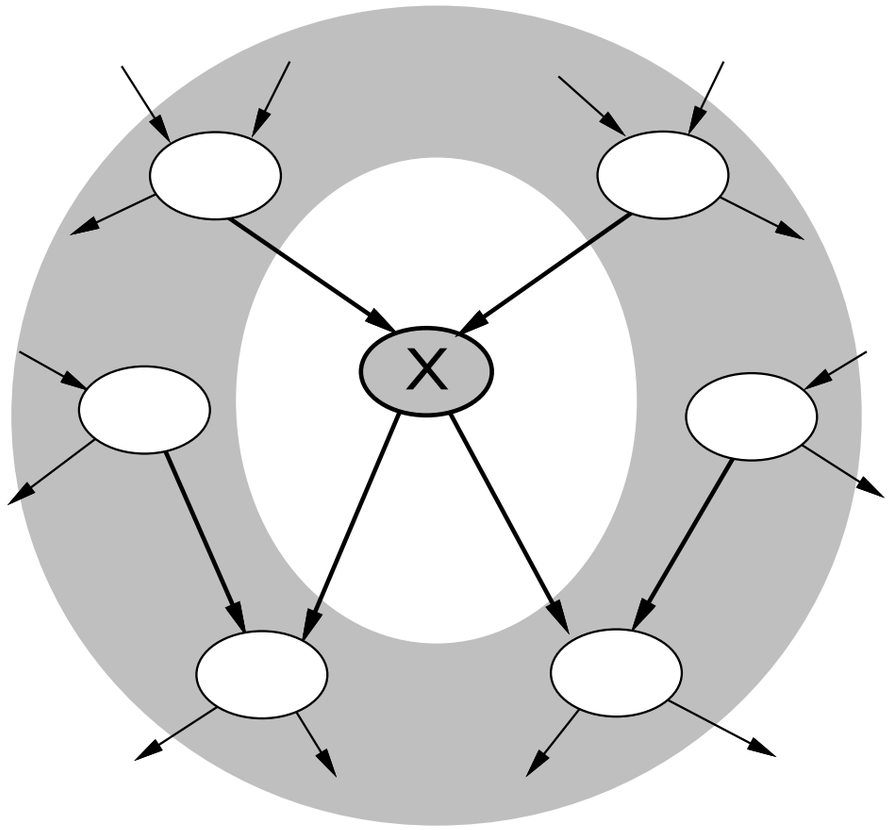
  initialize $\mathbf{x}$ with random values for the variables in $\mathbf{Y}$
  for $j = 1$ to N **do**
    for each Z_i in $\mathbf{Z}$ **do**
      sample the value of Z_i in $\mathbf{x}$ from $\mathbf{P}(Z_i|mb(Z_i))$
        given the values of $MB(Z_i)$ in $\mathbf{x}$
      $\mathbf{N}[x] \leftarrow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



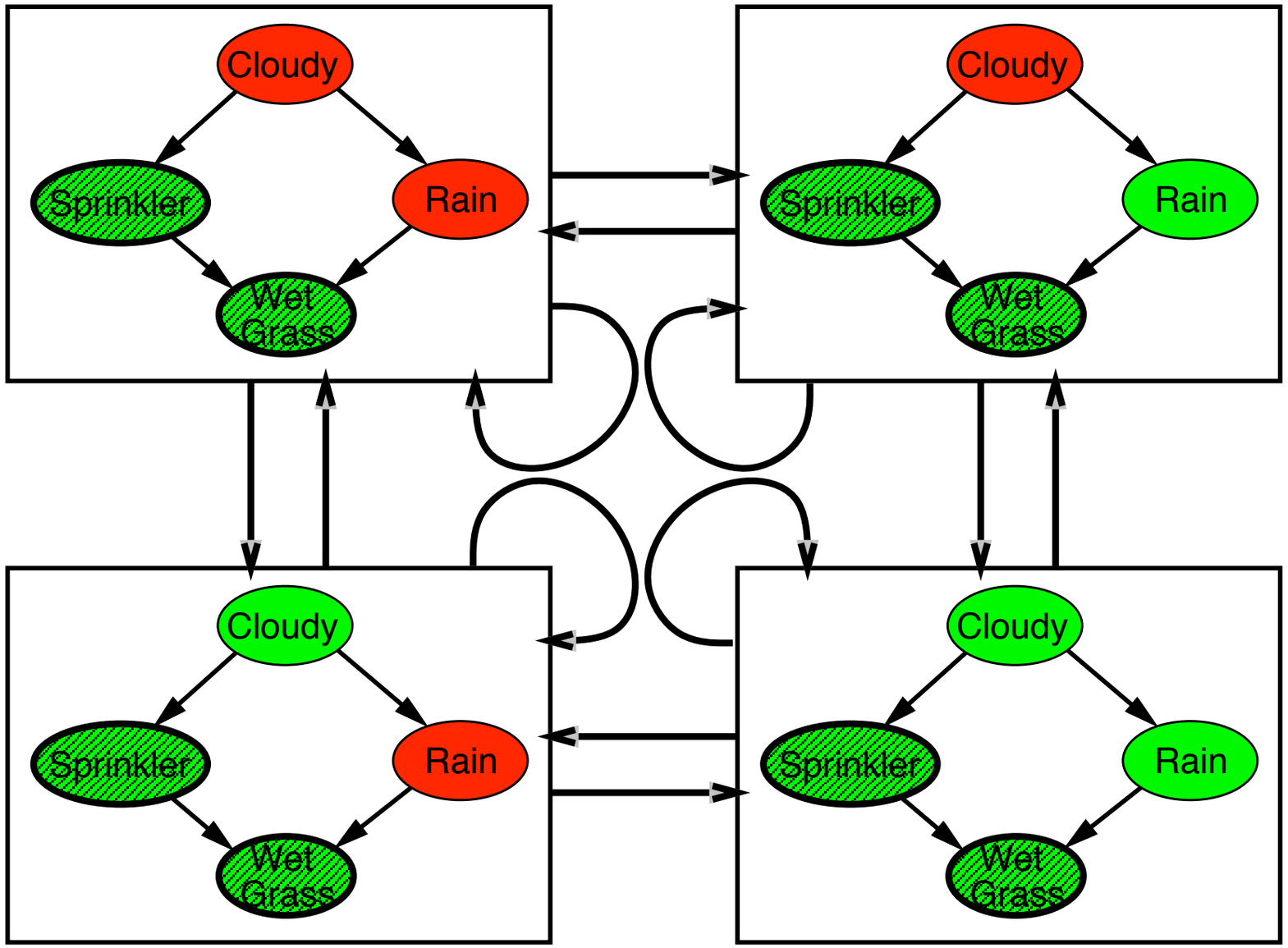
- A node X is conditionally independent of its non-descendants given its parents



- 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 $Rain = false$

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

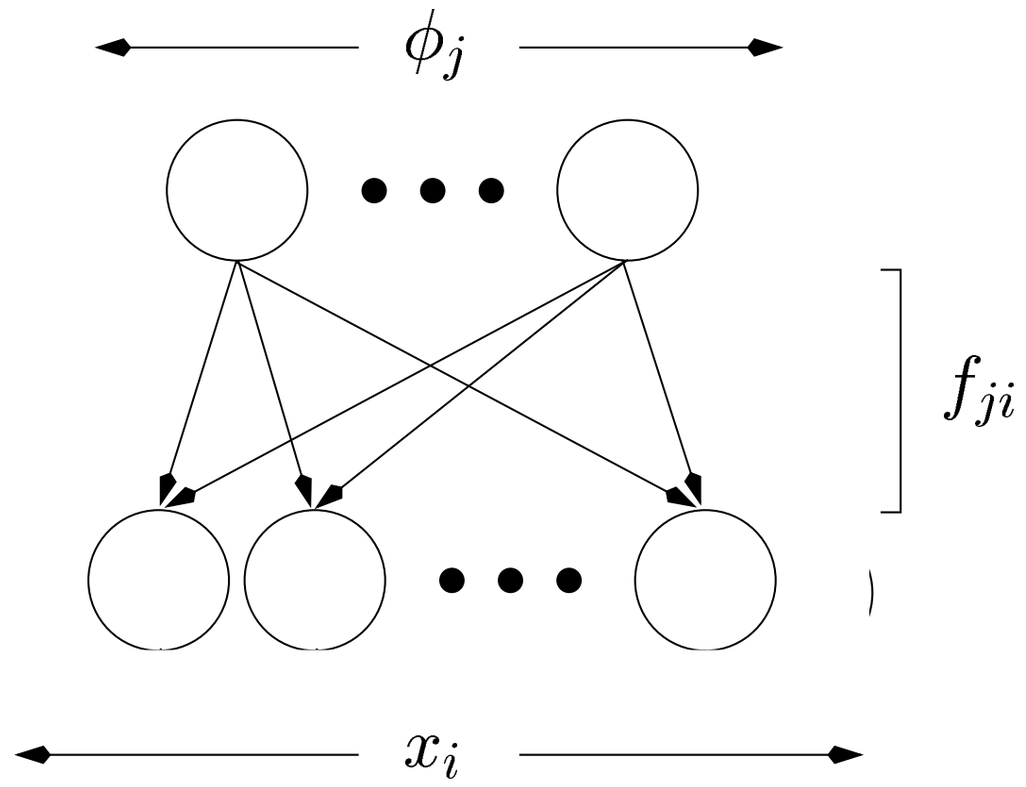
Theorem: chain approaches stationary distribution:
    long-run fraction of time spent in each state is exactly
    proportional to its posterior probability

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 ϕj 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:

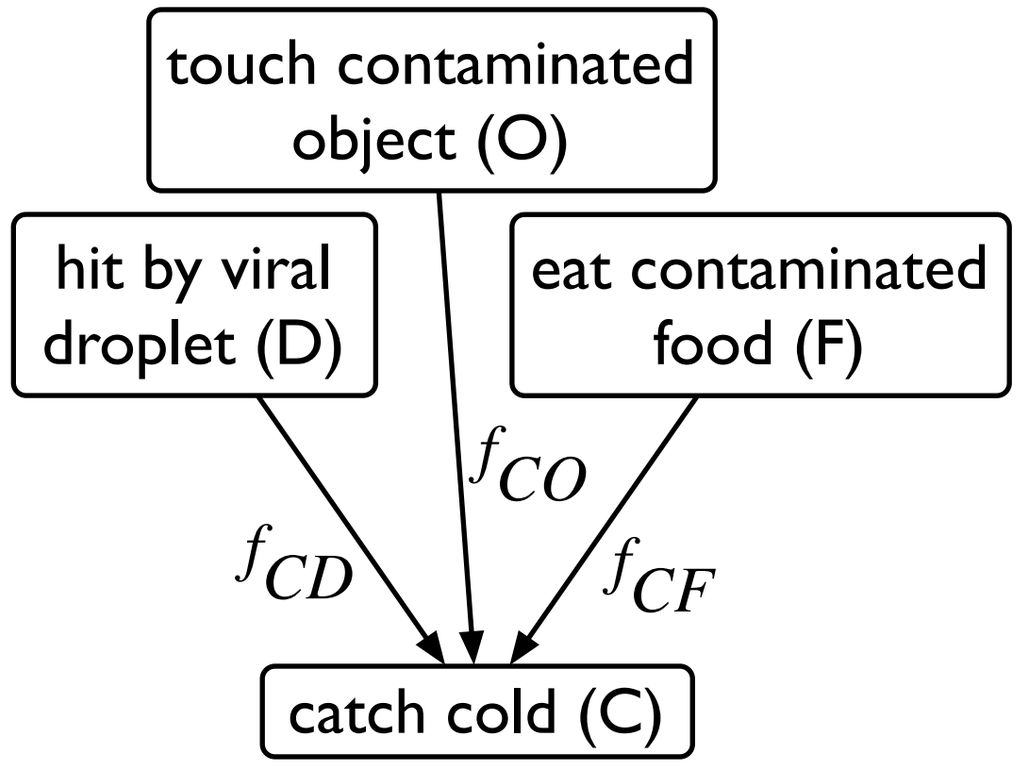
$$f = 1/(1 + \exp(-w))$$



Beyond tables: modeling causal relationships using Noisy-OR

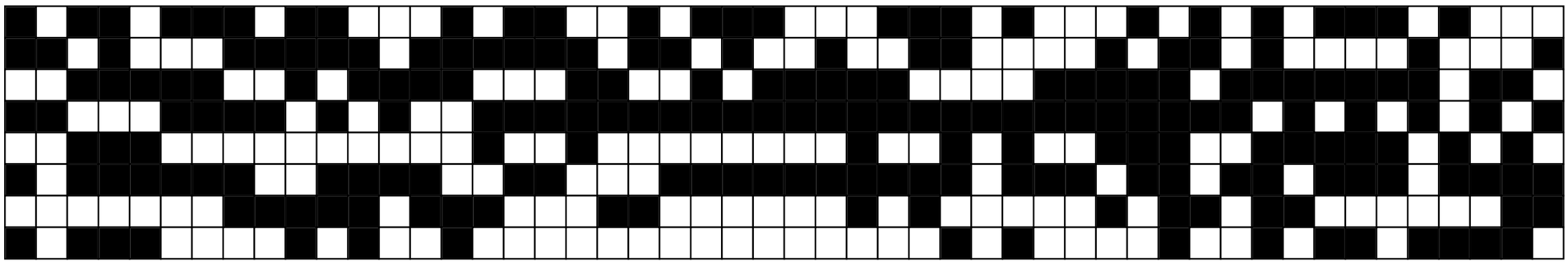
- We assume each cause C_j can produce effect E_i with probability f_{ij}.
- The noisy-OR model assumes the parent causes of effect E_i contribute independently.
- The probability that none of them caused effect E_i is simply the product of the probabilities that each one *did not* cause E_i.
- The probability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

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

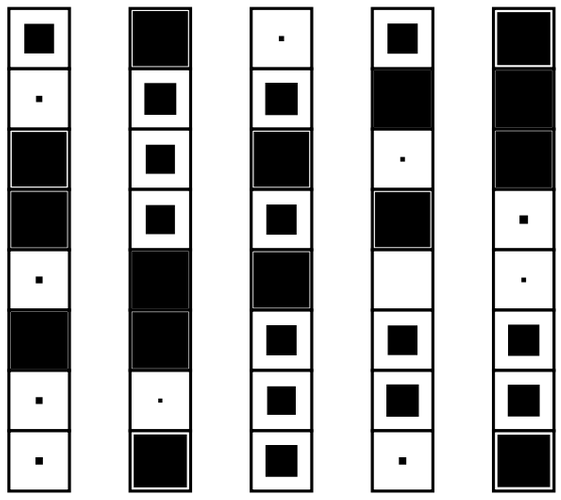


$$P(C|D,O,F) =$$
$$1 - (1 - f_{CD})(1 - f_{CO})(1 - f_{CF})$$

The data: a set of stochastic binary patterns



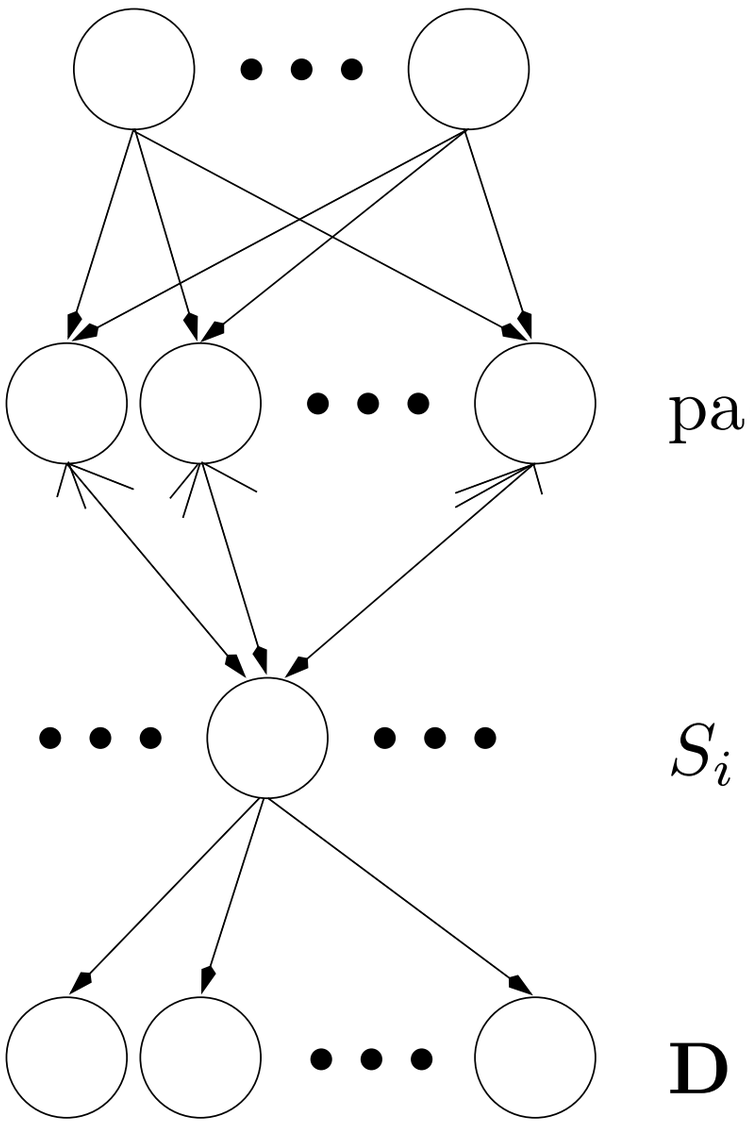
Each column is a distinct eight-dimensional binary feature.



true hidden causes of the data

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:

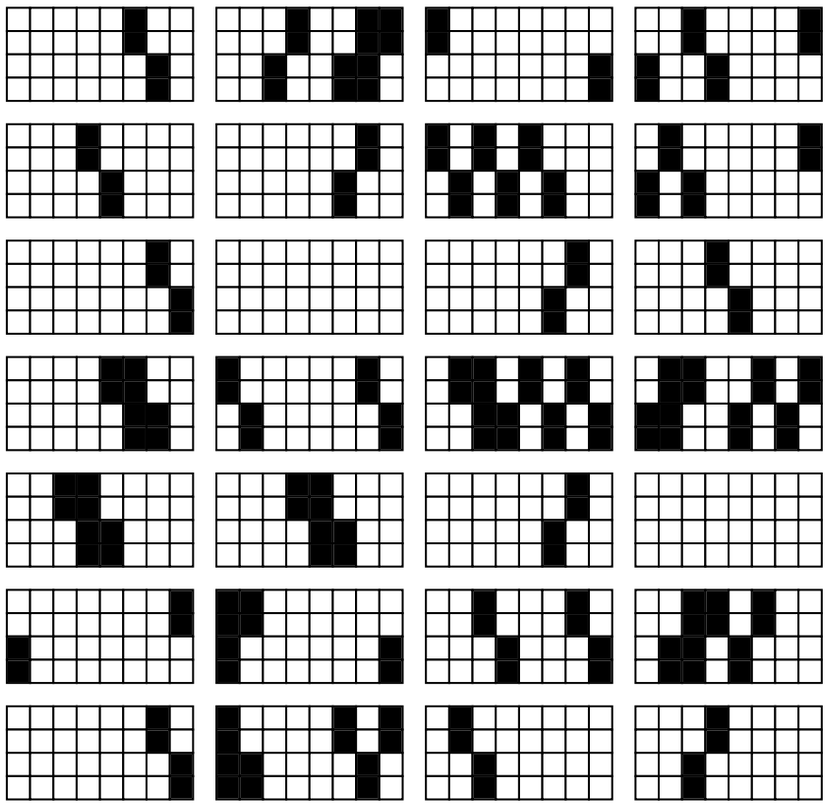
$$P(S_i|\mathrm{pa}(S_i), \mathbf{W}) = \begin{cases} h(\sum_j S_j w_{ji}) & \text{if } S_i = 1 \\ 1 - h(\sum_j S_j w_{ji}) & \text{if } S_i = 0 \end{cases}$$

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

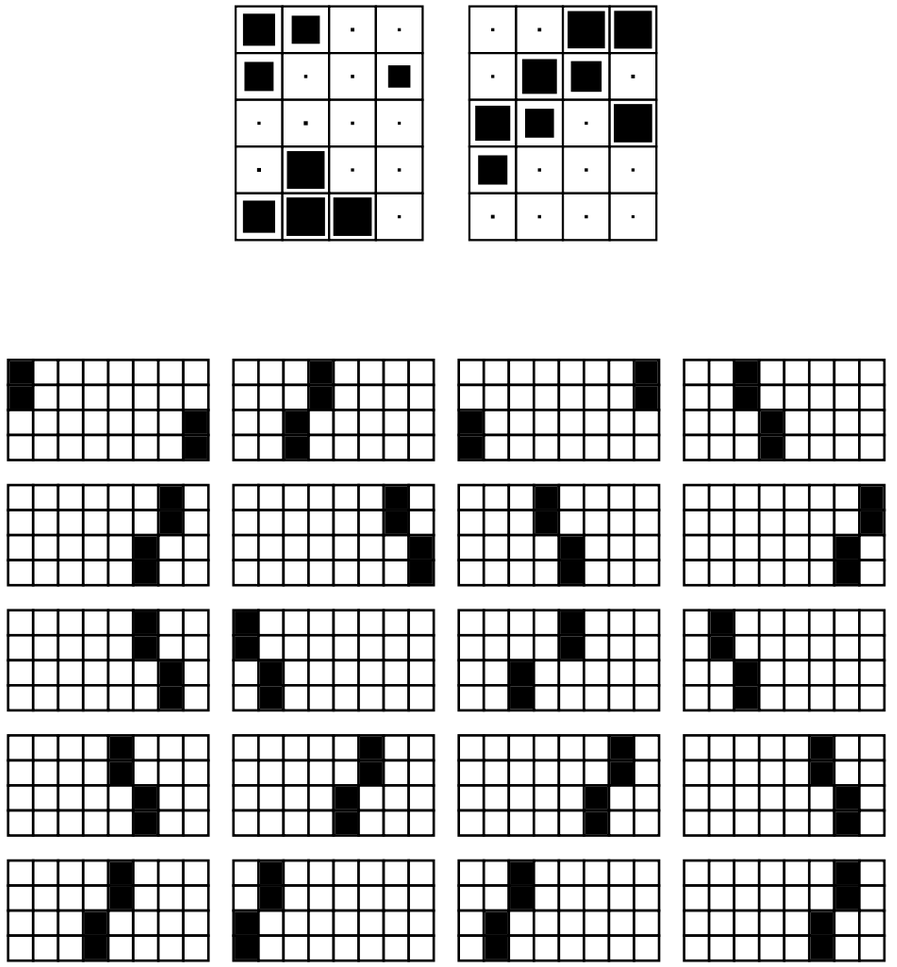
Main points:

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

The Shifter Problem



Shift patterns



weights of a 32-20-2 network after learning

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

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

- 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

Network Interpretation of the Gibbs Sampling Equation

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

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

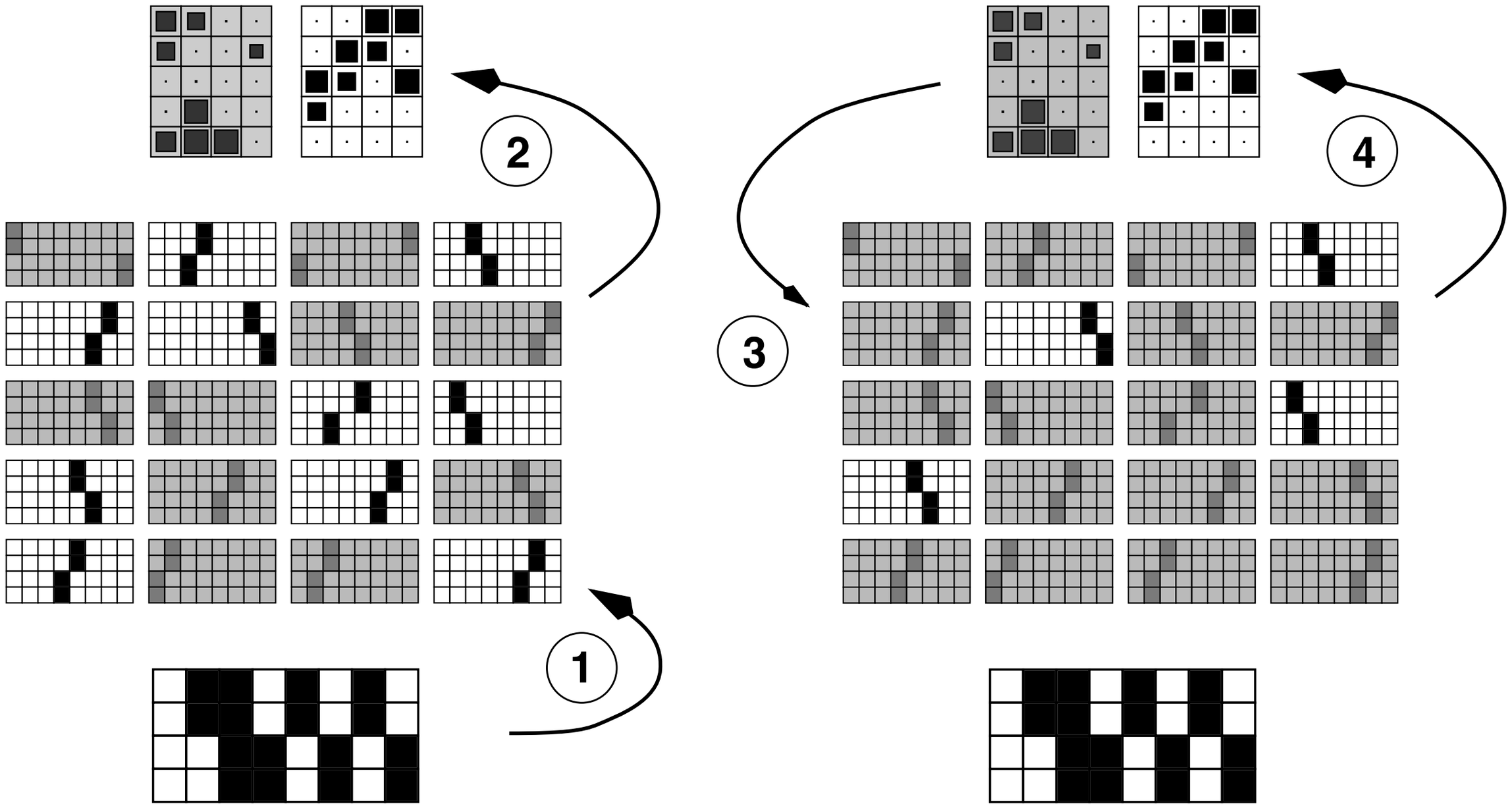
- u_i is the causal input to S_i, $u_i = \sum_k S_k w_{ki}$
- δ_j specifies the change in u_j for a change in S_i, $\delta_{ij} = +S_j w_{ij}$ if $S_i = 0$, or $-S_j w_{ij}$ if $S_i = 1$

Interpretation of Gibbs sampling equation

Gibbs equation can be interpreted as: *feedback* $+ \sum$ *feedforward*.

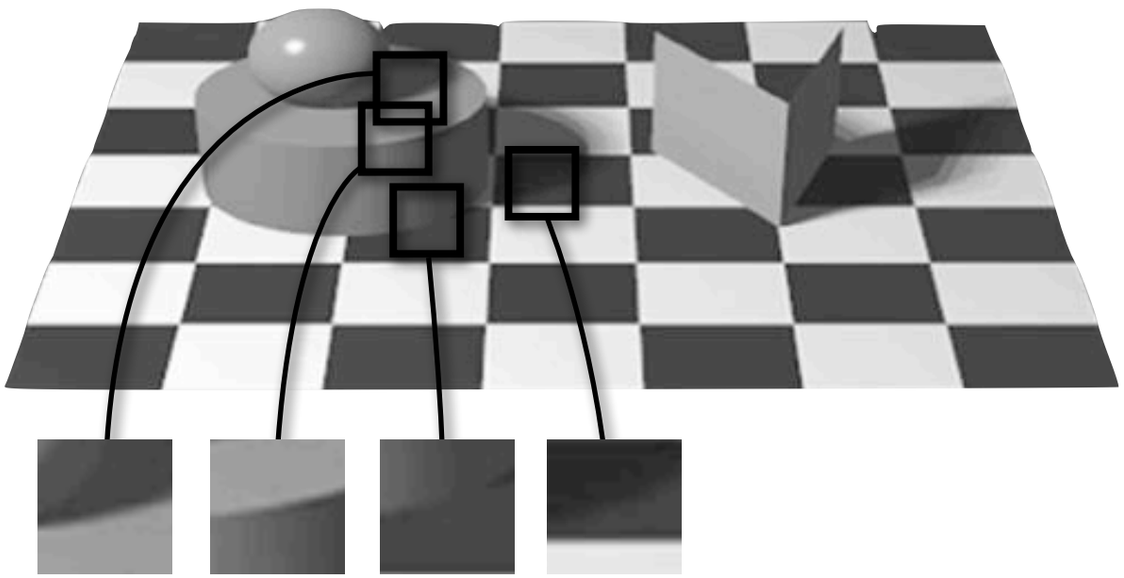
- *feedback*: how consistent is S_i with current causes?
- $\sum$ *feedforward*: how likely is S_i a cause of its children?
- feedback allows the lower level units to use information only computable at higher levels
- feedback determines state when the feedforward input is ambiguous

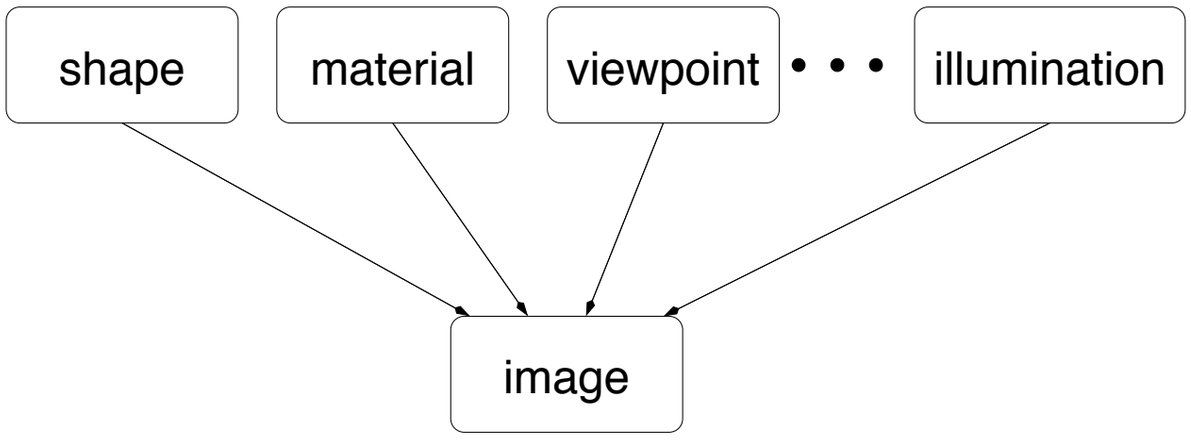
Gibbs sampling: feedback disambiguates lower-level states



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

Difficulties with identifying the causal structure

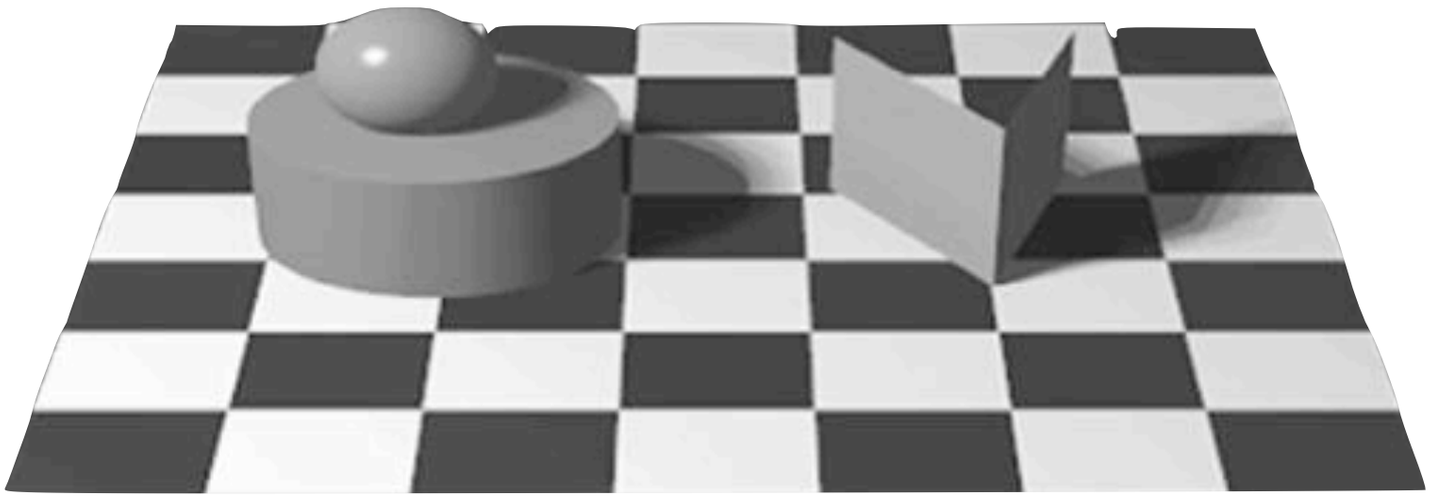


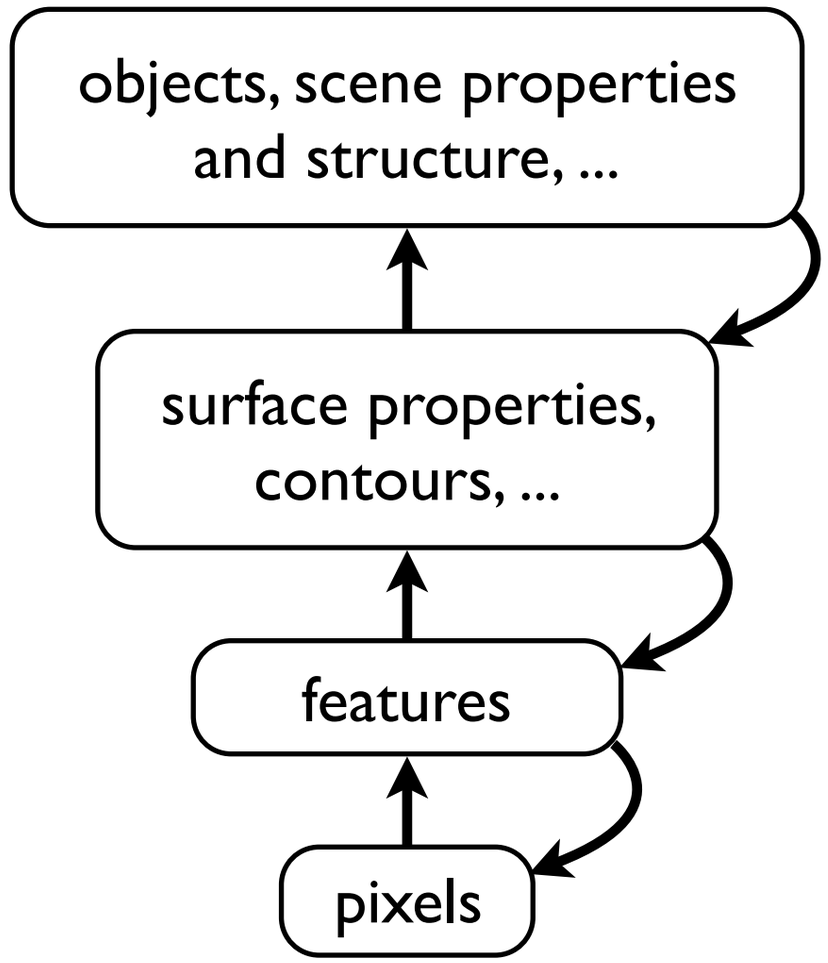


- Space of S, C is huge
- If we define $P(I|S,C)$, must be able to invert it or search it
- Need efficient algorithms
- Many unknowns: identity of objects, types of scene elements, illuminations
- Might never have encountered some structures
- Is it even the right approach?
- Can we solve a simple case?

Motivation: learn hierarchical, context dependent representations

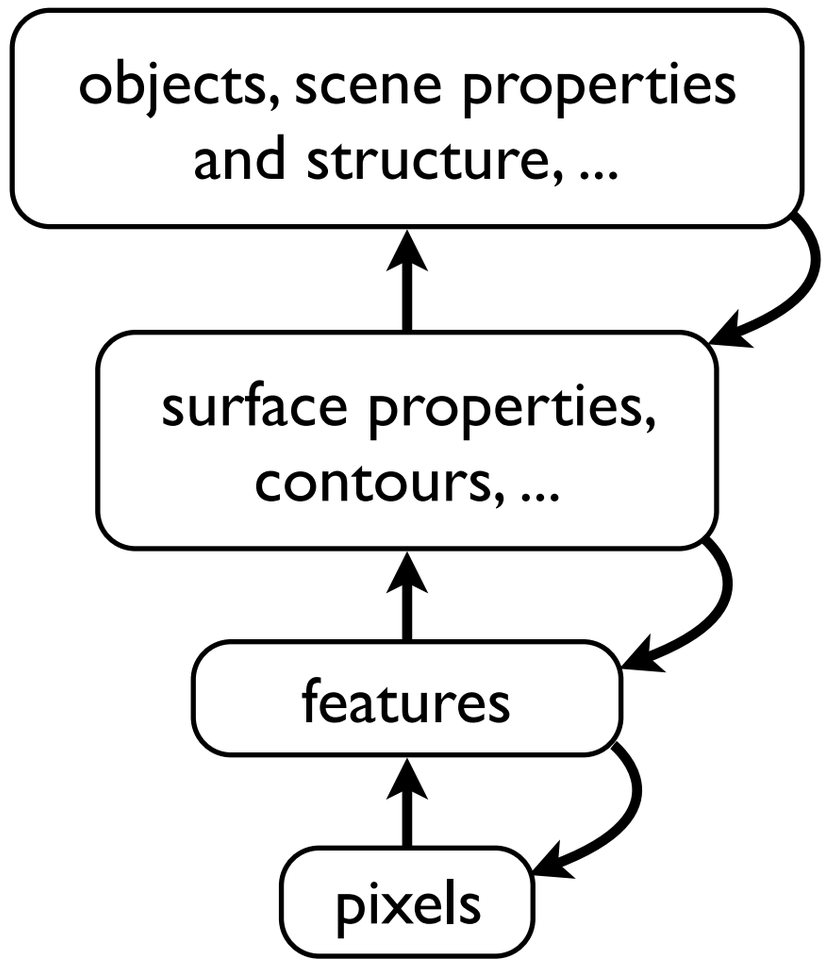
- many real-world patterns are hierarchical in structure
- interpretation of patterns depends on context
- essential for complex recognition tasks





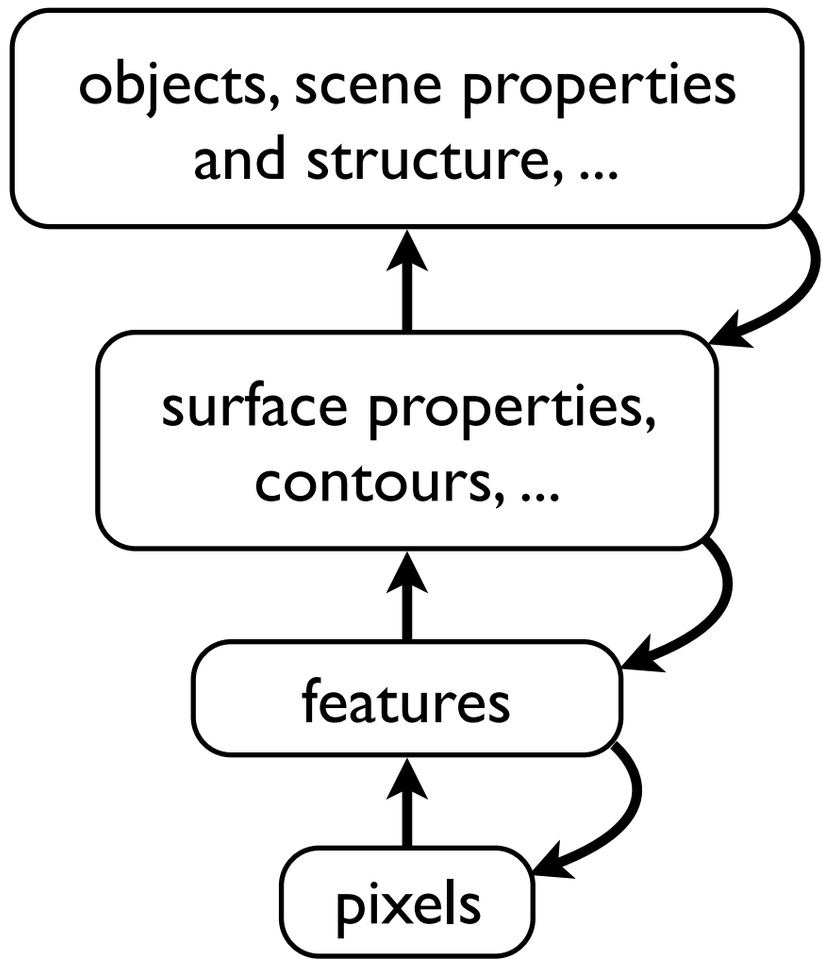
Motivation: hierarchical, context dependent representations

- many real-world patterns are hierarchical in structure
- interpretation of patterns depends on context
- essential for complex recognition tasks



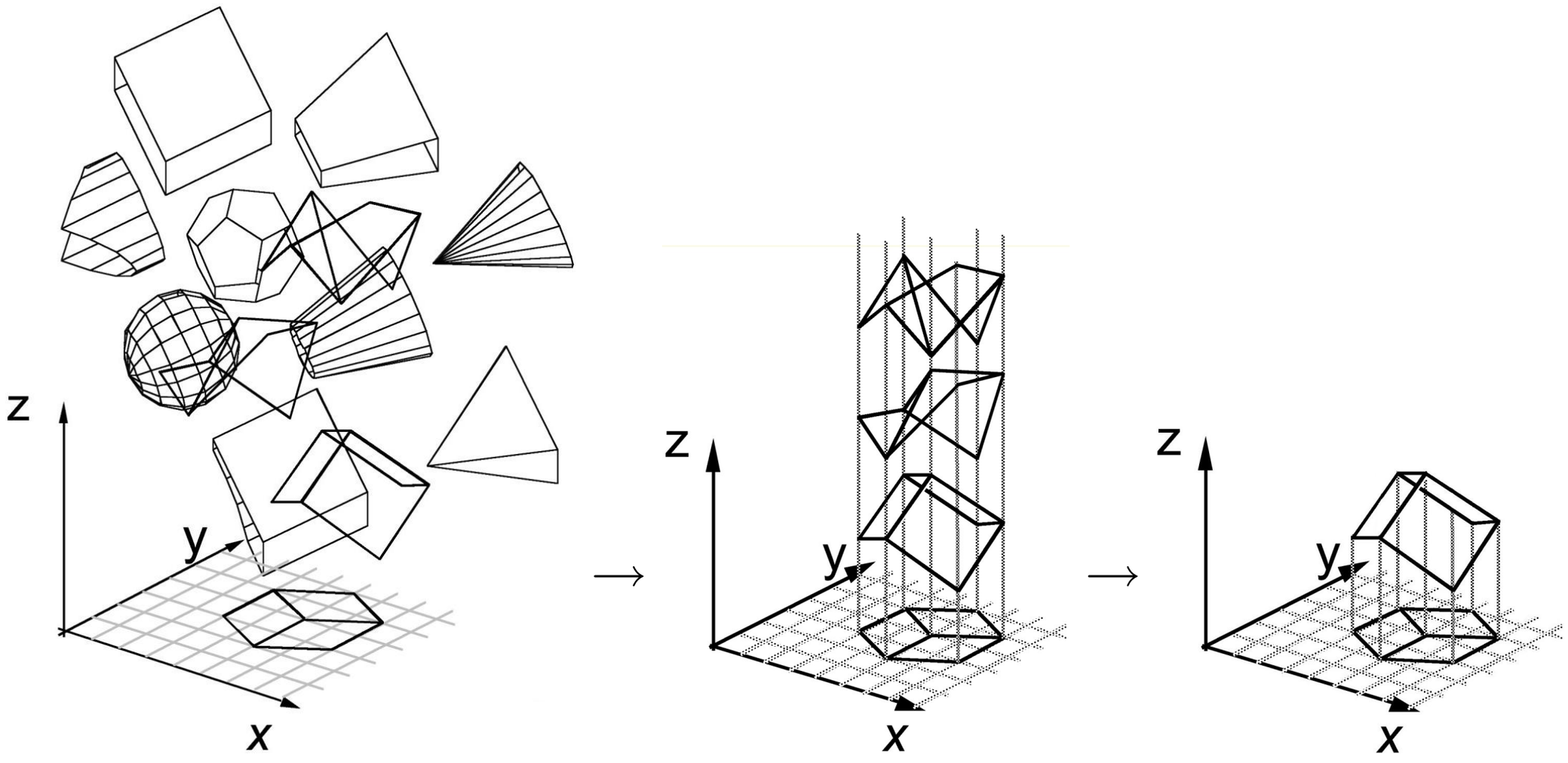
Motivation: hierarchical, context dependent representations

- many real-world patterns are hierarchical in structure
- interpretation of patterns depends on context
- essential for complex recognition tasks

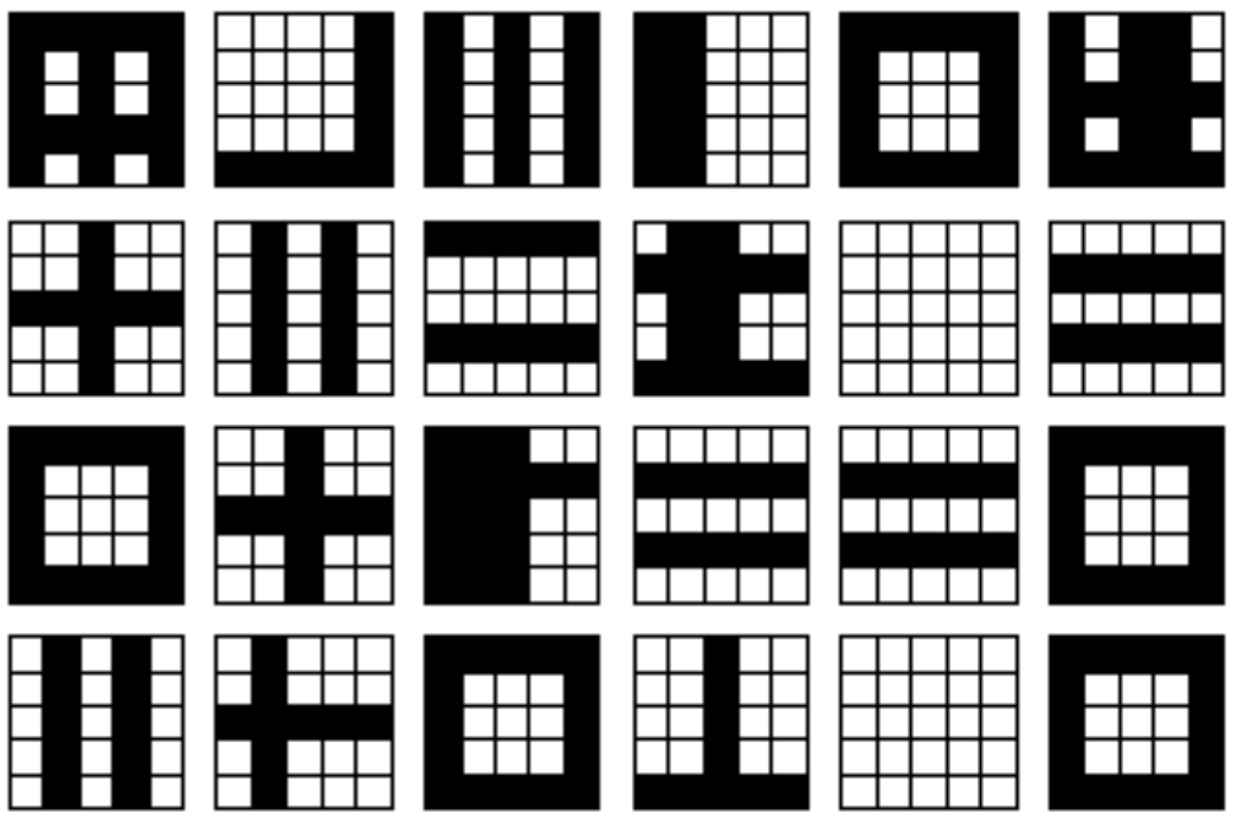


The inferred explanation is the most probable scene

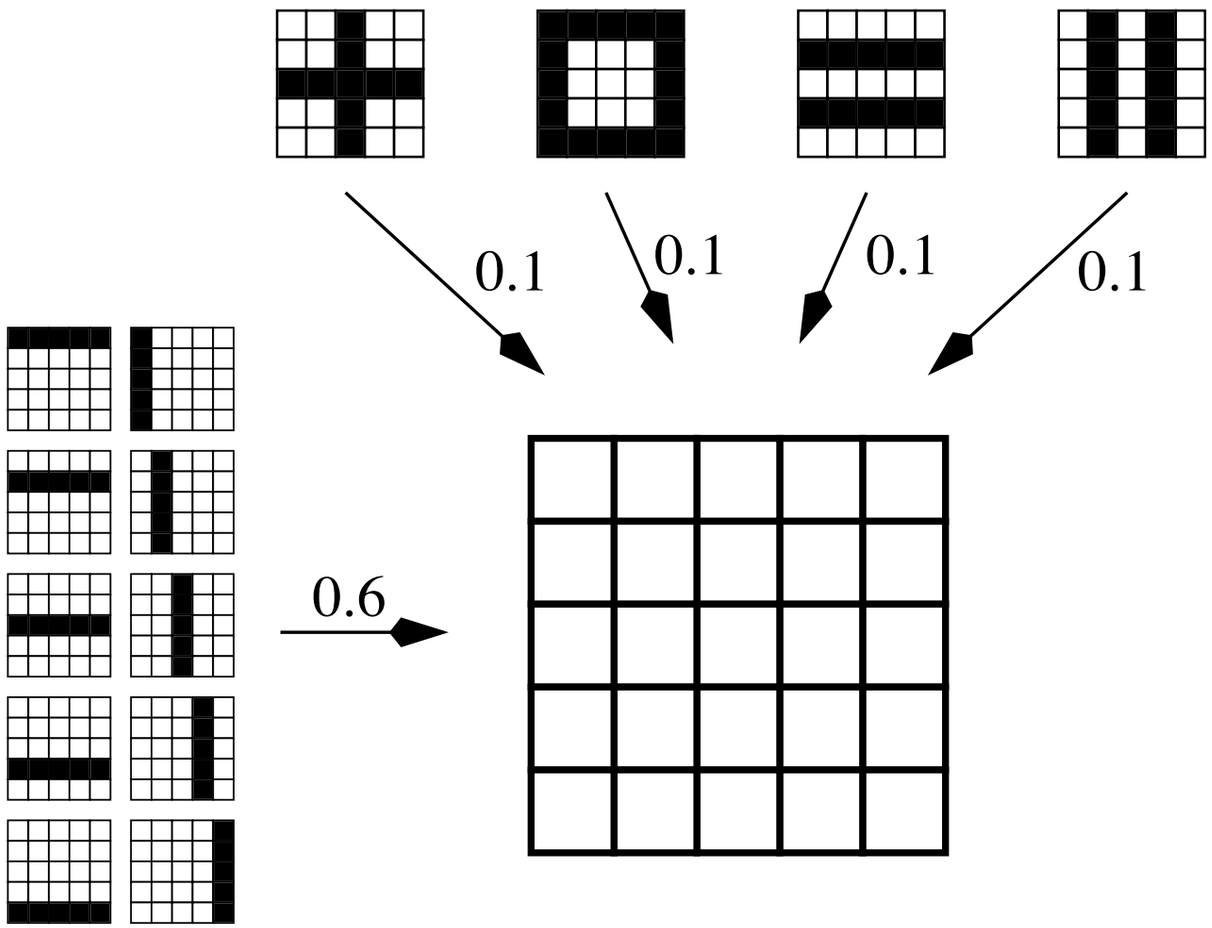
$$p(\hat{S}|I,C) = \arg\max_{S} \frac{p(I|S,C)p(S|C)}{p(I|C)}$$



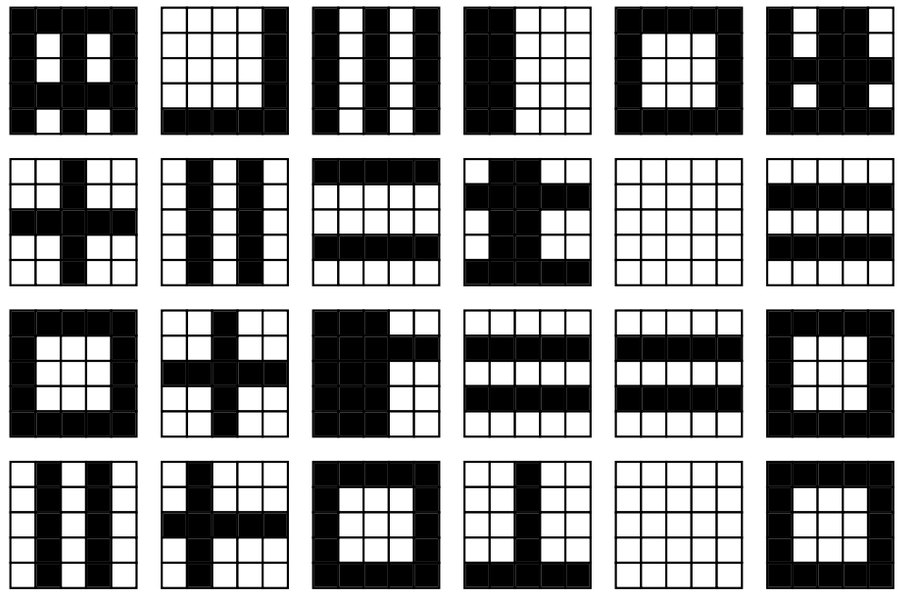
Toy Lines Problem: *Is it a pattern or a collection of features?*



The higher-order lines problem



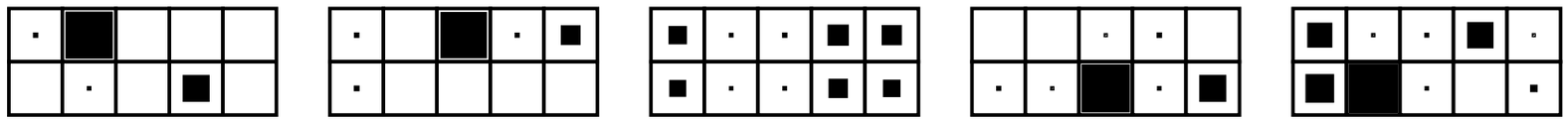
The true generative model

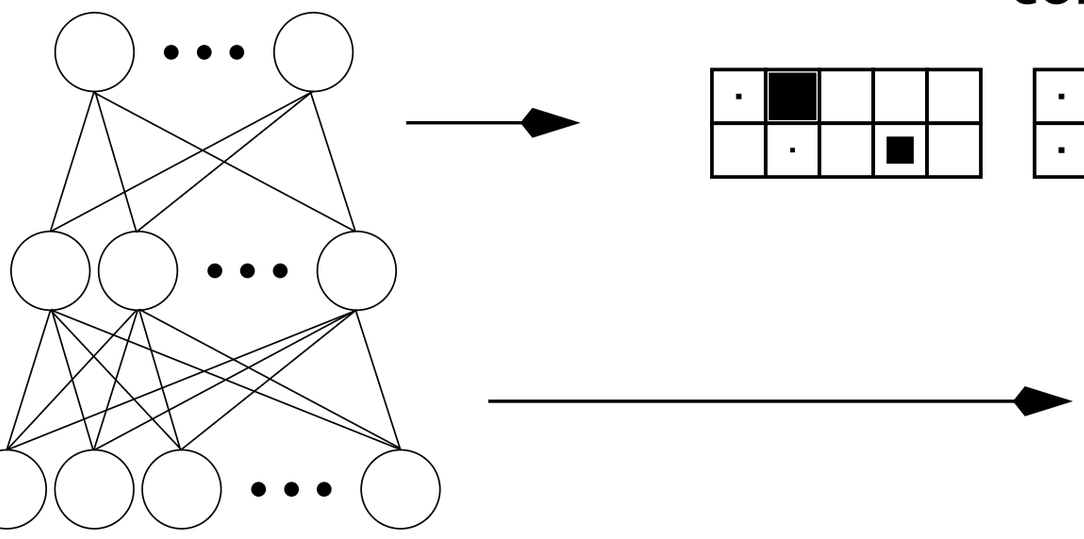


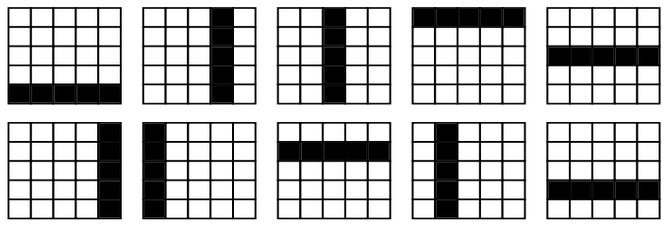
Patterns sampled from the model

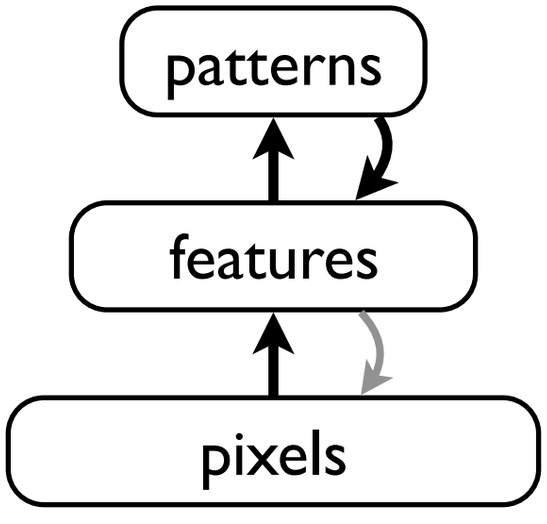
Weights in a 25-10-5 belief network after learning

2. The second layer learns patterns that are combinations of the first layer features

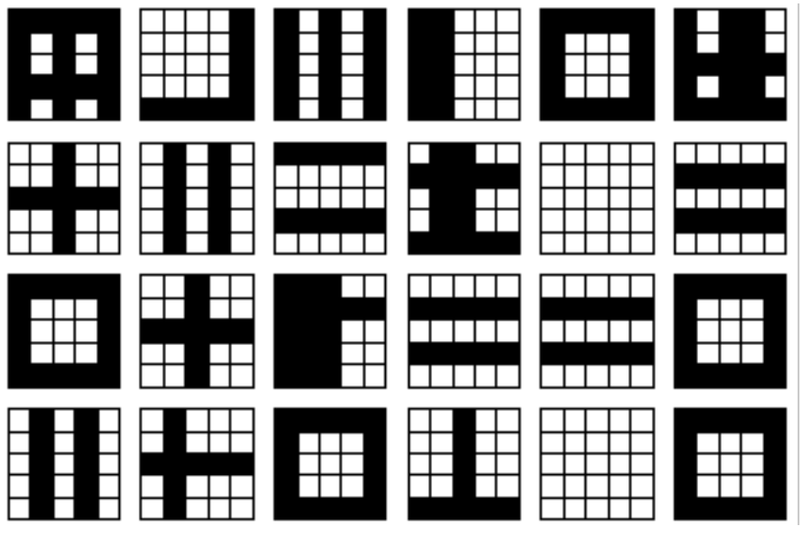






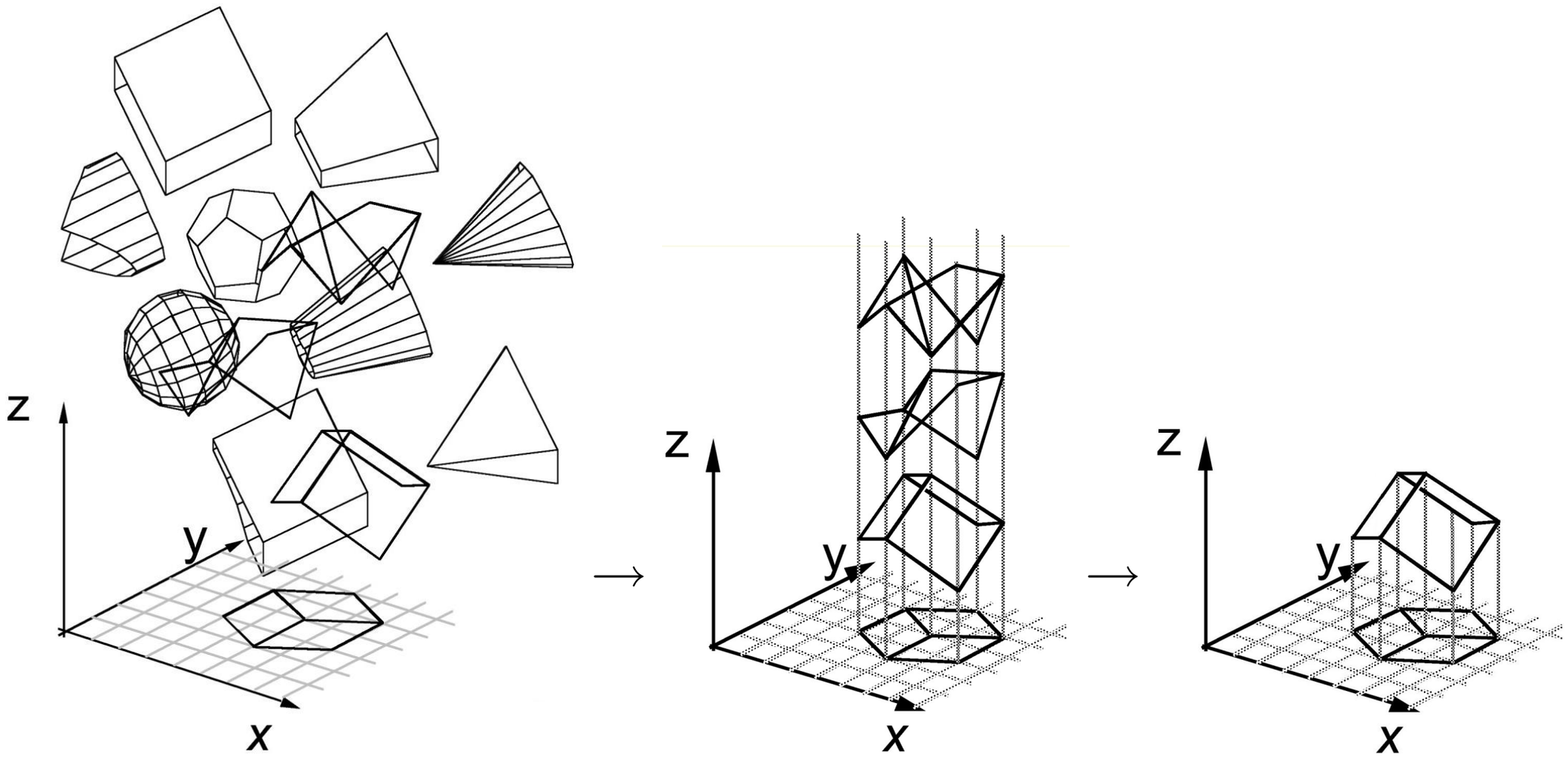


1. The first layer of weights learn line features are combinations of pixels.



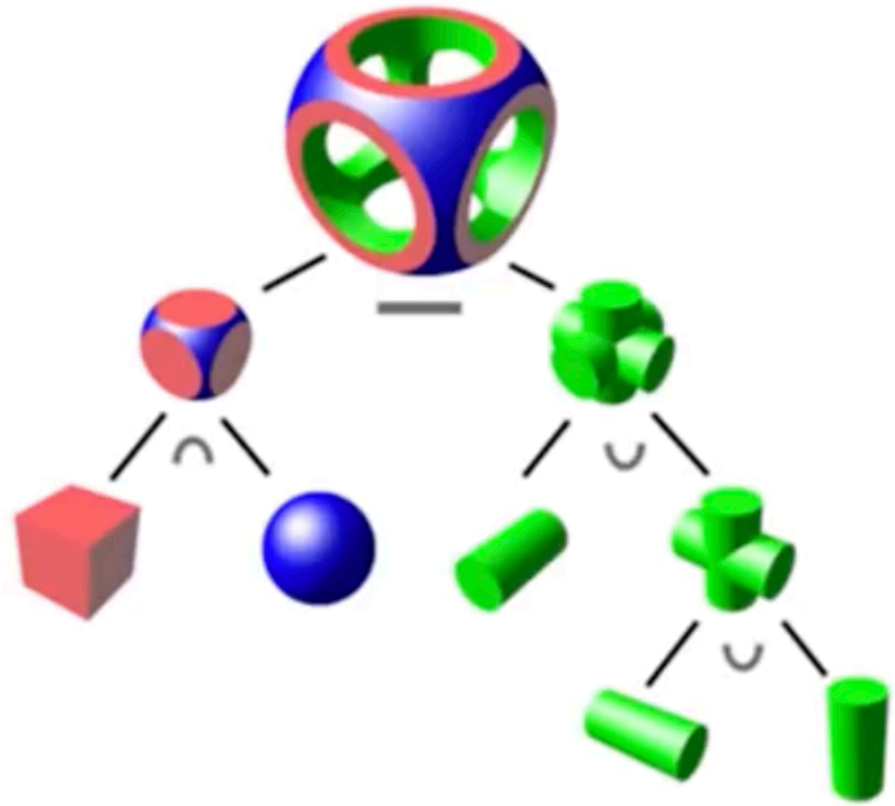
Analysis by Synthesis

$$p(\hat{S}|I,C) = \arg\max_{S} \frac{p(I|S,C)p(S|C)}{p(I|C)}$$



Inverse Constructive Solid Geometry

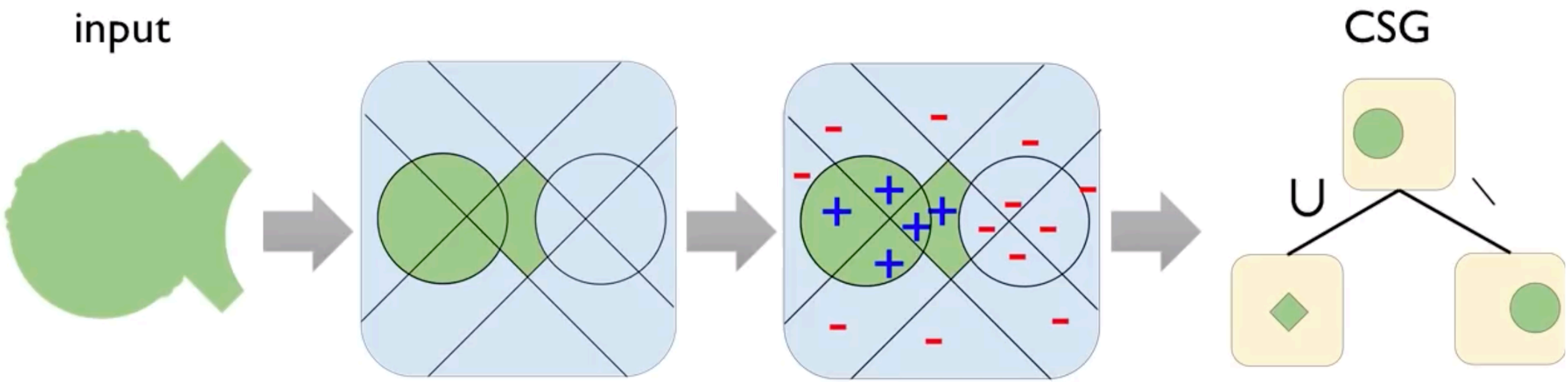
input mesh



CSG tree

MIT Computational Fabrication Group

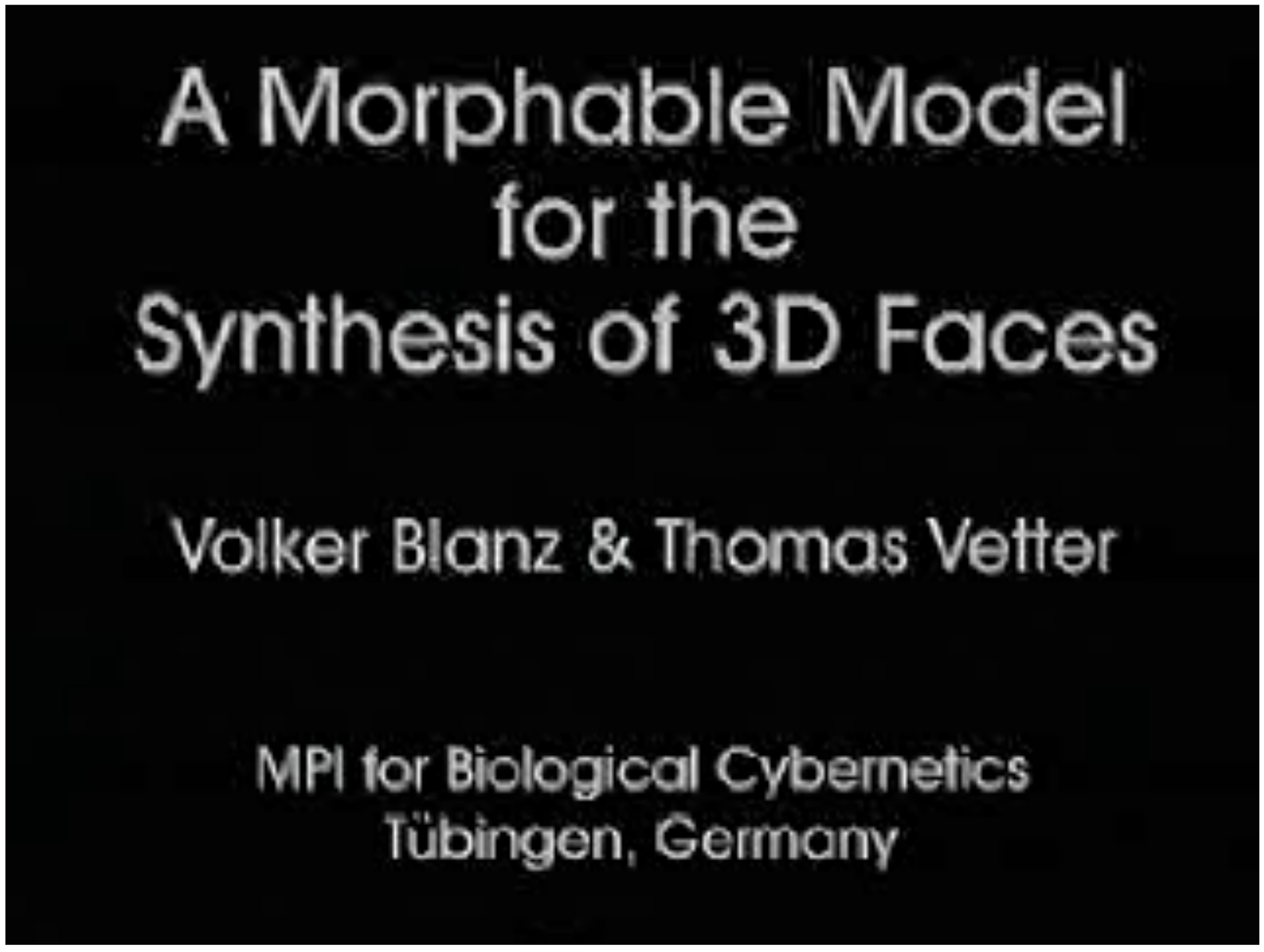
Inverse Constructive Solid Geometry



Inverse Constructive Solid Geometry

	original model	1% added noise	2% added noise
input			
result			

Blanz and Vetter (Siggraph, 1999)



$$E_I = \sum_{x,y} \left\| \mathbf{I}_{input}(x,y) - \mathbf{I}_{model}(x,y) \right\|^2.$$

Blanz and Vetter (Siggraph, 1999)

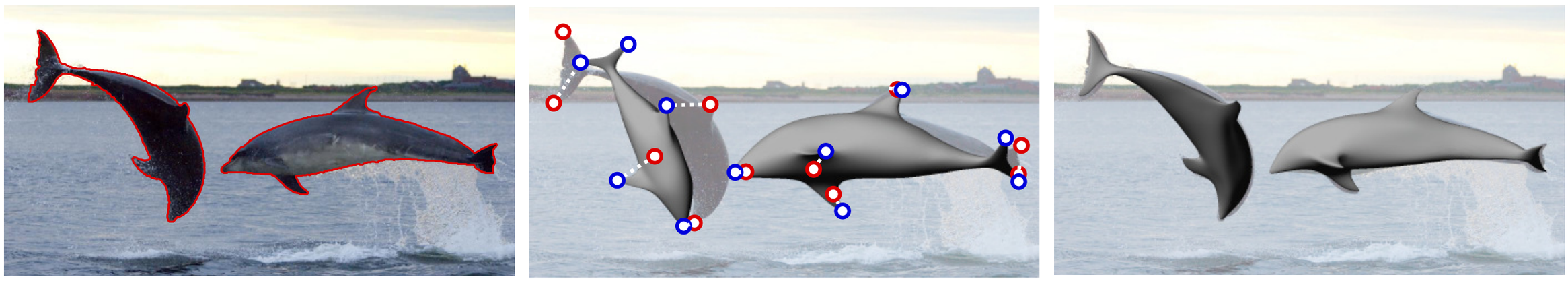
$$E_I = \sum_{x,y} \left\| \mathbf{I}_{input}(x,y) - \mathbf{I}_{model}(x,y) \right\|^2.$$

What Shape are Dolphins? Building 3D Morphable Models from 2D Images

Thomas J. Cashman and Andrew W. Fitzgibbon, *Senior Member, IEEE*

Abstract—3D morphable models are low-dimensional parametrizations of 3D object classes which provide a powerful means of associating 3D geometry to 2D images. However, morphable models are currently generated from 3D scans, so for general object classes such as animals they are economically and practically infeasible. We show that, given a small amount of user interaction (little more than that required to build a conventional morphable model), there is enough information in a collection of 2D pictures of certain object classes to generate a full 3D morphable model, even in the absence of surface texture. The key restriction is that the object class should not be strongly articulated, and that a very rough rigid model should be provided as an initial estimate of the 'mean shape'.

The model representation is a linear combination of subdivision surfaces, which we fit to image silhouettes and any identifiable key points using a novel combined continuous-discrete optimization strategy. Results are demonstrated on several natural object classes, and show that models of rather high quality can be obtained from this limited information.



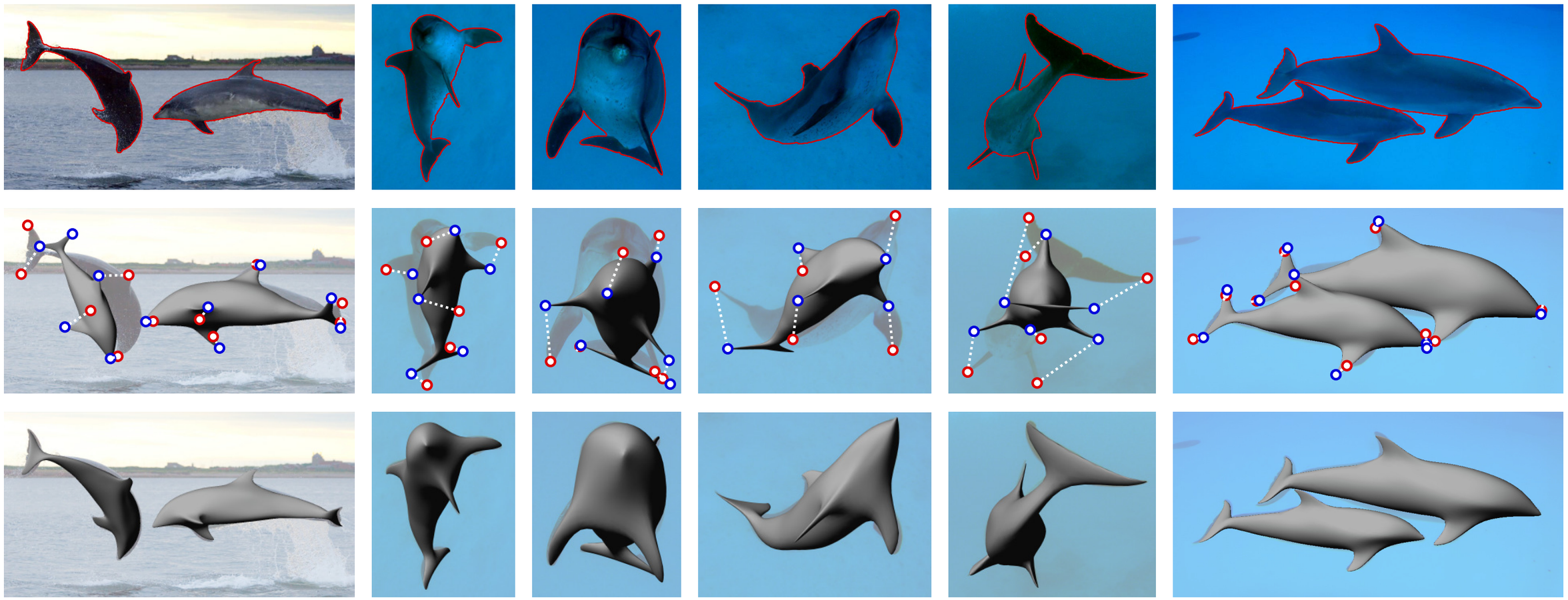


Fig. 1. Images showing eight dolphins out of 32, from which we build an 8-parameter morphable model. Top: input images with silhouette annotations. Middle: The rigid dolphin prototype in initial position for our optimization, showing user-provided point constraints (blue), each of which corresponds to a point in the image (shown in red). Bottom: final morphable model reconstruction overlaid on the input images.

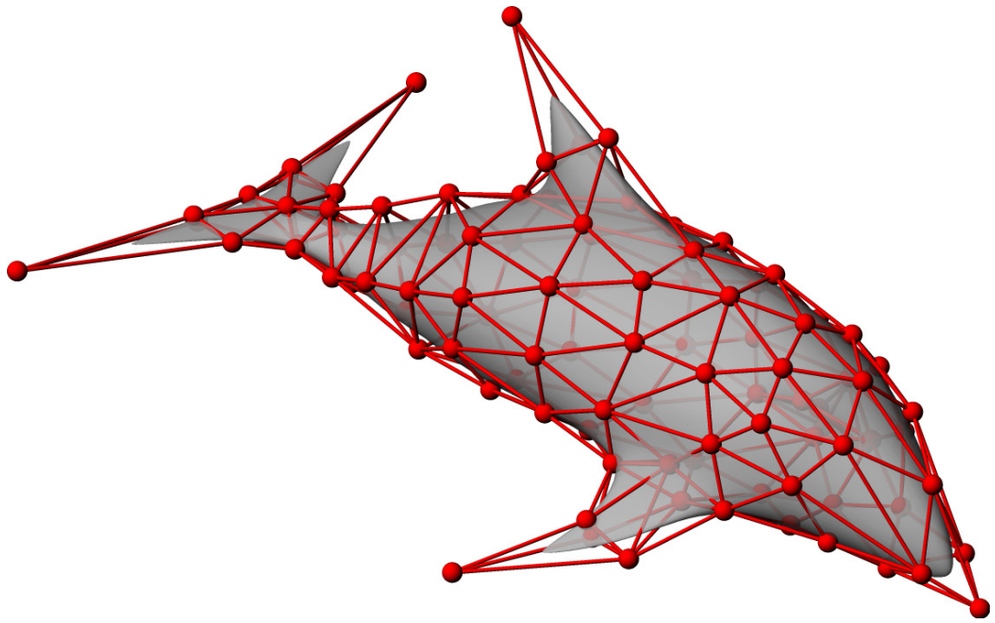


Fig. 2. A rigid template triangulation (red lines) which we use to initialize our dolphin model. Note that this triangulation defines only the mesh topology, and is defined just once per object class, *not* per image. The Loop subdivision surface defined by combining this triangulation with the vertex positions defined by the red spheres is shown in gray.

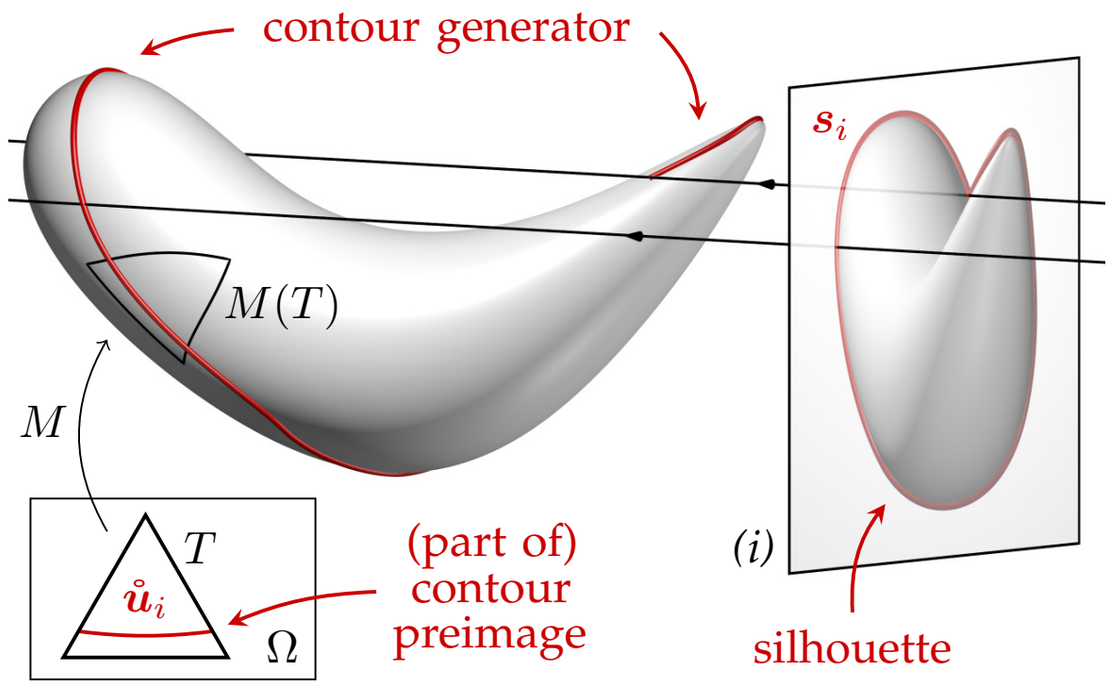
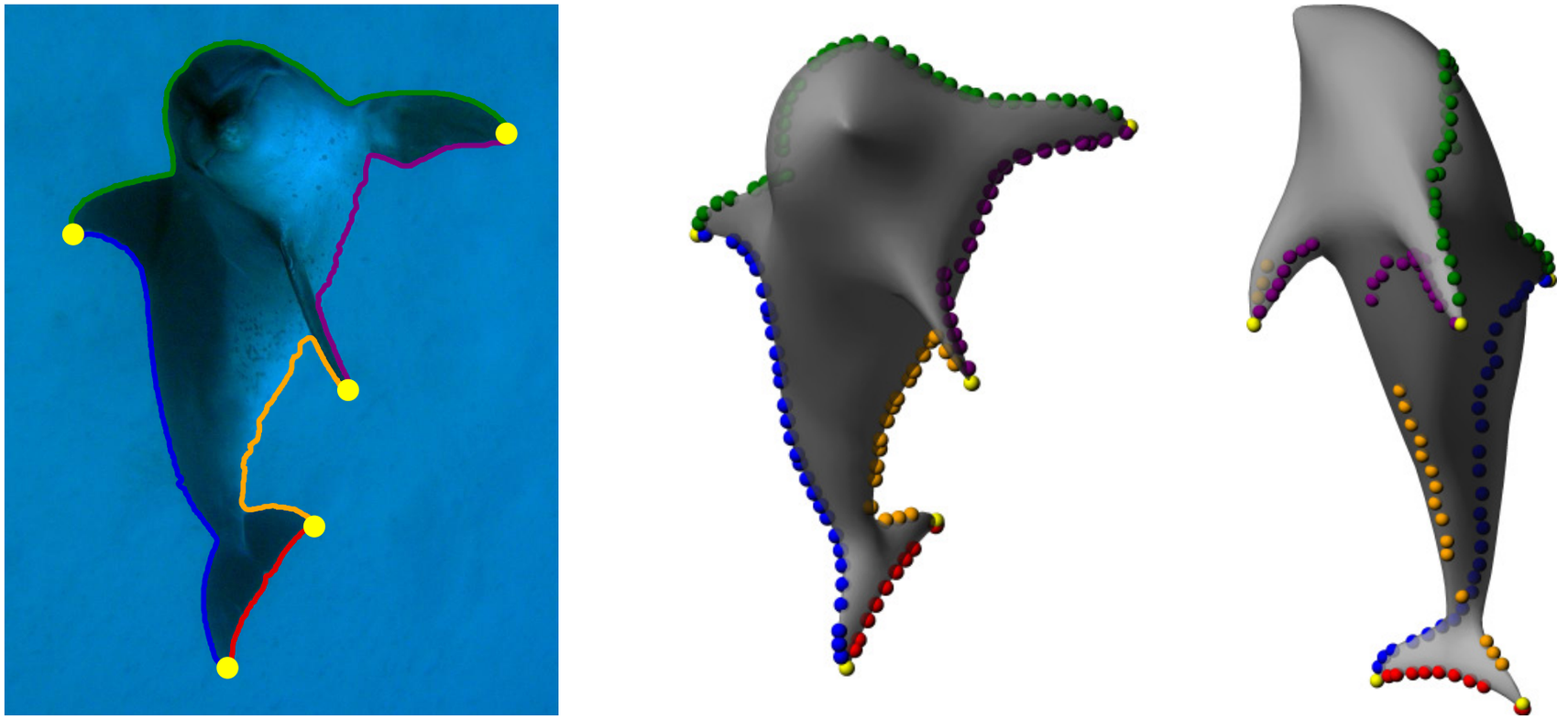


Fig. 3. An image *(i)* of an object with the silhouette $\boldsymbol{s}_i = \{\boldsymbol{s}_{ij}\}_{j=1}^{S_i}$. The contour generator projects to this silhouette in the image. We also show one triangle $T \subset \Omega$, and a part of the contour preimage $\mathring{\boldsymbol{u}}_i$ that appears in T. The contour generator (which is *discontinuous*, in this figure) is the result of using M to evaluate the surface at the contour preimage.

Constraints



(a) Silhouette, with constraints $\boldsymbol{c}_{ik}$ that lie on the silhouette marked in yellow (see Fig. 4b).

(b) Two different views showing the result of solving (14) separately on each coloured section, between the known points $M(\mu_{ik})$ marked in yellow.

Fig. 5. If one or more constraints $\boldsymbol{c}_{ik}$ lie on the silhouette, the global search for contour generators is partitioned into separate problems for non-circular paths. In this example, the constraints shown in (a) split the silhouette into the five sections shown in different colours, which we solve in turn to gain the contour generator solutions in (b).

Energy formulation (negative log probability)

4.4 Complete energy function

In summary, by combining all of the terms (5) to (10), we arrive at the following energy

$$E = \sum_{i=1}^{n} \left(E_i^{\text{sil}} + E_i^{\text{norm}} + E_i^{\text{con}} \right) + \\ + \sum_{i=1}^{n} \left(E_i^{\text{cg}} + E_i^{\text{reg}} \right) + \xi_0^2 E_0^{\text{tp}} + \xi_{\text{def}}^2 \sum_{m=1}^{D} E_m^{\text{tp}}. \tag{11}$$

$$E_i^{\text{sil}} = \frac{1}{2} \sigma_{\text{sil}}^{-2} \sum_{j=1}^{S_i} \left\| \boldsymbol{s}_{ij} - \pi_i \left(M(\mathring{u}_{ij} | \boldsymbol{X}_i) \right) \right\|^2, \tag{5}$$

silhouette as function of angle

$$E_i^{\text{con}} = \frac{1}{2} \sigma_{\text{con}}^{-2} \sum_{k=1}^{K_i} \left\| \boldsymbol{c}_{ik} - \pi_i \left(M(\mathring{\mu}_{ik} | \boldsymbol{X}_i) \right) \right\|^2. \tag{6}$$

user-specified constraints

$$E_i^{\text{norm}} = \frac{1}{2} \sigma_{\text{norm}}^{-2} \sum_{j=1}^{S_i} \left\| \begin{bmatrix} \boldsymbol{n}_{ij} \\ 0 \end{bmatrix} - \nu \left(\boldsymbol{R}_i N(\mathring{u}_{ij} | \boldsymbol{X}_i) \right) \right\|^2 \tag{7}$$

surface at $\mathring{u}_{ij}$ must be normal to viewing direction

$$E_m^{\text{tp}} = \frac{\bar{\lambda}^2}{2} \int_{\Omega} \| M_{xx}(\mathring{u} | \boldsymbol{B}_m) \|^2 + 2 \| M_{xy}(\mathring{u} | \boldsymbol{B}_m) \|^2 + \\ \| M_{yy}(\mathring{u} | \boldsymbol{B}_m) \|^2 \, \mathrm{d}\mathring{u}. \tag{8}$$

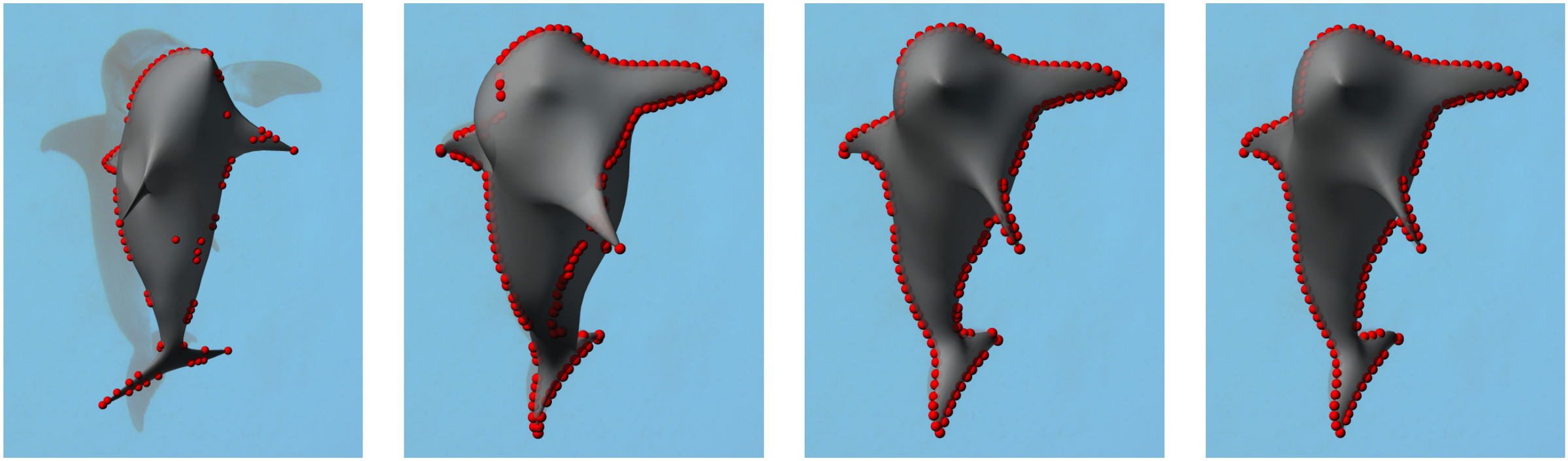
enforce smoothness

$$E_i^{\text{reg}} = \beta \sum_{m=1}^{D} \alpha_{im}^2. \tag{9}$$

$$E_i^{\text{cg}} = \gamma \sum_{j=1}^{S_i} \tau(d(\mathring{u}_{ij}, \mathring{u}_{i,j+1})) \tag{10}$$

allow select discontinuities

Optimizing the morphable model



(a) Initial estimate.

(b) Only continuous local optimization, as described in §5.1.

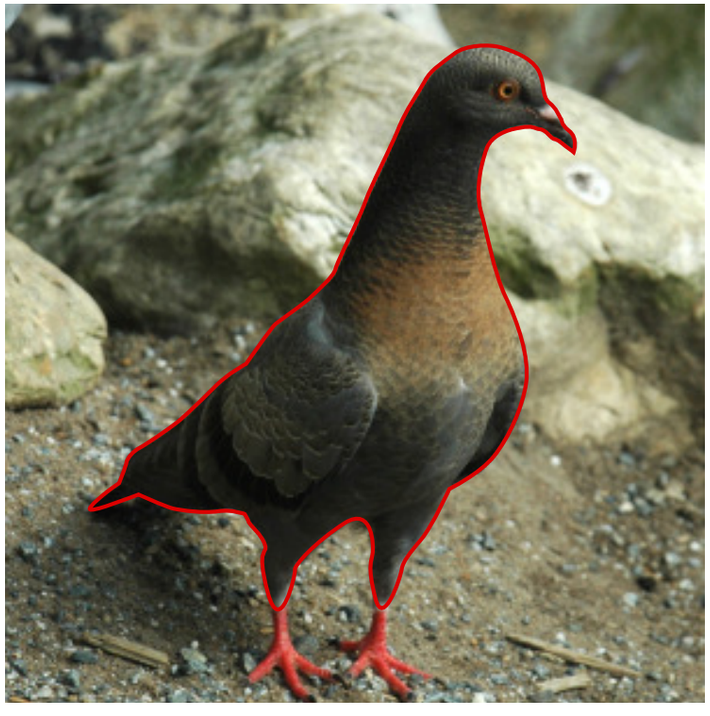
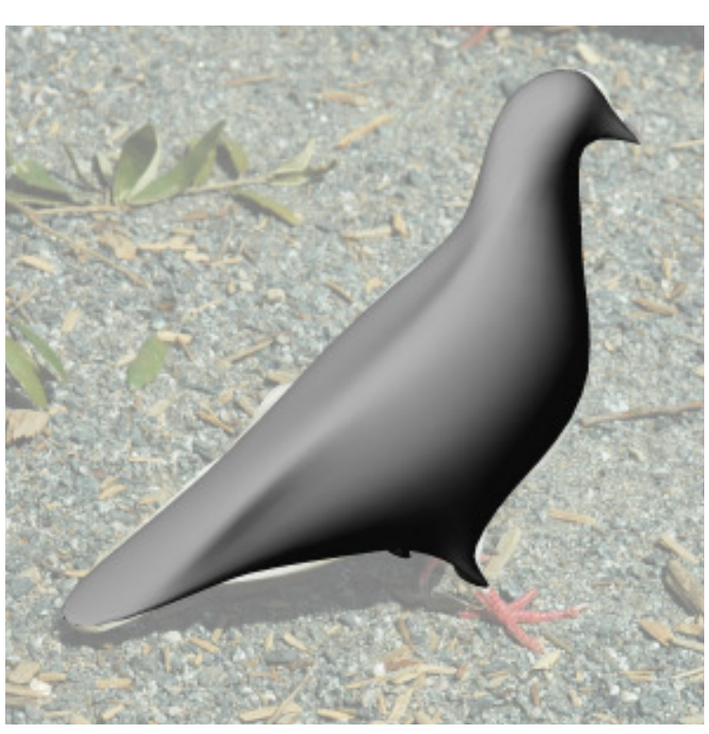
(c) As (b), but including iterations of our global search (§5.2).

(d) As (c), but with reparametrization around singularities.

Fig. 7. Different optimization methods for learning a morphable model with 8 parameters from 32 dolphin instances. In each image, the final solution surface is shown transparent and overlaid with the contour generator shown as red spheres. In (b), the preimage points belonging on the dolphin's foremost fin are stuck in local minima. This deficiency is repaired in (c), but poor parameter updates damage the convergence of the contour preimage points. In (d) the optimizer finds a good solution for nearly all preimage points.

Also works for pigeons



Probabilistic Programming

Traditional Methods

- Discriminative (e.g. Deep NNs)
  - data intensive
  - fast, but bottom-up
  - models are inflexible, specific to training set
- Generative Models
  - flexible, robust
  - efficient use of data
  - difficult to specify and train
  - models often scale poorly to large problems

Probabilistic Programming

- language designed to facilitate model-based inference
- express complex generative, probabilistic models
- provides a variety of state-of-the-art inference techniques (usually MC)
- integrates top-down and bottom-up inference

PP is WIP

1.2.1 Existing Languages

The design of any tutorial on probabilistic programming will have to include a mix of programming languages and statistical inference material along with a smattering of models and ideas germane to machine learning. In order to discuss modeling and programming languages one must choose a language to use in illustrating key concepts and for showing examples. Unfortunately there exist a very large number of languages from a number of research communities; programming languages: Hakaru (Narayanan et al., 2016), Augur (Tristan et al., 2014), R2 (Nori et al., 2014), Figaro (Pfeffer, 2009), IBAL (Pfeffer, 2001)), PSI (Gehr et al., 2016); machine learning: Church (Goodman et al., 2008), Anglican (Wood et al., 2014a) (updated syntax (Wood et al., 2015)), BLOG (Milch et al., 2005), Turing.jl (Ge et al., 2018), BayesDB (Mansinghka et al., 2015), Venture (Mansinghka et al., 2014), Probabilistic-C (Paige and Wood, 2014), webPPL (Goodman and Stuhlmüller, 2014), CPProb (Casado, 2017), (Koller et al., 1997), (Thrun, 2000); and statistics: Biips (Todeschini et al., 2014), LibBi (Murray, 2013), Birch (Murray et al., 2018), STAN (Stan Development Team, 2014), JAGS (Plummer, 2003), BUGS (Spiegelhalter et al., 1995)[1].

from: *An Introduction to Probabilistic Programming* by Meent et al (2018)
https://arxiv.org/abs/1809.10756

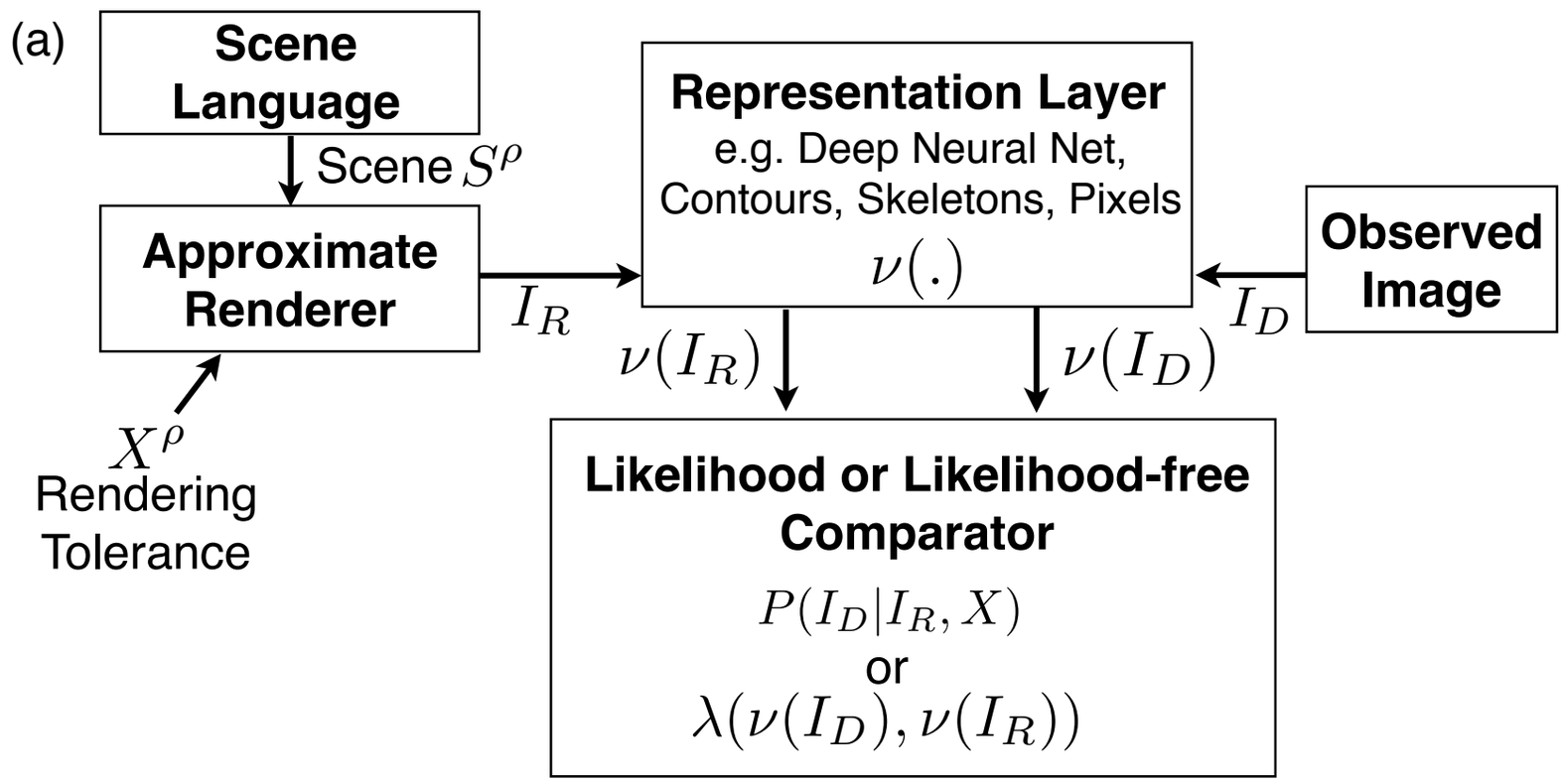
Picture: A Probabilistic Programming Language for Scene Perception

Tejas D Kulkarni
MIT
tejask@mit.edu

Pushmeet Kohli
Microsoft Research
pkohli@microsoft.com

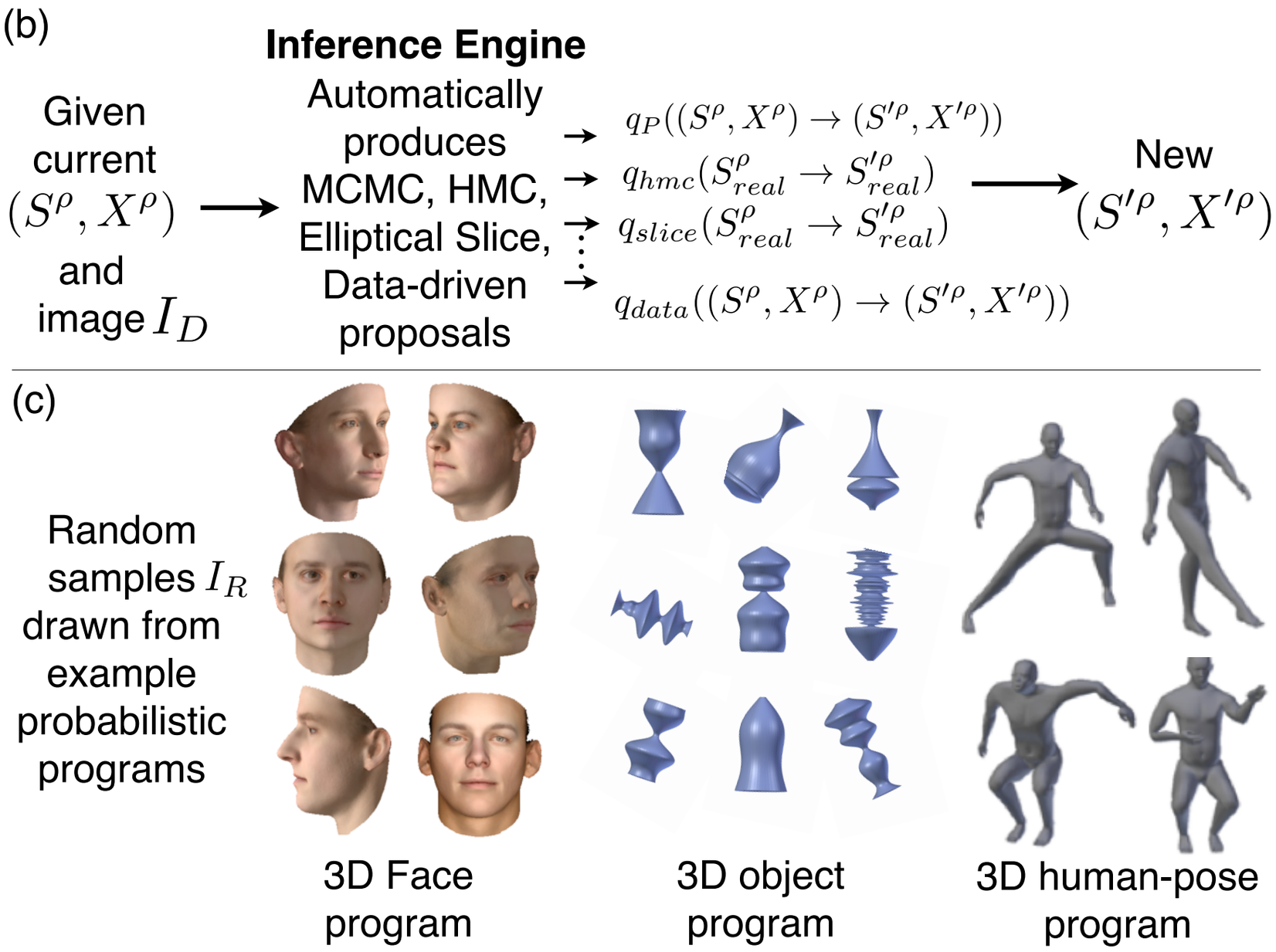
Joshua B Tenenbaum
MIT
jbt@mit.edu

Vikash Mansinghka
MIT
vkm@mit.edu



- programs specify scene description S and image I_D
- stochastically generates scene description and renders I_R to approximate I_D
- representation layer transforms I_D or I_R to to calculate likelihood
  - uses hierarchy of coarse-to fine representations from DNN, contours, and pixels

Inference Engine



- Automatically produce proposals to maximize probability of scene hypothesis S

Code

```
function PROGRAM(MU, PC, EV, VERTEX_ORDER)
  # Scene Language: Stochastic Scene Gen
  face=Dict();shape = []; texture = [];
  for S in ["shape", "texture"]
   for p in ["nose", "eyes", "outline", "lips"]
    coeff = MvNormal(0,1,1,99)
    face[S][p] = MU[S][p]+PC[S][p].*(coeff.*EV[S][p])
   end
  end
  shape=face["shape"][:]; tex=face["texture"][:];
  camera = Uniform(-1,1,1,2); light = Uniform(-1,1,1,2)

  # Approximate Renderer
  rendered_img= MeshRenderer(shape,tex,light,camera)

  # Representation Layer
  ren_ftrs = getFeatures("CNN_Conv6", rendered_img)

  # Comparator
  #Using Pixel as Summary Statistics
  observe(MvNormal(0,0.01), rendered_img-obs_img)
  #Using CNN last conv layer as Summary Statistics
  observe(MvNormal(0,10), ren_ftrs-obs_cnn)
end

global obs_img = imread("test.png")
global obs_cnn = getFeatures("CNN_Conv6", img)
#Load args from file
TR = trace(PROGRAM,args=[MU,PC,EV,VERTEX_ORDER])
# Data-Driven Learning
learn_datadriven_proposals(TR,100000,"CNN_Conv6")
load_proposals(TR)
# Inference
infer(TR,CB,20,["DATA-DRIVEN"])
infer(TR,CB,200,["ELLIPTICAL"])
```

Application to faces

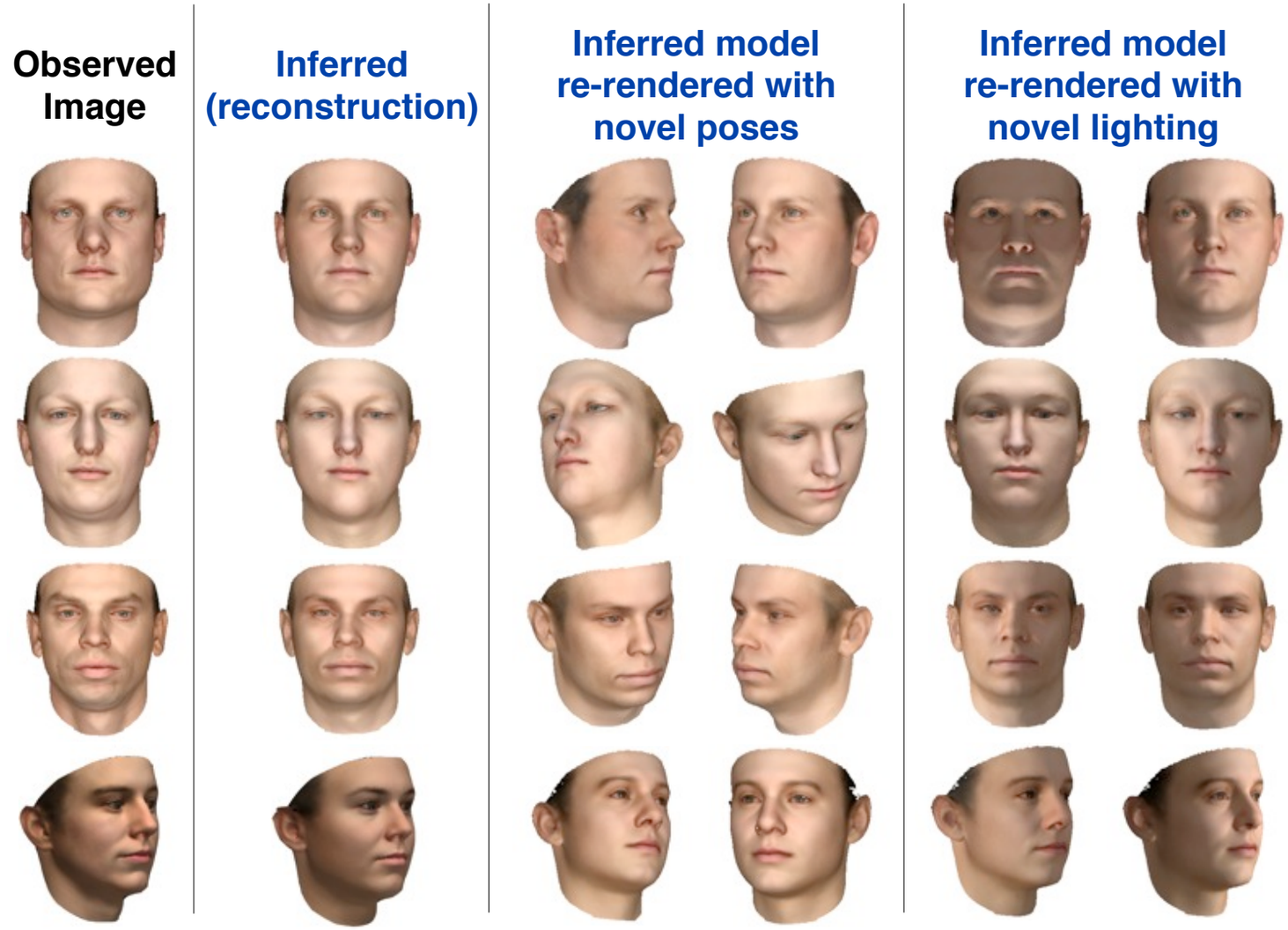
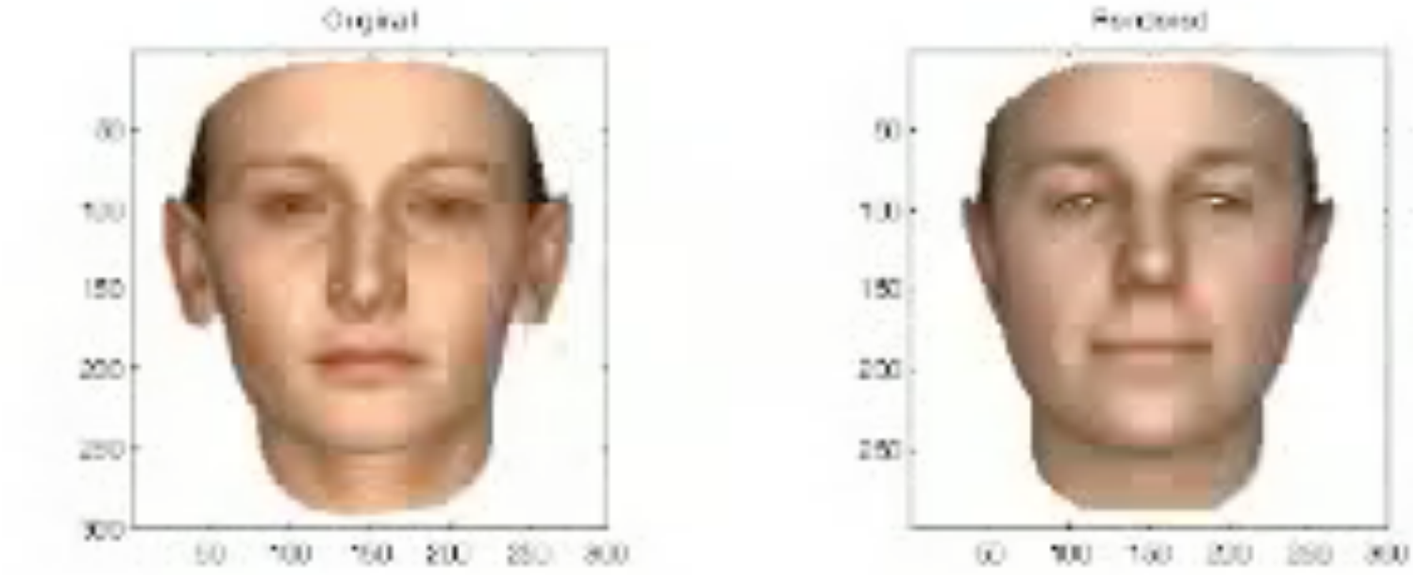
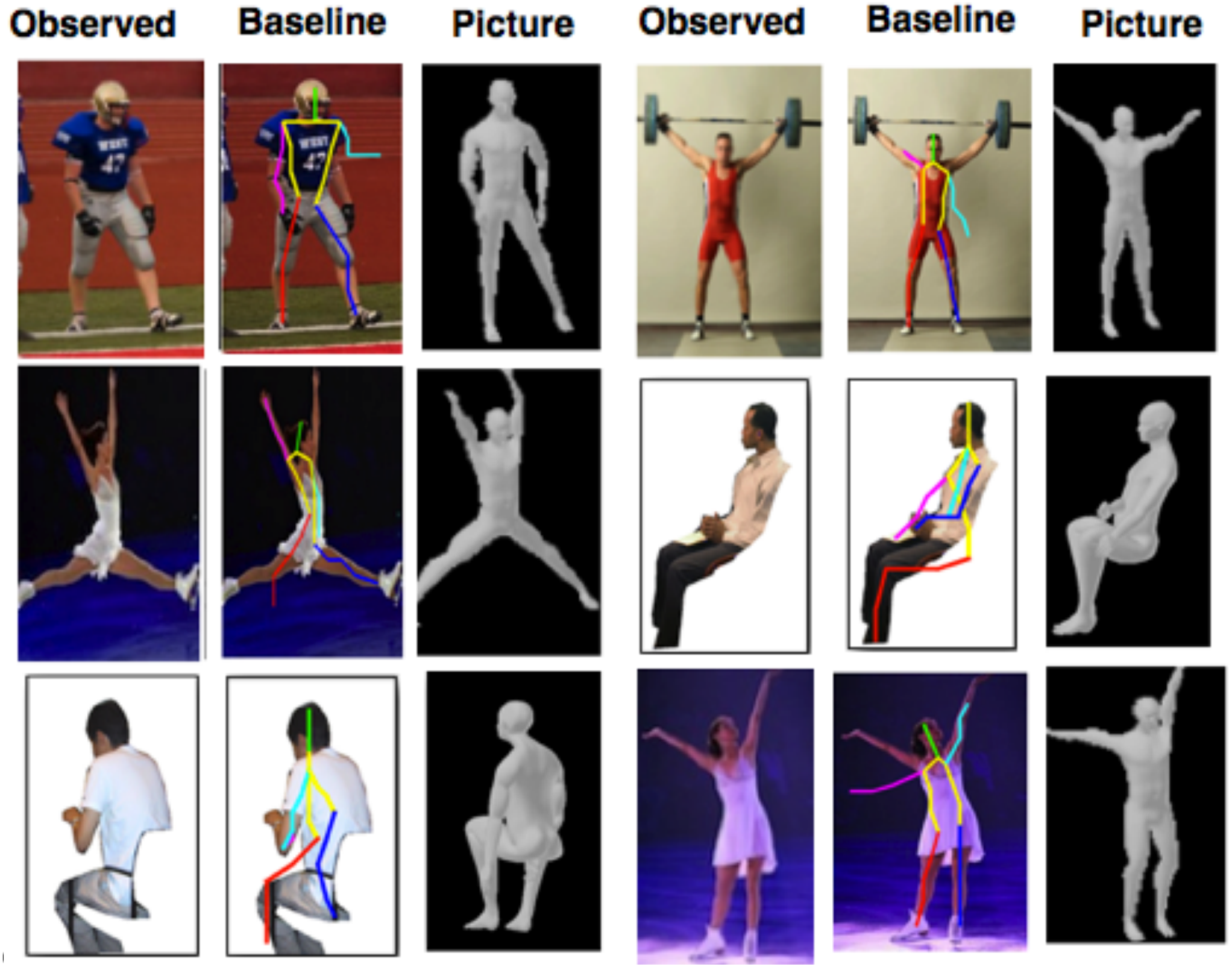


Figure 3: **Inference on representative faces using *Picture*:** We tested our approach on a held-out dataset of 2D image projections of laser-scanned faces from [36]. Our short probabilistic program is applicable to non-frontal faces and provides reasonable parses as illustrated above using only general-purpose inference machinery. For quantitative metrics, refer to section 4.1.

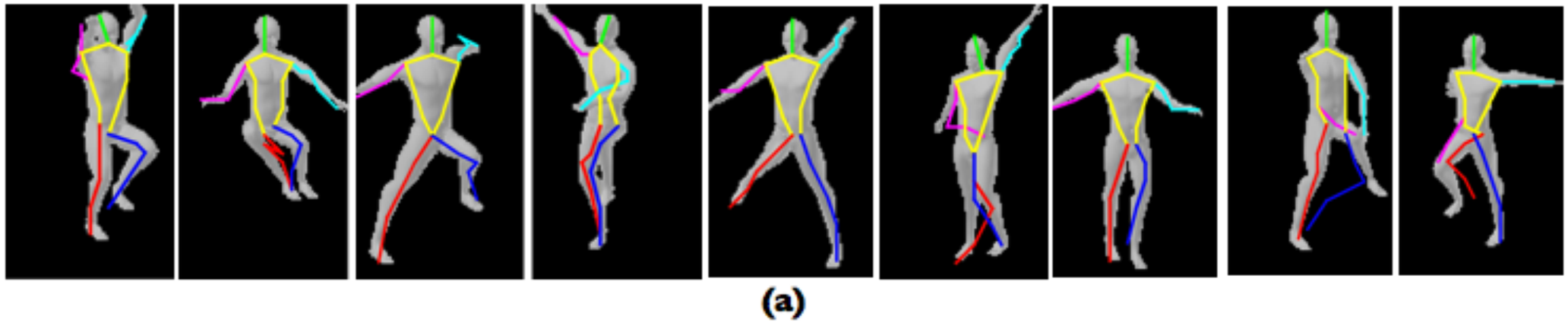

Original
Reflected

Inferring 3D human pose

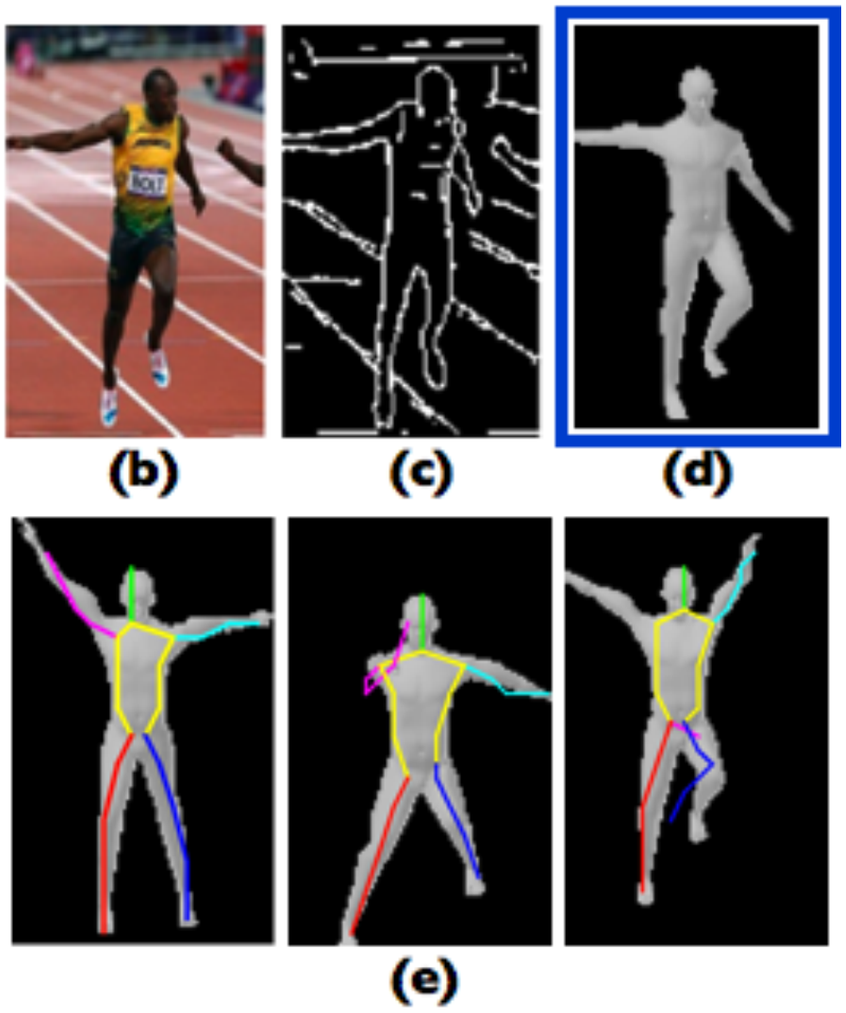


observed test image

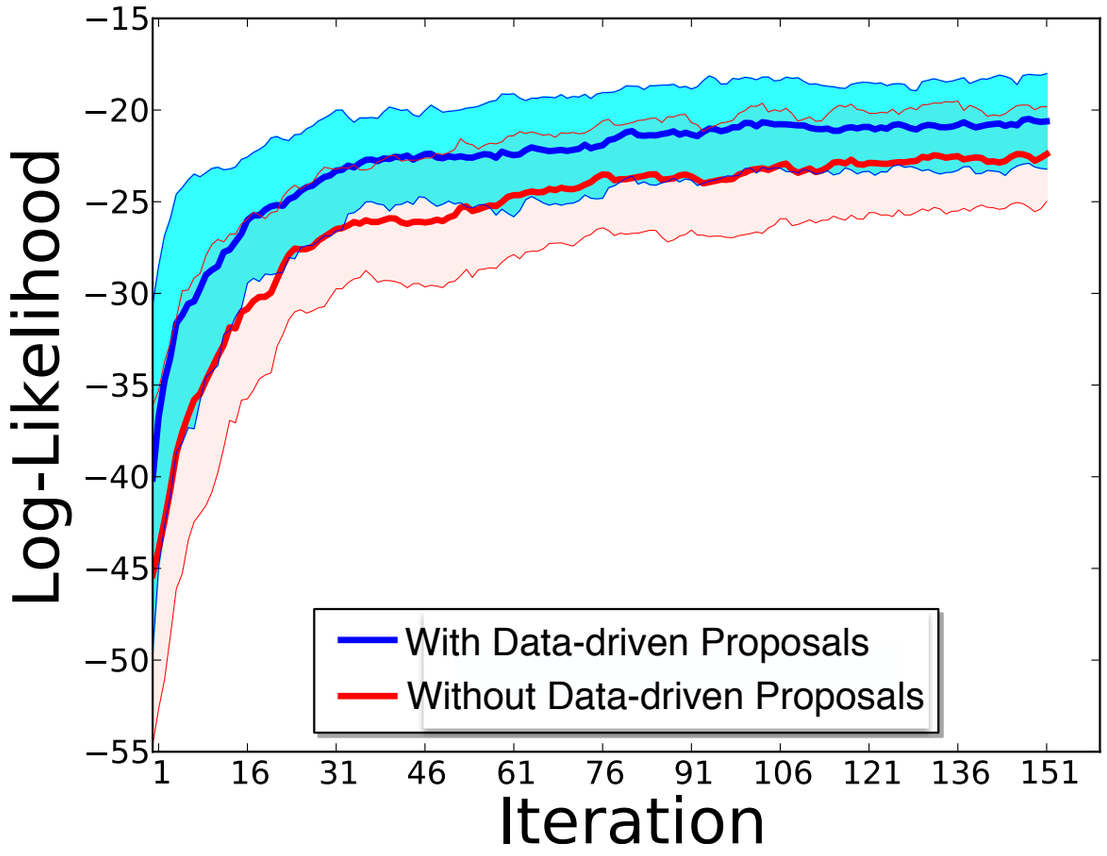
pose samples during inference



(a)



(b) (c) (d) (e)



(a) samples drawn from prior

(b) test image

(c) visualization of representation layer

(d) result after inference

(e) samples drawn when conditioned on test images