Spectral Learning
Part I

Geoff Gordon
http://www.cs.cmu.edu/~ggordon/
Machine Learning Department
Carnegie Mellon University

http://www.cs.cmu.edu/~ggordon/spectral-learning/

What is spectral learning?

machine learning is hard: fitting a good model = optimization with lots of local optima (e.g., learning HMMs is provably at least as hard as factoring big numbers)

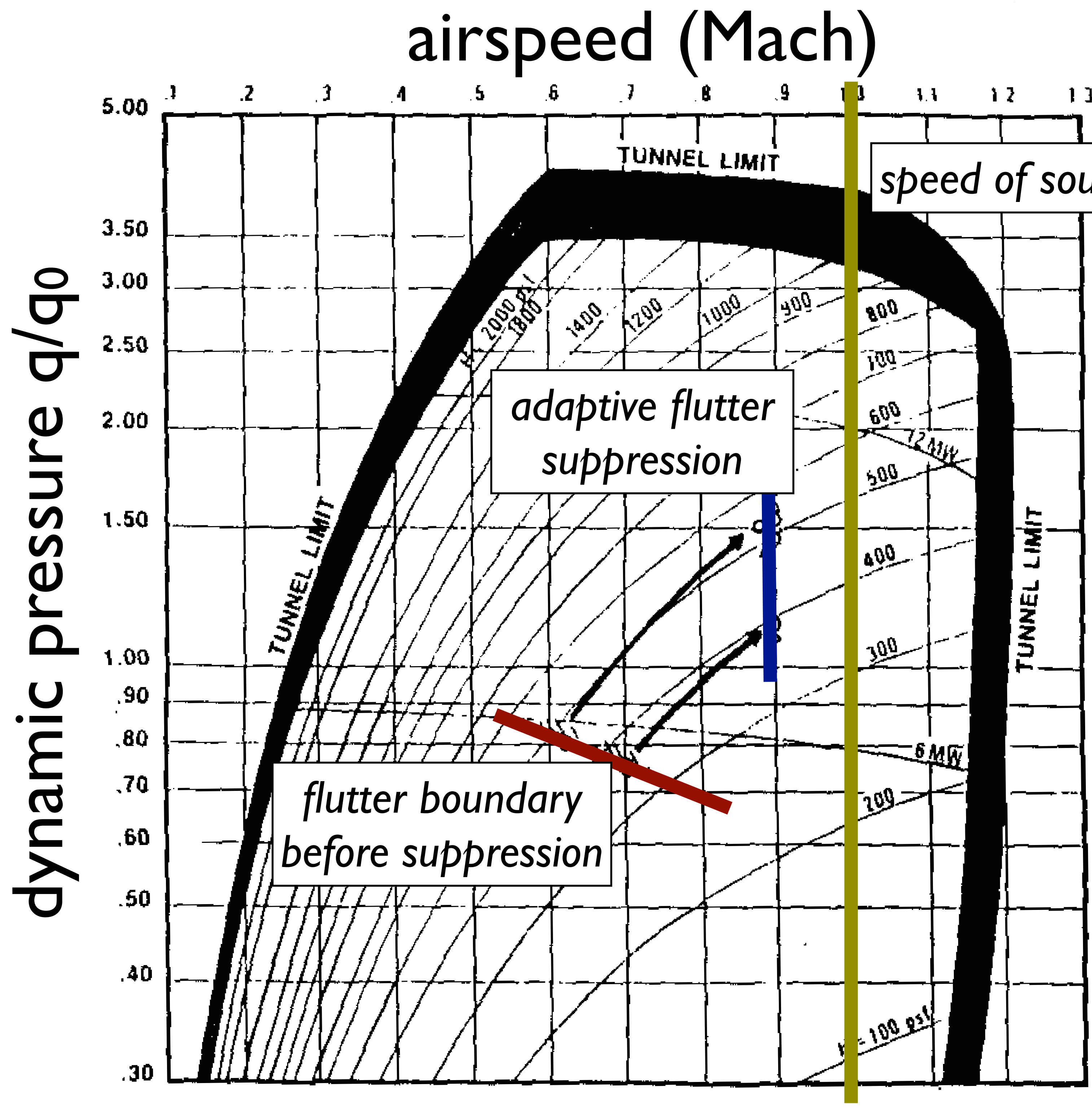
in some cases, by setting up the problem another way, we can avoid the difficult optimization entirely -- different set of tradeoffs (e.g., give up learning the factoring HMM, but make it a lot easier to learn other useful HMMs)

replace it with a spectral problem (= based on matrix eigenvalues/vectors)

What is spectral learning good for? We'll look at some examples...

Adaptive wing-flutter suppression

- High speeds can excite aircraft wing resonances
- Spectral LTI system ID adapts ***online***: use last 1s of flight data to ID, derive LQG controller, repeat



[Peloubet, Haller & Bolding, CDC 1990]

resonances depend on conditions—e.g., external fuel tanks attached at hard points—so, they can't easily be eliminated at design time

plot from wind tunnel tests w/ full-span wing model—intended to simulate 1990s-era fighter jet—typical flutter frequency 8.6Hz

plot shows tests w/ one wing configuration—not all tests moved the flutter boundary this much, and achievable speed varied widely depending on configuration (e.g., one configuration went from Mach 0.95 (before suppression) to 1.05 (after))

sensing was with 3 pairs of accelerometers placed on wing

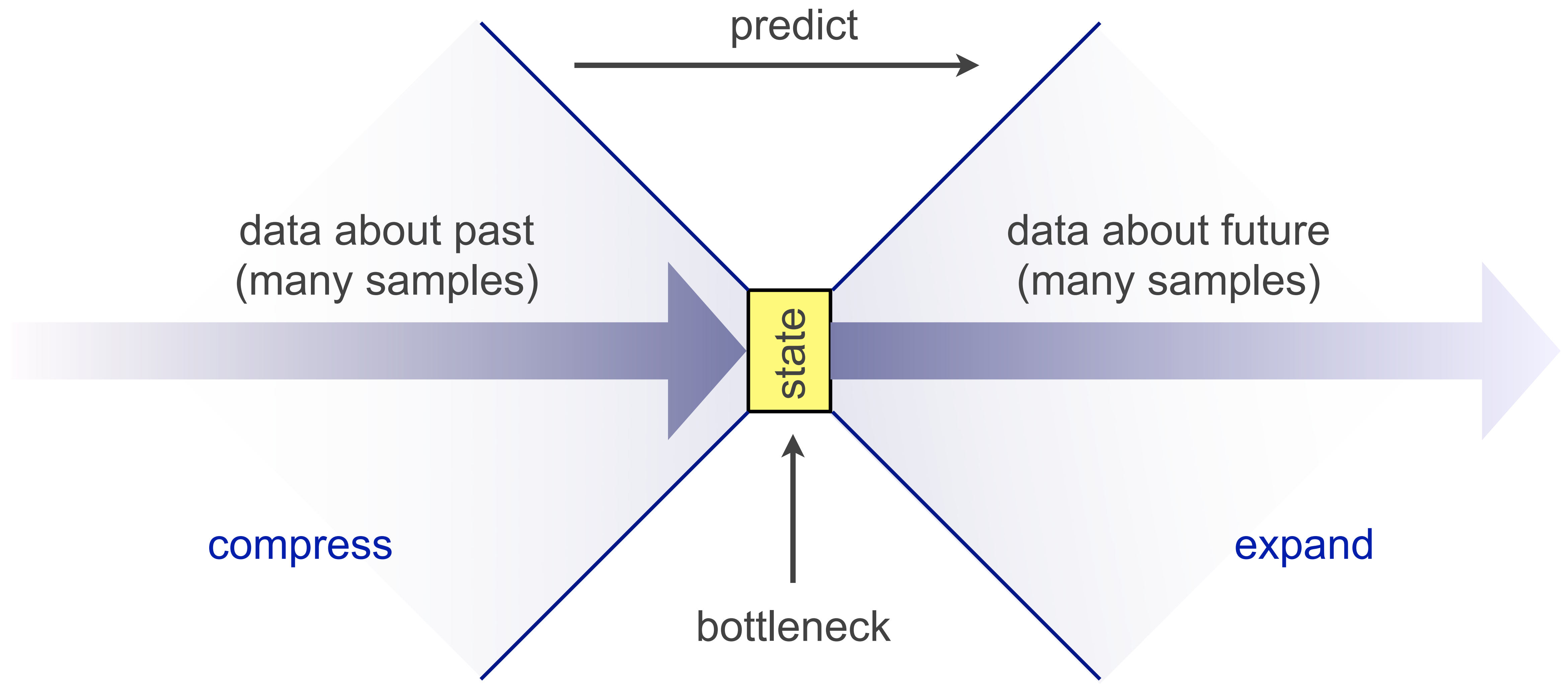
control was by actuating aircraft flaperons (wing surfaces) on top of input from pilot—100Hz updates

old-school: learning implemented on a VAX 11/750

===

dynamic pressure = kinetic energy of a fixed volume of air hitting the wing

Intuition: bottleneck



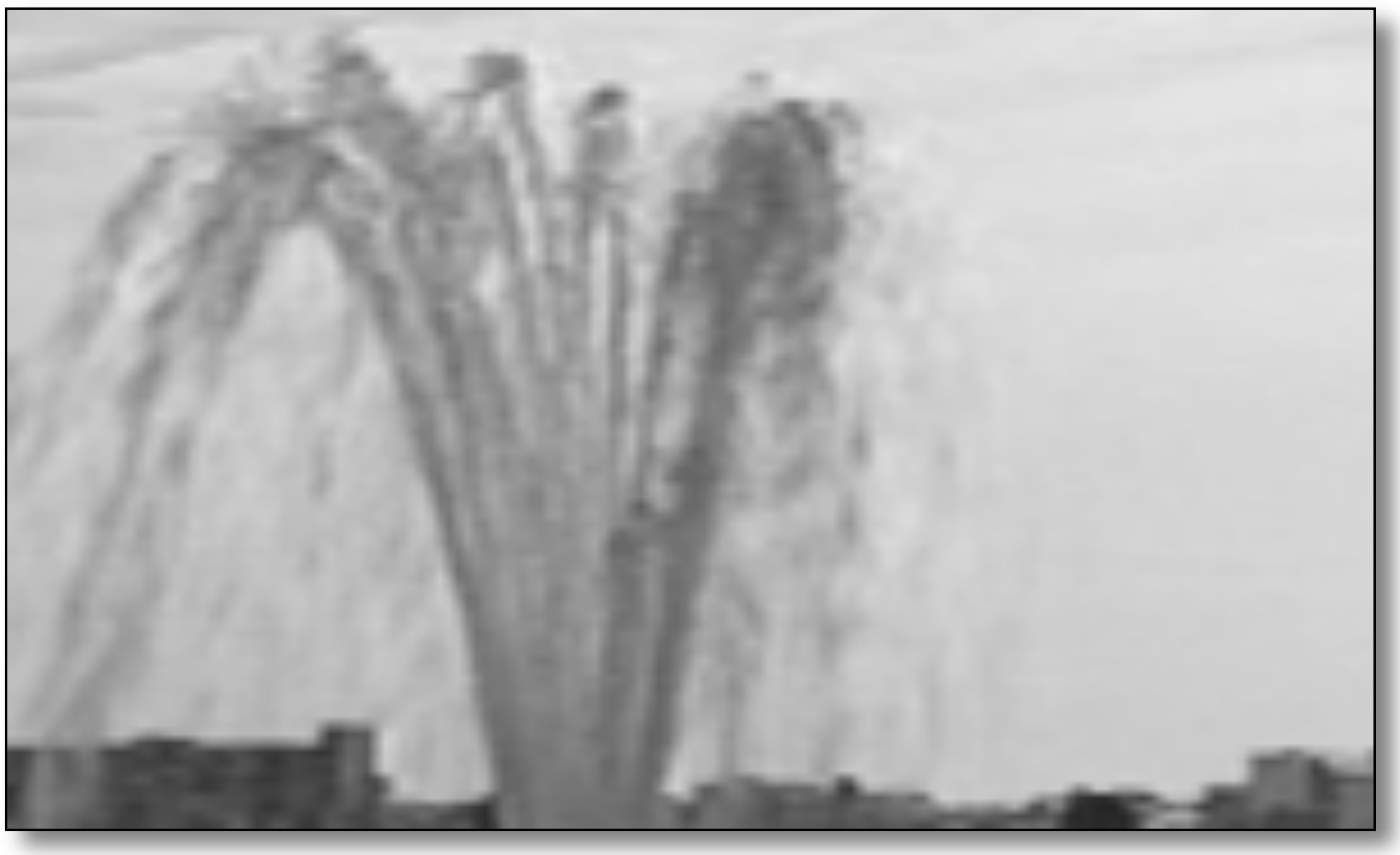
Can find best rank-k bottleneck via matrix factorization ⇒ ***spectral*** method

for regularization, don’t just want a state: want a low-d (hopefully minimal) one

so, select k (linear combinations of) predictions which explain as much variance in future as possible

Video textures

(train from a short video, then simulate learned model)



fountain

steam grate

observation = raw pixels (vector of reals over time)

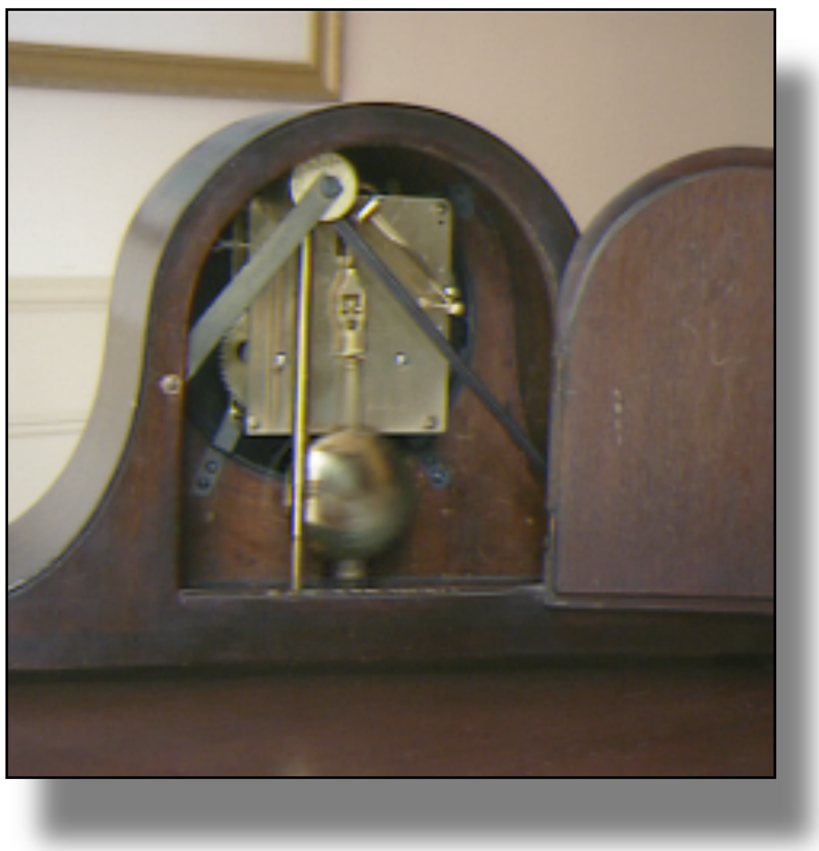
[Siddiqi, Boots & Gordon, 2007; Doretto & Soatto 2006]

Both this example and the last one: learned model is a linear time invariant (LTI) system (aka Kalman filter)

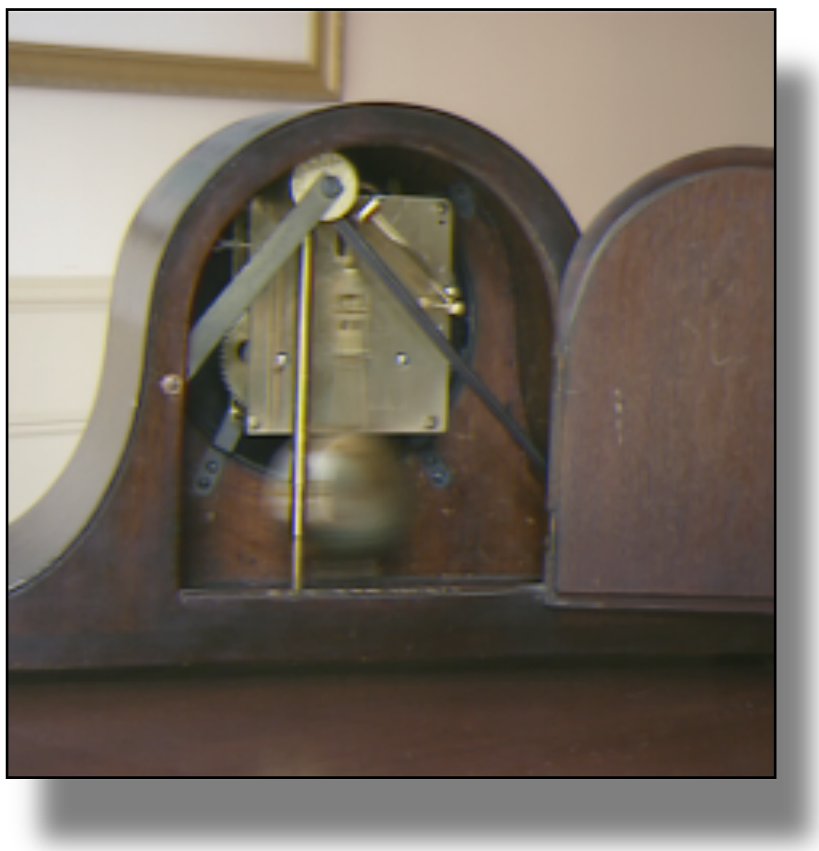
Spectral algorithms for LTI systems have been known for decades—but recently it was discovered that we can use spectral algorithms to learn more-interesting models

Video textures, redux

Original



Kalman Filter



PSR

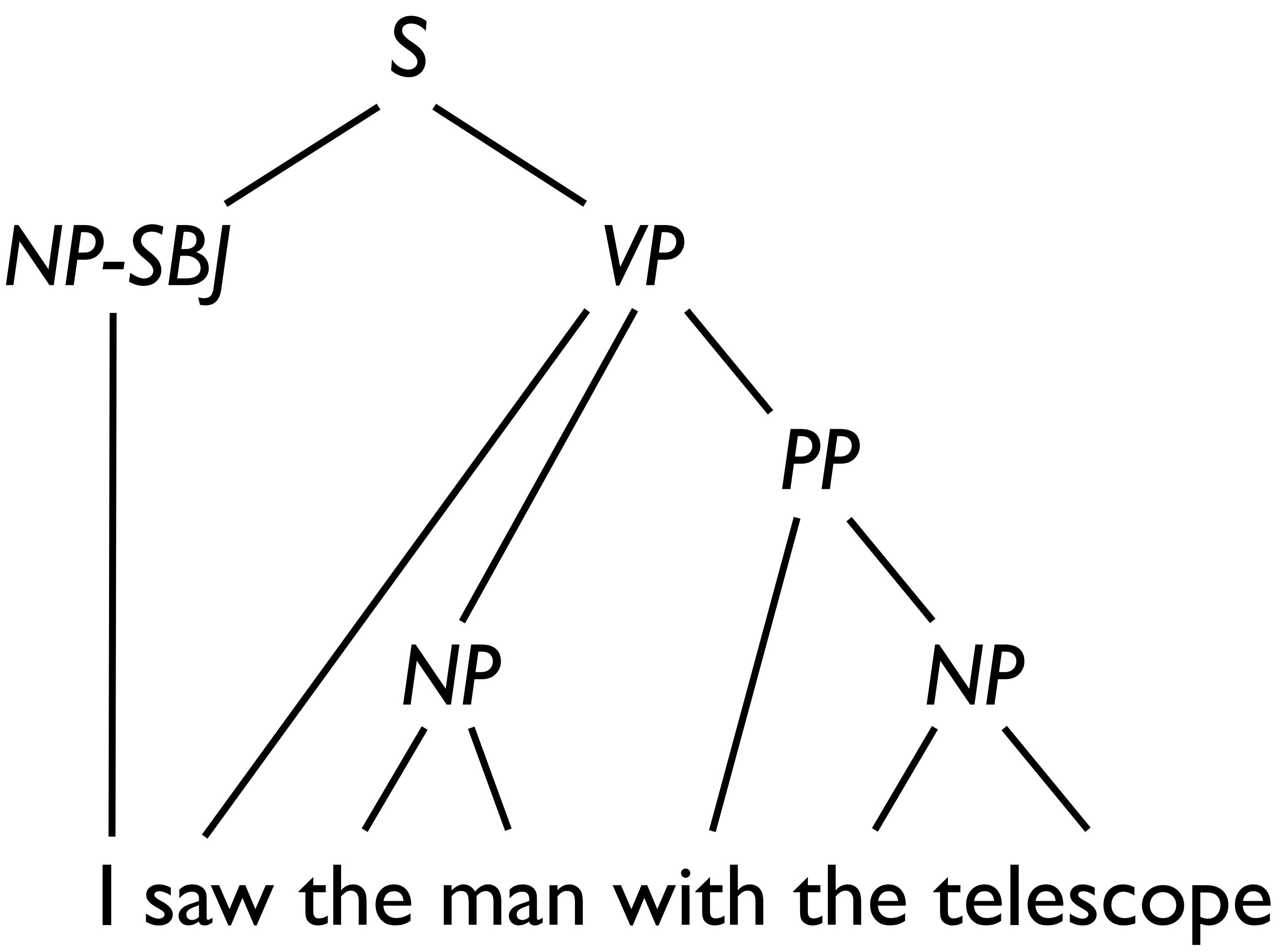
both models: 10 latent dimensions

reason for failure of Kalman: Gaussian is log-concave. So, if two images are likely, their *average* is at least as likely.

So: we need a *nonlinear* filter

Learning to parse

[Cohen, Stratos, Collins, Foster & Ungar, ACL 2012]



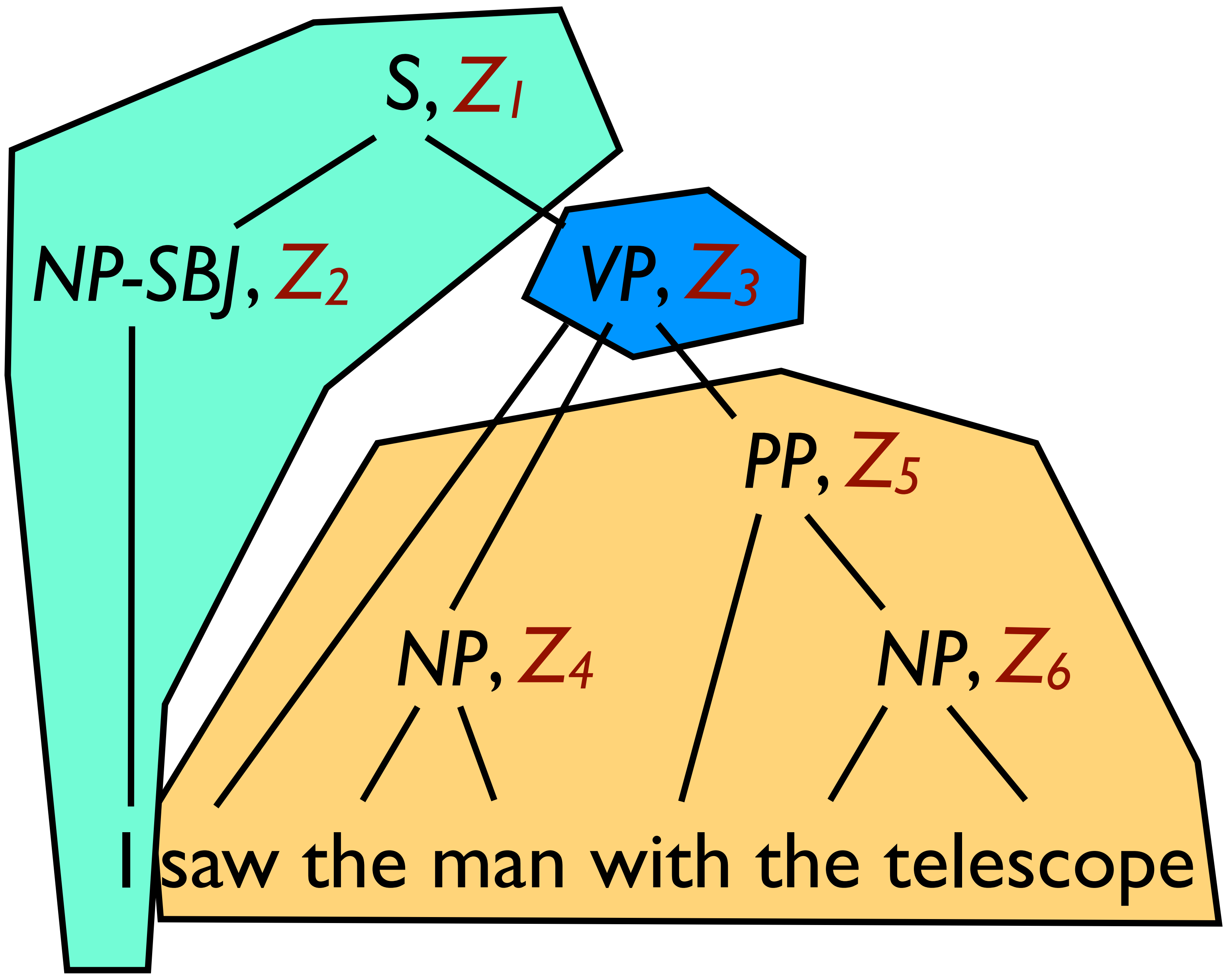
- Treebanks commonly used to train parsers
- But manual tags typically don't make fine-enough distinctions to enable local decisions by parser
  - e.g., helps to label NPs with gender, number, mass/count, animate/inanimate, human/nonhuman, …

parse is from Penn Treebank, omitting annotation of ambiguity -- could have been (NP (NP the man) (PP with the telescope))

the extra labels would help parsing, but they're not available—could we simulate them?

Learning to parse

[Cohen, Stratos, Collins, Foster & Ungar, ACL 2012]



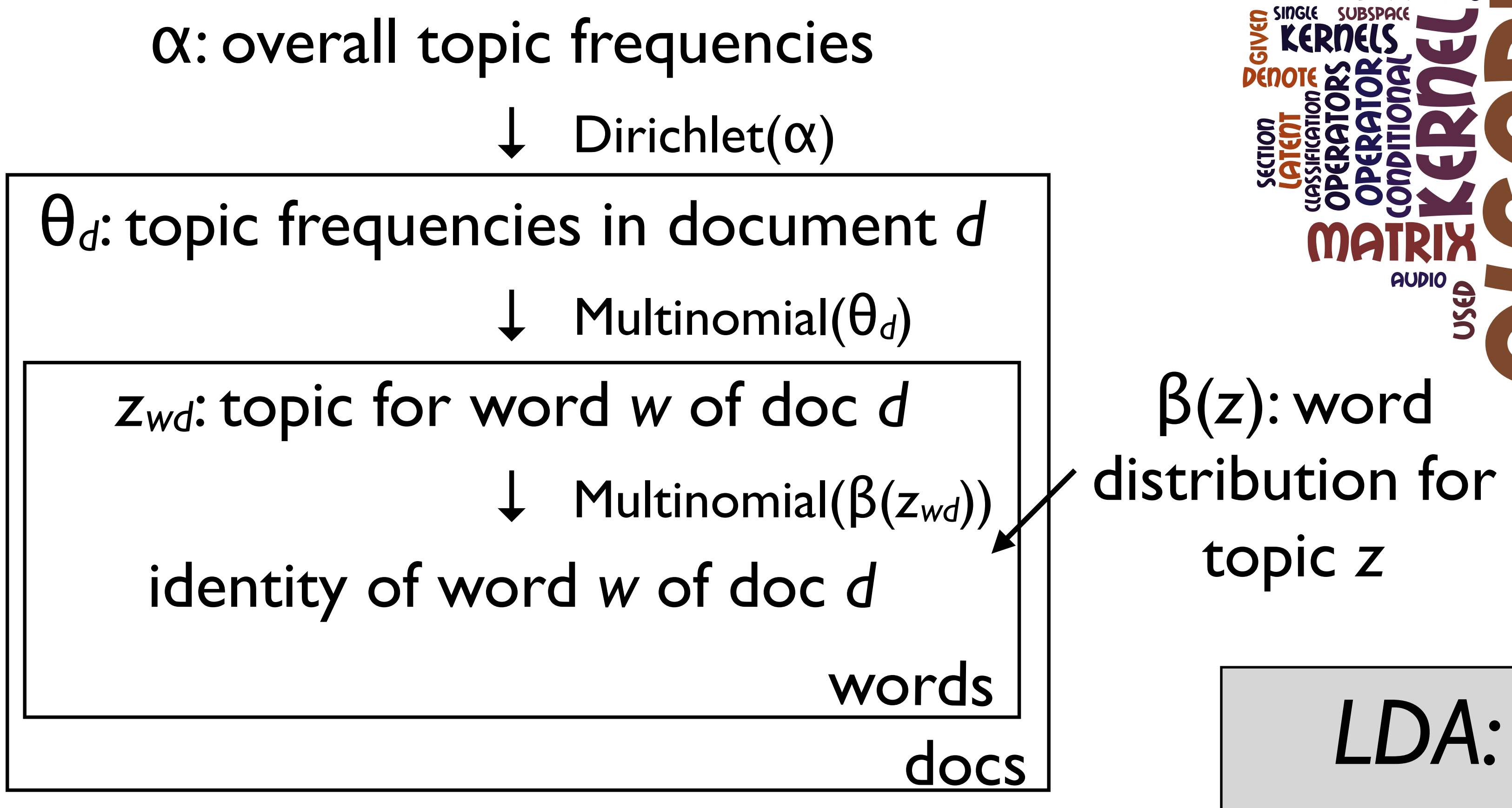
- Refine manual tags w/ learned latent variables
- Each latent depends on neighboring tags and latents
- Learn to fill in latents by trying to explain training parses
    - spectral method leads to low-d latents

intuition: to determine latent at each node, predict (features of) inside context from outside context, and vice versa

advantages of spectral: very fast to train (can handle bigger data sets); no need to think of a smart initialization to defeat local optima; can benefit from arbitrary smart features computed from inside and outside contexts

but so far, smart initialization of EM to find MLE still gets best final parsing accuracy

Latent Dirichlet Allocation



α: overall topic frequencies

↓ Dirichlet(α)

θ_d: topic frequencies in document *d*

↓ Multinomial(θ_d)

z_{wd}: topic for word *w* of doc *d*

↓ Multinomial($\beta(z_{wd})$)

identity of word *w* of doc *d*

words

docs

β(z): word distribution for topic z

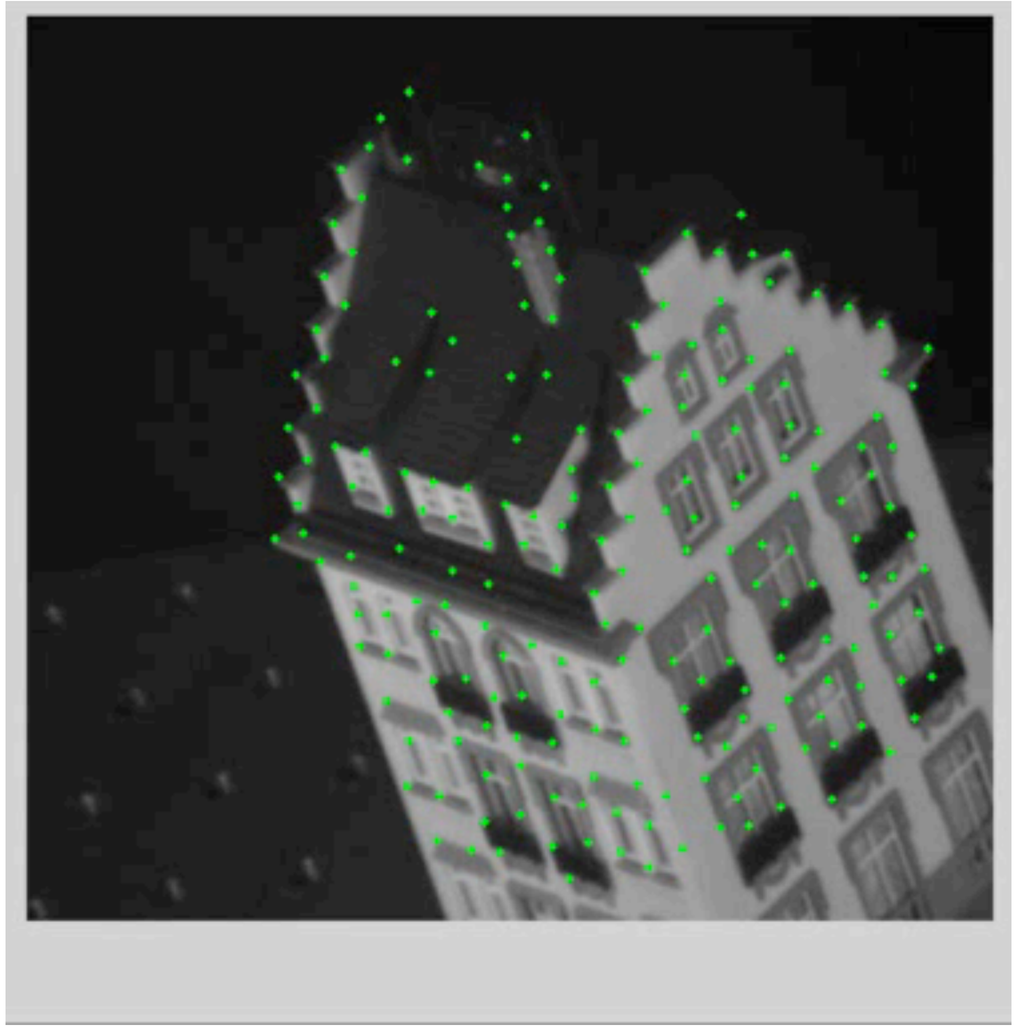
LDA: famous example of not knowing when MCMC converges

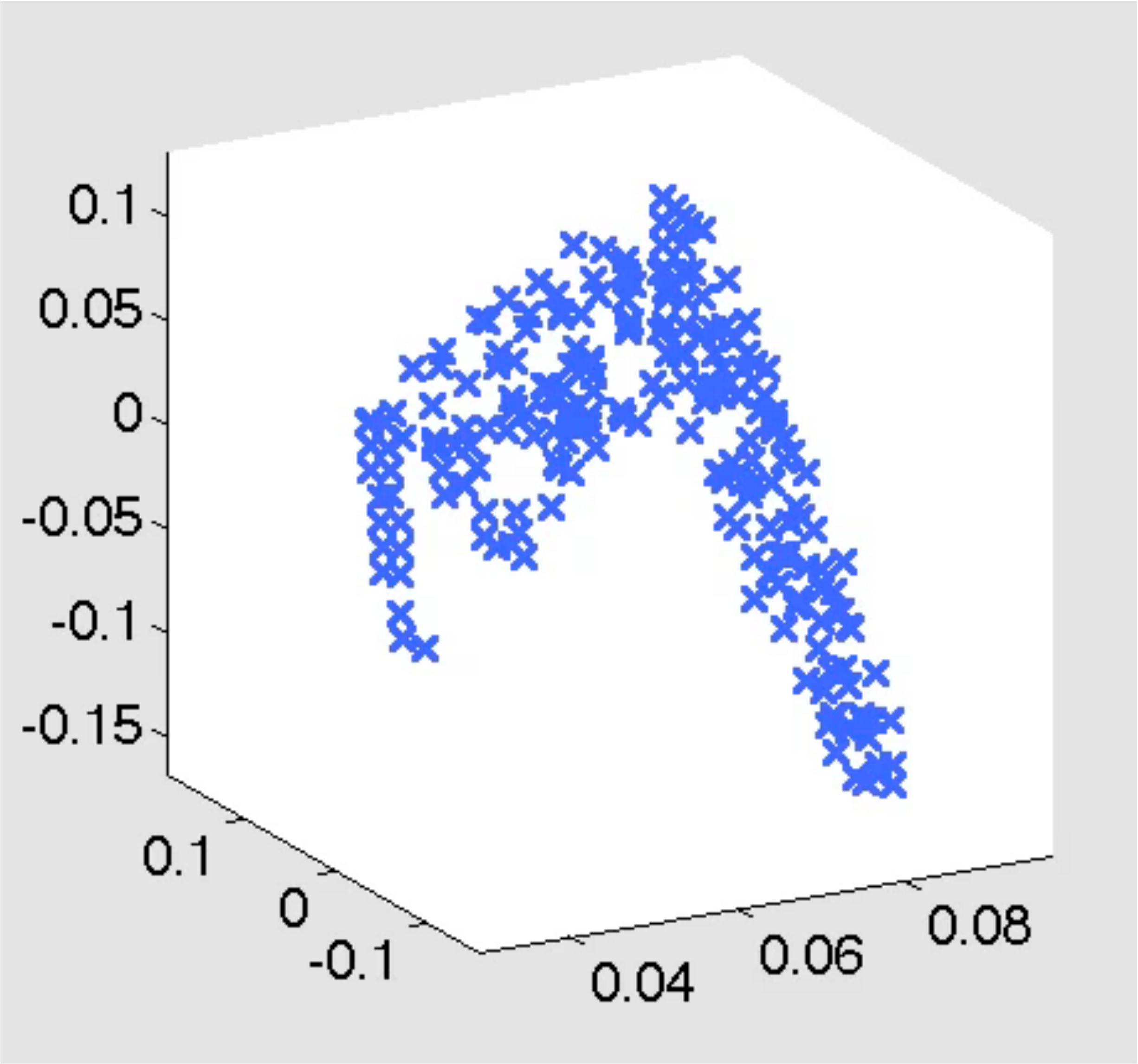
Spectral: single global optimum

[Blei, Ng and Jordan, JMLR 2003]
[Anandkumar et al., arXiv 2012]

but, practical implementation of spectral method still in progress

Structure from Motion

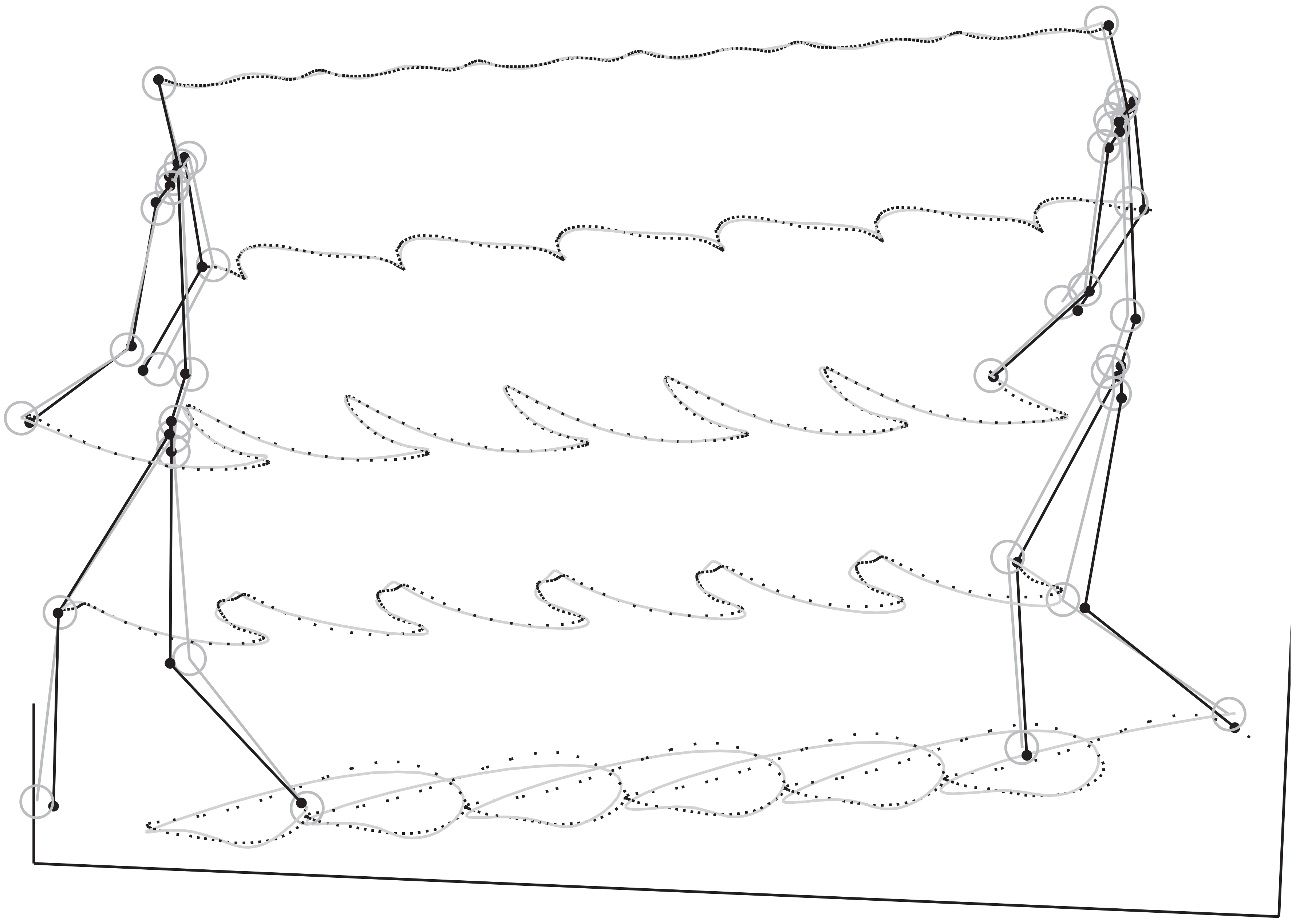




[Tomasi & Kanade, 1992]

measurement matrix = 2d image positions of tracked features over time

Structure from Motion

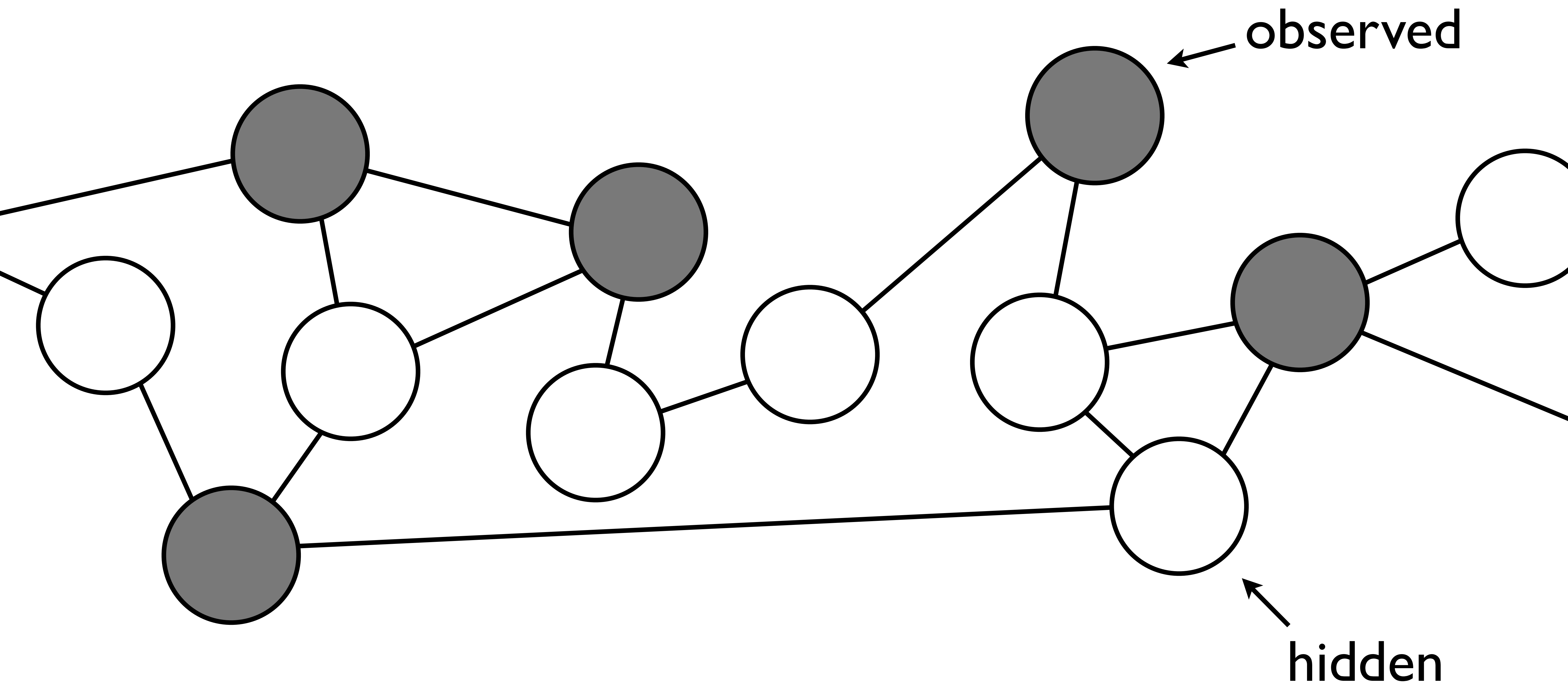


nonrigid version: learn a model of how shape changes over time

[Akter, Sheikh, Khan & Kanade, TPAMI 2011]

grey line: recovered
black dots: ground truth

The problems: inference & learning



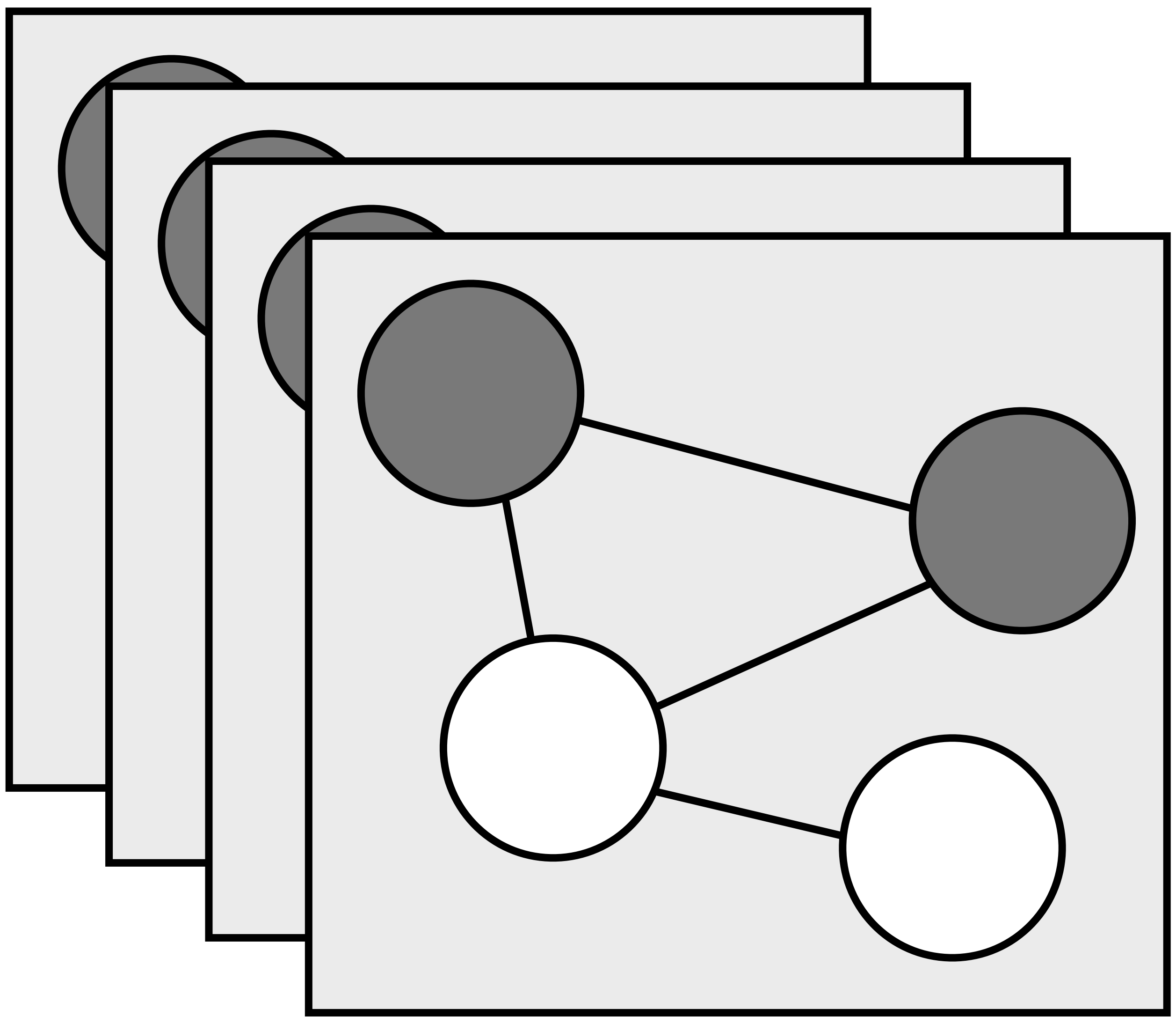
some structure of MRF or factor graph

some nodes always hidden

some nodes always or usually observed in training data (but perhaps not in test)

variables may be discrete, continuous, mixed; structure may vary from example to example (so long as there are common pieces)

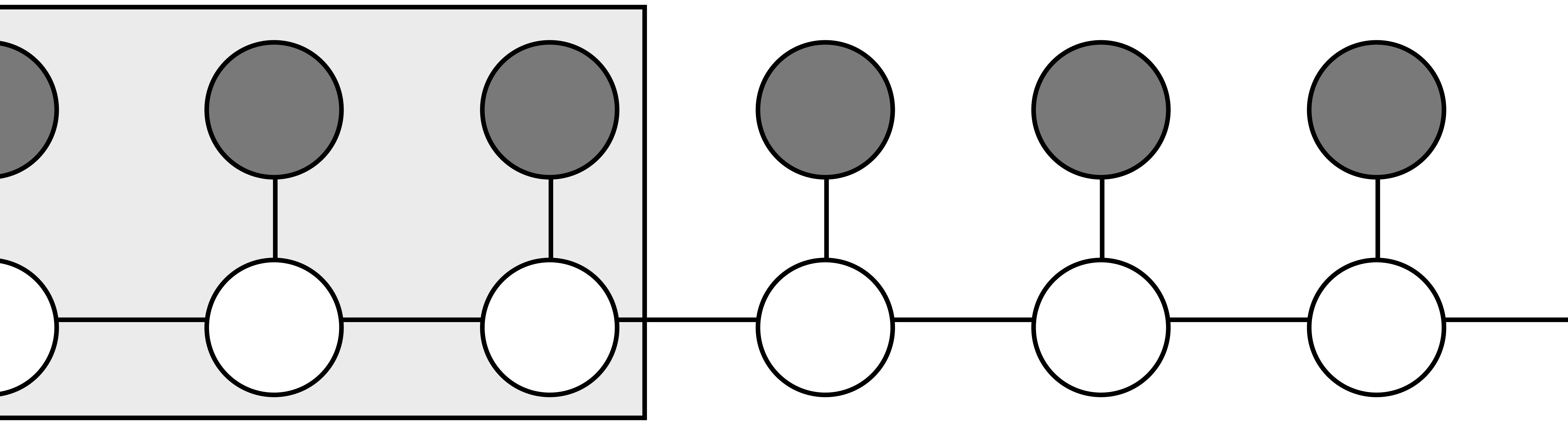
The data



Many replicates of our network

either: repeated observations of same graph, or…

The data



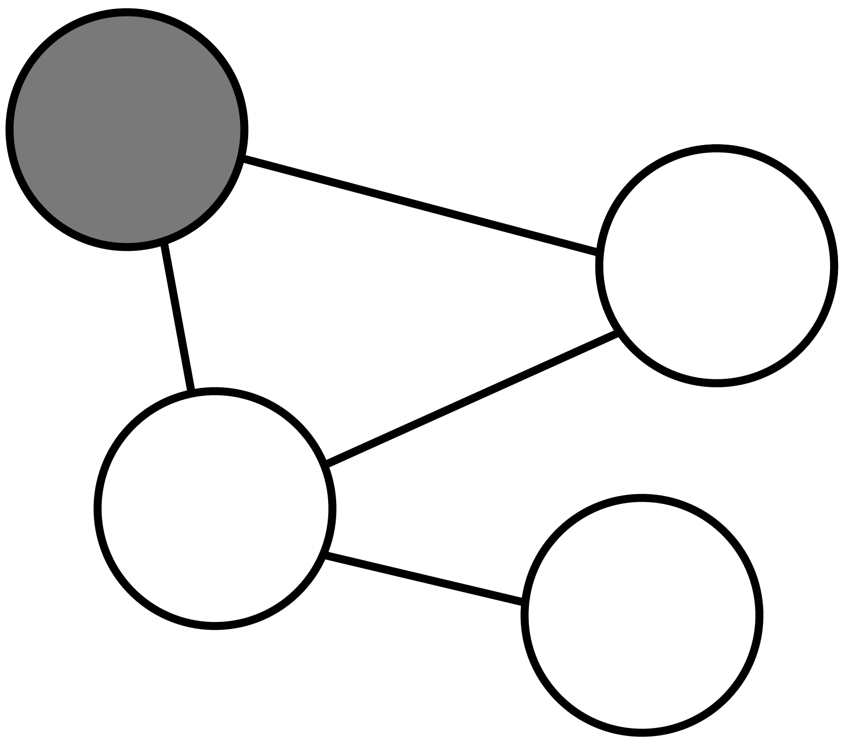
… or many common substructures
carved out of one big network

... repeated structure in one or a few big graphs

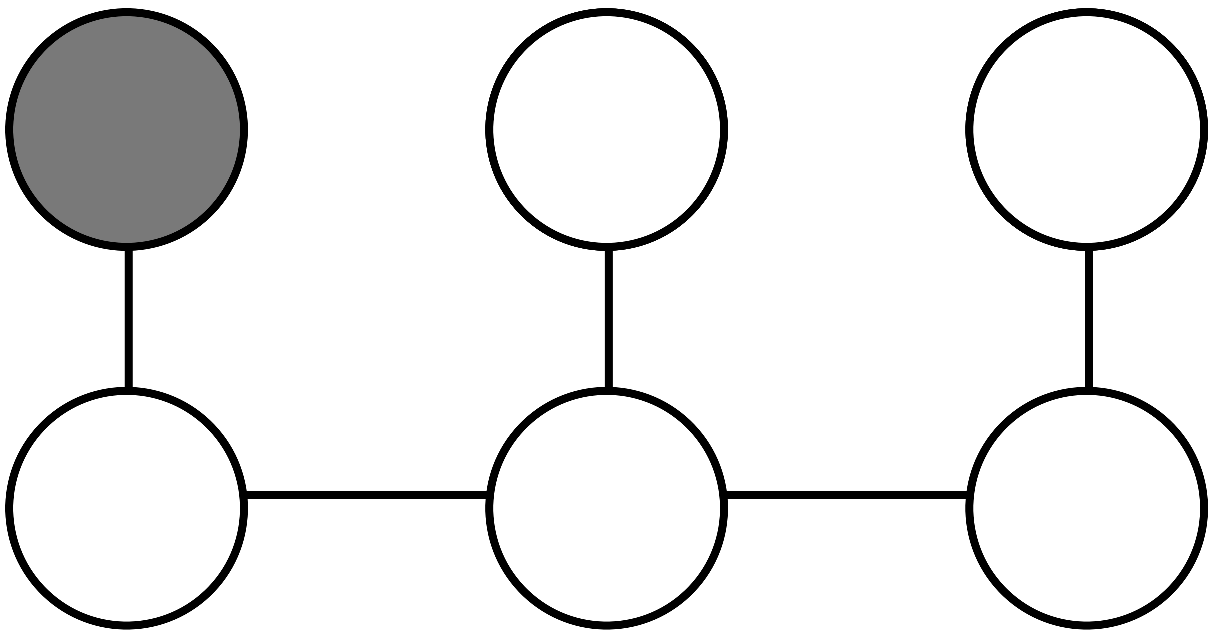
need repetition one way or the other so that we have a chance to learn something

could also have a similar sliding window for trees

The data



Test time: some vars observed in training set are missing



for simplicity of notation, assume for now repeated observations of same graph

Exact solutions

- Inference: ***belief propagation***
  - problem: exponential in treewidth (base = arity)
  - continuous vars: exponential in diameter too
- Learning: ***maximum likelihood***
  - e.g., gradient descent or EM
  - problem: inference is a subtask
  - problem: lots of local optima

inference is (very!) hard in general
learning is (very very!) hard in general

any hope of computationally efficient learning and inference? yes, w/ (surprisingly weak) assumptions, using spectral methods

Key ideas for spectral learning

- Spectral bottleneck

  *[a bunch of control theorists in '80s;
  van Overschee & de Moor, Automatica, 1993]*

- Predictive state

  [Littman, Sutton, Singh, NIPS 2001]

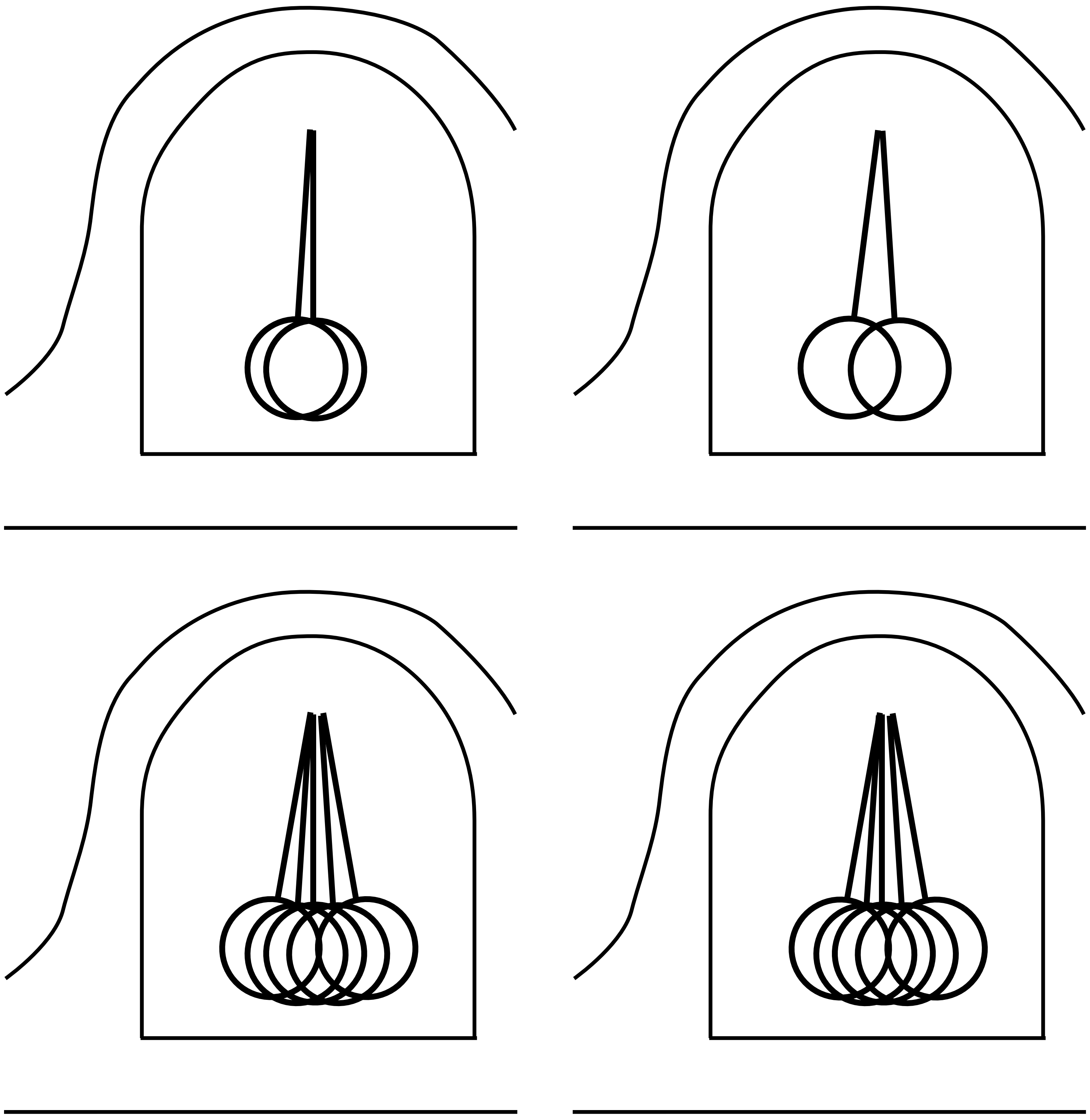
- Observable representation of model

  [Jaeger, Neural Computation, 1999]

already seen bottleneck idea
predictive state: next
observable rep'n: later
    expresses model in terms of predictive state and direct observables

Predictive state

Use a vector of predictions of (features of) observables as state



e.g., in movie, predicted future pixels
e.g., in parse tree, predicted nonterminals on opposite side of current nonterminal

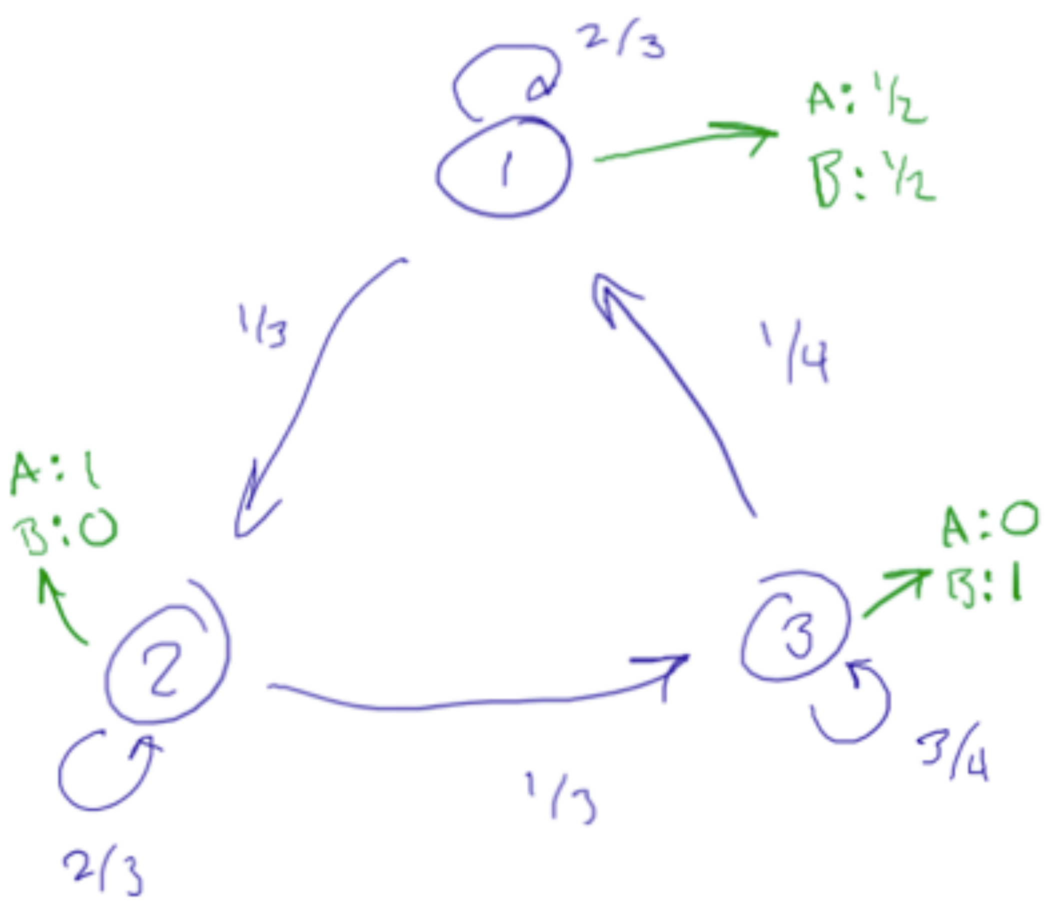
Here, we need predictions of pixels from at least two frames (to capture both position and direction of pendulum).

Predictions are E(features of future observations). Predictive state can represent uncertainty: e.g., uncertain position leads to average image of pendulum in various positions. (This smeared-out image doesn't mean that we will ever see the pendulum in multiple states at once, only that we are uncertain where it will be.)

Why do we want a predictive state? Easier to interpret, easier to learn. E.g., if learning algorithm wants an unbiased estimate of current state, all it has to do is wait and see what the future looks like.

In some cases we know a good set of features to predict (examples include HMM, Kalman, characteristic RKHS); if not, we guess feature space (typical ML feature engineering problem)

Predictive state: minimal example



- $E([\varphi(o_{t+1}) \ldots \varphi(o_{t+k})] \mid s_t) = W s_t$ with W invertible

here we derive a predictive state for a simple HMM (exact numbers don't matter)

transition matrix T, observation matrix O

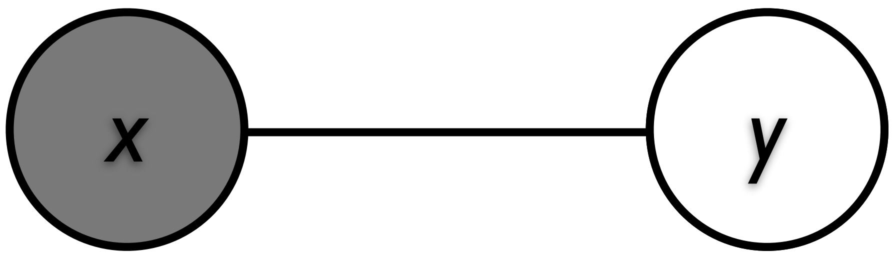
given state s, current expected observation is Os

next state is Ts, so next expected observation is OTs, etc.

pick W to be any 3 linearly independent rows of O, OT, OTT, OTTT, etc.

note: may not be able to get an invertible W under non-observability: e.g., 2 distinct hidden states w/ identical transitions and observations. But in this case we can pick a representative of each equivalence class of states, and use the corresponding pseudoinverse.

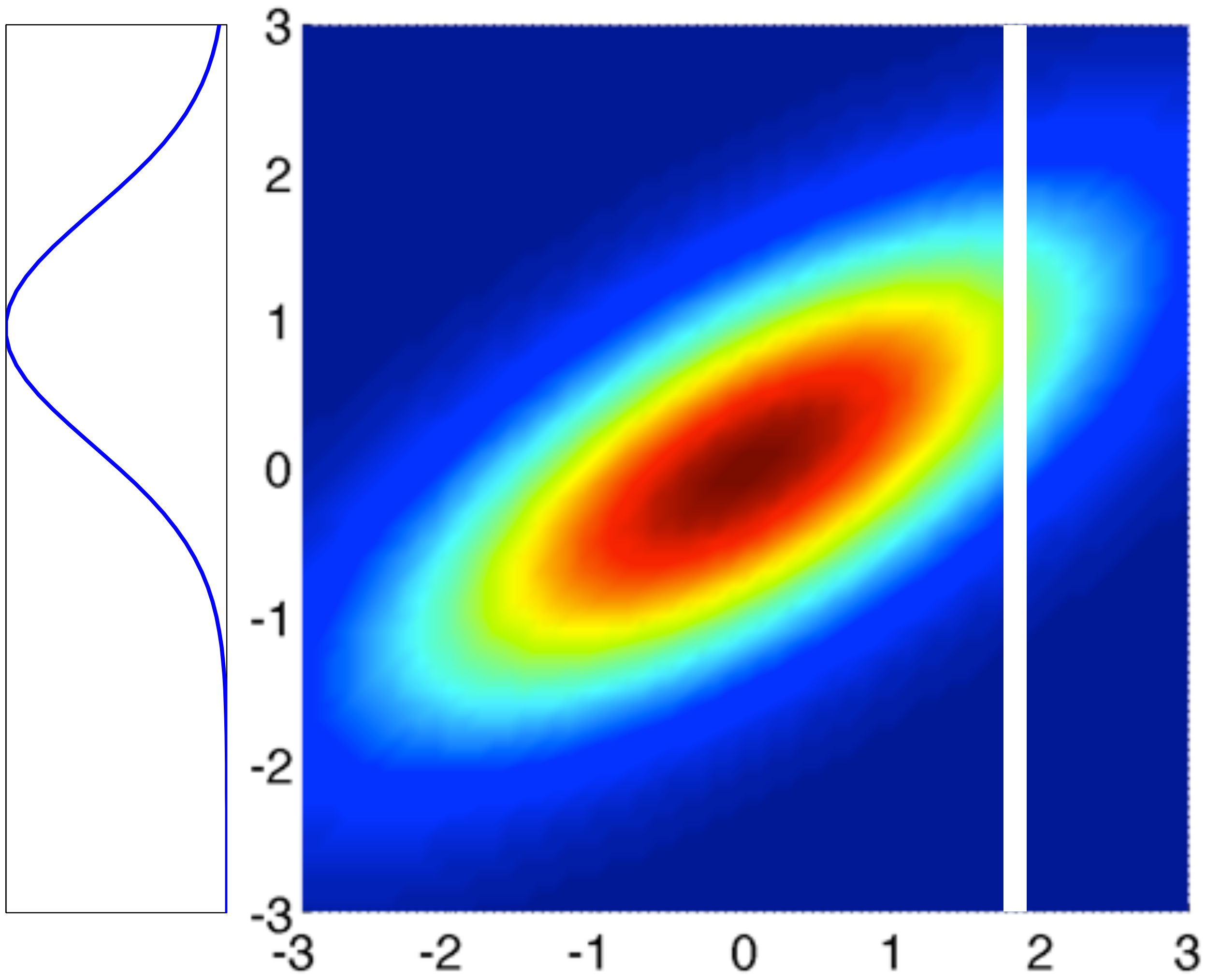
Start simple



$$\mathbb{P}(y \mid x) = N(\Sigma_{yx}\Sigma_{xx}^{-1}x, \ldots)$$

Inference: normal equations

Learning: estimate Σ_{xx} and Σ_{xy} from data



$$\mathbb{P}(x, y) = N(0, \Sigma)$$

$$\Sigma = \begin{pmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{pmatrix}$$

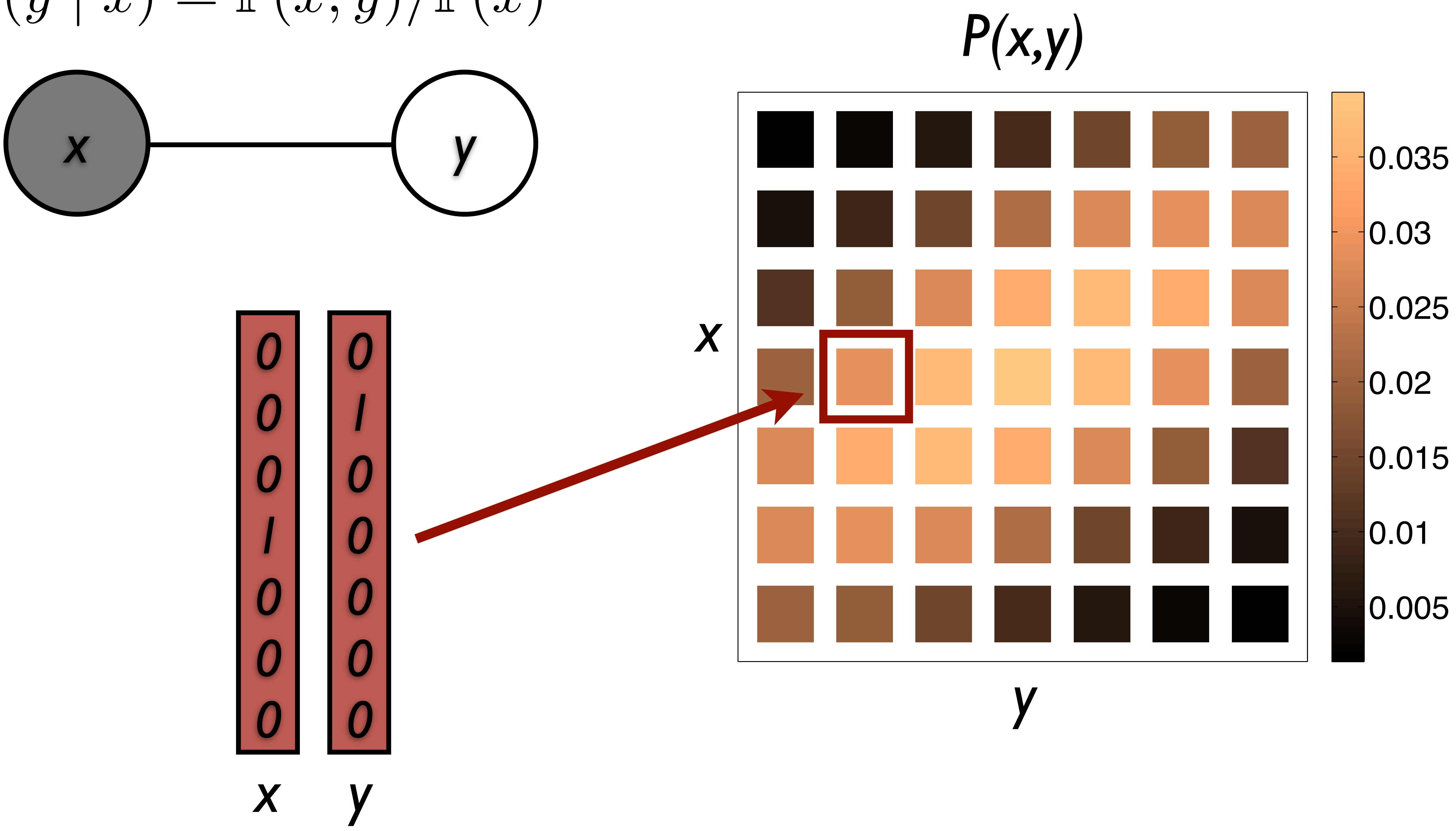
2D Gaussian case; training data has (x,y), while test data has x and we need to predict y

normal equations give Bayes rule update for E(y | x)

learning is just estimating covariances (= linear regression)

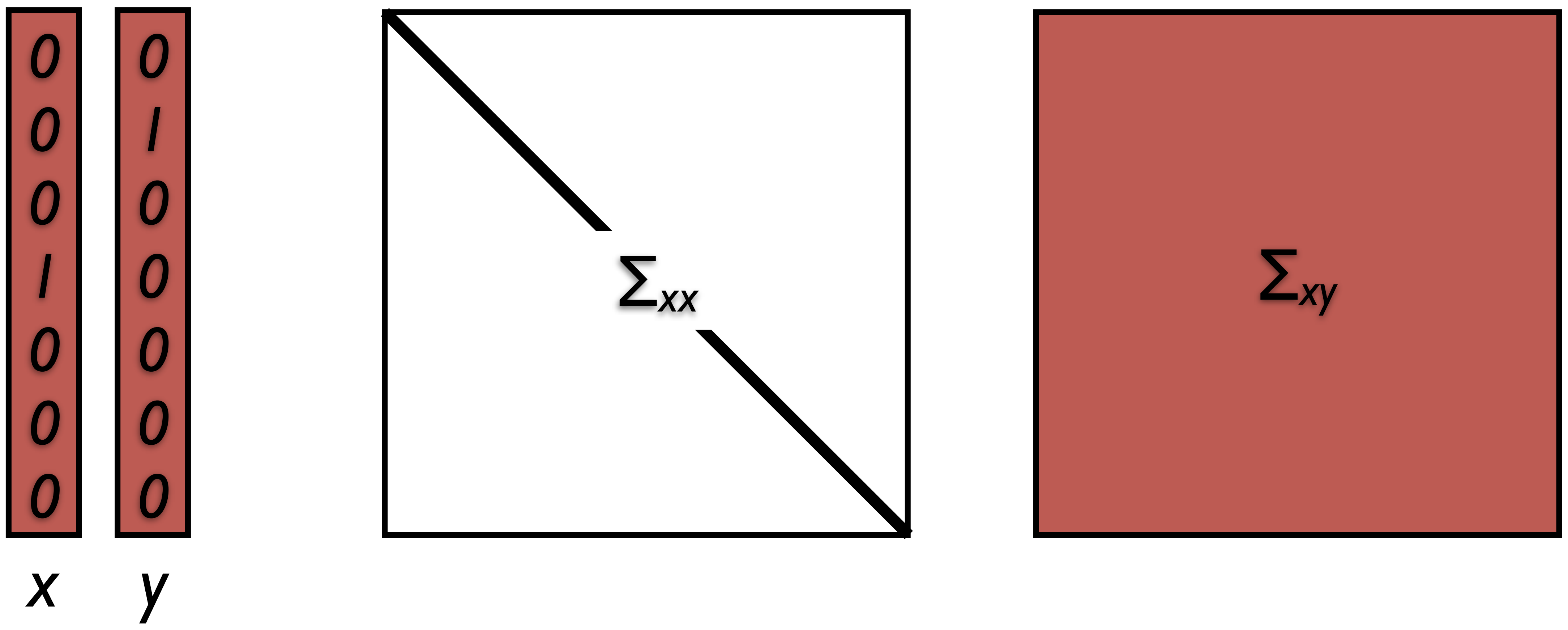
Another simple case

$$\mathbb{P}(y \mid x) = \mathbb{P}(x, y)/\mathbb{P}(x)$$



small number of discrete outcomes: P(x,y) specified by a probability table, can implement Bayes rule exactly by iterating over table

Inference



$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \boxed{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\Sigma_{xx} = \mathbb{E}(xx^\top) \qquad \Sigma_{xy} = \mathbb{E}(xy^\top)$$

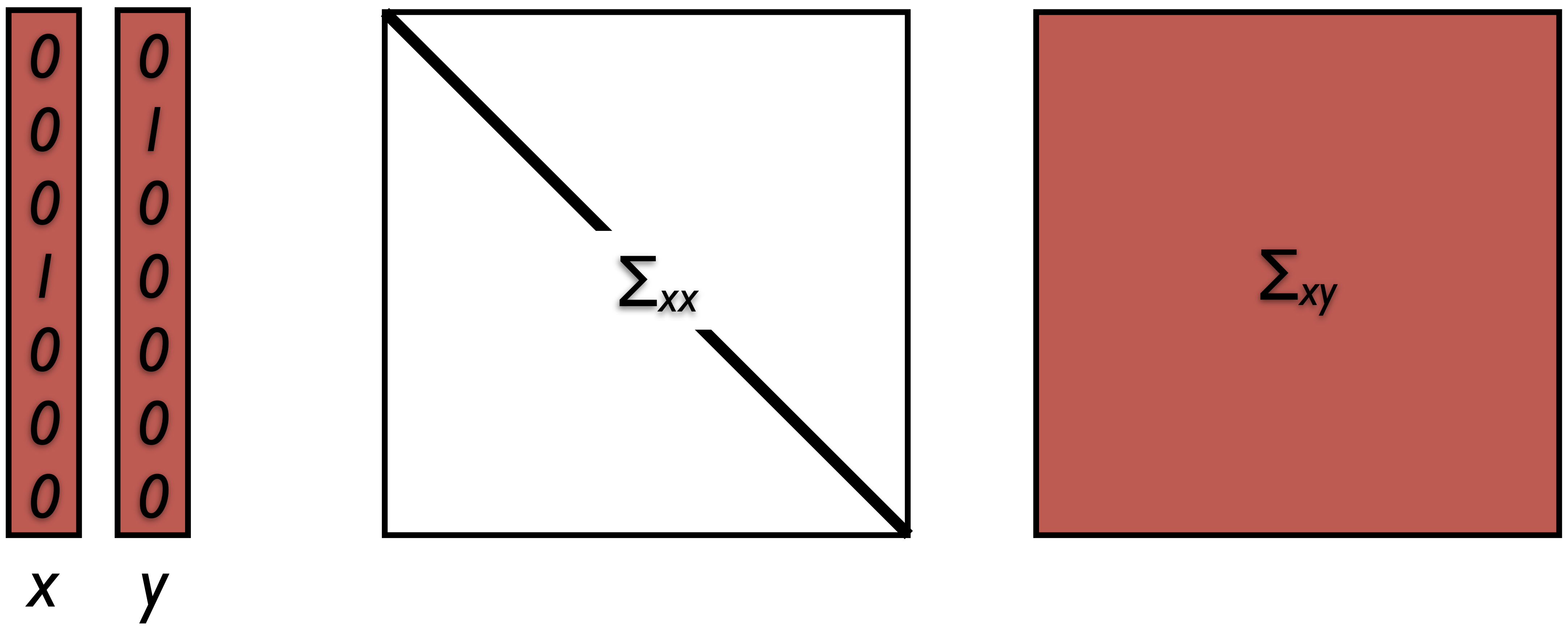
$$[\Sigma_{xx}]_{ii} = \mathbb{P}(x = e_i) \qquad [\Sigma_{xy}]_{ij} = \mathbb{P}(x = e_i, y = e_j)$$

$$\mathbb{P}(x) = x^\top \Sigma_{xx} x \qquad \mathbb{P}(x, y) = x^\top \Sigma_{xy} y$$

To connect to Gaussian case, let's look at covariances

Can write everything needed for Bayes rule in terms of Σ_{xx} and Σ_{xy}

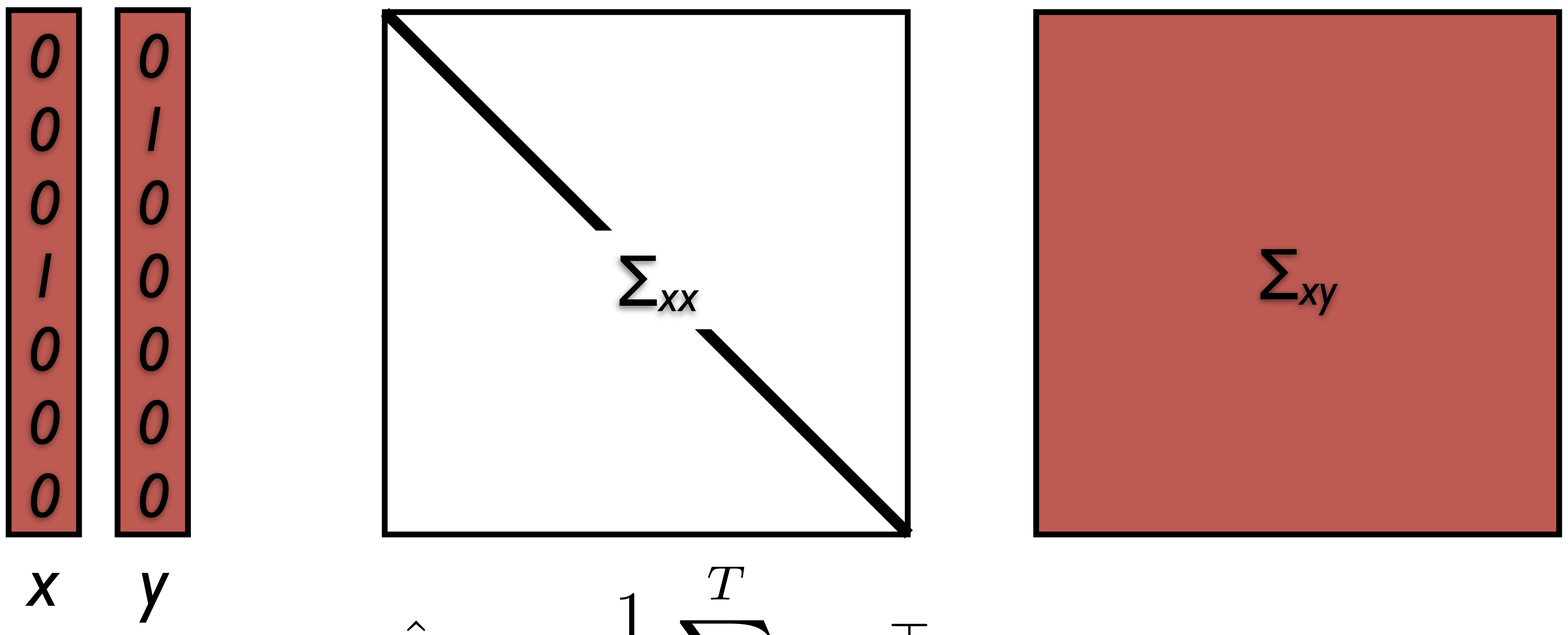
Inference



$$\begin{aligned}\mathbb{P}(y = e_j \mid x = e_i) &= \mathbb{P}(x = e_i, y = e_j)/\mathbb{P}(x = e_i)\\ &= [\Sigma_{xy}]_{ij}/[\Sigma_{xx}]_{ii}\\ \mathbb{P}(y \mid x) &= x^\top \Sigma_{xx}^{-1} \Sigma_{xy} y\end{aligned}$$

Bayes rule: matrix version

Learning



$$\hat{\Sigma}_{xy} = \frac{1}{T}\sum_{t=1}^{T} x_t y_t^\top$$

$$\hat{\Sigma}_{xx} = \frac{1}{T}\sum_{t=1}^{T} x_t x_t^\top$$

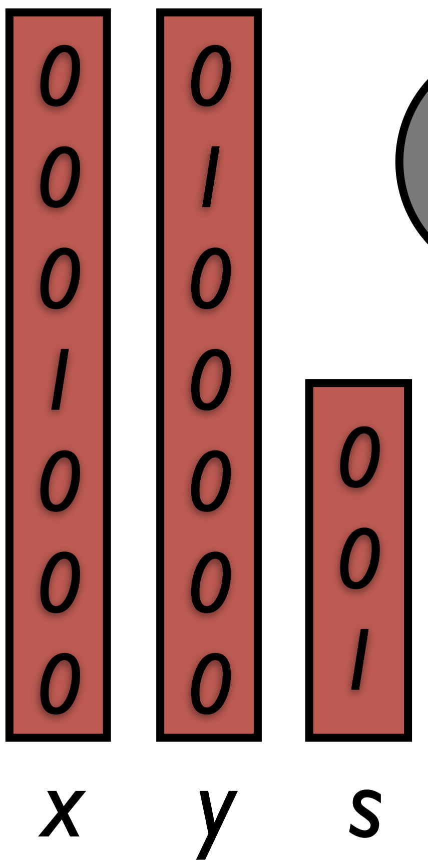
$$\hat{\mathbb{P}}(y \mid x) = x^\top \hat{\Sigma}_{xx}^{-1} \hat{\Sigma}_{xy} y$$

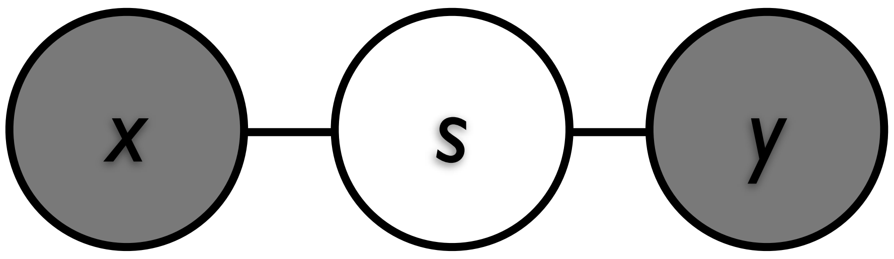
empirical matrix
Bayes rule
(consistent)

these are same formulas as for Gaussian case

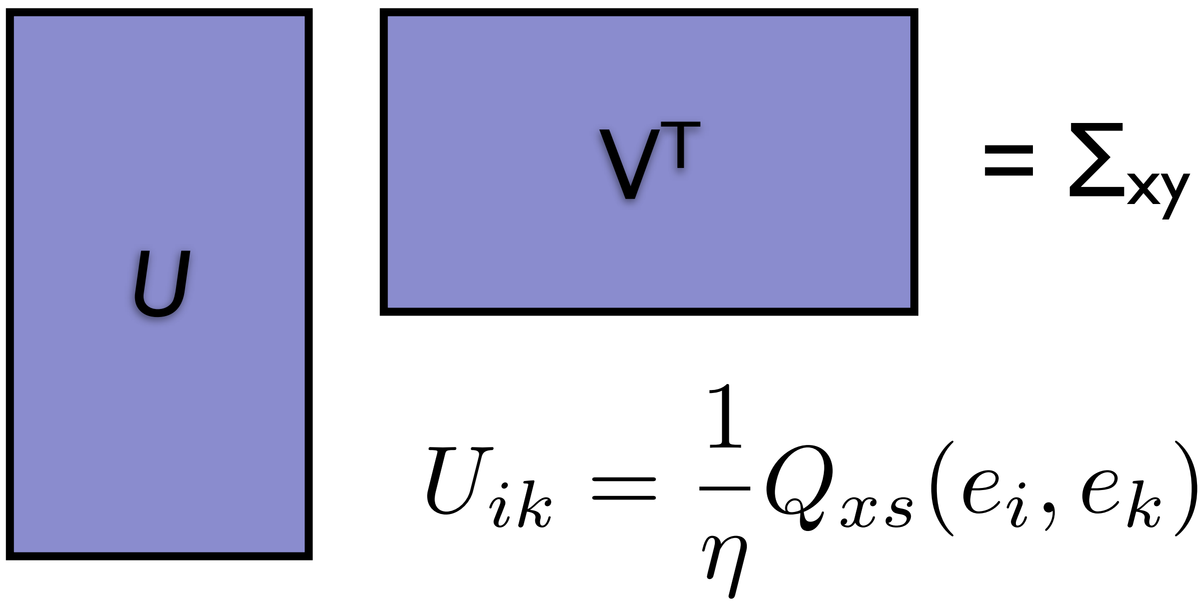
optional: can regularize Σ_{xx} by adding a ridge term λI (and possibly rescaling to maintain sum=1)

Slightly more complicated graph





x, y, s discrete

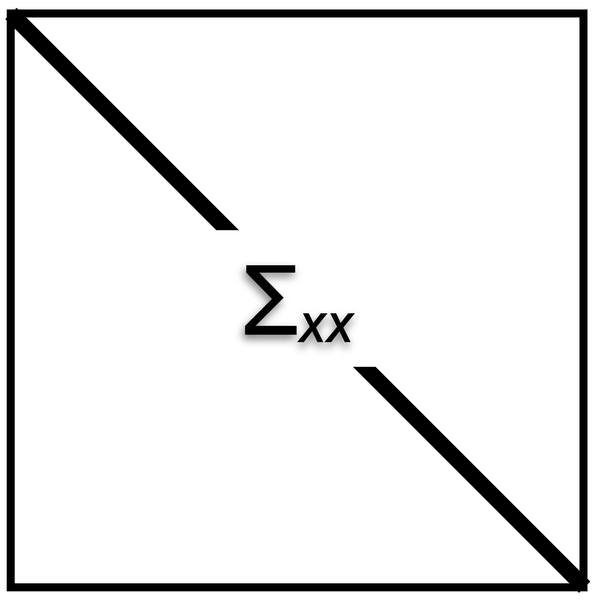


$$U_{ik} = \frac{1}{\eta} Q_{xs}(e_i, e_k)$$

$$V_{jk} = Q_{ys}(e_j, e_k)$$

$$\mathbb{P}(x, y, s) = \frac{1}{\eta} Q_{xs}(x, s) Q_{ys}(y, s)$$

$$\mathbb{P}(x, y) = \sum_s \frac{1}{\eta} Q_{xs}(x, s) Q_{ys}(y, s)$$



Start to work our way up in graph complexity: suppose a latent variable z separates x from y

then something interesting happens: Σ_{xy} is low rank

just as before, Σ_{xx} is diagonal

Inference and learning

$$\mathbb{P}(y = e_j \mid x = e_i) = [\Sigma_{xy}]_{ij} / [\Sigma_{xx}]_{ii}$$
$$\mathbb{P}(y \mid x) = x^\top \Sigma_{xx}^{-1} \Sigma_{xy} y$$

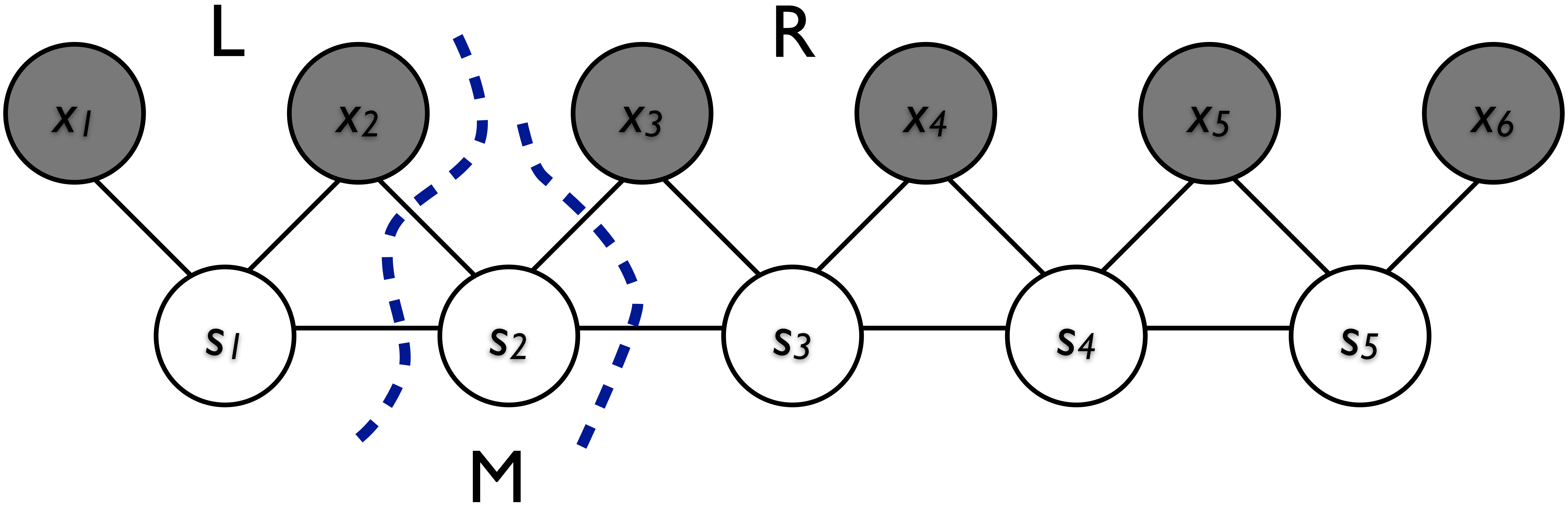
inference by normal equations

$$\hat{\Sigma}_{xy} = \mathrm{rank}_k \left(\frac{1}{T} \sum_{t=1}^{T} x_t y_t^\top \right)$$
$$\hat{\Sigma}_{xx} = \frac{1}{T} \sum_{t=1}^{T} x_t x_t^\top$$

learning: project empirical covariance onto rank k

the learned rule is exact if k ≥ rank(true Σ_{xy}), but clearly we can still use it as an approximation if not

So what do we do w/ a real graph?

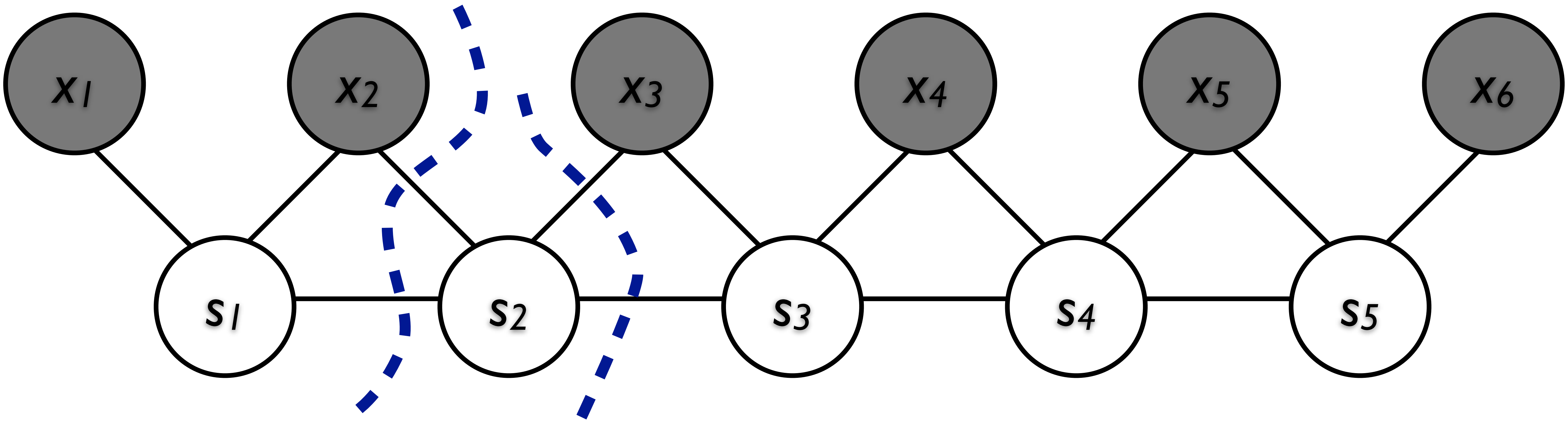


Goal: ***Bayes filter***

Given ***belief*** $P(s_2 \mid x_{12})$: dist'n over a cut, given left context

Update recursively: get belief $P(s_3 \mid x_{123})$

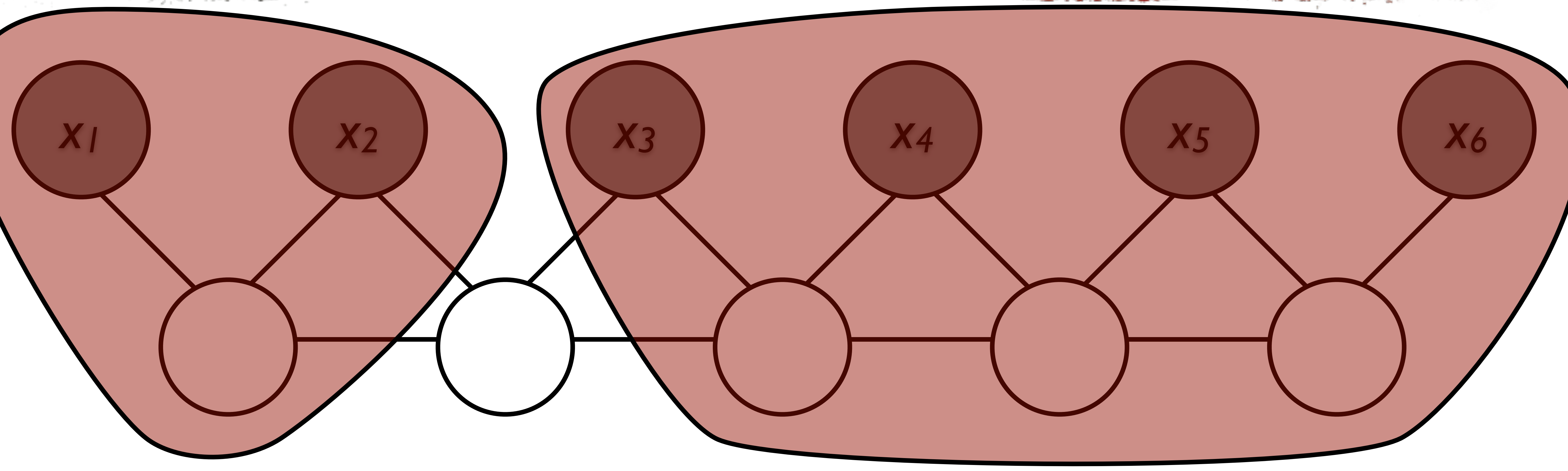
Bayes filter

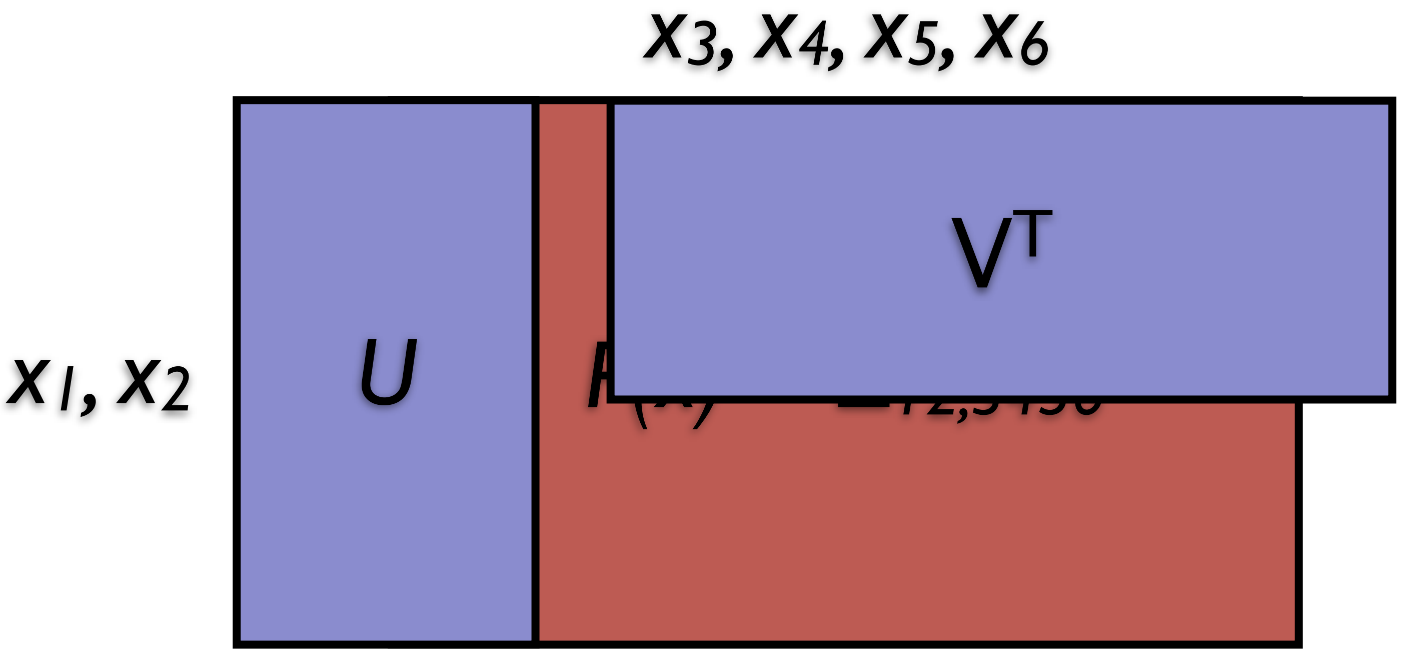


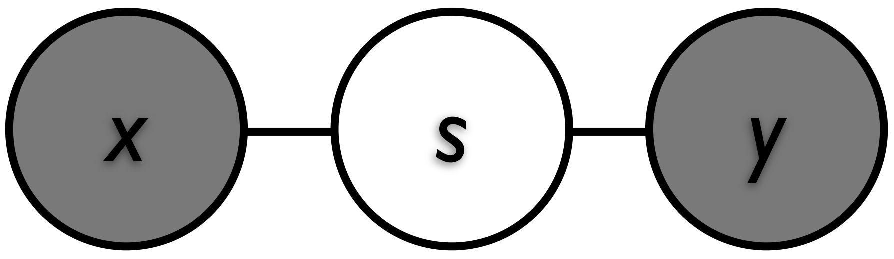
- ***Extend***: $P(s_2, x_3, s_3 \mid \ldots) = P(s_2 \mid \ldots)\ P(s_3, x_3 \mid s_2)$
- ***Marginalize***: drop s_2 to get $P(x_3, s_3 \mid \ldots)$
- ***Condition***: $P(s_3 \mid \ldots) = P(x_3, s_3 \mid \ldots) / P(x_3 \mid \ldots)$

We'll show how to do each of these steps efficiently; the result will look like repeated application of the matrix Bayes rule. (And as above this result is exact if we use a big enough rank, or approximate if we pick a smaller rank.)

So what do we do w/ a real graph?







If we group variables, we get right back to X-S-Y case

Write P(x) as a matrix: it is covariance of x_{12} vs x_{3456}. Each row is an event x_{12} = something; each column is an event x_{3456} = something.

This matrix is low-rank (by same argument above)

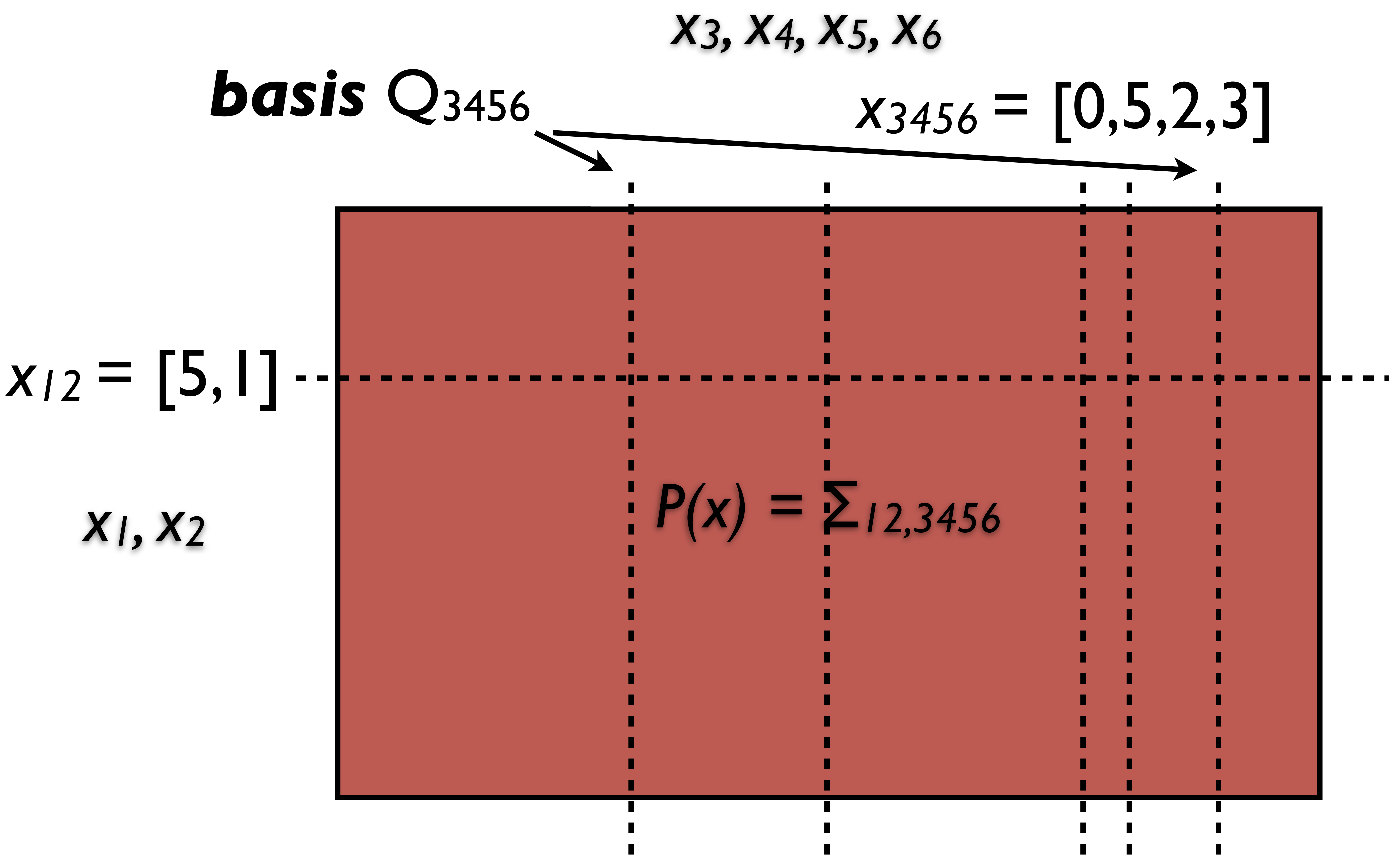
So, if we have $U^T * x_{12}$, that's enough to compute P(x) for any value of x matching x_{12} -- we don't need the actual value of x_{12}

Similarly, can compute $P(x_{3456} | x_{12})$ by normalizing this slice of P(x)

Problem: can't actually write down a table as big as $P(x_{3456})$ -- much less $P(x_{123456})$

So what do we do?

Cutting down on storage



Say rank is 5

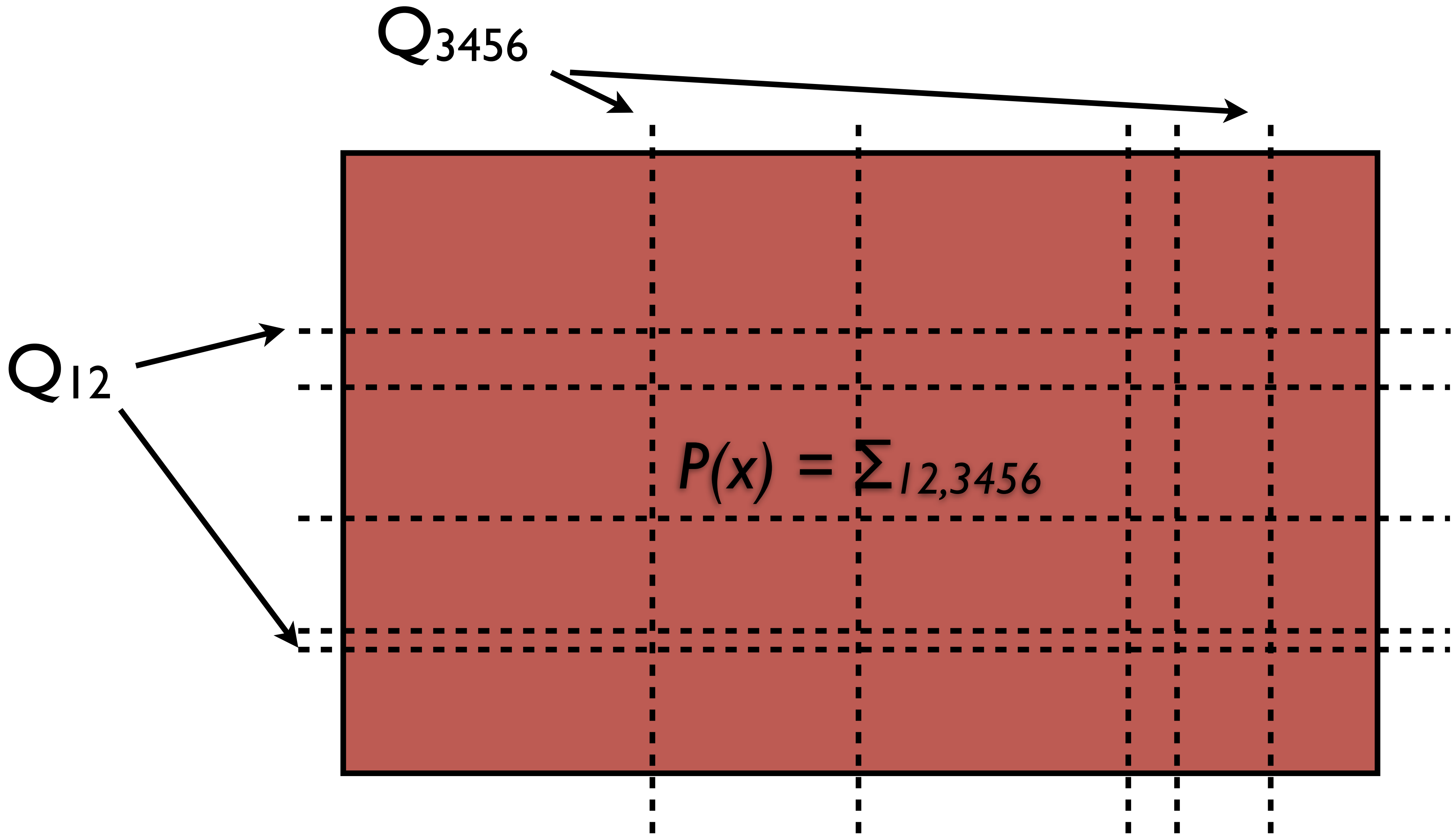
Then we can find 5 linearly independent columns (a basis for the column space) -- each column gives probabilities of all settings of x_{12} for one setting of x_{3456}

And we can reconstruct any other column as a linear combination of this basis

Now we can work with the (much smaller) matrix which has only the basis columns, and we'll be able to reconstruct any other desired column as needed

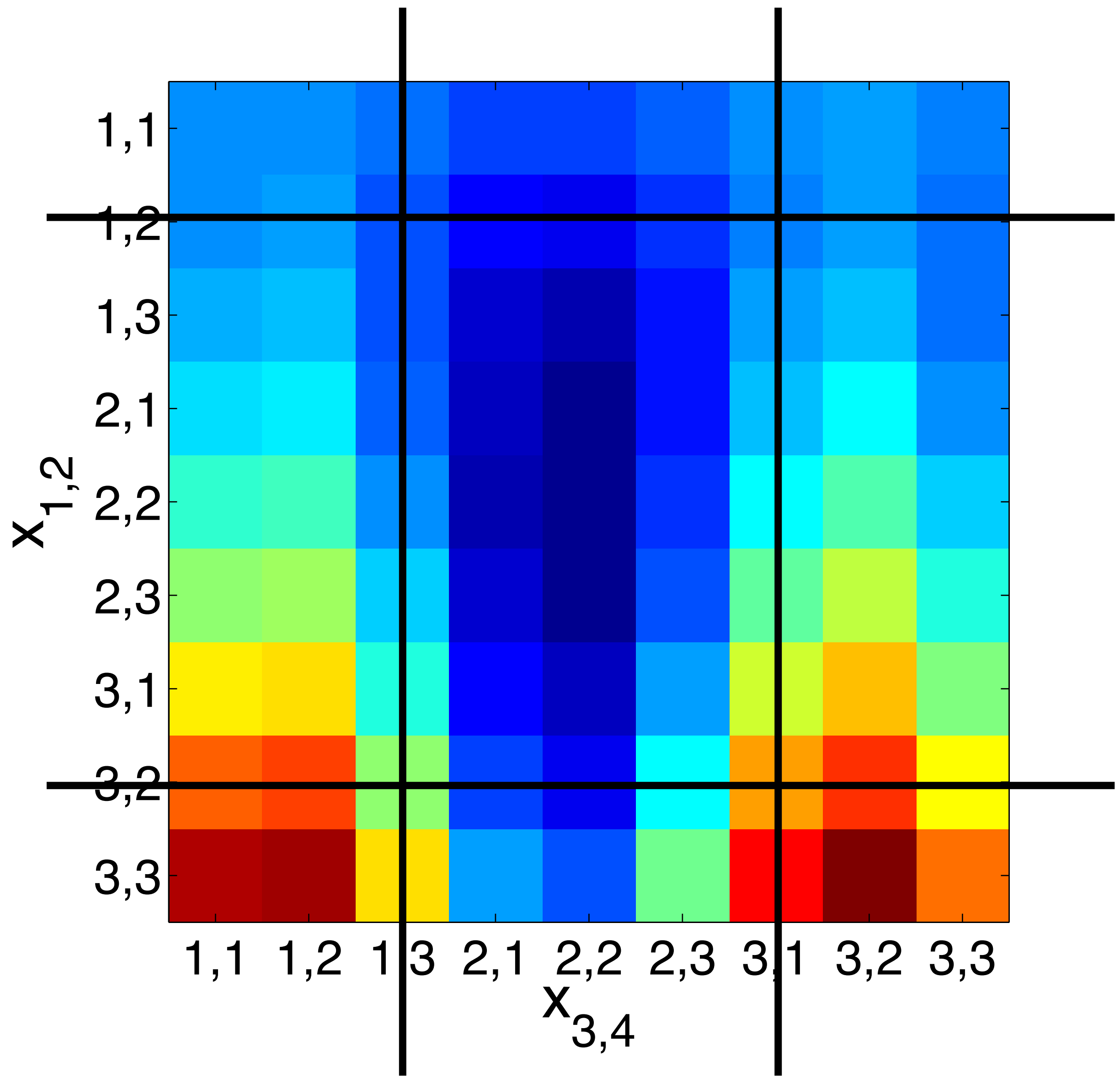
E.g., our belief will be represented as a prediction of probabilities for the events corresponding to basis columns -- this is our predictive state

It works for rows too

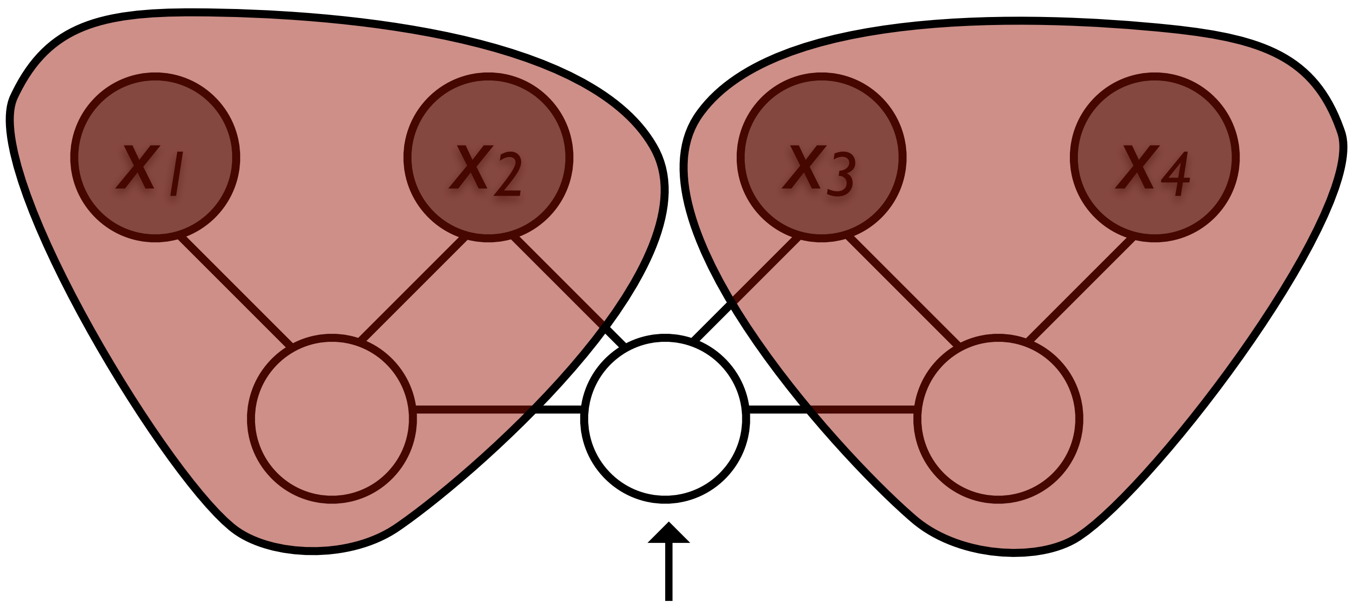


Same idea works for rows, so we only need a small square submatrix of Σ

Numerical example

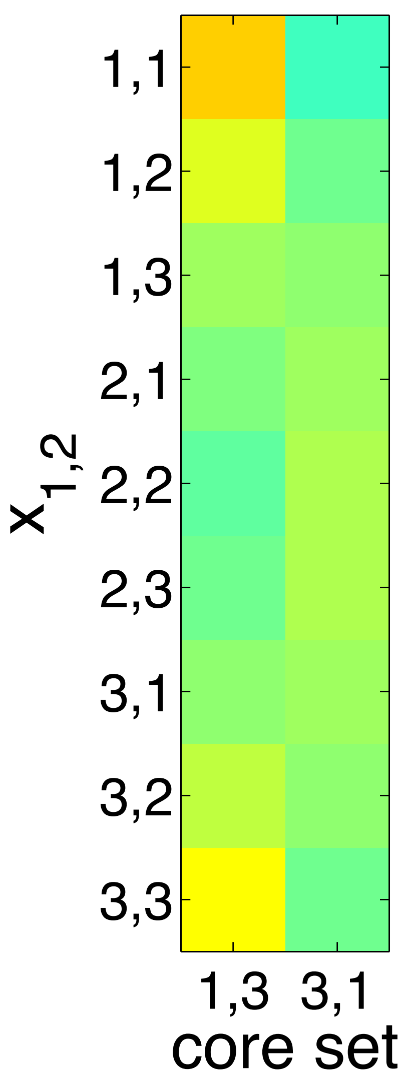


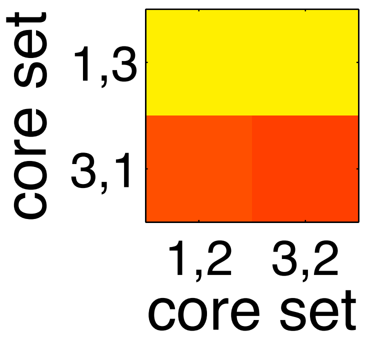
this matrix ↑ has rank 2

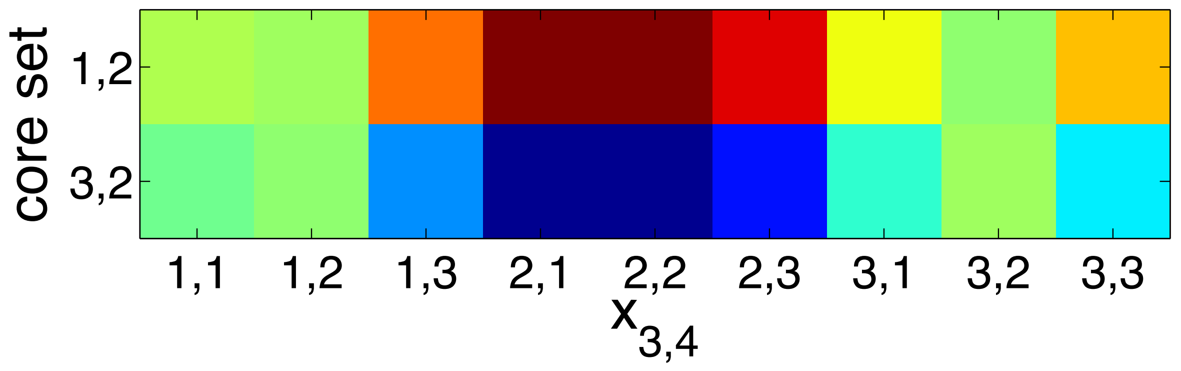


since this variable has arity 2

Numerical example







↑ *R^T = regress from B to P(core, :)*

↑ *B = basis subset of P*

$$\boxed{P = LBR^T}$$

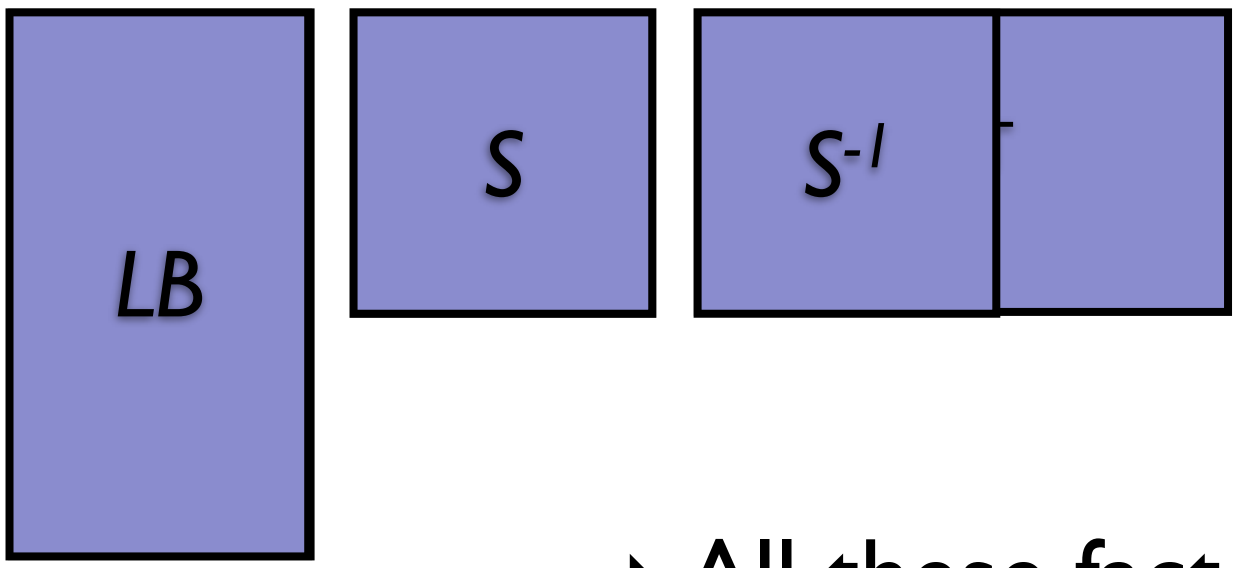
↑ *L = regress from B to P(:, core)*

we’ve represented the entire distribution P as the product of these 3 matrices -- many fewer parameters

If you’ve heard PSR terminology “core events” and “indicative events”, these are other names for a basis set

note: common caxis, except B, which is factor of 500 larger

Ambiguity

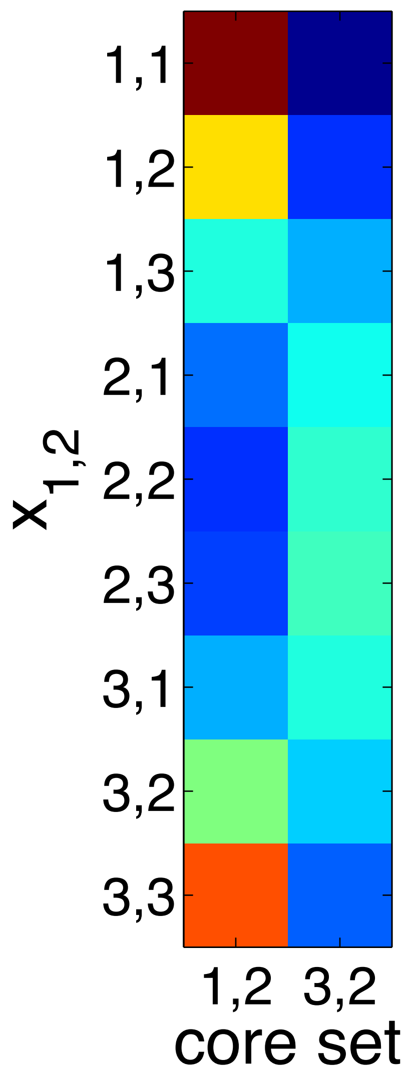


- All these factorizations are equivalent
- Each corresponds to a linear transform of the basis
- Instead of P(basis event), E(basis statistic)

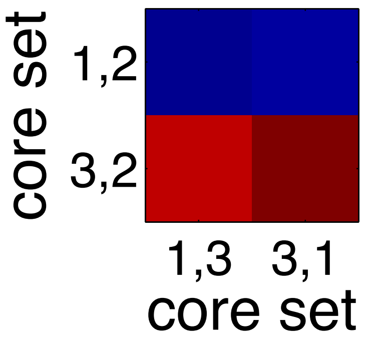
$$\sum_{x_1,x_2} \mathbb{P}(x_1,x_2)f(x_1,x_2)$$

We can rearrange factorization:
L B R^T
(LB) R^T
(LB) S S^{-1} R^T

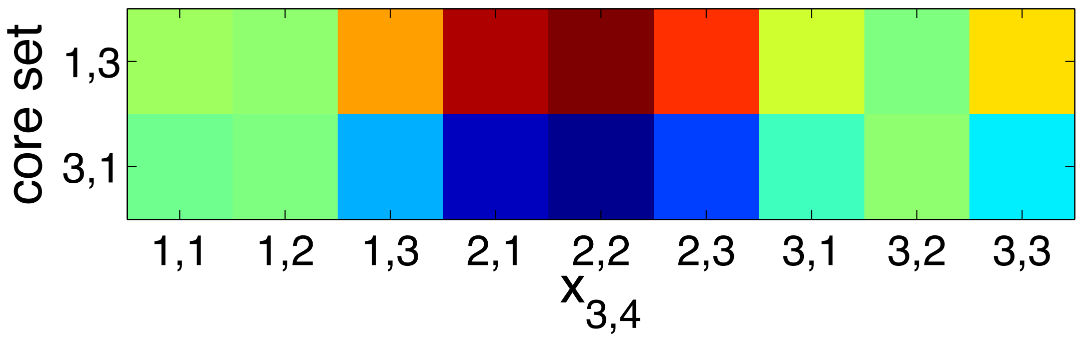
Learning and inference



L



B



R

Learn L and R by SVD

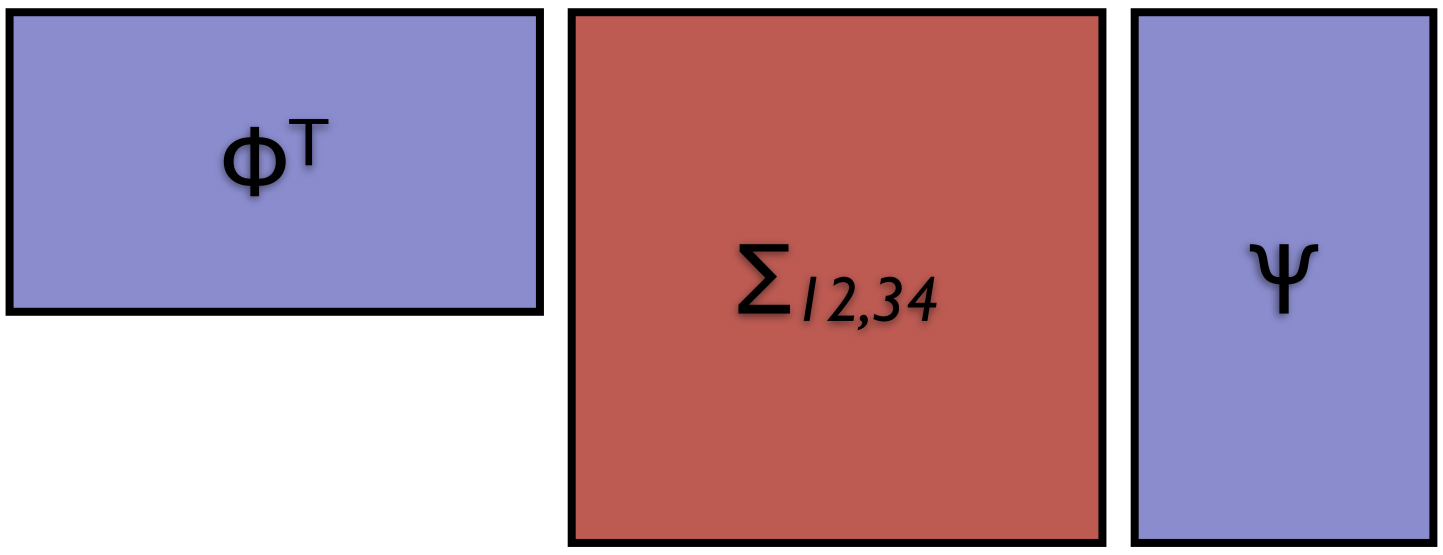
If desired, learn B by counting co-occurrences of core events, adjust L and R

Inference by usual formula:

$$\mathbb{P}(x_{3,4} \mid x_{1,2}) = x_{12}^\top \Sigma_{12,12}^{-1} \Sigma_{12,34} x_{34}$$

In reality, though, for big sets of variables, Σ_{xy} will be too big to write down

Features



$$\Phi^\top \hat{\Sigma}_{xy} \Psi = \mathrm{rank}_k \left(\frac{1}{T} \sum_{t=1}^{T} \phi_t \psi_t^\top \right)$$

$x_{1,2}$	x_1+x_2	x_2	...
1, 1	2	1	...
1, 2	3	2	...
1, 3	4	3	...
2, 1	3	1	...
2, 2	4	2	...
2, 3	5	3	...
3, 1	4	1	...
3, 2	5	2	...
3, 3	6	3	...

Φ

So, define features of x_{12} and x_{34} -- call them phi and psi, and suppose there are ≥k of them on each side -- each feature is a linear combination of indicators for events

Compute empirical covariance of features, and project onto rank k

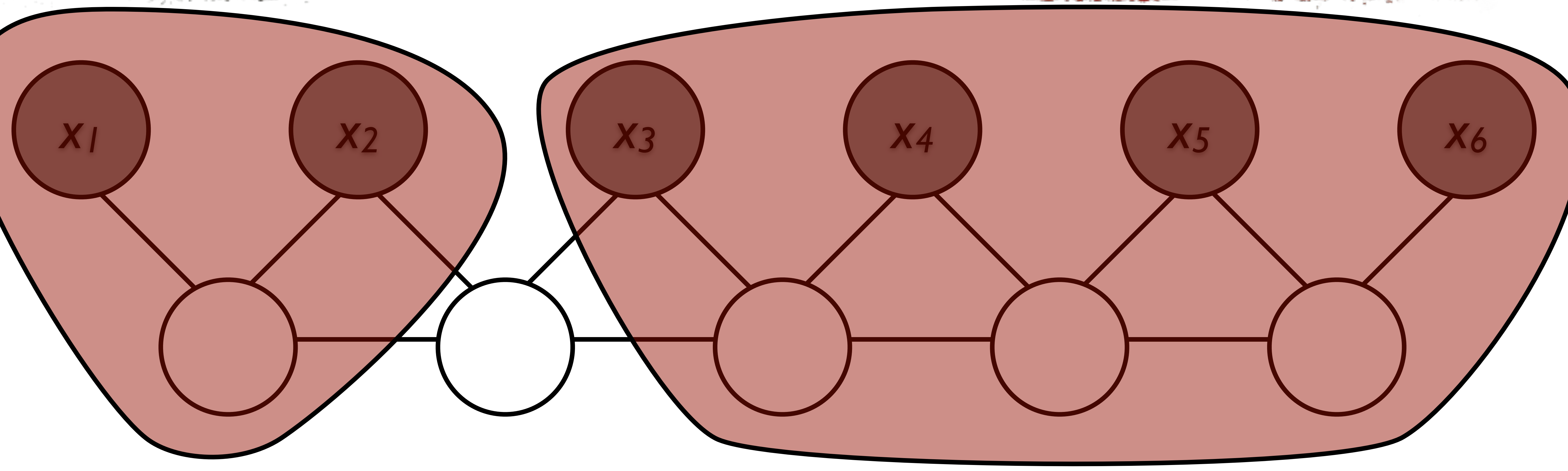
big advantage: we never need to represent a PDF over domain of X or Y, which can be huge -- instead work only with shorter feature vectors

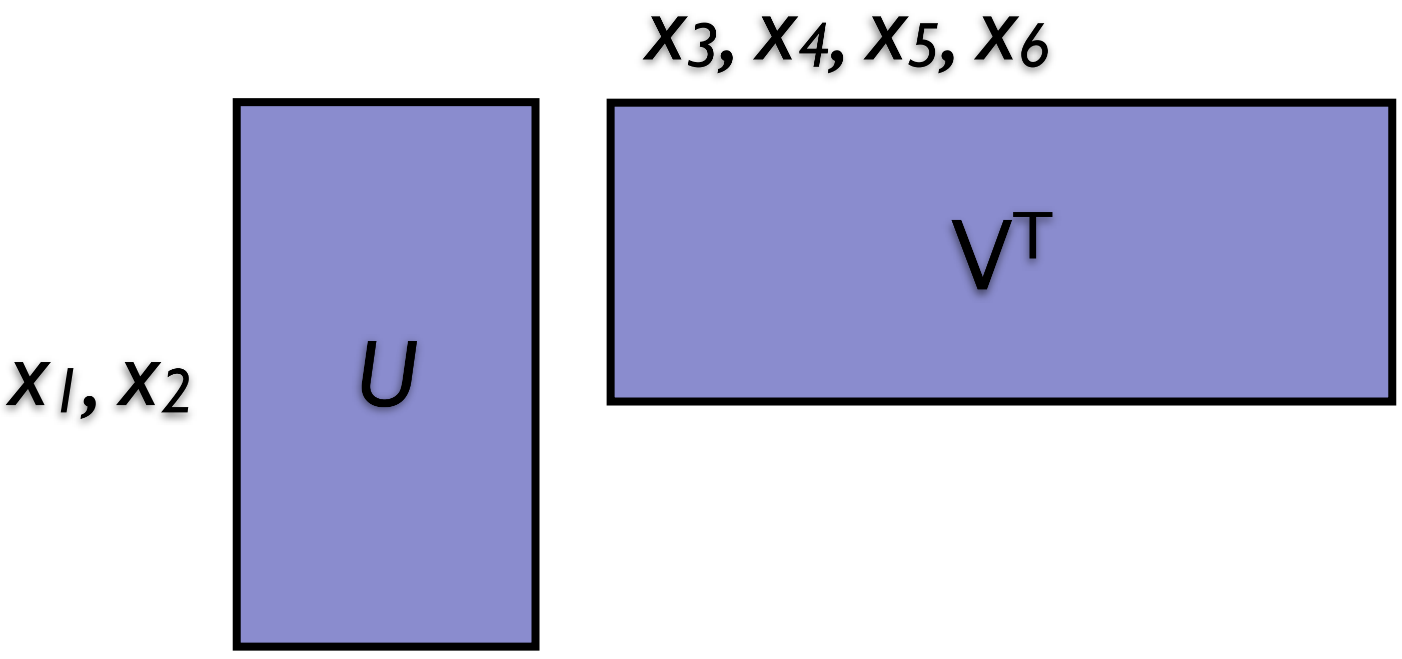
big advantage #2: fixes a numerical/statistical problem. If each core event has small probability (the usual case, since it's a conjunction of settings for many variables), then it might take a lot of data to estimate the core submatrix, and it might be ill-conditioned to reconstruct other event probabilities from the core events. But as long as features are not (nearly) perfectly correlated, projecting feature covariance onto rank k is well-conditioned.

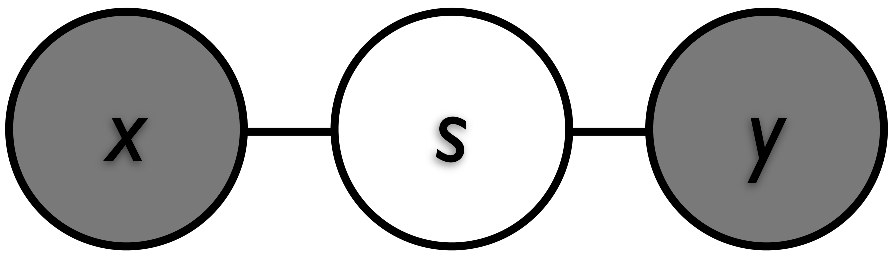
Problem: can't do this with Σ_{xx} (it's not low-rank)

Solution: we'll use Bayes filter to condition on observed variables one at a time, instead of all at once

OK, we can work with groups

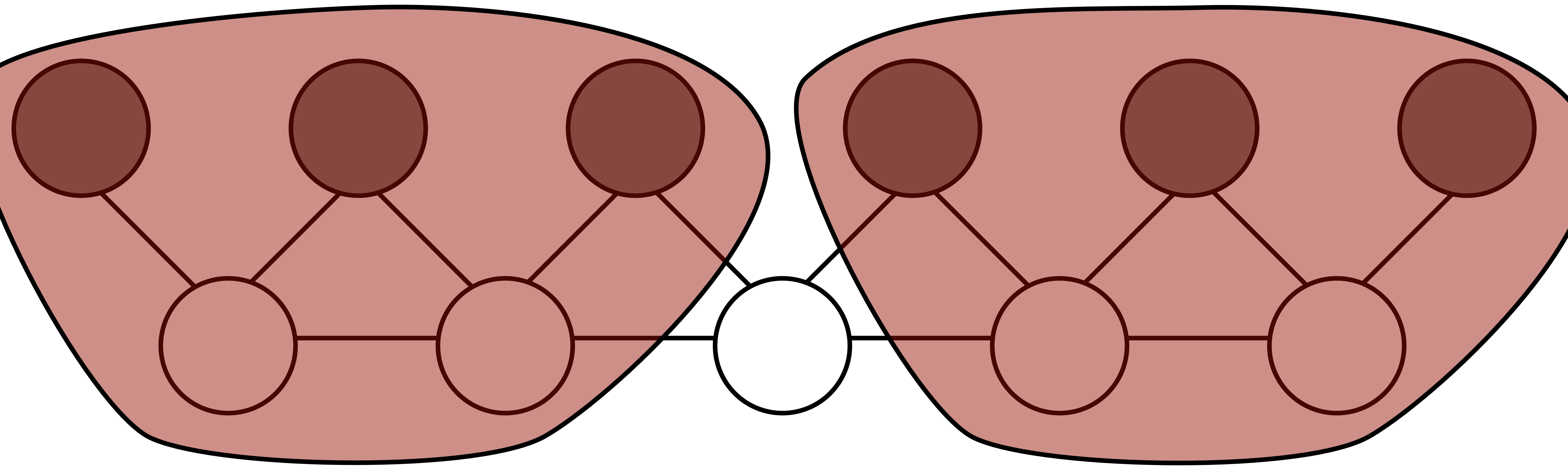


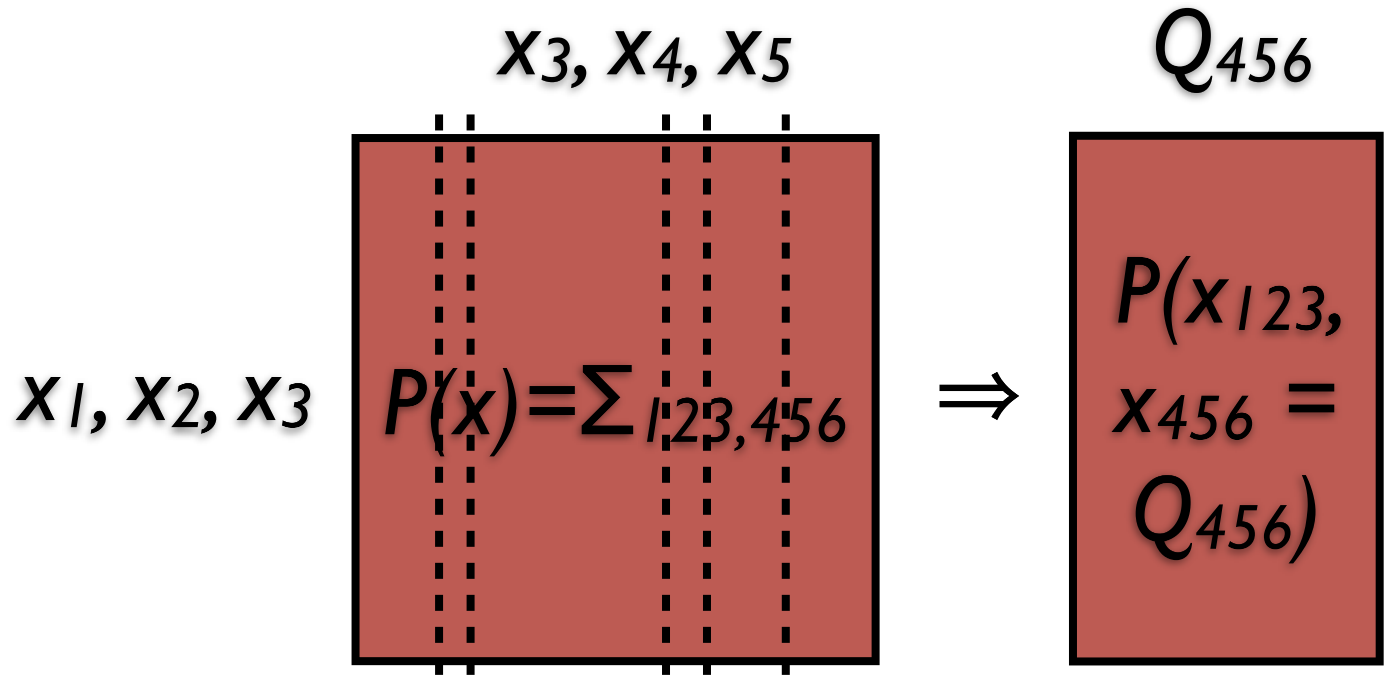




So now we’ve shown how to work efficiently with this grouping

Do it again



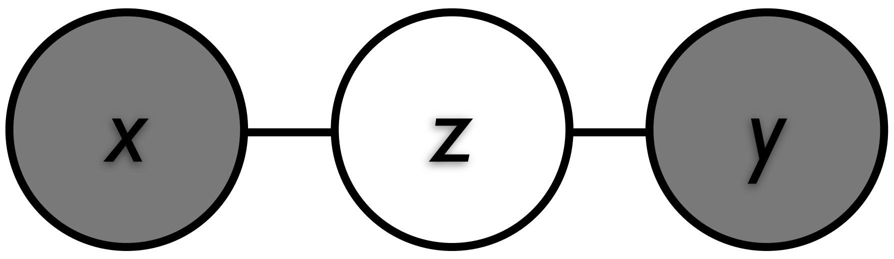


x_3, x_4, x_5

Q_{456}

x_1, x_2, x_3

$P(x)=\Sigma_{123,456}$ $\Rightarrow$ $P(x_{123}, x_{456} = Q_{456})$

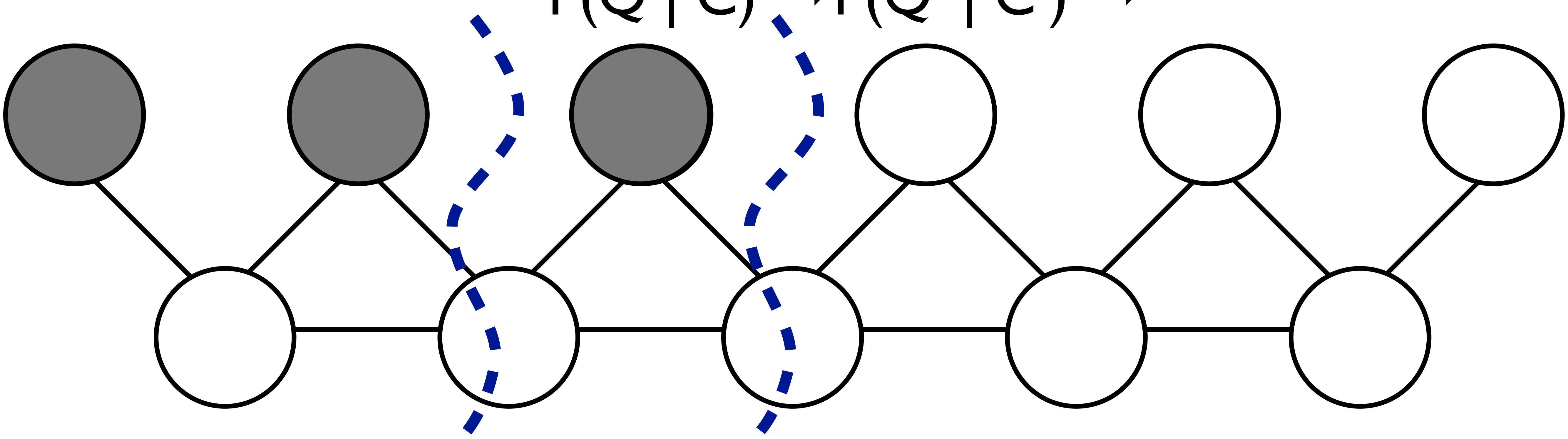


we can also regroup variables, and find a basis set for x_{345}

or, if desired, a transformed basis (not just a subset of atomic events)

Bayes filter

P(Q | C) → P(Q’ | C’) →



context C context C’

$q \in Q$
$q' \in Q'$

P(Q | C) =
vector of P(q | C)

xq’ = (x, then q’): e.g., if
q’ is x_{456} = (4, 2, 7), then
5q’ is x_{3456} = (5, 4, 2, 7)

We are given a belief over a core set: P(Q | context C) -- this is a short vector -- length 5 in our example

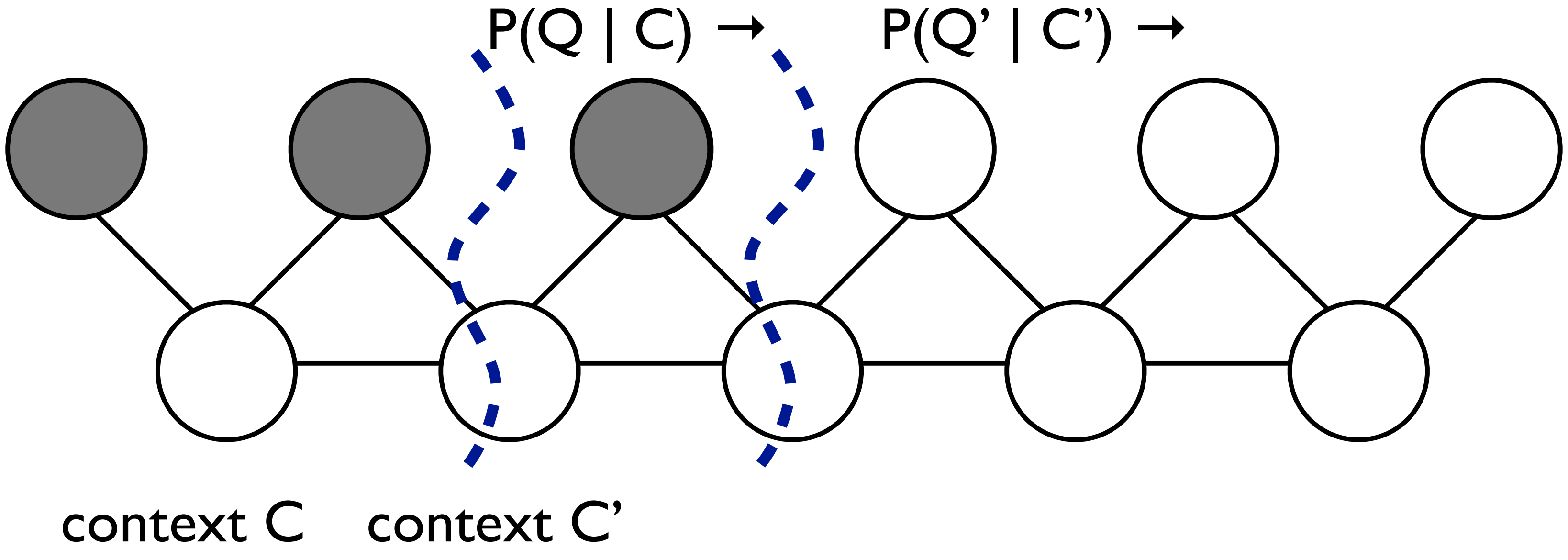
Given an observation x_3 = x, we need to update: P(Q’ | C’) where C’ = (context C, observation x)

Marginalize: not really clear where this is happening since latents are never explicitly represented

Bayes filter

P(Q | C) =
vector of P(q | C)

xq' = (x, then q'): e.g., if q' is
x_{456} = (4, 2, 7), then 5q' is
x_{3456} = (5, 4, 2, 7)



$$\forall q' \in Q', \forall \text{ symbols } x, \exists m_{xq'} \in \mathbb{R}^k,$$
$$P(xq' \mid C) = m_{xq'}^\top P(Q \mid C)$$

Extend: by assumption, for any q_j' in Q', P(x, q_j' | context C) is a linear function of P(Q | C)

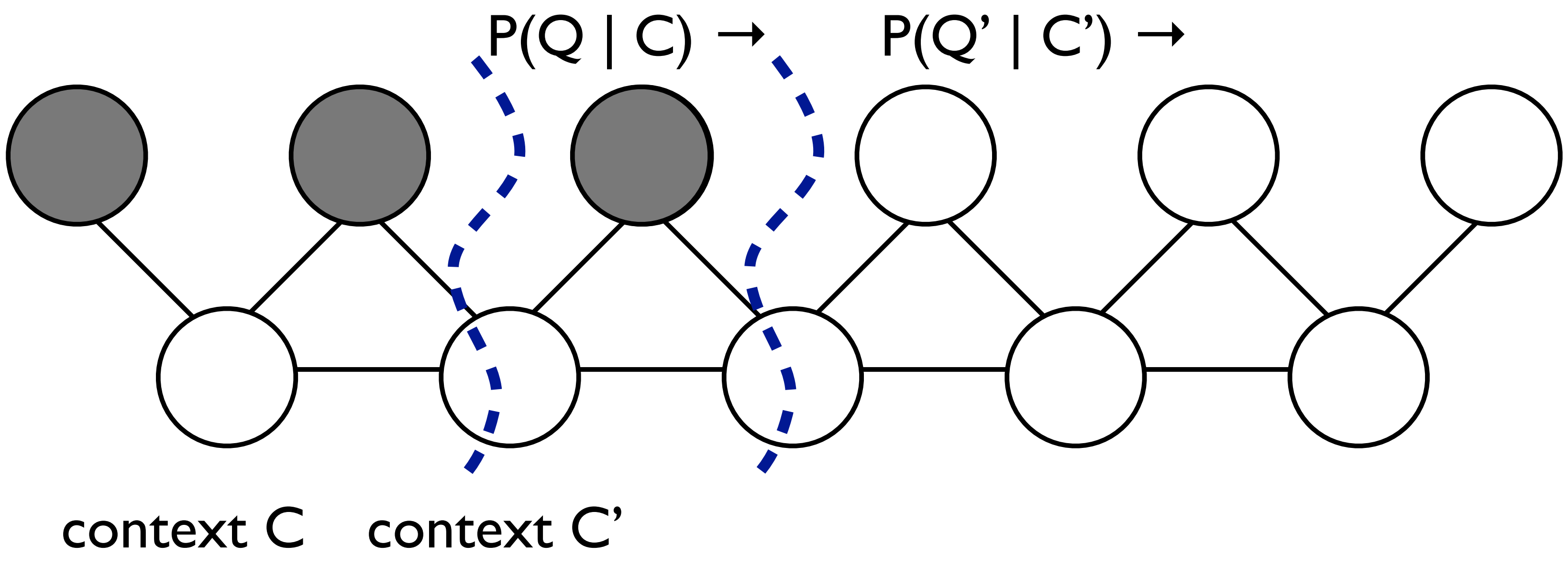
in symbols, P(xQ' | C) = M_x P(Q | C) for some matrix M_x

again by assumption, P(x | C) = m_x P(Q | C) for some vector m_x

Bayes filter

P(Q | C) =
vector of P(q | C)

xq' = (x, then q'): e.g., if q' is
x_{456} = (4, 2, 7), then 5q' is
x_{3456} = (5, 4, 2, 7)

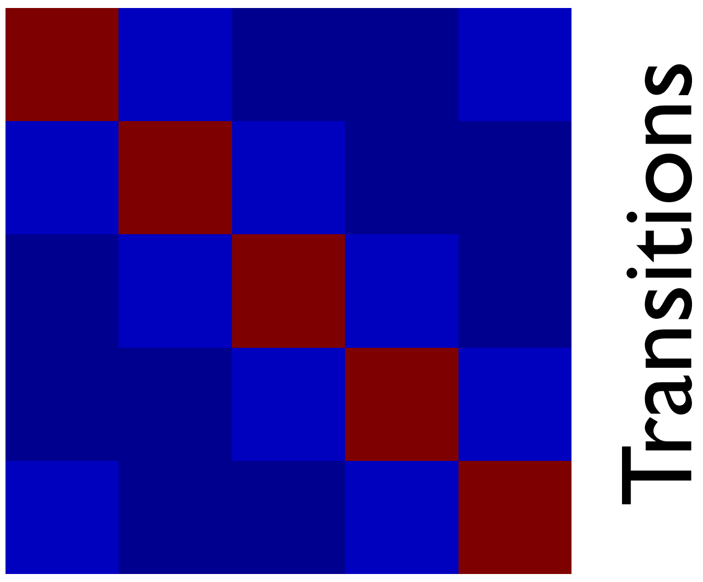


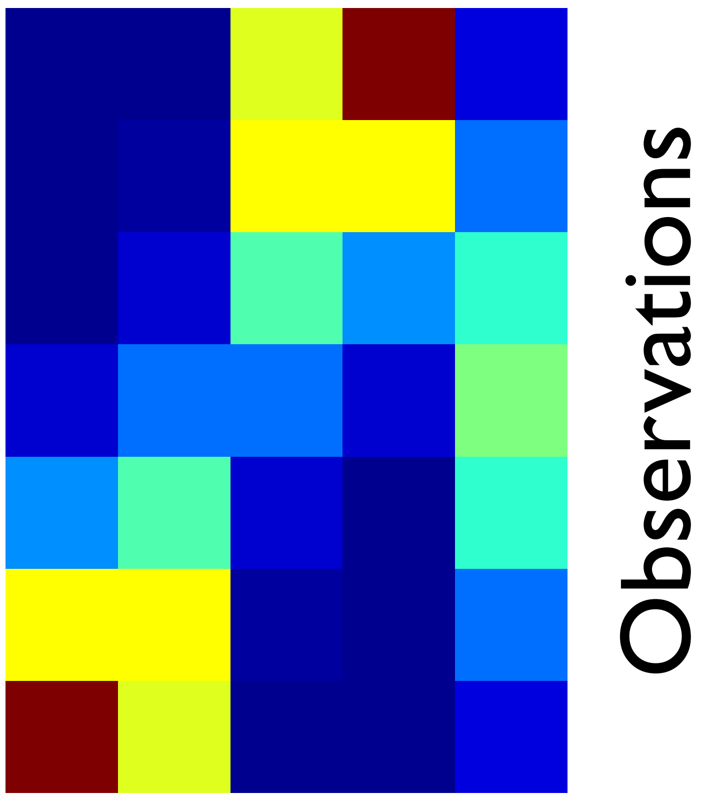
$$\begin{aligned} P(q' \mid C') &= P(q' \mid C, x) \\ &= P(xq' \mid C) \;/\; P(x \mid C) \\ &= m_{xq'}^{\top} P(Q \mid C) \;/\; m_x^{\top} P(Q \mid C) \end{aligned}$$

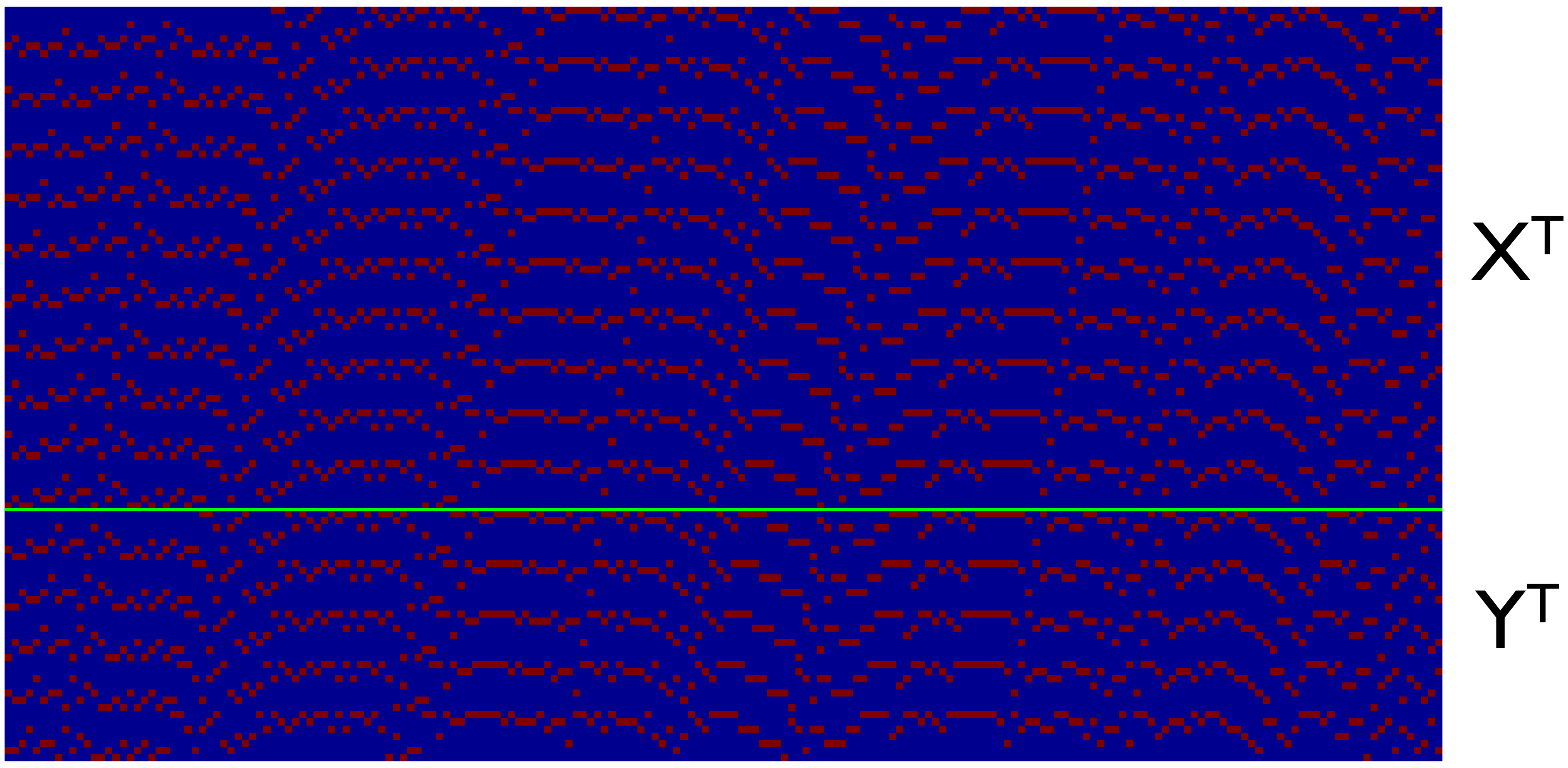
Condition: Bayes rule: $P(q_j \mid C, x) = P(x, q_j \mid C) / P(x \mid C)$

Or, all at once: $P(Q \mid C') = M_x P(Q \mid C) / m_x P(Q \mid C)$

Does it work?

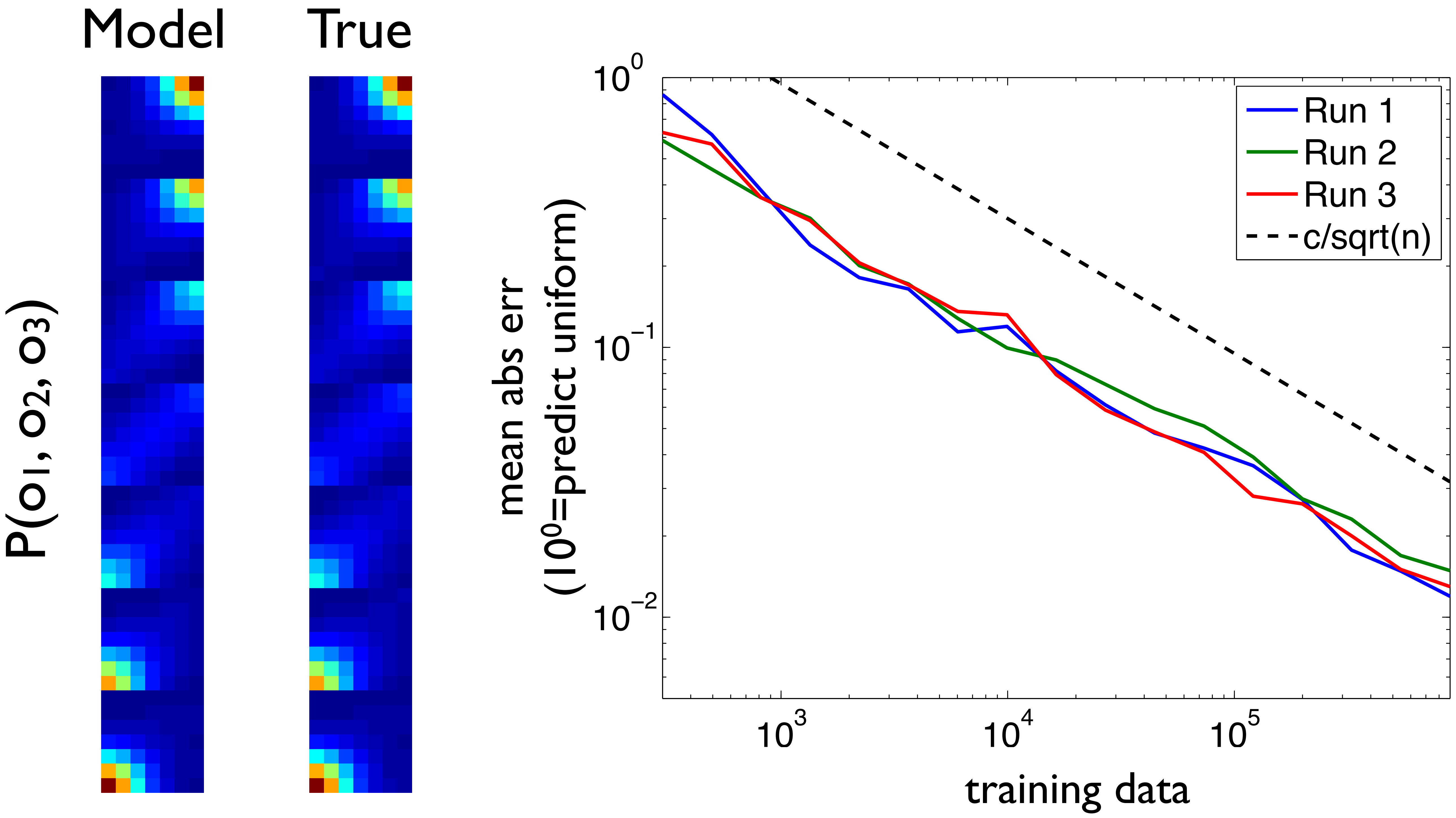






- Simple HMM: 5 states, 7 observations
- N=300 thru N=900k

Does it work?



In heat maps,
first observation: major blocks
2nd: up&down within block
3rd: across rows

Discussion

- ***Impossibility***—learning DFA in poly time = breaking crypto primitives [Kearns & Valiant 89]
  - so, clearly, we can't always be statistically efficient
  - but, see McDonald [11], HKZ [09], us [09]: convergence depends on ***mixing rate***
- ***Nonlinearity***—Bayes filter update is highly nonlinear in state (matrix inverse), even though we use a *linear* regression to identify the model
  - this is essential for expressiveness (e.g., clock)
- ***Infinite memory horizon***—even if we learn from a finite window

there are data sets from which it would be possible to learn a good dynamical system (using arbitrary computational power) where we learn little or nothing

failure mode: we need exponentially more data than would be possible with arbitrary computational power—contrast breaking RSA by factoring vs. by building a table of (plaintext, cyphertext) pairs

any proof of efficiency would need to use some parameter such as beta-mixing which excludes "bad" HMMs

Summary so far

- We can write Bayes rule updates in terms of covariances
  - Often, they are low rank
  - We can learn them from data
- We can chain Bayes rule updates together to make a Bayes filter
- Lots of cool applications of this simple idea