$\LaTeX$ command declarations here.

In [1]:

```
from __future__ import division

# scientific
%matplotlib inline
from matplotlib import pyplot as plt;
import numpy as np;
import sklearn as skl;
import sklearn.datasets;
import sklearn.cluster;

# ipython
import IPython;

# python
import os;

##########################################################

# image processing
import PIL;

# trim and scale images
def trim(im, percent=100):
    print("trim:", percent);
    bg = PIL.Image.new(im.mode, im.size, im.getpixel((0,0)))
    diff = PIL.ImageChops.difference(im, bg)
    diff = PIL.ImageChops.add(diff, diff, 2.0, -100)
    bbox = diff.getbbox()
    if bbox:
        x = im.crop(bbox)
        return x.resize(((x.size[0]*percent)//100, (x.size[1]*percent)//1
 PIL.Image.ANTIALIAS);


#########################################################

# daft (rendering PGMs)
import daft;

# set to FALSE to load PGMs from static images
RENDER_PGMS = False;

# decorator for pgm rendering
def pgm_render(pgm_func):
    def render_func(path, percent=100, render=None, *args, **kwargs):
        print("render_func:", percent);
        # render
        render = render if (render is not None) else RENDER_PGMS;

        if render:
            print("rendering");
            # render
            pgm = pgm_func(*args, **kwargs);
            pgm.render();
            pgm.figure.savefig(path, dpi=300);

            # trim
```

```
            img = trim(PIL.Image.open(path), percent);
            img.save(path, 'PNG');
        else:
            print("not rendering");

        # error
        if not os.path.isfile(path):
            raise "Error:  Graphical model image %s not found.  You may
 need to set RENDER_PGMS=True.";

        # display
        return IPython.display.Image(filename=path);#trim(PIL.Image.open
(path), percent);

    return render_func;

##########################################################
```

EECS 445: Machine Learning

Lecture 16: Latent Variables, d-Separation, Gaussian Mixture Models

- Instructor: **Jacob Abernethy**
- Date: November 9, 2016

References

- **[MLAPP]** Murphy, Kevin. *Machine Learning: A Probabilistic Perspective* (https://mitpress.mit.edu/books/machine-learning-0). 2012.
- **[PRML]** Bishop, Christopher. *Pattern Recognition and Machine Learning* (http://research.microsoft.com/en-us/um/people/cmbishop/prml/). 2006.
- **[Koller & Friedman 2009]** Koller, Daphne and Nir Friedman. *Probabilistic Graphical Models* (https://mitpress.mit.edu/books/probabilistic-graphical-models). 2009.
- **Book Chapter**: The language of directed acyclic graphical models (http://idiom.ucsd.edu/~rlevy/pmsl_textbook/chapters/pmsl_12.pdf)
  - This notes are really nice

Outline

- Review of Exponential Families
  - MLE
  - Brief discussion of conjugate priors and MAP estimation
- Probabilistic Graphical Models
  - Review of Conditional Indep. Assumptions
  - Intro to Hidden Markov Models
  - Latent Variable Models in general
  - d-separation in Bayesian Networks
- Mixture Models
  - Gaussian Mixture Model
  - Relationship to Clustering

Exponential Family Distributions

DEF: $p(x|\theta)$ has **exponential family form** if:

$$\begin{aligned} p(x|\theta) &= \frac{1}{Z(\theta)} h(x) \exp\left[\eta(\theta)^T \phi(x)\right] \\ &= h(x) \exp\left[\eta(\theta)^T \phi(x) - A(\theta)\right] \end{aligned}$$

- $Z(\theta)$ is the **partition function** for normalization
- $A(\theta) = \log Z(\theta)$ is the **log partition function**
- $\phi(x) \in \mathbb{R}^d$ is a vector of **sufficient statistics**
- $\eta(\theta)$ maps θ to a set of **natural parameters**
- $h(x)$ is a scaling constant, usually $h(x) = 1$

Exponential Family: MLE

To find the maximum, recall $\nabla_\theta A(\theta) = E_\theta[\phi(x)]$, so \begin{align} \nabla\theta \log p(\D | \theta) &= \nabla\theta(\theta^T \phi(\D) - N A(\theta)) \\ &= \phi(\D) - N E_\theta[\phi(X)] = 0 \end{align} Which gives

$$E_\theta[\phi(X)] = \frac{\phi(\mathcal{D})}{N} = \frac{1}{N} \sum_{k=1}^{N} \phi(x_k)$$

Obtaining the maximum likelihood is simply solving the calculus problem $\nabla_\theta A(\theta) = \frac{1}{N} \sum_{k=1}^{N} \phi(x_k)$ which is often easy

Bayes for Exponential Family

Exact Bayesian analysis is considerably simplified if the prior is **conjugate** to the likelihood.

- Simply, this means that prior $p(\theta)$ has the same form as the posterior $p(\theta|\mathcal{D})$.

This requires likelihood to have finite sufficient statistics

- Exponential family to the rescue!

Note: We will release some notes on cojugate priors + exponential families. It's hard to learn from slides and needs a bit more description.

Likelihood for exponential family

Likelihood:

$$p(\mathcal{D}|\theta) \propto g(\theta)^N \exp[\eta(\theta)^T s_N]$$
$$s_N = \sum_{i=1}^N \phi(x_i)$$

In terms of canonical parameters:

$$p(\mathcal{D}|\eta) \propto \exp[N\eta^T \bar{s} - NA(\eta)]$$
$$\bar{s} = \frac{1}{N} s_N$$

Conjugate prior for exponential family

- The prior and posterior for an exponential family involve two parameters, τ and ν, initially set to τ_0, ν_0

$$p(\theta|\nu_0, \tau_0) \propto g(\theta)^{\nu_0} \exp[\eta(\theta)^T \tau_0]$$

- Denote $\tau_0 = \nu_0 \bar{\tau}_0$ to separate out the size of the **prior pseudo-data**, ν_0 , from the mean of the sufficient statistics on this pseudo-data, τ_0 . Hence,

$$p(\theta|\nu_0, \bar{\tau}_0) \propto \exp[\nu_0 \eta(\theta)^T \bar{\tau}_0 - \nu_0 A(\eta)]$$

- Think of τ_0 as a "guess" of the future sufficient statistics, and ν_0 as the strength of this guess

Prior: Example

$$p(\theta|\nu_0,\tau_0) \propto (1-\theta)^{\nu_0} \exp[\tau_0 \log(\frac{\theta}{1-\theta})]$$
$$= \theta^{\tau_0}(1-\theta)^{\nu_0-\tau_0}$$

Define $\alpha = \tau_0 + 1$ and $\beta = \nu_0 - \tau_0 + 1$ to see that this is a **beta distribution**.

Posterior

Posterior:

$$p(\theta|\mathcal{D}) = p(\theta|\nu_N,\tau_N) = p(\theta|\nu_0+N, \tau_0+s_N)$$

Note that we obtain **hyper-parameters** by adding. Hence,

$$p(\eta|\mathcal{D}) \propto \exp[\eta^T(\nu_0\bar{\tau}_0 + N\bar{s}) - (\nu_0+N)A(\eta)]$$
$$= p(\eta|\nu_0+N, \frac{\nu_0\bar{\tau}_0+N\bar{s}}{\nu_0+N})$$

where $\bar{s} = \frac{1}{N}\sum_{i=1}^N \phi(x_i)$.

- *posterior hyper-parameters are a convex combination of the prior mean hyper-parameters and the average of the sufficient statistics.*

Back to Graphical Models!

Recall: Bayesian Networks: Definition

A **Bayesian Network** $\mathcal{G}$ is a directed acyclic graph whose nodes represent random variables $X_1, \ldots, X_n$.

- Let $\mathbf{Parents}_{\mathcal{G}}(X_k)$ denote the parents of X_k in $\mathcal{G}$
- Let $\mathbf{NonDesc}_{\mathcal{G}}(X_k)$ denote the variables in $\mathcal{G}$ who are not descendants of X_k.

> Examples will come shortly...

Bayesian Networks: Local Independencies

Every Bayesian Network $\mathcal{G}$ encodes a set $\mathcal{I}_\ell(\mathcal{G})$ of **local independence assumptions**:

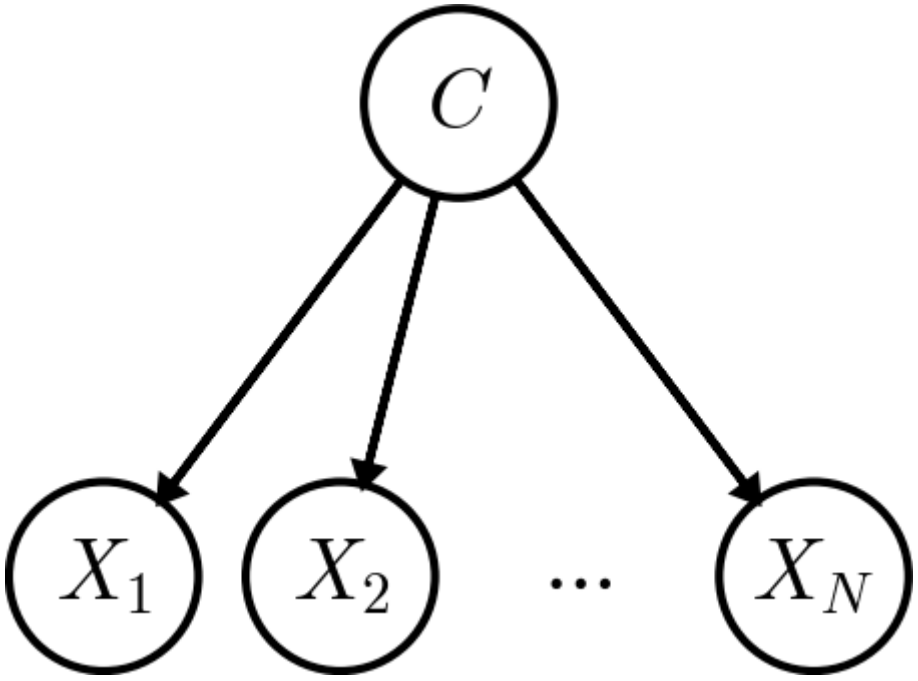
> For each variable X_k, we have $(X_k \perp \mathbf{NonDesc}_\mathcal{G}(X_k) \mid \mathbf{Parents}_\mathcal{G}(X_k))$

Every node X_k is conditionally independent of its nondescendants given its parents.

Example: Naive Bayes

The graphical model for Naive Bayes is shown below:

- $\mathbf{Parents}_\mathcal{G}(X_k) = \{C\}$, $\mathbf{NonDesc}_\mathcal{G}(X_k) = \{X_j\}_{j \neq k} \cup \{C\}$
- Therefore $X_j \perp X_k \mid C$ for any $j \neq k$



Factorization Theorem: Statement

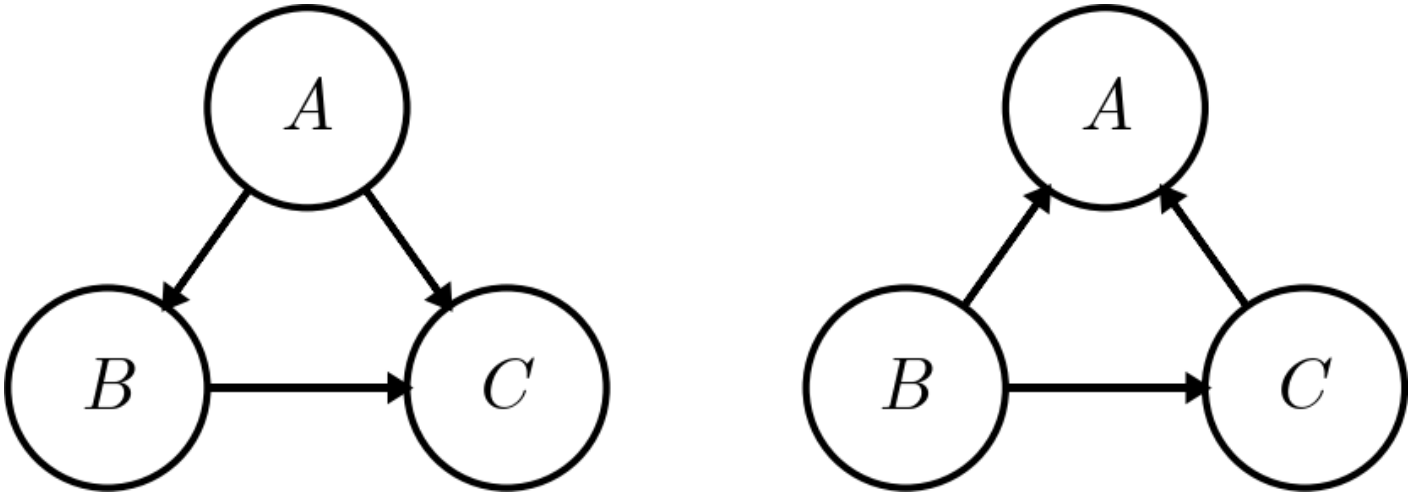
Theorem: *(Koller & Friedman 3.1)* If $\mathcal{G}$ is an I-map for P, then P **factorizes** as follows:

$$P(X_1, \ldots, X_N) = \prod_{k=1}^{N} P(X_k \mid \mathbf{Parents}_\mathcal{G}(X_k))$$

Example: Fully Connected Graph

A **fully connected graph** makes no independence assumptions.

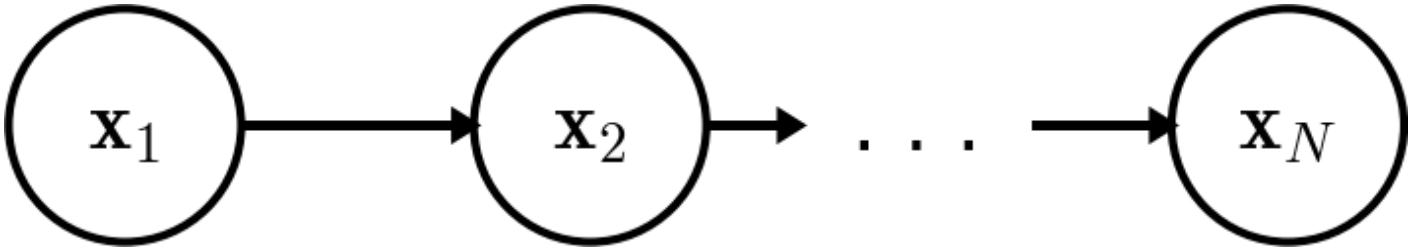
$$P(A, B, C) = P(A)P(B|A)P(C|A, B)$$



Importan PGM Example: Markov Chain

State at time t depends only on state at time $t-1$.

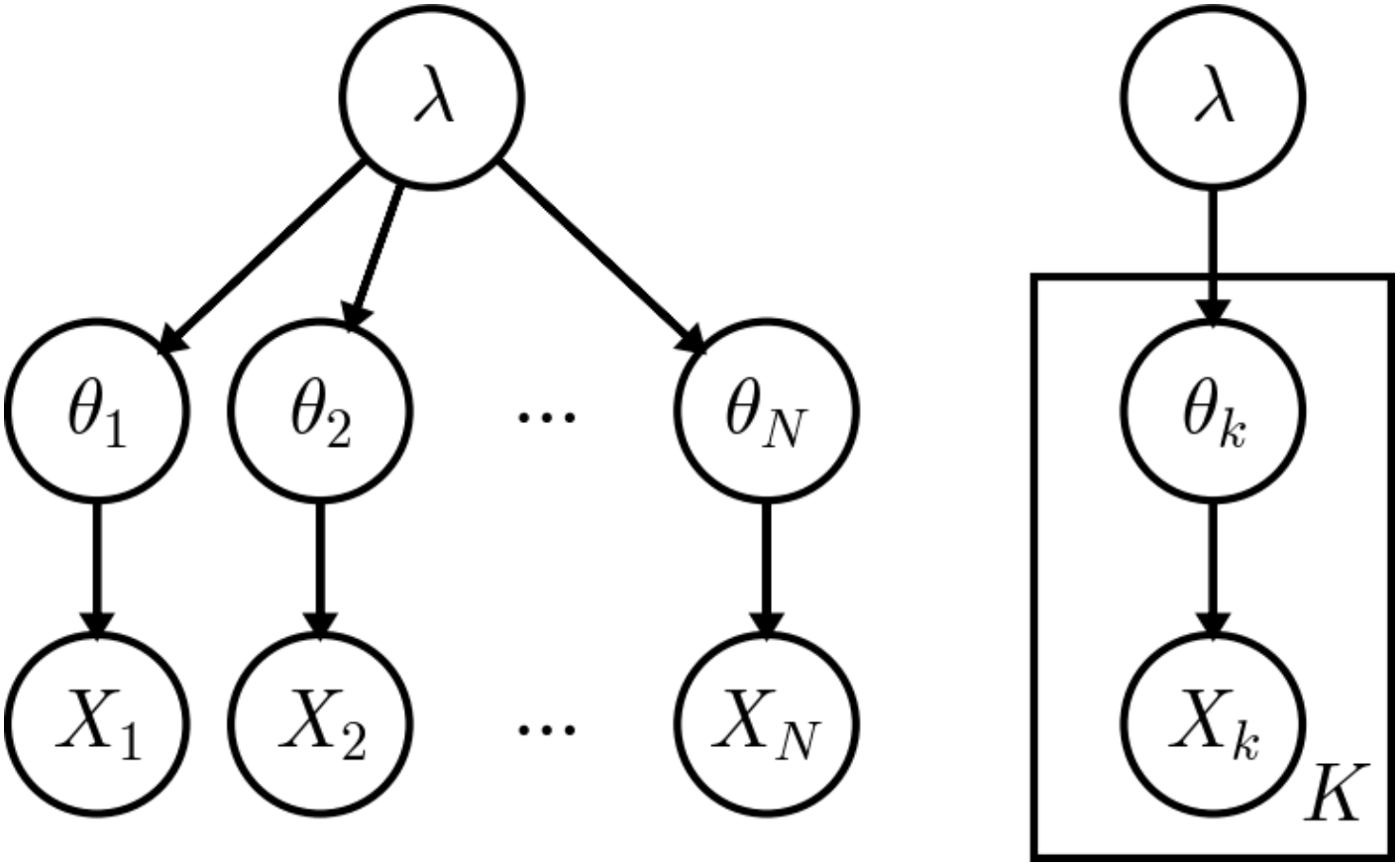
$$P(X_0, X_1, \ldots, X_N) = P(X_0) \prod_{t=1}^{N} P(X_t \mid X_{t-1})$$



Compact Representation of PGM: Plate Notation

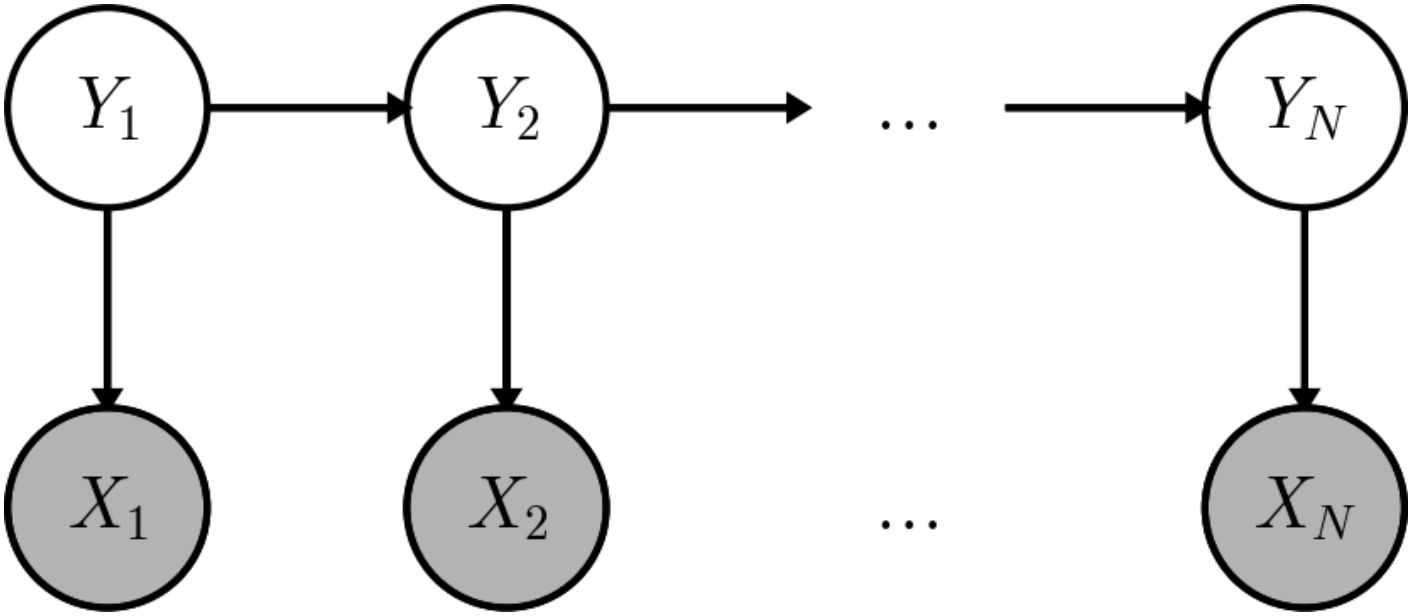
We can represent (conditionally) iid variables using **plate notation**.

A box around a variable (or set of variables), with a K, means that we have access to K iid samples of this variable.



Unobserved Variables: Hidden Markov Model

Noisy observations X_k generated from hidden Markov chain Y_k. (More on this soon)



This Brings us to: Latent Variable Models

> Uses material from **[MLAPP]** §10.1-10.4, §11.1-11.2

Latent Variable Models

In general, the goal of probabilistic modeling is to

> Use what we know to make *inferences* about what we don't know.

Graphical models provide a natural framework for this problem.

- Assume unobserved variables are correlated due to the influence of unobserved **latent variables**.
- Latent variables encode beliefs about the generative process.

In a graphical model, we will often **shade in** the observed variables to distinguish them from hidden variables.

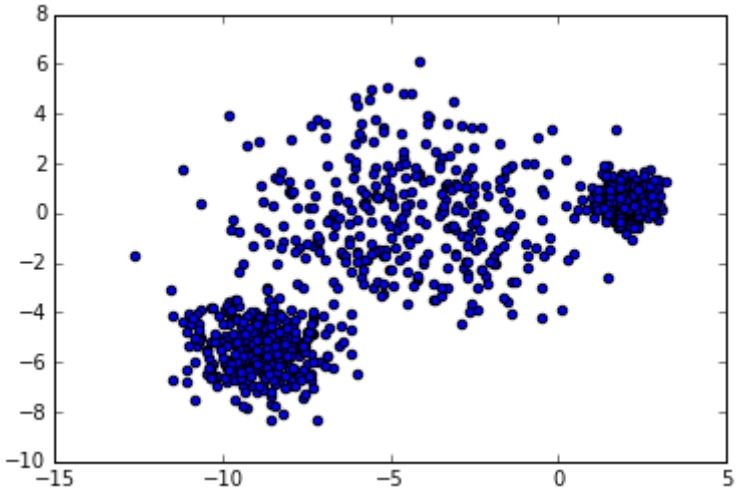
Example: Gaussian Mixture Models

This dataset is hard to explain with a single distribution.

- Underlying density is complicated overall...
- But it's clearly three Gaussians!

In [2]:
```
X, y = skl.datasets.make_blobs(1000, cluster_std=[1.0, 2.5, 0.5], random
_state=170)
plt.scatter(X[:,0], X[:,1])
```

Out[2]: <matplotlib.collections.PathCollection at 0x7ff59a84b908>



Example: Mixture Models

Instead, introduce a latent **cluster label** $z_j \in [K]$ for each datapoint x_j,

$$\begin{aligned} z_j &\sim \text{Cat}(\pi_1, \ldots, \pi_K) && \forall\, j = 1, \ldots, N \\ x_j \mid z_j &\sim \mathcal{N}(\mu_{z_j}, \Sigma_{z_j}) && \forall\, j = 1, \ldots, N \end{aligned}$$

This allows us to explain a complicated density as a **mixture** of simpler densities:

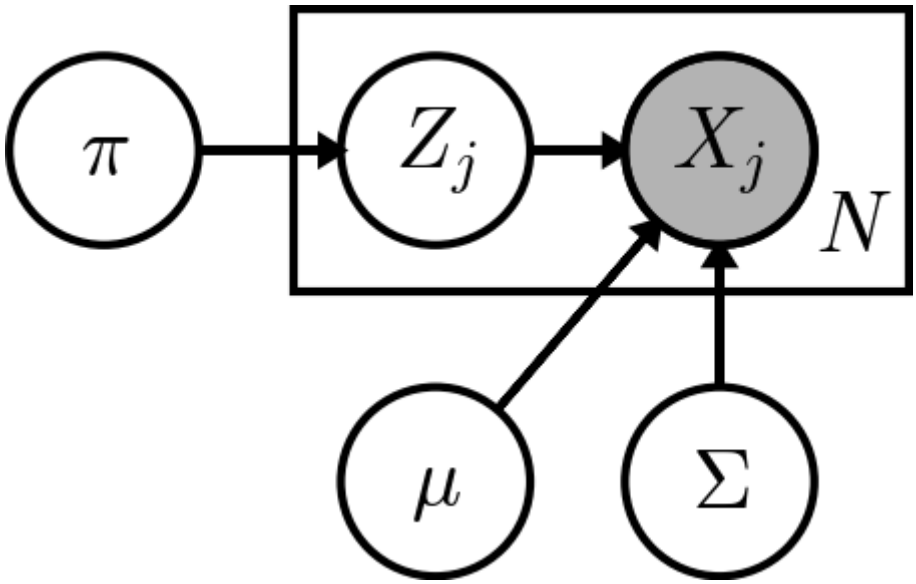
$$P(x|\mu, \Sigma) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

Example: Mixture Models

In [3]:
```
@pgm_render
def pgm_gmm():
    pgm = daft.PGM([4,4], origin=[-2,-1], node_unit=0.8, grid_unit=2.0);
    # nodes
    pgm.add_node(daft.Node("pi", r"$\pi$", 0, 1));
    pgm.add_node(daft.Node("z", r"$Z_j$", 0.7, 1));
    pgm.add_node(daft.Node("x", r"$X_j$", 1.3, 1, observed=True));
    pgm.add_node(daft.Node("mu", r"$\mu$", 0.7, 0.3));
    pgm.add_node(daft.Node("sigma", r"$\Sigma$", 1.3, 0.3));

    # edges
    pgm.add_edge("pi", "z", head_length=0.08);
    pgm.add_edge("z", "x", head_length=0.08);
    pgm.add_edge("mu", "x", head_length=0.08);
    pgm.add_edge("sigma", "x", head_length=0.08);

    pgm.add_plate(daft.Plate([0.4,0.8,1.3,0.5], label=r"$\qquad\qquad\qq
uad\;\; N$",
    shift=-0.1))

    return pgm;
```

In [4]:
```
%%capture
pgm_gmm("images/pgm/pgm-gmm.png")
```

Out[4]:



Example: Hidden Markov Models

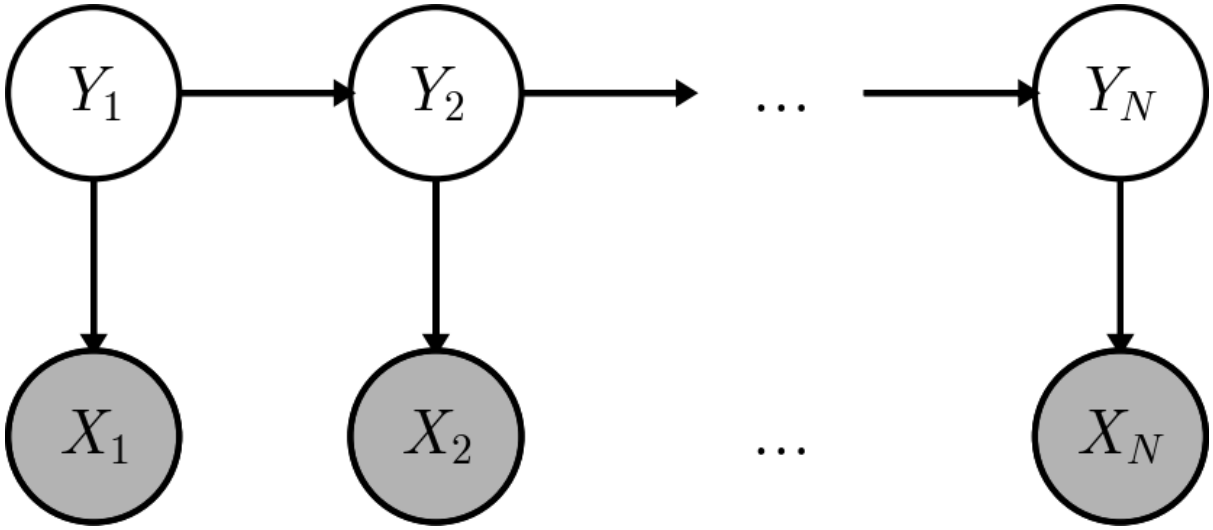
Noisy observations X_k generated from hidden Markov chain Y_k.

$$P(\mathbf{X}, \mathbf{Y}) = P(Y_1)P(X_1 \mid Y_1)\prod_{k=2}^{N}(P(Y_k \mid Y_{k-1})P(X_k \mid Y_k))$$

In [5]:
```
@pgm_render
def pgm_hmm():
    pgm = daft.PGM([7, 7], origin=[0, 0])

    # Nodes
    pgm.add_node(daft.Node("Y1", r"$Y_1$", 1, 3.5))
    pgm.add_node(daft.Node("Y2", r"$Y_2$", 2, 3.5))
    pgm.add_node(daft.Node("Y3", r"$\dots$", 3, 3.5, plot_params={'ec':'n

    pgm.add_node(daft.Node("Y4", r"$Y_N$", 4, 3.5))

    pgm.add_node(daft.Node("x1", r"$X_1$", 1, 2.5, observed=True))
    pgm.add_node(daft.Node("x2", r"$X_2$", 2, 2.5, observed=True))
    pgm.add_node(daft.Node("x3", r"$\dots$", 3, 2.5, plot_params={'ec':'n

    pgm.add_node(daft.Node("x4", r"$X_N$", 4, 2.5, observed=True))


    # Add in the edges.
    pgm.add_edge("Y1", "Y2", head_length=0.08)
    pgm.add_edge("Y2", "Y3", head_length=0.08)
    pgm.add_edge("Y3", "Y4", head_length=0.08)

    pgm.add_edge("Y1", "x1", head_length=0.08)
    pgm.add_edge("Y2", "x2", head_length=0.08)
    pgm.add_edge("Y4", "x4", head_length=0.08)

    return pgm;
```

In [6]:
```
%%capture
pgm_hmm("images/pgm/hmm.png");
```

Out[6]:



Example: Unsupervised Learning

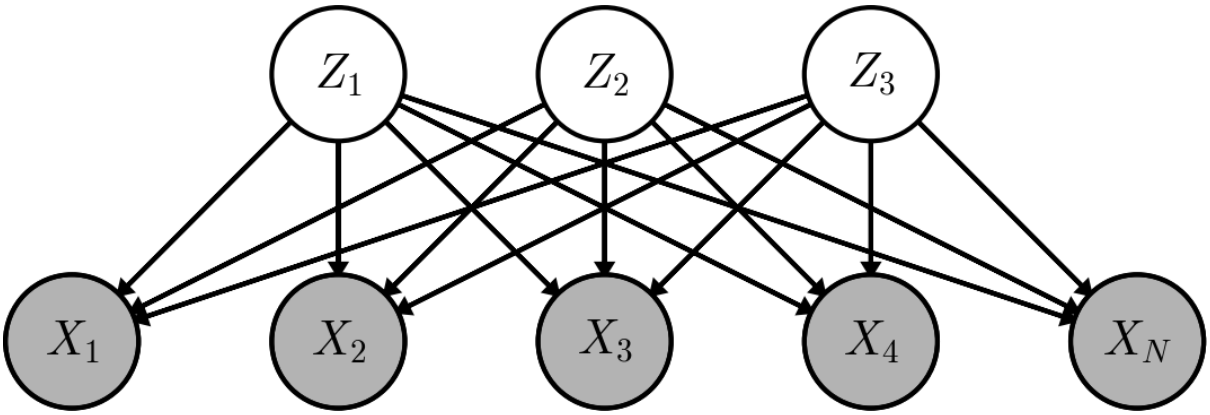
Latent variables are fundamental to unsupervised and deep learning.

- Serve as a **bottleneck**
- Compute a **compressed** representation of data

In [7]:
```
@pgm_render
def pgm_unsupervised():
    pgm = daft.PGM([6, 6], origin=[0, 0])

    # Nodes
    pgm.add_node(daft.Node("d1", r"$Z_1$", 2, 3.5))
    pgm.add_node(daft.Node("di", r"$Z_2$", 3, 3.5))
    pgm.add_node(daft.Node("dn", r"$Z_3$", 4, 3.5))

    pgm.add_node(daft.Node("f1", r"$X_1$", 1, 2.50, observed=True))
    pgm.add_node(daft.Node("fi-1", r"$X_2$", 2, 2.5, observed=True))
    pgm.add_node(daft.Node("fi", r"$X_3$", 3, 2.5, observed=True))
    pgm.add_node(daft.Node("fi+1", r"$X_4$", 4, 2.5, observed=True))
    pgm.add_node(daft.Node("fm", r"$X_N$", 5, 2.5, observed=True))


    # Add in the edges.
    pgm.add_edge("d1", "f1", head_length=0.08)
    pgm.add_edge("d1", "fi-1", head_length=0.08)
    pgm.add_edge("d1", "fi", head_length=0.08)
    pgm.add_edge("d1", "fi+1", head_length=0.08)
    pgm.add_edge("d1", "fm", head_length=0.08)

    pgm.add_edge("di", "f1", head_length=0.08)
    pgm.add_edge("di", "fi-1", head_length=0.08)
    pgm.add_edge("di", "fi", head_length=0.08)
    pgm.add_edge("di", "fi+1", head_length=0.08)
    pgm.add_edge("di", "fm", head_length=0.08)

    pgm.add_edge("dn", "f1", head_length=0.08)
    pgm.add_edge("dn", "fi-1", head_length=0.08)
    pgm.add_edge("dn", "fi", head_length=0.08)
    pgm.add_edge("dn", "fi+1", head_length=0.08)
    pgm.add_edge("dn", "fm", head_length=0.08)

    return pgm
```

In [8]:
```
%%capture
pgm_unsupervised("images/pgm/unsupervised.png");
```

Out[8]:



Other Latent Variable Models

Many other models in machine learning involve latent variables:

- Neural Networks / Multilayer Perceptrons
- Restricted Boltzmann Machines
- Deep Belief Networks
- Probabilistic PCA

Latent Variable Models: Complexity

Latent variable models exhibit **emergent complexity**.

- Although each conditional distribution is simple,
- The joint distribution is capable of modeling complex interactions.

However, latent variables make learning difficult.

- Inference is challening in models with latent variables.
- They can introduce new dependencies between observed variables.

Break time

Bayesian Networks

Part II: Inference, Learning, and d-Separation

> Uses material from **[Koller & Friedman 2009]** Chapter 3, **[MLAPP]** Chapter 10, and **[PRML]** §8.2.1

Bayesian Networks: Terminology

Typically, our models will have

- Observed variables X
- Hidden variables Z
- Parameters θ

Occasionally, we will distinguish between **inference** and **learning**.

Bayesian Networks: Inference

Inference: Estimate hidden variables Z from observed variables X.

$$P(Z|X,\theta) = \frac{P(X,Z|\theta)}{P(X|\theta)}$$

- Denominator $P(X|\theta)$ is sometimes called the probability of the **evidence**.
- Occasionally we care only about a subset of the hidden variables, and marginalize out the rest.

Bayesian Networks: Learning

Learning: Estimate parameters θ from observed data X.

$$P(\theta \mid X) = \sum_{z \in Z} P(\theta, z \mid X) = \sum_{z \in Z} P(\theta \mid z, X) P(z \mid X)$$

To Bayesians, parameters *are* hidden variables, so inference and learning are equivalent.

Bayesian Networks: Probability Queries

In general, it is useful to compute $P(A|B)$ for arbitrary collections A and B of variables.

- Both inference and learning take this form.

To accomplish this, we must understand the **independence structure** of any given graphical model.

Review: Local Independencies

Every Bayesian Network $\mathcal{G}$ encodes a set $\mathcal{I}_\ell(\mathcal{G})$ of **local independence assumptions**:

> For each variable X_k, we have $(X_k \perp \mathbf{NonDesc}_\mathcal{G}(X_k) \mid \mathbf{Parents}_\mathcal{G}(X_k))$

Every node X_k is conditionally independent of its nondescendants given its parents.

> For arbitrary sets of variables, when does $(A \perp B \mid C)$ hold?

Review: I-Maps

If P satisfies the independence assertions made by $\mathcal{G}$, we say that

- $\mathcal{G}$ is an **I-Map** for P
- or that P **satisfies** $\mathcal{G}$.

Any distribution satisfying $\mathcal{G}$ shares common structure.

- We will exploit this structure in our algorithms
- This is what makes graphical models so **powerful**!

Review: Factorization Theorem

Last time, we proved that for any P satisfying $\mathcal{G}$,

$$P(X_1, \ldots, X_N) = \prod_{k=1}^{N} P(X_k \mid \text{Parents}_{\mathcal{G}}(X_k))$$

If we understand independence structure, we can factorize arbitrary conditional distributions:

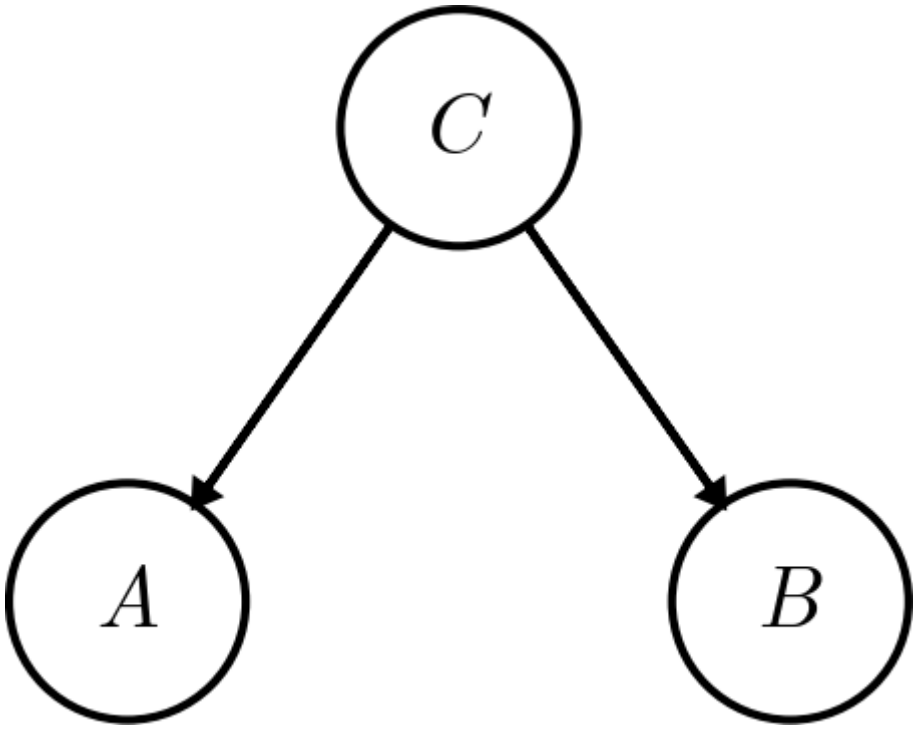
$$P(A_1, \ldots, A_n \mid B_1, \ldots, B_m) = ?$$

Question 1: Is $(A \perp B)$?

In [9]:
```
@pgm_render
def pgm_question1():
    pgm = daft.PGM([4, 4], origin=[0, 0])

    # Nodes
    pgm.add_node(daft.Node("c", r"$C$", 2, 3.5))
    pgm.add_node(daft.Node("a", r"$A$", 1.3, 2.5))
    pgm.add_node(daft.Node("b", r"$B$", 2.7, 2.5))

    # Add in the edges.
    pgm.add_edge("c", "a", head_length=0.08)
    pgm.add_edge("c", "b", head_length=0.08)

    return pgm;
```

In [10]:
```
%%capture
pgm_question1("images/pgm/question1.png")
```

Out[10]:



Answer 1: No!

No! A and B are not marginally independent.

- Note C is not shaded, so we don't observe it.

In general,

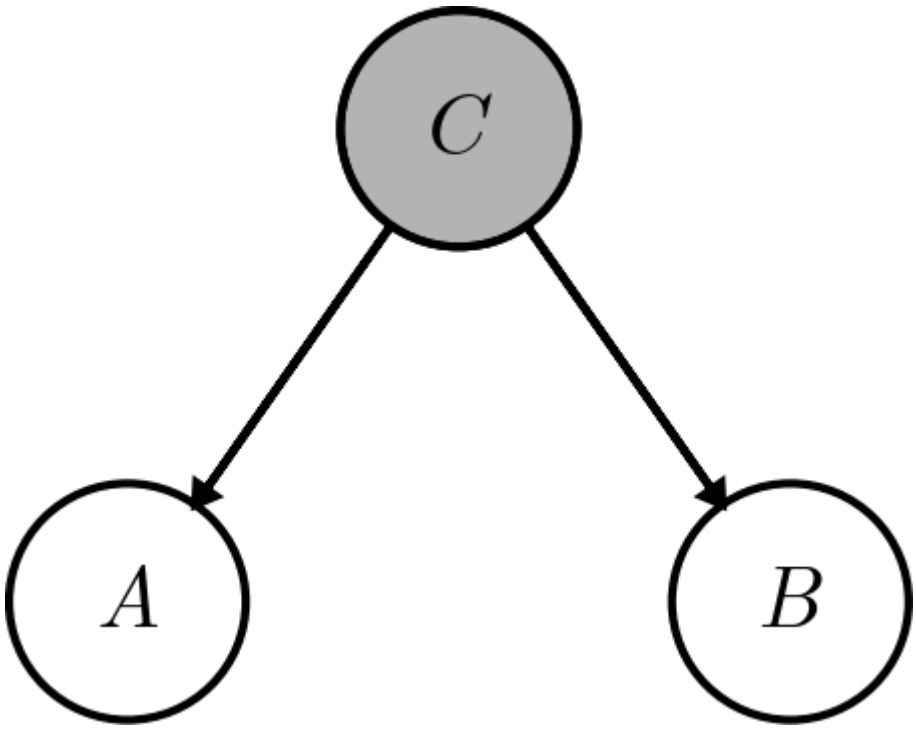
$$P(A,B) = \sum_{c \in C} P(A,B,c) = \sum_{c \in C} P(A|c)P(B|c)P(c) \neq P(A)P(B)$$

Question 2: Is $(A \perp B \mid C)$?

In [11]:
```
@pgm_render
def pgm_question2():
    pgm = daft.PGM([4, 4], origin=[0, 0])

    # Nodes
    pgm.add_node(daft.Node("c", r"$C$", 2, 3.5,
                           observed=True))
    pgm.add_node(daft.Node("a", r"$A$", 1.3, 2.5))
    pgm.add_node(daft.Node("b", r"$B$", 2.7, 2.5))

    # Add in the edges.
    pgm.add_edge("c", "a", head_length=0.08)
    pgm.add_edge("c", "b", head_length=0.08)

    return pgm
```

In [12]:
```
%%capture
pgm_question2("images/pgm/question2.png")
```

Out[12]:



Answer 2: Yes!

Yes! $(A \perp B|C)$ follows from the local independence properties of Bayesian networks.

> Every variable is conditionally independent of its nondescendants given its parents.

Observing C *blocks* the path of influence from A to B. Or, using factorization theorem:

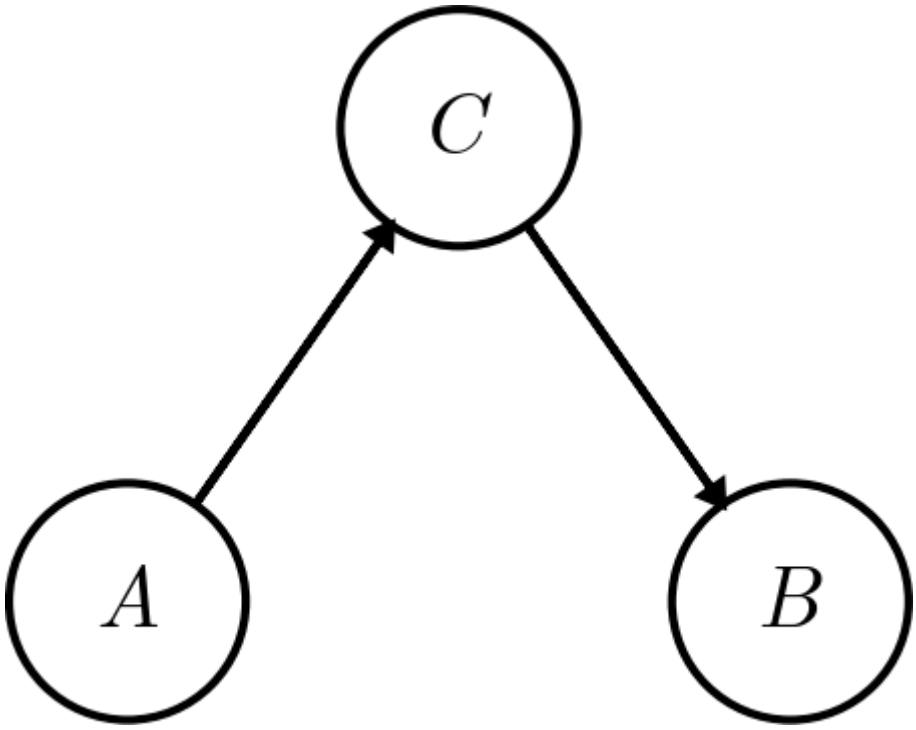
$$\begin{aligned} P(A, B|C) &= \frac{P(A, B, C)}{P(C)} \\ &= \frac{P(C)P(A|C)P(B|C)}{P(C)} \\ &= P(A|C)P(B|C) \end{aligned}$$

Question 3: Is $(A \perp B)$?

In [13]:
```
@pgm_render
def pgm_question3():
    pgm = daft.PGM([4, 4], origin=[0, 0])

    # Nodes
    pgm.add_node(daft.Node("c", r"$C$", 2, 3.5))
    pgm.add_node(daft.Node("a", r"$A$", 1.3, 2.5))
    pgm.add_node(daft.Node("b", r"$B$", 2.7, 2.5))

    # Add in the edges.
    pgm.add_edge("a", "c", head_length=0.08)
    pgm.add_edge("c", "b", head_length=0.08)

    return pgm
```

In [14]:
```
%%capture
pgm_question3("images/pgm/question3.png")
```

Out[14]:



Answer 3: No!

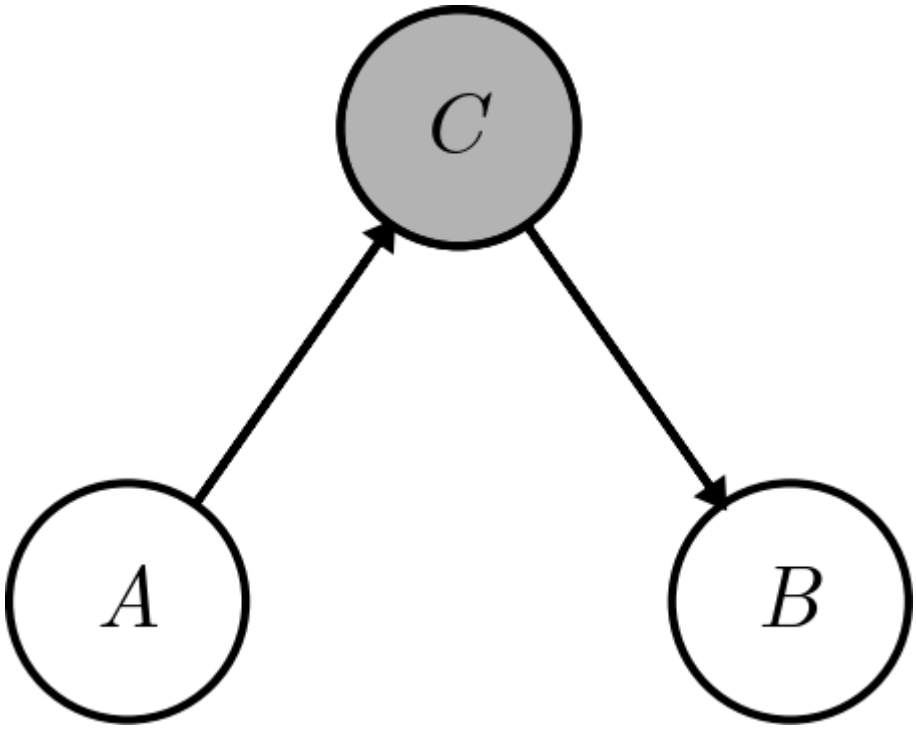
Again, C is not given, so A and B are dependent.

Question 4: Is $(A \perp B \mid C)$?

In [15]:
```
@pgm_render
def pgm_question4():
    pgm = daft.PGM([4, 4], origin=[0, 0])

    # Nodes
    pgm.add_node(daft.Node("c", r"$C$", 2, 3.5, observed=True))
    pgm.add_node(daft.Node("a", r"$A$", 1.3, 2.5))
    pgm.add_node(daft.Node("b", r"$B$", 2.7, 2.5))

    # Add in the edges.
    pgm.add_edge("a", "c", head_length=0.08)
    pgm.add_edge("c", "b", head_length=0.08)

    return pgm
```

In [16]:
```
%%capture
pgm_question4("images/pgm/question4.png")
```

Out[16]:



Answer 4: Yes!

Again, observing C *blocks* influence from A to B.

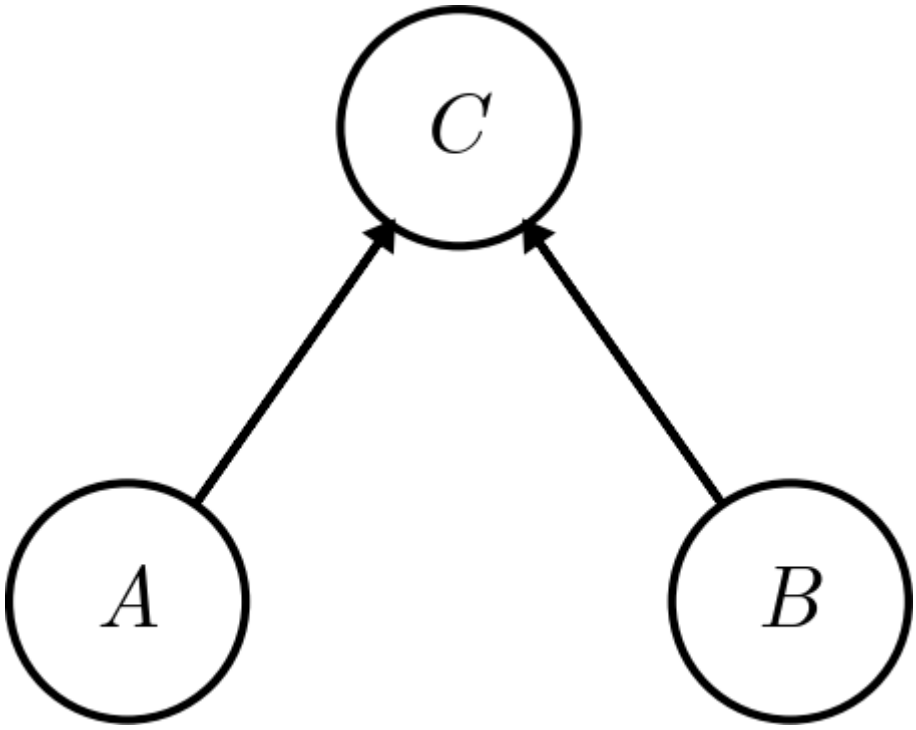
> Every variable is conditionally independent of its nondescendants given its parents.

Question 5: Is $(A \perp B)$?

In [17]:
```
@pgm_render
def pgm_question5():
    pgm = daft.PGM([4, 4], origin=[0, 0])

    # Nodes
    pgm.add_node(daft.Node("c", r"$C$", 2, 3.5))
    pgm.add_node(daft.Node("a", r"$A$", 1.3, 2.5))
    pgm.add_node(daft.Node("b", r"$B$", 2.7, 2.5))

    # Add in the edges.
    pgm.add_edge("a", "c", head_length=0.08)
    pgm.add_edge("b", "c", head_length=0.08)

    return pgm
```

In [18]:
```
%%capture
pgm_question5("images/pgm/question5.png")
```

Out[18]:



Answer 5: Yes!

Using the factorization rule,

$$P(A, B, C) = P(A)P(B)P(C \mid A, B)$$

Therefore, marginalizing out C,

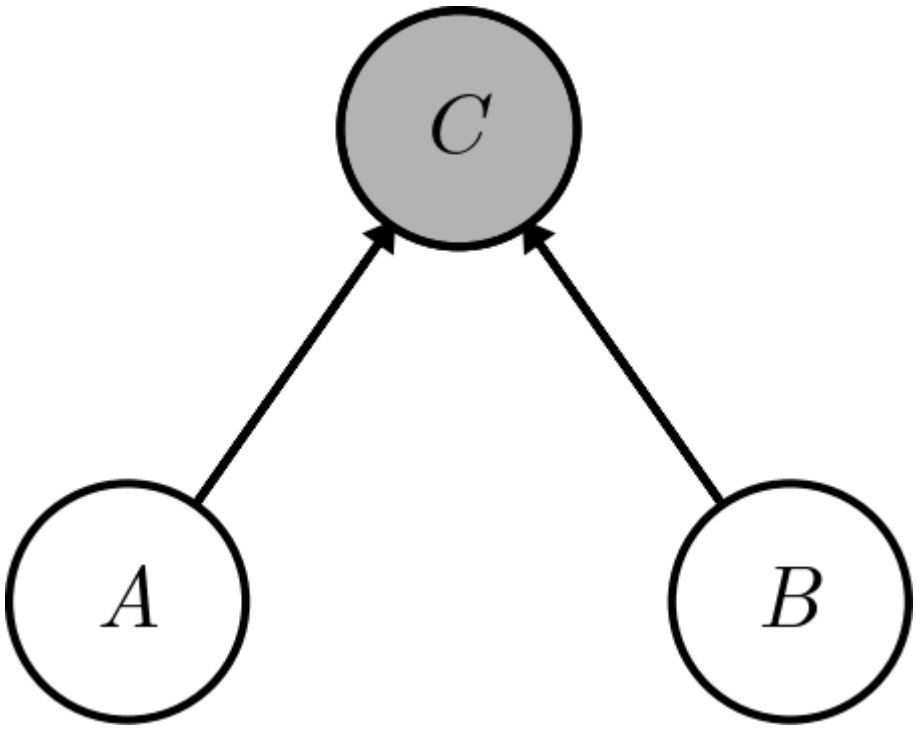
$$\begin{aligned} P(A, B) &= \sum_{c \in C} P(A, B, c) \\ &= \sum_{c \in C} P(A)P(B)P(c \mid A, B) \\ &= P(A)P(B) \sum_{c \in C} P(c \mid A, B) = P(A)P(B) \end{aligned}$$

Question 6: Is $P(A \perp B \mid C)$?

In [19]:
```
@pgm_render
def pgm_question6():
    pgm = daft.PGM([4, 4], origin=[0, 0])

    # Nodes
    pgm.add_node(daft.Node("c", r"$C$", 2, 3.5, observed=True))
    pgm.add_node(daft.Node("a", r"$A$", 1.3, 2.5))
    pgm.add_node(daft.Node("b", r"$B$", 2.7, 2.5))

    # Add in the edges.
    pgm.add_edge("a", "c", head_length=0.08)
    pgm.add_edge("b", "c", head_length=0.08)

    return pgm
```

In [20]:
```
%%capture
pgm_question6("images/pgm/question6.png")
```

Out[20]:



Answer: No!

A can influence B via C.

$$P(A,B|C) = \frac{P(A,B,C)}{P(C)} = \frac{P(A)P(B)P(C|A,B)}{P(C)}$$

This does not factorize in general to $P(A|C)P(B|C)$.

Example: Battery, Fuel, and Gauge

Consider three binary random variables

- Battery B is either charged $(B = 1)$ or dead, $(B = 0)$
- Fuel tank F is either full $(F = 1)$ or empty, $(F = 0)$
- Fuel gauge G either indicates full $(G = 1)$ or empty, $(G = 0)$

Assume $(B \perp F)$ with priors

- $P(B = 1) = 0.9$
- $P(F = 1) = 0.9$

Example: Battery, Fuel, and Gauge

Given the state of the fuel tank and the battery, the fuel gauge reads full with probabilities:

- $p(G = 1 \mid B = 1, F = 1) = 0.8$
- $p(G = 1 \mid B = 1, F = 0) = 0.2$
- $p(G = 1 \mid B = 0, F = 1) = 0.2$
- $p(G = 1 \mid B = 0, F = 0) = 0.1$

Example: Battery, Fuel, and Gauge

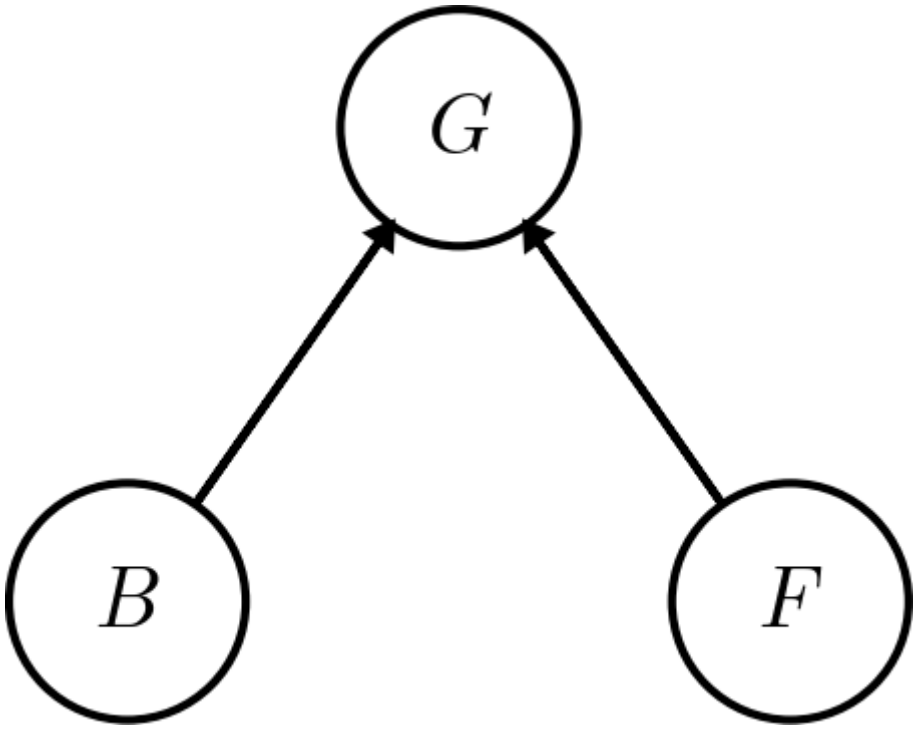
Without any observations, the probability of an empty fuel tank is

$$P(F = 0) = 1 - P(F = 1) = 0.1$$

In [21]:
```
@pgm_render
def pgm_bfg_1():
    pgm = daft.PGM([4, 4], origin=[0, 0])

    # Nodes
    pgm.add_node(daft.Node("G", r"$G$", 2, 3.5))
    pgm.add_node(daft.Node("B", r"$B$", 1.3, 2.5))
    pgm.add_node(daft.Node("F", r"$F$", 2.7, 2.5))

    # Add in the edges.
    pgm.add_edge("B", "G", head_length=0.08)
    pgm.add_edge("F", "G", head_length=0.08)

    return pgm;
```

In [22]:
```
%%capture
pgm_bfg_1("images/pgm/bfg-1.png")
```

Out[22]:



Example: Empty Gauge

Now, suppose the gauge reads $G = 0$. We have

$$P(G = 0) = \sum_{B \in \{0,1\}} \sum_{F \in \{0,1\}} P(G = 0 \mid B, F) P(B) P(F) = 0.315$$

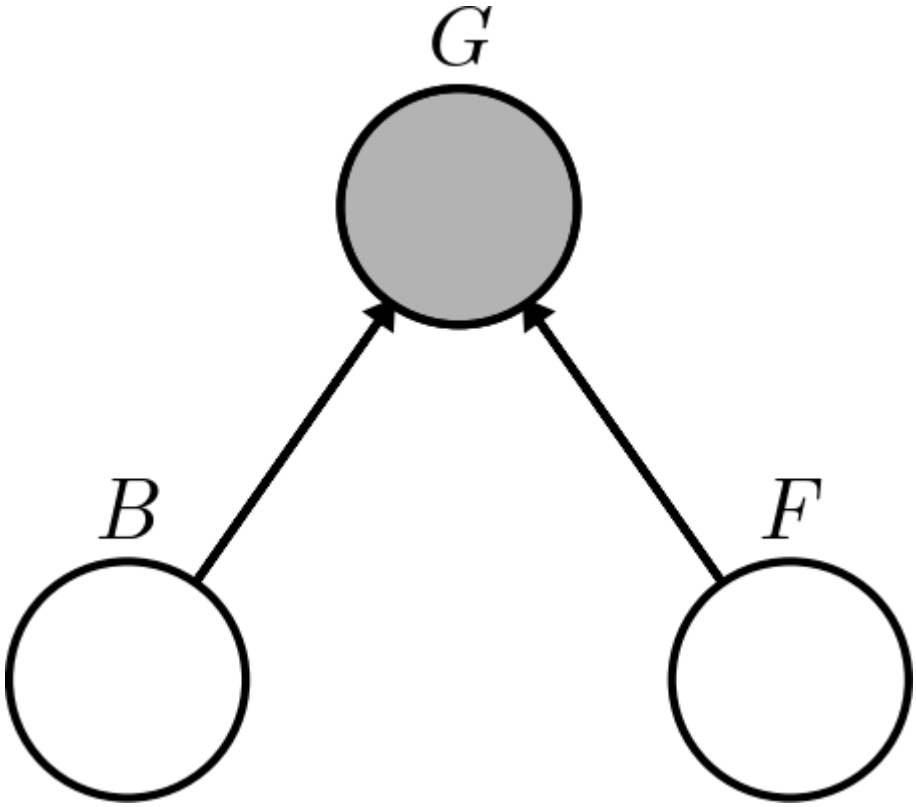
> Verify this!

Example: Emtpy Gauge

In [23]:
```
@pgm_render
def pgm_bfg_2():
    pgm = daft.PGM([4, 4], origin=[0, 0])

    # Nodes
    pgm.add_node(daft.Node("G", r"$G$", 2, 3.5, offset=(0, 20), observed=
e))
    pgm.add_node(daft.Node("B", r"$B$", 1.3, 2.5, offset=(0, 20)))
    pgm.add_node(daft.Node("F", r"$F$", 2.7, 2.5, offset=(0, 20)))

    # Add in the edges.
    pgm.add_edge("B", "G", head_length=0.08)
    pgm.add_edge("F", "G", head_length=0.08)

    return pgm;
```

In [24]:
```
%%capture
pgm_bfg_2("images/pgm/bfg-2.png");
```

Out[24]:



Example: Empty Gauge

Now, we also have

$$p(G=0 \mid F=0) = \sum_{B \in \{0,1\}} p(G=0 \mid B, F=0) p(B) = 0.81$$

Applying Bayes' Rule,

$$\begin{aligned} p(F=0 \mid G=0) &= \frac{p(G=0 \mid F=0)p(F=0)}{p(G=0)} \\ &\approx 0.257 > p(F=0) = 0.10 \end{aligned}$$

Observing an empty gauge makes it more likely that the tank is empty!

Example: Emtpy Gauge, Dead Battery

Now, suppose we *also* observe a dead battery $B=0$. Then,

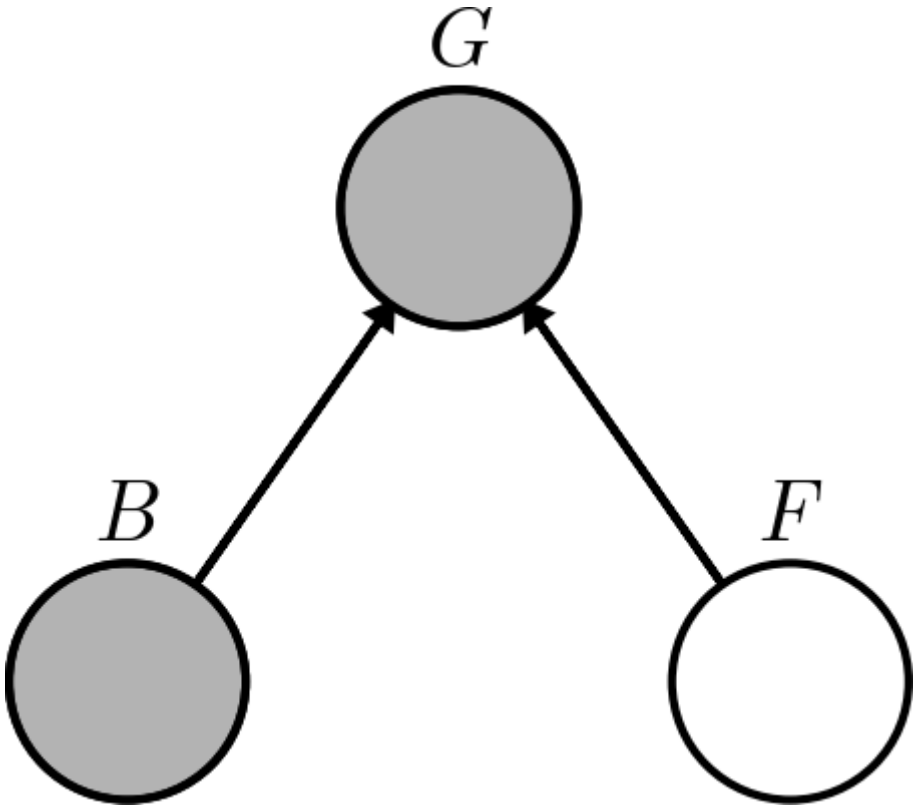
$$\begin{aligned} p(F=0 \mid G=0, B=0) &= \frac{p(G=0 \mid B=0, F=0)p(F=0)}{\sum_{F \in \{0,1\}} p(G=0 \mid B=0, F)p(F)} \\ &\approx 0.111 \end{aligned}$$

Example: Emtpy Gauge, Dead Battery

In [25]:
```
@pgm_render
def pgm_bfg_3():
    pgm = daft.PGM([4, 4], origin=[0, 0])

    # Nodes
    pgm.add_node(daft.Node("G", r"$G$", 2, 3.5, offset=(0, 20), observed=
e))
    pgm.add_node(daft.Node("B", r"$B$", 1.3, 2.5, offset=(0, 20), observ
ed=True))
    pgm.add_node(daft.Node("F", r"$F$", 2.7, 2.5, offset=(0, 20)))

    # Add in the edges.
    pgm.add_edge("B", "G", head_length=0.08)
    pgm.add_edge("F", "G", head_length=0.08)

    return pgm;
```

In [26]:
```
%%capture
pgm_bfg_3("images/pgm/bfg-3.png")
```

Out[26]:



Example: Empty Gauge, Dead Battery

This is the **explaining away** phenomenon.

- The probability of an empty tank has decreased from 0.257 to 0.111 after observing the dead battery!
- A dead battery *explans* why the guage indicates an empty tank, reducing the probability that the tank is *really* empty.

Conditioning on a common child makes its parents dependent.

Bayesian Networks: d-Separation

We say a trail $X \leftrightharpoons Z \leftrightharpoons Y$ is **active** when influence can flow from X to Y via Z.

- **Causal Trail:** $X \rightarrow Z \rightarrow Y$ is active iff Z is *hidden* (unobserved).
- **Evidential Trail:** $X \leftarrow Z \leftarrow Y$ is active iff Z is *hidden*.
- **Common Cause:** $X \leftarrow Z \rightarrow Y$ is active iff Z is *hidden*.
- **Common Effect:** $X \rightarrow Z \leftarrow Y$ is active iff Z or a descendant of Z is *observed*.

Bayesian Networks: d-Separation

A longer trail $X_1 \leftrightharpoons X_2 \leftrightharpoons \cdots \leftrightharpoons X_n$ is **active** if influence can flow from X_1 to X_n along the trail.

- Requires that every $X_{k-1} \leftrightharpoons X_k \leftrightharpoons X_{k+1}$ is active.

In general, $X_1 \leftrightharpoons X_2 \leftrightharpoons \cdots \leftrightharpoons X_n$ is **active** given observed variables $\mathcal{Y}$ if

- Whenever we have a v-structure $X_{k-1} \rightarrow X_k \leftarrow X_{k+1}$ then X_k or one of its descendants is in $\mathcal{Y}$.
- No other node along the trail is in $\mathcal{Y}$.

Bayesian Networks: d-Separation

Some graphs may have more than one trail between two given nodes.

- One node may inflence another if there exists an active trail between them.

Let $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ be sets of nodes in $\mathcal{G}$. We say $\mathbf{X}$ and $\mathbf{Y}$ are **d-separated** given $\mathbf{Z}$, denoted $\text{d-sep}_{\mathcal{G}}(\mathbf{X}; \mathbf{Y} \mid \mathbf{Z})$ if there is no active trail between any nodes $X \in \mathbf{X}$ and $Y \in \mathbf{Y}$ given $\mathbf{Z}$.

Bayesian Networks: d-Separation

Theorem: (Koller & Friedman 3.5) For almost all distributions P that factorize over $\mathcal{G}$, we have

$$\text{d-sep}_{\mathcal{G}}(X; Y \mid Z) \iff (X \perp Y \mid Z)$$

That is, except on a set of measure zero in the space of distributions, the concept of d-separation in $\mathcal{G}$ exactly captures the concept of conditional independence.

- To check if variables are independent, check for d-separation.

Challenging Independence Questions:

- Why is the claim **False**? It actually isn't true that $B \perp E \mid D$!

Notice that there is an active path from B to E:

$$B \rightarrow \boxed{D} \leftarrow A \rightarrow C \rightarrow E$$