Sampling for General Inference

Chris Piech
CS109, Stanford University

Midterm

Midterm (part 1)

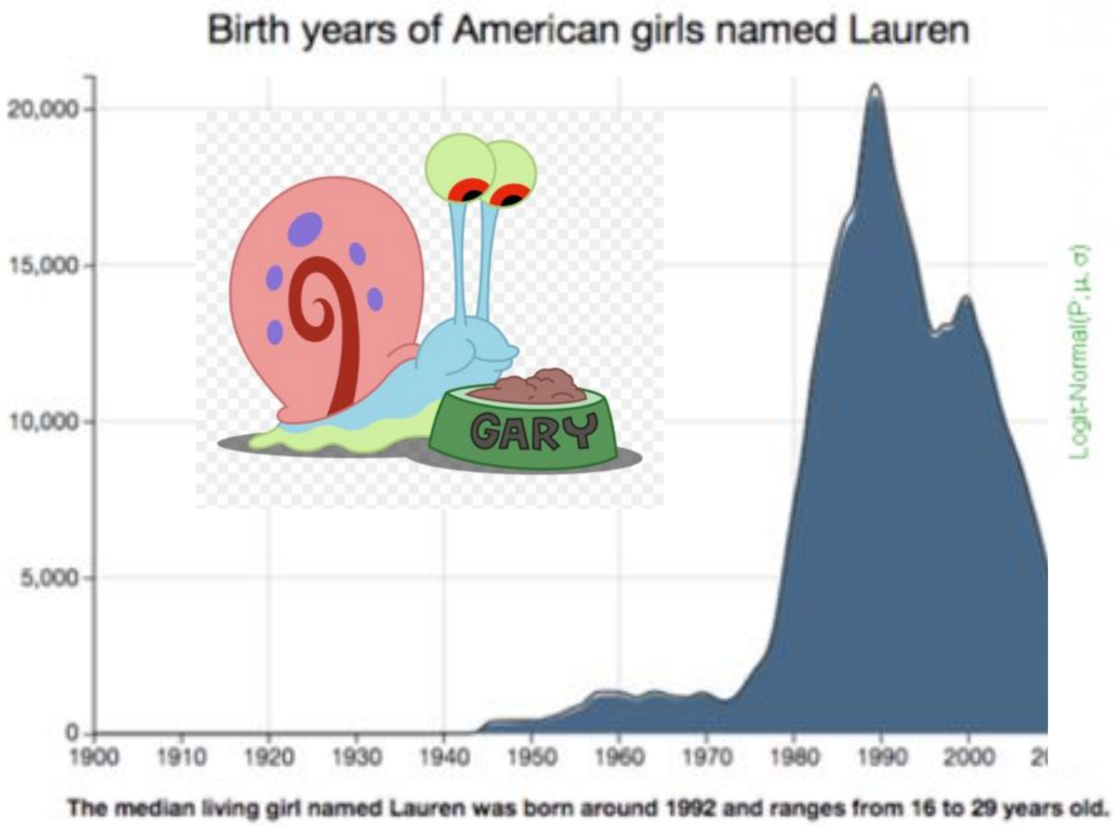
1

2

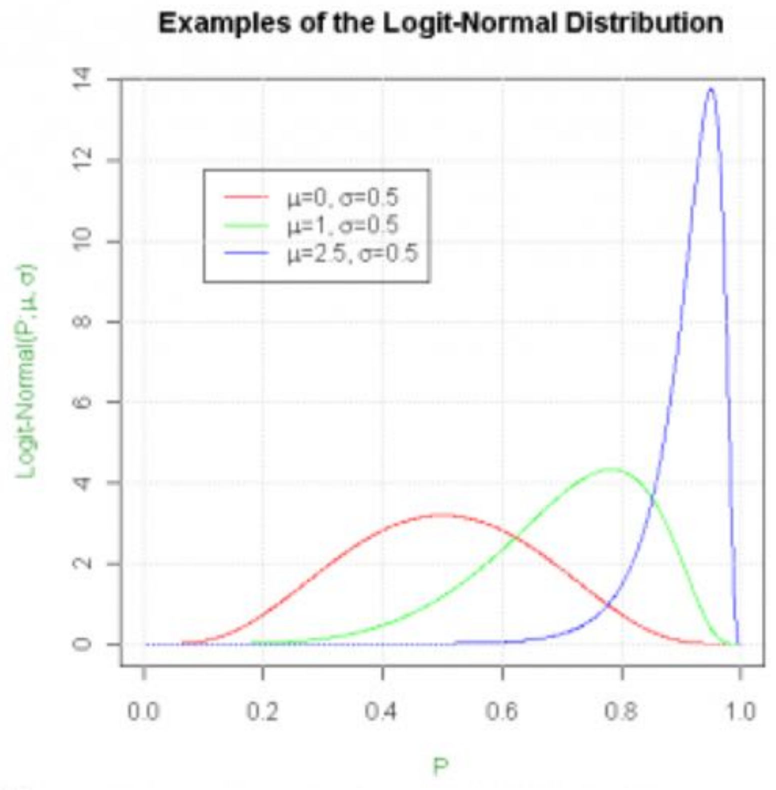
3

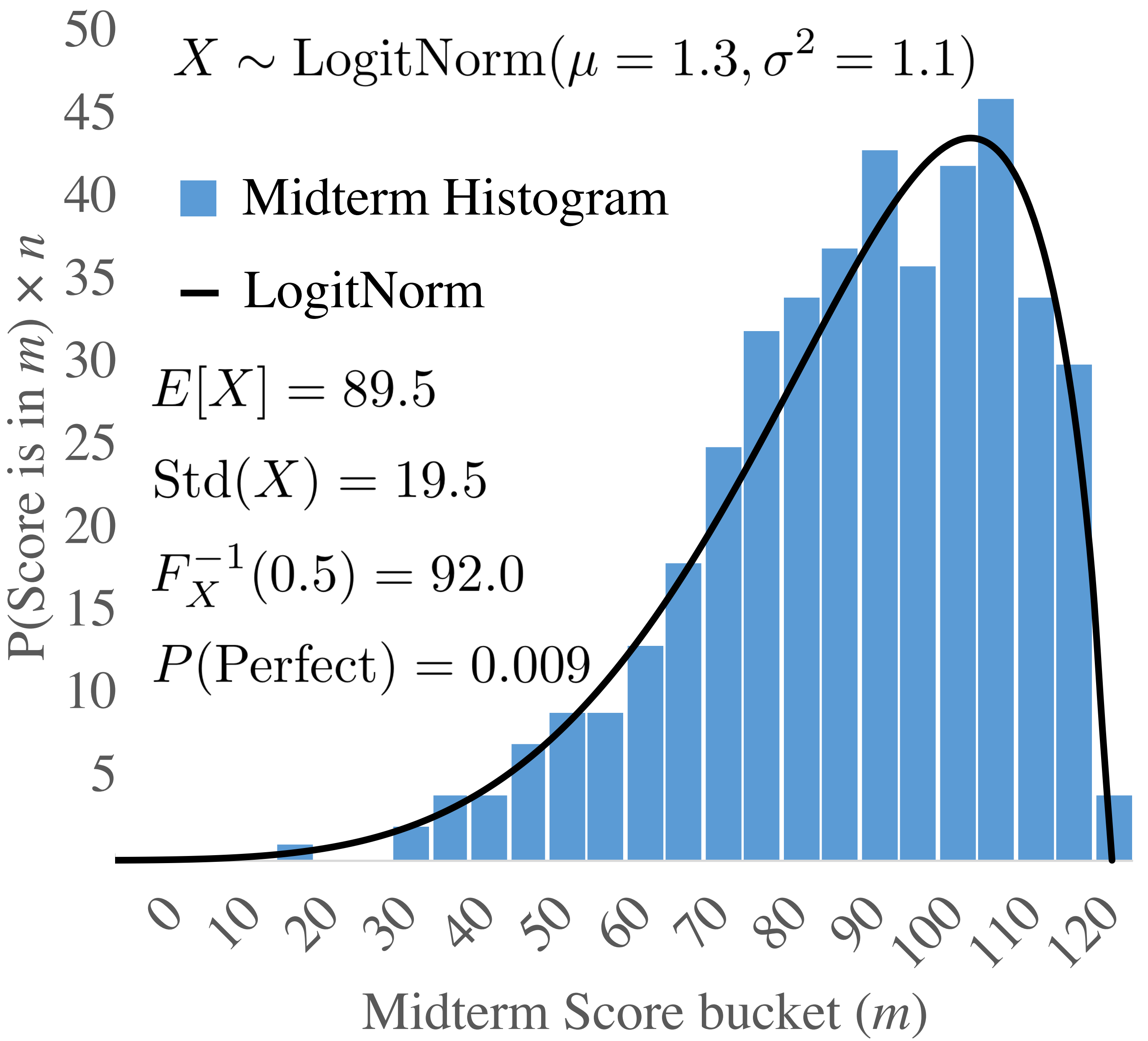
Midterm (part 2)

4



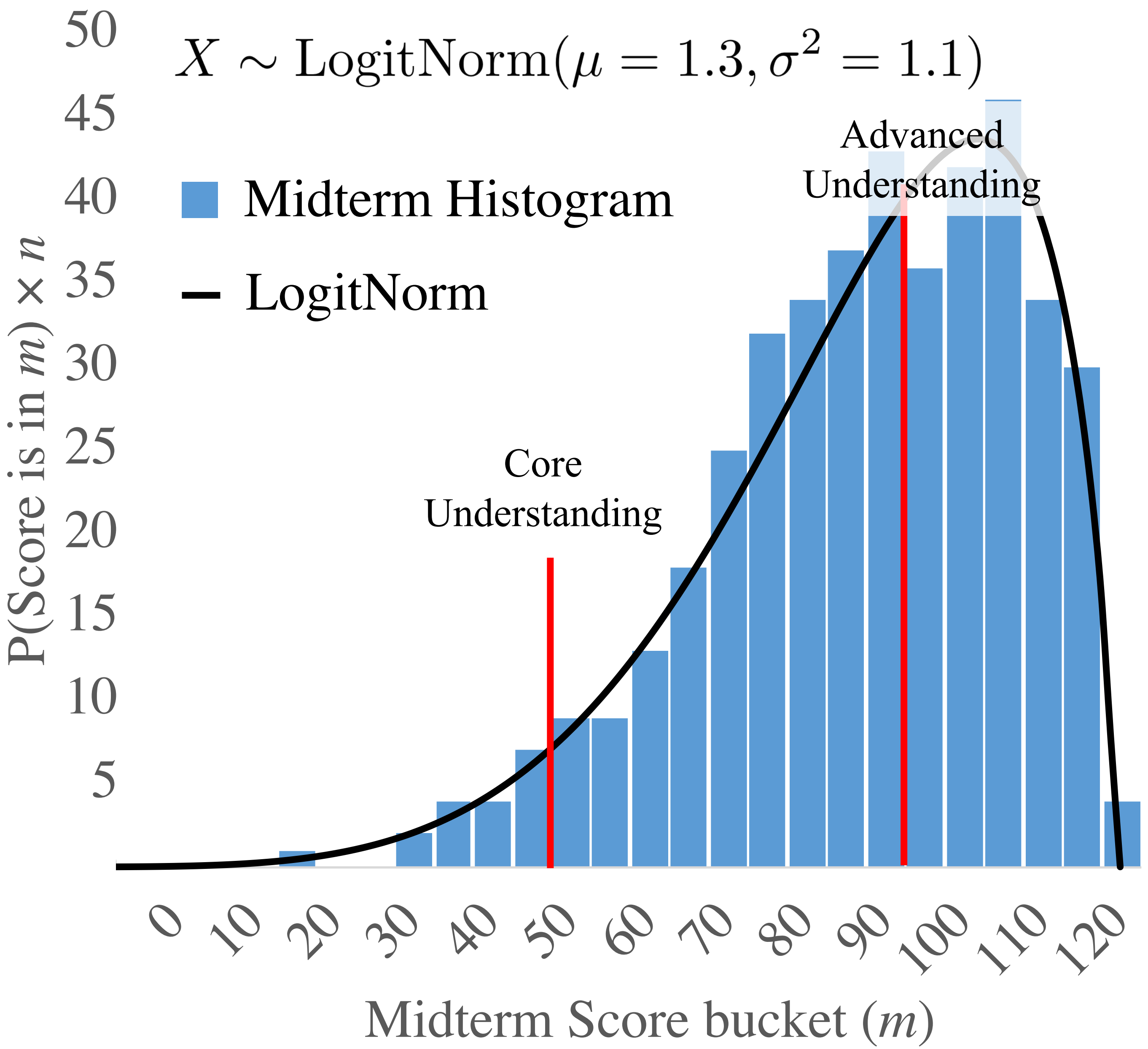
5



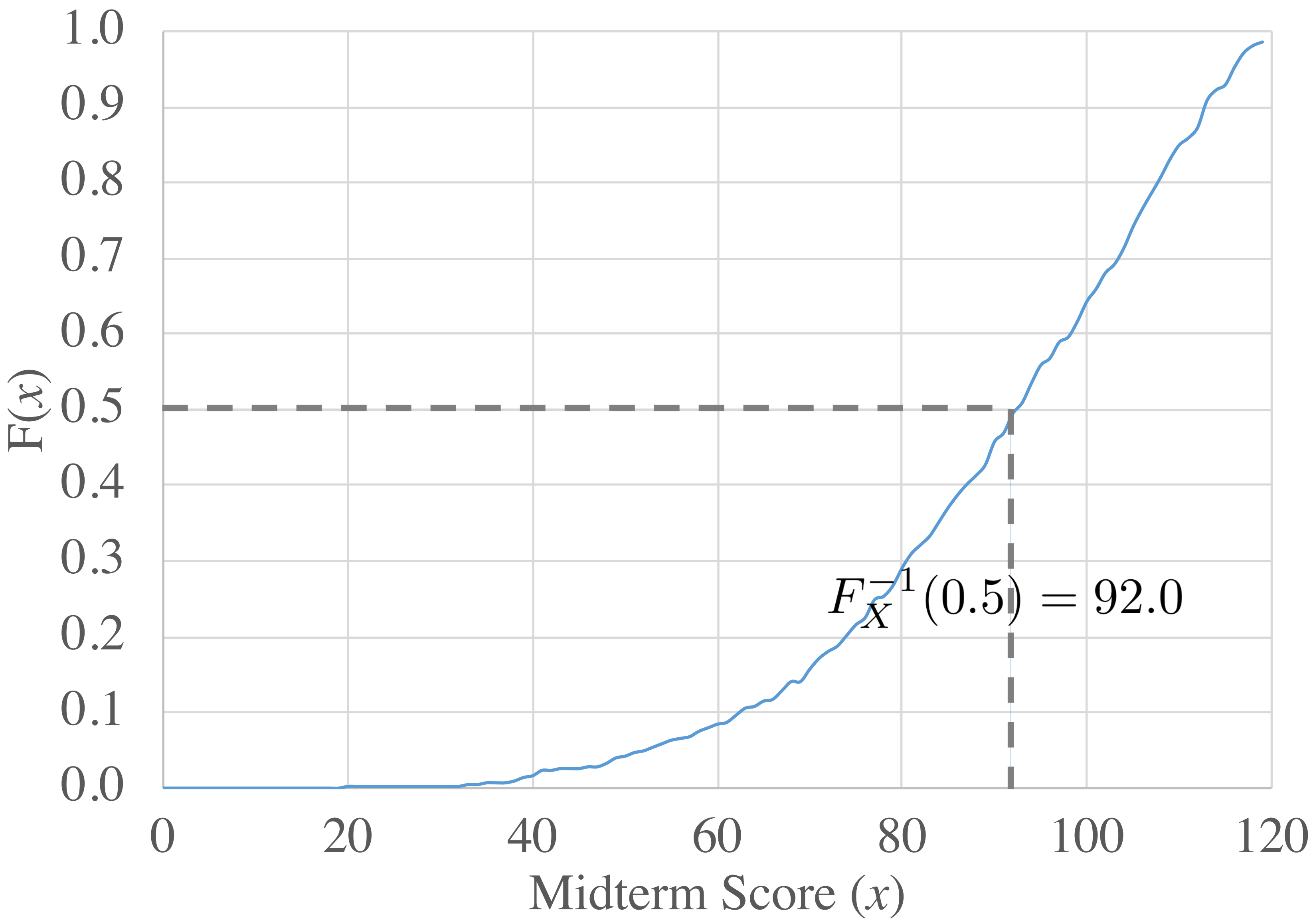


$X \sim \text{LogitNorm}(\mu = 1.3, \sigma^2 = 1.1)$
Midterm Histogram
LogitNorm
$E[X] = 89.5$
$\text{Std}(X) = 19.5$
$F_X^{-1}(0.5) = 92.0$
$P(\text{Perfect}) = 0.009$
P(Score is in m) $\times n$
Midterm Score bucket (m)
50
45
40
35
30
25
20
15
10
5
0
10
20
30
40
50
60
70
80
90
100
110
120

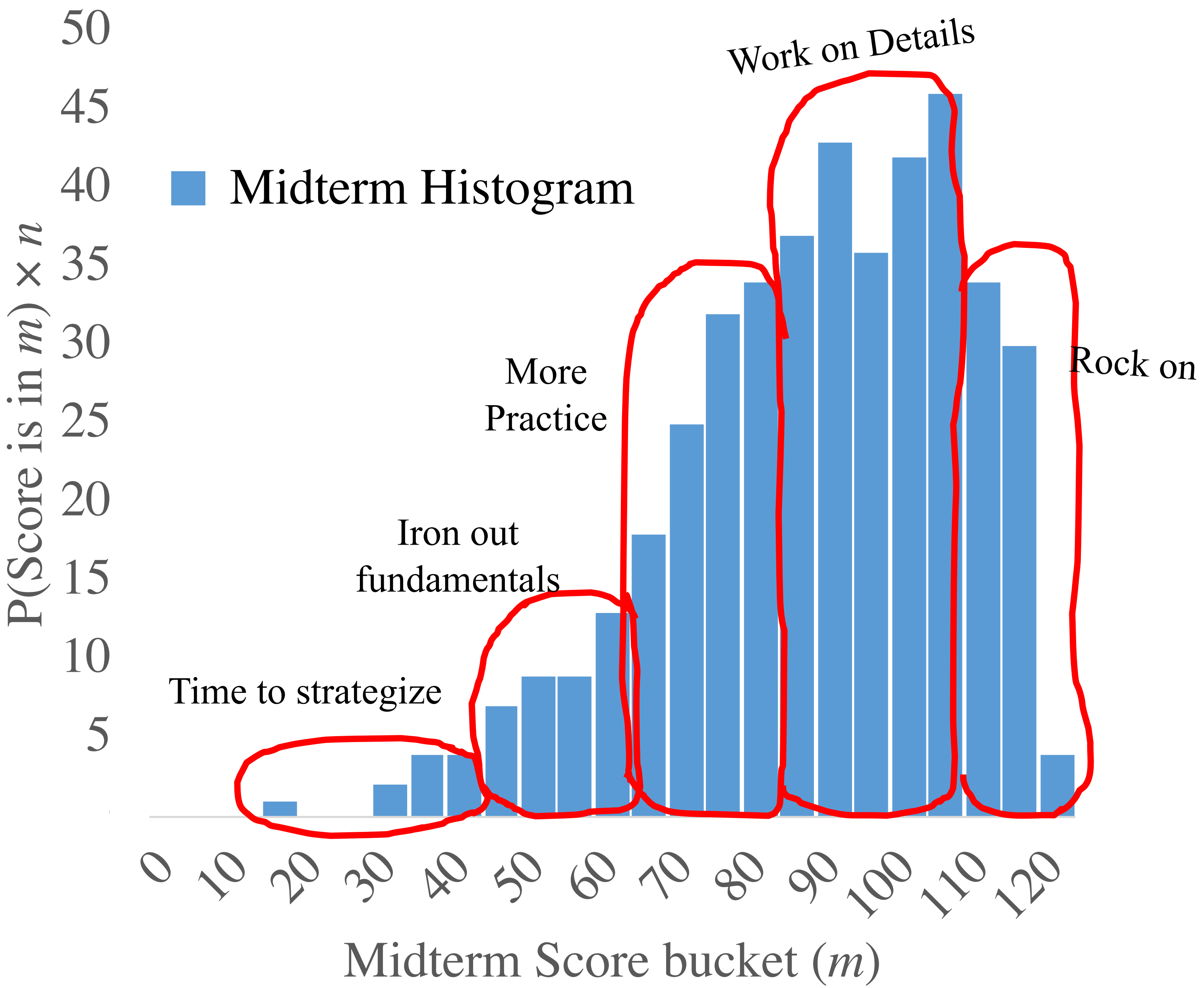
Midterm Distribution



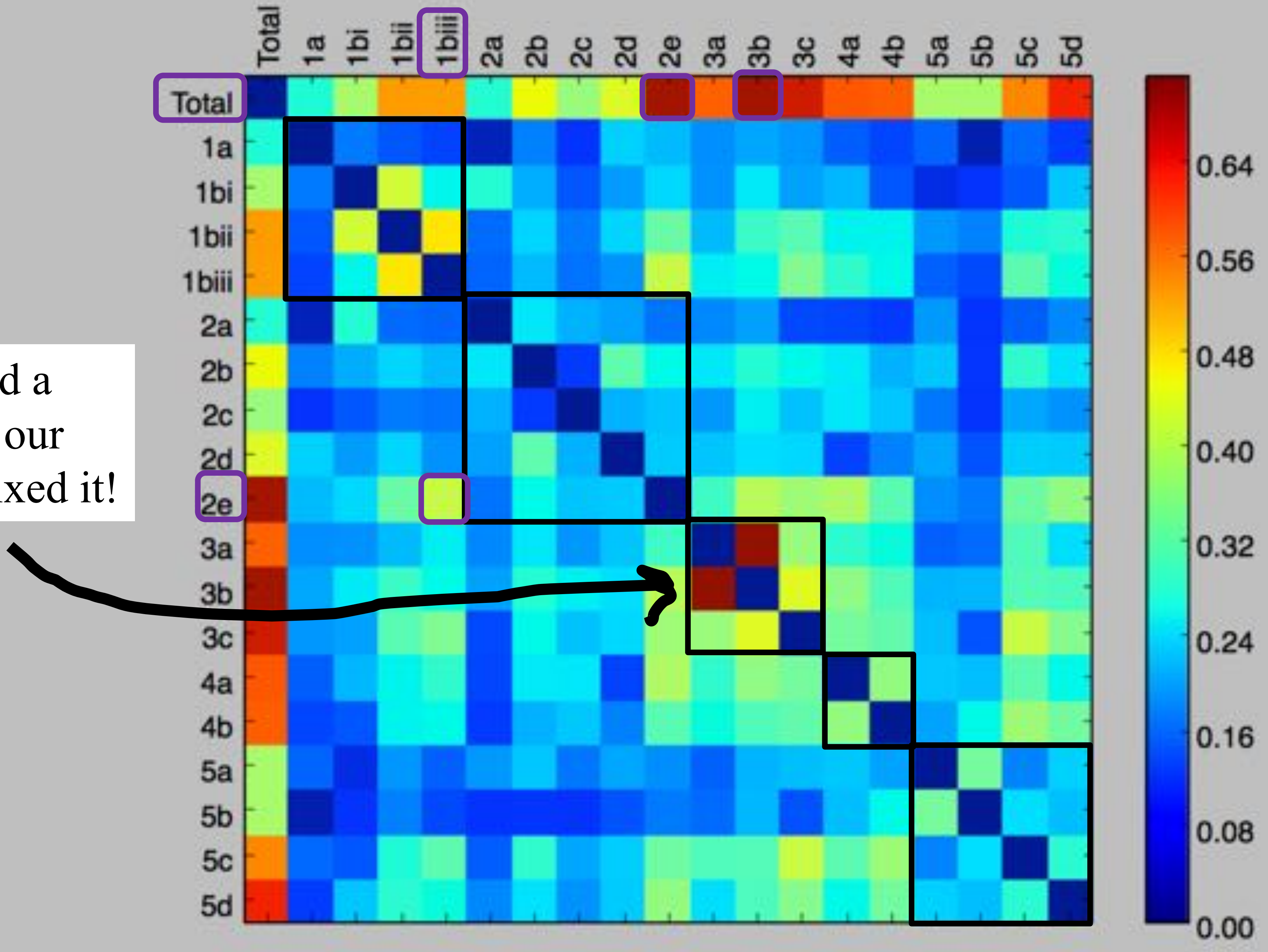
Midterm Cumulative Density



Midterm Distribution



We noticed a mistake in our grading and fixed it!

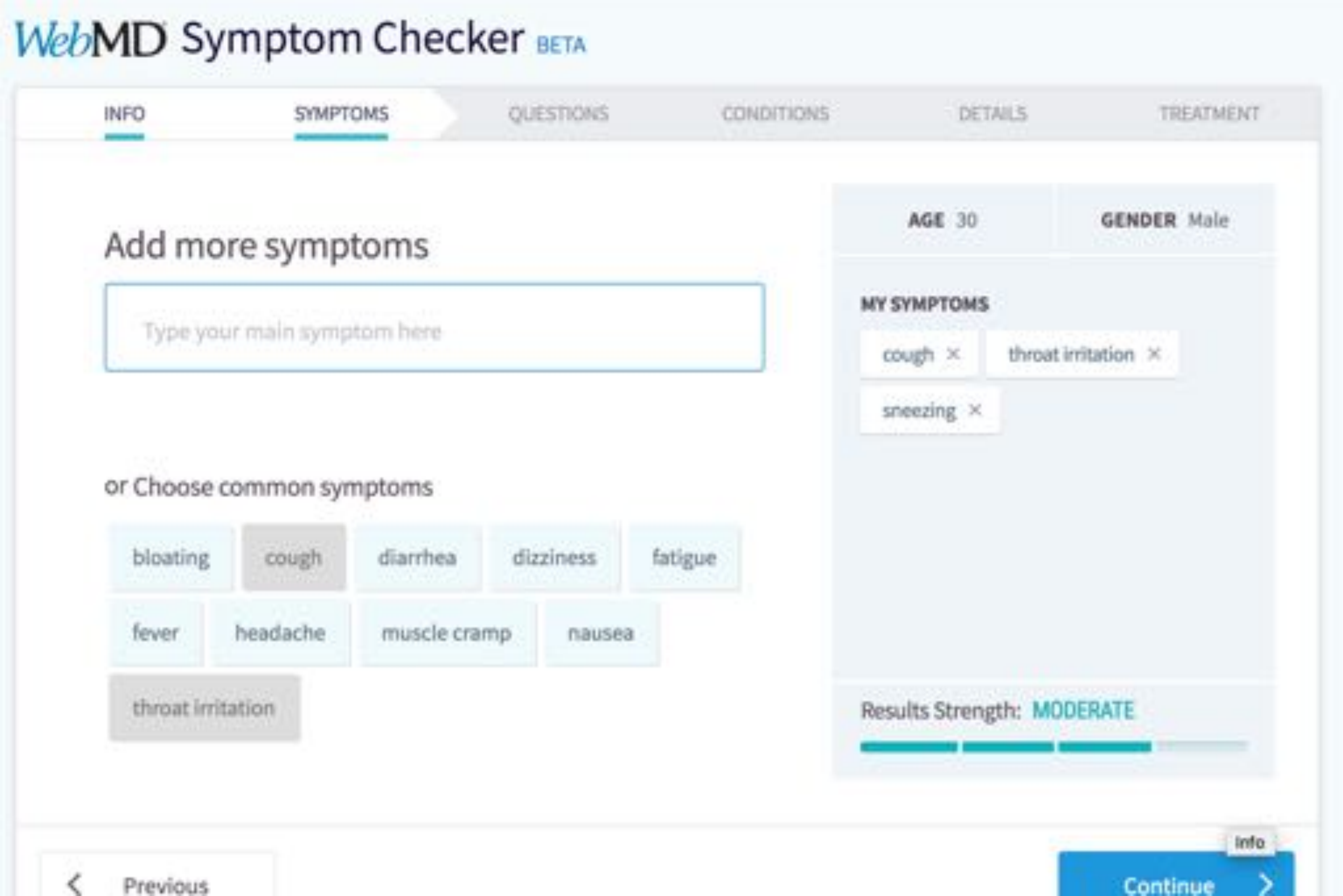


CS109 Contest

Something brand **new**…

General “Inference”

General "Inference"



Conditions that match your symptoms

UNDERSTANDING YOUR RESULTS ℹ

Influenza (flu) adults
Moderate match

Pneumococcal Infections
Moderate match

H1N1 Flu Virus (Swine Flu)
Moderate match

Bacterial Pneumonia
Moderate match

Sepsis (blood infection)
Moderate match

Gender **Male** Age **30** Edit

My Symptoms Edit

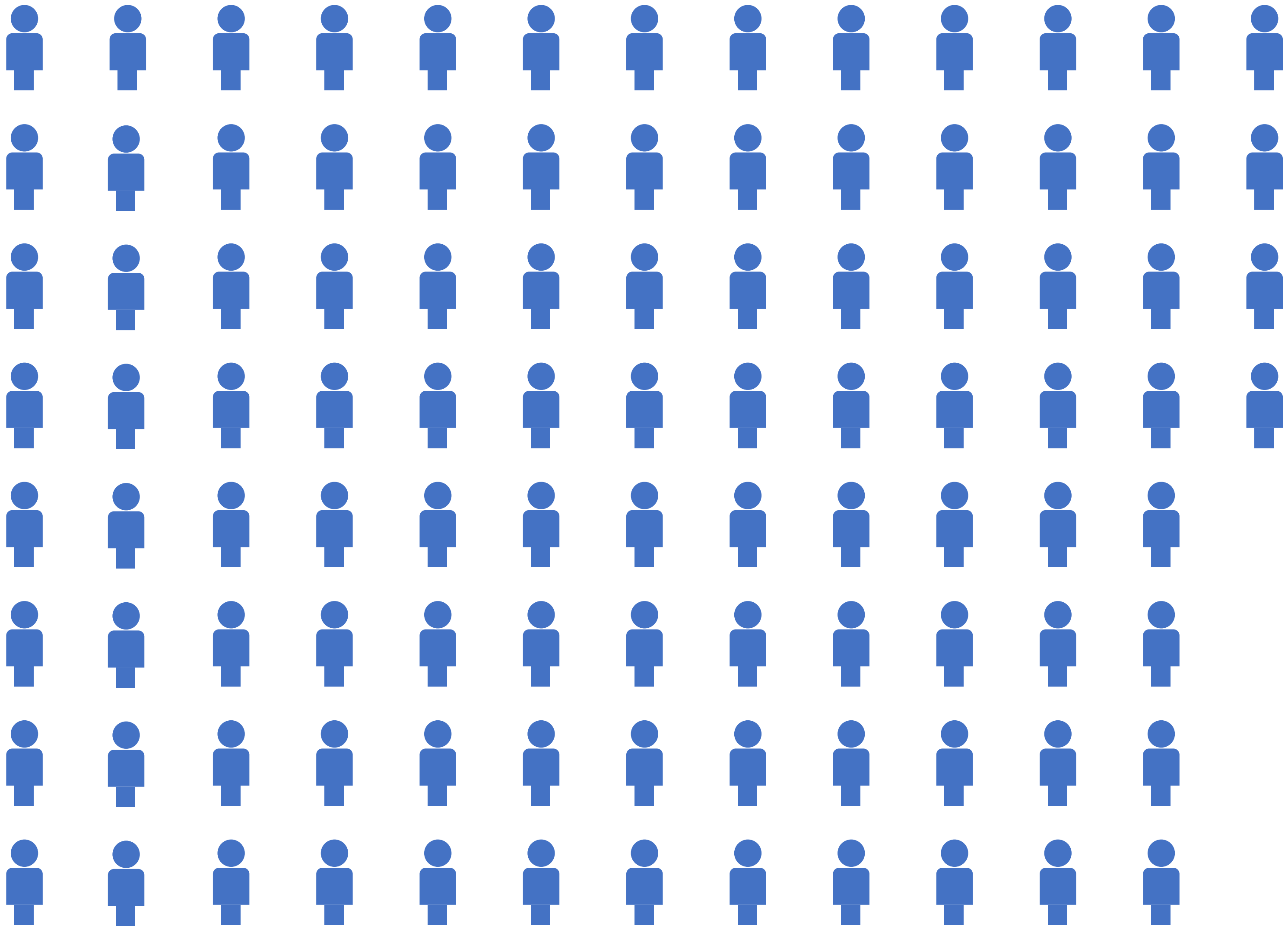
fever 103f to 104f, dizziness,

throat irritation, migraine headache

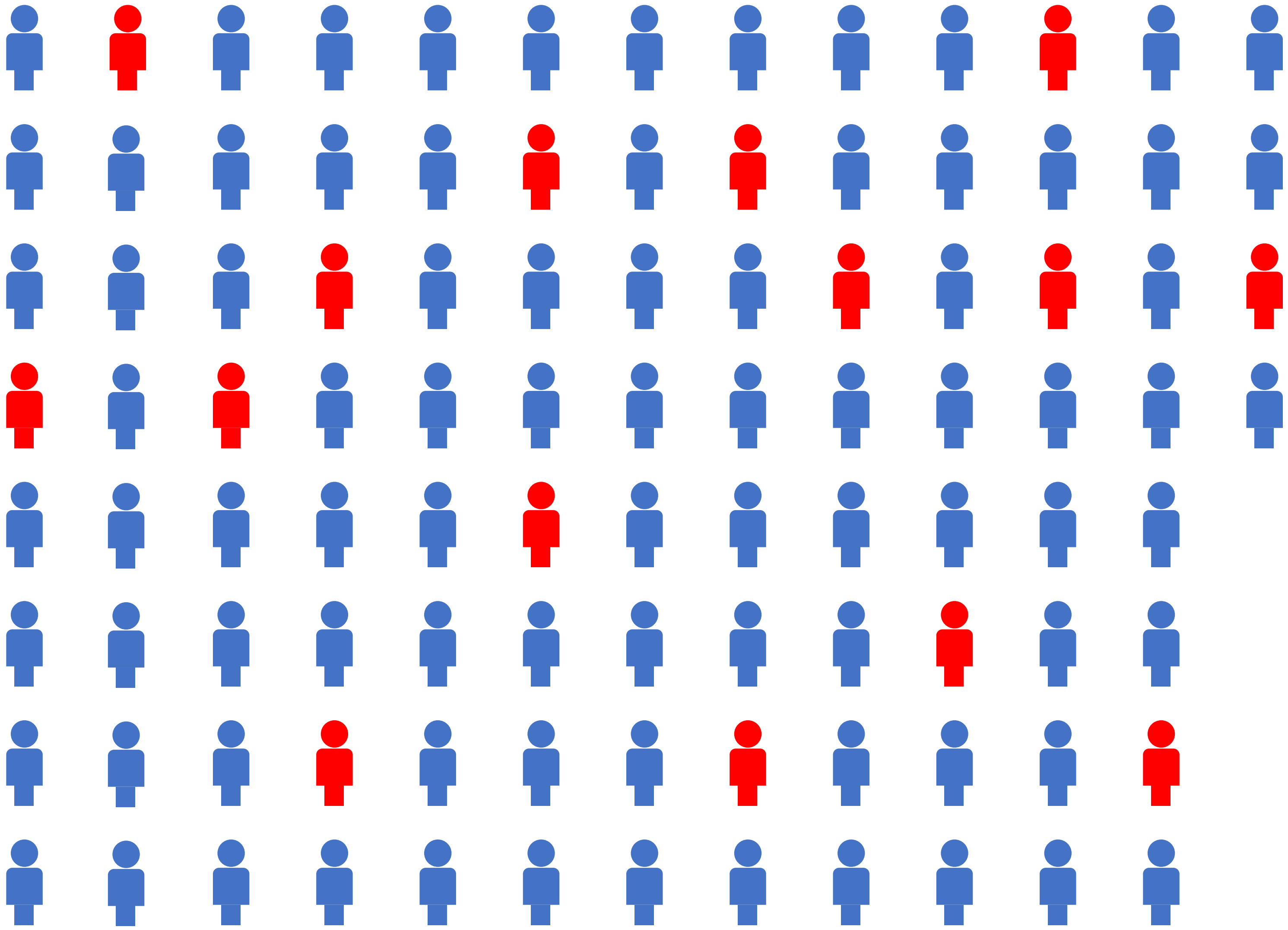
Start Over

Review

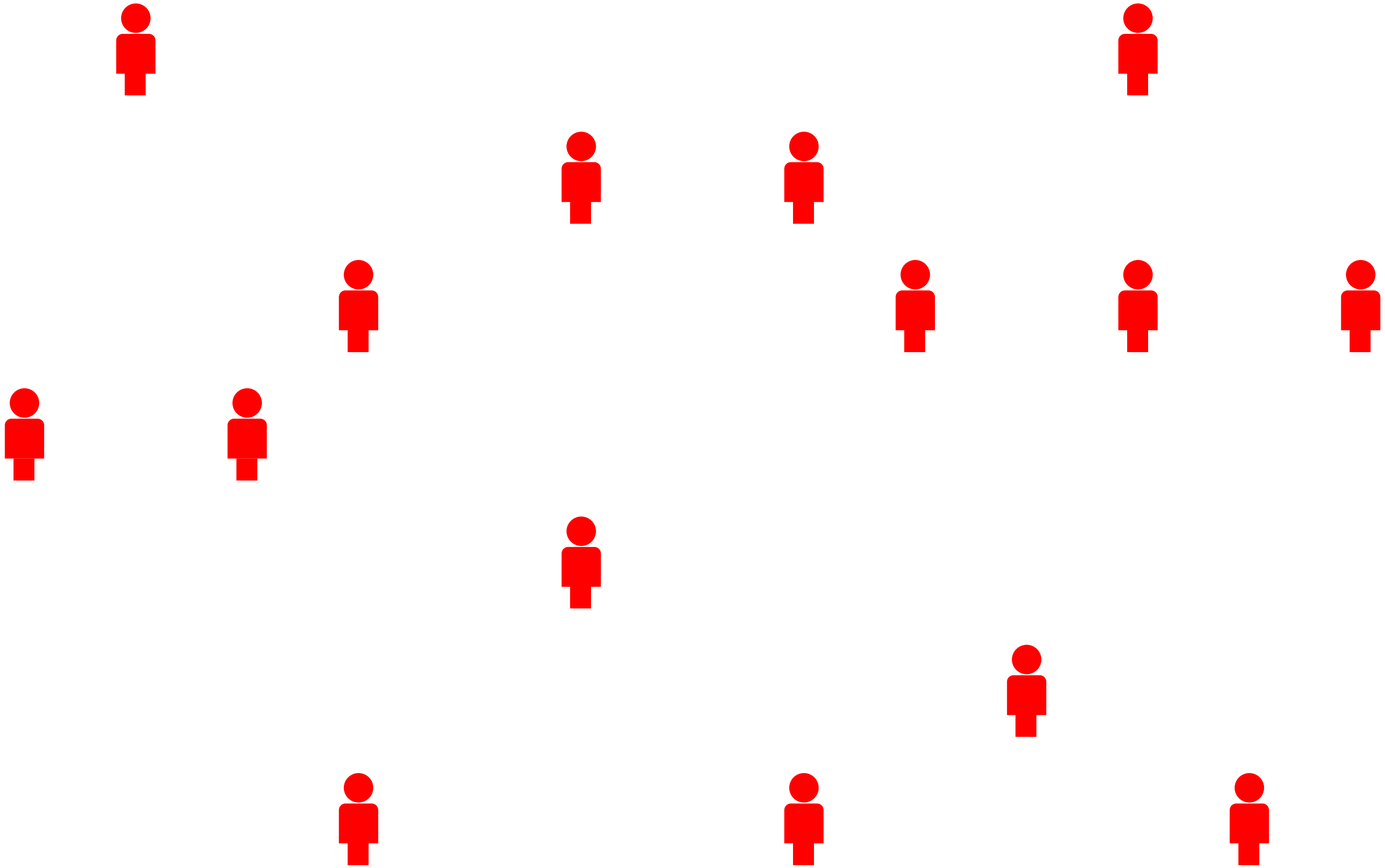
Population



Sample

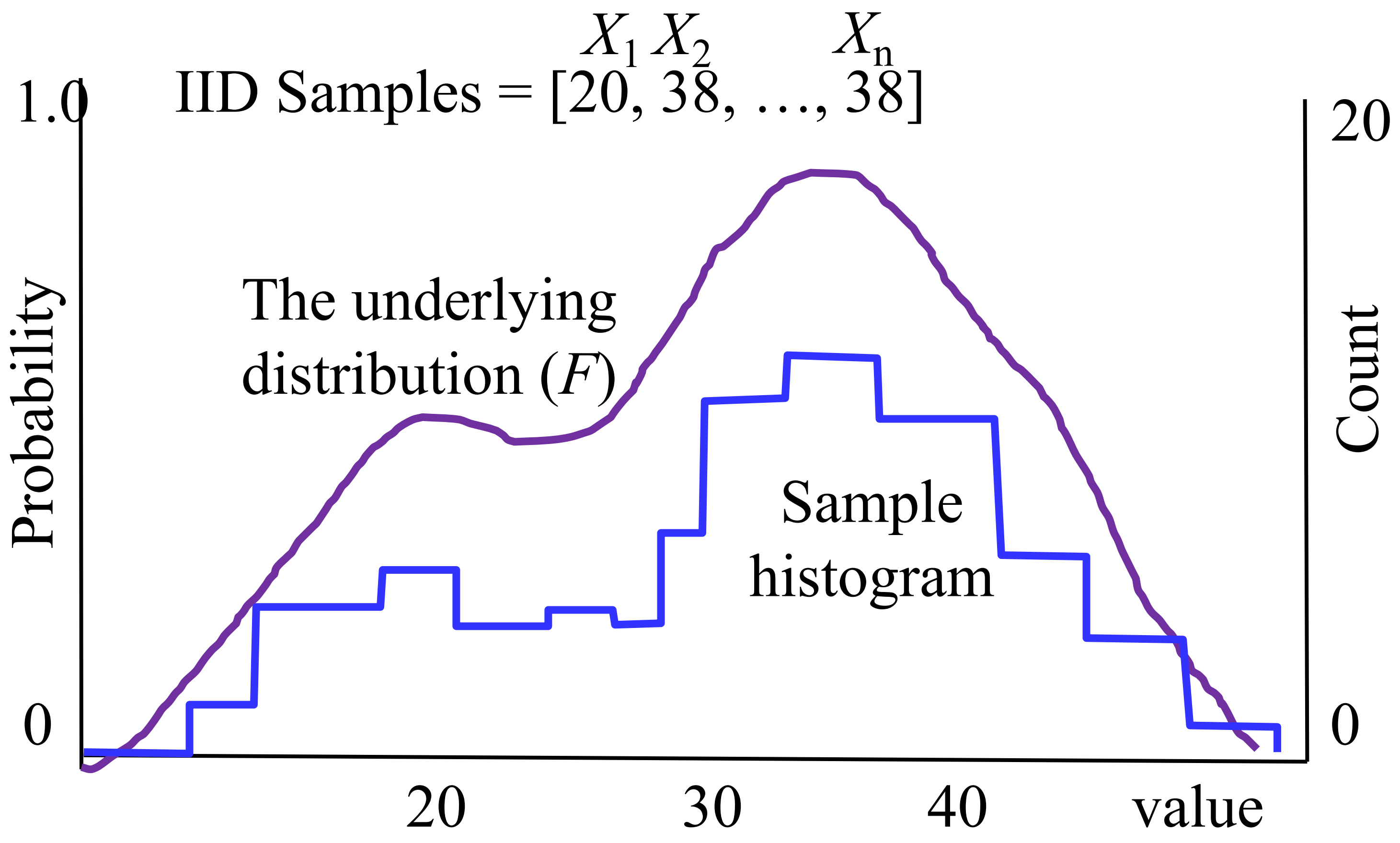


Sample



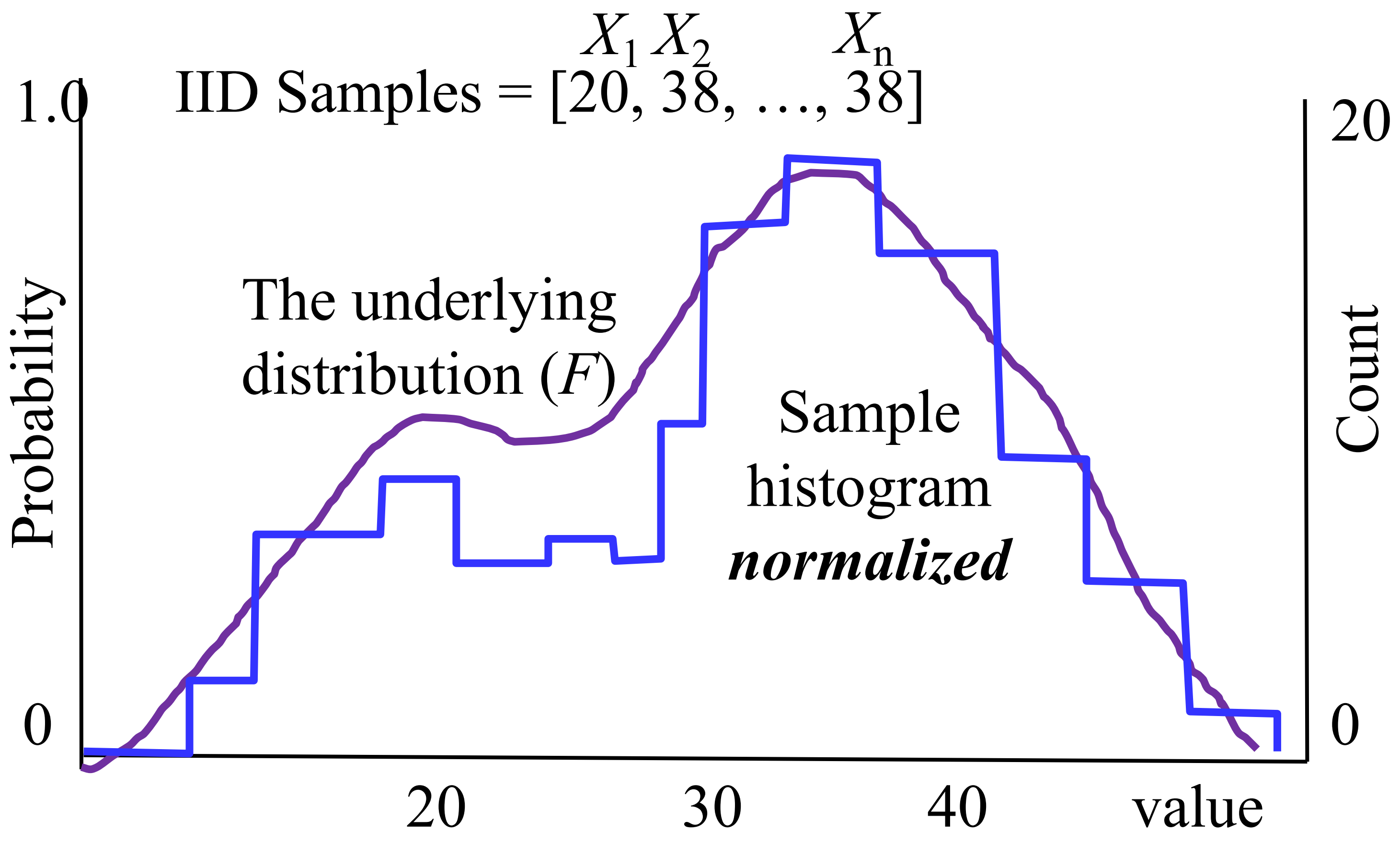
Collect one (or more) numbers from each person

Samples



X_1 X_2 X_n
IID Samples = [20, 38, …, 38]
1.0
0
Probability
The underlying distribution (F)
Sample histogram
20
0
Count
20
30
40
value

Samples



X1 X2 Xn
IID Samples = [20, 38, …, 38]
1.0
0
Probability
The underlying distribution (F)
Sample histogram normalized
20
0
Count
20
30
40
value

Sample Statistics

Sample Mean

$$\bar{X} = \sum_{i=1}^{n} \frac{X_i}{n}$$

Sample Variance

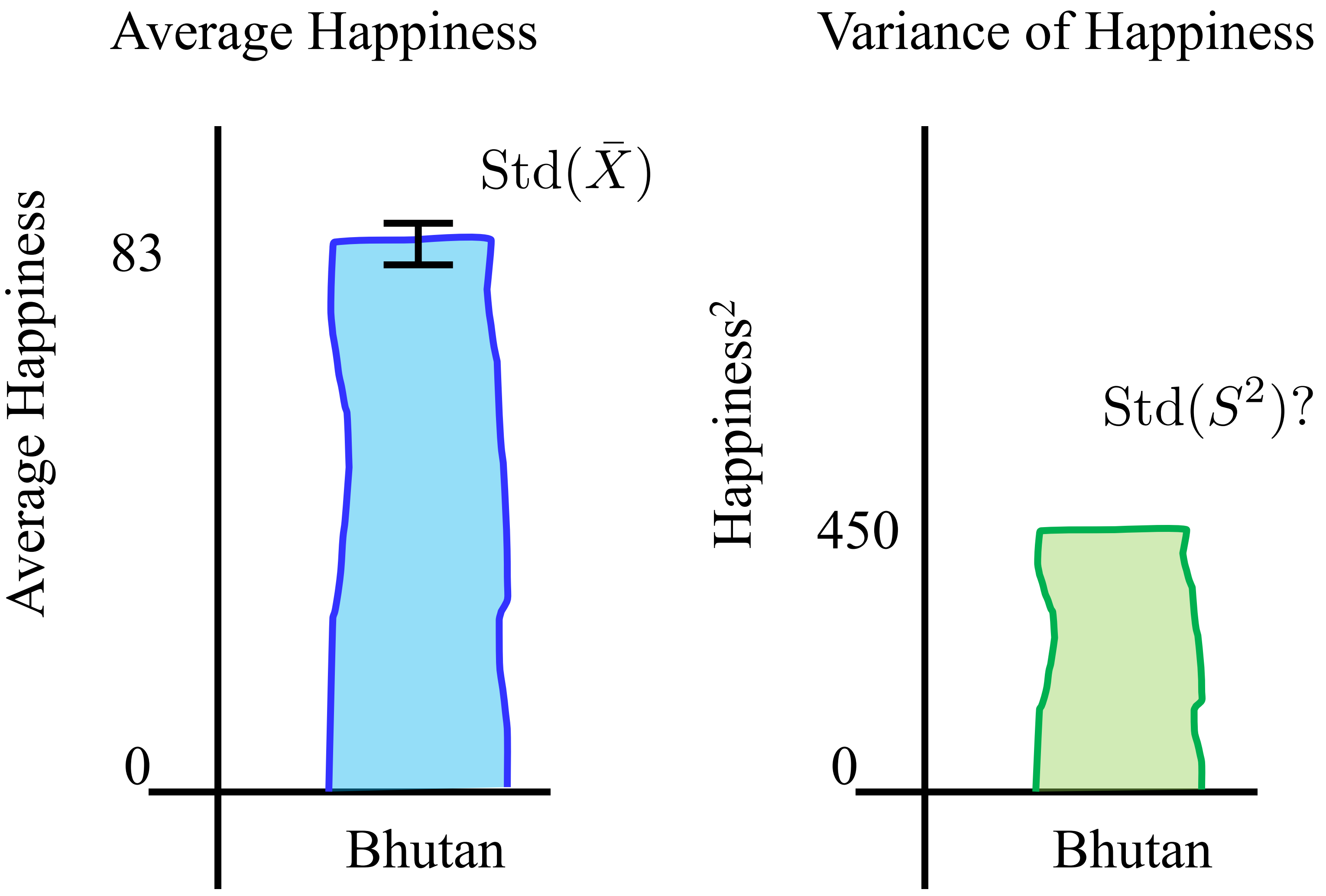
$$S^2 = \sum_{i=1}^{n} \frac{(X_i - \bar{X})^2}{n-1}$$

Oh my that can be thought of as a random variable

Var of Sample Mean

$$\mathrm{Var}(\bar{X}) = \frac{S^2}{n}$$

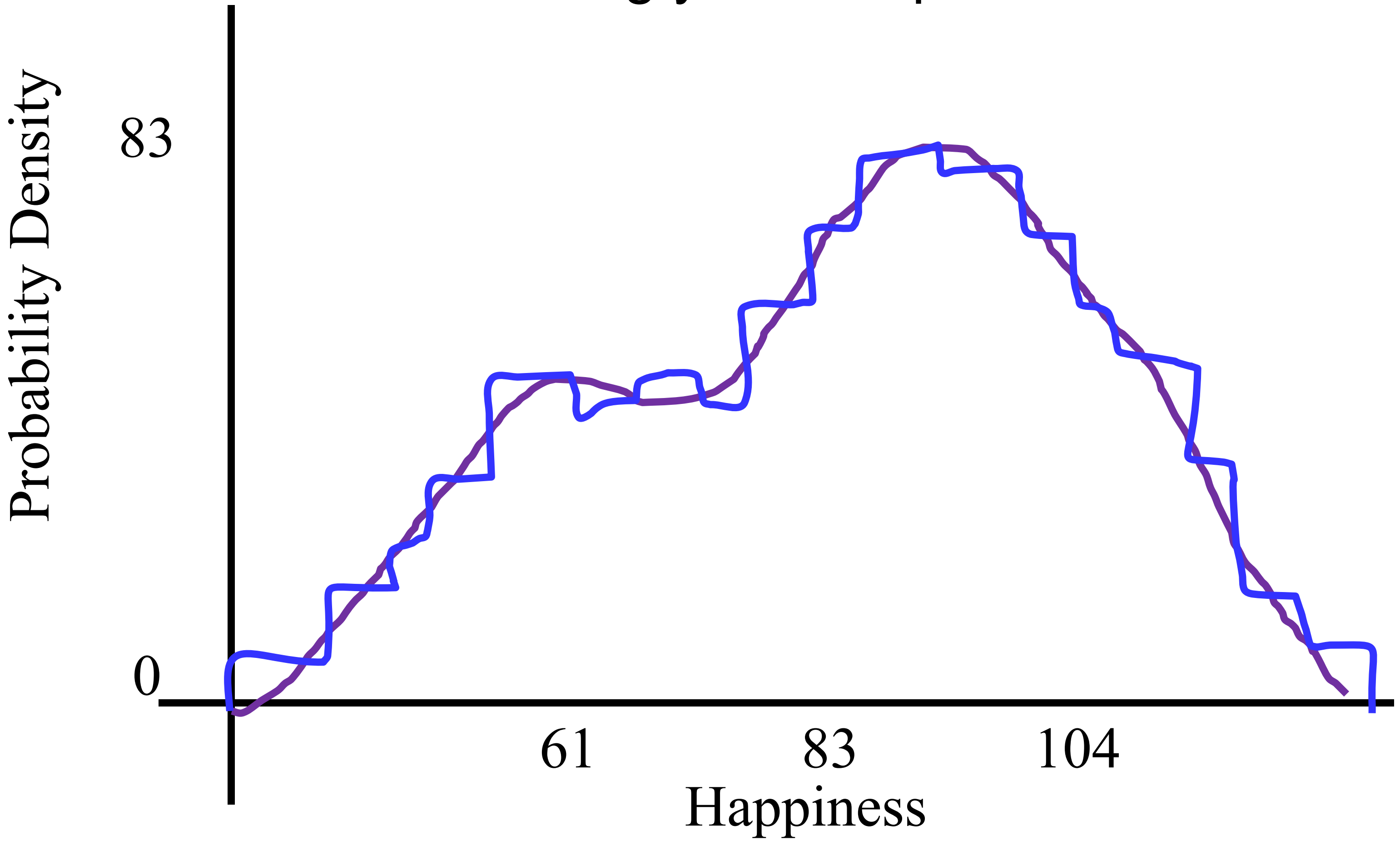
Sample Mean



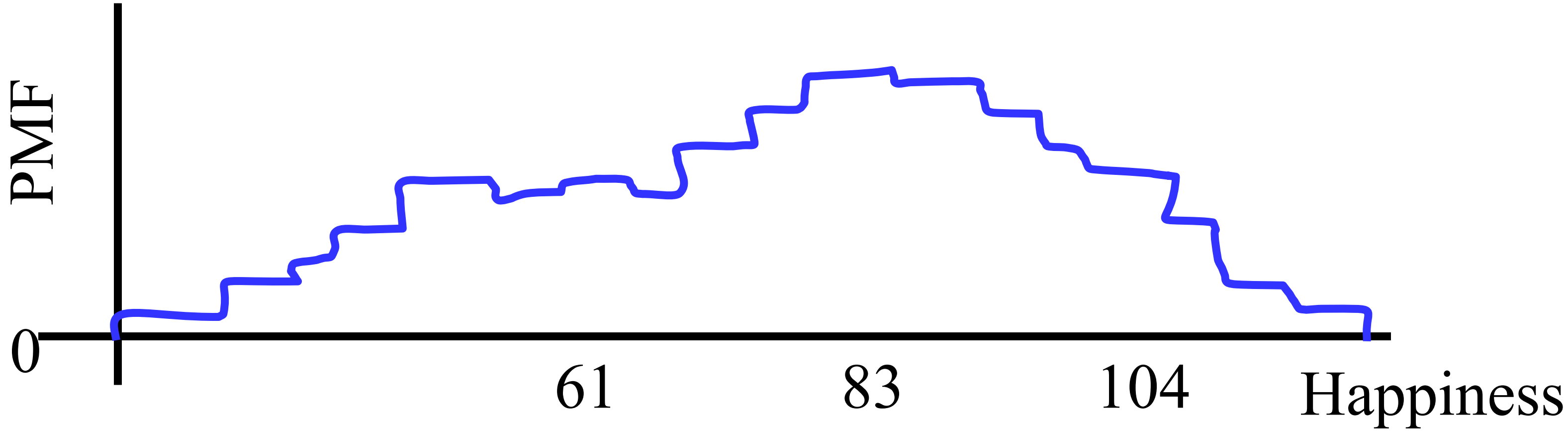
Claim: The average happiness of Bhutan is 83 ± 2

Bootstrap Insight

You can estimate the PMF of the underlying distribution, using your sample.



Bootstrap for Any Stat



Bootstrap Algorithm (sample):

1. Repeat **10,000** times:
   a. **Choose len(sample) elems from sample, with replacement**
   b. Recalculate the stat on the resample
2. You now have a **distribution of your stat**

A **non-parametric** continuous distribution can be represented in a **computer** as a **list** of numbers sampled from the distribution

Vars = [472.7, 478.4, 469.2, …, 476.2]

End Review

Bootstrap for p values

Null Hypothesis Test

Population 1
4.44
3.36
5.87
2.31
...
3.70

Population 2
2.15
3.01
2.02
1.43
...
1.83

$\mu_1 = 3.1$

$\mu_2 = 2.4$

Null Hypothesis Test

Nepal Happiness
4.44
3.36
5.87
2.31
...
3.70

$\mu_1 = 3.1$

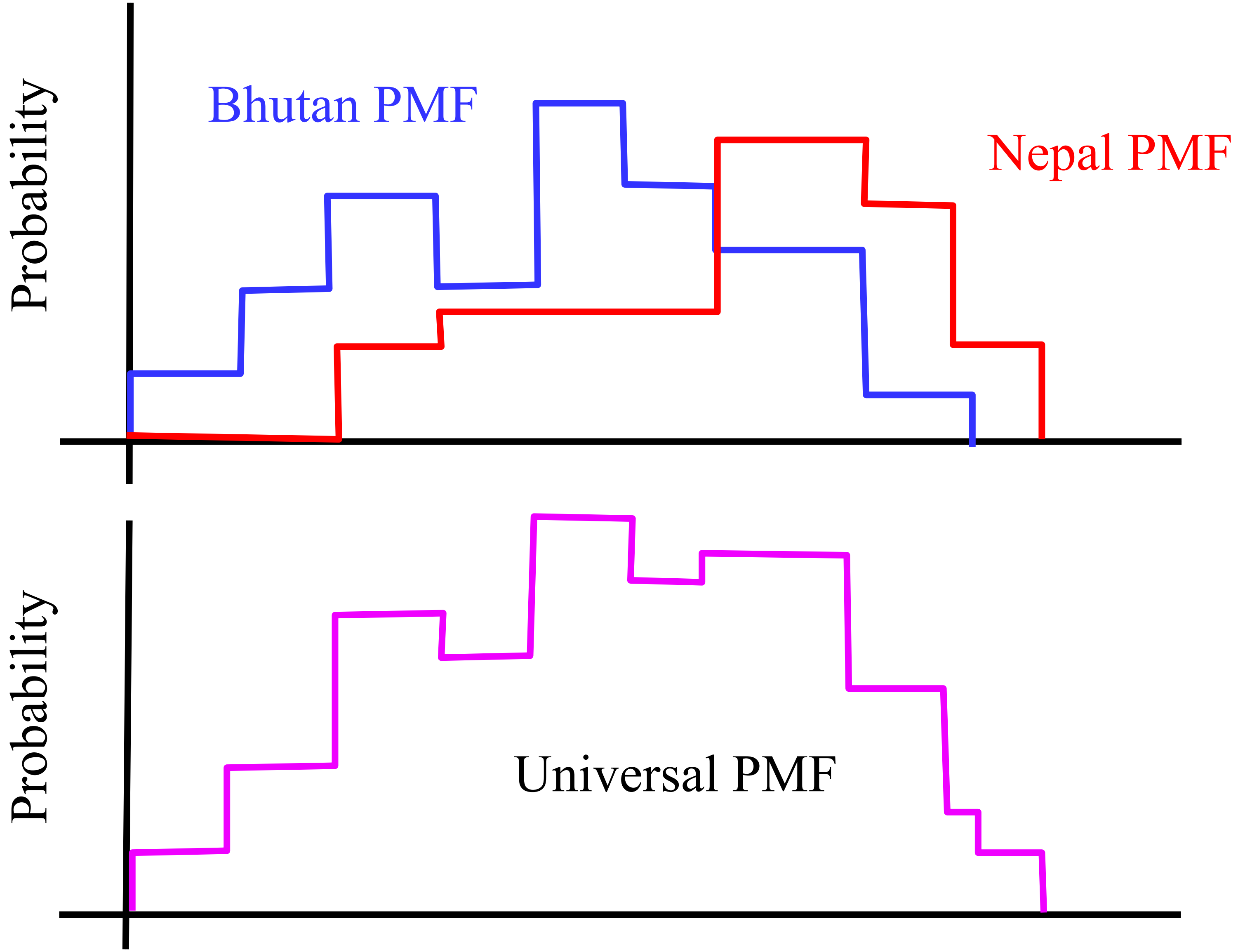
Bhutan Happiness
2.15
3.01
2.02
1.43
...
1.83

$\mu_2 = 2.4$

Claim: The difference in happiness between Nepal and Bhutan is 0.7 happiness points.

Null hypothesis: even if **there is no pattern** (ie the two samples are identically distributed) your claim might have arisen by **chance**.

Universal Sample



Bootstrap for P Values

```
def pvalueBootstrap(bhutanSample,nepalSample):
   N = size of the bhutanSample
   M = size of the nepalSample

   uniSamples = combine bhutanSamples and nepalSamples
   count = 0

   repeat 10,000 times:
      bhutanResample = draw N resamples from the uniSamples
      nepalResample = draw M resamples from the uniSamples
      muBhutan = sample mean of the bhutanResample
      muNepal = sample mean of the nepalResample
      meanDiff = |muNepal – muBhutan|
      if meanDiff > observedDifference:
         count += 1

   pValue = count / 10,000
```

Bootstrap for P Values

```
def pvalueBootstrap(bhutanSample,nepalSample):
   N = size of the bhutanSample
   M = size of the nepalSample

   uniSamples = combine bhutanSamples and nepalSamples
   count = 0

   repeat 10,000 times:
      bhutanResample = draw N resamples from the uniSamples
      nepalResample = draw M resamples from the uniSamples
      muBhutan = sample mean of the bhutanResample
      muNepal = sample mean of the nepalResample
      meanDiff = |muNepal - muBhutan|
      if meanDiff > observedDifference:
         count += 1

   pValue = count / 10,000
```

Bootstrap for P Values

```
def pvalueBootstrap(bhutanSample,nepalSample):
   N = size of the bhutanSample
   M = size of the nepalSample

   uniSamples = combine bhutanSamples and nepalSamples
   count = 0

   repeat 10,000 times:
      bhutanResample = draw N resamples from the uniSamples
      nepalResample = draw M resamples from the uniSamples
      muBhutan = sample mean of the bhutanResample
      muNepal = sample mean of the nepalResample
      meanDiff = |muNepal - muBhutan|
      if meanDiff > observedDifference:
         count += 1

   pValue = count / 10,000
```

Bootstrap for P Values

```
def pvalueBootstrap(bhutanSample,nepalSample):
   N = size of the bhutanSample
   M = size of the nepalSample

   uniSamples = combine bhutanSamples and nepalSamples
   count = 0

   repeat 10,000 times:
      bhutanResample = draw N resamples from the uniSamples
      nepalResample = draw M resamples from the uniSamples
      muBhutan = sample mean of the bhutanResample
      muNepal = sample mean of the nepalResample
      meanDiff = |muNepal - muBhutan|
      if meanDiff > observedDifference:
         count += 1

   pValue = count / 10,000
```

Bootstrap for P Values

```
def pvalueBootstrap(bhutanSample,nepalSample):
   N = size of the bhutanSample
   M = size of the nepalSample

   uniSamples = combine bhutanSamples and nepalSamples
   count = 0

   repeat 10,000 times:
      bhutanResample = draw N resamples from the uniSamples
      nepalResample = draw M resamples from the uniSamples
      muBhutan = sample mean of the bhutanResample
      muNepal = sample mean of the nepalResample
      meanDiff = |muNepal - muBhutan|
      if meanDiff > observedDifference:
         count += 1

   pValue = count / 10,000
```

Bootstrap for P Values

```
def pvalueBootstrap(bhutanSample,nepalSample):
   N = size of the bhutanSample
   M = size of the nepalSample

   uniSamples = combine bhutanSamples and nepalSamples
   count = 0

   repeat 10,000 times:
      bhutanResample = draw N resamples from the uniSamples
      nepalResample = draw M resamples from the uniSamples
      muBhutan = sample mean of the bhutanResample
      muNepal = sample mean of the nepalResample
      meanDiff = |muNepal - muBhutan|
      if meanDiff > observedDifference:
         count += 1

   pValue = count / 10,000
```

Bootstrap for P Values

```
def pvalueBootstrap(bhutanSample,nepalSample):
   N = size of the bhutanSample
   M = size of the nepalSample

   uniSamples = combine bhutanSamples and nepalSamples
   count = 0

   repeat 10,000 times:
      bhutanResample = draw N resamples from the uniSamples
      nepalResample = draw M resamples from the uniSamples
      muBhutan = sample mean of the bhutanResample
      muNepal = sample mean of the nepalResample
      meanDiff = |muNepal - muBhutan|
      if meanDiff > observedDifference:
         count += 1

   pValue = count / 10,000
```

Bootstrap for P Values

```
def pvalueBootstrap(bhutanSample,nepalSample):
   N = size of the bhutanSample
   M = size of the nepalSample

   uniSamples = combine bhutanSamples and nepalSamples
   count = 0

   repeat 10,000 times:
      bhutanResample = draw N resamples from the uniSamples
      nepalResample = draw M resamples from the uniSamples
      muBhutan = sample mean of the bhutanResample
      muNepal = sample mean of the nepalResample
      meanDiff = |muNepal - muBhutan|
      if meanDiff > observedDifference:
         count += 1

   pValue = count / 10,000
```

With replacement!

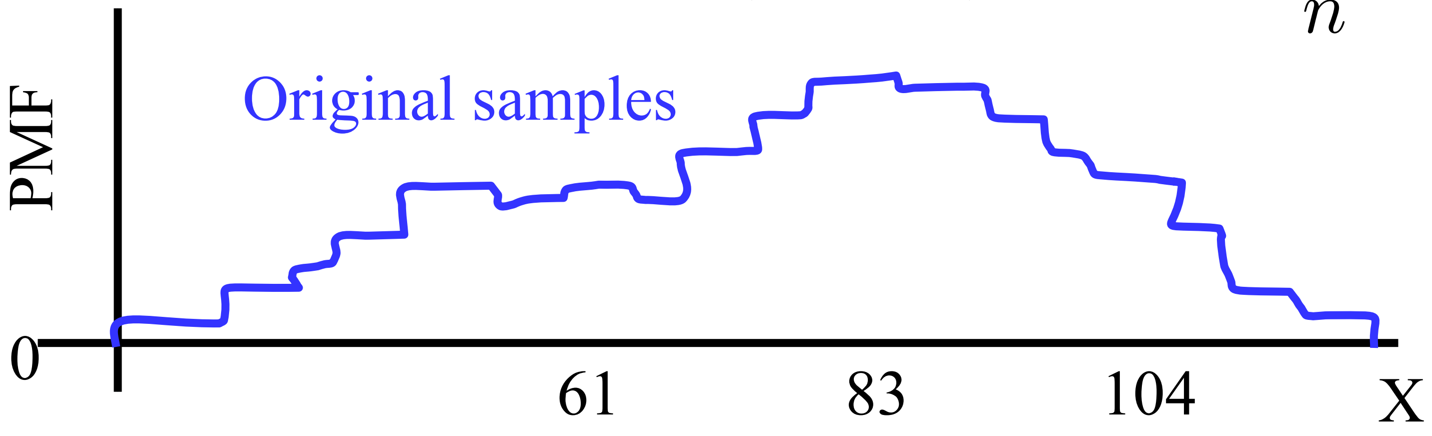
Algorithm in Practice

```
def resample(samples):
    # Estimate the PMF using the samples
    # Draw K new samples from the PMF
```

Algorithm in Practice

```
def resample(samples):
    # Estimate the PMF using the samples
    # Draw K new samples from the PMF
    return np.random.choice(samples, K,
                    replace = True)
```

$$P(X = k) = \frac{\text{count}(X = k)}{n}$$



Bootstrap

Lets try it!

Null Hypothesis Test

Nepal Happiness
4.44
3.36
5.87
2.31
...
3.70

$\mu_1 = 3.1$

Bhutan Happiness
2.15
3.01
2.02
1.43
...
1.83

$\mu_2 = 2.4$

Claim: The difference in happiness between Nepal and Bhutan is 0.7 happiness points ($p < 0.008$).

Null Hypothesis Test

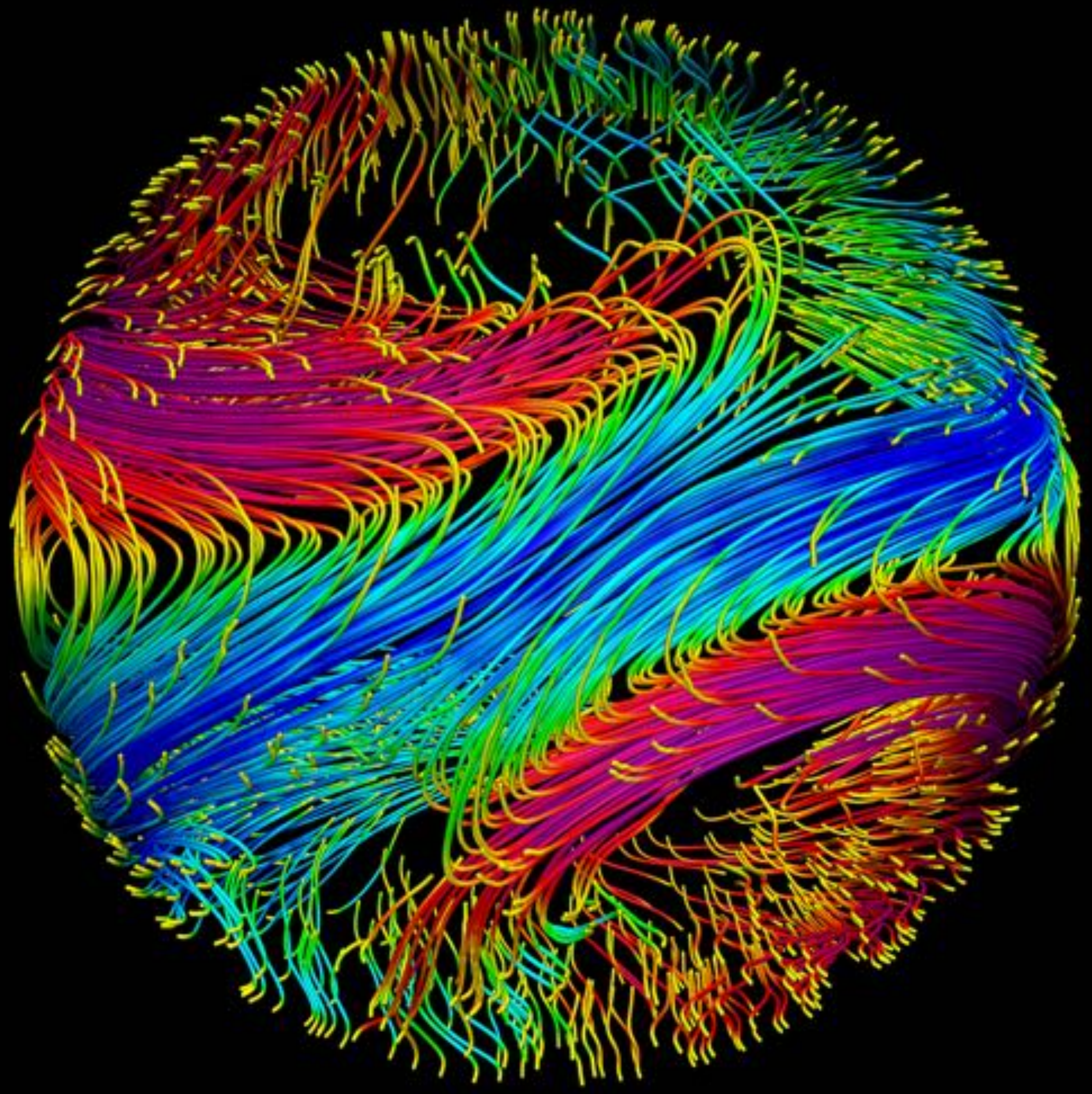
Nepal Happiness	Bhutan Happiness
4.44	2.15
3.36	3.01
5.87	2.02
2.31	
...	
3.70	

What about your confidence in this number?

$\mu_1 = 3.1$ $\mu_2 = 2.4$

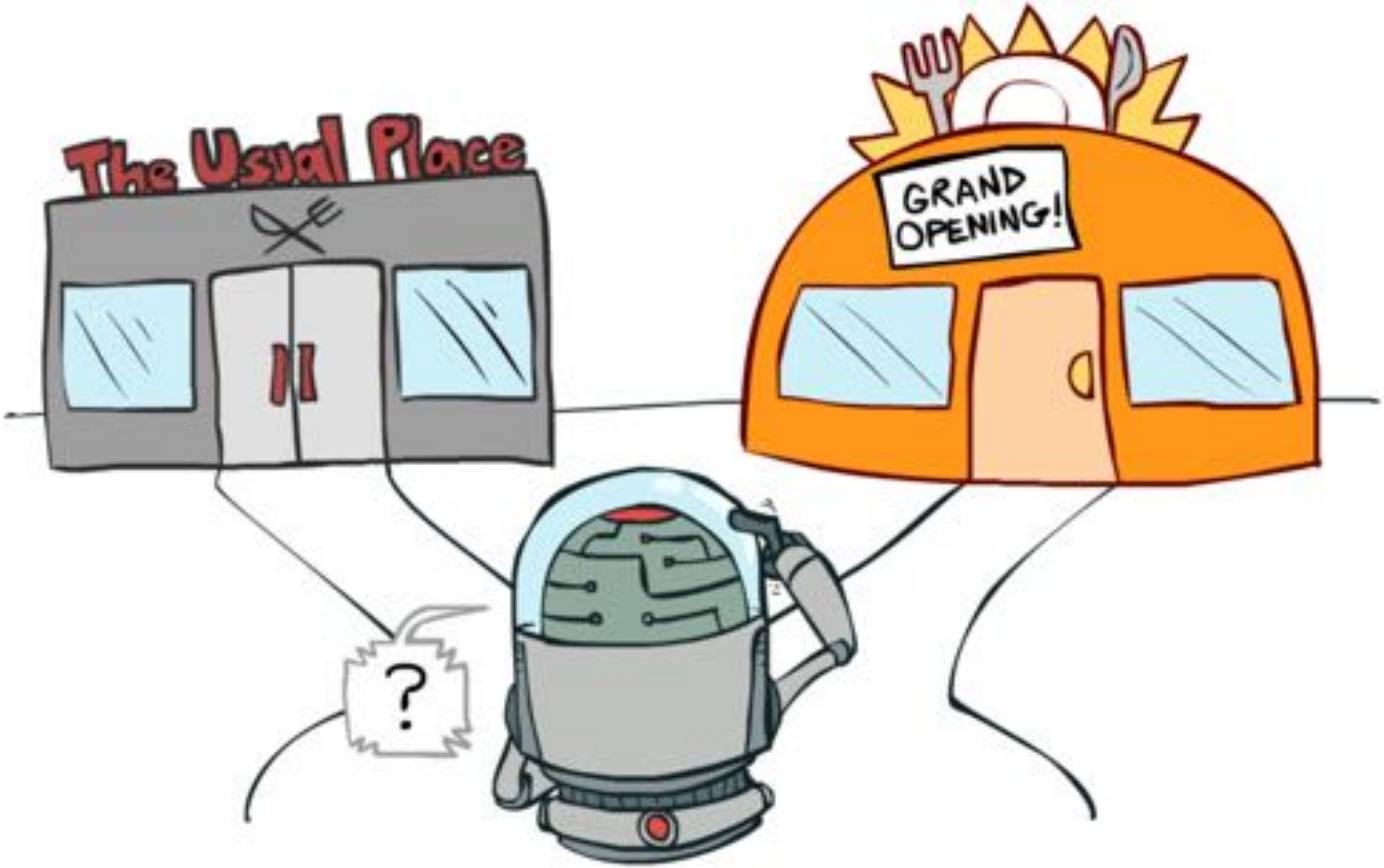
Claim: The difference in happiness between Nepal and Bhutan is 0.7 happiness points (p < 0.008).

Randomized Algorithms



Bootstrapping

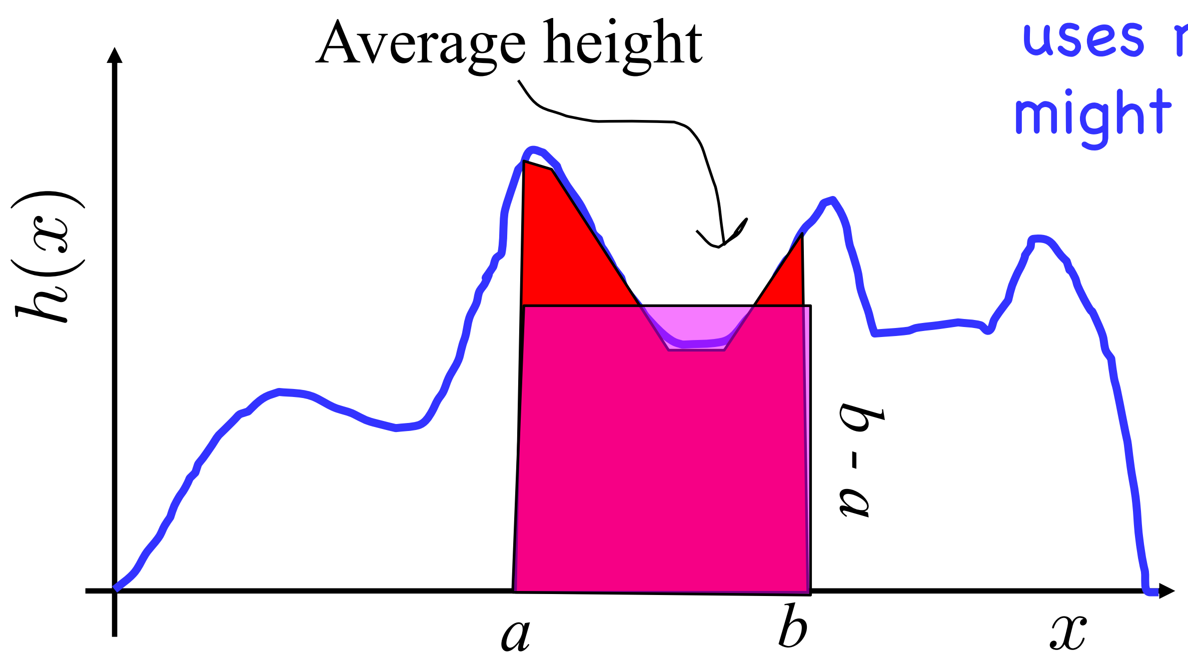
Thompson Sampling



Monte Carlo Integration

Generate N values $(X_1, X_2, \ldots, X_N)$ uniformly sampled over a range (a, b). We can approximate the integral of a function h over (a, b) as:

$$\int_a^b h(x)\, dx \approx \frac{(b-a)}{N} \sum_{i=1}^{N} h(X_i)$$



A "Monte Carlo" algorithm uses randomization but might not get the right answer

A Rose by Any Other Name

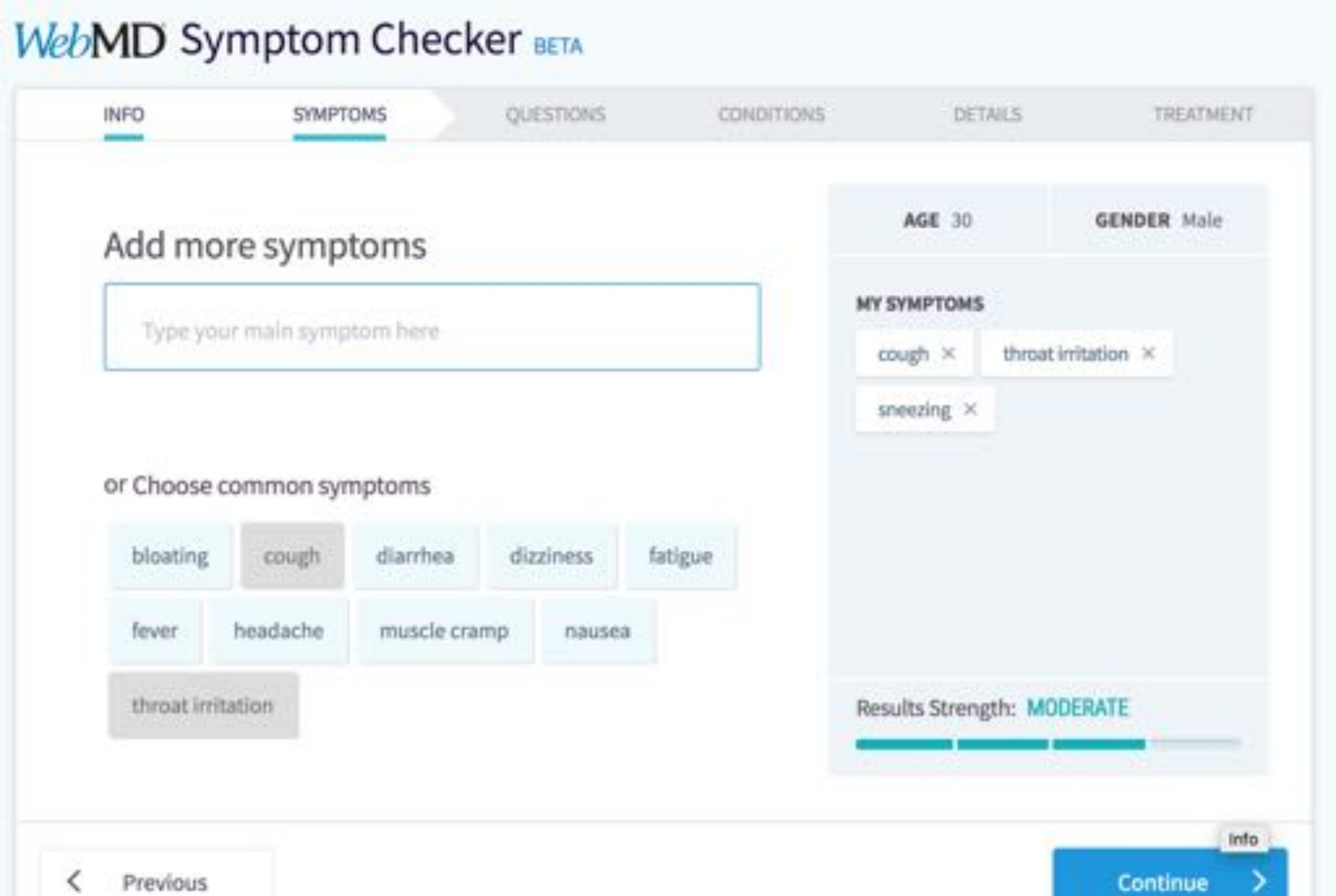
Las Vegas, Nevada

Monte Carlo, Monaco

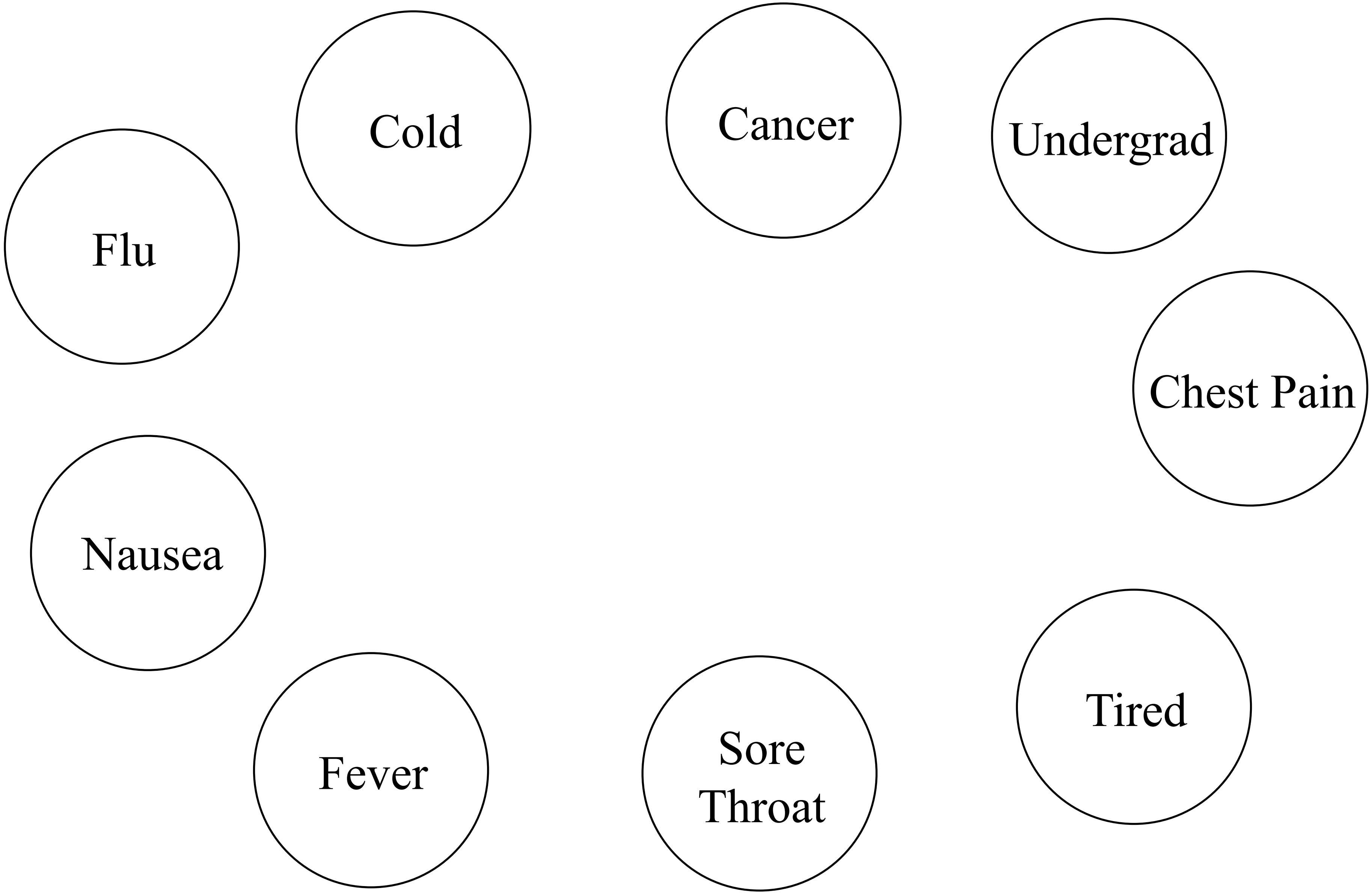
Something brand **new**…

General “Inference”

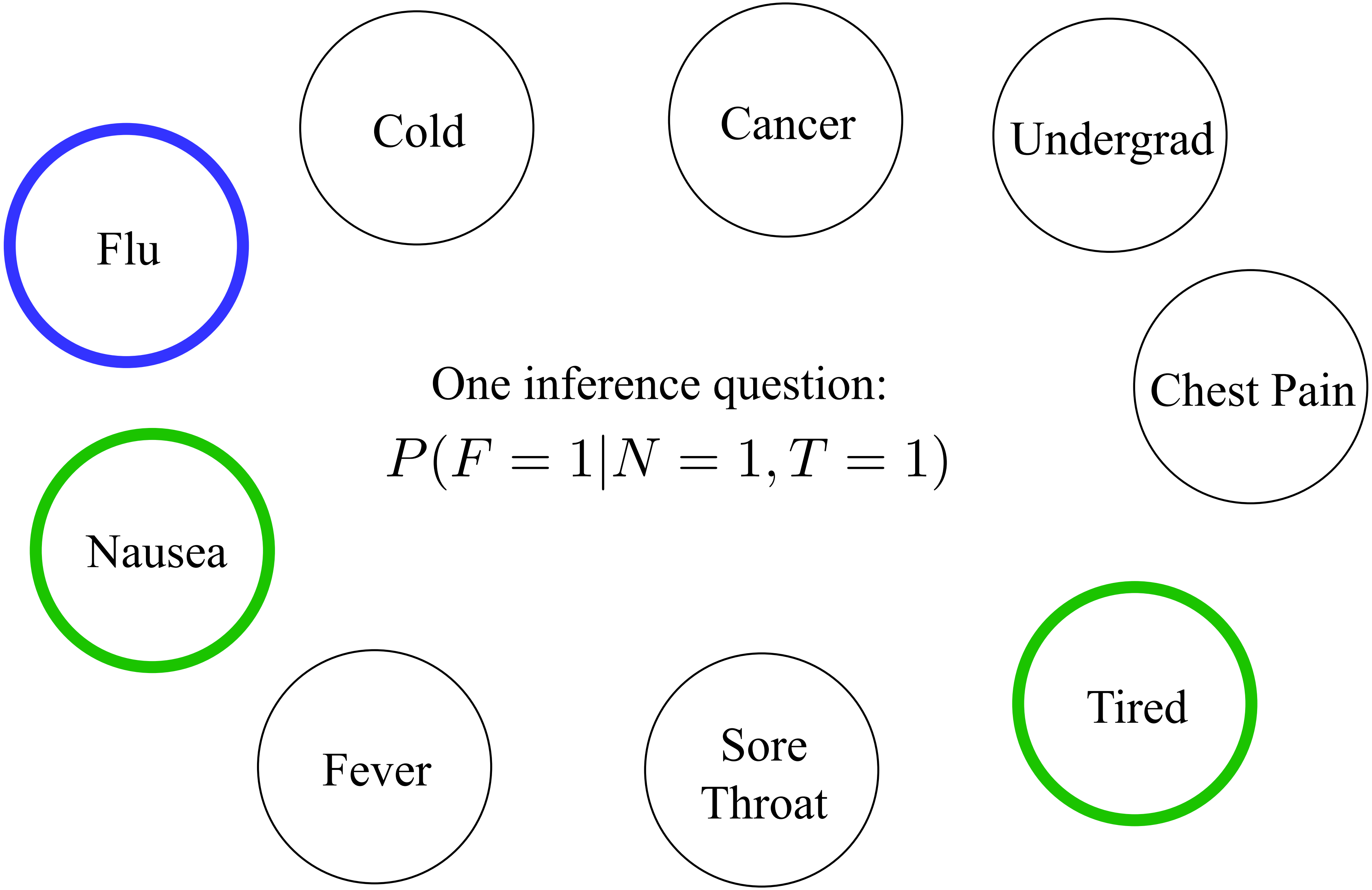
General "Inference"



General “Inference”



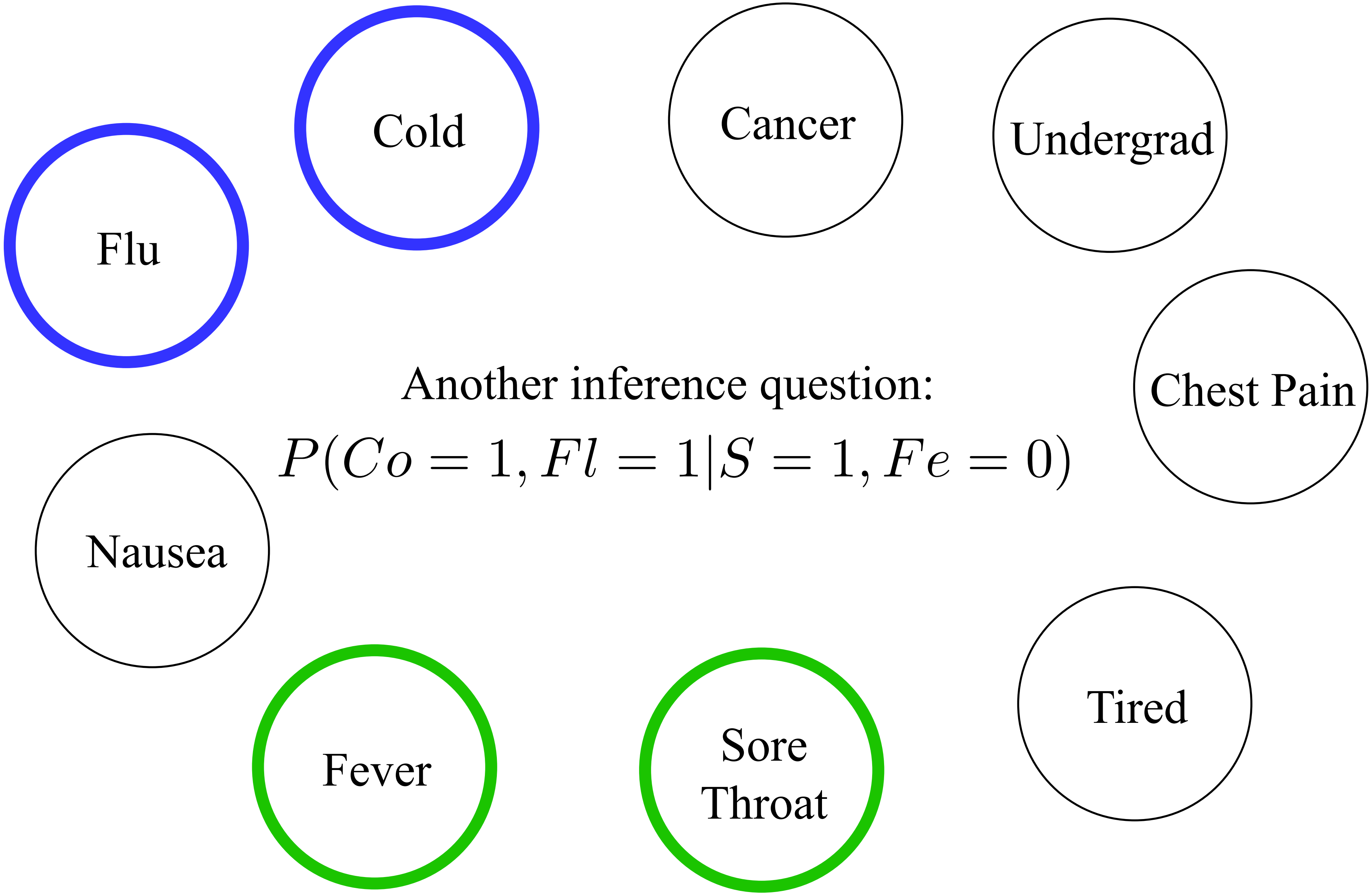
General “Inference”



One inference question:

$$P(F = 1 | N = 1, T = 1)$$

General "Inference"



Another inference question:

$$P(Co = 1, Fl = 1 | S = 1, Fe = 0)$$

How Many Things Can You Condition On?

Cold

Cancer

Undergrad

Flu

Chest Pain

N things:

$$3^N$$

Possible conditions

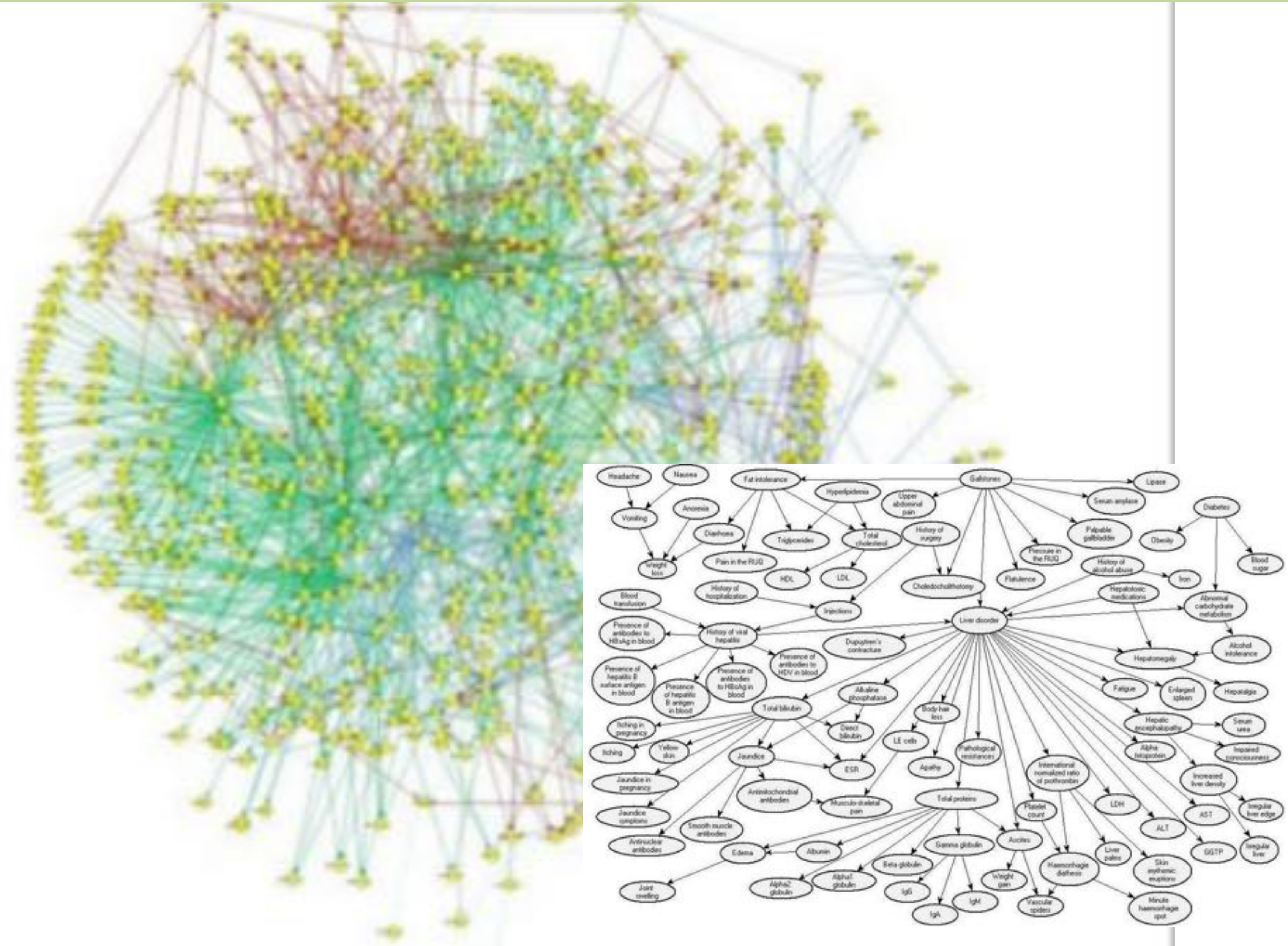
Nausea

Fever

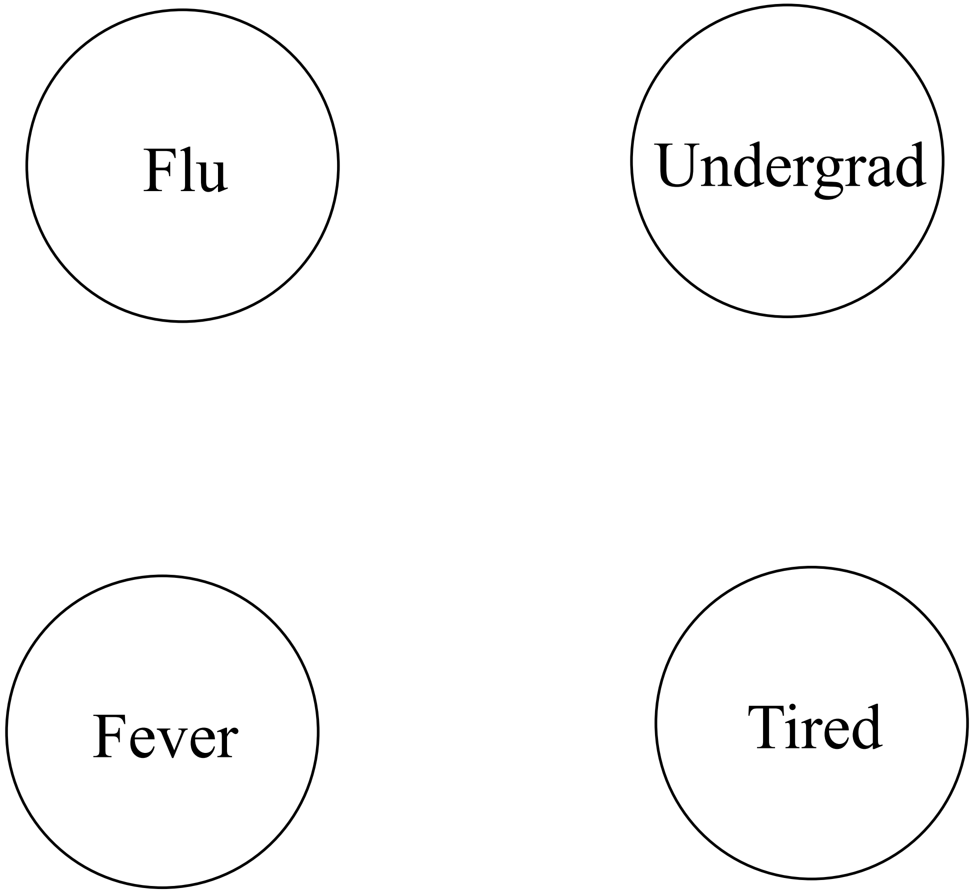
Sore Throat

Tired

N is large…


Headache
Nausea
Vomiting
Anorexia
Diarrhoea
Weight loss
Fat intolerance
Hyperlipidemia
Upper abdominal pain
Triglycerides
Total cholesterol
History of surgery
Pain in the RUQ
HDL
LDL
Choledocholithotomy
History of hospitalization
Blood transfusion
Injections
Presence of antibodies to HBsAg in blood
History of viral hepatitis
Dupuytren's contracture
Presence of antibodies to HDV in blood
Presence of hepatitis B surface antigen in blood
Presence of hepatitis B antigen in blood
Presence of antibodies to HBcAg in blood
Alkaline phosphatase
Total bilirubin
Direct bilirubin
Body hair loss
LE cells
Itching in pregnancy
Itching
Yellow skin
Jaundice
ESR
Apathy
Pathological resistances
Jaundice in pregnancy
Jaundice symptoms
Antimitochondrial antibodies
Musculo-skeletal pain
Smooth muscle antibodies
Antinuclear antibodies
Edema
Albumin
Total proteins
Gamma globulin
Beta globulin
Alpha1 globulin
Alpha2 globulin
Joint swelling
IgG
IgA
IgM
Gallstones
Lipase
Serum amylase
Diabetes
Palpable gallbladder
Obesity
Pressure in the RUQ
History of alcohol abuse
Blood sugar
Flatulence
Iron
Hepatotoxic medications
Abnormal carbohydrate metabolism
Liver disorder
Alcohol intolerance
Hepatomegaly
Fatigue
Enlarged spleen
Hepatalgia
Hepatic encephalopathy
Serum urea
Alpha fetoprotein
Impaired consciousness
International normalized ratio of prothrombin
Increased liver density
Platelet count
LDH
AST
Irregular liver edge
ALT
Ascites
Liver palms
GGTP
Irregular liver
Weight gain
Haemorrhagic diathesis
Skin erythematic eruptions
Vascular spiders
Minute haemorrhagic spot

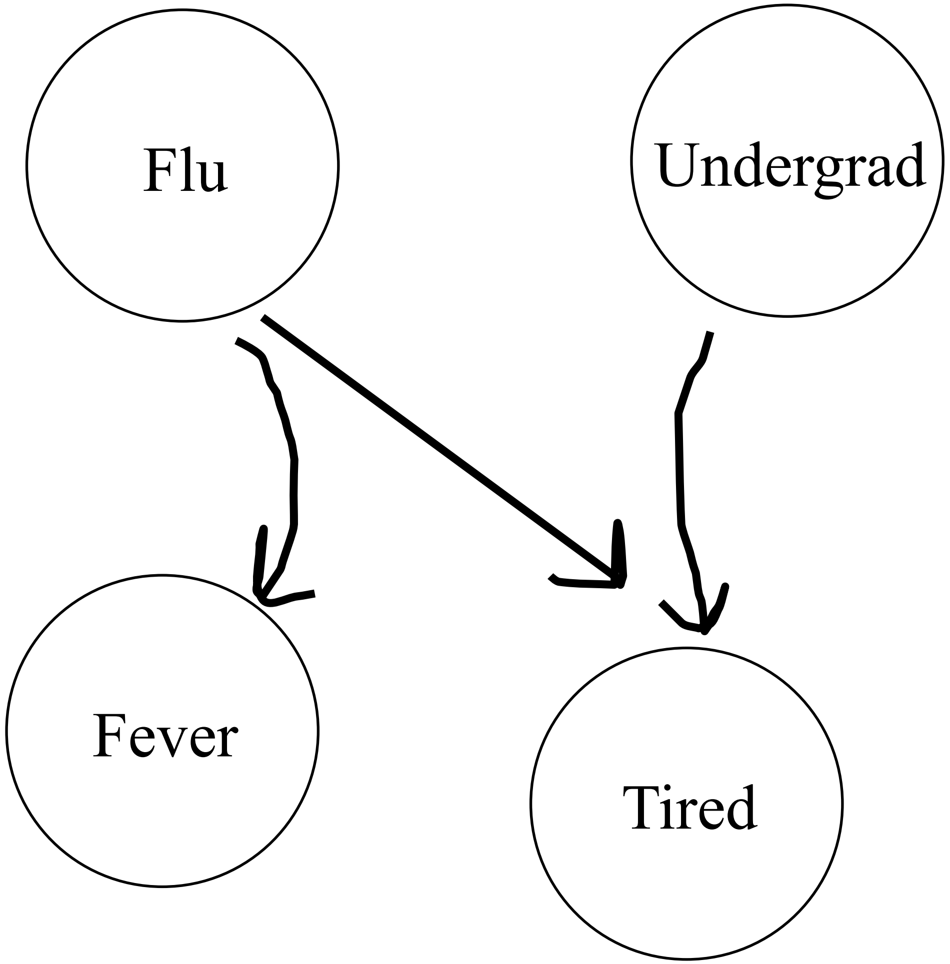
Simple WebMd



Naively specifying a joint is often impossible…

Probabilistic Model

Describe the joint using causality!



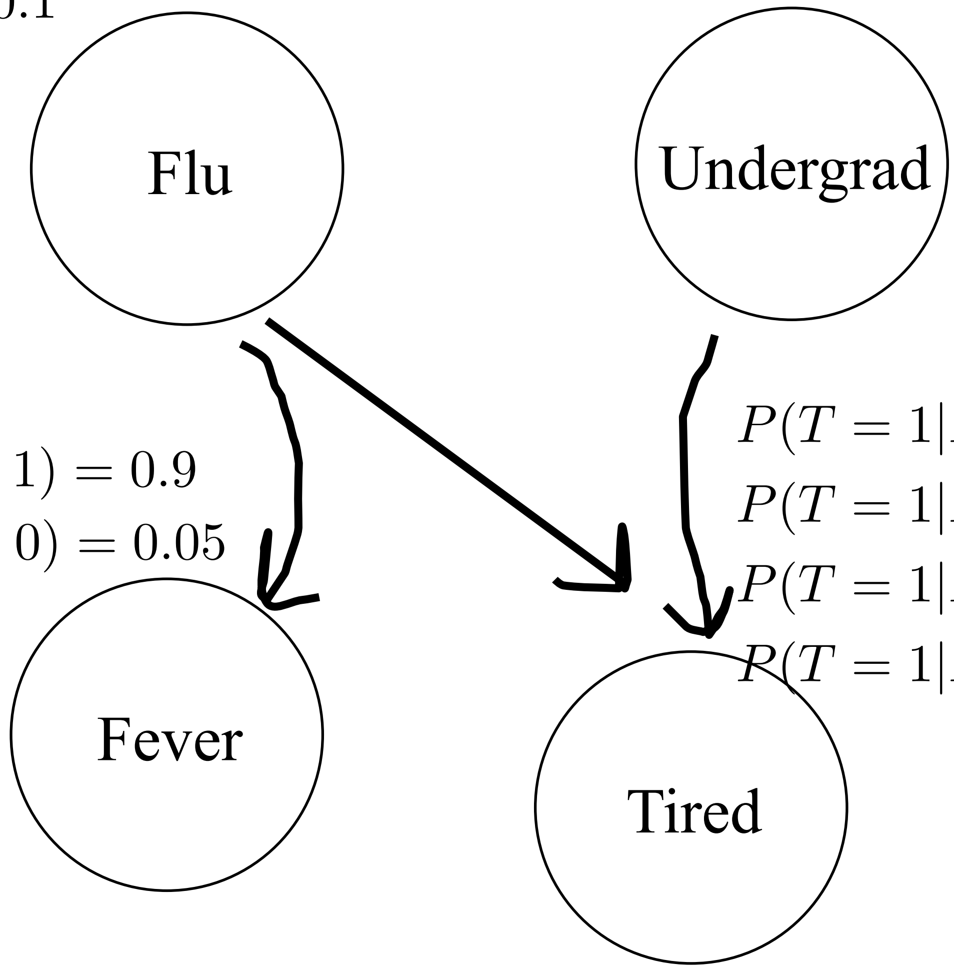
$$P(Fl = a, Fe = b, U = c, T = d)?$$

Probabilistic Model

Describe the joint using causality!

$P(Fl = 1) = 0.1$

$P(U = 1) = 0.8$



$P(Fev = 1|Flu = 1) = 0.9$

$P(Fev = 1|Flu = 0) = 0.05$

$P(T = 1|Flu = 0, U = 0) = 0.1$

$P(T = 1|Flu = 0, U = 1) = 0.8$

$P(T = 1|Flu = 1, U = 0) = 0.9$

$P(T = 1|Flu = 1, U = 1) = 1.0$

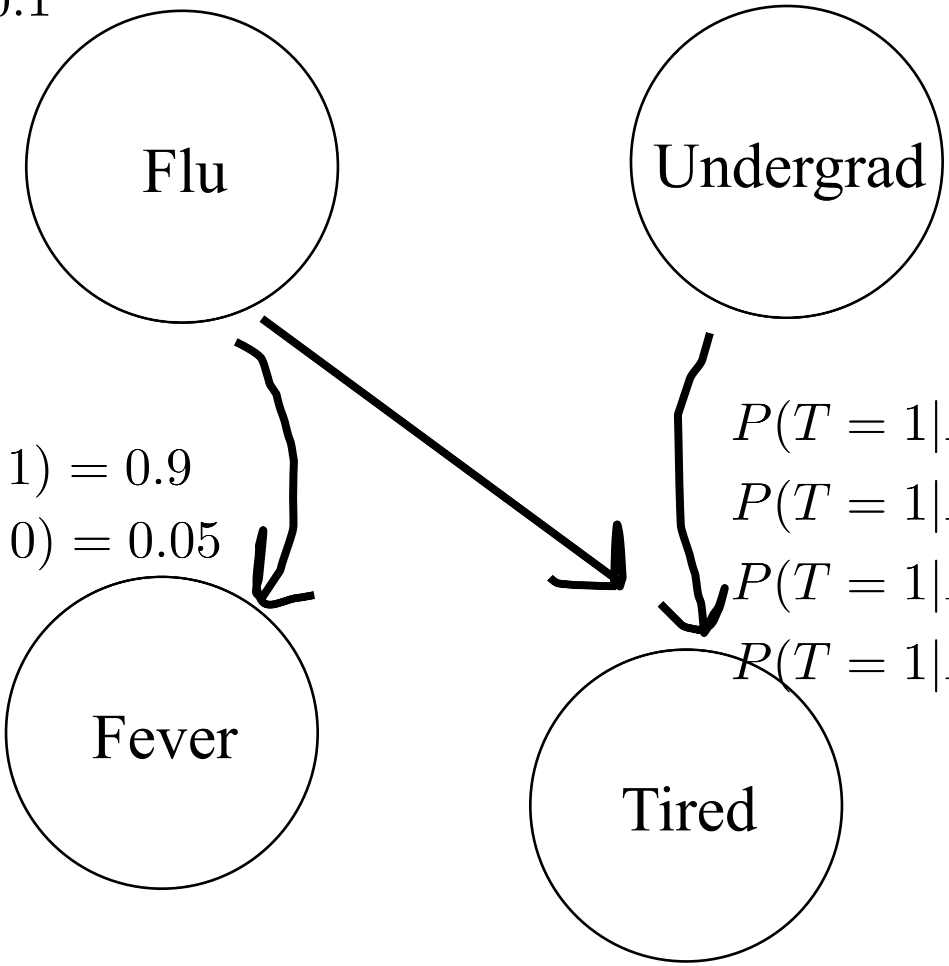
$$P(Fl = a, Fe = b, U = c, T = d)?$$

Bayesian Network

Describe the joint using causality!

$P(Fl = 1) = 0.1$

$P(U = 1) = 0.8$



$P(Fev = 1|Flu = 1) = 0.9$

$P(Fev = 1|Flu = 0) = 0.05$

$P(T = 1|Flu = 0, U = 0) = 0.1$

$P(T = 1|Flu = 0, U = 1) = 0.8$

$P(T = 1|Flu = 1, U = 0) = 0.9$

$P(T = 1|Flu = 1, U = 1) = 1.0$

$$P(Fl = a, Fe = b, U = c, T = d)?$$

Bayesian Network:

If you know causality,

Make a network of assumed direct causality for your random vars.

You simply need to give:

$P(\text{values} \mid \text{parents})$

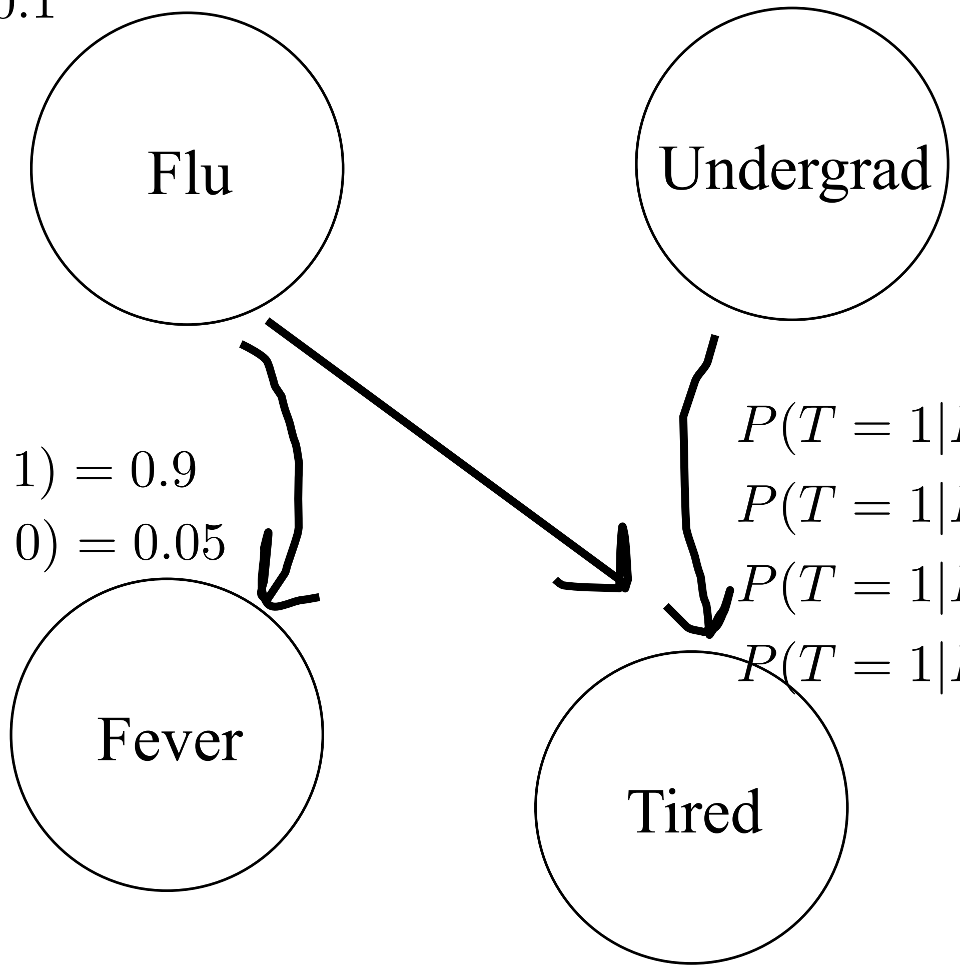
for each random variable.

Prob can be a conditional probability table or an equation!

Probabilistic Model

$P(Fl = 1) = 0.1$

$P(U = 1) = 0.8$



$P(Fev = 1 | Flu = 1) = 0.9$

$P(Fev = 1 | Flu = 0) = 0.05$

$P(T = 1 | Flu = 0, U = 0) = 0.1$

$P(T = 1 | Flu = 0, U = 1) = 0.8$

$P(T = 1 | Flu = 1, U = 0) = 0.9$

$P(T = 1 | Flu = 1, U = 1) = 1.0$

Alg #0: Straight Math

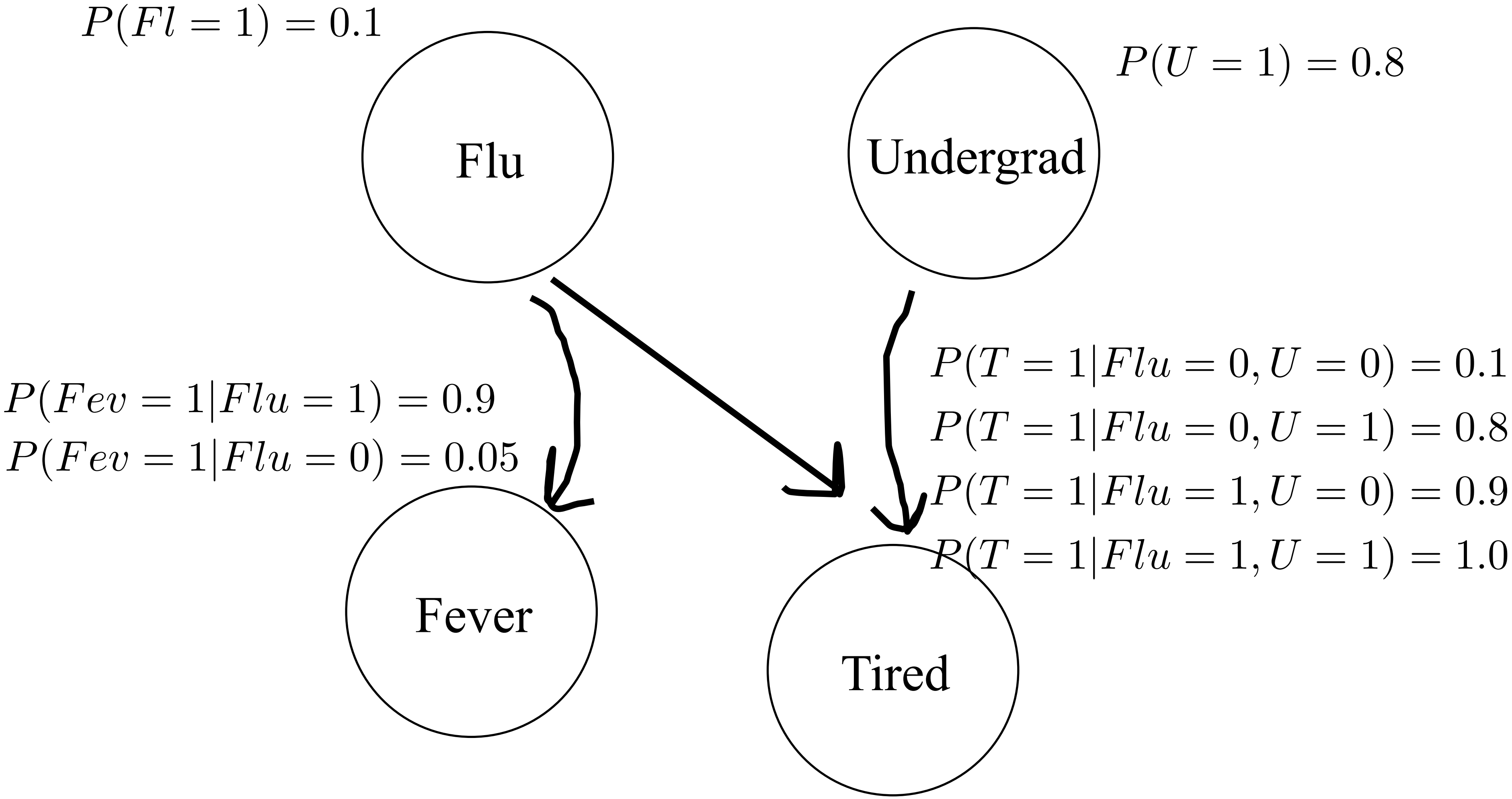
Too many possible **inference** questions one could ask…

Alg #1: Joint Sampling

```
N_SAMPLES = 100000

# Program: Joint Sample
# ---------------------
# we can answer any probability question
# with multivariate samples from the joint,
# where conditioned variables match
def main():
    obs = getObservation()
    print 'Observation = ', obs

    samples = sampleATon()
    prob = probFluGivenObs(samples, obs)
    print 'Pr(Flu) = ', prob
```

```
# Method: Sample A Ton
# ----------------------
# chose N_SAMPLES with likelhood proportional
# to the joint distribution
def sampleATon():
    samples = []
    for i in range(N_SAMPLES):
        sample = makeSample()
        samples.append(sample)
    return samples
```

Recall: Probabilistic Model



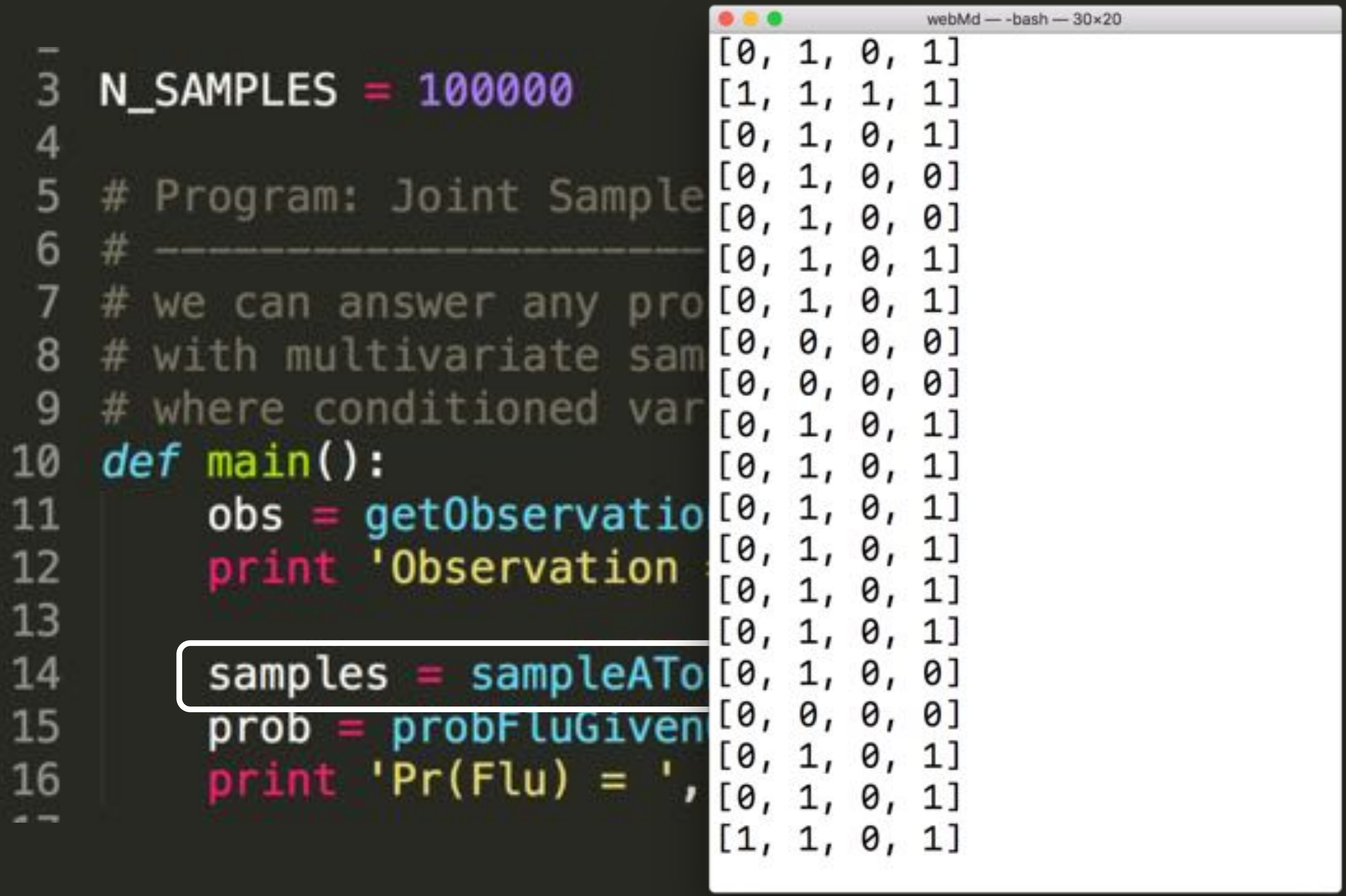
```python
# Method: Make Sample
# -------------------
# chose a single sample from the joint distribut
# based on the medical "Probabilistic Graphical
def makeSample():
    # prior on causal factors
    flu = bern(0.1)
    und = bern(0.8)

    # choose fever based on flue
    if flu == 1: fev = bern(0.9)
    else:        fev = bern(0.05)

    # choose tired based on (undergrade and flu)
    if und == 1 and flu == 1:   tir = bern(1.0)
    elif und == 1 and flu == 0: tir = bern(0.8)
    elif und == 0 and flu == 1: tir = bern(0.9)
    else:                       tir = bern(0.1)

    # a sample from the joint has an
    # assignment to *all* random variables
    return [flu, und, fev, tir]
```

```python
# Method: Make Sample
# -------------------
# chose a single sample from the joint distribut
# based on the medical "Probabilistic Graphical
def makeSample():
    # prior on causal factors
    flu = bern(0.1)
    und = bern(0.8)

    # choose fever based on flue
    if flu == 1: fev = bern(0.9)
    else:        fev = bern(0.05)

    # choose tired based on (undergrade and flu)
    if und == 1 and flu == 1:   tir = bern(1.0)
    elif und == 1 and flu == 0: tir = bern(0.8)
    elif und == 0 and flu == 1: tir = bern(0.9)
    else:                       tir = bern(0.1)

    # a sample from the joint has an
    # assignment to *all* random variables
    return [flu, und, fev, tir]
```

```python
# Method: Make Sample
# -------------------
# chose a single sample from the joint distribut
# based on the medical "Probabilistic Graphical
def makeSample():
    # prior on causal factors
    flu = bern(0.1)
    und = bern(0.8)

    # choose fever based on flue
    if flu == 1: fev = bern(0.9)
    else:        fev = bern(0.05)

    # choose tired based on (undergrade and flu)
    if und == 1 and flu == 1:   tir = bern(1.0)
    elif und == 1 and flu == 0: tir = bern(0.8)
    elif und == 0 and flu == 1: tir = bern(0.9)
    else:                       tir = bern(0.1)

    # a sample from the joint has an
    # assignment to *all* random variables
    return [flu, und, fev, tir]
```

```python
# Method: Make Sample
# -------------------
# chose a single sample from the joint distribut
# based on the medical "Probabilistic Graphical
def makeSample():
    # prior on causal factors
    flu = bern(0.1)
    und = bern(0.8)

    # choose fever based on flue
    if flu == 1: fev = bern(0.9)
    else:        fev = bern(0.05)

    # choose tired based on (undergrade and flu)
    if und == 1 and flu == 1:   tir = bern(1.0)
    elif und == 1 and flu == 0: tir = bern(0.8)
    elif und == 0 and flu == 1: tir = bern(0.9)
    else:                       tir = bern(0.1)

    # a sample from the joint has an
    # assignment to *all* random variables
    return [flu, und, fev, tir]
```

Alg #1: Joint Sampling



Alg #1: Joint Sampling

```
N_SAMPLES = 100000

# Program: Joint Sample
# ---------------------
# we can answer any probability question
# with multivariate samples from the joint,
# where conditioned variables match
def main():
    obs = getObservation()
    print 'Observation = ', obs

    samples = sampleATon()
    prob = probFluGivenObs(samples, obs)
    print 'Pr(Flu) = ', prob
```

```python
# Method: Probability of Flu Given Observation
# ------------------------------
# Calculate the probability of flu given many
# samples from the joint distribution and a set
# of ovservations to condition on.
def probFluGivenObs(samples, obs):
    # reject all samples which don't align
    # with condition
    keepSamples = []
    for sample in samples:
        if checkObsMatch(sample, obs):
            keepSamples.append(sample)

    # from remaining, simply count...
    fluCount = 0
    for sample in keepSamples:
        [flu, und, fev, tir] = sample
        if flu == 1:
            fluCount += 1

    # counting can be so sweet...
    return float(fluCount) / len(keepSamples)
```

```python
# Method: Probability of Flu Given Observation
# ------------------------------
# Calculate the probability of flu given many
# samples from the joint distribution and a set
# of ovservations to condition on.
def probFluGivenObs(samples, obs):
    # reject all samples which don't align
    # with condition
    keepSamples = []
    for sample in samples:
        if checkObsMatch(sample, obs):
            keepSamples.append(sample)

    # from remaining, simply count...
    fluCount = 0
    for sample in keepSamples:
        [flu, und, fev, tir] = sample
        if flu == 1:
            fluCount += 1

    # counting can be so sweet...
    return float(fluCount) / len(keepSamples)
```

```python
# Method: Probability of Flu Given Observation
# ---------------------------------------------
# Calculate the probability of flu given many
# samples from the joint distribution and a set
# of ovservations to condition on.
def probFluGivenObs(samples, obs):
    # reject all samples which don't align
    # with condition
    keepSamples = []
    for sample in samples:
        if checkObsMatch(sample, obs):
            keepSamples.append(sample)

    # from remaining, simply count...
    fluCount = 0
    for sample in keepSamples:
        [flu, und, fev, tir] = sample
        if flu == 1:
            fluCount += 1

    # counting can be so sweet...
    return float(fluCount) / len(keepSamples)
```

Alg #1: Joint Sampling

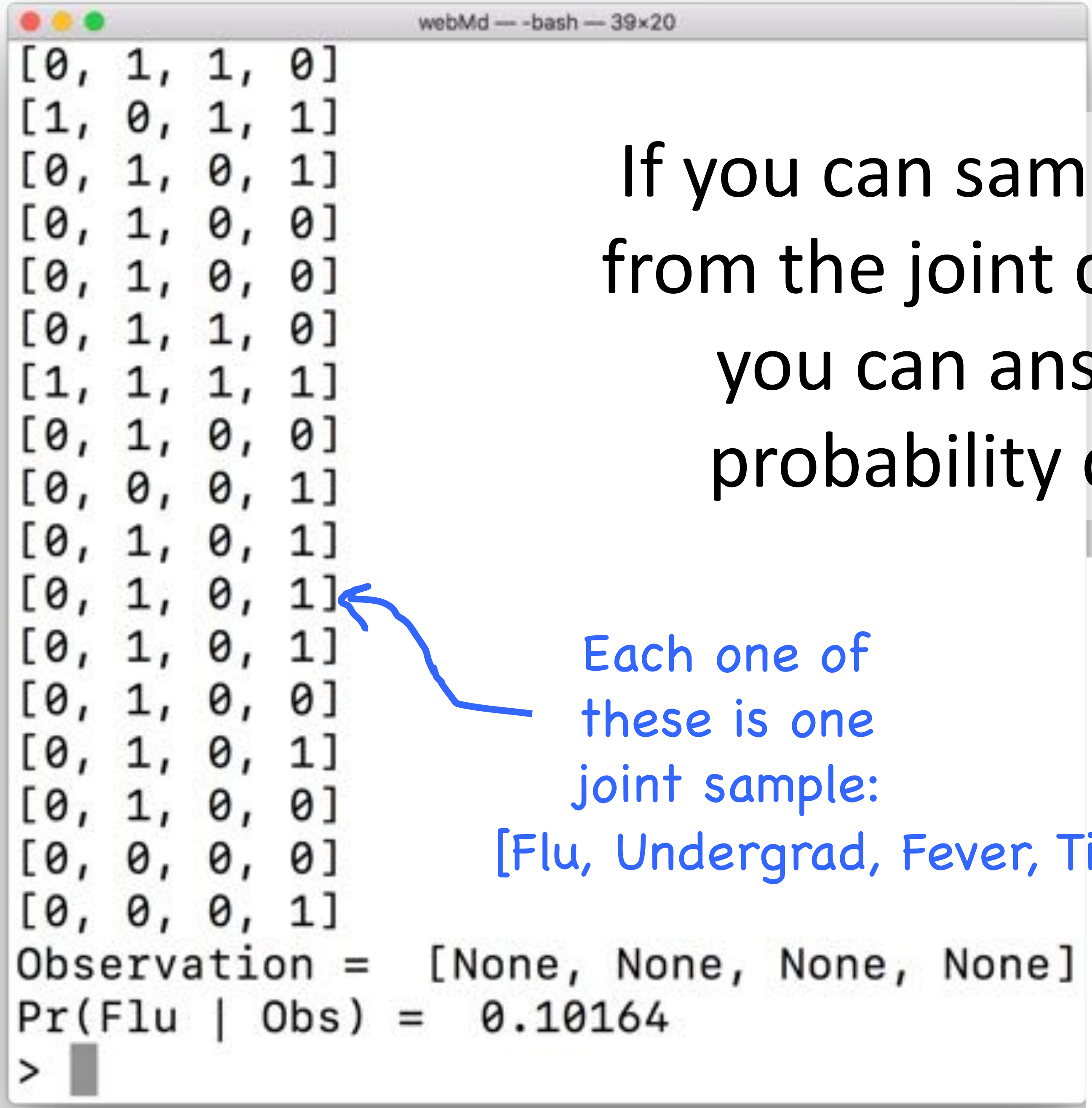
```
N_SAMPLES = 100000

# Program: Joint Sample
# -------------------------
# we can answer any pro
# with multivariate sam
# where conditioned var
def main():
    obs = getObservatio
    print 'Observation

    samples = sampleATo
    prob = probFluGiven
    print 'Pr(Flu) = ',
```

```
webMd — -bash — 30×20
[0, 1, 0, 1]
[0, 1, 0, 0]
[0, 1, 0, 0]
[0, 1, 0, 1]
[0, 1, 0, 1]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 1, 0, 1]
[0, 1, 0, 1]
[0, 1, 0, 1]
[0, 1, 0, 1]
[0, 1, 0, 1]
[0, 1, 0, 1]
[0, 1, 0, 0]
[0, 0, 0, 0]
[0, 1, 0, 1]
[0, 1, 0, 1]
[1, 1, 0, 1]
Pr(Flu) =  0.141503173687
>
```

Lets try it!

BACK TO THE CODE


The Magic School Bus™
Blast Off!



If you can sample enough from the joint distribution, you can answer any probability question

Each one of these is one joint sample:
[Flu, Undergrad, Fever, Tired]

Alg #1: Joint Sampling

With enough samples:

- Probability estimates will be correct
- Conditional probability estimates will be correct
- Expectation estimations will be correct

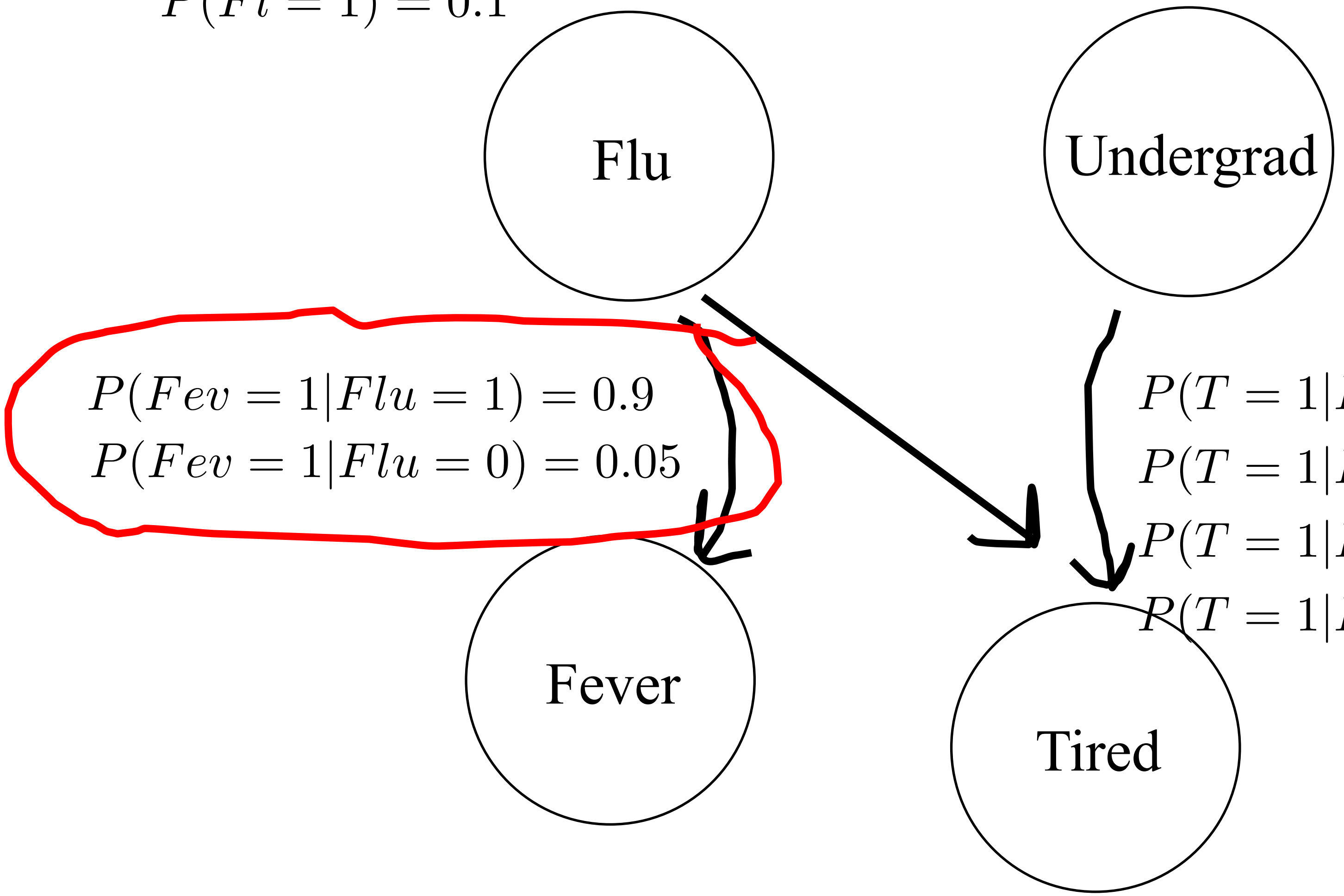
Because your samples are a representation of the joint distribution…

What’s the matter with joint sampling?

Probabilistic Model

$P(Fl = 1) = 0.1$

$P(U = 1) = 0.8$



$P(Fev = 1|Flu = 1) = 0.9$

$P(Fev = 1|Flu = 0) = 0.05$

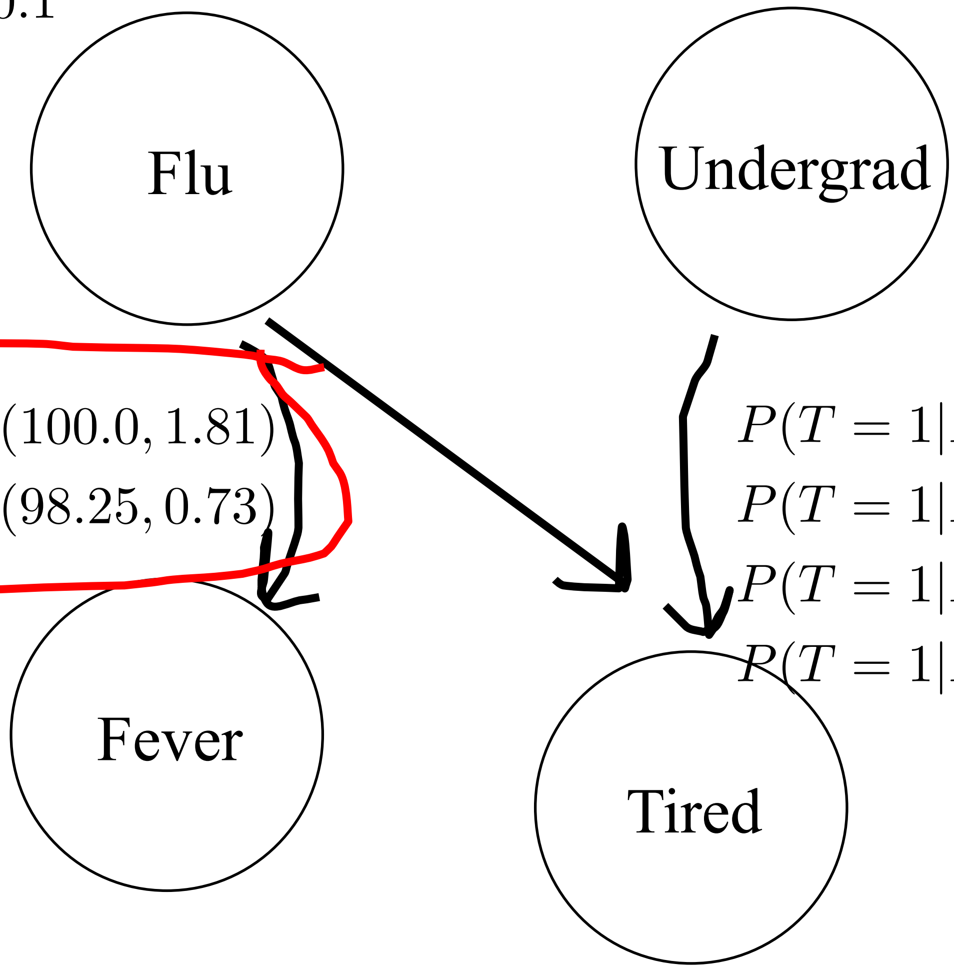
$P(T = 1|Flu = 0, U = 0) = 0.1$

$P(T = 1|Flu = 0, U = 1) = 0.8$

$P(T = 1|Flu = 1, U = 0) = 0.9$

$P(T = 1|Flu = 1, U = 1) = 1.0$

Probabilistic Model



$$P(Fl = 1) = 0.1$$

$$P(U = 1) = 0.8$$

$$Fev|Flu = 0 \sim N(100.0, 1.81)$$
$$Fev|Flu = 1 \sim N(98.25, 0.73)$$

$$P(T = 1|Flu = 0, U = 0) = 0.1$$
$$P(T = 1|Flu = 0, U = 1) = 0.8$$
$$P(T = 1|Flu = 1, U = 0) = 0.9$$
$$P(T = 1|Flu = 1, U = 1) = 1.0$$


The Magic School Bus™
Blast Off!

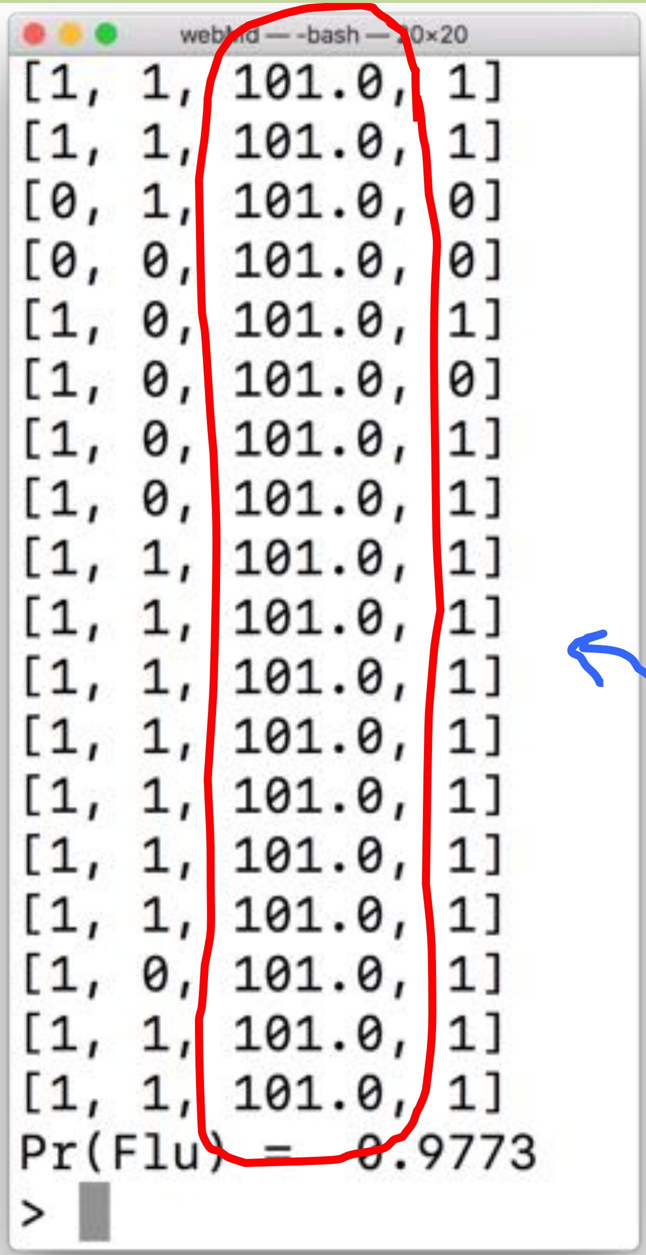
Markov Chain

MCMC

Monte Carlo

This algorithm is not tested. I just want to have a little cheeky peek into the future…

Alg #2: MCMC



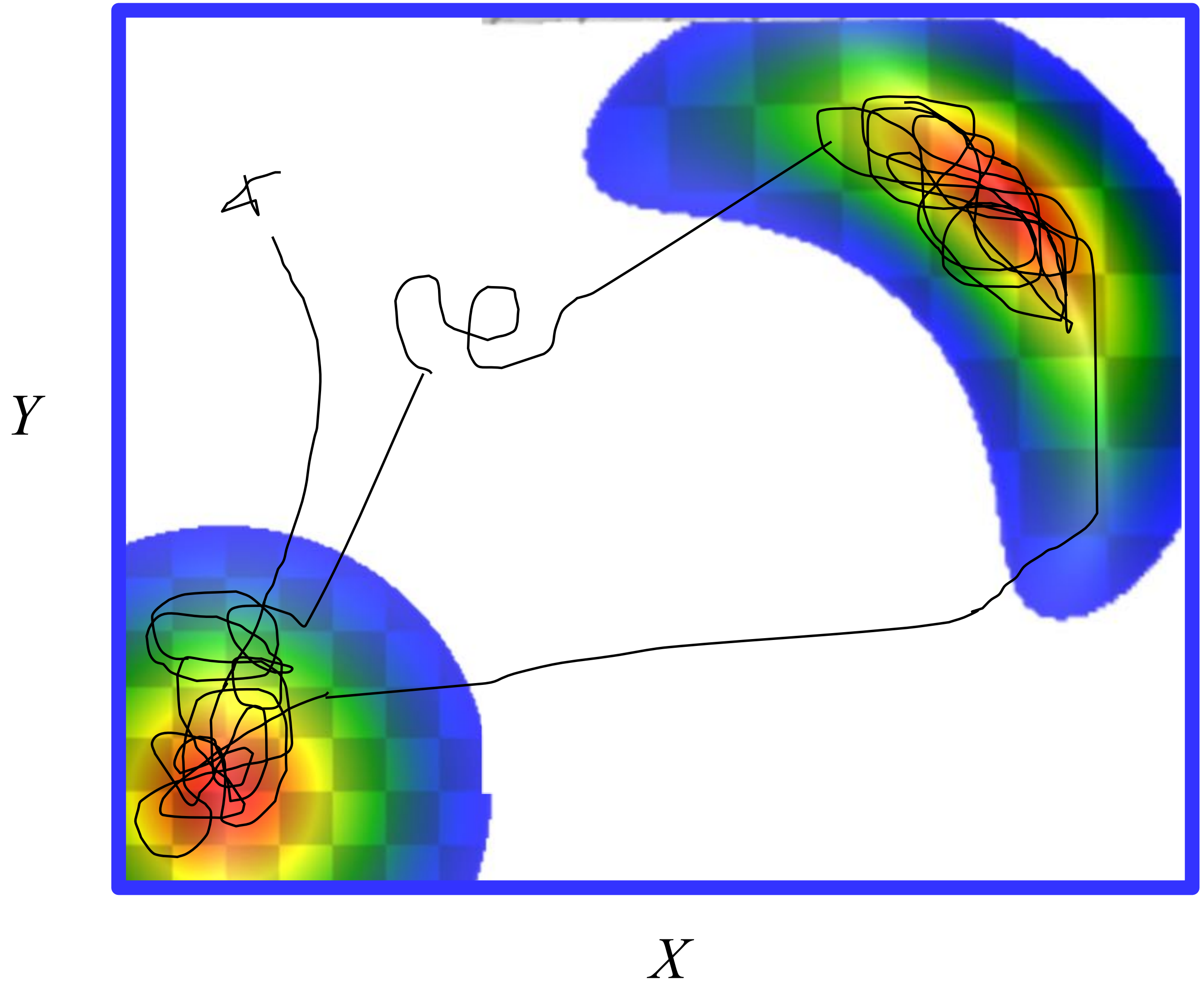
```
[1, 1, 101.0, 1]
[1, 1, 101.0, 1]
[0, 1, 101.0, 0]
[0, 0, 101.0, 0]
[1, 0, 101.0, 1]
[1, 0, 101.0, 0]
[1, 0, 101.0, 1]
[1, 0, 101.0, 1]
[1, 1, 101.0, 1]
[1, 1, 101.0, 1]
[1, 1, 101.0, 1]
[1, 1, 101.0, 1]
[1, 1, 101.0, 1]
[1, 1, 101.0, 1]
[1, 1, 101.0, 1]
[1, 0, 101.0, 1]
[1, 1, 101.0, 1]
[1, 1, 101.0, 1]
Pr(Flu) = 0.9773
>
```

MCMC is a way to sample with conditioned variables fixed

Each one of these is one posterior sample:

[Flu, Undergrad, Fever, Tired]

Alg #2: MCMC



Alg #2: MCMC

```
All Samples = []
```

Flu, Undergrad, Fever, Tired

```
Initial Sample = [0, 0, 101.0, 1]
```

Alg #2: MCMC

```
All Samples = []
```

Flu, Undergrad, Fever, Tired

$S^{(0)} = [0, 0, 101.0, 1]$

Alg #2: MCMC

All Samples = [$S^{(0)}$]

Flu, Undergrad, Fever, Tired

$S^{(0)} = [0,\ 0,\ 101.0,\ 1]$

From S_t make S_{t+1}

Alg #2: MCMC

```
All Samples = [S(0)]
```

Flu Undergrad Fever Tired

```
S(1) = [0, 0, 101.0, 1]
```

$$P(Flu = 1 | \text{All others})$$

$$= P(Flu = 1 | Und = 0, Fev = 98.3, Tir = 1)$$

$$= 0.21$$

$$Flu_1 = \text{Sample}\Big[P(Flu = 1 | \text{All others})\Big]$$

Alg #2: MCMC

```
All Samples = [S⁽⁰⁾]
```

Flu, Undergrad, Fever, Tired

$S^{(1)} = [1, 0, 101.0, 1]$

$$P(Flu = 1 | \text{All others})$$
$$= P(Flu = 1 | Und = 0, Fev = 98.3, Tir = 1)$$
$$= 0.21$$

$$Flu_1 = \text{Sample}\Big[P(Flu = 1 | \text{All others})\Big]$$

Alg #2: MCMC

```
All Samples = [S(0)]
```

```
S(1) = [1, 0, 101.0, 1]
```

$$P(Und = 1 | \text{All others})$$

$$= P(Und = 1 | Flu = 1, Fev = 98.3, Tir = 1)$$

$$= 0.91$$

$$Und_1 = \text{Sample}\Big[P(Und = 1 | \text{All others})\Big]$$

Alg #2: MCMC

```
All Samples = [S(0)]
```

Flu Undergrad Fever Tired

$S^{(1)} = [1, 1, 101.0, 1]$

$$P(Und = 1|\text{All others})$$
$$= P(Und = 1|Flu = 1, Fev = 98.3, Tir = 1)$$
$$= 0.91$$

$$Und_1 = \text{Sample}\Big[P(Und = 1|\text{All others})\Big]$$

Alg #2: MCMC

All Samples = [$S^{(0)}$]

Flu, Undergrad, Fever, Tired

$S^{(1)}$ = [1, 1, 101.0, 1]

Let's say you are conditioning on fever being 101.0...
then don't change that value

Alg #2: MCMC

All Samples = $[S^{(0)}]$

Flu Undergrad Fever Tired

$S^{(1)} = [1, 1, 101.0, 1]$

Alg #2: MCMC

All Samples = $[S^{(0)}]$

Flu
Undergrad
Fever
Tired

$S^{(1)} = [1,\ 1,\ 101.0,\ 1]$

Alg #2: MCMC

```
All Samples = [S(0)]
```

Flu, Undergrad, Fever, Tired

$S^{(1)} = [1, 1, 101.0, 1]$

Alg #2: MCMC

All Samples = $[S^{(0)}, S^{(1)}]$

Flu Undergrad Fever Tired

$S^{(1)} = [1,\ 1,\ 101.0,\ 1]$

Alg #2: MCMC

```
All Samples = [S(0), S(1)]
```

Flu Undergrad Fever Tired

```
S(2) = [1, 1, 101.0, 1]
```

$$P(Flu = 1 | \text{All others})$$

$$Flu_1 = \text{Sample}\Big[P(Flu = 1 | \text{All others})\Big]$$

Alg #2: MCMC

```
All Samples = [S(0), S(1)]
```

Flu, Undergrad, Fever, Tired

$S^{(2)}$ = [1, 1, 101.0, 1]

$$P(Flu = 1 | \text{All others})$$

$$Flu_1 = \text{Sample}\Big[P(Flu = 1 | \text{All others})\Big]$$

Alg #2: MCMC

```
All Samples = [S⁽⁰⁾, S⁽¹⁾]
```

Flu Undergrad Fever Tired

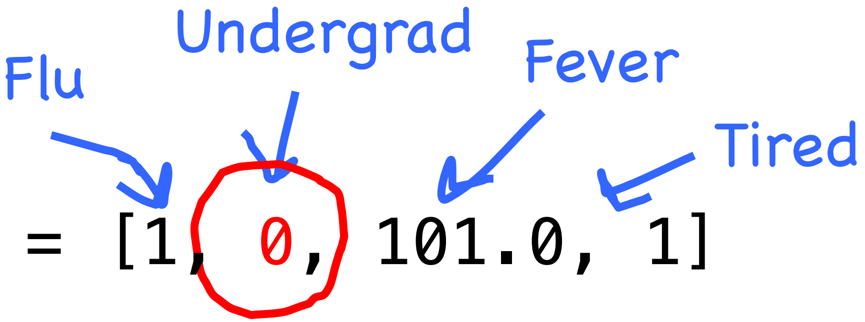
$S^{(2)}$ = [1, 1, 101.0, 1]

$$P(Flu = 1 | \text{All others})$$

$$Flu_1 = \text{Sample}\Big[P(Flu = 1 | \text{All others})\Big]$$

Alg #2: MCMC

All Samples = $[S^{(0)}, S^{(1)}]$



$S^{(2)} = [1, 0, 101.0, 1]$

Alg #2: MCMC

All Samples = [$S^{(0)}$, $S^{(1)}$]

Flu Undergrad Fever Tired

$S^{(2)}$ = [1, 0, 101.0, 1]

Alg #2: MCMC

All Samples = $[S^{(0)}, S^{(1)}]$

Flu, Undergrad, Fever, Tired

$S^{(2)} = [1, 0, 101.0, 1]$

Alg #2: MCMC

$\text{All Samples} = [S^{(0)}, S^{(1)}]$

Flu

Undergrad

Fever

Tired

$S^{(2)} = [1, 0, 101.0, 1]$

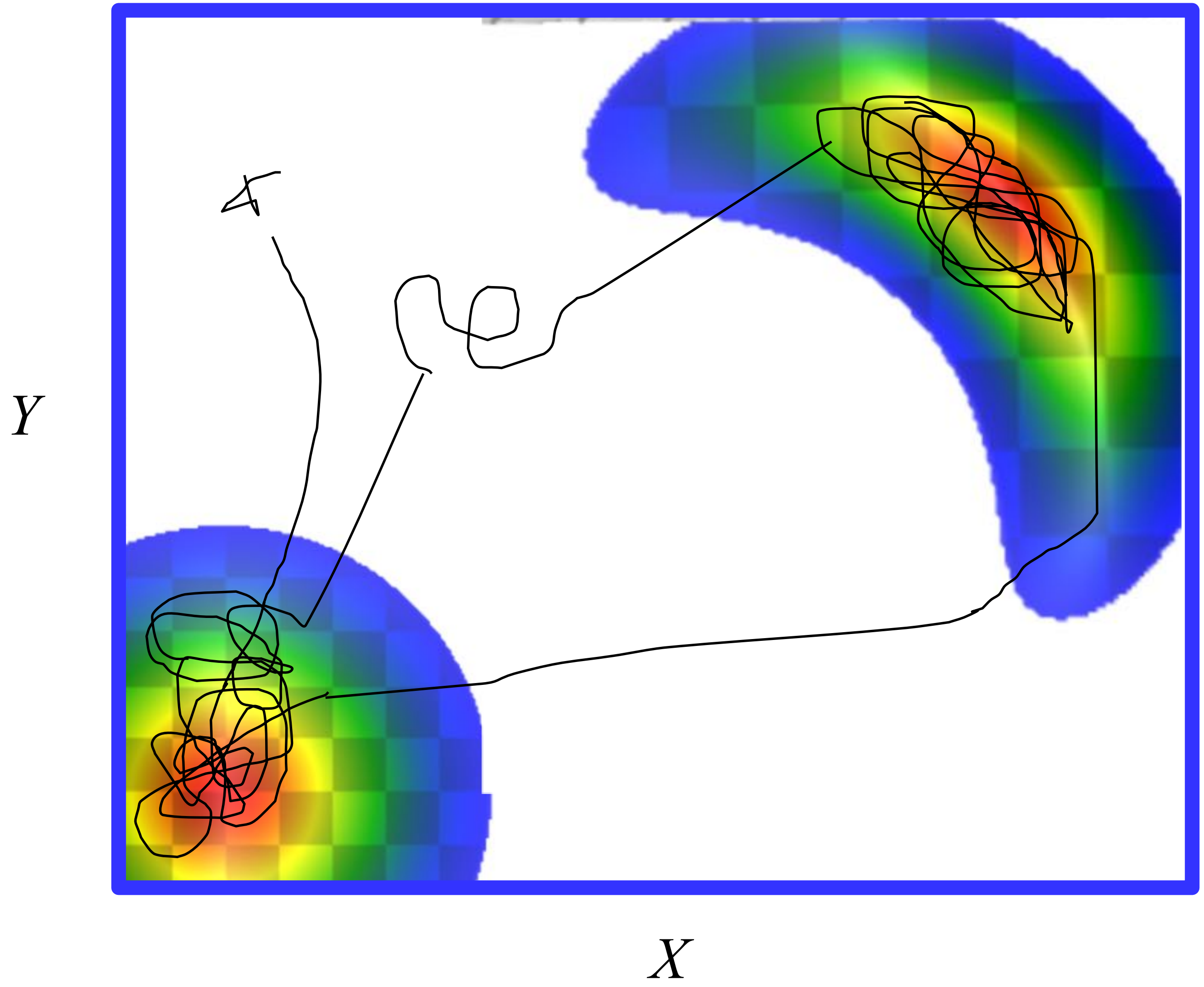
Alg #2: MCMC

All Samples = [$S^{(0)}$, $S^{(1)}$, $S^{(2)}$]

Flu Undergrad Fever Tired

$S^{(2)}$ = [1, 0, 101.0, 1]

Alg #2: MCMC



BAE's Theorem?

$$P(A \mid B\, E) = \frac{P(B \mid A\, E)\, P(A \mid E)}{P(B \mid E)}$$

$P(F = 1 \mid \text{all other rvs})$

Know: P(symptom | flu, undergrad) P(flu) P(undergrad)

Flu is independent of undergrad

Tired and fever are conditionally independent given flu, undergrad

$$P(F = 1 | \text{symptoms}, U = u)$$

$$= \frac{P(\text{symptoms} | F = 1, U = u) P(F = 1 | U = u)}{P(\text{symptoms} | U = u)}$$

$$\propto P(\text{symptoms} | F = 1, U = u) P(F = 1 | U = u)$$

$$\propto P(F = 1) P(\text{symptoms} | F = 1, U = u)$$

$$\propto P(F = 1) \prod_i P(\text{symptom}_i | F = 1, U = u)$$

$$P(F=1|\text{all other rvs}) \propto P(F=1)\prod_i P(\text{symptom}_i|F=1,U=u)$$

```python
def sampleFlu(sample):
    f1 = getFluPr1(sample)
    f0 = getFluPr0(sample)
    p1 = f1 / (f1 + f0)
    return bern(p1)

def getFluPr0(sample):
    [_, und, fev, tir] = sample
    pFlu0 = 0.9
    pFev = getPrFeverX(fev, flu=0)
    pTir = getPrTiredX(tir, und=und, flu=0)
    return pFlu0 * pFev * pTir

def getFluPr1(sample):
    [_, und, fev, tir] = sample
    pFlu1 = 0.1
    pFev = getPrFeverX(fev, flu=1)
    pTir = getPrTiredX(tir, und=und, flu=1)
    return pFlu1 * pFev * pTir
```

$$P(F = 1 | \text{all other rvs}) \propto P(F = 1) \prod_i P(\text{symptom}_i | F = 1, U = u)$$

```python
def sampleFlu(sample):
    f1 = getFluPr1(sample)
    f0 = getFluPr0(sample)
    p1 = f1 / (f1 + f0)
    return bern(p1)

def getFluPr0(sample):
    [_, und, fev, tir] = sample
    pFlu0 = 0.9
    pFev = getPrFeverX(fev, flu=0)
    pTir = getPrTiredX(tir, und=und, flu=0)
    return pFlu0 * pFev * pTir

def getFluPr1(sample):
    [_, und, fev, tir] = sample
    pFlu1 = 0.1
    pFev = getPrFeverX(fev, flu=1)
    pTir = getPrTiredX(tir, und=und, flu=1)
    return pFlu1 * pFev * pTir
```

$$P(F=1|\text{all other rvs}) \propto P(F=1)\prod_i P(\text{symptom}_i|F=1, U=u)$$

```python
def sampleFlu(sample):
    f1 = getFluPr1(sample)
    f0 = getFluPr0(sample)
    p1 = f1 / (f1 + f0)
    return bern(p1)

def getFluPr0(sample):
    [_, und, fev, tir] = sample
    pFlu0 = 0.9
    pFev = getPrFeverX(fev, flu=0)
    pTir = getPrTiredX(tir, und=und, flu=0)
    return pFlu0 * pFev * pTir

def getFluPr1(sample):
    [_, und, fev, tir] = sample
    pFlu1 = 0.1
    pFev = getPrFeverX(fev, flu=1)
    pTir = getPrTiredX(tir, und=und, flu=1)
    return pFlu1 * pFev * pTir
```

$$P(F = 1 | \text{all other rvs}) \propto P(F = 1) \prod_i P(\text{symptom}_i | F = 1, U = u)$$

```
def sampleFlu(sample):
    f1 = getFluPr1(sample)
    f0 = getFluPr0(sample)
    p1 = f1 / (f1 + f0)
    return bern(p1)

def getFluPr0(sample):
    [_, und, fev, tir] = sample
    pFlu0 = 0.9
    pFev = getPrFeverX(fev, flu=0)
    pTir = getPrTiredX(tir, und=und, flu=0)
    return pFlu0 * pFev * pTir

def getFluPr1(sample):
    [_, und, fev, tir] = sample
    pFlu1 = 0.1
    pFev = getPrFeverX(fev, flu=1)
    pTir = getPrTiredX(tir, und=und, flu=1)
    return pFlu1 * pFev * pTir
```

$$P(F = 1 | \text{all other rvs}) \propto P(F = 1) \prod_i P(\text{symptom}_i | F = 1, U = u)$$

```python
def sampleFlu(sample):
    f1 = getFluPr1(sample)
    f0 = getFluPr0(sample)
    p1 = f1 / (f1 + f0)
    return bern(p1)

def getFluPr0(sample):
    [_, und, fev, tir] = sample
    pFlu0 = 0.9
    pFev = getPrFeverX(fev, flu=0)
    pTir = getPrTiredX(tir, und=und, flu=0)
    return pFlu0 * pFev * pTir

def getFluPr1(sample):
    [_, und, fev, tir] = sample
    pFlu1 = 0.1
    pFev = getPrFeverX(fev, flu=1)
    pTir = getPrTiredX(tir, und=und, flu=1)
    return pFlu1 * pFev * pTir
```

$$P(F = 1 | \text{all other rvs}) \propto P(F = 1) \prod_i P(\text{symptom}_i | F = 1, U = u)$$

```python
def sampleFlu(sample):
    f1 = getFluPr1(sample)
    f0 = getFluPr0(sample)
    p1 = f1 / (f1 + f0)
    return bern(p1)

def getFluPr0(sample):
    [_, und, fev, tir] = sample
    pFlu0 = 0.9
    pFev = getPrFeverX(fev, flu=0)
    pTir = getPrTiredX(tir, und=und, flu=0)
    return pFlu0 * pFev * pTir

def getFluPr1(sample):
    [_, und, fev, tir] = sample
    pFlu1 = 0.1
    pFev = getPrFeverX(fev, flu=1)
    pTir = getPrTiredX(tir, und=und, flu=1)
    return pFlu1 * pFev * pTir
```

See you soon!

Summary

General Inference Summary

- **Straight Math** is fast, but can be prohibitively hard for complex models (see hw).
- **Joint Sampling** is really easy to program but fails for continuous variables (and when what you are conditioning on is rare)
- **MCMC** works well when conditioning on rare events, but is *much* harder to code / derive.
- All sampling is **slow.**