Computation & Cognition Lab

ngoodman@stanford.edu

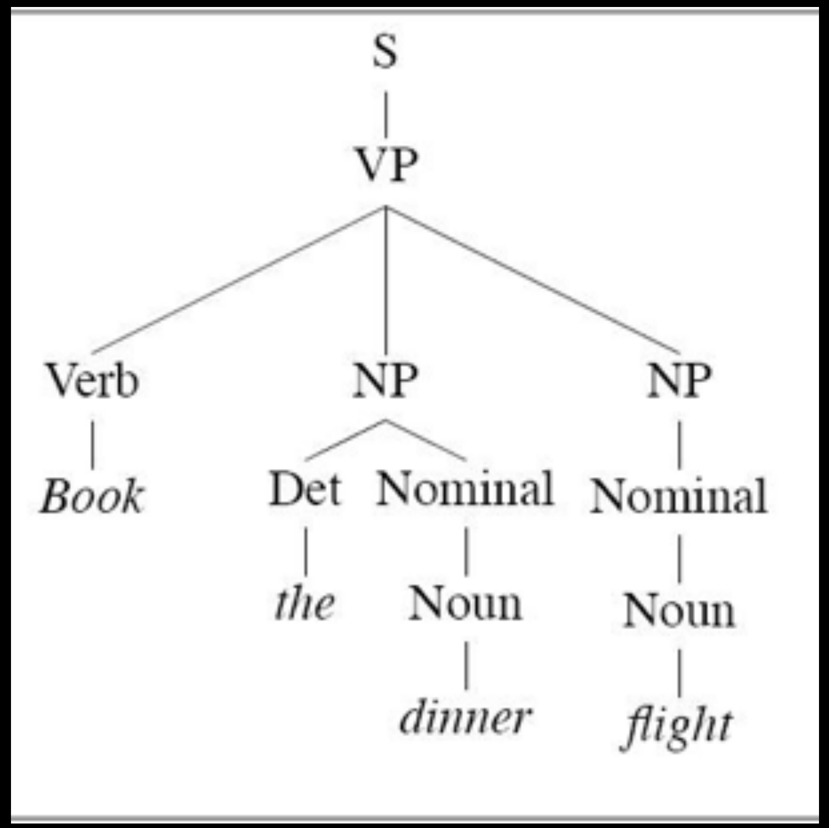
# Probabilistic programming languages

Noah D. Goodman
Stanford University

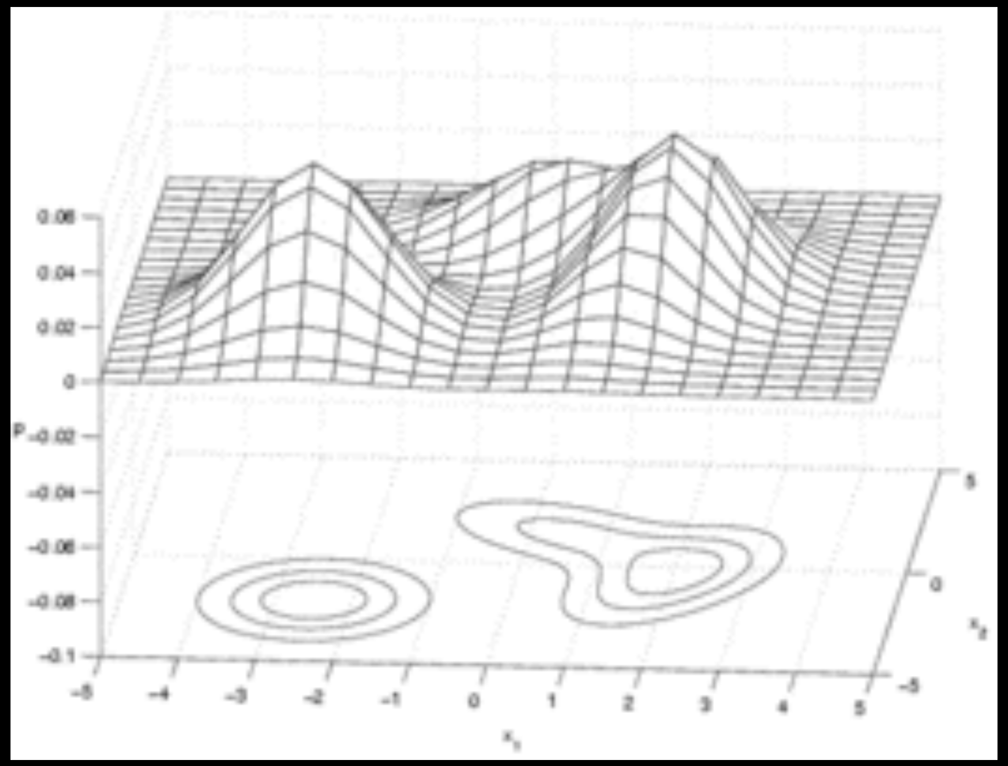
Microsoft Faculty Summit
July 9, 2015

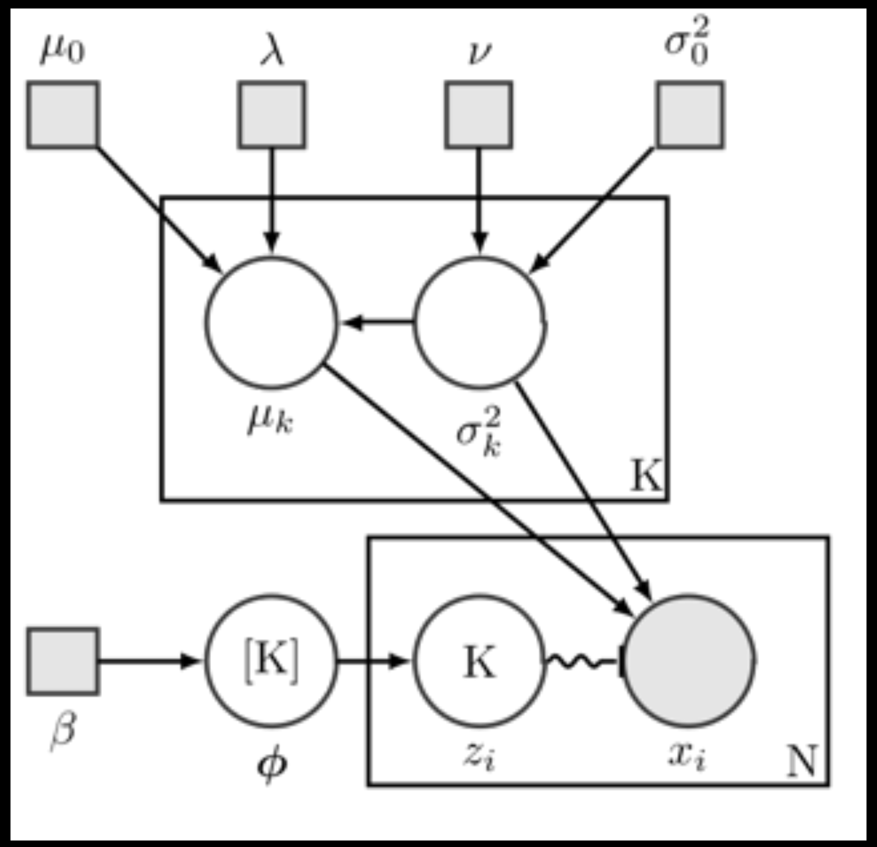
# Probabilistic Models

$$P(H|d)$$




| | |
|---|---|
| 1. $\mathfrak{S} \rightarrow$ S | 1.0 |
| 2. S $\rightarrow$ NP V | 0.7 |
| 3. S $\rightarrow$ NP | 0.3 |
| 4. NP $\rightarrow$ N | 0.8 |
| 5. NP $\rightarrow$ N N | 0.2 |
| 6. N $\rightarrow$ Fido | 0.6 |
| 7. N $\rightarrow$ runs | 0.4 |
| 8. V $\rightarrow$ runs | 1.0 |






$K$ = number of mixture components
$N$ = number of observations
$\theta_{i=1\ldots K}$ = parameter of distribution of observation associated with component $i$
$\phi_{i=1\ldots K}$ = mixture weight, i.e., prior probability of a particular component $i$
$\boldsymbol{\phi}$ = $K$-dimensional vector composed of all the individual $\phi_{1\ldots K}$; must sum to 1
$z_{i=1\ldots N}$ = component of observation $i$
$x_{i=1\ldots N}$ = observation $i$
$F(x|\theta)$ = probability distribution of an observation, parametrized on $\theta$
$z_{i=1\ldots N}$ ~ Categorical($\boldsymbol{\phi}$)
$x_{i=1\ldots N}$ ~ $F(\theta_{z_i})$

- A powerful representation of uncertain knowledge and reasoning.
- Specification is a heterogeneous mess of math, english, dependence diagrams, etc.

# Programming languages

$$\lambda$$

- Uniform, universal specification of process, with high-level abstractions.
- No intrinsic ability to represent and reason about uncertainty.

# Probabilistic programs

$$P(H|d)$$

+

$$\lambda$$

# Probabilistic programming languages

- Build a formal language for describing probabilistic models starting from a universal programming language.
- Probabilistic programming language =
  - Deterministic language +
  - primitive distributions (ERPs) +
  - `sample` and `factor` operators +
  - marginal inference operators.

# Probabilistic programming languages

**webppl** *probabilistic programming for the web*

On Github

webppl is a small but feature-rich probabilistic programming language embedded in Javascript.

```
print(
Enumerate(
  function(){
    var a = sample(bernoulliERP, [0.3])
    var b = sample(bernoulliERP, [0.1])
    factor(a|b ? 0 : -100)
    return a & b
  })
)
```

run


webppl.org

# Probabilistic programming languages

- See also:
  - Church, IBAL, Figaro, Venture, Hansei, Anglican, Fun, etc.
  - Infer.net, MLNs, BLOG, JAGS, Stan, Factorie, etc.

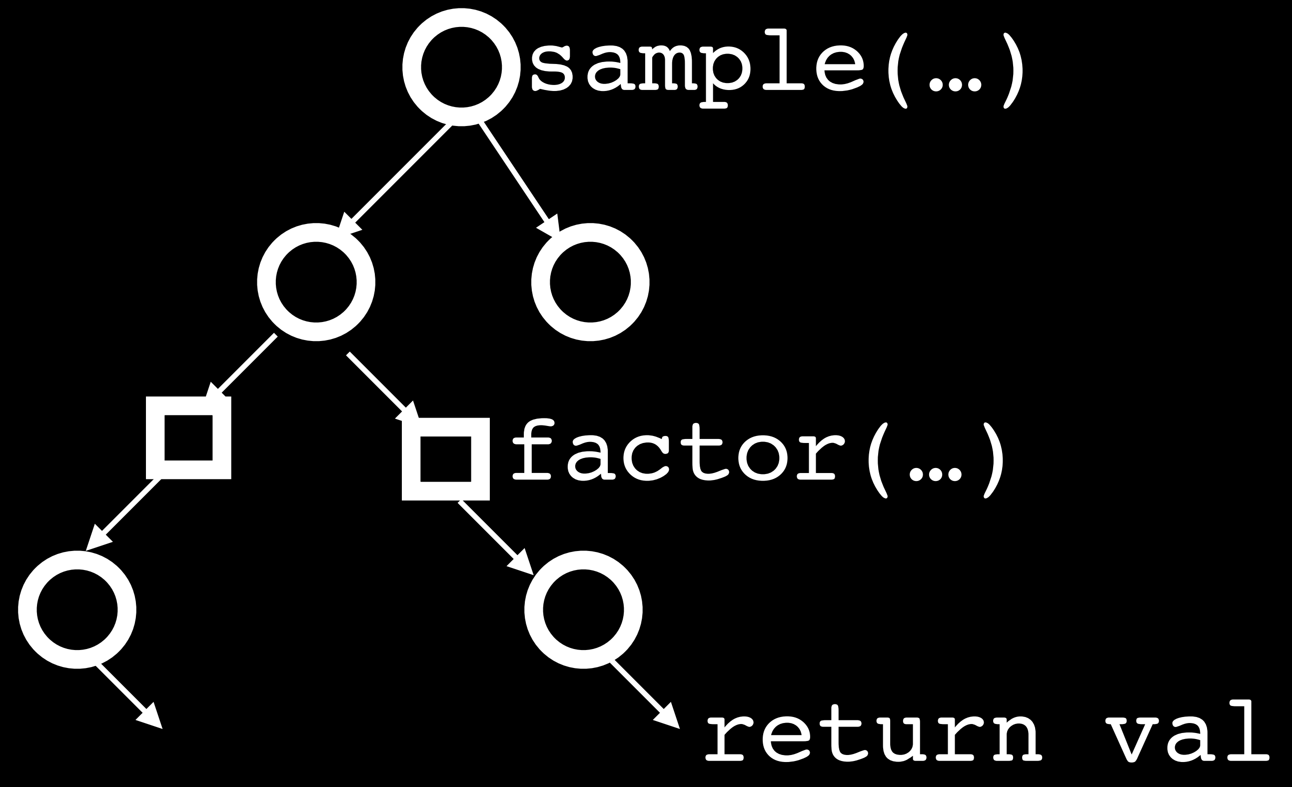
# Probabilistic programming languages

- Deterministic language: a (purely functional) subset of Javascript
- primitive distributions: ERP objects can sample, score, etc.
- `sample` operator: draw random sample from an ERP
- `factor` operator: re-weight an execution (to encode observations, etc)
- marginal inference operators:…

```
print(
Enumerate(
  function(){
    var a = sample(bernoulliERP, [0.3])
    var b = sample(bernoulliERP, [0.1])
    factor(a|b ? 0 : -100)
    return a & b
  })
)
```

# Marginal inference

```
var foo = function() {…; return val}
var erp = Marginal(foo)
```



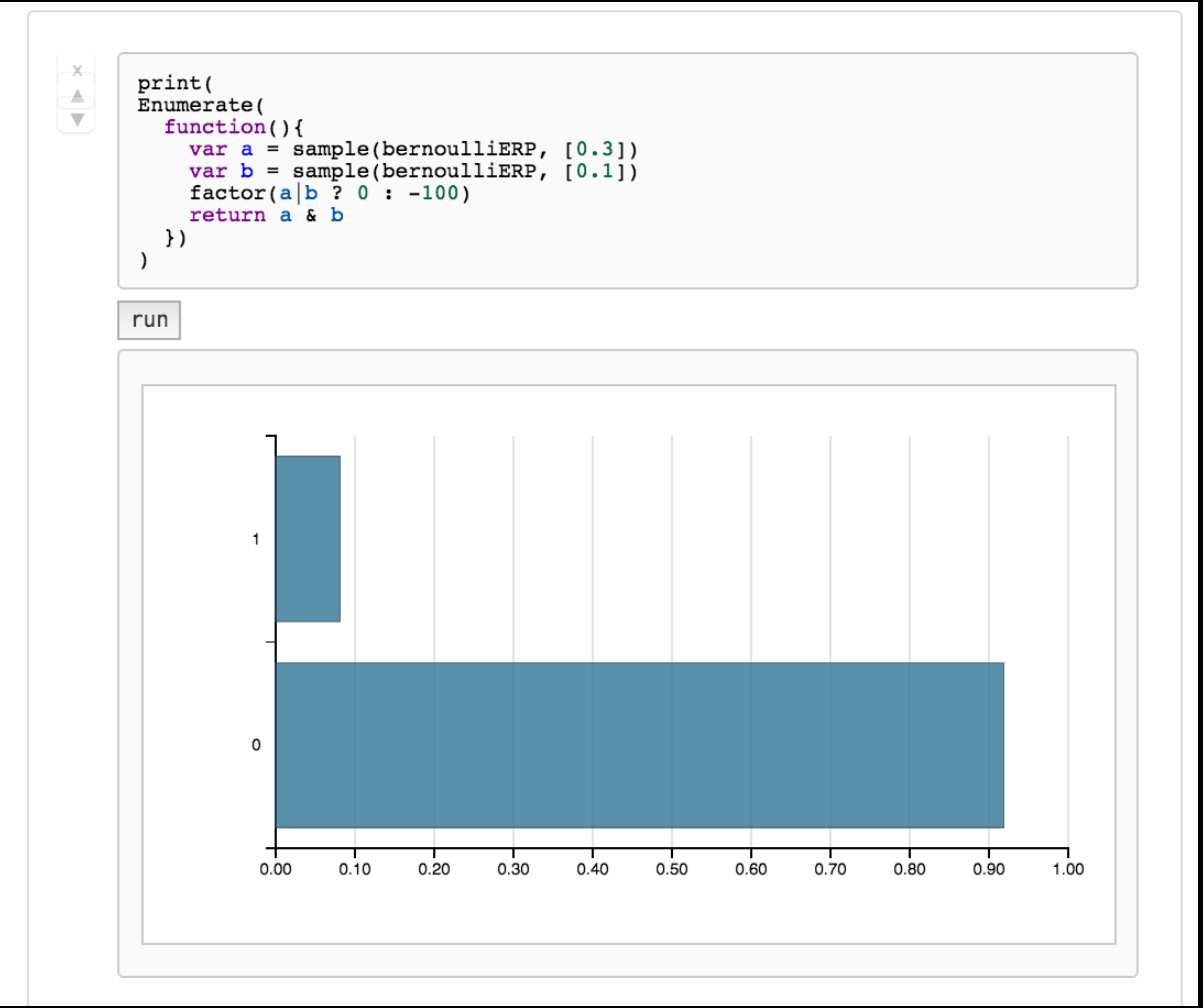

- `erp` is the marginal distribution on `val`, weighted by factors.

$$P(val) \propto \sum_{\text{leaves}} \delta_{\text{return}=val} \prod_{\text{sampled } x} e^{erp.score(x)} \prod_{\text{factor}(s)} e^{s}$$

- Inference: How do we explore the tree of executions?

# Marginal inference

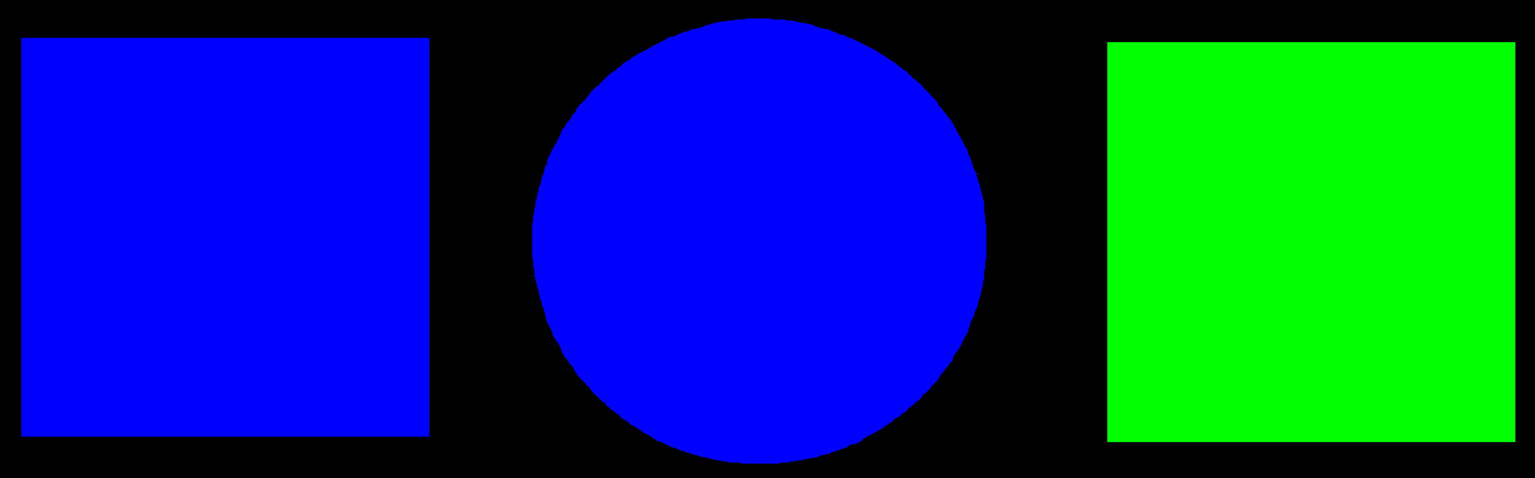
- Enumeration (with caching)
- Sequential Monte Carlo
- Markov chain Monte Carlo
- Hamiltonian Monte Carlo
- Variational inference




See dippl.org for a tutorial on implementation.

# Reference games

**Speaker**: Imagine you are talking to someone and want to refer to the middle object. Would you say "blue" or "circle"?

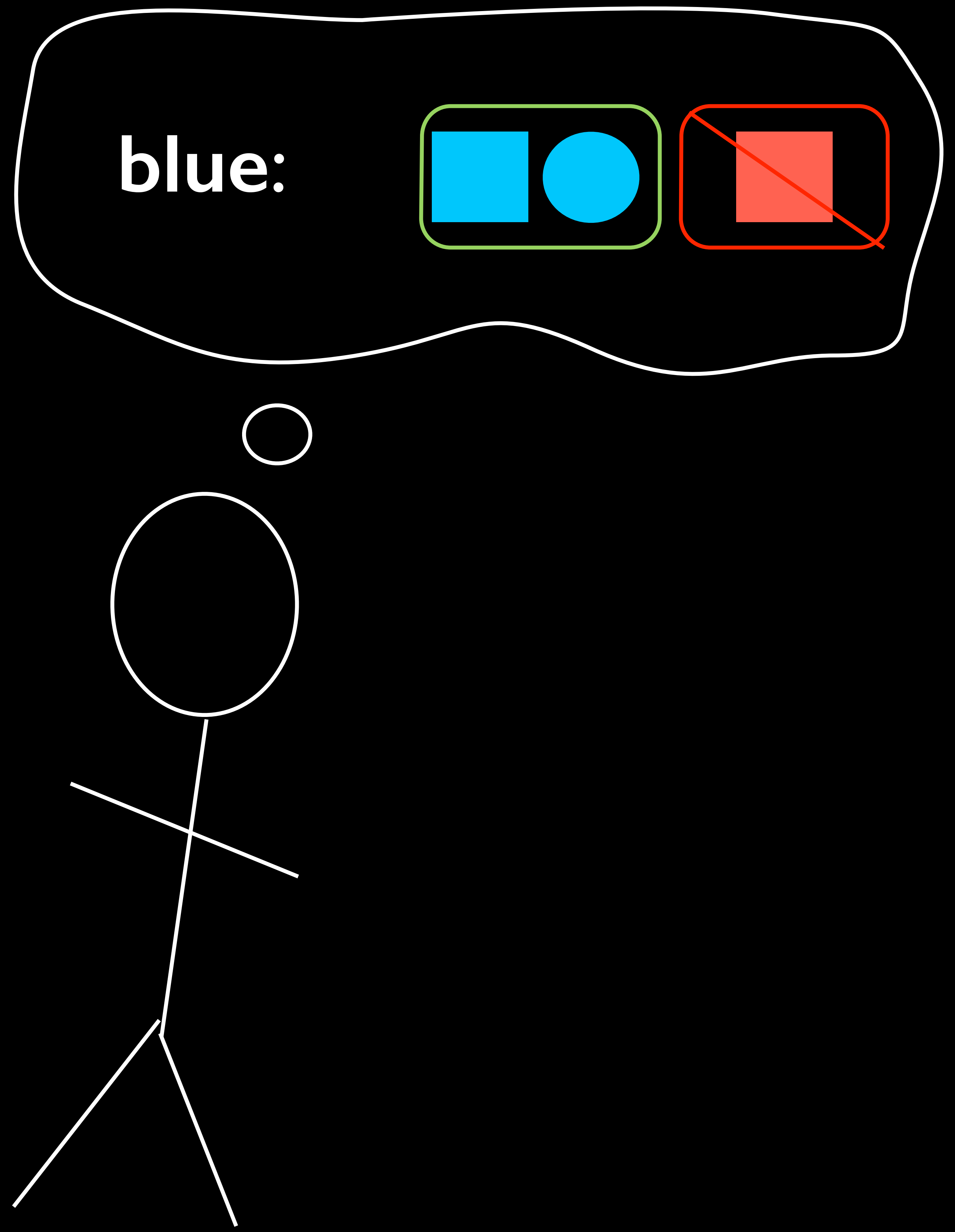


**Listener**: Someone uses the word "blue" to refer to one of these objects. Which object are they talking about?

Frank and Goodman (2012)

# Recursive reasoning

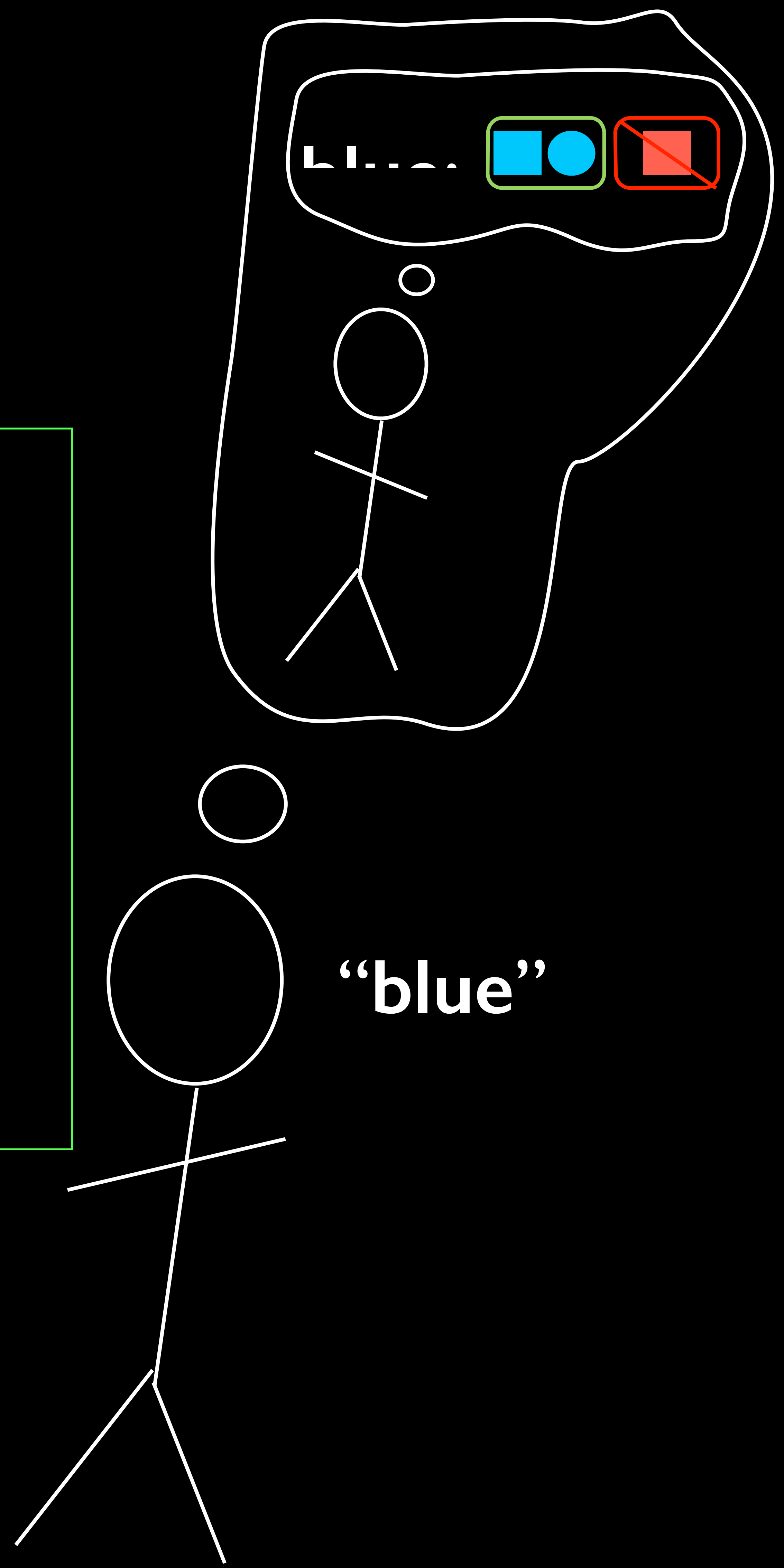
```
var literalListener = function(property)
{ Enumerate(function(){
    var object = refPrior(context)
    factor(object[property]?0:-Infinity)
    return object
  })}
```



# Recursive reasoning

```
var literalListener = function(property)
{ Enumerate(function(){
    var object = refPrior(context)
    factor(object[property]?0:-Infinity)
    return object
```

```
var speaker = function(object) {
  Enumerate(function(){
    var property = propPrior()
    factor(object ==
           sample(literalListener(property))
           ?0:-Infinity)
    return property
  })}
```

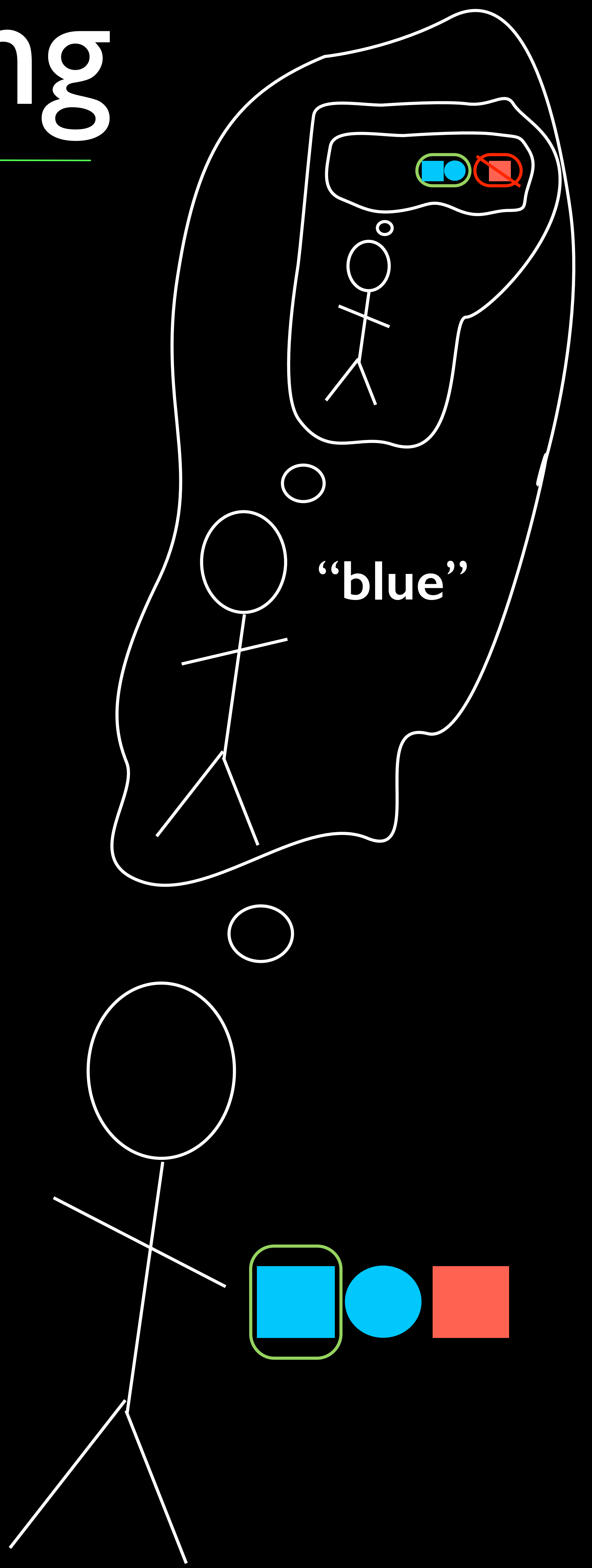


# Recursive reasoning

```
var literalListener = function(property)
{ Enumerate(function(){
    var object = refPrior(context)
    factor(object[property]?0:-Infinity)
    return object
```

```
var speaker = function(object) {
  Enumerate(function(){
    var property = propPrior()
    factor(object ==
           sample(literalListener(property))
```

```
var listener = function(property) {
  Enumerate(function(){
    var object = refPrior(context)
    factor(utterance ==
           sample(speaker(world))
           ?0:-Infinity)
    return object
  })}
```

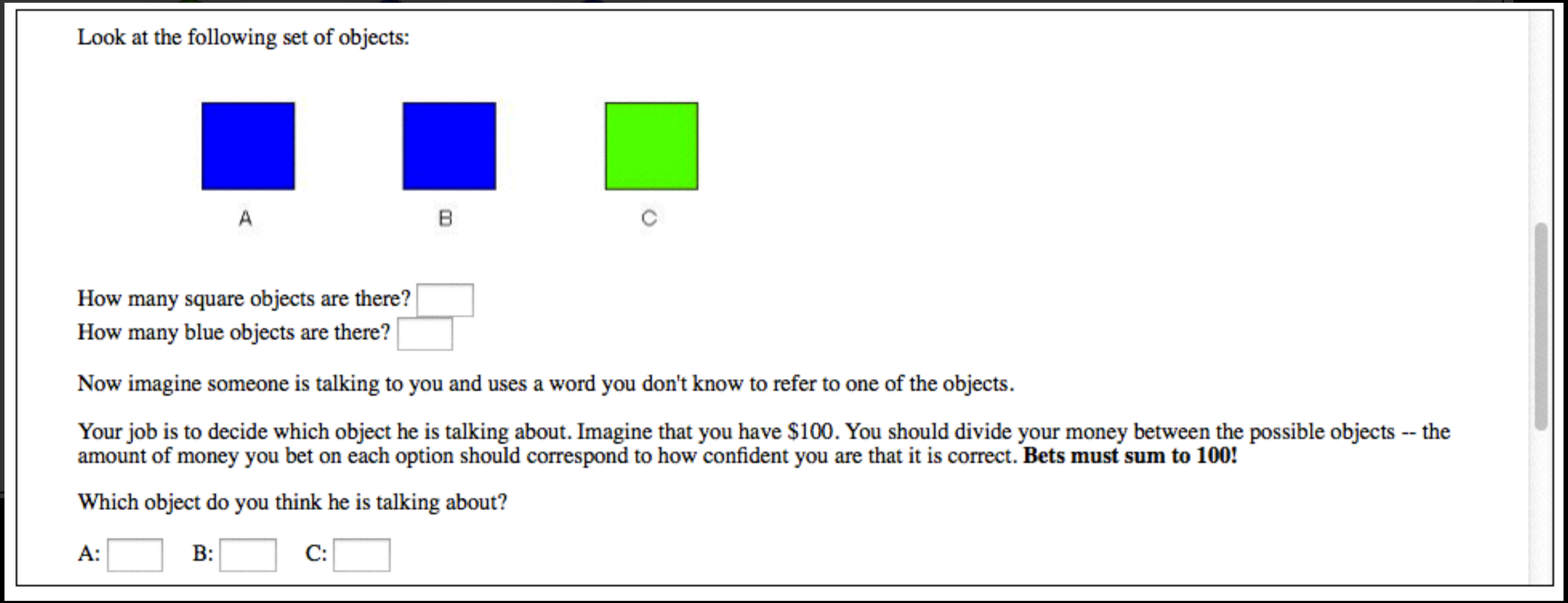


# Experiment

Speaker (N=206)

Listener (N=263)

Prior (N=276)


Look at the following set of objects:

A B C

How many square objects are there?

How many blue objects are there?

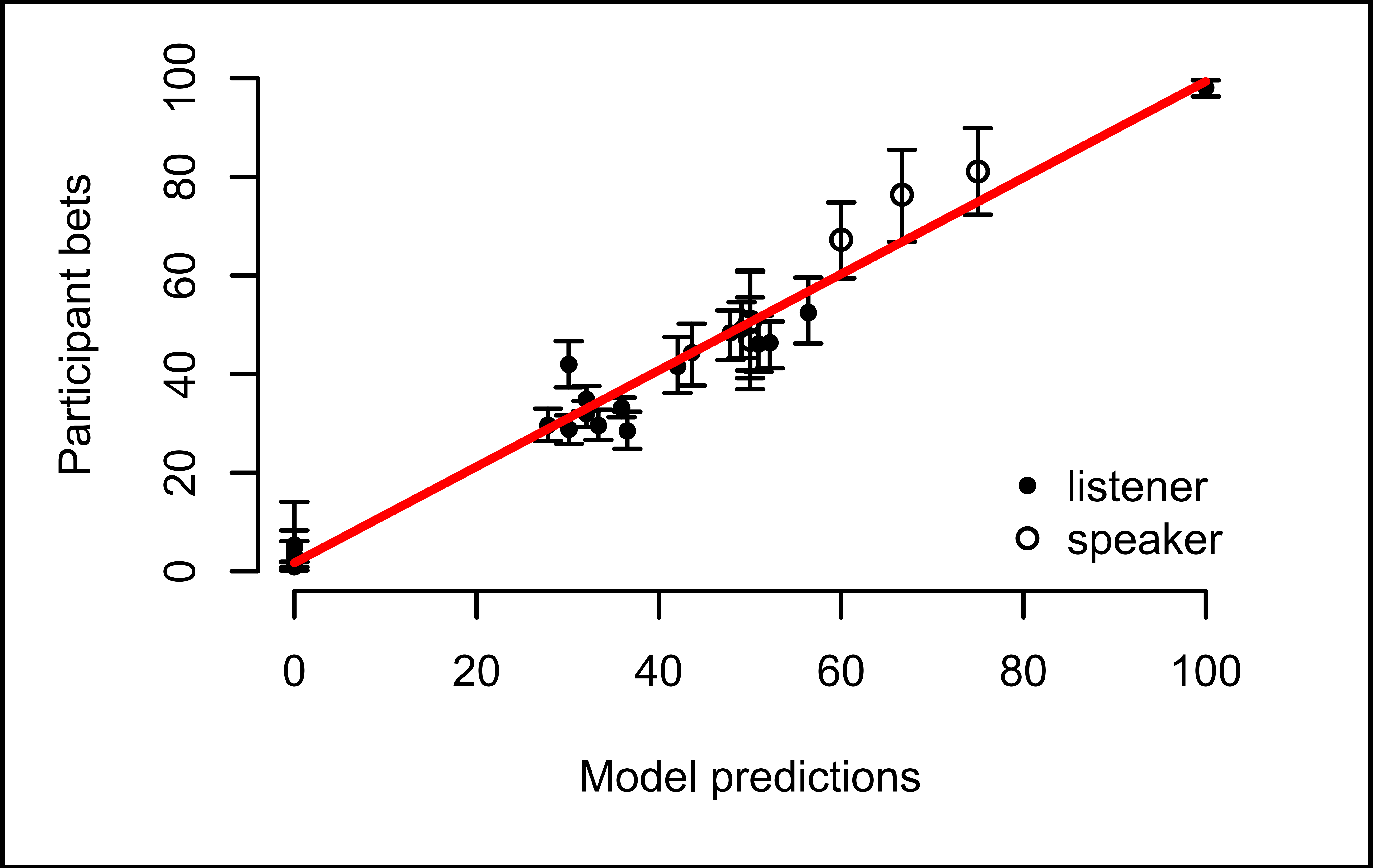
Now imagine someone is talking to you and uses a word you don't know to refer to one of the objects.

Your job is to decide which object he is talking about. Imagine that you have $100. You should divide your money between the possible objects -- the amount of money you bet on each option should correspond to how confident you are that it is correct. **Bets must sum to 100!**

Which object do you think he is talking about?

A: B: C:

# Results




- Model explains 98% of variance in data.

# Probabilistic programming languages

- A formal language for describing probabilistic models starting from a universal programming language.
- With universal inference algorithms.
- Makes it easy to:
  - prototype and explore probabilistic models
  - evaluate different inference strategies
  - make complexly structured models