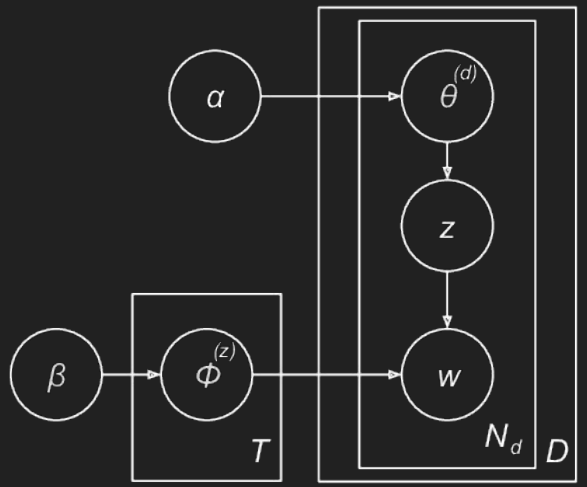
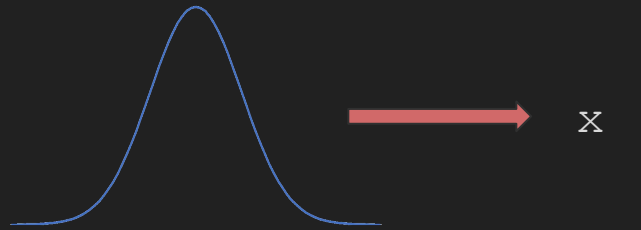
Modular, Composable Probabilistic Models

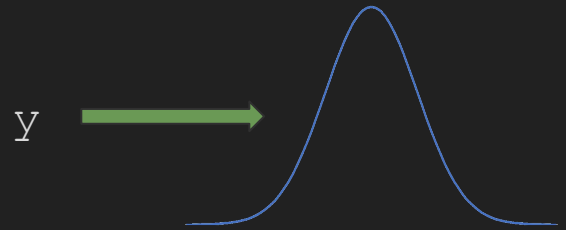


In a Probabilistic Programming Language (PPL), we can:

```
x <- sample(dist)
```




```
observe(dist, y)
```



PPLs can be categorized into either:

- Query-based PPLs
- Model-based PPLs

Query-Based PPLs

Probabilistic Query

```
def query(...):
  ...
  sample(dist)
  ...
  observe(dist, y)
  ...
```

Query for Simulation (Monad Bayes)

```
linRegr :: MonadSample m
  => Double -> Double -> Double -> Double -> m Double
linRegr x μ c σ = do
 y <- normal (μ * x + c) σ
 return y
```

Query for Inference (Monad Bayes)

```
linRegr :: MonadSample m
 => Double -> Double -> m (Double, Double, Double)
linRegr x y = do
 μ <- normal 0 3
 c <- normal 0 2
 σ <- uniform 1 3
 observe $ normalPdf (μ* x + c) σ y
 return (μ , c , σ)
```

Problem: We can't capture a universal description of a model.

Query-Based PPLs

Query for Simulation (WebPPL)

```
var linearRegr = function(mu, sigma, x) {
 var y = sample(Normal(mu * x, sigma))
 return y
}
var linearRegrModel = function () {
 linearRegr({mu = 0, sigma = 1, x = 4})
}
```

Query for Inference (WebPPL)

```
var linearRegr = function(mu, sigma, x, data_y) {
 observe(Normal(mu * x, sigma), data_y)
 return (mu, sigma)
}
var linearRegrModel = function () {
 linearRegr({mu = 0, sigma = 1, x = 4, data_y = 3})
}
var params = Infer({ model: linearRegrModel })
```

Query for Simulation (Anglican)

```
(defquery linear-regression [mu-prior, x]
 (let [mu    (sample mu-prior)
       sigma (sample sigma-prior)
       y (reduce + (map (* mu) x)) sigma)]
   {:output y})))
```

Query for Inference (Anglican)

```
(defquery linear-regression [mu-prior data_y x]
 (let [mu    (sample mu-prior)
       sigma (sample sigma-prior)
       predictive (fn [x] (normal (reduce + (map (* mu) x)) sigma))]
       (observe (predictive x) data_y)
   {:mu mu :sigma sigma :predictor (predictive x)})))
```

Model-Based PPLs

Writing a Model (Turing.jl)

```
@model function linearRegression(μ, c, σ, x,
y)
    μ ~ Normal(0, 3)
    c ~ Normal(0, 2)
    σ ~ Uniform(1, 3)
    y ~ Normal(μ * x + c, σ)
end
```

Problem: Models are not first-class citizens

Composable, Modular Probabilistic Models

- Models are interpretable for simulation *and* inference
- Models are first-class citizens - they can be combined and composed

- Extensible Algebraic Effects and Distribution Types
- Extensible Records and Affine Reader Effects

Example: A Hidden Markov Model

```
transitionModel :: Double -> Int -> Model env ts Int
transitionModel transition_p x_prev = do
 dX <- bernoulli transition_p Nothing
 return (x_prev + dX)

observationModel :: (Observable env "y" Int)
     => Double -> Int -> Model env ts Int
observationModel observation_p x = do
 binomial' x observation_p #y


hmm :: (Observable s "y" Int)
     => Double -> Double -> Int -> Model env ts Int
hmm transition_p observation_p x_prev = do
 x <- transitionModel transition_p x_prev
 y <- observationModel observation_p x
 return x


hmmNSteps :: (Observable s "y" Int)
     => Double -> Double -> Int -> (Int -> Model env ts Int)
hmmNSteps transition_p observation_p n =
 foldl (>=>) return (replicate n (hmm transition_p observation_p))
```