Probabilistic Programming

Frank Wood
fwood@robots.ox.ac.uk

NIPS TUTORIAL 2015

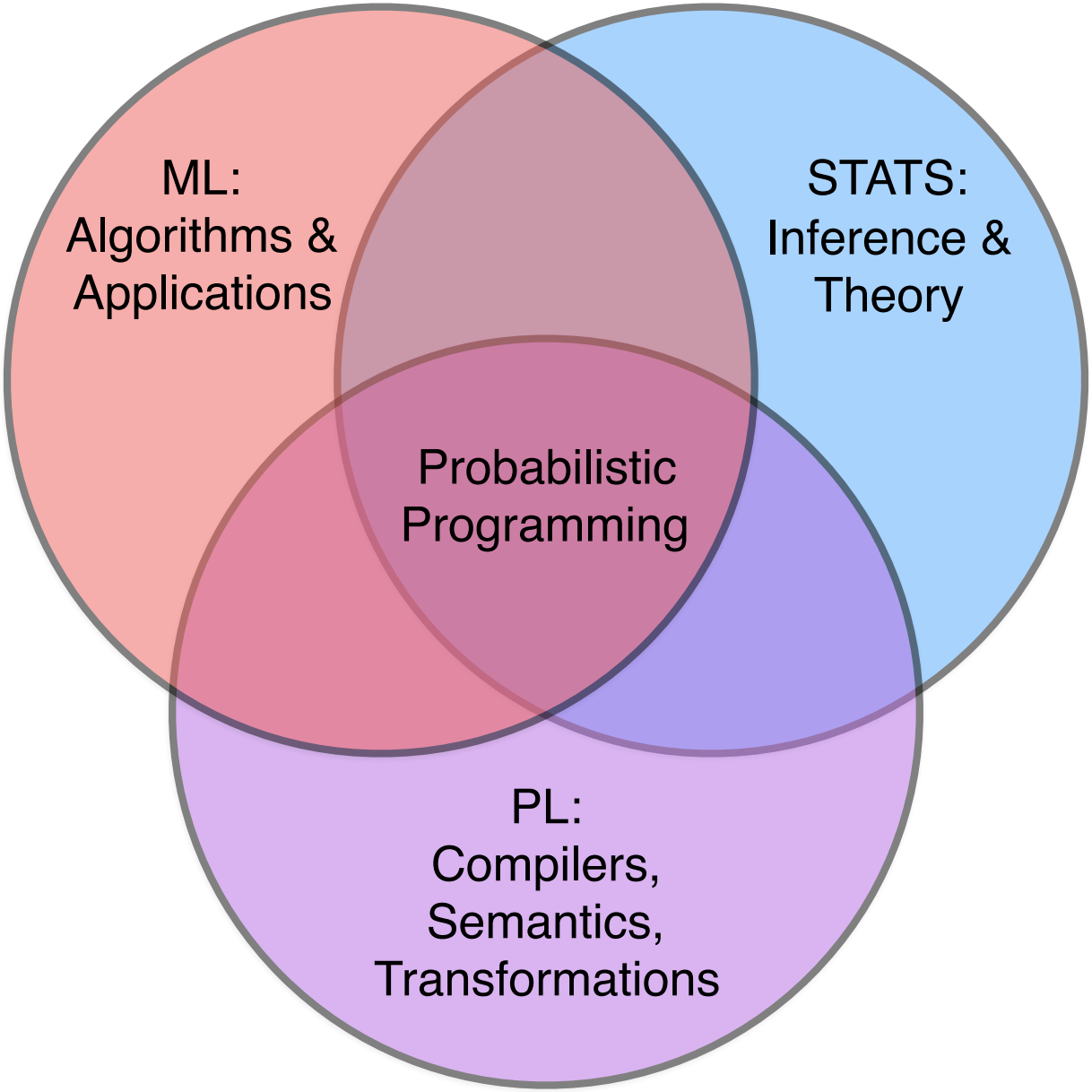
My Objectives

Get you to

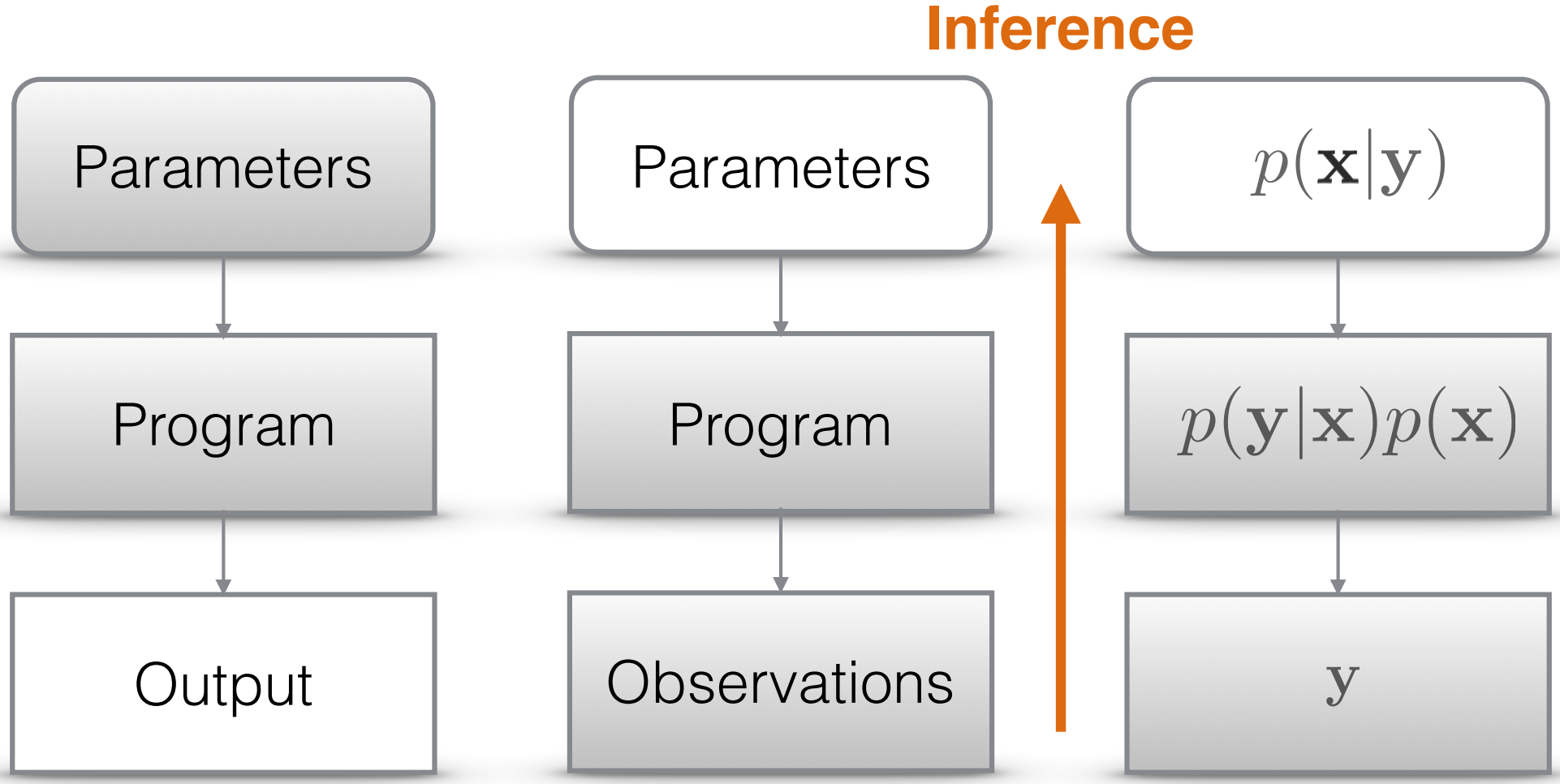
- Know what probabilistic program is and how it's different to a normal program.
- Roughly understand how to write a probabilistic program and have the resources to get started if you want to.
- Understand the literature at a very high level.
- Know one way to roll your own state-of-the-art universal probabilistic programming system.

What is probabilistic programming?

The Field


ML:
Algorithms &
Applications
STATS:
Inference &
Theory
Probabilistic
Programming
PL:
Compilers,
Semantics,
Transformations

Intuition



CS Probabilistic Programming Statistics

A Probabilistic *Program*

"Probabilistic programs are usual functional or imperative programs with two added constructs:

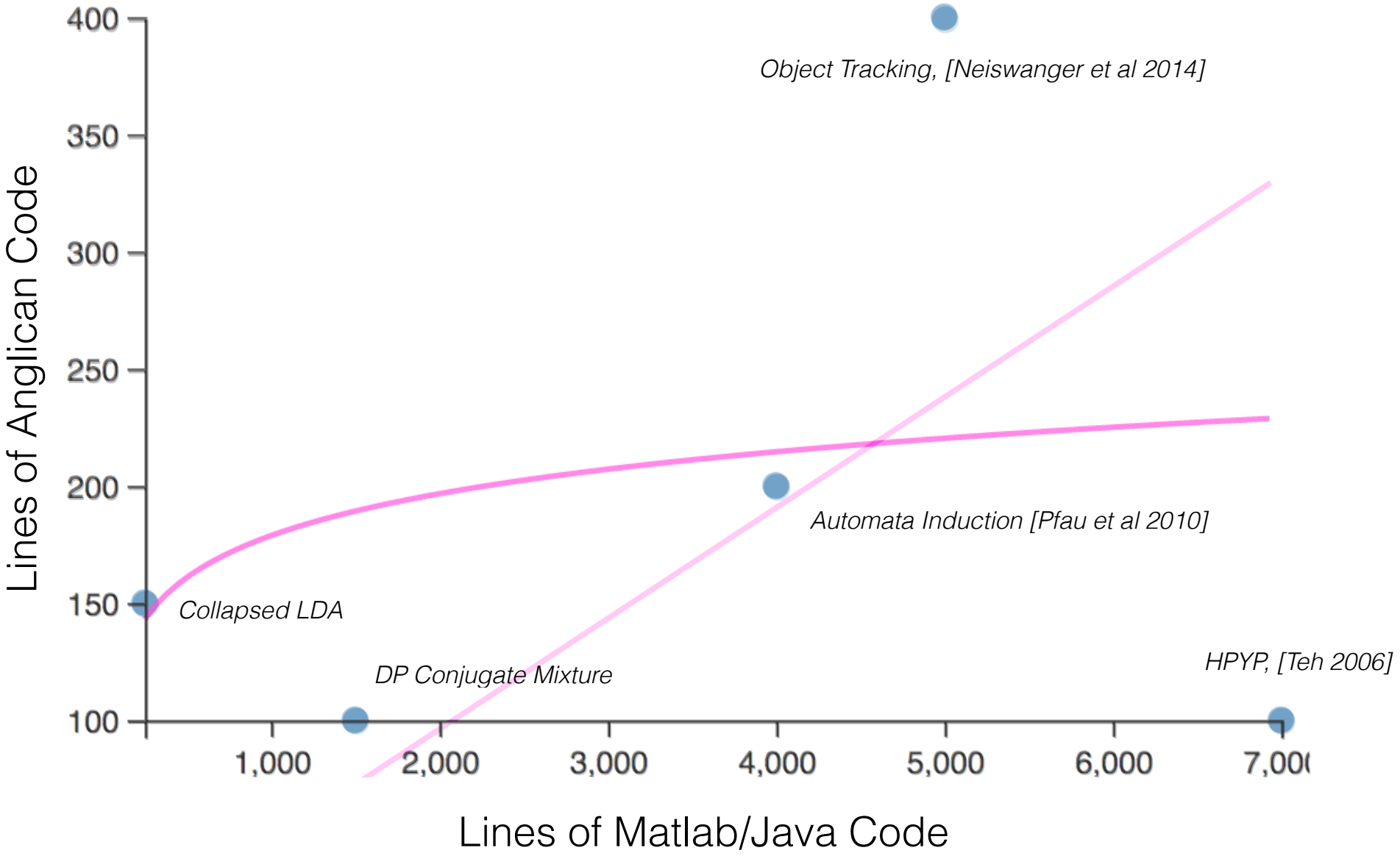
(1) the ability to draw values at random from distributions, and

(2) the ability to condition values of variables in a program via observations."

Gordon, Henzinger, Nori, and Rajamani
"Probabilistic programming." In Proceedings of On The Future of Software Engineering (2014).

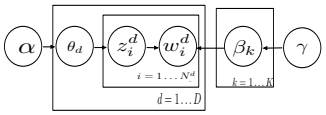
Goals of the Field

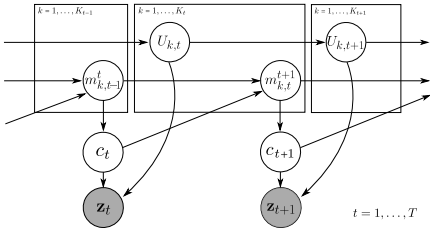
Increase Programmer Productivity

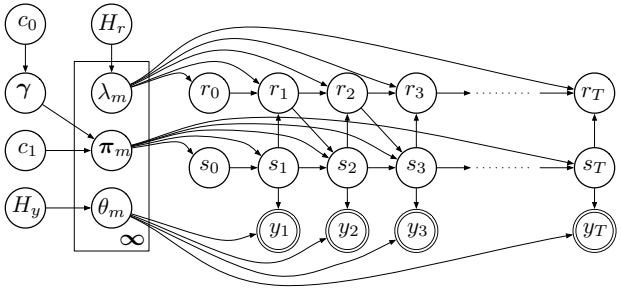


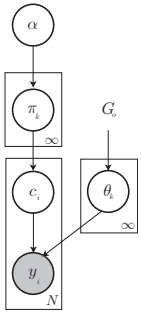
Commodify Inference

Models / Simulators

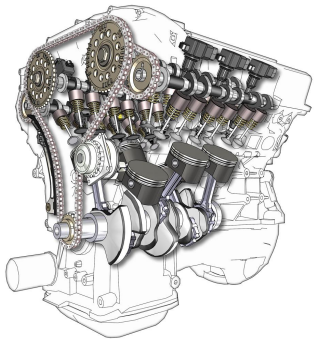








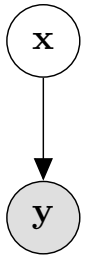
Language Representation / Abstraction Layer



Inference engines

New Kinds of Models

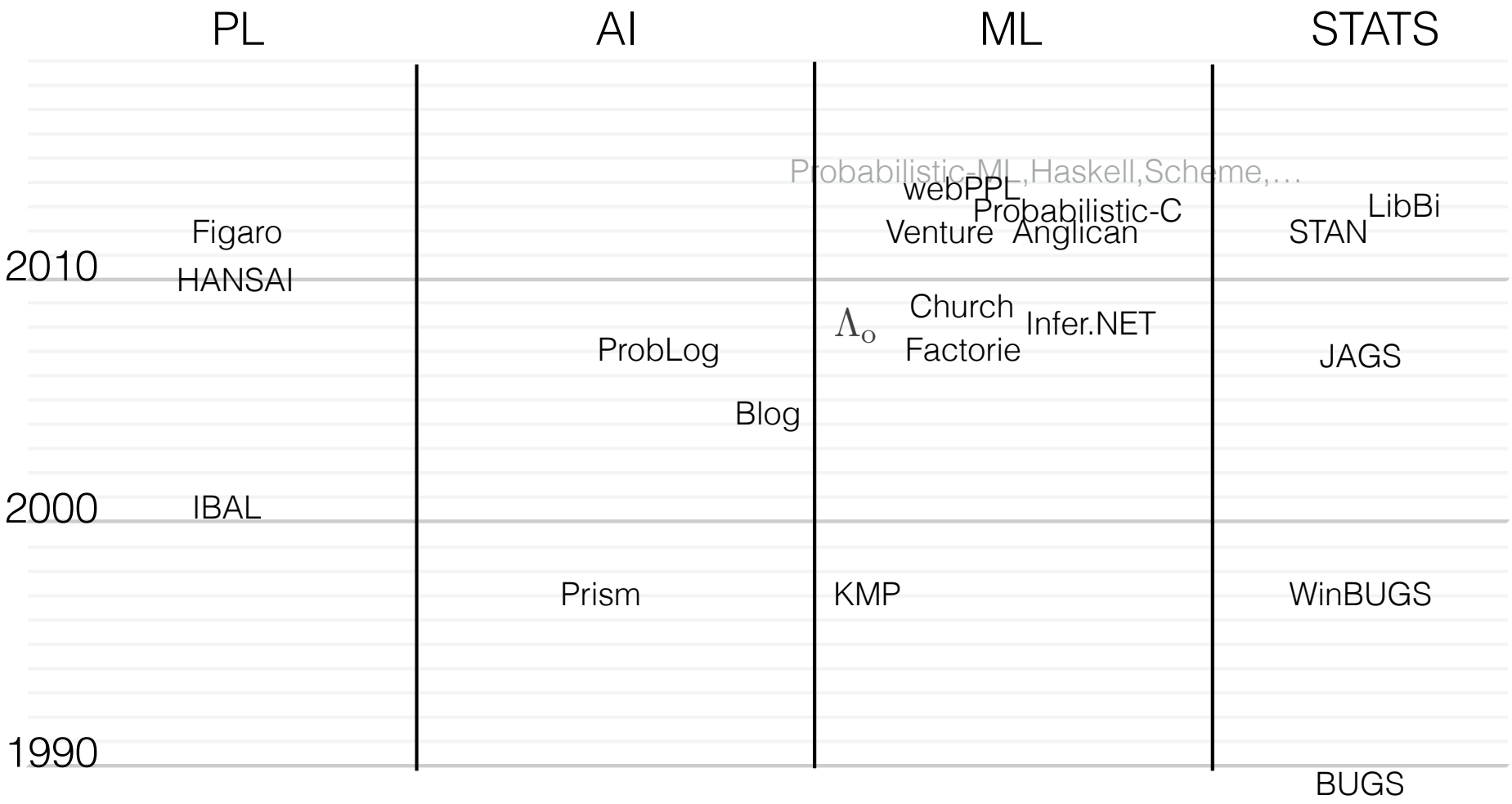
$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{y})}$$



$\mathbf{x}$	$\mathbf{y}$
program source code	program output
scene description	image
policy and world	rewards
cognitive process	behavior
simulation	constraint

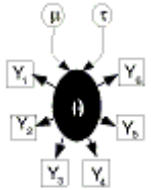
History

Long


PL
AI
ML
STATS
Probabilistic-ML,Haskell,Scheme,…
webPPL
Probabilistic-C
LibBi
Figaro
Venture
Anglican
STAN
2010
HANSAI
Church
Infer.NET
ProbLog
Factorie
JAGS
Blog
2000
IBAL
Prism
KMP
WinBUGS
1990
BUGS
Simula
Prolog

Success Stories

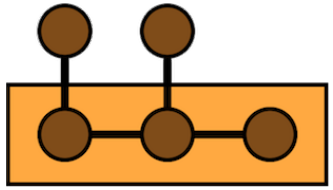
Graphical Models



BUGS

STAN

Factor Graphs

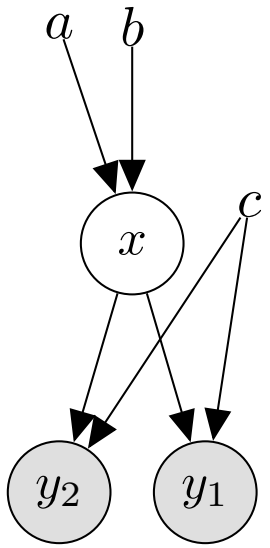


Factorie

Infer.NET

BUGS

```
model {
    x ~ dnorm(a, 1/b)
    for (i in 1:N) {
       y[i] ~ dnorm(x, 1/c)
    }
}
```



- Language restrictions
  - Bounded loops
  - No branching
- Model class
  - Finite graphical models
- Inference - sampling
  - Gibbs

Spiegelhalter et al. "BUGS: Bayesian inference using Gibbs sampling, Version 0.50." Cambridge 1995.

STAN : Finite Dimensional Differentiable Distributions

```
parameters {
    real xs[T];
}
model {
    xs[1] ~ normal(0.0, 1.0);
    for (t in 2:T)
        xs[t] ~ normal(a * xs[t - 1], q);
    for (t in 1:T)
        ys[t] ~ normal(xs[t], 1.0);
}
```

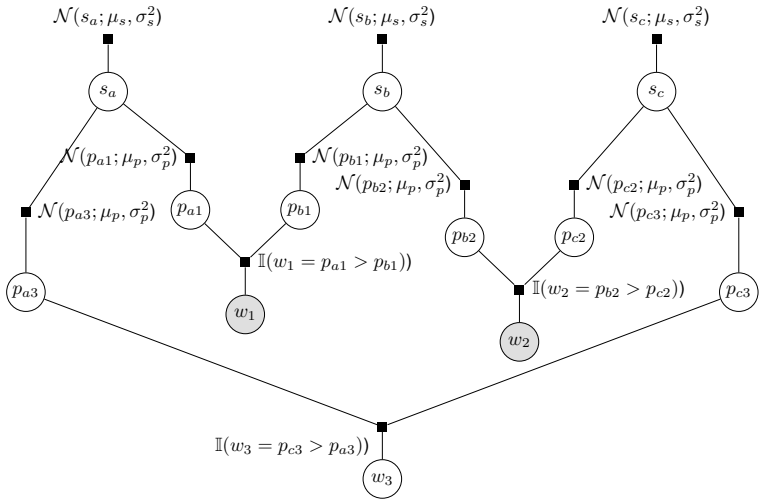
$$\nabla_{\mathbf{x}} \log p(\mathbf{x}, \mathbf{y})$$

- Language restrictions
  - Bounded loops
  - No discrete random variables*
- Model class
  - Finite dimensional differentiable distributions
- Inference - sampling
  - Hamiltonian Monte Carlo
    - Reverse-mode automatic differentiation
  - Black box variational inference, etc.

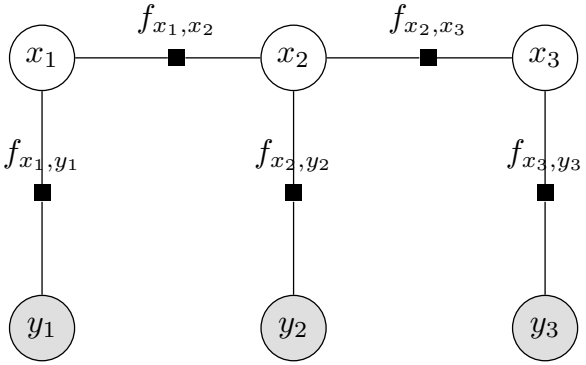
STAN Development Team "Stan: A C++ Library for Probability and Sampling." 2014.

Factorie and Infer.NET

- Language restrictions
  - Finite compositions of factors
- Model class
  - Finite factor graphs
- Inference - message passing, etc.



TrueSkill



CRF

Minka, Winn, Guiver, and Knowles "Infer .NET 2.4, Microsoft Research Cambridge." 2010.

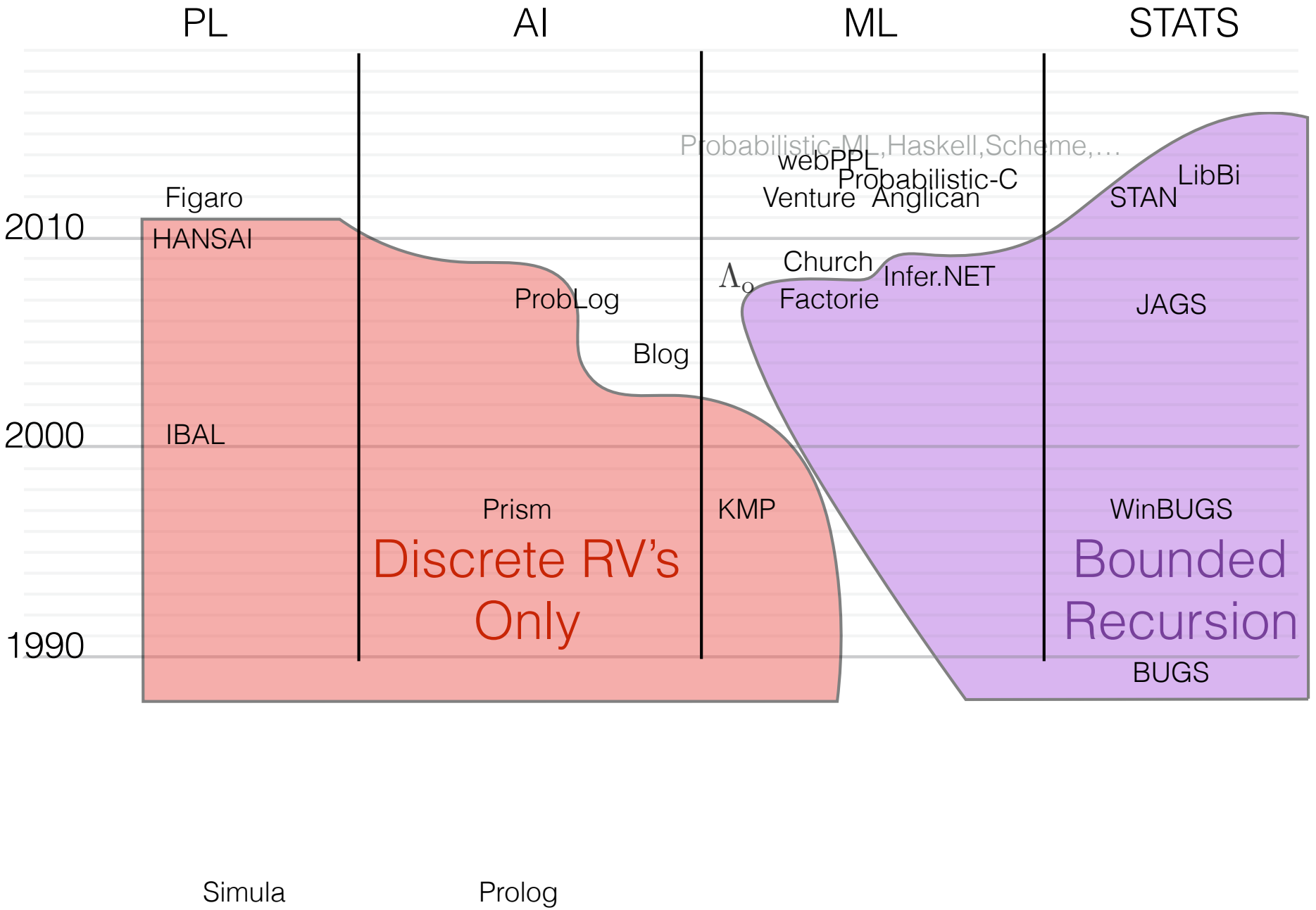
McCallum, Schultz, and Singh. “Factorie Probabilistic programming via imperatively defined factor graphs.“ NIPS 2009

Modeling language desiderata

- Unrestricted
  - “Open-universe” / infinite dim. parameter spaces
  - Mixed variable types
- Unfettered access to existing libraries
- Easily extensible

- Will come at a cost
  - Inference is going to be harder
  - More ways to shoot yourself in the foot

Languages and Systems

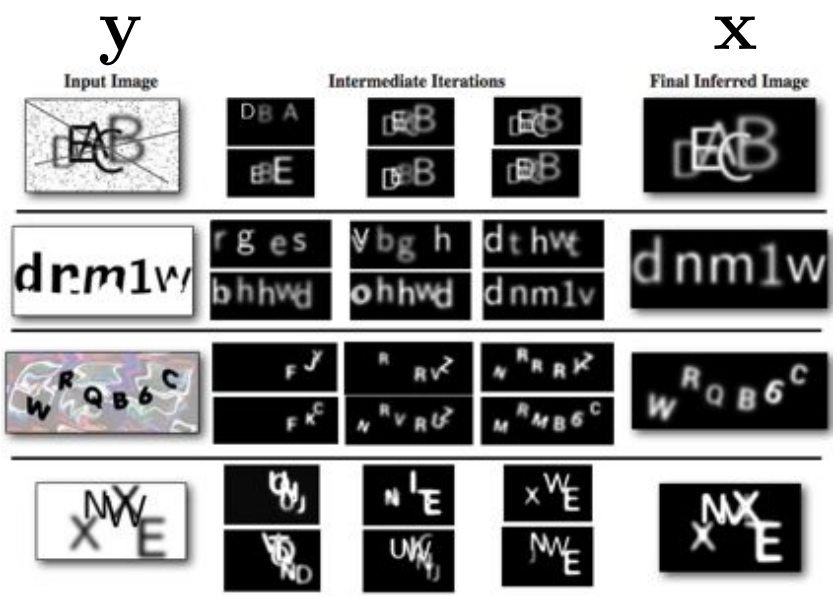


What people are doing with these languages

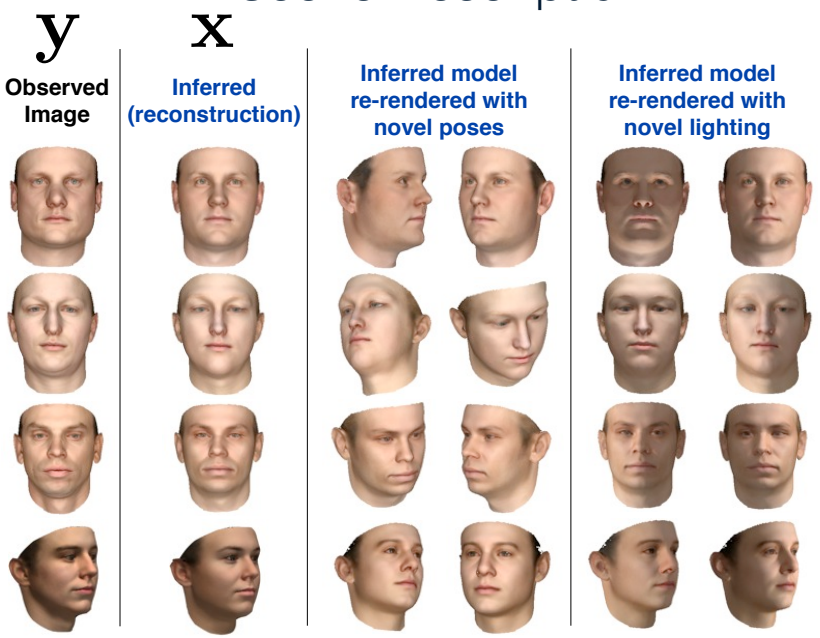
$\mathbf{x}$	$\mathbf{y}$
program source code	program output
scene description	image
policy and world	rewards
cognitive process	behavior
simulation	constraint

Perception / Inverse Graphics

Captcha Solving



Scene Description

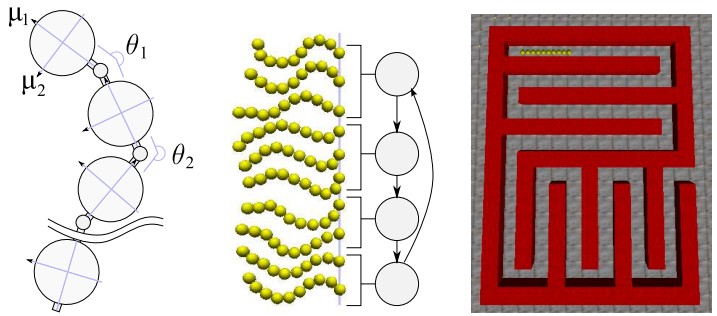


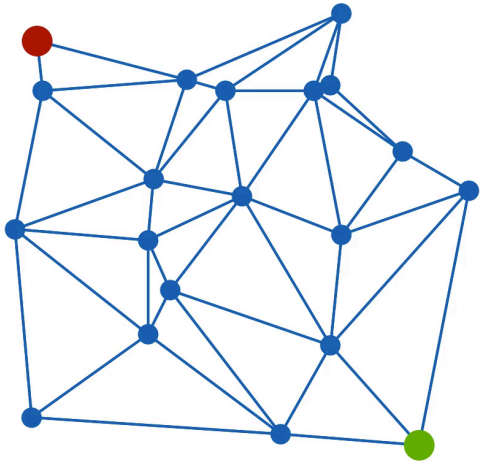
x	y
scene description	image

Mansinghka,, Kulkarni, Perov, and Tenenbaum. "Approximate Bayesian image interpretation using generative probabilistic graphics programs." NIPS (2013).

Kulkarni, Kohli, Tenenbaum, Mansinghka "Picture: a probabilistic programming language for scene perception." CVPR (2015).

Reinforcement Learning





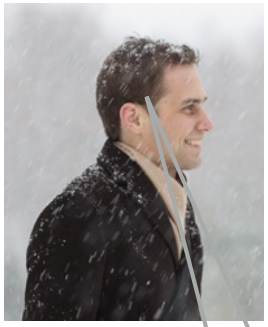
x	**y**
policy and world	reward

Wingate, Goodman, Roy, Kaelbling, and Tenenbaum. "Bayesian policy search with policy priors." (IJCAI), 2011.

van de Meent, Tolpin, Paige, and Wood. "Black-Box Policy Search with Probabilistic Programs." arXiv:1507.04635 (2015).

Reasoning about reasoning

Want to meet up but phones are dead…

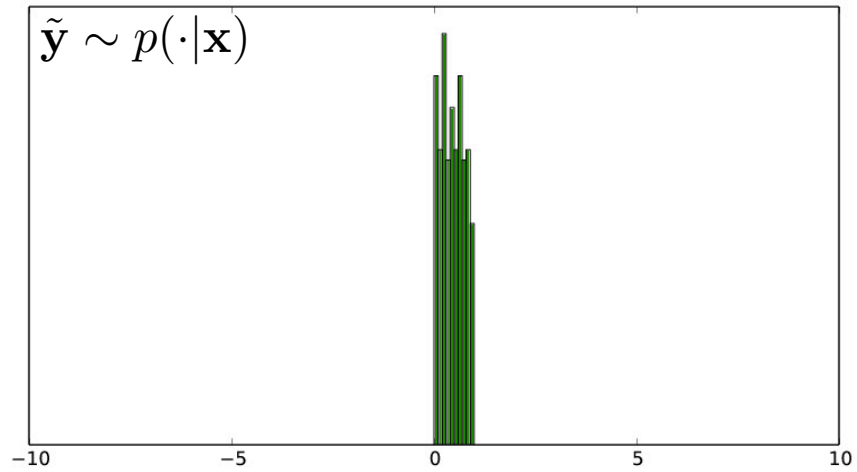


I prefer the pub.
Where will Noah go?
Simulate Noah:
Noah prefers pub
but will go wherever Andreas is
Simulate Noah simulating Andreas:
…
-> both go to pub

$\mathbf{x}$	$\mathbf{y}$
cognitive process	behavior

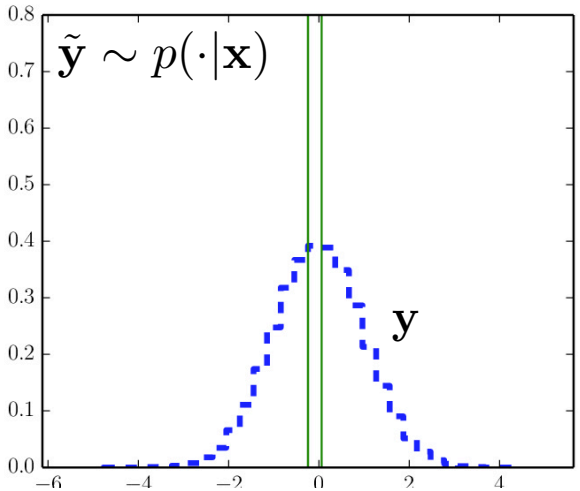
Stuhlmüller, and Goodman.
"Reasoning about reasoning by nested conditioning: Modeling theory of mind with probabilistic programs."
Cognitive Systems Research 28 (2014): 80-99.

Program Induction




```
(lambda (stack-id) (safe-uc (* (if (< 0.0 (* (* (* -1.0 (begin (define
 G__1147 (safe-uc 1.0 1.0)) 0.0)) (* 0.0 (+ 0.0 (safe-uc (* (* (dec -2
.0) (safe-sqrt (begin (define G__1148 3.14159) (safe-log -1.0)))) 2.0)
 0.0)))) 1.0)) (+ (safe-div (begin (define G__1149 (* (+ 3.14159 -1.0)
 1.0)) 1.0) 0.0) (safe-log 1.0)) (safe-log -1.0)) (begin (define G__11
...
```

$$\mathbf{x} \sim p(\mathbf{x})$$



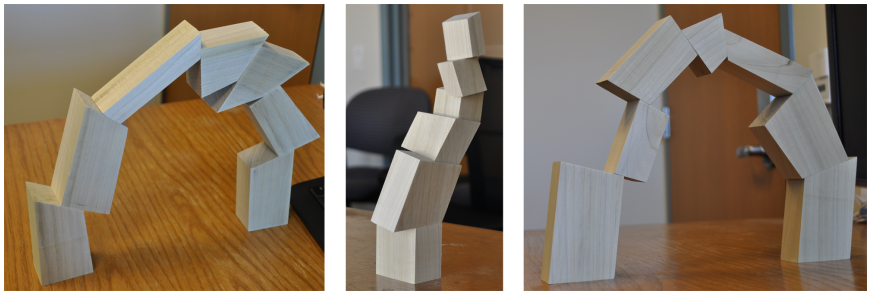
$$\mathbf{x} \sim p(\mathbf{x}|\mathbf{y})$$

$\mathbf{x}$	$\mathbf{y}$
program source code	program output

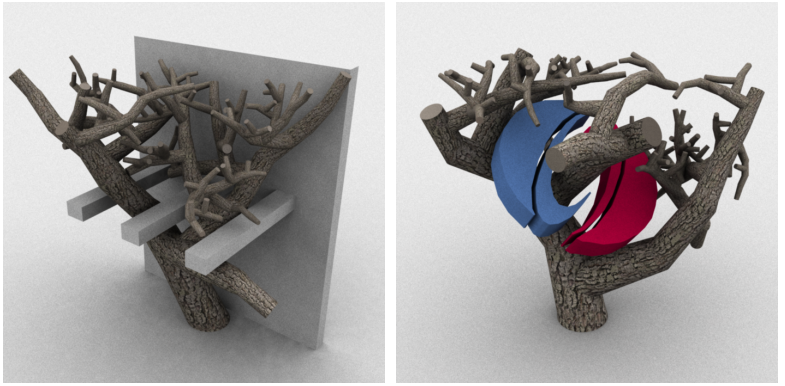
Perov and Wood.
"Learning Probabilistic Programs."
arXiv:1407.2646 (2014).

Constrained Stochastic Simulation

Stable Static Structures



Procedural Graphics



$\mathbf{x}$	$\mathbf{y}$
simulation	constraint

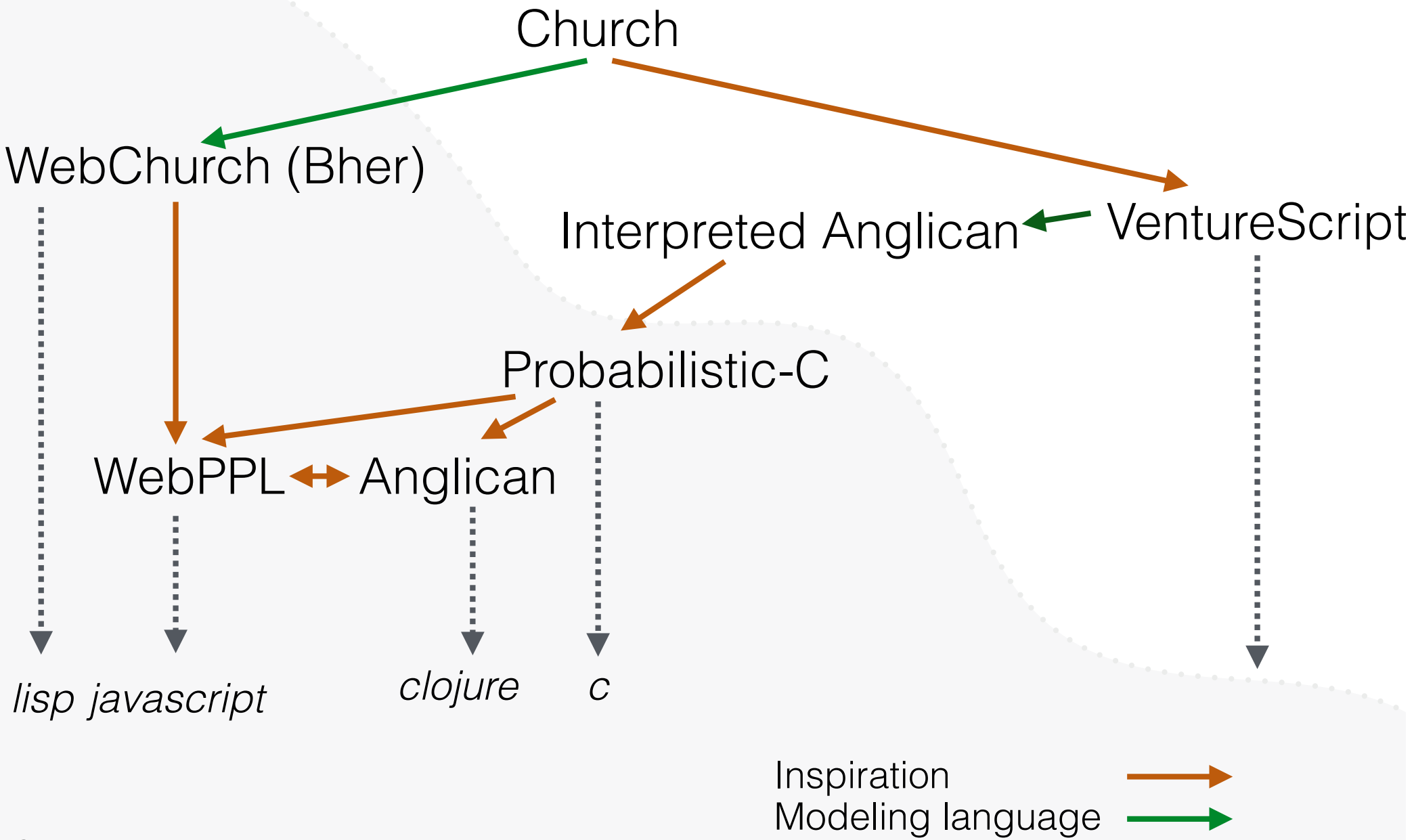
Ritchie, Lin, Goodman, & Hanrahan.
Generating Design Suggestions under Tight Constraints with Gradient-based Probabilistic Programming.
In Computer Graphics Forum, (2015)

Ritchie, Mildenhall, Goodman, & Hanrahan.
"Controlling Procedural Modeling Programs with Stochastically-Ordered Sequential Monte Carlo."
SIGGRAPH (2015)

Universal Probabilistic Programming Modeling Language

Introduction to Anglican/Church/Venture/WebPPL…

A Language Family Tree



Syntax : Anglican ≈ Clojure ≈ Church ≈ Lisp

- Notation : *Prefix* vs. infix

```
;; Add two numbers
(+ 1 1)

;; Subtract: "10 - 3"
(- 10 3)

;; (10 * (2.1 + 4.3) / 2)
(/ (* 10 (+ 2.1 4.3)) 2)
```

Syntax

- Notation : *Prefix* vs. infix

```
;; Add two numbers
(+ 1 1)

;; Subtract: "10 - 3"
(- 10 3)

;; (10 * (2.1 + 4.3) / 2)
(/ (* 10 (+ 2.1 4.3)) 2)
```

- Branching

```
;; outputs 4
(+ (if (< 4 5) 1 2) 3)
```

Functions

- Functions are first class

```
;; evaluates to 32
((fn [x y] (+ (* x 3) y))
  10
  2)
```

Functions

- Functions are first class

```
;; evaluates to 32
((fn [x y] (+ (* x 3) y))
  10
  2)
```

- Local bindings

```
;; let is syntactic "sugar" for the same
(let [x 10
      y 2]
  (+ (* x 3) y))
```

Higher-Order

- map

```
;; Apply the function f(x,y) = x + 2y to the
;; x values [1 2 3] and the y values [10 9 8]
;; Produces [21 20 19]
(map (fn [x y] (+ x (* 2 y)))
     [1 2 3]    ; these are values x1, x2, x3
     [10 9 8]) ; these are values y1, y2, y3
```

- reduce

```
;; Reduce recursively applies function,
;; to result and next element, i.e.
(reduce + 0 [1 2 3 4])
;; does (+ (+ (+ 0 1) 2) ...
;; and evaluates to 10
```

Anglican By Example : Graphical Model

```
(defquery gaussian-model [data]
  (let [x (sample (normal 1 (sqrt 5)))
        sigma (sqrt 2)]
    (map (fn [y] (observe (normal x sigma) y)) data)
    (predict :x x)))
```

$$x \sim \text{Normal}(1, \sqrt{5})$$
$$y_i|x \sim \text{Normal}(x, \sqrt{2})$$

```
(def dataset [9 8])

(def posterior
  ((conditional gaussian-model
                :pgibbs
                :number-of-particles 1000) dataset))

(def posterior-samples
  (repeatedly 20000 #(sample posterior)))
```

$$y_1 = 9, y_2 = 8$$

$$x|\mathbf{y} \sim \text{Normal}(7.25, 0.91)$$

Graphical Model

```
(defquery gaussian-model [data]
  (let [x (sample (normal 1 (sqrt 5)))
        sigma (sqrt 2)]
    (map (fn [y] (observe (normal x sigma) y)) data)
    (predict :x x)))
```

$$x \sim \text{Normal}(1, \sqrt{5})$$

$$y_i | x \sim \text{Normal}(x, \sqrt{2})$$

```
(def dataset [9 8])
```

$$y_1 = 9, y_2 = 8$$

```
(def posterior
  ((conditional gaussian-model
                :pgibbs
                :number-of-particles 1000) dataset))
```

$$x|\mathbf{y} \sim \text{Normal}(7.25, 0.91)$$

```
(def posterior-samples
  (repeatedly 20000 #(sample posterior)))
```

Graphical Model

```
(defquery gaussian-model [data]
  (let [x (sample (normal 1 (sqrt 5)))
        sigma (sqrt 2)]
    (map (fn [y] (observe (normal x sigma) y)) data)
    (predict :x x)))
```

$$x \sim \text{Normal}(1, \sqrt{5})$$

$$y_i | x \sim \text{Normal}(x, \sqrt{2})$$

```
(def dataset [9 8])
```

$$y_1 = 9, y_2 = 8$$

```
(def posterior
  ((conditional gaussian-model
                :pgibbs
                :number-of-particles 1000) dataset))
```

$$x|\mathbf{y} \sim \text{Normal}(7.25, 0.91)$$

```
(def posterior-samples
  (repeatedly 20000 #(sample posterior)))
```

Graphical Model

```
(defquery gaussian-model [data]
  (let [x (sample (normal 1 (sqrt 5)))
        sigma (sqrt 2)]
    (map (fn [y] (observe (normal x sigma) y)) data)
    (predict :x x)))
```

$$x \sim \text{Normal}(1, \sqrt{5})$$

$$y_i | x \sim \text{Normal}(x, \sqrt{2})$$

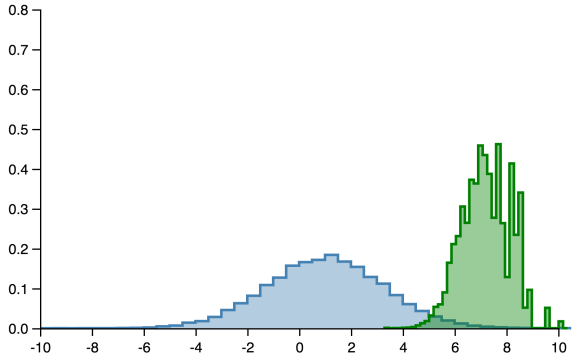
```
(def dataset [9 8])
```

$$y_1 = 9, y_2 = 8$$

```
(def posterior
  ((conditional gaussian-model
                :pgibbs
                :number-of-particles 1000) dataset))
```

$$x|\mathbf{y} \sim \text{Normal}(7.25, 0.91)$$

```
(def posterior-samples
  (repeatedly 20000 #(sample posterior)))
```



Anglican : Syntax ≈ Clojure, Semantics ≠ Clojure

```
(defquery gaussian-model [data]
  (let [x (sample (normal 1 (sqrt 5)))
        sigma (sqrt 2)]
    (map (fn [y] (observe (normal x sigma) y)) data)
    (predict :x x)))
```

$$x \sim \text{Normal}(1, \sqrt{5})$$

$$y_i | x \sim \text{Normal}(x, \sqrt{2})$$

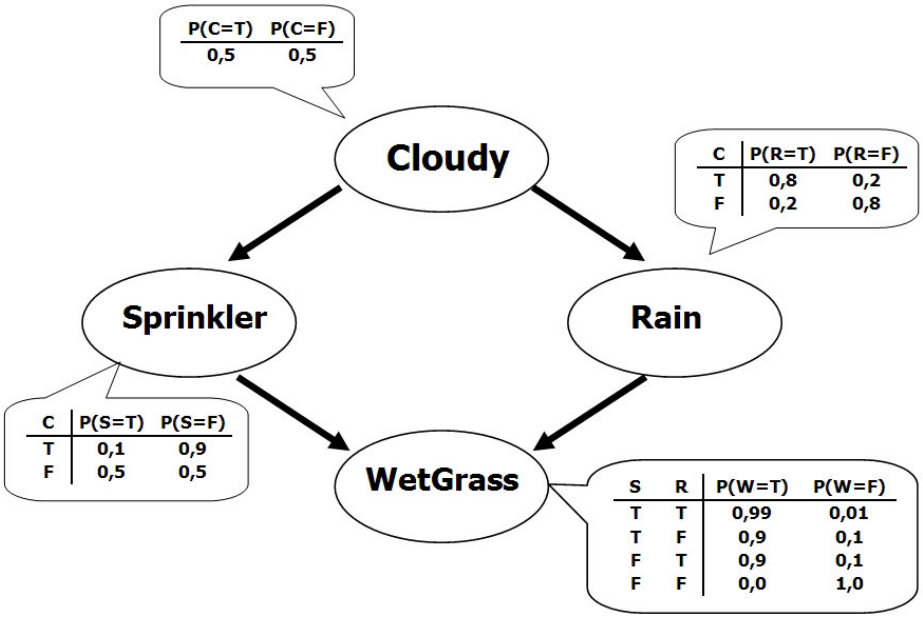
```
(def dataset [9 8])

(def posterior
  ((conditional gaus                    l
                :pgi
                :num                  rticles 1000) dataset))
```

$$y_1 = 9, y_2 = 8$$

$$x | \mathbf{y} \sim \text{Normal}(7.25, 0.91)$$

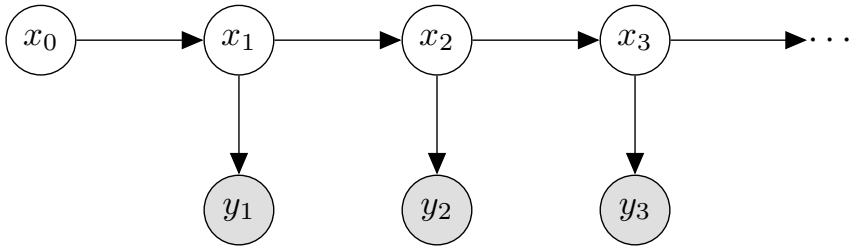
```
(def posterior-samples
  (repeatedly 20000 #(sample posterior)))
```

Bayes Net




```
(defquery sprinkler-bayes-net
  (let [cloudy (sample (flip 0.5))

        raining (sample (if cloudy
                          (flip 0.8)
                          (flip 0.2)))

        sprinkler-dist (if cloudy
                         (flip 0.1)
                         (flip 0.5))
        sprinkler true
        _ (observe sprinkler-dist sprinkler)

        wetgrass-dist  (cond (and (= sprinkler true)
                                  (= raining   true))
                             (flip 0.99)
                             (and (= sprinkler false)
                                  (= raining   false))
                             (flip 0.00)
                             (or  (= sprinkler true)
                                  (= raining   true))
                             (flip 0.90))
        wetgrass true
        _ (observe wetgrass-dist wetgrass)]

    (predict :cloudy cloudy)
    (predict :raining raining)
    (predict :wetgrass wetgrass)))
```

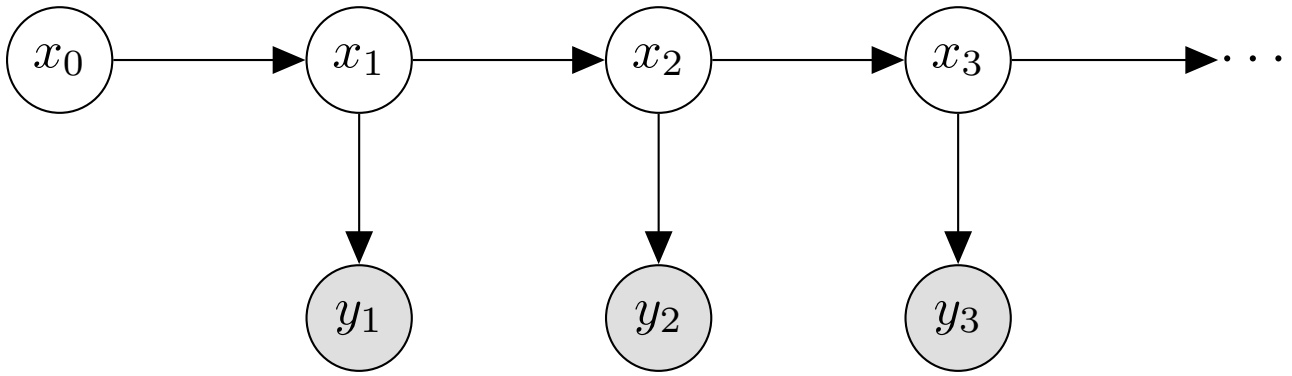
One Hidden Markov Model




```
(defquery hmm
  (let [init-dist (discrete [1 1 1])
        trans-dist (fn [s]
                     (cond
                       (= s 0) (discrete [0 1 1])
                       (= s 1) (discrete [0 0 1])
                       (= s 2) (dirac 2)))
        obs-dist (fn [s] (normal s 1))
        y-1 1
        y-2 1
        x-0 (sample init-dist)
        x-1 (sample (trans-dist x-0))
        x-2 (sample (trans-dist x-1))]

    (observe (obs-dist x-1) y-1)
    (observe (obs-dist x-2) y-2)
    (predict :x-0 x-0)
    (predict :x-1 x-1)
    (predict :x-2 x-2)))
```

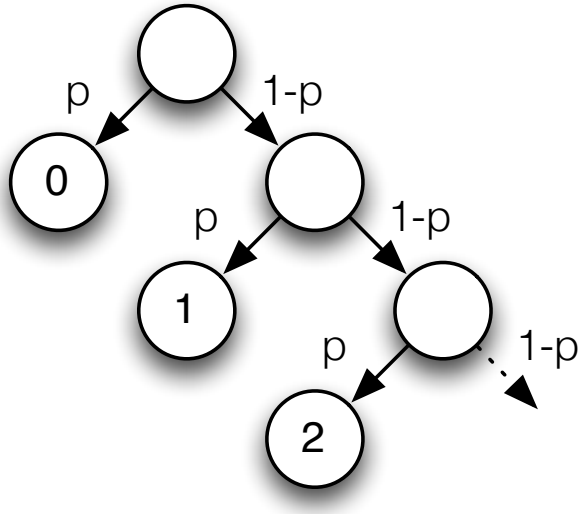
All Hidden Markov Models

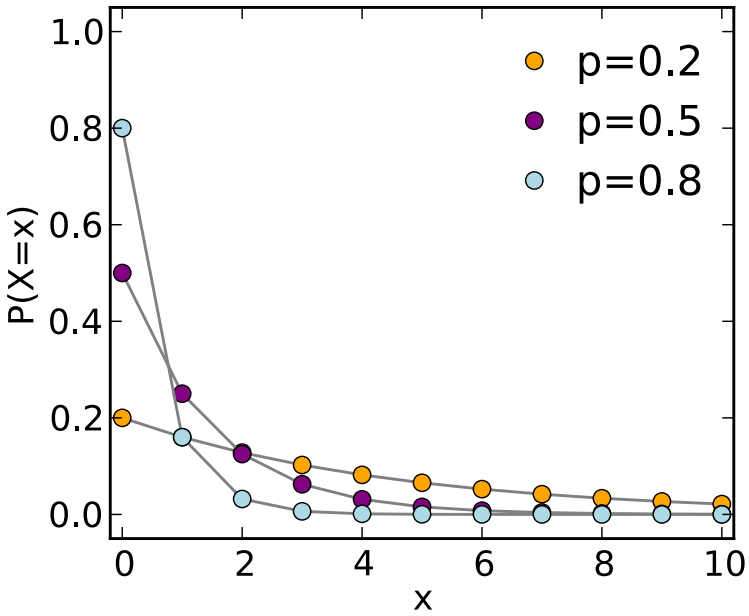
```
(defquery hmm
  [ys init-dist trans-dists obs-dists]
  (predict
    :x
    (reduce
      (fn [xs y]
        (let [x (sample (get trans-dists (peek xs)))]
          (observe (get obs-dists x) y)
          (conj xs x)))
      [(sample init-dist)]
      ys)))
```



An Unbounded Recursion

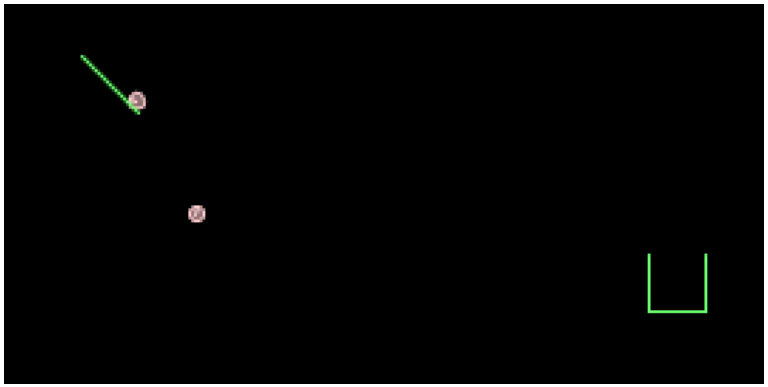
```
(defquery geometric [p]
  "geometric distribution"
  (let [dist (flip p)
        samp (loop [n 0]
               (if (sample dist)
                 n
                 (recur (+ n 1))))]
    (predict :x samp)))
```





Deterministic Simulation

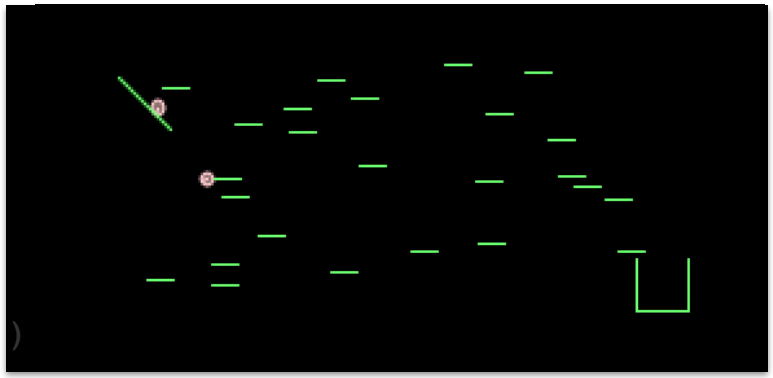
```
(defquery arrange-bumpers []
    (let [bumper-positions []

          ;; code to simulate the world
          world (create-world bumper-positions)
          end-world (simulate-world world)
          balls (:balls end-world)

          ;; how many balls entered the box?
          num-balls-in-box (balls-in-box end-world)]

      (predict :balls balls)
      (predict :num-balls-in-box num-balls-in-box)
      (predict :bumper-positions bumper-positions)))
```

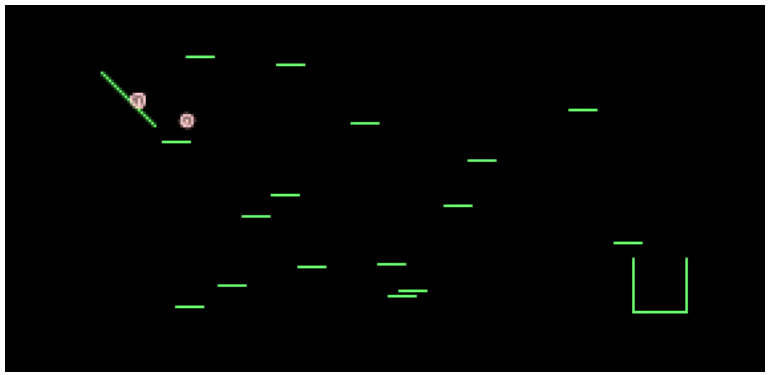


goal: “world” that puts ~20% of balls in box…

Stochastic Simulation

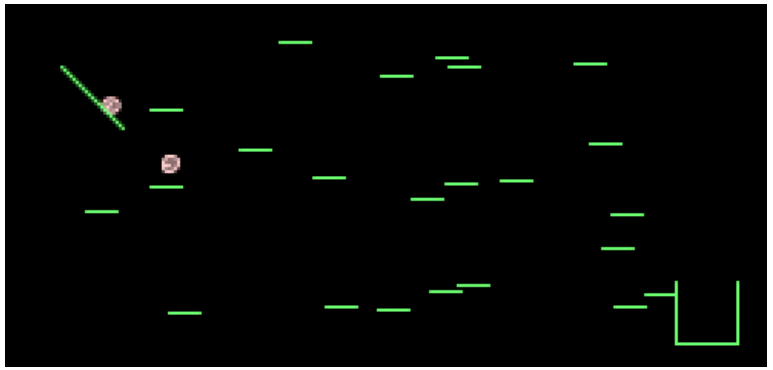
```
(defquery arrange-bumpers []
    (let [number-of-bumpers (sample (poisson 20))
          bumpydist (uniform-continuous 0 10)
          bumpxdist (uniform-continuous -5 14)
          bumper-positions (repeatedly
                            number-of-bumpers
                            #(vector (sample bumpxdist)
                                     (sample bumpydist)))

          ;; code to simulate the world
          world (create-world bumper-positions)
          end-world (simulate-world world)
          balls (:balls end-world)

          ;; how many balls entered the box?
          num-balls-in-box (balls-in-box end-world)]

      (predict :balls balls)
      (predict :num-balls-in-box num-balls-in-box)
      (predict :bumper-positions bumper-positions)))
```

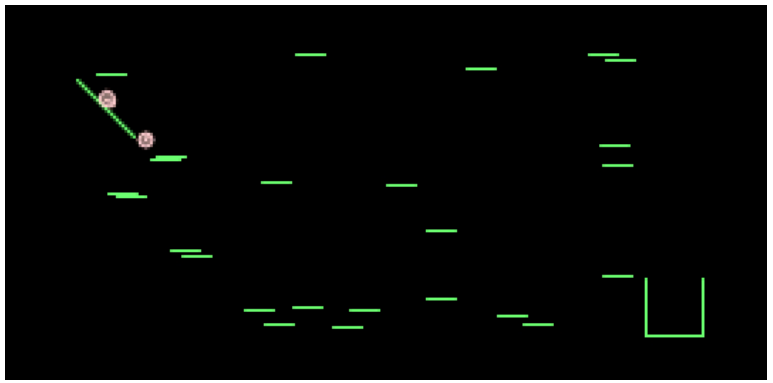


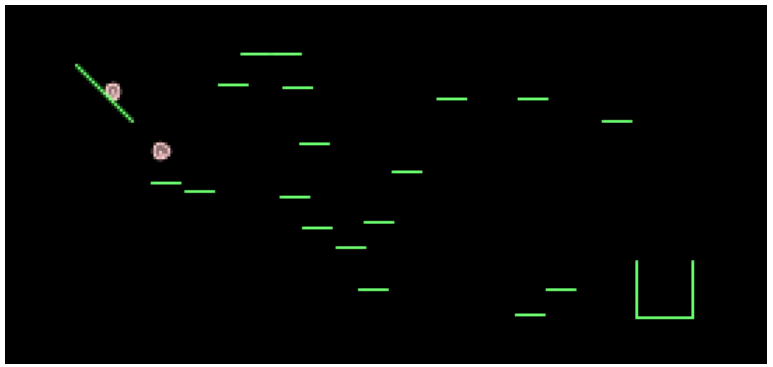


Constrained Stochastic Simulation

```
(defquery arrange-bumpers []
    (let [number-of-bumpers (sample (poisson 20))
          bumpydist (uniform-continuous 0 10)
          bumpxdist (uniform-continuous -5 14)
          bumper-positions (repeatedly
                            number-of-bumpers
                            #(vector (sample bumpxdist)
                                     (sample bumpydist)))

          ;; code to simulate the world
          world (create-world bumper-positions)
          end-world (simulate-world world)
          balls (:balls end-world)

          ;; how many balls entered the box?
          num-balls-in-box (balls-in-box end-world)

          obs-dist (normal 4 0.1)]

      (observe obs-dist num-balls-in-box)

      (predict :balls balls)
      (predict :num-balls-in-box num-balls-in-box)
      (predict :bumper-positions bumper-positions)))
```







A Hard Inference Problem

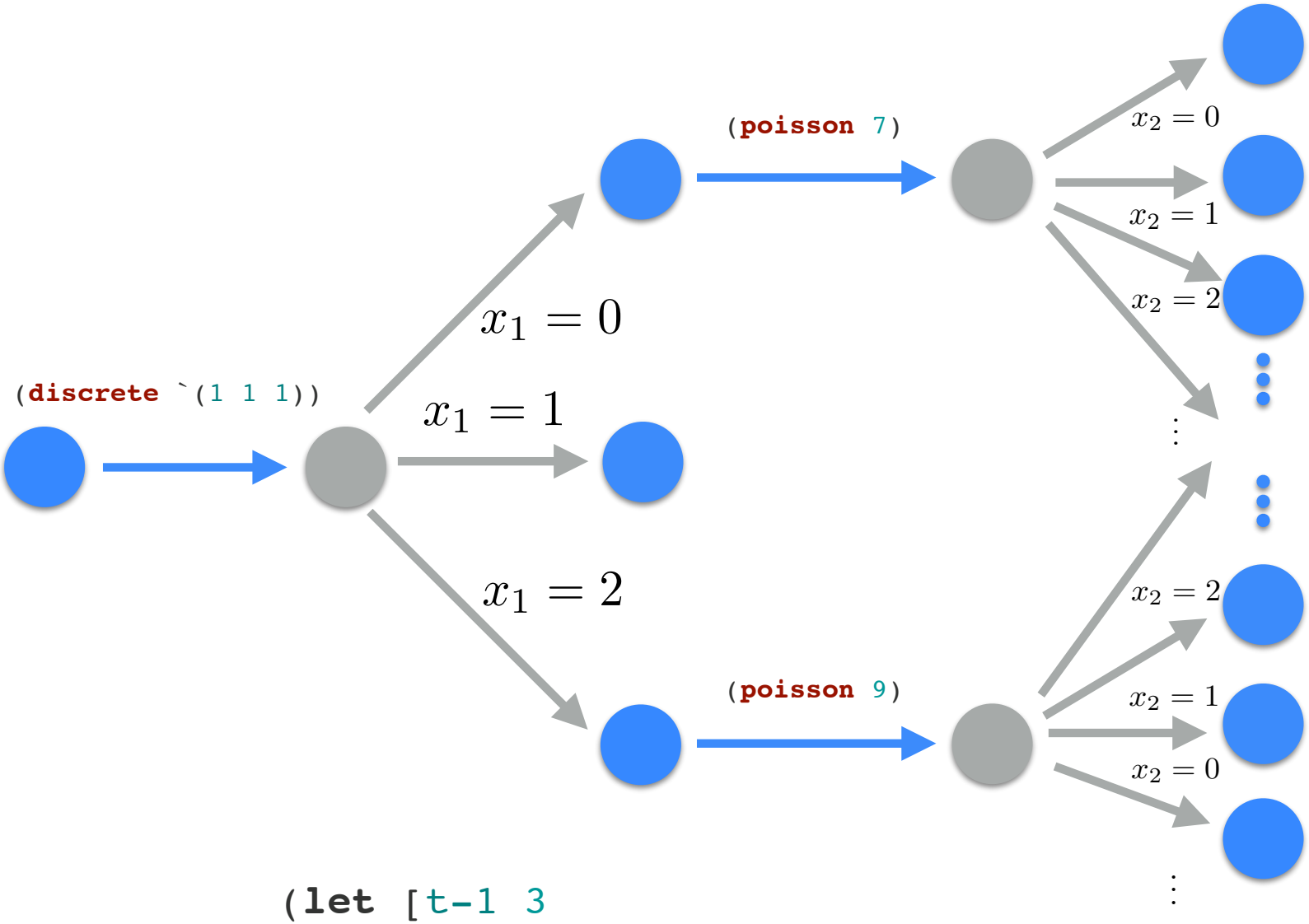
```
(defquery md5-inverse [L md5str]
    "conditional distribution of strings
     that map to the same MD5 hashed string"
    (let [mesg (sample (string-generative-model L))]
      (observe (dirac md5str) (md5 mesg))
      (predict :message mesg))))
```

An Inference Framework For Universal Probabilistic Programming Languages

The Gist

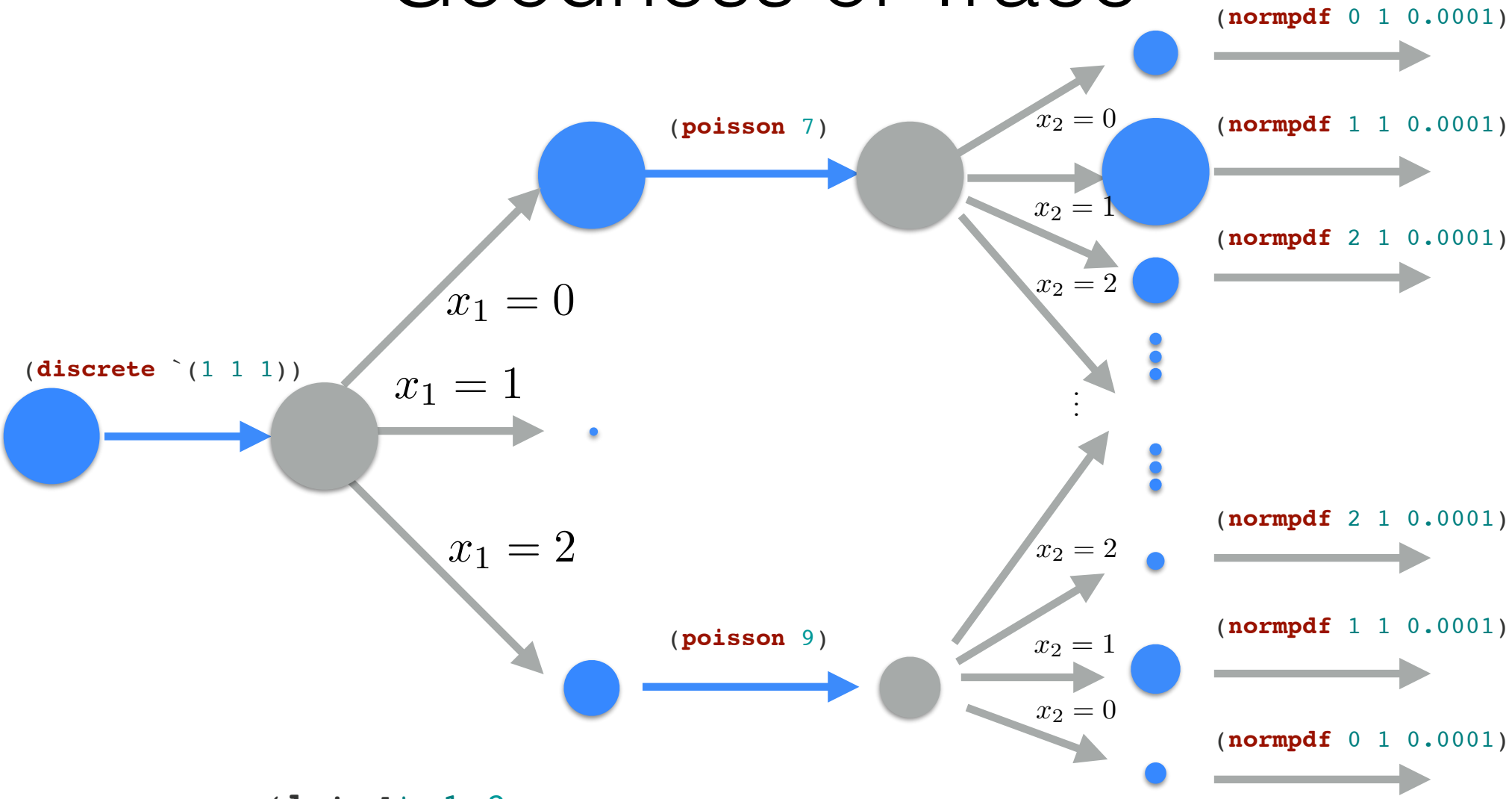
- Explore as many “traces” as possible, intelligently
  - Each trace contains all random choices made during the execution of a generative model
- Compute trace “goodness” (probability) as side-effect
- Combine weighted traces probabilistically coherently
- Report projection of posterior over traces

Traces




```
(let [t-1 3
      x-1 (sample (discrete (repeat t-1 1)))]
  (if (not= x-1 1)
    (let [t-2 (+ x-1 7)
          x-2 (sample (poisson t-2))])))
```

Goodness of Trace




```
(let [t-1 3
      x-1 (sample (discrete (repeat t-1 1)))]
  (if (not= x-1 1)
    (let [t-2 (+ x-1 7)
          x-2 (sample (poisson t-2))]
      (observe (gaussian x-2 0.0001) 1))))
```

Trace

- Sequence of N **observe**'s

$$\{(g_i, \phi_i, y_i)\}_{i=1}^{N}$$

- Sequence of M **sample**'s

$$\{(f_j, \theta_j)\}_{j=1}^{M}$$

- Sequence of M sampled values

$$\{x_j\}_{j=1}^{M}$$

- Conditioned on these sampled values the entire computation is *deterministic*

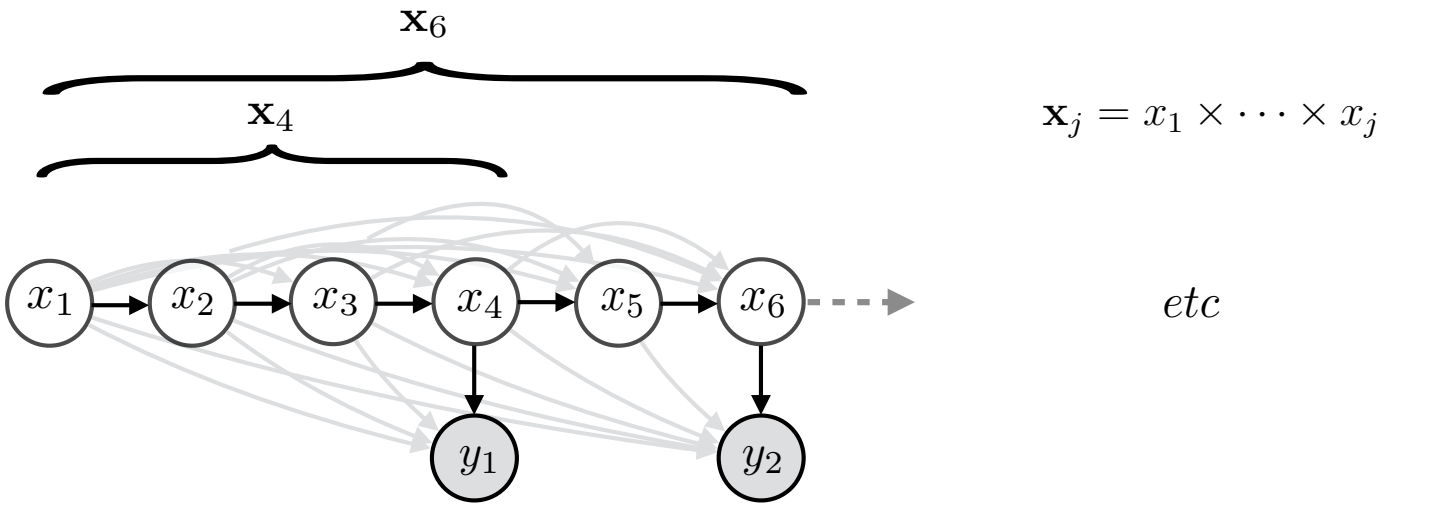
Trace Probability

- Defined as (up to a normalization constant)

$$\gamma(\mathbf{x}) \triangleq p(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{N} g_i(y_i|\phi_i) \prod_{j=1}^{M} f_j(x_j|\theta_j)$$

- Hides true dependency structure

$$\gamma(\mathbf{x}) = p(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{N} \tilde{g}_i(\mathbf{x}_{n_i}) \left(y_i \middle| \tilde{\phi}_i(\mathbf{x}_{n_i}) \right) \prod_{j=1}^{M} \tilde{f}_j(\mathbf{x}_{j-1}) \left(x_j \middle| \tilde{\theta}_j(\mathbf{x}_{j-1}) \right)$$



Inference Goal

- Posterior over traces

$$\pi(\mathbf{x}) \triangleq p(\mathbf{x}|\mathbf{y}) = \frac{\gamma(\mathbf{x})}{Z} \qquad Z = p(\mathbf{y}) = \int \gamma(\mathbf{x}) d\mathbf{x}$$

- Output

$$\mathbb{E}[z] = \mathbb{E}[Q(\mathbf{x})] = \int Q(\mathbf{x})\pi(\mathbf{x})d\mathbf{x} = \frac{1}{Z}\int Q(\mathbf{x})\frac{\gamma(\mathbf{x})}{q(\mathbf{x})}q(\mathbf{x})d\mathbf{x}$$

Three Base Algorithms

- Likelihood Weighting
- Sequential Monte Carlo
- Metropolis Hastings

Likelihood Weighting

- Run K independent copies of program simulating from the prior

$$q(\mathbf{x}^k) = \prod_{j=1}^{M^k} f_j(x_j^k | \theta_j^k)$$

- Accumulate *unnormalized* weights (likelihoods)

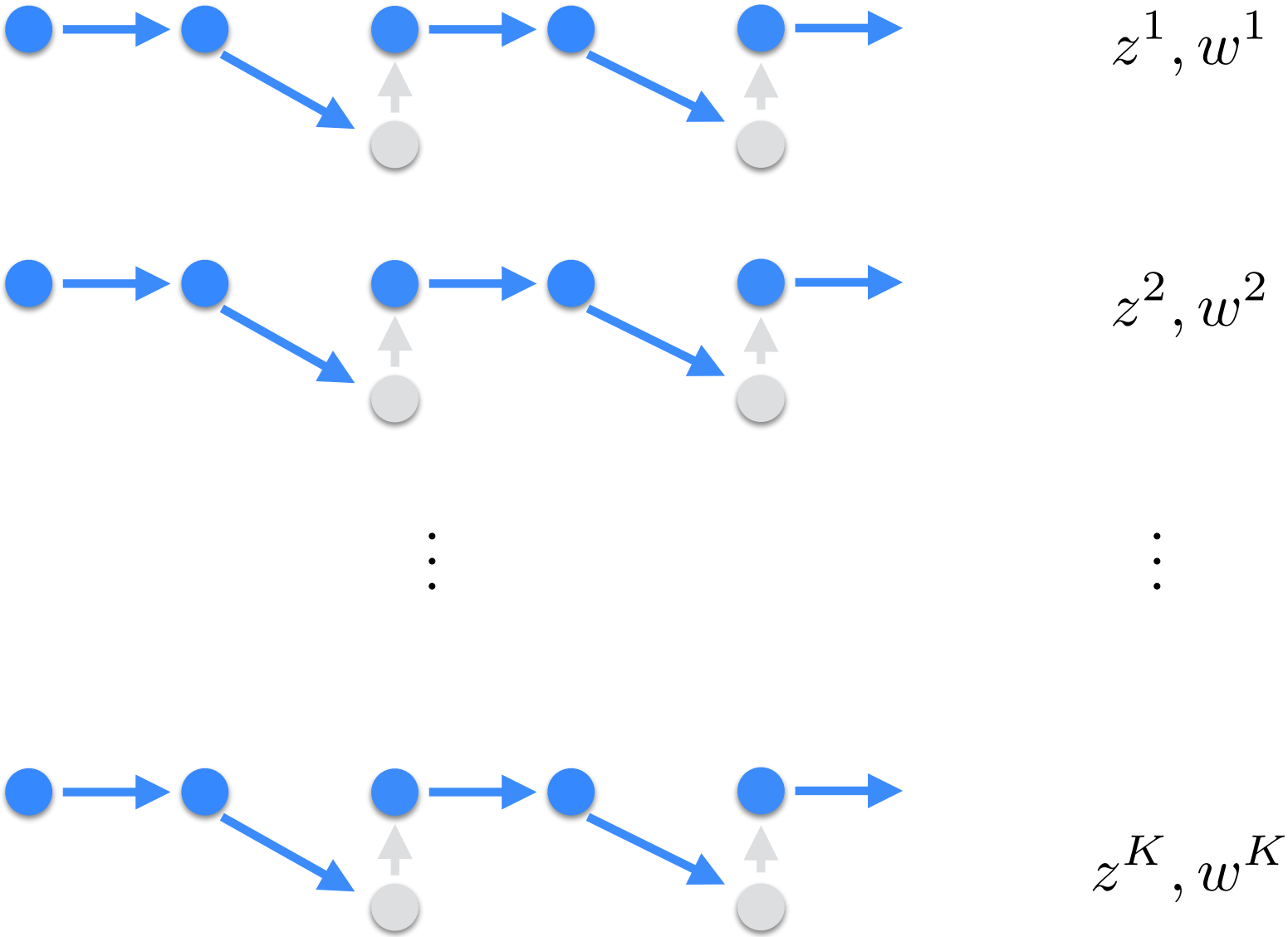
$$w(\mathbf{x}^k) = \frac{\gamma(\mathbf{x}^k)}{q(\mathbf{x}^k)} = \prod_{i=1}^{N^k} g_i^k(y_i^k | \phi_i^k)$$

- Use in approximate (Monte Carlo) integration

$$W^k = \frac{w(\mathbf{x}^k)}{\sum_{\ell=1}^K w(\mathbf{x}^\ell)} \qquad \widehat{\mathbb{E}}_\pi[Q(\mathbf{x})] = \sum_{k=1}^K W^k Q(\mathbf{x}^k)$$

BLOG default inference engine:
http://bayesianlogic.github.io/pages/users-manual.html

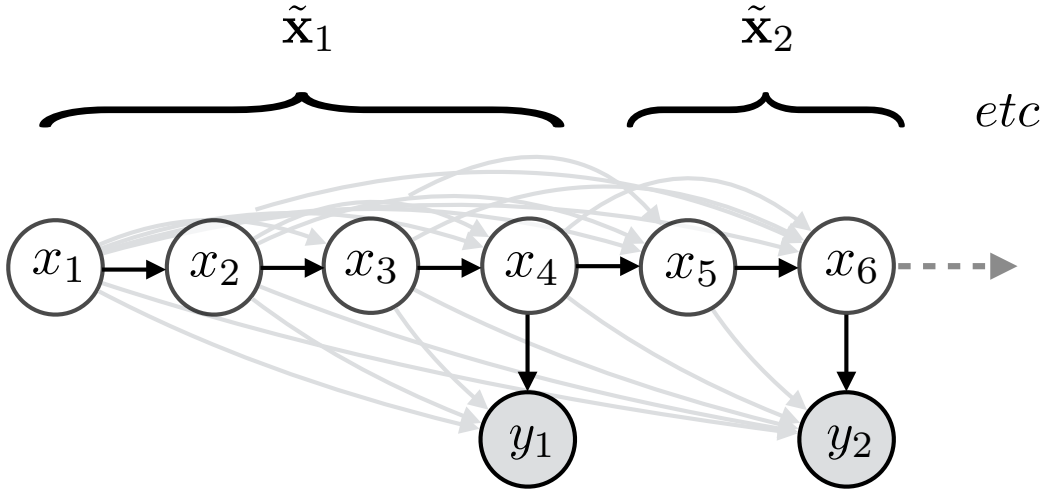
Likelihood Weighting Schematic



Sequential Monte Carlo

- Notation

$$\tilde{\mathbf{x}}_{1:n} = \tilde{\mathbf{x}}_1 \times \cdots \times \tilde{\mathbf{x}}_n$$



- Incrementalized joint

$$\gamma_n(\tilde{\mathbf{x}}_{1:n}) = \prod_{n=1}^{N} g(y_n|\tilde{\mathbf{x}}_{1:n}) p(\tilde{\mathbf{x}}_n|\tilde{\mathbf{x}}_{1:n-1})$$

- Incrementalized target

$$\pi_n(\tilde{\mathbf{x}}_{1:n}) = \frac{1}{Z_n}\gamma_n(\tilde{\mathbf{x}}_{1:n})$$

SMC for Probabilistic Programming

Want samples from

$$\pi_n(\tilde{\mathbf{x}}_{1:n}) \propto p(y_n|\tilde{\mathbf{x}}_{1:n})p(\tilde{\mathbf{x}}_n|\tilde{\mathbf{x}}_{1:n-1})\pi_{n-1}(\tilde{\mathbf{x}}_{1:n-1})$$

Have a sample-based approximation to

$$\hat{\pi}_{n-1}(\tilde{\mathbf{x}}_{1:n-1}) \triangleq \sum_{k=1}^{K} W_{n-1}^k \delta_{\tilde{\mathbf{x}}_{1:n-1}^k}(\tilde{\mathbf{x}}_{1:n-1})$$

Sample from

$$\tilde{\mathbf{x}}_{1:n-1}^{a_{n-1}^k} \sim \hat{\pi}_{n-1}(\tilde{\mathbf{x}}_{1:n-1}) \qquad \tilde{\mathbf{x}}_n^k|\tilde{\mathbf{x}}_{1:n-1}^{a_{n-1}^k} \sim p(\tilde{\mathbf{x}}_n|\tilde{\mathbf{x}}_{1:n-1}^{a_{n-1}^k})$$

$$\tilde{\mathbf{x}}_{1:n}^k = \tilde{\mathbf{x}}_{1:n-1}^{a_{n-1}^k} \times \tilde{\mathbf{x}}_n^k$$

Importance weight by

$$w(\tilde{\mathbf{x}}_{1:n}^k) = p(y_n|\tilde{\mathbf{x}}_{1:n}^k) = g_n^k(y_n|\tilde{\mathbf{x}}_{1:n}^k) \qquad W_n^k \triangleq \frac{w(\tilde{\mathbf{x}}_{1:n}^k)}{\sum_{k'=1}^{K} w(\tilde{\mathbf{x}}_{1:n}^{k'})}$$

Wood, van de Meent, and Mansinghka “A New Approach to Probabilistic Programming Inference” AISTATS 2014
Paige and Wood “A Compilation Target for Probabilistic Programming Languages” ICML 2014

SMC for Probabilistic Programming

Want samples from

$$\pi_n(\tilde{\mathbf{x}}_{1:n}) \propto p(y_n|\tilde{\mathbf{x}}_{1:n})p(\tilde{\mathbf{x}}_n|\tilde{\mathbf{x}}_{1:n-1})\pi_{n-1}(\tilde{\mathbf{x}}_{1:n-1})$$

Have a sample-based approximation to

$$\hat{\pi}_{n-1}(\tilde{\mathbf{x}}_{1:n-1}) \triangleq \sum_{k=1}^{K} W_{n-1}^k \delta_{\tilde{\mathbf{x}}_{1:n-1}^k}(\tilde{\mathbf{x}}_{1:n-1})$$

Sample from

$$\tilde{\mathbf{x}}_{1:n-1}^{a_{n-1}^k} \sim \hat{\pi}_{n-1}(\tilde{\mathbf{x}}_{1:n-1}) \qquad \tilde{\mathbf{x}}_n^k|\tilde{\mathbf{x}}_{1:n-1}^{a_{n-1}^k} \sim p(\tilde{\mathbf{x}}_n|\tilde{\mathbf{x}}_{1:n-1}^{a_{n-1}^k})$$

$$\tilde{\mathbf{x}}_{1:n}^k = \tilde{\mathbf{x}}_{1:n-1}^{a_{n-1}^k} \times \tilde{\mathbf{x}}_n^k$$

Importance weight by

$$w(\tilde{\mathbf{x}}_{1:n}^k) = p(y_n|\tilde{\mathbf{x}}_{1:n}^k) = g_n^k(y_n|\tilde{\mathbf{x}}_{1:n}^k) \qquad W_n^k \triangleq \frac{w(\tilde{\mathbf{x}}_{1:n}^k)}{\sum_{k'=1}^{K} w(\tilde{\mathbf{x}}_{1:n}^{k'})}$$

Wood, van de Meent, and Mansinghka “A New Approach to Probabilistic Programming Inference” AISTATS 2014
Paige and Wood “A Compilation Target for Probabilistic Programming Languages” ICML 2014

SMC for Probabilistic Programming

Want samples from

$$\pi_n(\tilde{\mathbf{x}}_{1:n}) \propto p(y_n|\tilde{\mathbf{x}}_{1:n})p(\tilde{\mathbf{x}}_n|\tilde{\mathbf{x}}_{1:n-1})\pi_{n-1}(\tilde{\mathbf{x}}_{1:n-1})$$

Have a sample-based approximation to

$$\hat{\pi}_{n-1}(\tilde{\mathbf{x}}_{1:n-1}) \triangleq \sum_{k=1}^{K} W_{n-1}^k \delta_{\tilde{\mathbf{x}}_{1:n-1}^k}(\tilde{\mathbf{x}}_{1:n-1})$$

Sample from

$$\tilde{\mathbf{x}}_{1:n-1}^{a_{n-1}^k} \sim \hat{\pi}_{n-1}(\tilde{\mathbf{x}}_{1:n-1}) \qquad \tilde{\mathbf{x}}_n^k|\tilde{\mathbf{x}}_{1:n-1}^{a_{n-1}^k} \sim p(\tilde{\mathbf{x}}_n|\tilde{\mathbf{x}}_{1:n-1}^{a_{n-1}^k})$$

$$\tilde{\mathbf{x}}_{1:n}^k = \tilde{\mathbf{x}}_{1:n-1}^{a_{n-1}^k} \times \tilde{\mathbf{x}}_n^k$$

Importance weight by

$$w(\tilde{\mathbf{x}}_{1:n}^k) = p(y_n|\tilde{\mathbf{x}}_{1:n}^k) = g_n^k(y_n|\tilde{\mathbf{x}}_{1:n}^k) \qquad W_n^k \triangleq \frac{w(\tilde{\mathbf{x}}_{1:n}^k)}{\sum_{k'=1}^{K} w(\tilde{\mathbf{x}}_{1:n}^{k'})}$$

Wood, van de Meent, and Mansinghka “A New Approach to Probabilistic Programming Inference” AISTATS 2014
Paige and Wood “A Compilation Target for Probabilistic Programming Languages” ICML 2014

SMC for Probabilistic Programming

Want samples from

$$\pi_n(\tilde{\mathbf{x}}_{1:n}) \propto p(y_n|\tilde{\mathbf{x}}_{1:n})p(\tilde{\mathbf{x}}_n|\tilde{\mathbf{x}}_{1:n-1})\pi_{n-1}(\tilde{\mathbf{x}}_{1:n-1})$$

Have a sample-based approximation to

$$\hat{\pi}_{n-1}(\tilde{\mathbf{x}}_{1:n-1}) \triangleq \sum_{k=1}^{K} W_{n-1}^k \delta_{\tilde{\mathbf{x}}_{1:n-1}^k}(\tilde{\mathbf{x}}_{1:n-1})$$

Sample from

$$\tilde{\mathbf{x}}_{1:n-1}^{a_{n-1}^k} \sim \hat{\pi}_{n-1}(\tilde{\mathbf{x}}_{1:n-1}) \qquad \tilde{\mathbf{x}}_n^k|\tilde{\mathbf{x}}_{1:n-1}^{a_{n-1}^k} \sim p(\tilde{\mathbf{x}}_n|\tilde{\mathbf{x}}_{1:n-1}^{a_{n-1}^k})$$

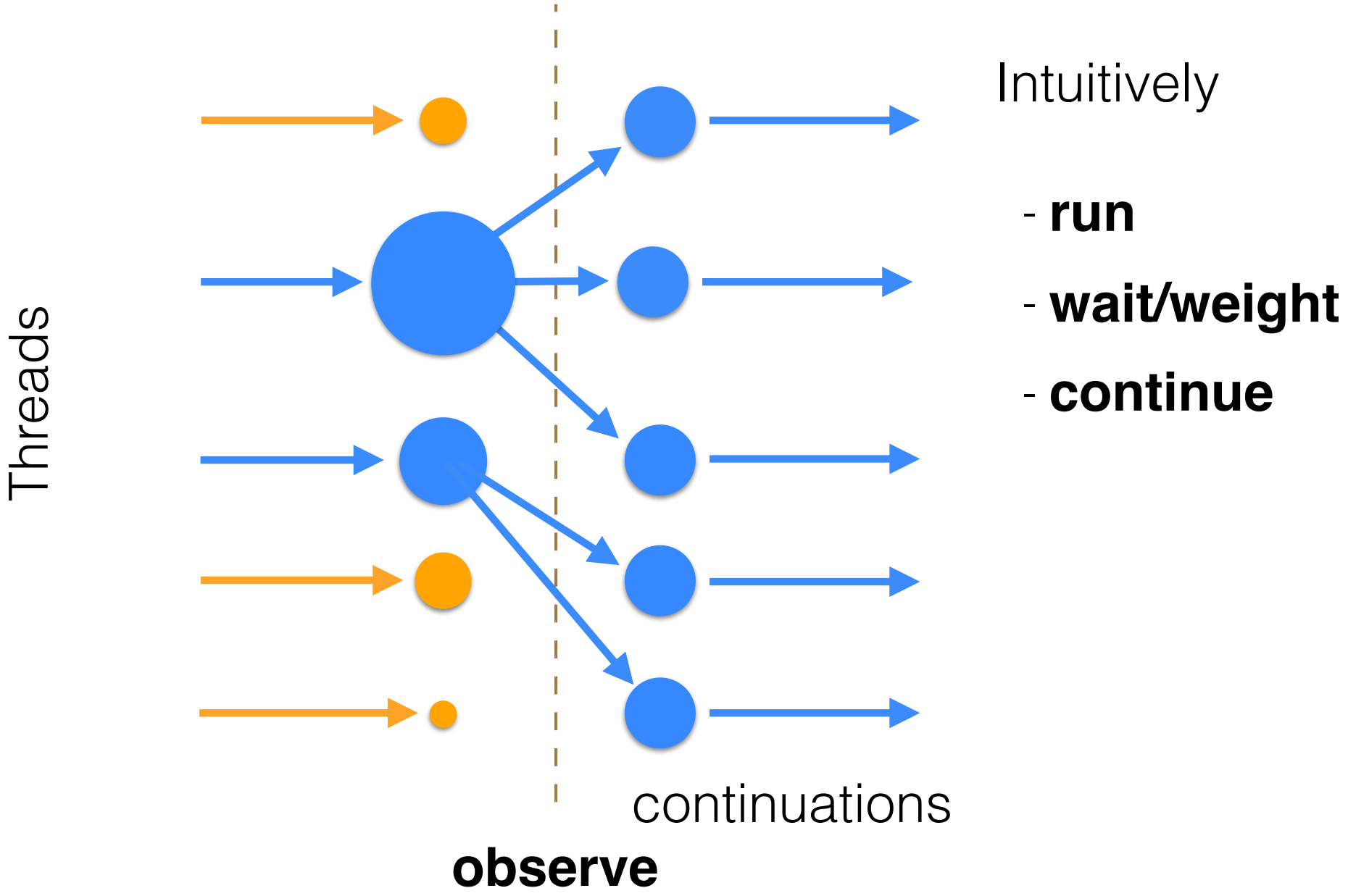
$$\tilde{\mathbf{x}}_{1:n}^k = \tilde{\mathbf{x}}_{1:n-1}^{a_{n-1}^k} \times \tilde{\mathbf{x}}_n^k$$

Importance weight by

$$w(\tilde{\mathbf{x}}_{1:n}^k) = p(y_n|\tilde{\mathbf{x}}_{1:n}^k) = g_n^k(y_n|\tilde{\mathbf{x}}_{1:n}^k) \qquad W_n^k \triangleq \frac{w(\tilde{\mathbf{x}}_{1:n}^k)}{\sum_{k'=1}^{K} w(\tilde{\mathbf{x}}_{1:n}^{k'})}$$

Wood, van de Meent, and Mansinghka “A New Approach to Probabilistic Programming Inference” AISTATS 2014
Paige and Wood “A Compilation Target for Probabilistic Programming Languages” ICML 2014

SMC Schematic



Metropolis Hastings = “Single Site” MCMC = LMH

Posterior distribution of execution traces is proportional to trace score with observed values plugged in

$$\gamma(\mathbf{x}) \triangleq p(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{N} g_i(y_i|\phi_i) \prod_{j=1}^{M} f_j(x_j|\theta_j) \qquad \pi(\mathbf{x}) \triangleq p(\mathbf{x}|\mathbf{y}) = \frac{\gamma(\mathbf{x})}{Z}$$

Metropolis-Hastings acceptance rule

$$\alpha = \min\left(1, \frac{\pi(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{\pi(\mathbf{x})q(\mathbf{x}'|\mathbf{x})}\right)$$

Need proposal

Milch and Russell “General-Purpose MCMC Inference over Relational Structures.” UAI 2006.
Goodman, Mansinghka, Roy, Bonawitz, and Tenenbaum “Church: a language for generative models.” UAI 2008.
Wingate, Stuhlmüller, Goodman “Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation” AISTATS 2011

LMH Proposal

Probability of new part of proposed execution trace

$$q(\mathbf{x}'|\mathbf{x}^s) = \frac{1}{M^s}\kappa(x'_\ell|x^s_\ell)\prod_{j=\ell+1}^{M'} f'_j(x'_j|\theta'_j)$$

Number of samples in original trace

LMH Acceptance Ratio

"Single site update" = sample from the prior = run program forward

$$\kappa(x'_m|x_m) = f_m(x'_m|\theta_m), \theta_m = \theta'_m$$

MH acceptance ratio

Probability of original trace continuation restarting proposal trace at m[th] sample

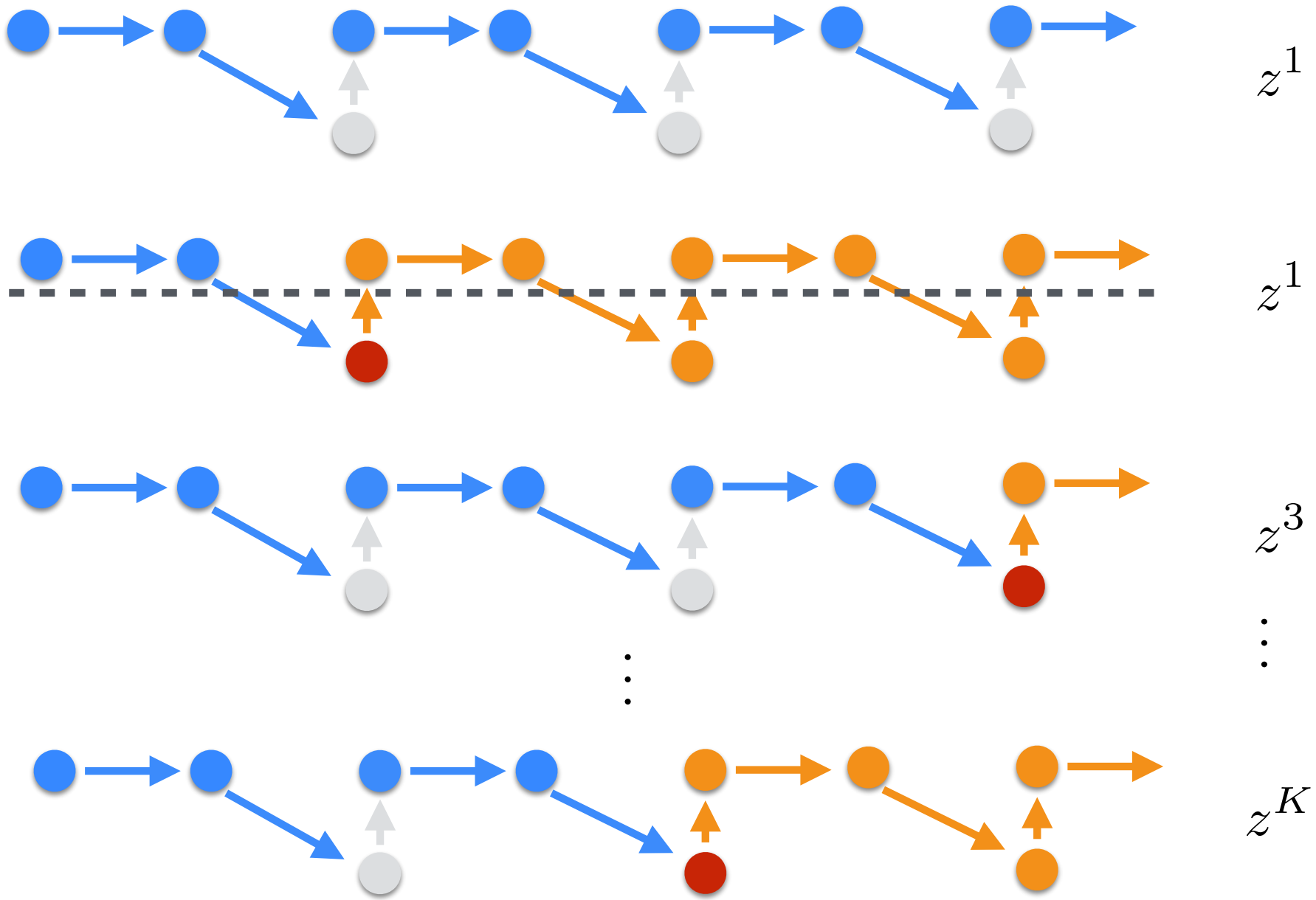
Number of sample statements in original trace

$$\alpha = \min\left(1, \frac{\gamma(\mathbf{x}')M\prod_{j=m}^{M} f_j(x_j|\theta_j)}{\gamma(\mathbf{x})M'\prod_{j=m}^{M'} f'_j(x'_j|\theta'_j)}\right)$$

Number of sample statements in new trace

Probability of proposal trace continuation restarting original trace at m[th] sample

LMH Schematic

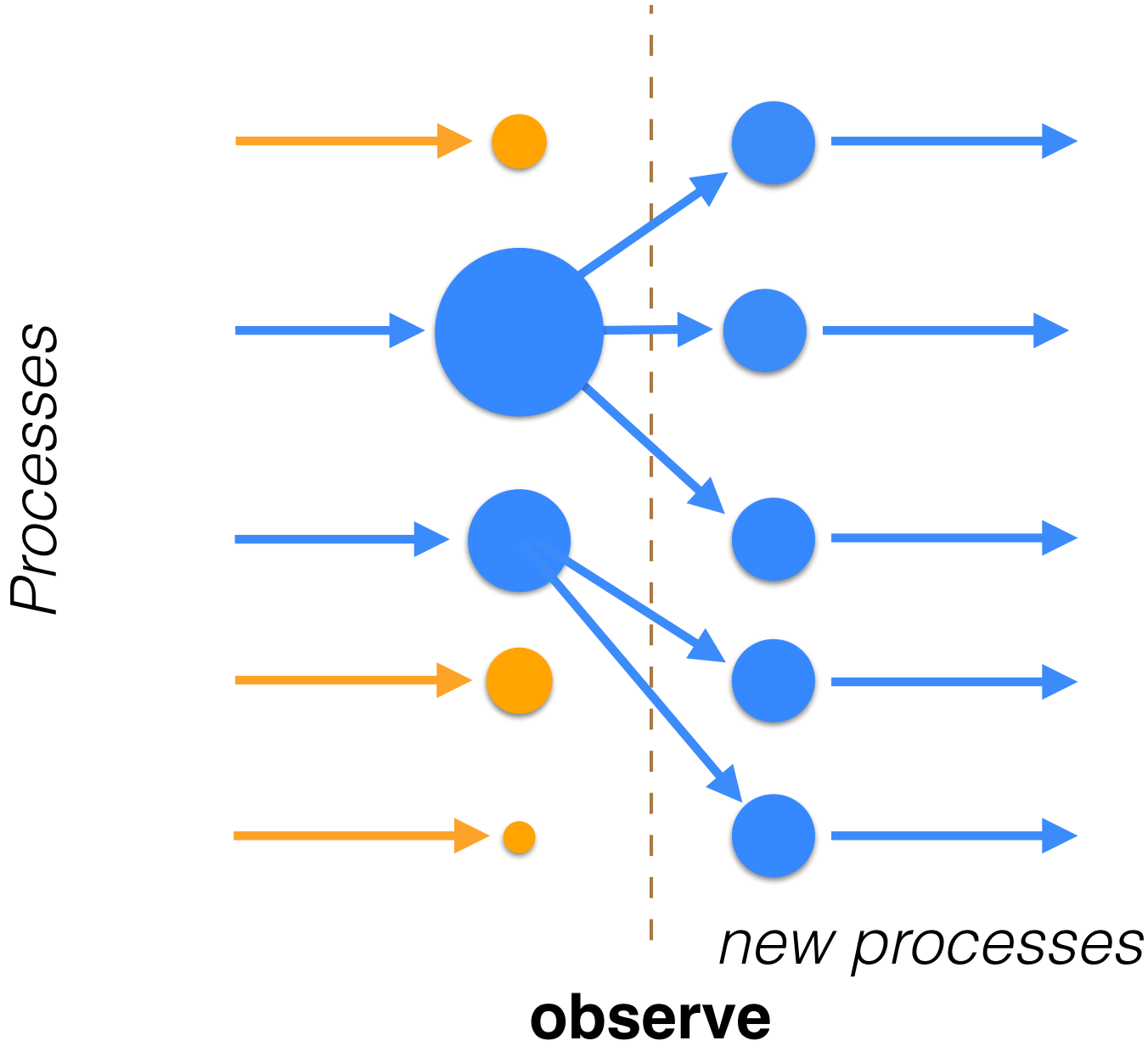


Implementation Strategy

- Interpreted
  - Interpreter tracks side effects and directs control flow for inference
- Compiled
  - Leverages existing compiler infrastructure
  - Can only exert control over flow from *within* function calls
    - e.g. sample, observe, predict

Wingate, Stuhlmüller, Goodman “Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation” AISTATS 2011
Paige and Wood “A Compilation Target for Probabilistic Programming Languages” ICML 2014

Probabilistic C



Intuitively

- **run**
- **wait/weight**
- ***fork***

Paige and Wood “A Compilation Target for Probabilistic Programming Languages” ICML 2014

Compilation

- LW
  - sample: inject random values
- LMH
  - catalog all random choices and compare traces by running new future
- SMC - run multiple independent futures without corrupting past
    - Must have control over the “rest of the computation”
- No longer control execution, can only interrupt and exert control at key points
    - Start, Sample, Observe, Predict, Terminate

Continuations

- A *continuation* is a function that encapsulates the “rest of the computation”
- A Continuation Passing Style (CPS) transformation rewrites programs so
  - no function ever returns
  - every function takes an extra argument, a function called the *continuation*
- Standard programming language technique
- No limitations

Friedman and Wand. “Essentials of programming languages.” MIT press, 2008.
Fischer, Kiselyov, and Shan “Purely functional lazy non-deterministic programming” ACM Sigplan 2009
Goodman and Stuhlmüller http://dippl.org/ 2014
Tolpin https://bitbucket.org/probprog/anglican/ 2014

Example CPS Transformation

```
;; Standard Clojure:
(println (+ (* 2 3) 4))

;; CPS transformed:
(*& 2 3 (fn [x] (+& x 4 println)))
```


```
;; CPS-transformed "primitives"
(defn +& [a b k] (k (+ a b)))
(defn *& [a b k] (k (* a b)))
```

CPS Explicitly Linearizes Execution

```
(defn pythag&
  "compute sqrt(x^2 + y^2)"
  [x y k]
  (square& x
           (fn [xx]
             (square& y
                      (fn [yy]
                        (+& xx yy
                            (fn [xxyy]
                              (sqrt& xxyy k)))))))))
```

$$xx = x^2$$

$$yy = y^2$$

$$xxyy = xx + yy$$

$$\cdot = \sqrt{xxyy}$$

- Compiling to a pure language with lexical scoping ensures
  - A. variables needed in subsequent computation are bound in the environment
  - B. can't be modified by multiple calls to the continuation function

Anglican Programs

```
(defquery flip-example [outcome]
  (let [p (sample (uniform-continuous 0 1))]
    (observe (flip p) outcome)
    (predict :p p))
```

```
(let [u (uniform-continuous 0 1)

      p (sample u)

      dist (flip p)]

      (observe dist outcome)

      (predict :p p))
```

Anglican

Anglican “linearized”

Are “Compiled” to Native CPS-Clojure

```
(defn flip-query& [outcome k1]
  (uniform-continuous& 0 1 ◄·····► (let [u (uniform-continuous 0 1)
    (fn [dist1]
      (sample& dist1 ◄·····► p (sample u)
        (fn [p] ((fn [p k2]
                   (flip& p ◄·····► dist (flip p)]
                     (fn [dist2]
                       (observe& dist2 outcome ◄·····► (observe dist outcome)
                         (fn []
                           (predict& :p p k2)))))) ◄·····► (predict :p p))
                 p k1))))))

;; CPS-ed distribution constructors
(defn uniform-continuous& [a b k]
  (k (uniform-continuous a b)))

(defn flip& [p k]
  (k (flip p)))
```

Clojure

Anglican “linearized”

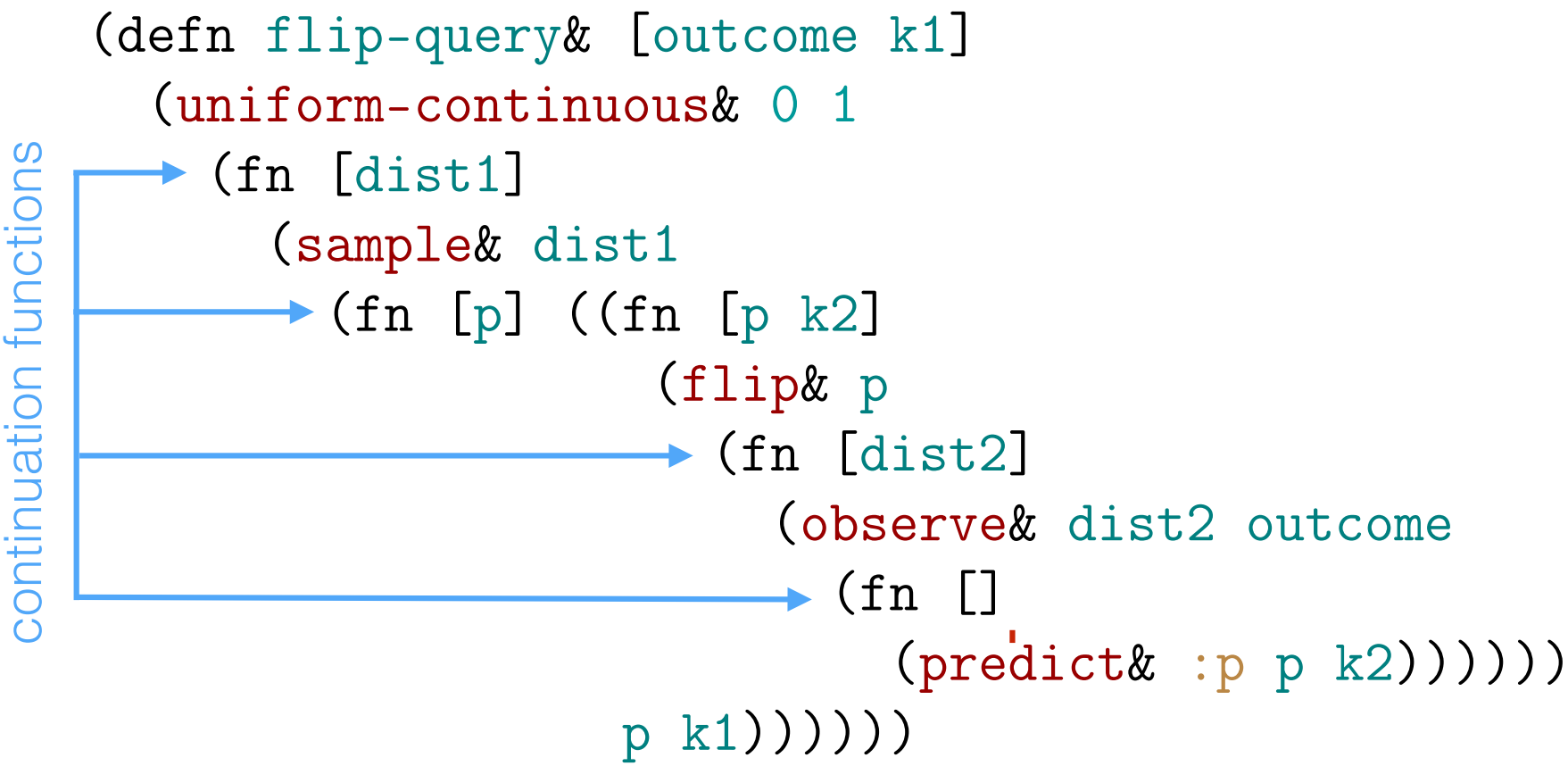
Are “Compiled” to Native CPS-Clojure

```
(defn flip-query& [outcome k1]
  (uniform-continuous& 0 1 ⟷ (let [u (uniform-continuous 0 1)
    (fn [dist1]
      (sample& dist1 ⟷ p (sample u)
        (fn [p] ((fn [p k2]
                   (flip& p ⟷ dist (flip p)]
                     (fn [dist2]
                       (observe& dist2 outcome ⟷ (observe dist outcome)
                         (fn []
                           (predict& :p p k2)))))) ⟷ (predict :p p))
                 p k1))))))

;; CPS-ed distribution constructors
(defn uniform-continuous& [a b k]
  (k (uniform-continuous a b)))

(defn flip& [p k]
  (k (flip p)))
```

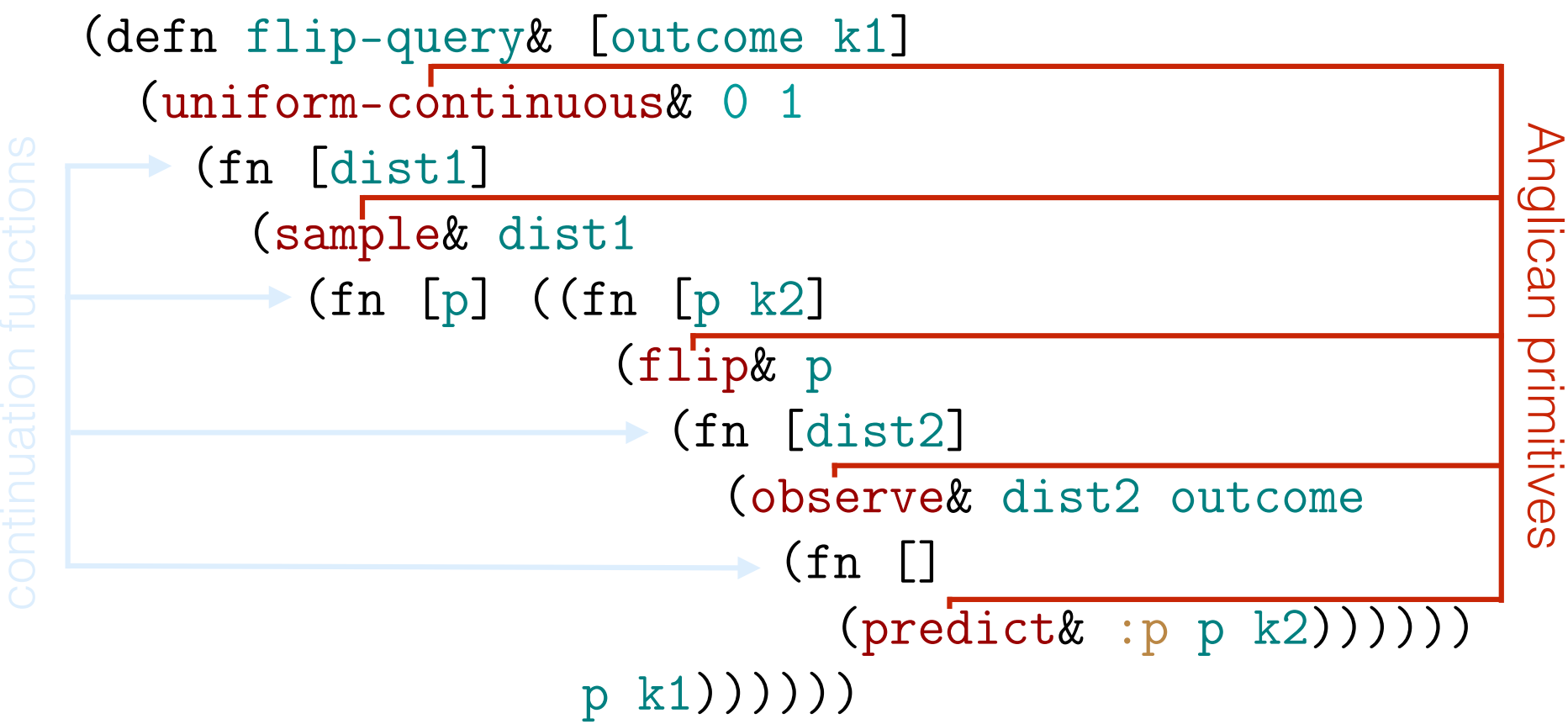
Clojure

Anglican “linearized”

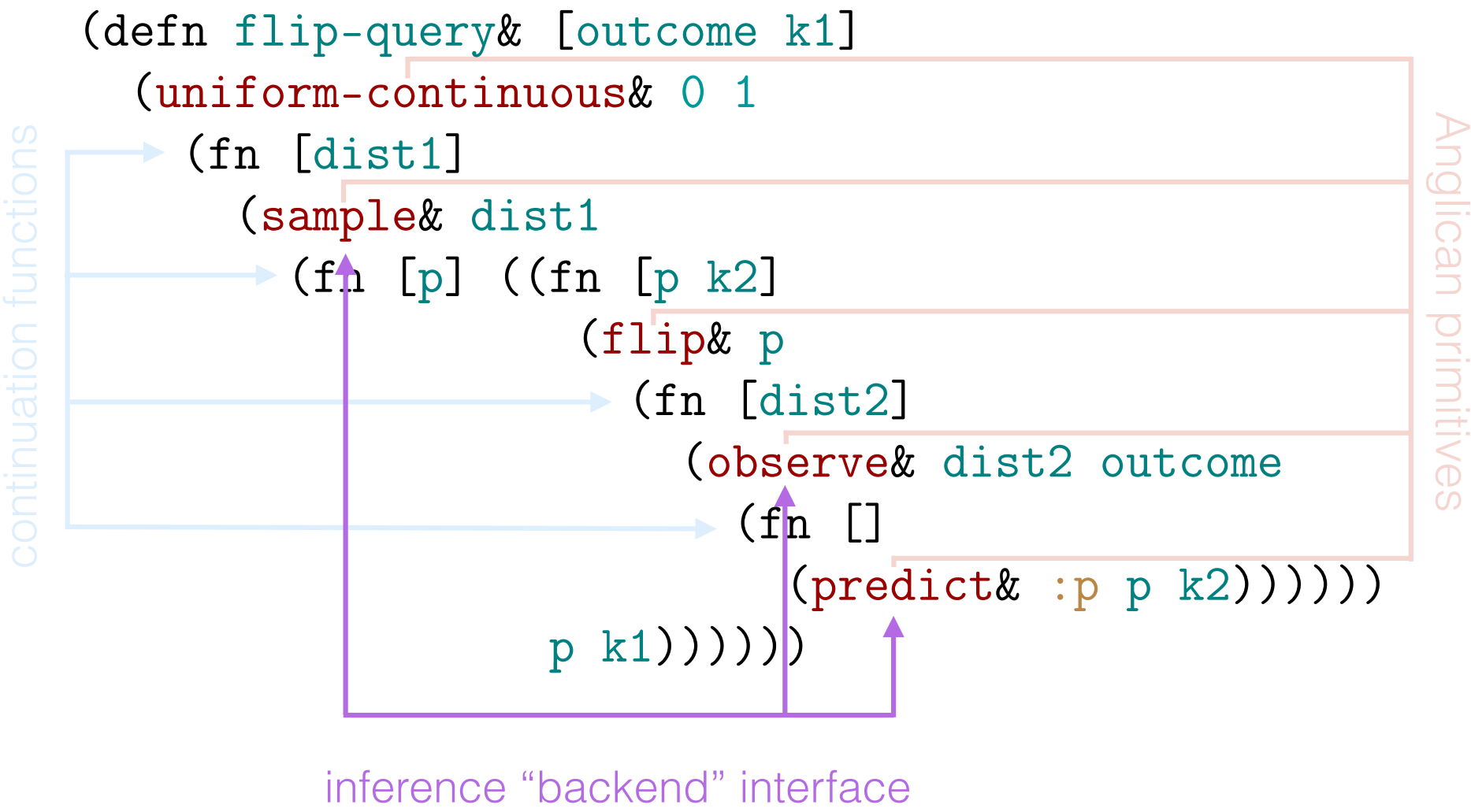
Explicit Functional Form for “Rest of Program”



Interruptible



Controllable



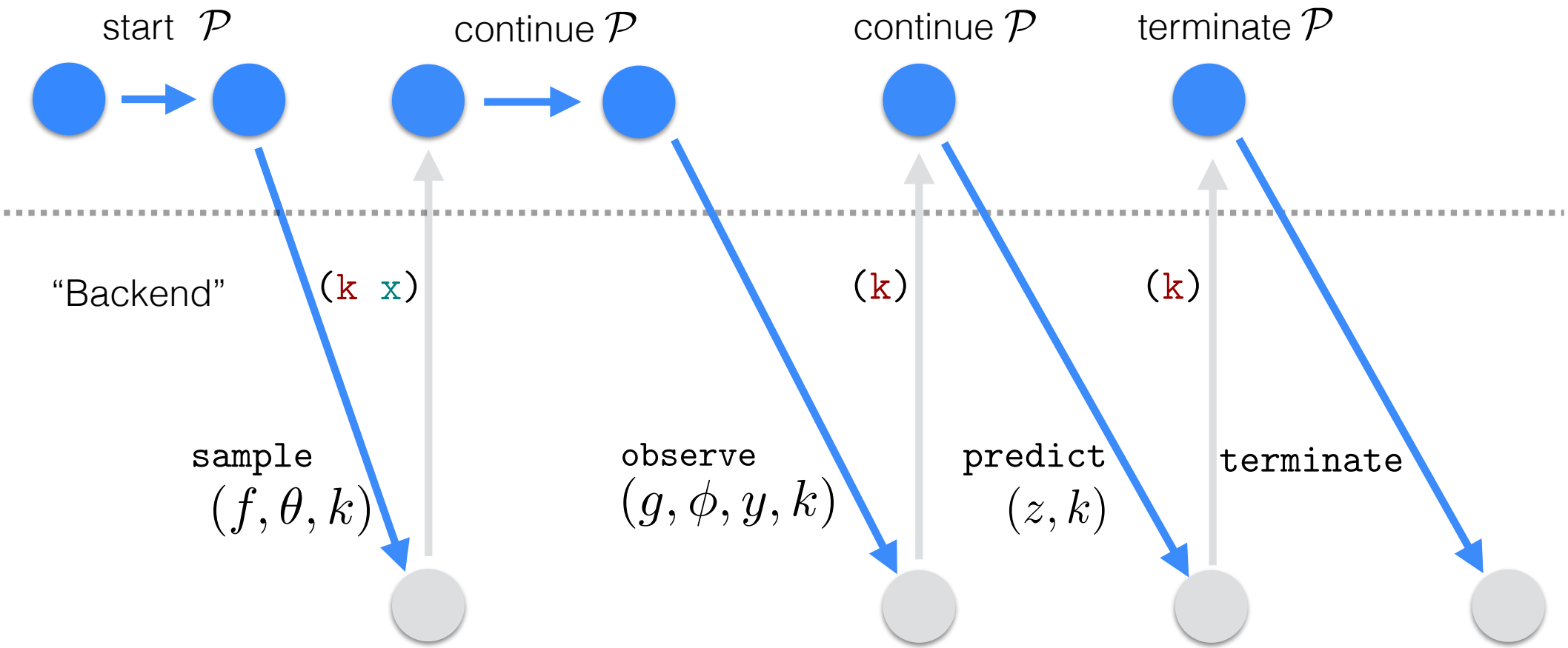
webPPL CPS compiles to pure functional Javascript

Inference “Backend”

```
(defn sample& [dist k]
  ;; [ ALGORITHM-SPECIFIC IMPLEMENTATION HERE ]
  ;; Pass the sampled value to the continuation
  (k (sample dist)))

(defn observe& [dist value k]
  (println "log-weight =" (observe dist value))
  ;; [ ALGORITHM-SPECIFIC IMPLEMENTATION HERE ]
  ;; Call continuation with no arguments
  (k))

(defn predict& [label value k]
  ;; [ ALGORITHM-SPECIFIC IMPLEMENTATION HERE ]
  (k label value))
```

Common Framework

Pure compiled deterministic computation

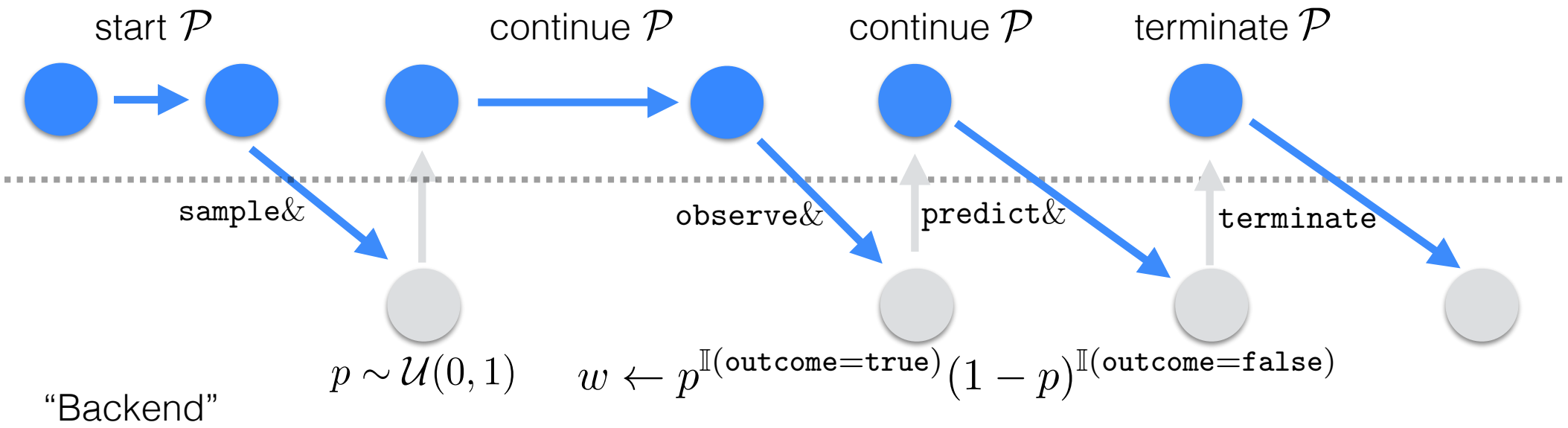


Likelihood Weighting “Backend”

```
(defn sample& [dist k]
  ;; Call the continuation with a sampled value
  (k (sample dist)))

(defn observe& [dist value k]
  ;; Compute and record the log weight
  (add-log-weight! (observe dist value))
  ;; Call the continuation with no arguments
  (k))

(defn predict& [label value k]
  ;; Store predict, and call continuation
  (store! label value)
  (k))
```

Likelihood Weighting Example

Compiled pure deterministic computation




```
(defquery flip-example [outcome]
  (let [p (sample (uniform-continuous 0 1))]
    (observe (flip p) outcome)
    (predict :p p)))
```

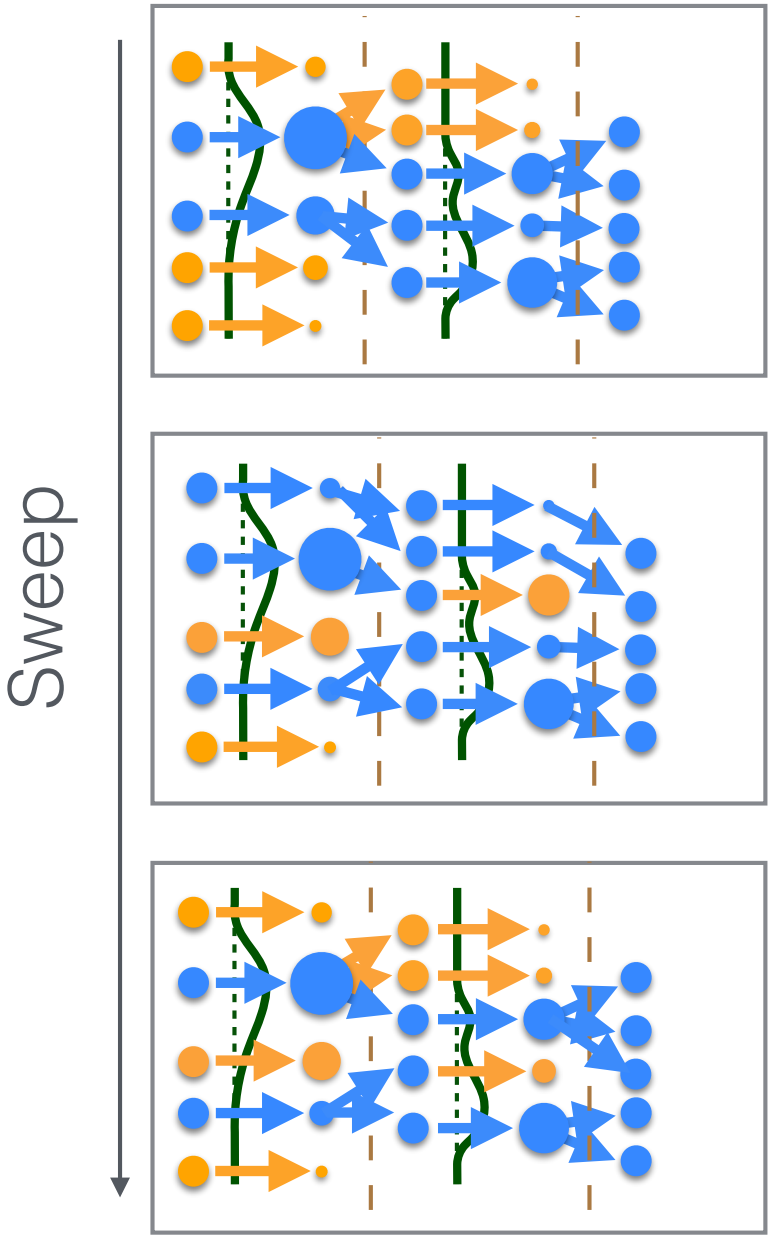
SMC Backend

```
(defn sample& [dist k]
  ;; Call the continuation with a sampled value
  (k (sample dist)))

(defn observe& [dist value k]
  ;; Block and wait for K calls to reach observe&
  ;; Compute weights
  ;; Use weights to subselect continuations to call
  ;; Call K sampled continuations (often multiple times)
  )

(defn predict& [label value k]
  ;; Store predict, and call continuation
  (store! label value)
  (k))
```

Particle Markov Chain Monte Carlo

Andrieu, Doucet, Holenstein “Particle Markov chain Monte Carlo methods.“ JRSSB 2010

- Iterable SMC
  - PIMH : “particle independent Metropolis-Hastings”
  - PGIBBS : “iterated conditional SMC”
  - PGAS : “particle Gibbs ancestral sampleing"



Wood, van de Meent, Mansinghka “A new approach to probabilistic programming inference” AISTATS 2014

PIMH Math

- Each sweep of SMC can compute

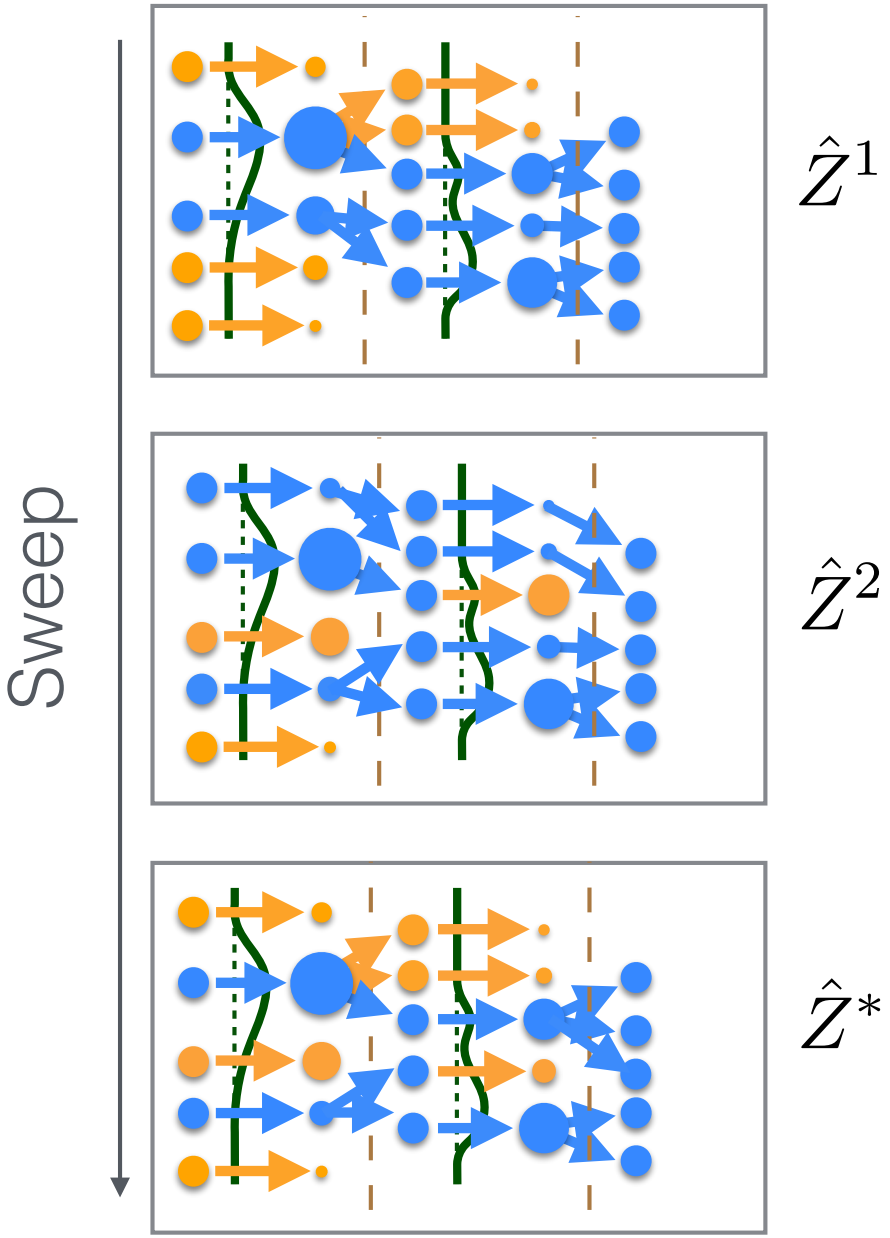
$$\hat{Z} = \prod_{n=1}^{N} \hat{Z}_n = \prod_{n=1}^{N} \frac{1}{K} \sum_{k=1}^{K} w(\tilde{\mathbf{x}}_{1:n}^k)$$

- PIMH is MH that accepts entire new particle sets w.p.

$$\alpha_{PIMH}^s = \min\left(1, \frac{\hat{Z}^\star}{\hat{Z}^{s-1}}\right)$$

- And all particles can be used

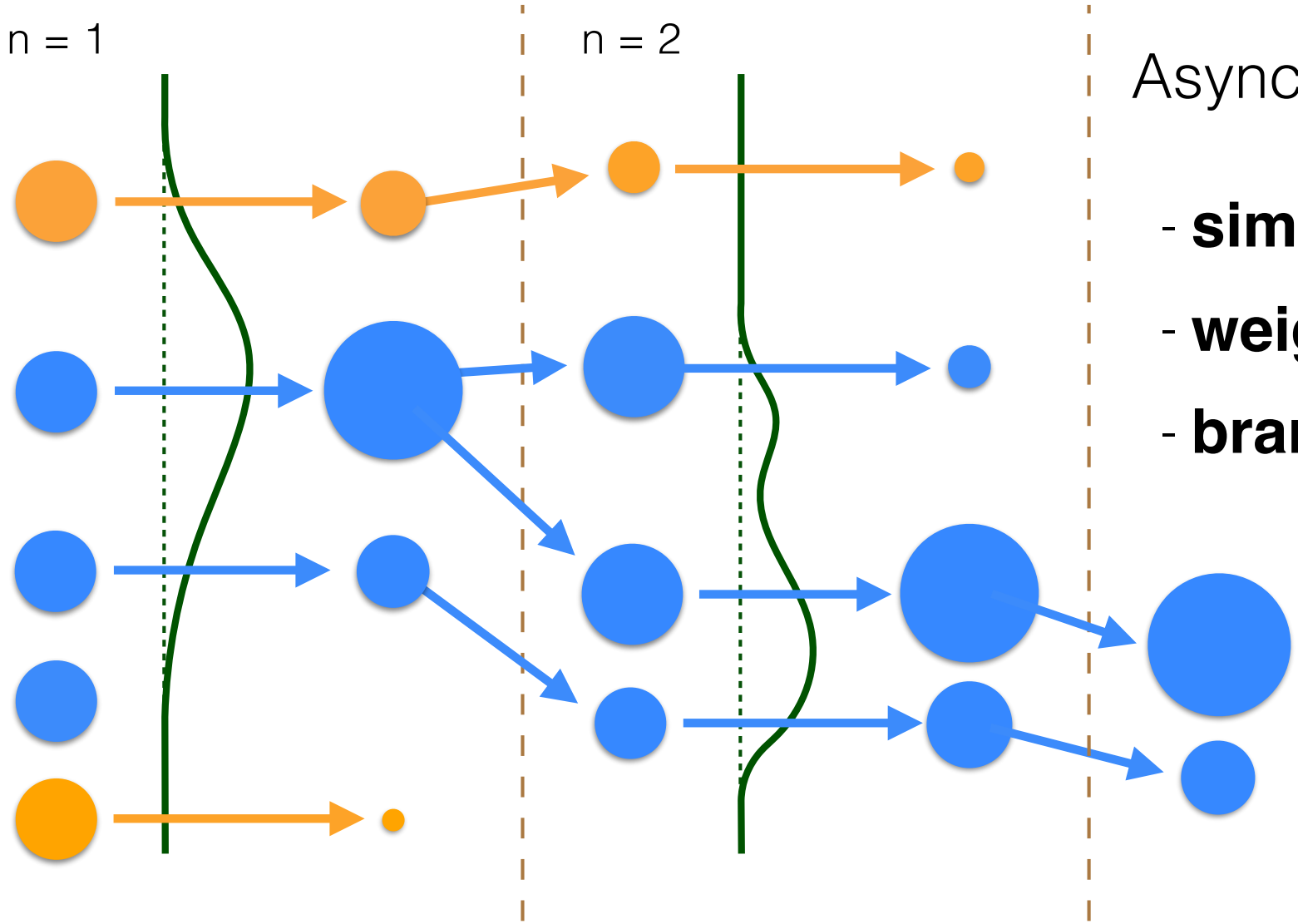
$$\hat{\mathbb{E}}_{PIMH}[Q(\mathbf{x})] = \frac{1}{S} \sum_{s=1}^{S} \sum_{k=1}^{K} W^{s,k} Q(\mathbf{x}^{s,k})$$



Wood, van de Meent, Mansinghka “A new approach to probabilistic programming inference” AISTATS 2014

Paige and Wood “A Compilation Target for Probabilistic Programming Languages” ICML 2014

Particle Cascade



Asynchronously

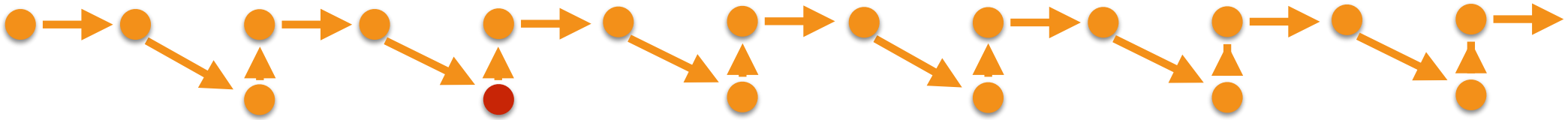
- **simulate**
- **weight**
- **branch**

Paige, W., Doucet, Teh; NIPS 2014

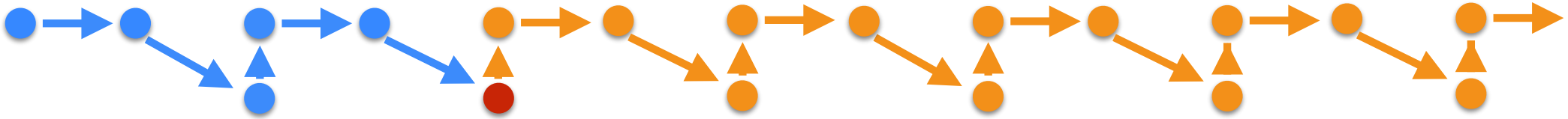
LMH Backend

```
(defn sample& [a dist k]
  (let [;; reuse previous value,
        ;; or sample from prior
        x (or (get-cache a)
              (sample dist))]
    ;; add to log-weight when reused
    (when (get-cache a)
      (add-log-weight! (observe dist x)))
    ;; store value and its log prob in trace
    (store-in-trace! a x dist)
    ;; continue with value x
    (k x)))

(defn observe& [dist value k]
  ;; Compute and record the log weight
  (add-log-weight! (observe dist value))
  ;; Call the continuation with no arguments
  (k))
```

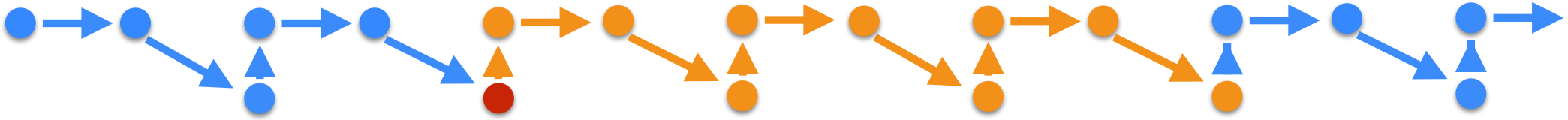
LMH Variants



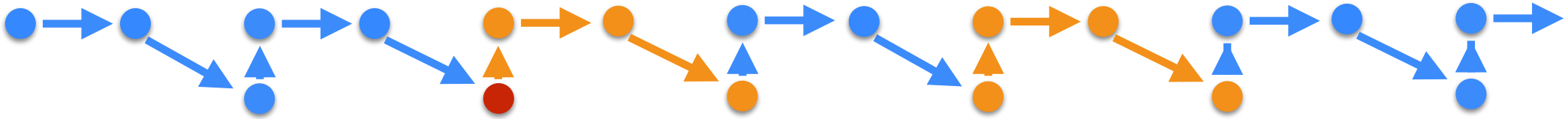
D. Wingate, A. Stuhlmueller, and N. D. Goodman.
"Lightweight implementations of probabilistic programming languages via transformational compilation." AISTATS (2011).



WebPPL
Anglican



"C3: Lightweight Incrementalized MCMC for Probabilistic Programs using Continuations and Callsite Caching."
D. Ritchie, A. Stuhlmuller, and N. D. Goodman. arXiv:1509.02151 (2015).



"Venture: a higher-order probabilistic programming platform with programmable inference."
V. Mansinghka, D. Selsam, and Y. Perov. arXiv:1404.0099 (2014).

Inference Backends in Anglican

- 14+ algorithms
- Average 165 lines of code per!
- Can implement and use without touching core code base.

Algorithm	Type	Lines of Code	Citation	Description
smc	IS	127	Wood et al. AISTATS, 2014	Sequential Monte Carlo
importance	IS	21		Likelihood weighting
pcascade	IS	176	Paige et al., NIPS, 2014	Particle cascade: Anytime asynchronous sequential Monte Carlo
pgibbs	PMCMC	121	Wood et al. AISTATS, 2014	Particle Gibbs (iterated conditional SMC)
pimh	PMCMC	68	Wood et al. AISTATS, 2014	Particle independent Metropolis-Hastings
pgas	PMCMC	179	van de Meent et al., AISTATS, 2015	Particle Gibbs with ancestor sampling
lmh	MCMC	177	Wingate et al., AISTATS, 2011	Lightweight Metropolis-Hastings
almh	MCMC	320	Tolpin et al., ECML PKDD, 2015	Adaptive scheduling lightweight Metropolis-Hastings
rmh*	MCMC	319	-	Random-walk Metropolis-Hastings
palmh	MCMC	66	-	Parallelised adaptive scheduling lightweight Metropolis-Hastings
plmh	MCMC	62	-	Parallelised lightweight Metropolis-Hastings
bamc	MAP	318	Tolpin et al., SoCS, 2015	Bayesian Ascent Monte Carlo
siman	MAP	193	Tolpin et al., SoCS, 2015	MAP estimation via simulated annealing

Wrap Up

Where We Stand

- Probabilistic programming concept
  - Long well established

- Tool maturity
  - Homework
  - Prototyping
  - Research
  - Advanced research
  - Small real-world applications

- Put-offs
  - Some highly optimized models that you know to scale well don’t necessarily scale well in current probabilistic programming systems.

Opportunities

Static Efficiencies

- Automated program transformations that simplify or eliminate inference (moving observes up and out)

```
(defquery beta-bernoulli [observation]
  (let [dist (beta 1 1)
        theta (sample dist)
        like (flip theta)]
    (observe like observation)
    (predict :theta theta)))
```

```
(defquery beta-bernoulli [observation]
  (let [dist (beta
               (if observation 2 1)
               (if observation 1 2))
        theta (sample dist)]
    (predict :theta theta)))
```

Carette and Shan. “Simplifying Probabilistic Programs Using Computer Algebra⋆.” T.R. 719, Indiana University (2015)
Yang - Keynote Lecture, APLAS (2015)

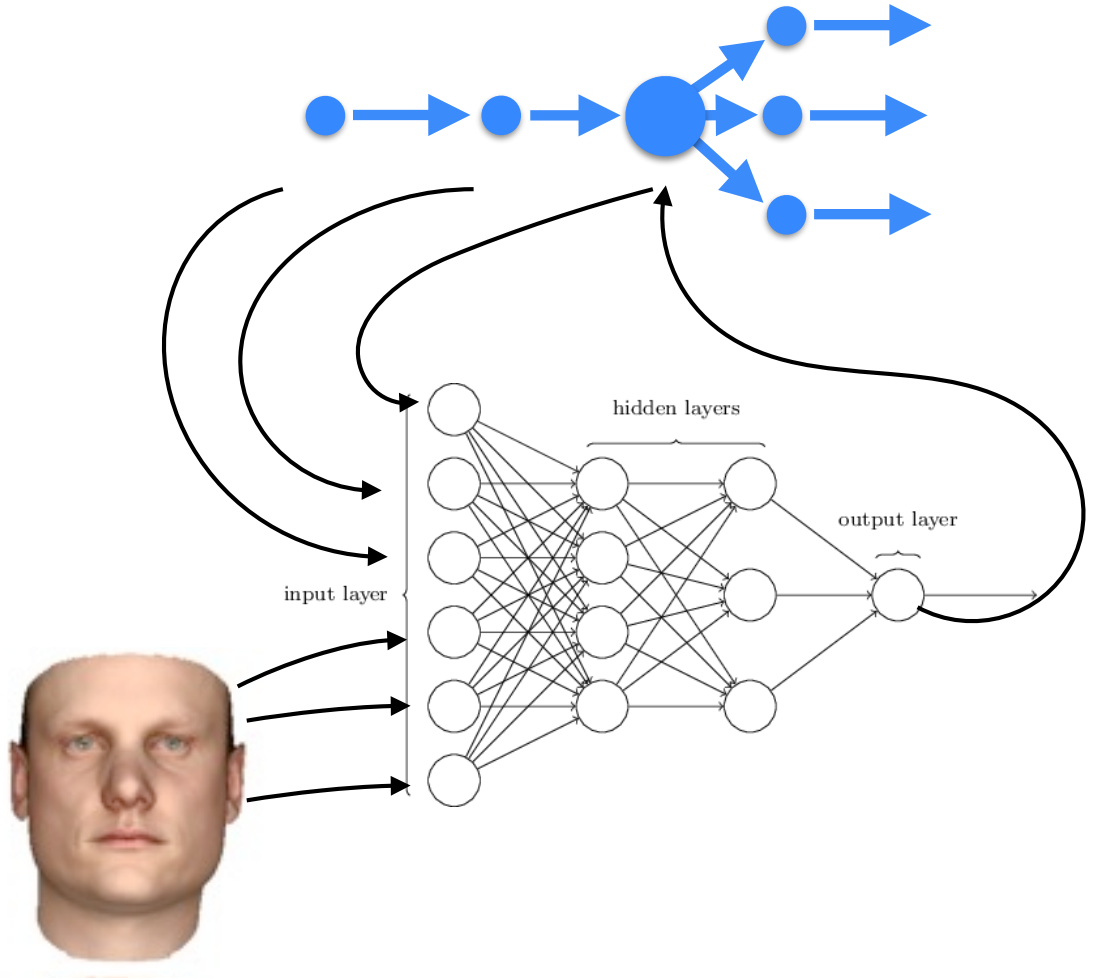
Normalization

- Program analyses that identify algebraic or algorithmic normalization opportunities

```
(defquery marsaglia [mu sigma]
  (let [u (uniform-continuous -1.0 1.0)]
    (predict :x
       (loop [x (sample u)
              y (sample u)]
         (let [s (+ (* x x) (* y y))]
           (if (< s 1.0)
             (+ mu (* sigma
                      (* x (sqrt (* -2.0 (/ (log s) s))))))
             (recur (sample u) (sample u))))))))
```


$$p(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Data driven proposals



Kulkarni, Kohli, Tenenbaum, Mansinghka "Picture: a probabilistic programming language for scene perception." CVPR (2015).
Perov, Le, Wood “Data-driven Sequential Monte Carlo in Probabilistic Programming” NIPS BBLI Workshop (2015).
Paige, Wood “Inference Networks for Graphical Models” NIPS AABI Workshop (2015).

Wrap Up

A Gentle Plea

- Bayesians
  - Stop writing assembly code!

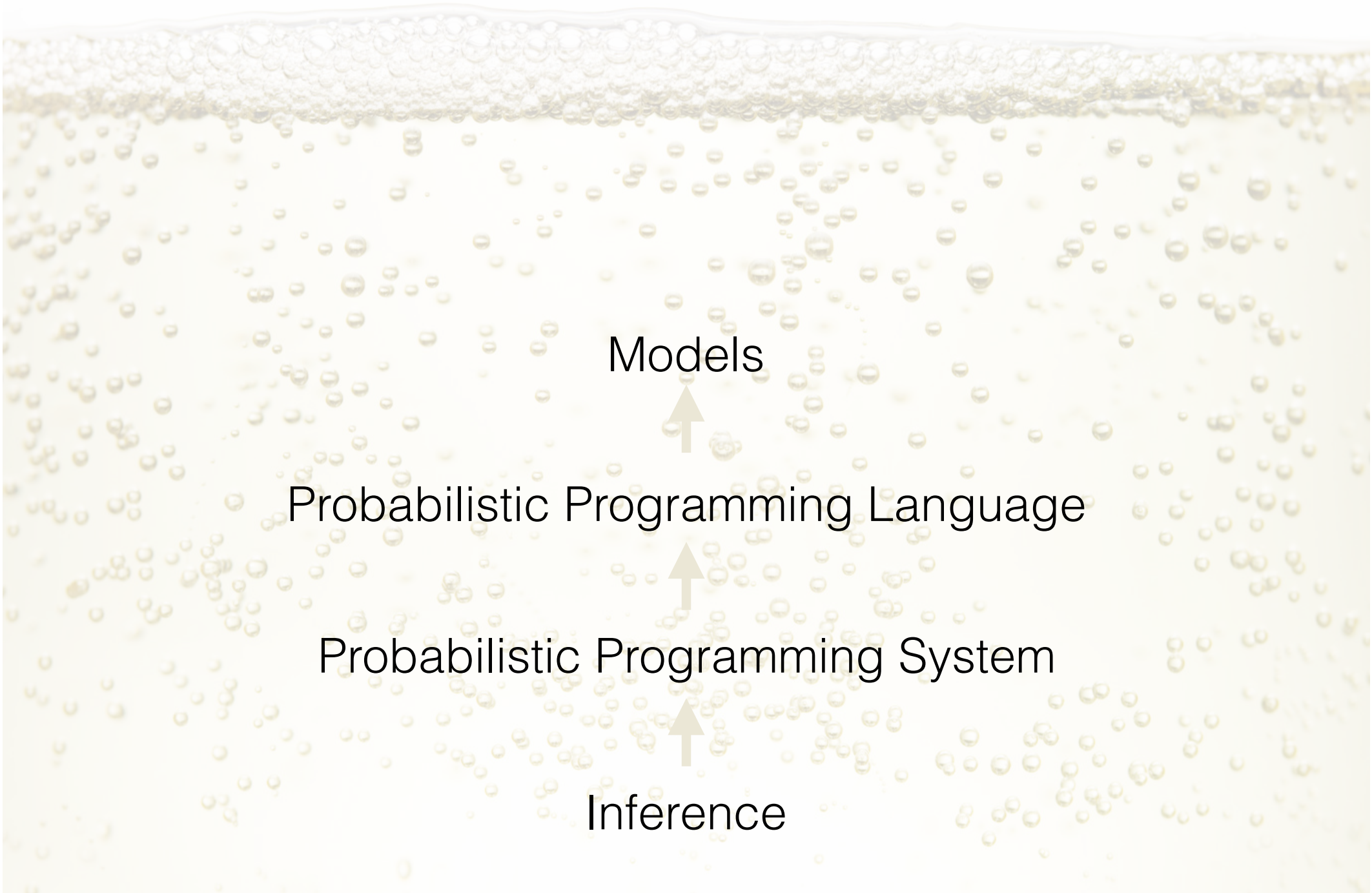
  - Join us
    - Try writing models in our languages
    - Contribute inference algorithms

- Neural net people
  - Help make inference better
  - Train your neural nets to do something interpretable

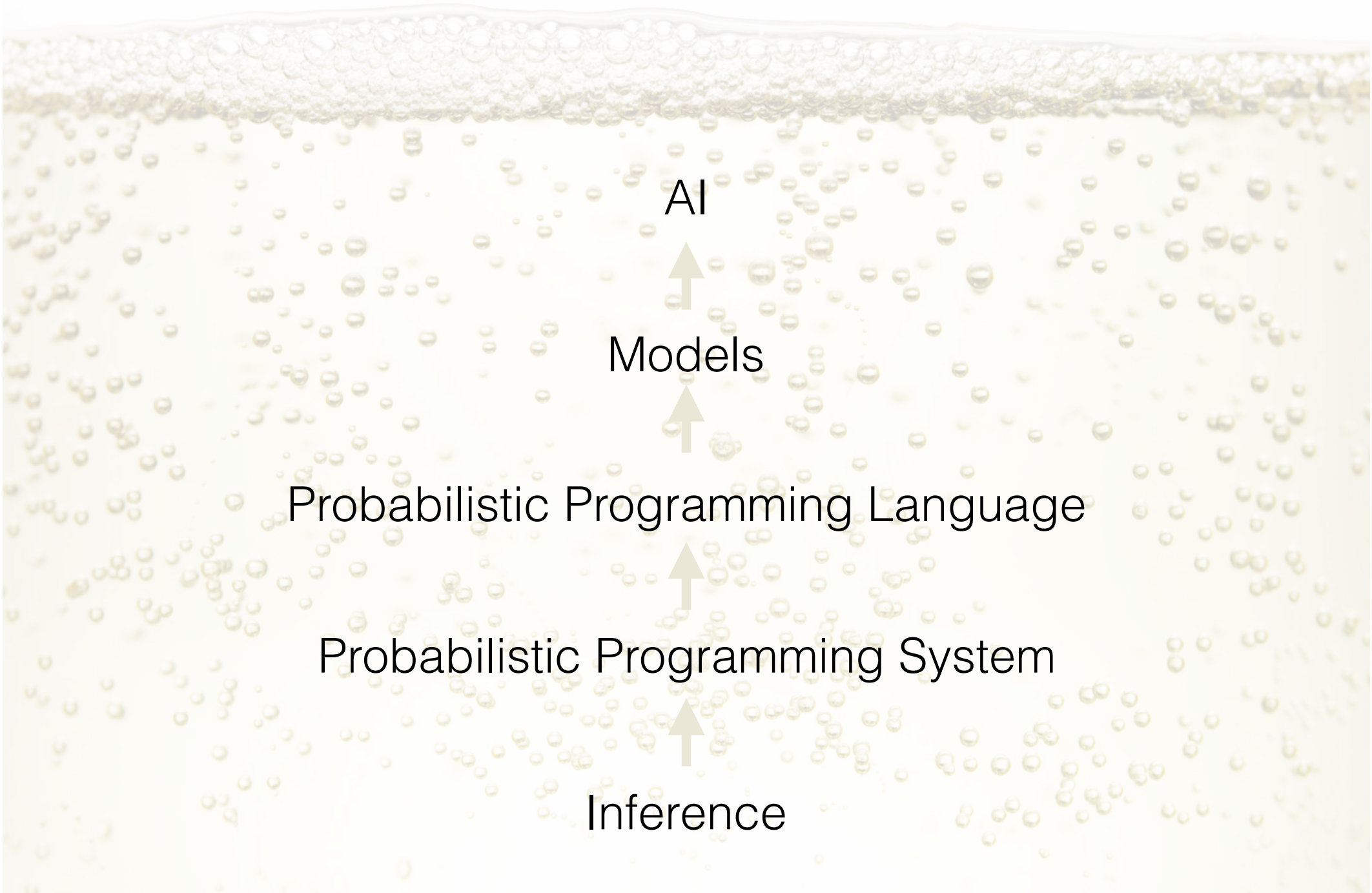
Best Way to AI

- Neural nets end to end (DeepMind)
- Generic probabilistic programs used to impose evolutionary regularity on that which is computed by deep networks.

Bubble Up


Models
Probabilistic Programming Language
Probabilistic Programming System
Inference

Bubble Up


AI
Models
Probabilistic Programming Language
Probabilistic Programming System
Inference

Anglican Resources

- General
  - http://www.robots.ox.ac.uk/~fwood/anglican/
- Learning probabilistic programming and Anglican
  - https://bitbucket.org/probprog/mlss2015
- Writing applications
  - https://bitbucket.org/probprog/anglican-user
- The core / looking at inference algorithms
  - https://bitbucket.org/probprog/anglican
- Trying it out (5 min. install)
  - https://bitbucket.org/probprog/anglican-examples

Go-To Resources

- Writing your own probabilistic programming language
  - http://dippl.org
- Model example repository
  - http://forestdb.org/
- Easiest places to start (browser-based)
  - http://webppl.org/
  - https://probmods.org/
- Place to find all the literature in one place
  - http://probabilistic-programming.org/wiki/Home
- Place to go for the most advanced ideas in prob. prog.
  - http://probcomp.csail.mit.edu/venture/

Some Final Thoughts

Adopting the kinds of abstraction boundaries suggested by probabilistic programming practice will move the field of machine learning forward much faster make it easier for inference and modeling experts to work together.

Probabilistic programming is not about making what you already do faster or somehow better but instead about making it possible to do things that would otherwise be nearly impossible to do.

Thank You

Faculty Market

van de Meent

Graduating

Paige

Too Late

Tolpin

Perov

Le

Yang

Tenenbaum

Mansinghka

- Funding : DARPA

Postdoc Openings

- 3 probabilistic programming postdoc openings
  - http://goo.gl/BtoCEr

Next Tutorial By

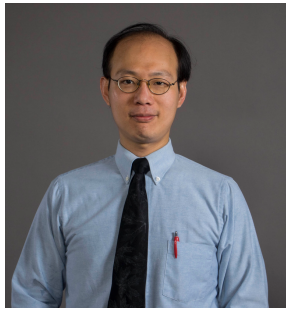
van de Meent

Paige

Mansinghka

Pfeffer

Shan

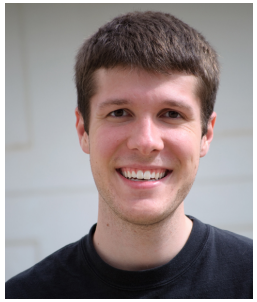


Perov

Wingate

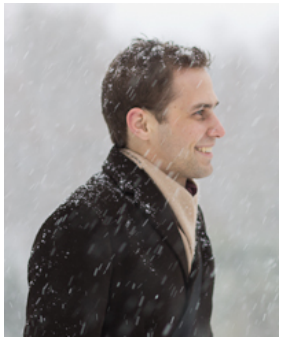
Goodman

Ritchie



Gordon

Stuhlmüller



Roy

Russell

Scibior