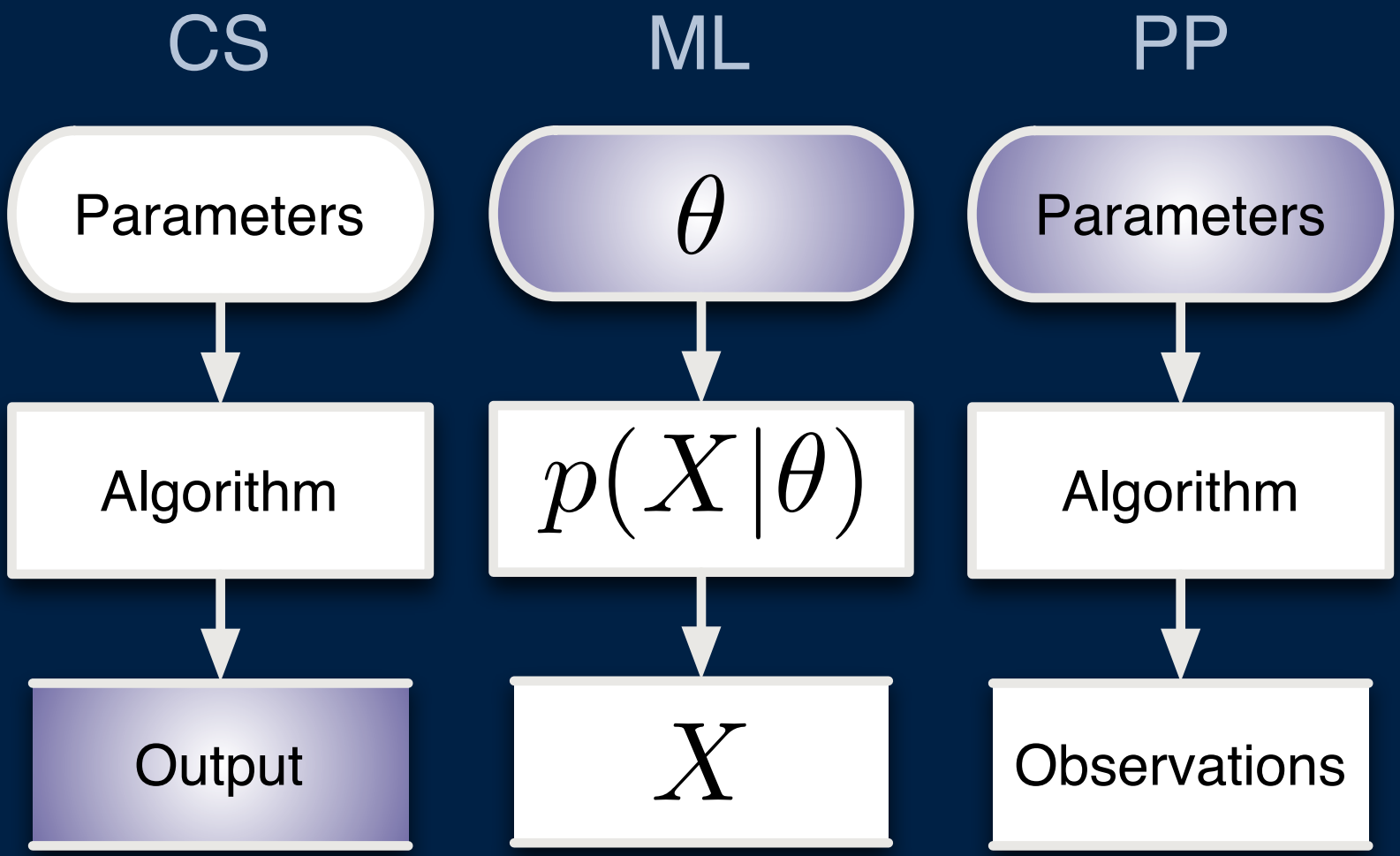
DEPARTMENT OF ENGINEERING SCIENCE
Information, Control, and Vision Engineering

A New Approach to Probabilistic Programming Inference

UNIVERSITY OF
OXFORD

Frank Wood
fwood@robots.ox.ac.uk
http://www.robots.ox.ac.uk/~fwood
Avances In Scalable Bayesian Computation
March 3-7, 2014 Banff

What is Probabilistic Programming?



UNIVERSITY OF OXFORD

Probabilistic Programming Goals

(i) Accelerate iteration over models
- Code is easier to read and write than math
- Lower technical barrier of entry to development of new models

(ii) Accelerate iteration over inference procedures
- Computer language is an abstraction barrier
  - Inference procedures can be tested against a library of models
  - Inference procedures become “compiler optimizations”

(iii) Enable development of more expressive models
- Probabilistic programs can express a superset of graphical models
- Modern machine learning models are tens of lines of code

UNIVERSITY OF
OXFORD

DEPARTMENT OF ENGINEERING SCIENCE
Information, Control, and Vision Engineering

Anglican

"A Church of England Venture"

http://www.robots.ox.ac.uk/~fwood/anglican/

UNIVERSITY OF OXFORD

xerox Research Centre Europe

Google

van de Meent

Mansinghka

Wood, van de Meent, Mansinghka "A New Approach to Probabilistic Programming Inference" AISTATS 2014

Venture*-Inspired Syntax and Semantics

```
[assume symbol <expr>]
[observe <expr> <const>]
[predict <expr>]
```

Assume : variable declaration
Observe : data
Predict : printout

All <>'s are Scheme/Lisp expressions

```
(<proc> <arg> ... <arg>)
```

UNIVERSITY OF OXFORD

* http://probcomp.csail.mit.edu/venture/

Anglican Interpretation

```
[assume sigma-squared 2]
[assume mu (marsaglia-normal 1 5)]
[observe (normal mu sigma-squared) 9]
[observe (normal mu sigma-squared) 8]
[predict mu]
```

```
mu,4.579108417224186
mu,5.84552255327749
mu,4.579108417224186
mu,5.84552255327749
mu,5.518797656065681
mu,5.518797656065681
mu,4.579108417224186
mu,5.84552255327749
mu,5.060325042296315
mu,5.518797656065681
mu,6.359401703719326
mu,4.714582404953012
mu,6.359401703719326
```

Probabilistic programs are constrained generative models with uncertainty represented by random variables that are assigned values by stochastic procedure calls

"Running" an Anglican program outputs "predict"-ed posterior samples of random variable assignments conditioned on observed data

UNIVERSITY OF OXFORD

Anglican Expressivity

- Programming language
  - Wide collection of built-in stochastic procedures
  - Turing complete
    - Procedures are first class objects
    - (eval <expr>) and (apply <proc> …) supported
  - Recursion
  - Memoisation
- Model family
  - All computable generative models

```
[assume fib (lambda (n)
  (cond ((= n 0) 1) ((= n 1) 1)
                (else (+ (fib (- n 1)) (fib (- n 2))))))]
[assume r (poisson 4)]
[assume l (if (< 4 r) 6 (+ (fib (* 3 r)) (poisson 4)))]
[observe (poisson l) 6]
[predict r]
⋮
```

UNIVERSITY OF OXFORD

Related Work

- STAN [Stan Dev. Team, 2013]
- Infer.NET [Minka, Winn et al, 2010]

- IBAL [Pfeffer, 2001]
- BLOG [Milch et al, 2004]

- Church [Goodman, Mansinghka, et al, 2008/2012]
  - Random Database [Wingate, Stuhlmüller et al, 2011]
- Venture* [Mansinghka, et al, in prep]
  - Anglican [Wood, van de Meent, Mansinghka, 2014]

UNIVERSITY OF OXFORD

* http://probcomp.csail.mit.edu/venture/

Execution Trace Preliminaries

- "Exploring" the space of execution traces = inference
- Different program traces arise from application of *Stochastic Procedures (SPs)*

  flip, normal, discrete, poisson, dirichlet, gamma, …

- An execution trace is uniquely determined by return values for all SP applications
- Observations weight execution traces

UNIVERSITY OF
OXFORD

Execution Trace Joint

- Observes
  - Likelihood terms

Observed value

Parameter of observation distribution

$$\tilde{p}(\mathbf{y}, \mathbf{x}) \equiv \prod_{n=1}^{N} p(y_n | \theta_{t_n}, \mathbf{x}_n) \tilde{p}(\mathbf{x}_n | \mathbf{x}_{n-1})$$

Type of observation distribution

Interpreter memory state

- Assumes
  - Generate sequences of stochastic procedure applications

Parameter of stochastic procedure

$$\tilde{p}(\mathbf{x}_n | \mathbf{x}_{n-1}) = \prod_{k=1}^{|\mathbf{x}_n \setminus \mathbf{x}_{n-1}|} p(x_{n,k} | \theta_{t_{n,k}}, x_{n,1:(k-1)}, \mathbf{x}_{n-1}).$$

Type of stochastic procedure

UNIVERSITY OF OXFORD

MCMC / Random DB Review

Sample from joint constrained by observes

$$\tilde{p}(\mathbf{x}|\mathbf{y}) \propto \tilde{p}(\mathbf{y},\mathbf{x})$$

Joint is a fixed syntactically allowable reordering of conditionals

$$\tilde{p}(\mathbf{y},\mathbf{x}) \equiv \tilde{p}(\mathbf{x}_1)\tilde{p}(\mathbf{x}_2|\mathbf{x}_1)\cdots\tilde{p}(\mathbf{x}_n|\mathbf{x}_{n-1})p(y_1|\mathbf{x}_1)\cdots p(y_n|\mathbf{x}_n)$$

Metropolis-Hastings

$$\min\left(1, \frac{p(\mathbf{y}|\mathbf{x}')p(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})q(\mathbf{x}'|\mathbf{x})}\right)$$

Single stochastic procedure (SP) output

Probability of new part of proposed execution trace

$$q(\mathbf{x}'|\mathbf{x}) = \frac{\kappa(x'_{m,j}|x_{m,j})}{|\mathbf{x}|}\frac{p(\mathbf{x}'\backslash\mathbf{x} \mid \mathbf{x}'\cap\mathbf{x})}{p(x'_{m,j}|\mathbf{x}'\cap\mathbf{x})}$$

Number of SP's in original trace

Probability of new SP return value (sample) given trace prefix

UNIVERSITY OF OXFORD

MCMC / Random DB Review

Proposal distribution == the prior

$$\kappa(x'_{m,j}|x_{m,j}) = p(x'_{m,j}|\mathbf{x}' \cap \mathbf{x})$$

Full MH acceptance ratio

Number of SP applications in original trace

Probability of regenerating current trace continuation given proposal trace beginning

$$\frac{p(\mathbf{y}|\mathbf{x}')\, p(\mathbf{x}')\, |\mathbf{x}|\, p(\mathbf{x} \backslash \mathbf{x}' \mid \mathbf{x} \cap \mathbf{x}')}{p(\mathbf{y}|\mathbf{x})\, p(\mathbf{x})\, |\mathbf{x}'|\, p(\mathbf{x}' \backslash \mathbf{x} \mid \mathbf{x}' \cap \mathbf{x})}$$

Number of SP applications in new trace

Probability of generating proposal trace continuation given current trace beginning

UNIVERSITY OF OXFORD

- Church [Goodman, Mansinghka, et al, 2012]
  - Random Database [Wingate, Stuhlmüller et al, 2011]

MCMC / Random DB Criticism

- Proposing from the prior is suboptimal
- Computation is wasted
  - Short-circuiting -> MIT Venture
- MH is local*

UNIVERSITY OF
OXFORD

SMC for Prob. Prog. Inference

Inference Goal

- Given samples (interpreter memory states)

$$\mathbf{x}_{n-1}^{(\ell)} \sim \tilde{p}(\mathbf{x}_{n-1}|y_{1:(n-1)})$$

- sample from (find program traces that reflect the next observe).

$$\tilde{p}(\mathbf{x}_n|y_{1:n})$$

SMC / SIR Approach

- Propose from the prior = interpret the program up to next observe

$$q(\hat{\mathbf{x}}_n^{(\ell)}|\mathbf{x}_{n-1}^{(\ell)}, y_{1:n}) \equiv p(\hat{\mathbf{x}}_n^{(\ell)}|\mathbf{x}_{n-1}^{(\ell)})$$

- Calc. importance weights (weight by observe outer application likelihood)

$$\tilde{w}_n^{(\ell)} = p(y_n|\hat{\mathbf{x}}_n^{(\ell)})$$

and resample (new interpreter memory states)

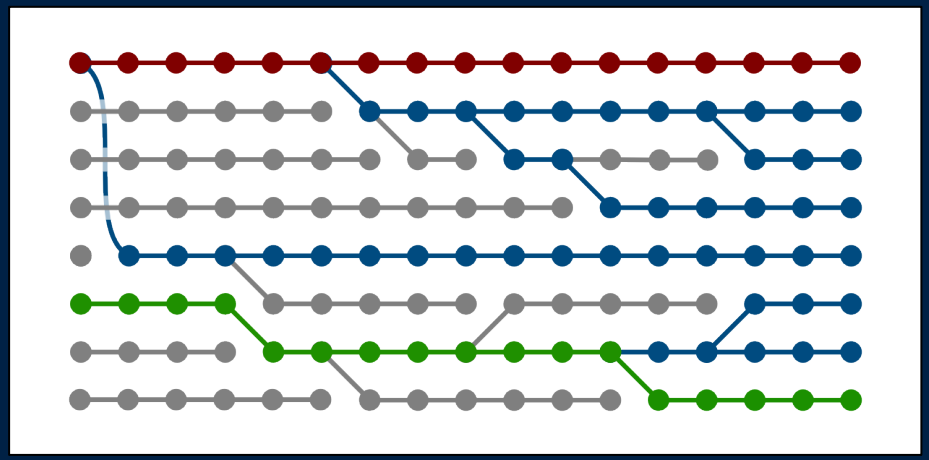
$$\mathbf{x}_n^{(\ell)} \sim \sum_\ell w_n^{(\ell)} \delta_{\hat{\mathbf{x}}_n^{(\ell)}} \qquad w_n^{(\ell)} = \tilde{w}_n^{(\ell)} / \sum_j \tilde{w}_n^{(j)}$$

UNIVERSITY OF OXFORD

Anglican = PMCMC for Prob. Prog.

- Example : Particle Gibbs
  - MH w/ accept prob. = 1
  - “Retained particle” x*
  - *Non-local*
    - Potentially changes many variable values at once

[Holenstein 2009; Andrieu, Doucet, Holenstein 2010; etc]



Algorithm 1 PMCMC for Prob. Prog. Inference

$L \leftarrow$ number of particles
$S \leftarrow$ number of sweeps
$\{\tilde{w}_N^{(\ell)}, \mathbf{x}_N^{(\ell)}\} \leftarrow$ Run SMC
for $s < S$ **do**
  $\{\cdot, \mathbf{x}_N^*\} \leftarrow \mathtt{r}(1, \{1/L, \mathbf{x}_N^{(\ell)}\})$
  $\{\cdot, \mathbf{x}_0^{(\ell)}\} \leftarrow$ initialize $L-1$ interpreters
  for $d \in$ ordered lines of program **do**
    for $\ell < L-1$ **do**
      $\bar{\mathbf{x}}_{n-1}^{(\ell)} \leftarrow \mathtt{fork}(\mathbf{x}_{n-1}^{(\ell)})$
    end for
    if $\mathtt{directive}(\mathrm{d}) ==$ “`assume`” **then**
      for $\ell < L-1$ **do**
        $\bar{\mathbf{x}}_n^{(\ell)} \leftarrow \mathtt{interpret}(d, \bar{\mathbf{x}}_{n-1}^{(\ell)})$
      end for
      $\{\mathbf{x}_n^{(\ell)}\} \leftarrow \{\bar{\mathbf{x}}_n^{(\ell)}\} \cup \mathbf{x}_n^*$
    else if $\mathtt{directive}(\mathrm{d}) ==$ “`predict`” **then**
      for $\ell < L-1$ **do**
        $\mathtt{interpret}(d, \bar{\mathbf{x}}_{n-1}^{(\ell)})$
      end for
      $\mathtt{interpret}(d, \mathbf{x}_{n-1}^*)$
    else if $\mathtt{directive}(\mathrm{d}) ==$ “`observe`” **then**
      for $\ell < L-1$ **do**
        $\{\bar{w}_n^{(\ell)}, \bar{\mathbf{x}}_n^{(\ell)}\} \leftarrow \mathtt{interpret}(d, \bar{\mathbf{x}}_{n-1}^{(\ell)})$
      end for
      $\mathcal{T} \leftarrow \mathtt{r}(L-1, \{\bar{w}_n^{(\ell)}, \bar{\mathbf{x}}_n^{(\ell)}\} \cup \{\tilde{w}_n^*, \mathbf{x}_n^*\})$
      $\{\tilde{w}_n^{(\ell)}, \mathbf{x}_n^{(\ell)}\} \leftarrow \mathcal{T} \cup \{\tilde{w}_n^*, \mathbf{x}_n^*\}$
    end if
  end for
end for

UNIVERSITY OF OXFORD

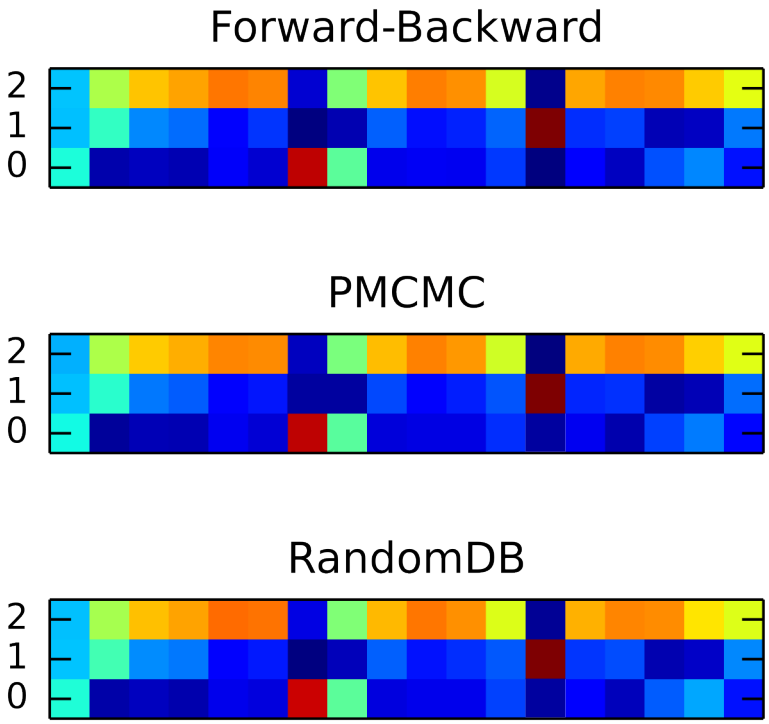
Wood, van de Meent, Mansinghka “A New Approach to Probabilistic Programming Inference” AISTATS 2014

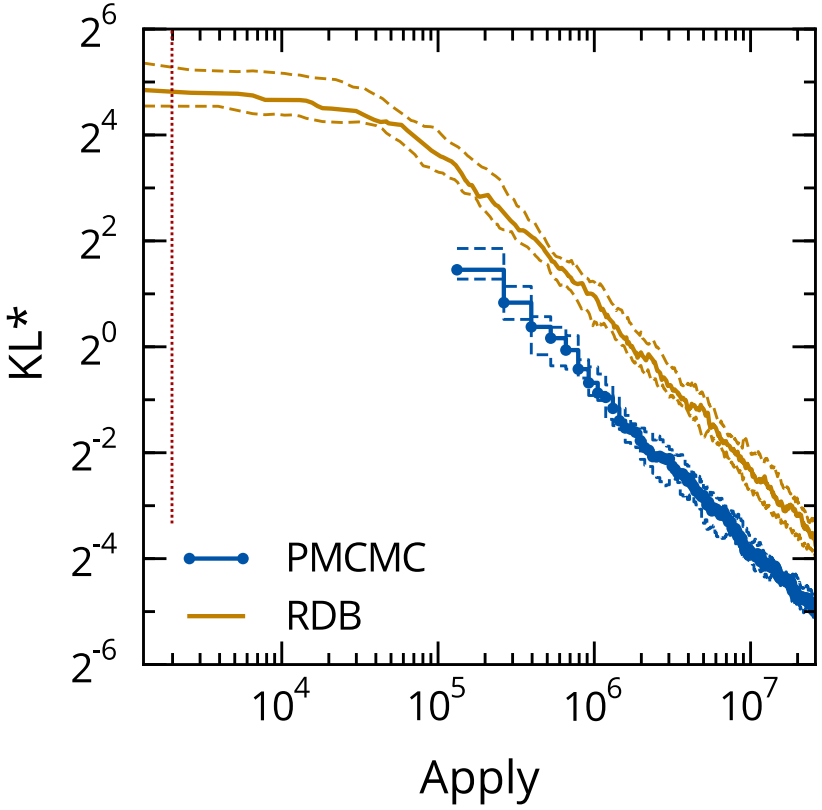
Example Program : Hidden Markov Model

```
[assume initial-state-dist (list (/ 1 3) (/ 1 3) (/ 1 3))]
[assume get-state-transition-dist (lambda (s)
  (cond ((= s 0) (list .1 .5 .4)) ((= s 1) (list .2 .2 .6))
        ((= s 2) (list .15 .15 .7))))]
[assume transition (lambda (prev-state)
  (discrete (get-state-transition-dist prev-state)))]
[assume get-state (mem (lambda (index)
  (if (<= index 0) (discrete initial-state-dist)
                   (transition (get-state (- index 1))))))]
[assume get-state-observation-mean (lambda (s)
        (cond ((= s 0) -1) ((= s 1) 1) ((= s 2) 0)))]
[observe (normal (get-state-obs-mean (get-state 1)) 1) .9]
[observe (normal (get-state-obs-mean (get-state 2)) 1) .8]
⋮
[observe (normal (get-state-obs-mean (get-state 16)) 1) -1]
[predict (get-state 0)]
[predict (get-state 1)]
⋮
[predict (get-state 16)]
⋮
```

UNIVERSITY OF
OXFORD

Anglican : Particle MCMC Inference



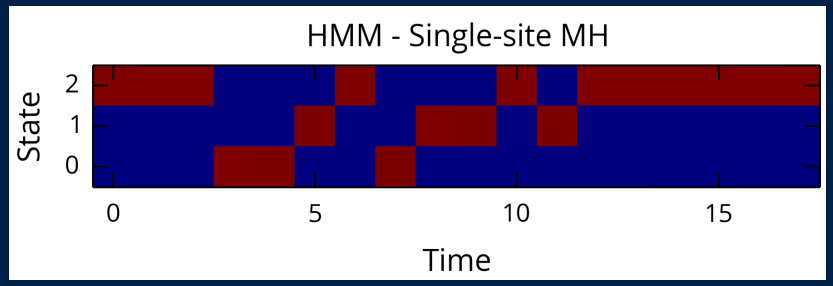


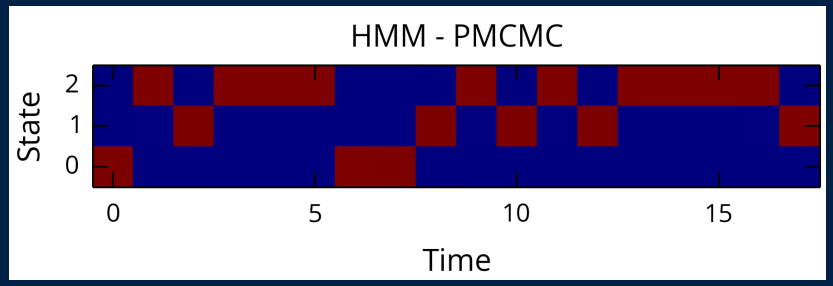
Wood, van de Meent, Mansinghka “A new approach to probabilistic programming inference.” AISTATS, 2014

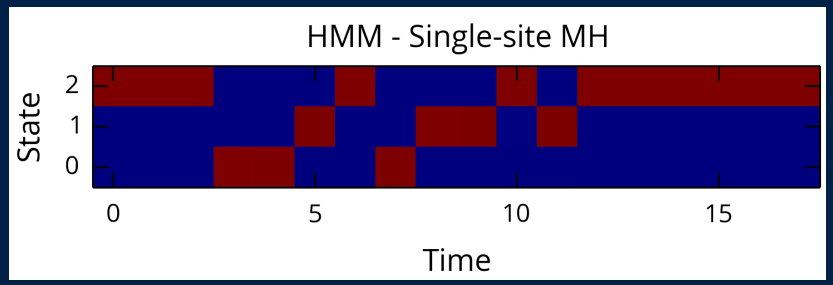
Wingate et al “Lightweight implementations of probabilistic programming languages via transformational compilation” AISTATS, 2011

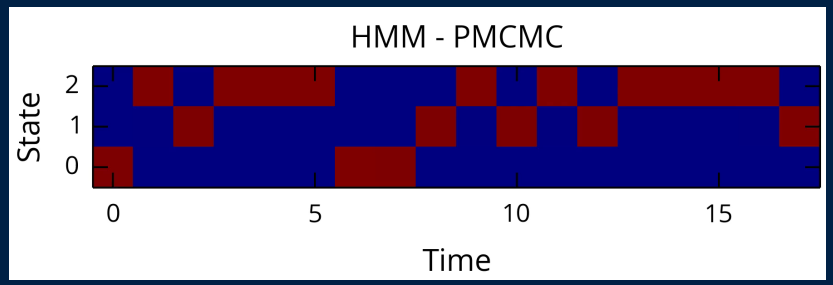
UNIVERSITY OF OXFORD

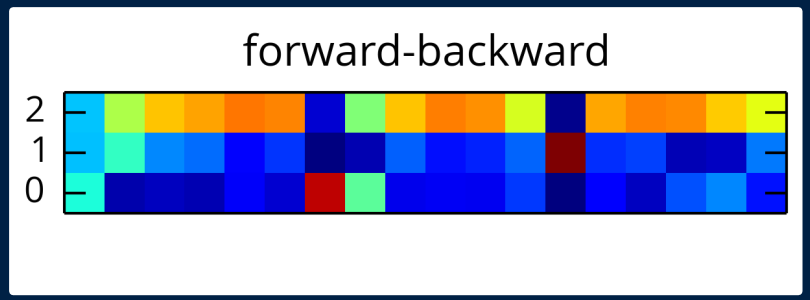
PMCMC HMM Visualization





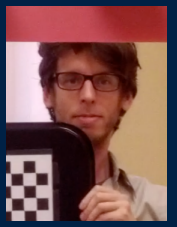






UNIVERSITY OF OXFORD

Compiled PMCMC – Probabilistic-C

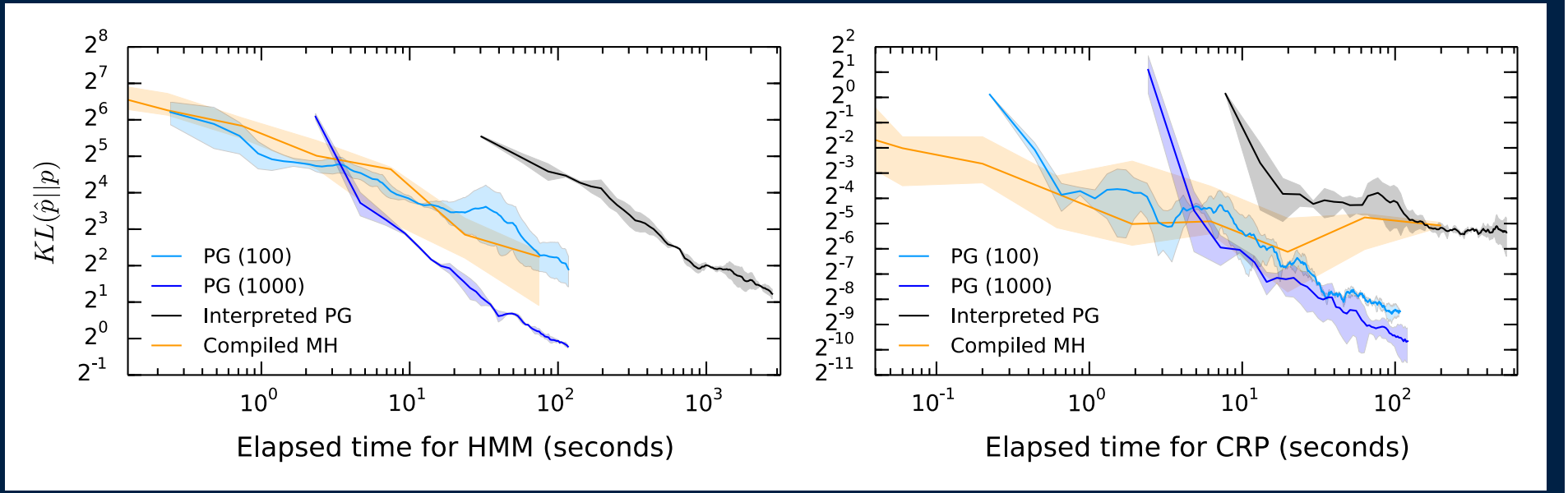


```c
#include "probabilistic.h"
#define K 3
#define N 11

/* Markov transition matrix */
static double T[K][K] = { { 0.1,  0.5,  0.4 },
                          { 0.2,  0.2,  0.6 },
                          { 0.15, 0.15, 0.7 } };

/* Observed data */
static double data[N] = { NAN, .9, .8,  .7,   0, -.025,
                               -5, -2, -.1,   0,  0.13 };

/* Prior distribution on initial state */
static double initial_state[K] = { 1.0/3, 1.0/3, 1.0/3 };

/* Per-state mean of Gaussian emission distribution */
static double state_mean[K] = { -1, 1, 0 };

/* Generative program for a HMM */
int main(int argc, char **argv) {

    int states[N];
    for (int n=0; n<N; i++) {
        states[n] = (n==0) ? discrete_rng(initial_state, K)
                           : discrete_rng(T[states[n-1]], K);
        if (n > 0) {
            observe(normal_lnp(data[n], state_mean[states[n]], 1));
        }
        predictf("state[%d],%d\n", n, states[n]);
    }

    return 0;
}
```

UNIVERSITY OF OXFORD

Paige et al "A Compilation Target for Probabilistic Programming Languages." in submission, 2014

Compiled PMCMC ≈ 100× Speedup

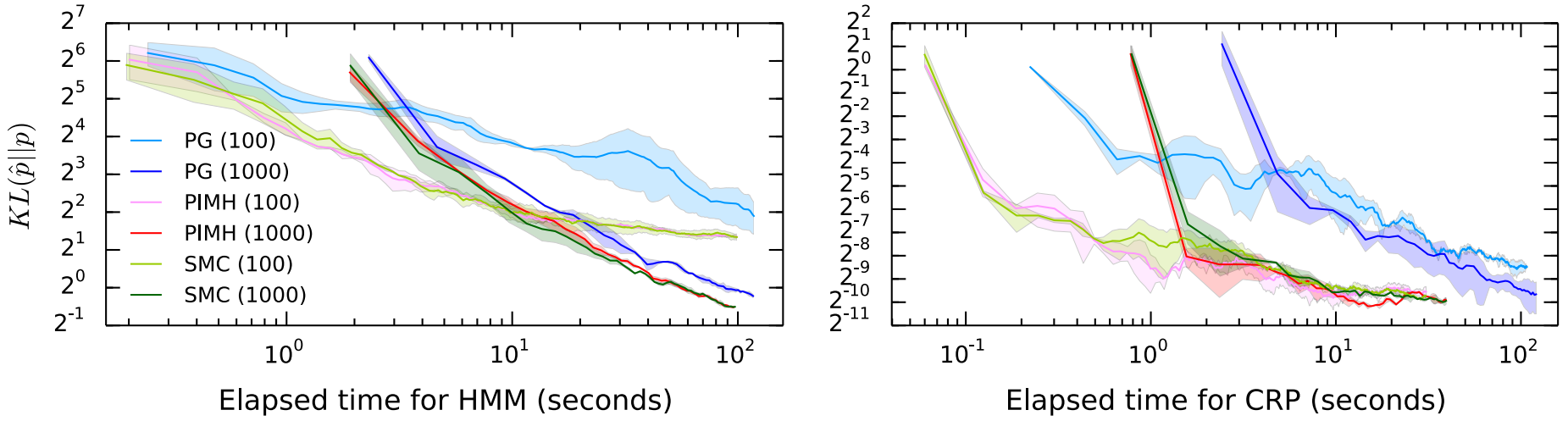


- HMM 10-states, 50 observations
- CRP 10 observation mixture of 1-D Gaussian

UNIVERSITY OF OXFORD

Compiled MH - https://github.com/dritchie/probabilistic-js

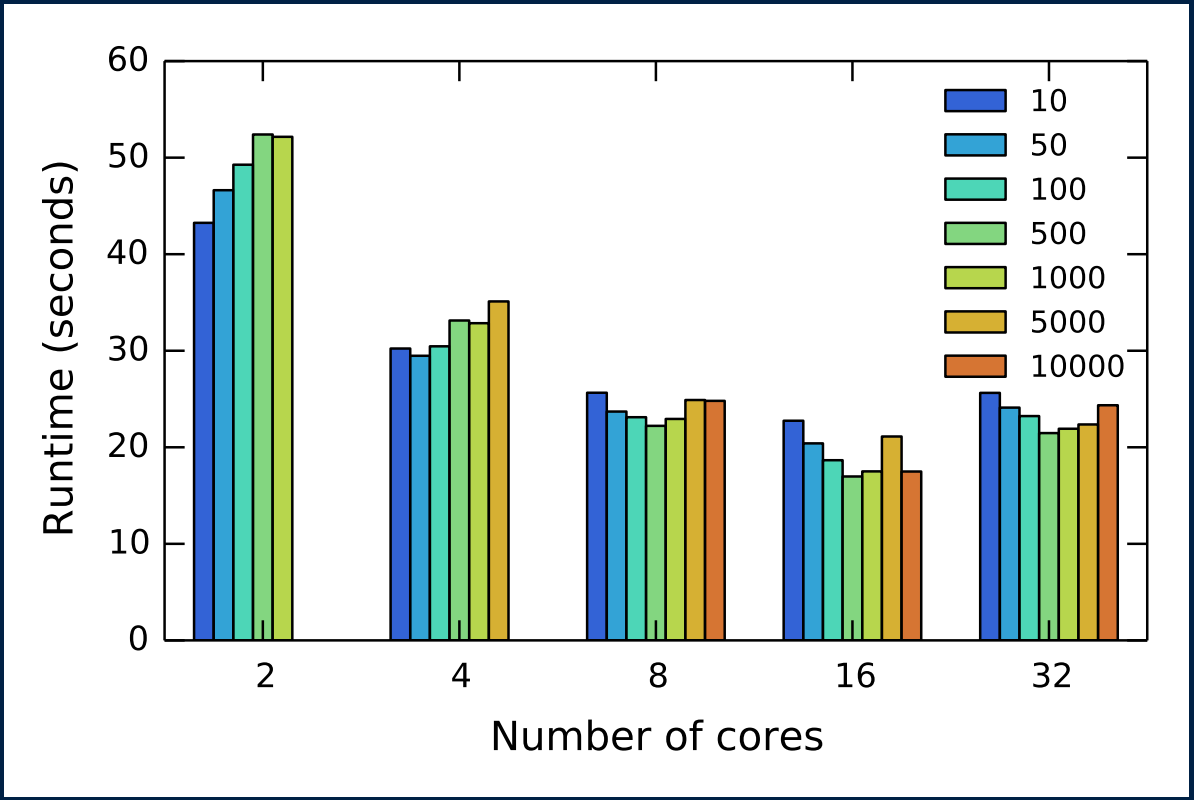
Compiled PMCMC Algorithm Performance



- What's the right inference algorithm for the model?
- What's the right performance measure?

UNIVERSITY OF OXFORD

Systems Optimization Path to Scalability



Time to produce 10,000 samples running probabilistic-C HMM code on multi-core EC2 instances with identical processor type while varying number of particles (bars). Both more cores and more particles eventually degrade performance suggesting possible operating system optimizations.

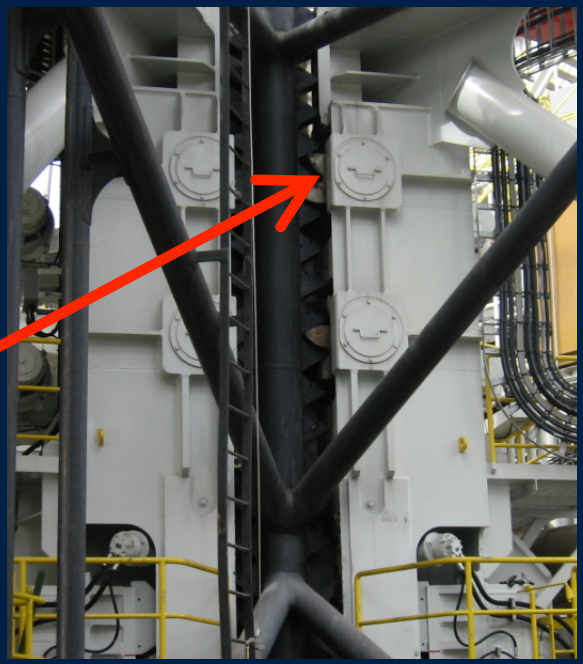
UNIVERSITY OF
OXFORD

Houlsby

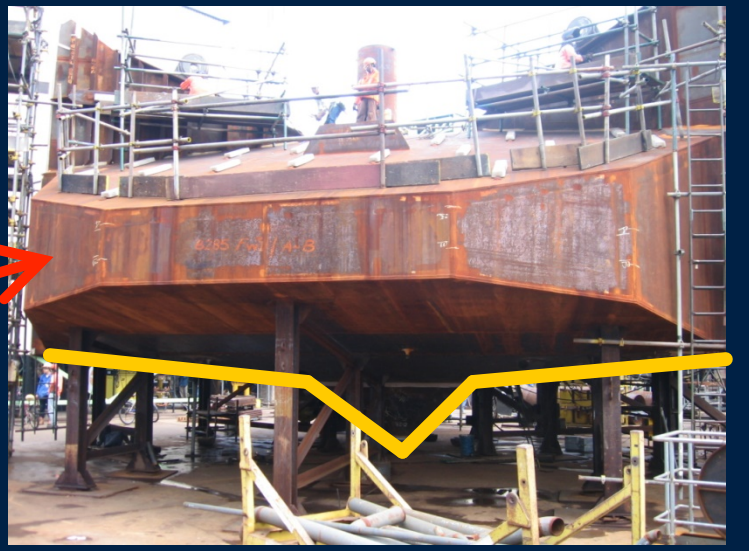
Jack-Up Units

60m

Maersk



Keppel FELS

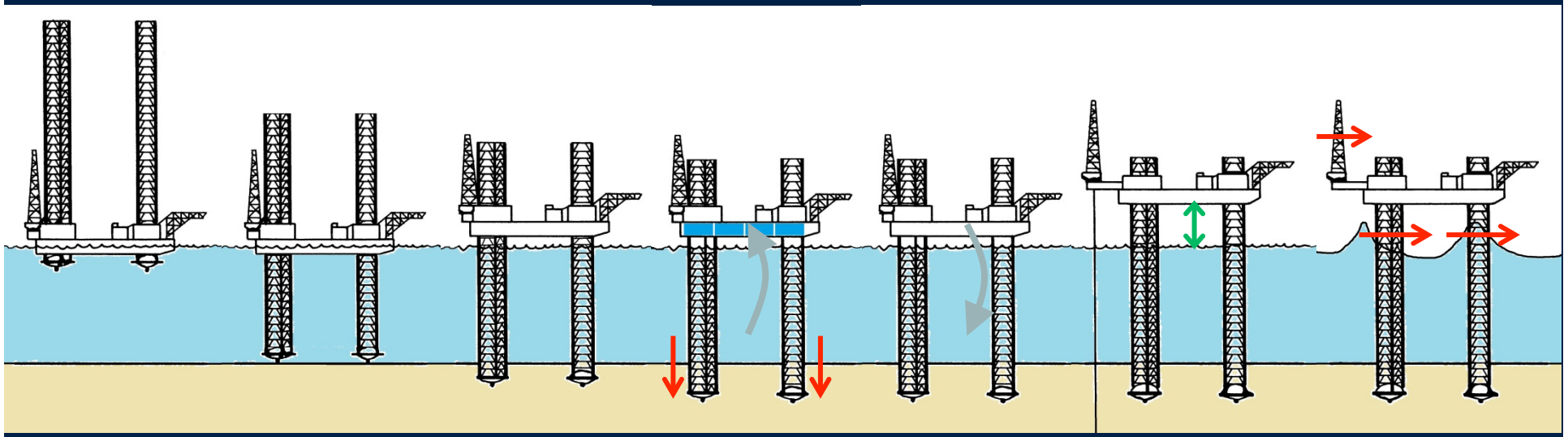


Keppel FELS

UNIVERSITY OF OXFORD

Slide from Houlsby

Jack-up operations



Float to site | Lower legs | Light ship load | Preload | Dump preload | Climb to air-gap and operate | Storm

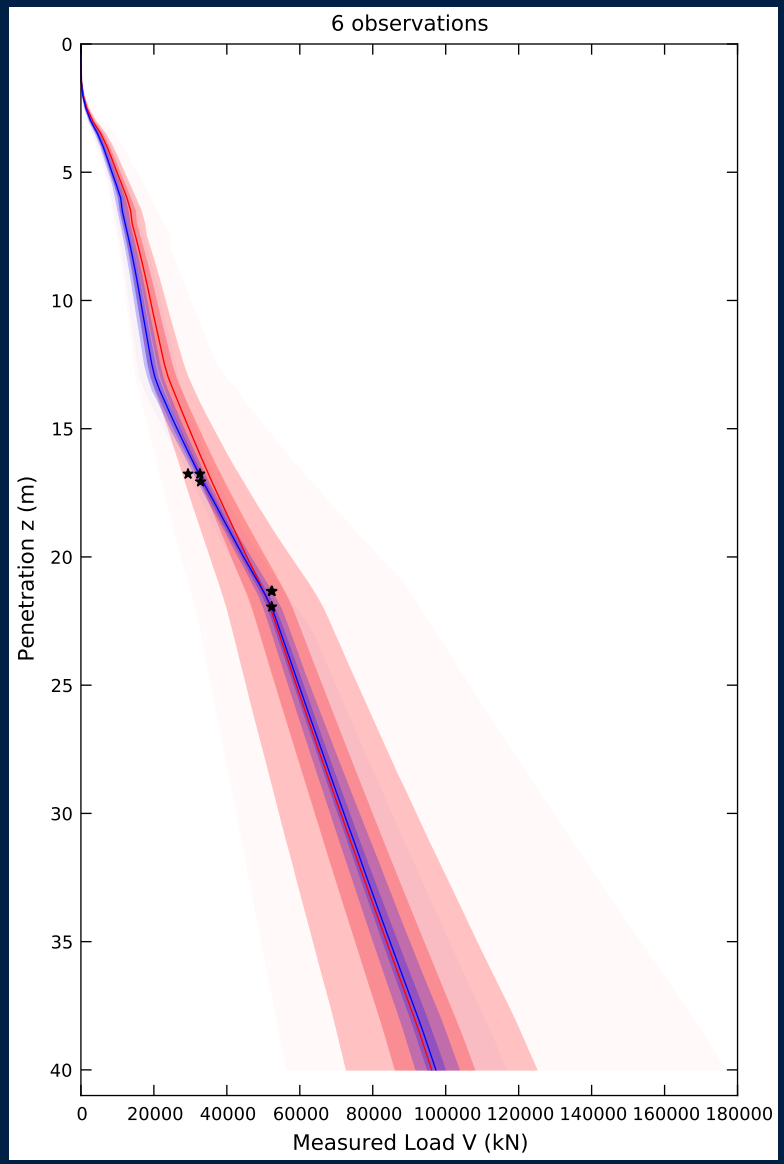
UNIVERSITY OF OXFORD

sketches after Poulos (1988)
Slide from Houlsby

Forward Spudcan Simulation / Inference

- Deterministic simulation
  - ~750 lines of C code
- Stochastic simulation
  - ~900 lines of C code
- Inverse simulation
  - +15 lines of C code

- ~1000 samples / second



UNIVERSITY OF OXFORD

Conclusion

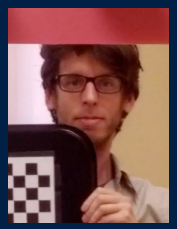
- Forward inference methods for probabilistic programming
  - Easy to understand and implement
    - Requires only POSIX abstraction
  - Automatic parallel scalability
  - No new language skills required
    - Little new programming in some cases

- Thanks to the team

van de Meent

Perov

Paige



collaborators

Mansinghka

and sponsors

UNIVERSITY OF OXFORD

xerox Research Centre Europe

DARPA

Google

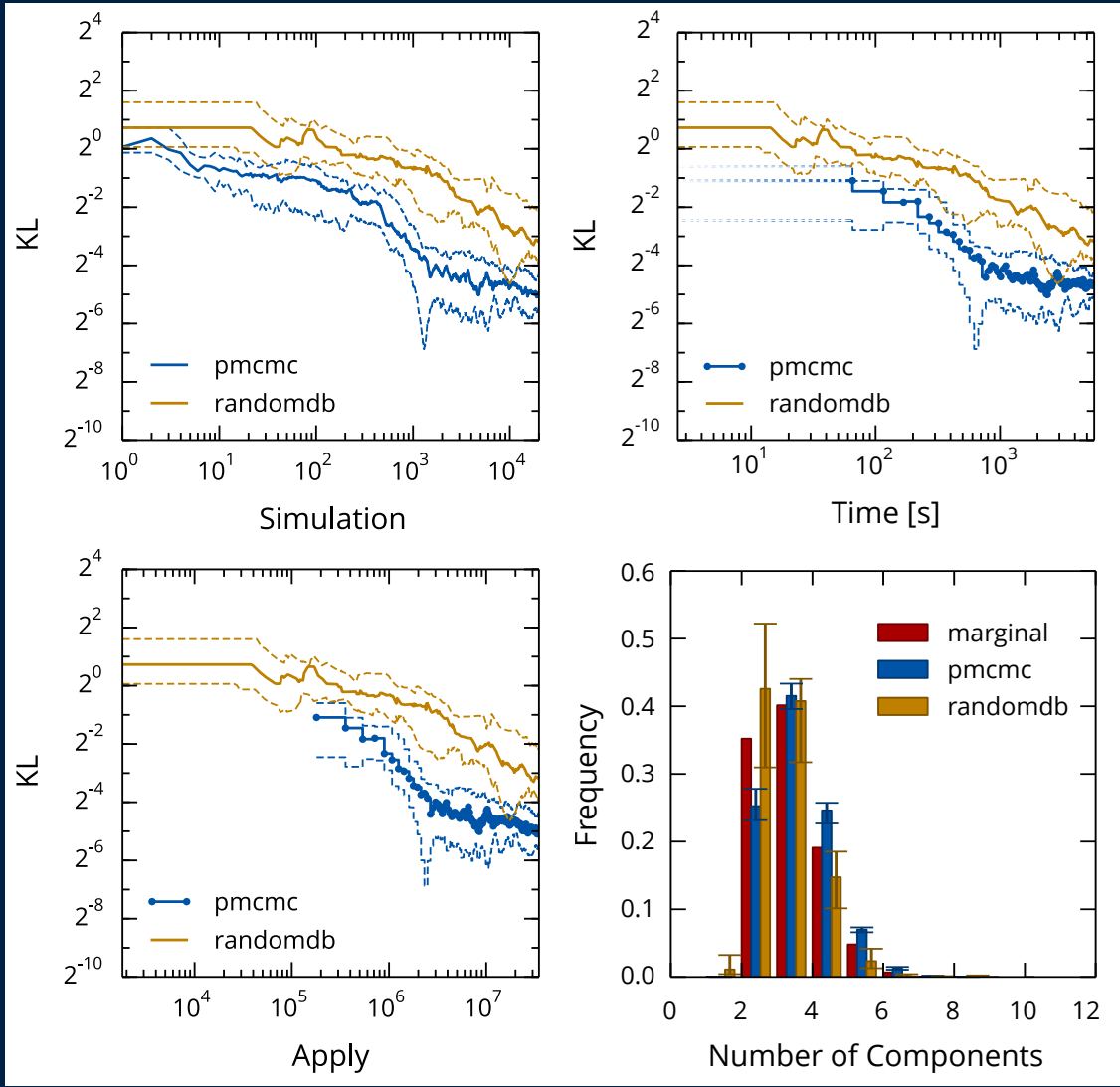
bp

DP Mixture Code

```
[assume class-generator (crp 1.72)]
[assume class (mem (lambda (n) (class-generator)))]
[assume var (mem (lambda (c) (* 10 (/ 1 (gamma 1 10)))))]
[assume mean (mem (lambda (c) (normal 0 (var c))))]
[assume u (lambda () (list (class 1) (class 2) ...
  (class 9) (class 10)))]
[assume K (lambda () (count (unique (u))))]
[assume means (lambda (i c)
  (if (= i c) (list (mean c))
              (cons (mean i) (means (+ i 1) c) )))]
[assume stds (lambda (i c)
  (if (= i c) (list (sqrt (* 10 (var c))))
    (cons (var i) (stds (+ i 1) c) )))]
[observe (normal (mean (class 1)) (var (class 1))) 1.0]
[observe (normal (mean (class 2)) (var (class 2))) 1.1]
⋮
[observe (normal (mean (class 10)) (var (class 10))) 0]
[predict (u)]
[predict (K)]
[predict (means 1 (K))]
[predict (stds 1 (K))]
⋮
```

UNIVERSITY OF
OXFORD

DP Mixture Results



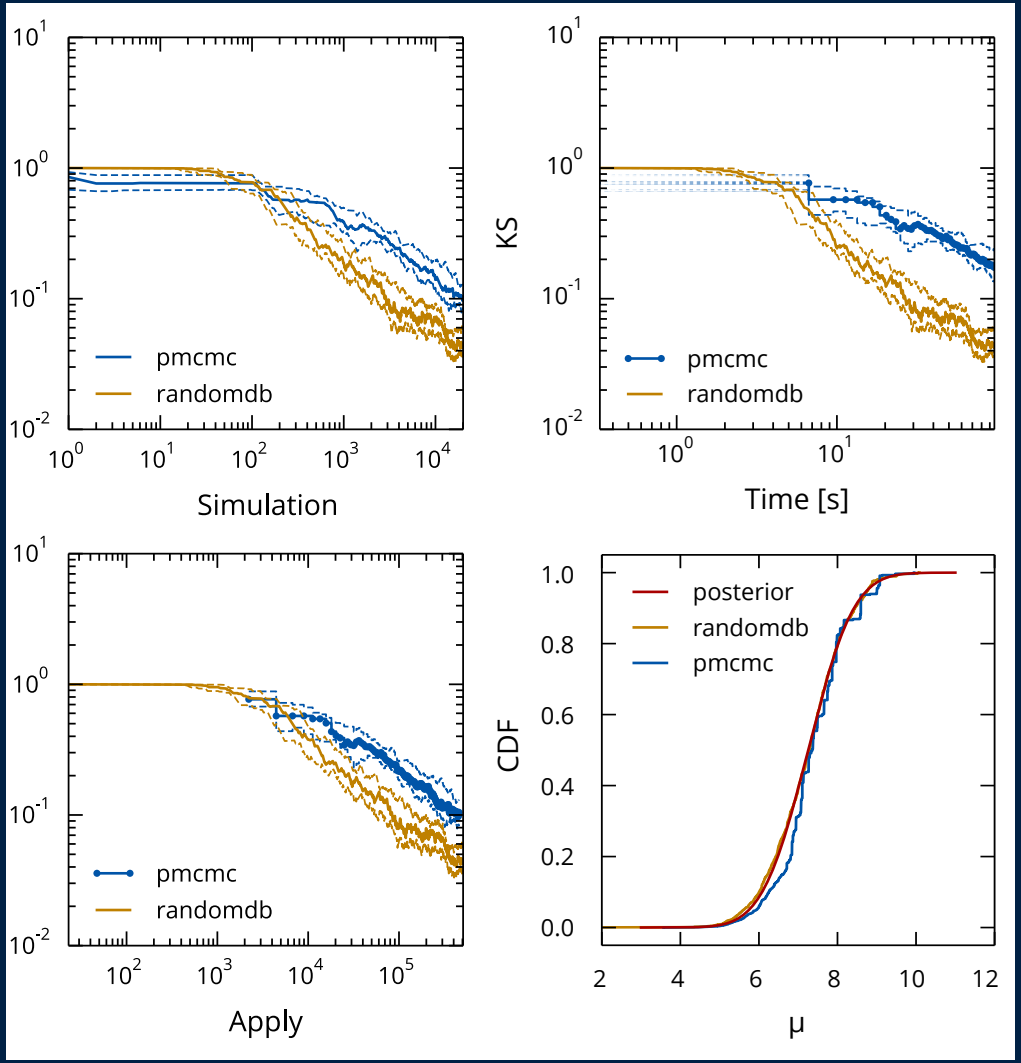
UNIVERSITY OF OXFORD

Marsaglia Code

```
[assume (marsaglia-normal mean var)
 (begin
  (define x (uniform-continuous -1.0 1.0))
  (define y (uniform-continuous -1.0 1.0))
  (define s (+ (* x x) (* y y)))
  (if (< s 1)
   (+ mean (* (sqrt var)
              (* x (sqrt (* -2 (/ (log s) s))))))
   (marsaglia-normal mean var)))]
[assume sigma-squared 2]
[assume mu (marsaglia-normal 1 5)]
[observe (normal mu sigma-squared) 9]
[observe (normal mu sigma-squared) 8]
[predict mu]
⋮
```

UNIVERSITY OF
OXFORD

Marsaglia Results



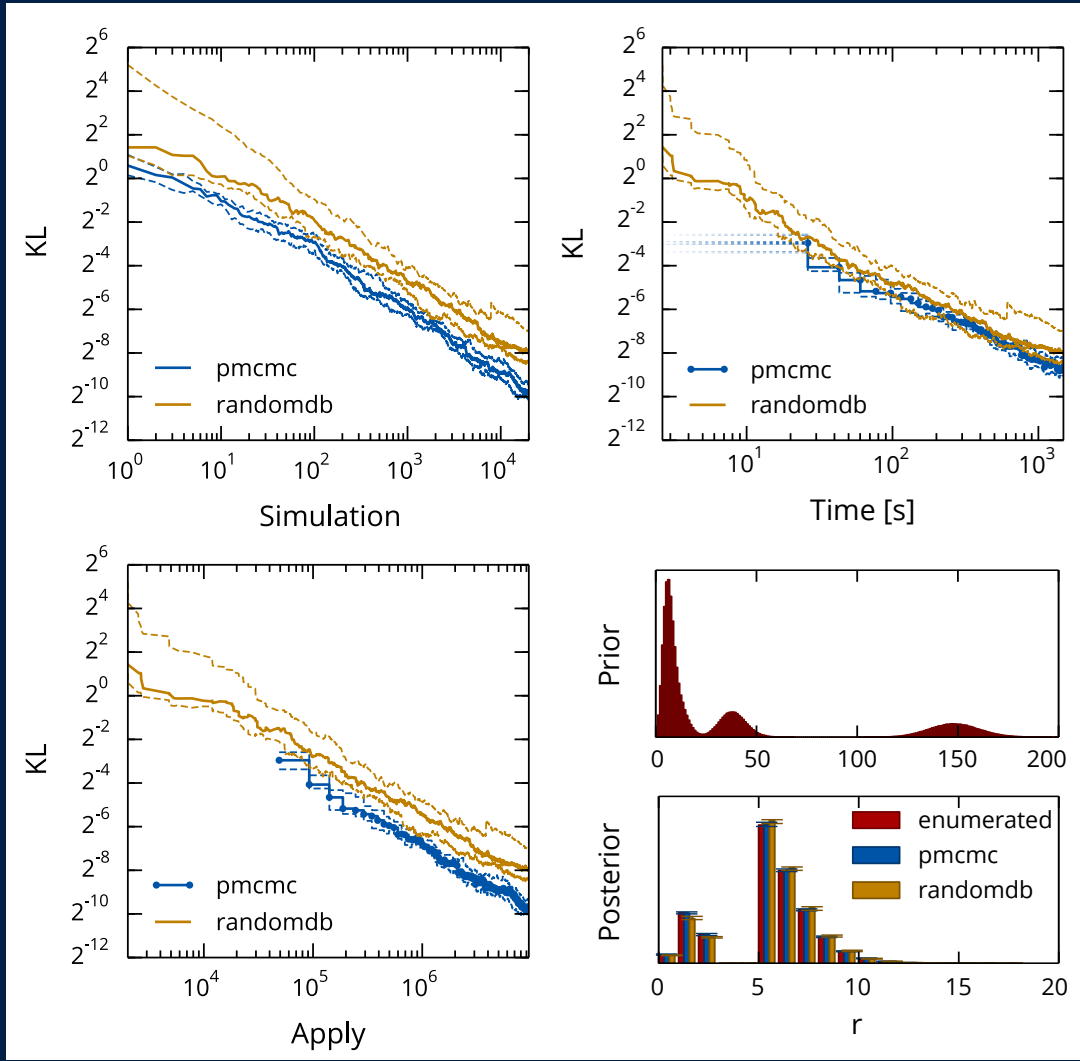
UNIVERSITY OF OXFORD

Branching Code

```
[assume fib (lambda (n)
  (cond ((= n 0) 1) ((= n 1) 1)
                (else (+ (fib (- n 1)) (fib (- n 2))))))]
[assume r (poisson 4)]
[assume l (if (< 4 r) 6 (+ (fib (* 3 r)) (poisson 4)))]
[observe (poisson l) 6]
[predict r]
⋮
```

UNIVERSITY OF
OXFORD

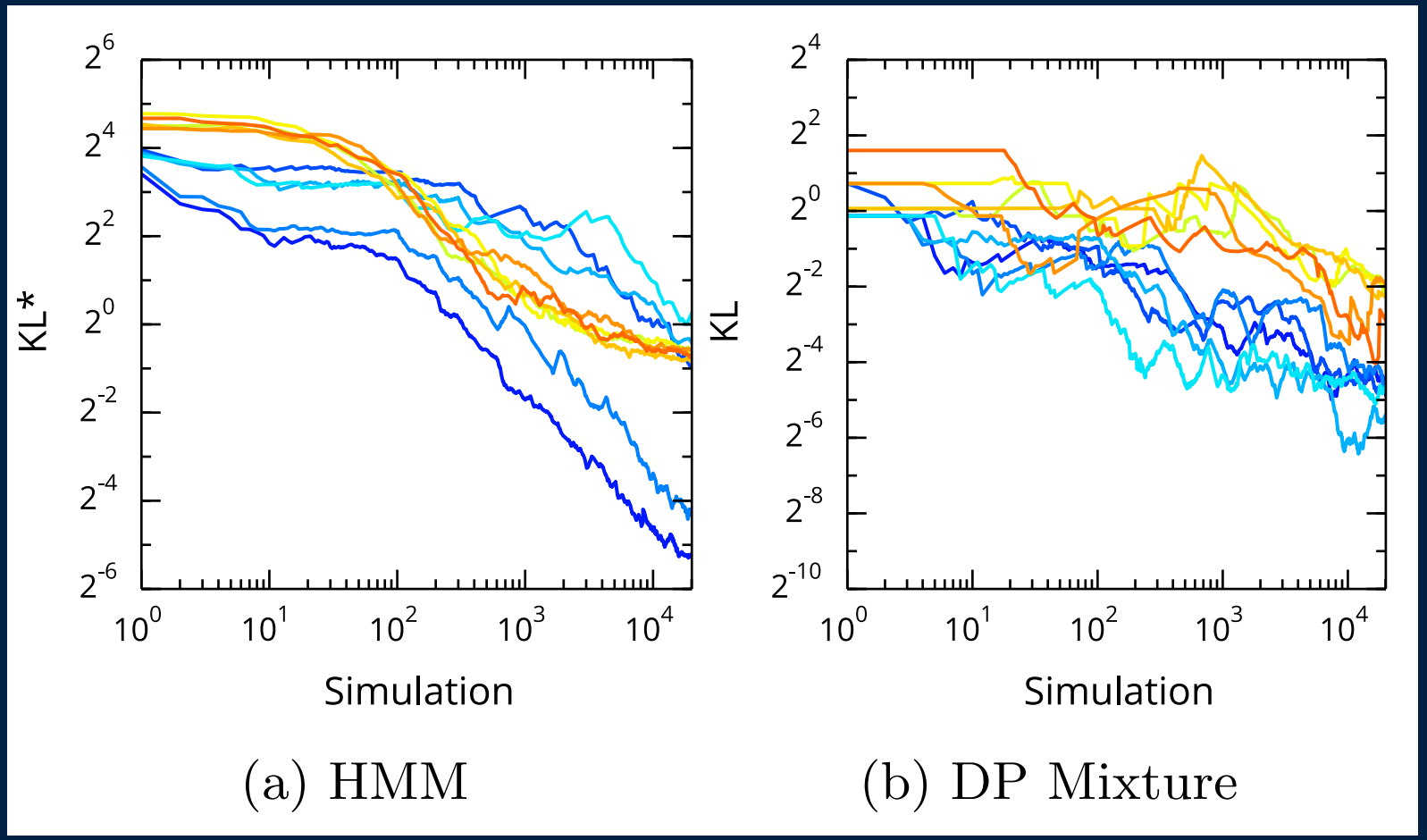
Branching Results



UNIVERSITY OF OXFORD

Opportunity

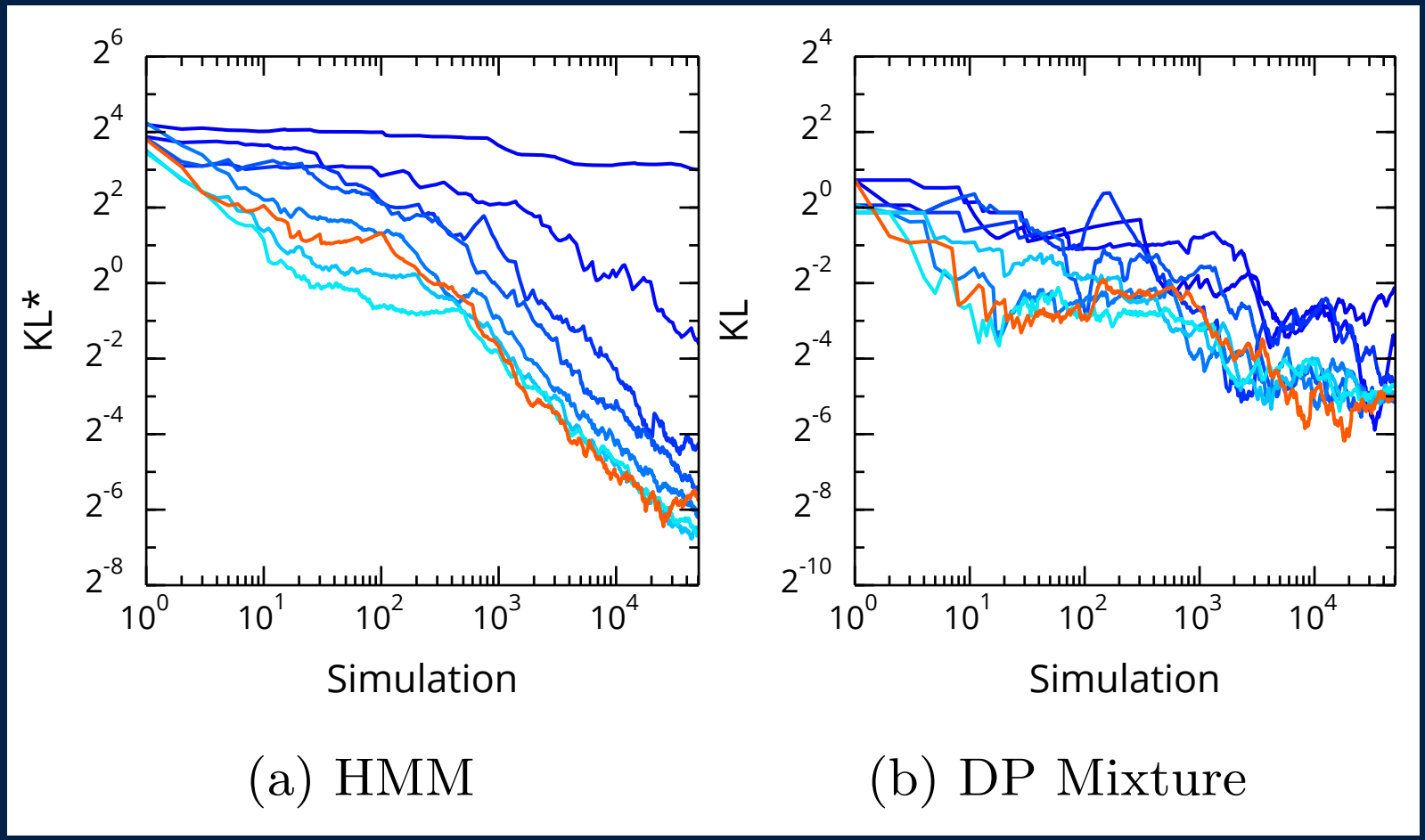
Inference optimization by program reordering



(a) HMM (b) DP Mixture

Another Opportunity

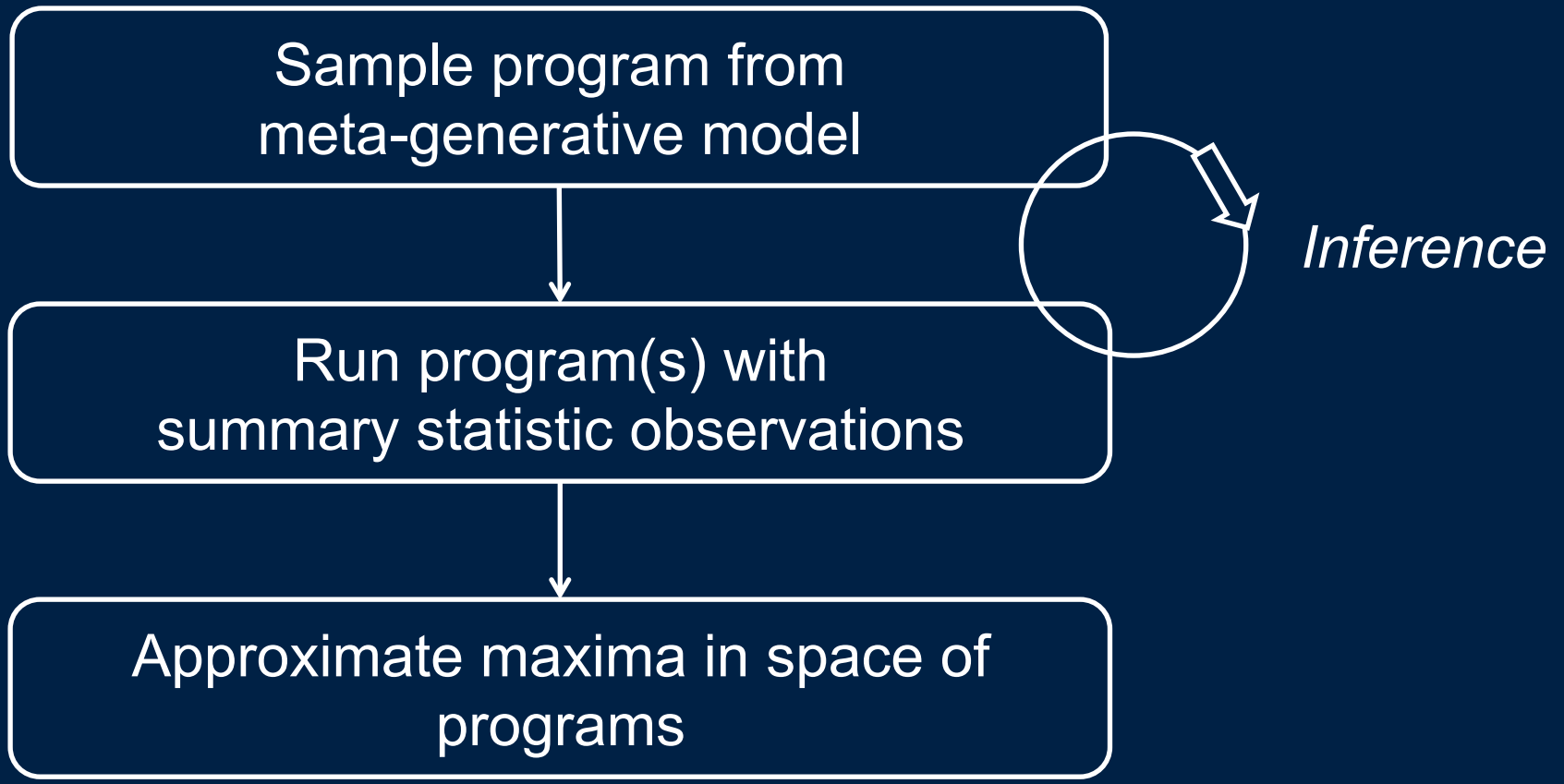
Parallelism



(a) HMM (b) DP Mixture

UNIVERSITY OF OXFORD

Expressivity Case Study : Probabilistic Program Synthesis



UNIVERSITY OF OXFORD

Meta-Program for Prob. Prog. Synthesis

```
[ASSUME productions …]
[ASSUME expression (list 'lambda '(…) (productions …))]
[ASSUME my-sampler (eval expression)]


[OBSERVE (mean (apply-n (my-sampler 5.7 3.5) 100)) 5.7]
[OBSERVE (std  (apply-n (my-sampler 5.7 3.5) 100)) 3.5]
[OBSERVE (kurt (apply-n (my-sampler 5.7 3.5) 100)) 0.0]
[OBSERVE (skew (apply-n (my-sampler 5.7 3.5) 100)) 0.0]


[PREDICT expression]
or
[PREDICT (apply-n (my-sampler -3.5 7.2) 100)]
```

Synthesis of Probabilistic Programs

Production rules for one meta-generator of samplers

```
[ASSUME productions
  (lambda (mysymbol level)
    (if (= mysymbol 'expr-real)
      ((lambda (output)
        (if (= output 0)
          (list 'get-real-constant (get-real-constant-id))
          (if (= output 1)
            (nth (list 'a 'b) (categorical (list 0.5 0.5)))
            (if (= output 2)
              (list (nth (list 'uniform-continuous (quote +) (quote *) (quote safe-div)) (categorical (list 0.25 0.25 0.25 0.25)))
                    (productions 'expr-real (+ level 1)) (productions 'expr-real (+ level 1)))
              (if (= output 3)
                (list (nth (list 'sqrt 'safe-log) (categorical (list 0.5 0.5))) (productions 'expr-real (+ level 1)))
                (if (= output 4)
                  (list 'if (list (quote <) 'stack-id 10)
                        (list 'my-sampler (productions 'expr-real (+ level 1))
                              (productions 'expr-real (+ level 1)) (list (quote +) 'stack-id 1)) 0.0)
                  (list 'if (productions 'expr-bool (+ level 1)) (productions 'expr-real (+ level 1))
                        (productions 'expr-real (+ level 1)))))))))
       (categorical
         ((lambda (a b)
           (list a a b b b b b b)
         ) (/ (- 1 (power 0.75 level)) 2) (/ (power 0.75 level) 4))))
       (if (= mysymbol 'expr-bool)
         (list (quote <) (productions 'expr-real (+ level 1)) (productions 'expr-real (+ level 1)))))))]
```

Synthesis of Probabilistic Programs

Sampled samplers (i.e. probabilistic generative programs)

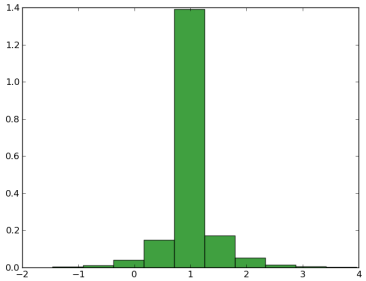
```
[ASSUME my-sampler
  (lambda (a b stack-id)
    (uniform-continuous
      (get-real-constant 1)
      (if (< (if (< (safe-log (get-real-constant 2)) b)
                 (get-real-constant 1) (get-real-constant 3)) a)
        (safe-log (if (< (get-real-constant 3) (get-real-constant 1))
                      (get-real-constant 1) a))
        b))))]
```

```
[ASSUME my-sampler
  (lambda (a b stack-id)
    (safe-sqrt (uniform-continuous a (get-real-constant 1)))))]
```

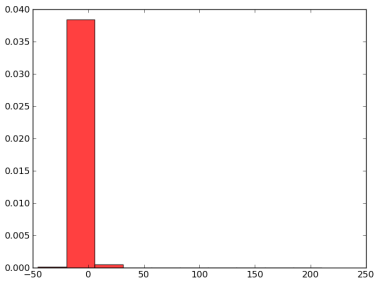
Synthesis of Probabilistic Programs

Samples from sampled samplers

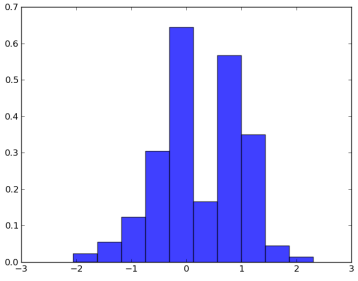
```
[ASSUME my-sampler
  (lambda (a b stack-id)
    (+ (uniform-continuous
         (if (< b (get-real-constant 1))
           a
           (safe-div
             (get-real-constant 2)
             (if (< (get-real-constant 2) b)
               a
               (if (< (get-real-constant 2) (get-real-constant 2))
                 (get-real-constant 2)
                 (safe-log b)))))
         (if (< (get-real-constant 2) (safe-div b a))
           (safe-sqrt (safe-sqrt
                        (if (< (get-real-constant 2) (get-real-constant 2))
                          (get-real-constant 1)
                          (get-real-constant 2))))
           (if (< a (get-real-constant 2)) (get-real-constant 3) a))) b))]
```




```
[ASSUME my-sampler
  (lambda (a b stack-id)
    (uniform-continuous
      b
      (uniform-continuous
        (if (< (if (< b (if (< a (get-real-constant 1))
                          b (get-real-constant 2))) (sqrt a) b) a)
          (if (< stack-id 10)
            (my-sampler (get-real-constant 3)
                        (+ a (get-real-constant 2)) (+ stack-id 1))
            0.0)
          (if (< (sqrt b) (get-real-constant 4))
            (if (< a b)
              (* (get-real-constant 5) b)
              (get-real-constant 4))
            b))
        (safe-div
          (get-real-constant 1)
          (if (< a b) (get-real-constant 4) (safe-log b)))))))]
```




```
[ASSUME my-sampler
  (lambda (a b stack-id)
    (uniform-continuous
      (get-real-constant 1)
      (if (< (if (< (safe-log (get-real-constant 2)) b)
                 (get-real-constant 1) (get-real-constant 3)) a)
        (safe-log (if (< (get-real-constant 3) (get-real-constant 1))
                    (get-real-constant 1) a))
        b)))]
```




```
[ASSUME my-sampler
  (lambda (a b stack-id)
    (safe-sqrt (uniform-continuous a (get-real-constant 1))))]
```