The Turing language for probabilistic programming

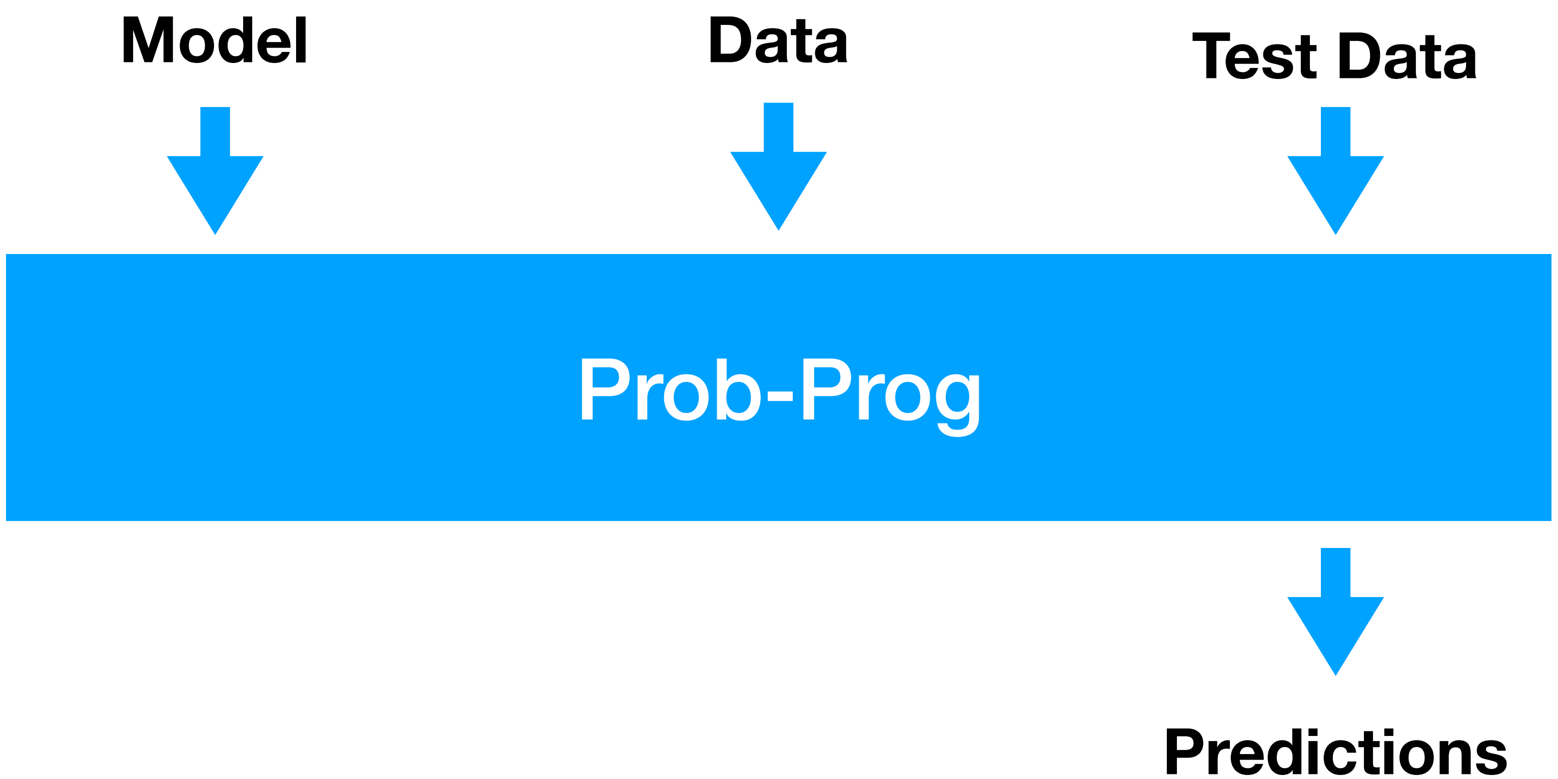
Hong Ge[1], Kai Xu[2]

Oct 2018

The first international conference on probabilistic programming

1: Department of Engineering, University of Cambridge
2: School of Informatics, University of Edinburgh

Talk plan



Probabilistic programming languages

- Probabilistic programs: computer programs represent probabilistic models with probabilistic statements:
  - Declaring *random variables*
  - Conditioning on observed data

- Universal probabilistic programming
  - Stochastic control flows
  - Allows representing arbitrary probabilistic models

- Generic inference engines: HMC, SMC, particle Gibbs, EP

- Two approaches to implement a PPL
  - Standalone: Stan, BUGS, Venture, etc
  - Embedded: Anglican, infer.NET, PyMC3, Pyro, Edward, **Turing**, etc

Talk plan

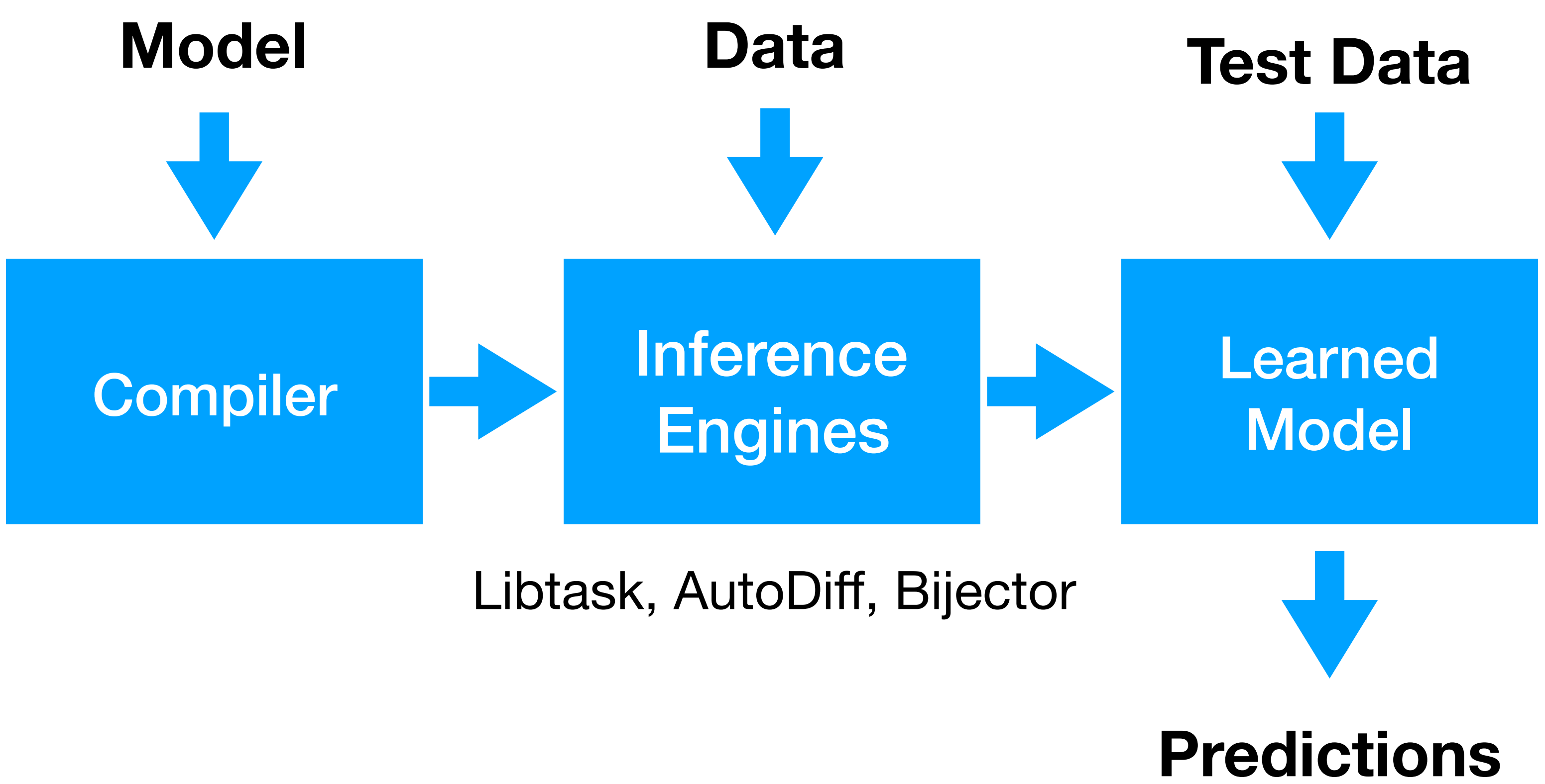
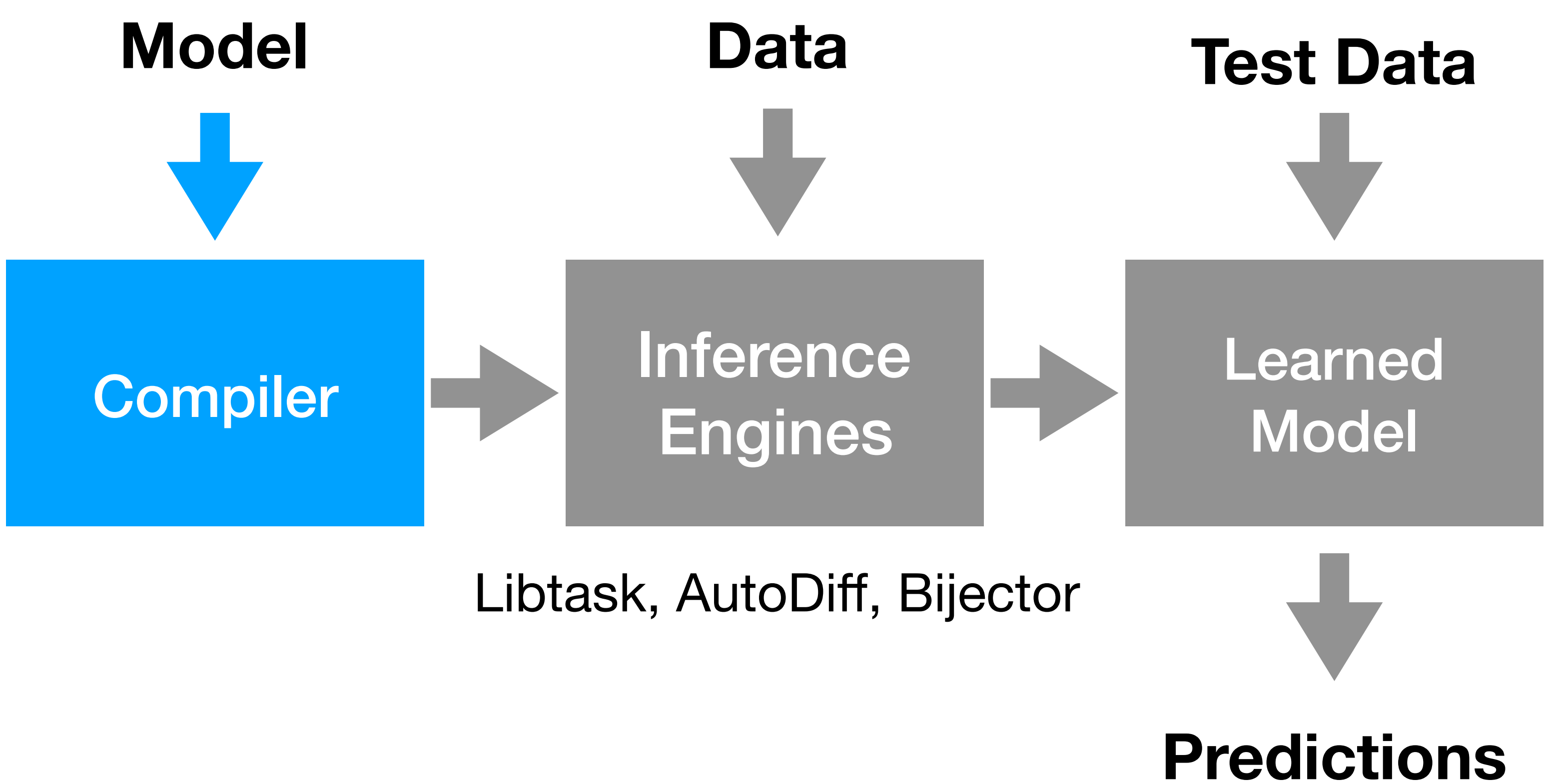


Fig 1. Workflow & components of Turing

Talk plan



Model
Data
Test Data
Compiler
Inference Engines
Learned Model
Libtask, AutoDiff, Bijector
Predictions

The modelling language in Turing

```julia
@model gdemo(x) = begin
  s ~ InverseGamma(2, 3)
  m ~ Normal(0, sqrt(s))
  for i = 1:length(x)
    x[i] ~ Normal(m, sqrt(s))
  end
  return s, m
end
```

Fig2: Simple Gaussian Model in Turing

The modelling language in Turing

2 `gdemo(x)` defines a Julia function.

1 `@model` translates a normal Julia program into a Turing model.

3 Observations are declared as the parameters in the function definition.

4 When the left hand side of `~` is not declared as an observation, the statement defines a random variable.

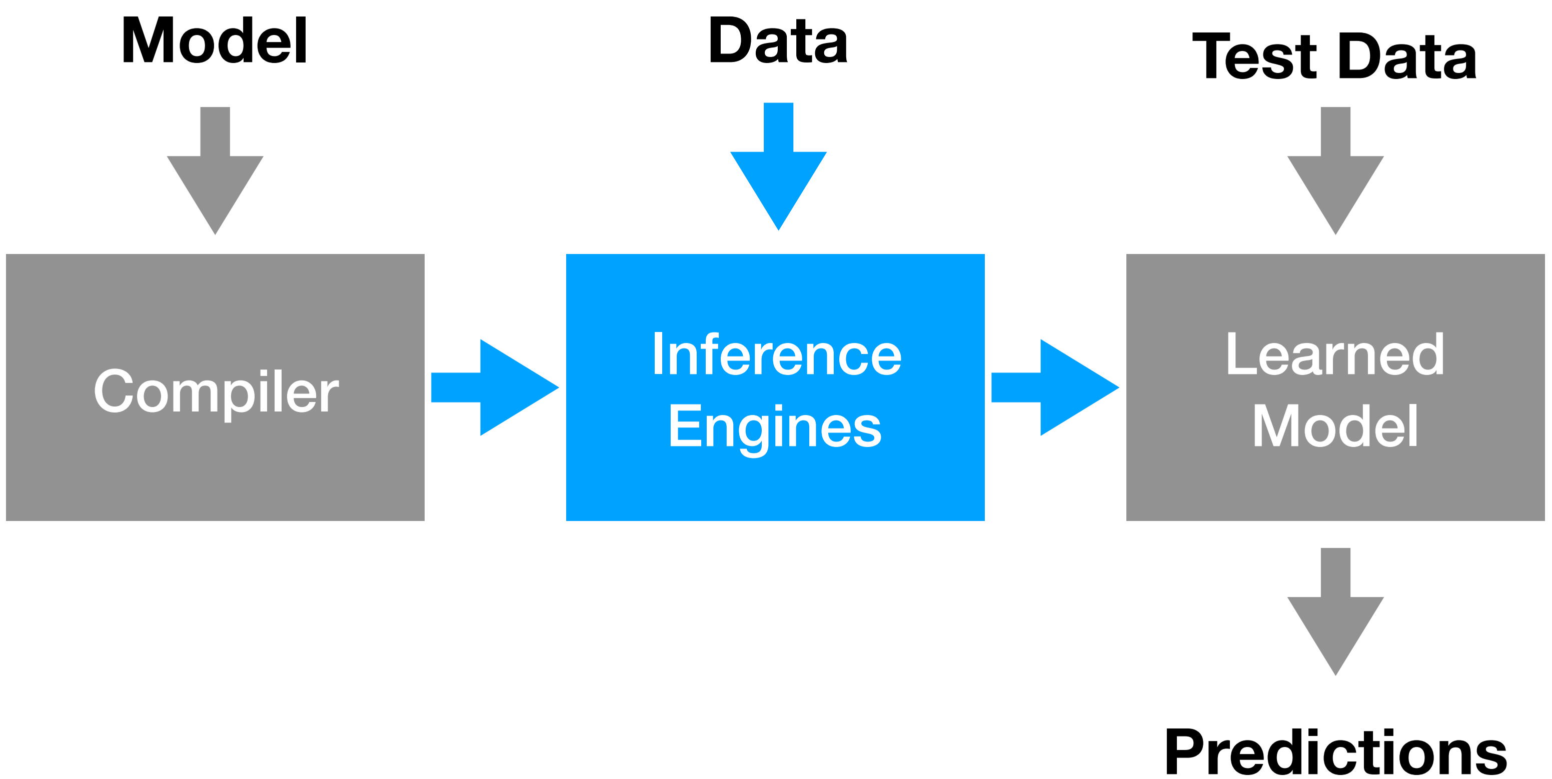
5 When the left hand side of `~` is an observation, it performs conditioning.

```
@model gdemo(x) = begin
   s ~ InverseGamma(2, 3)
   m ~ Normal(0, sqrt(s))
   for i = 1:length(x)
      x[i] ~ Normal(m, sqrt(s))
   end
   return s, m
end
```

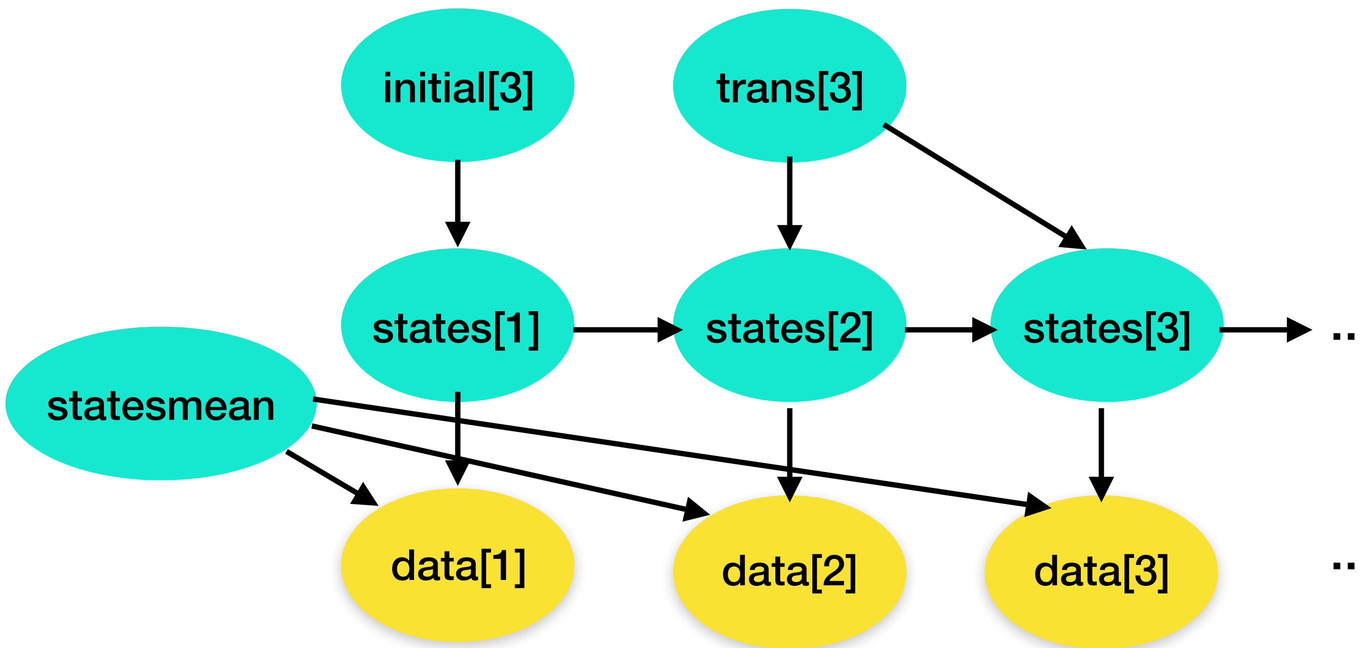
Everything else follows standard Julia syntax.

Fig3: Illustration of Turing's syntax

Talk plan


Model
Data
Test Data
Compiler
Inference Engines
Learned Model
Predictions

Simulation-based inference



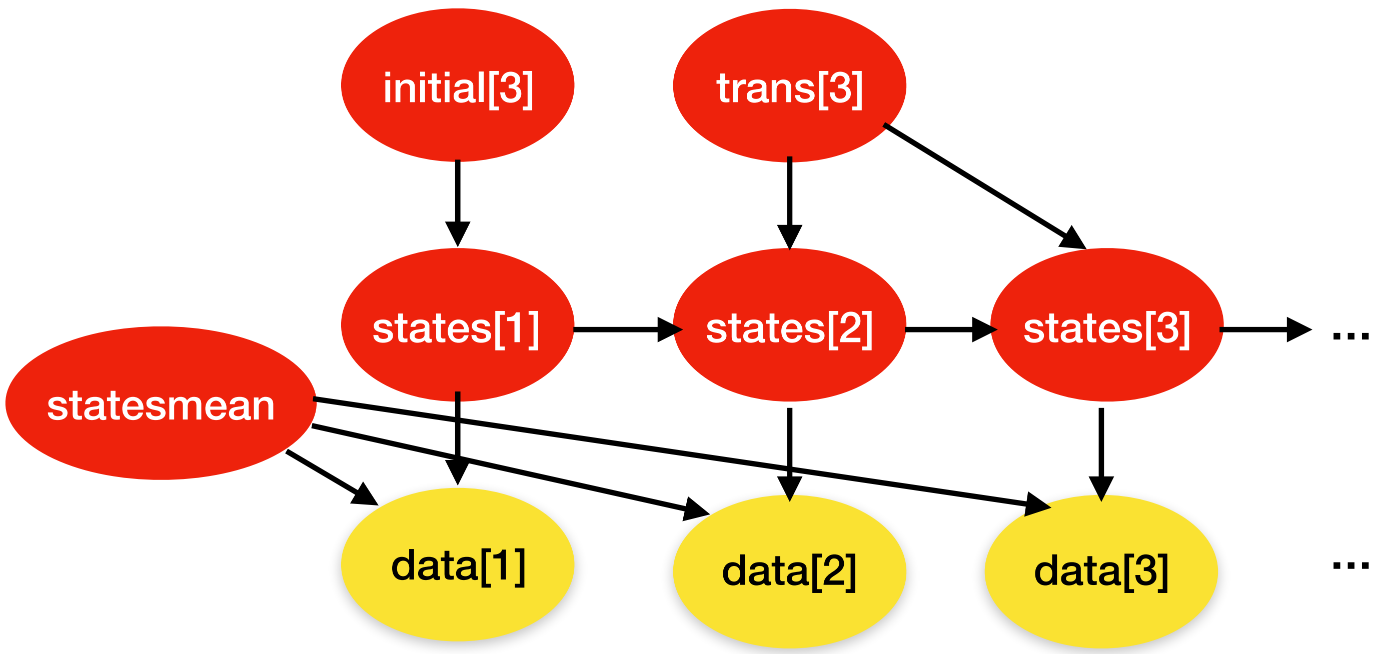
- Sample all variables using a forward simulation method
  - Sequential Monte Carlo
  - Particle MCMC
  - single-site MH, …
- *Universal*: applicable to models with stochastic control flows

Related work: Church, WebPPL, Venture, Anglican, Turing

Remove related work

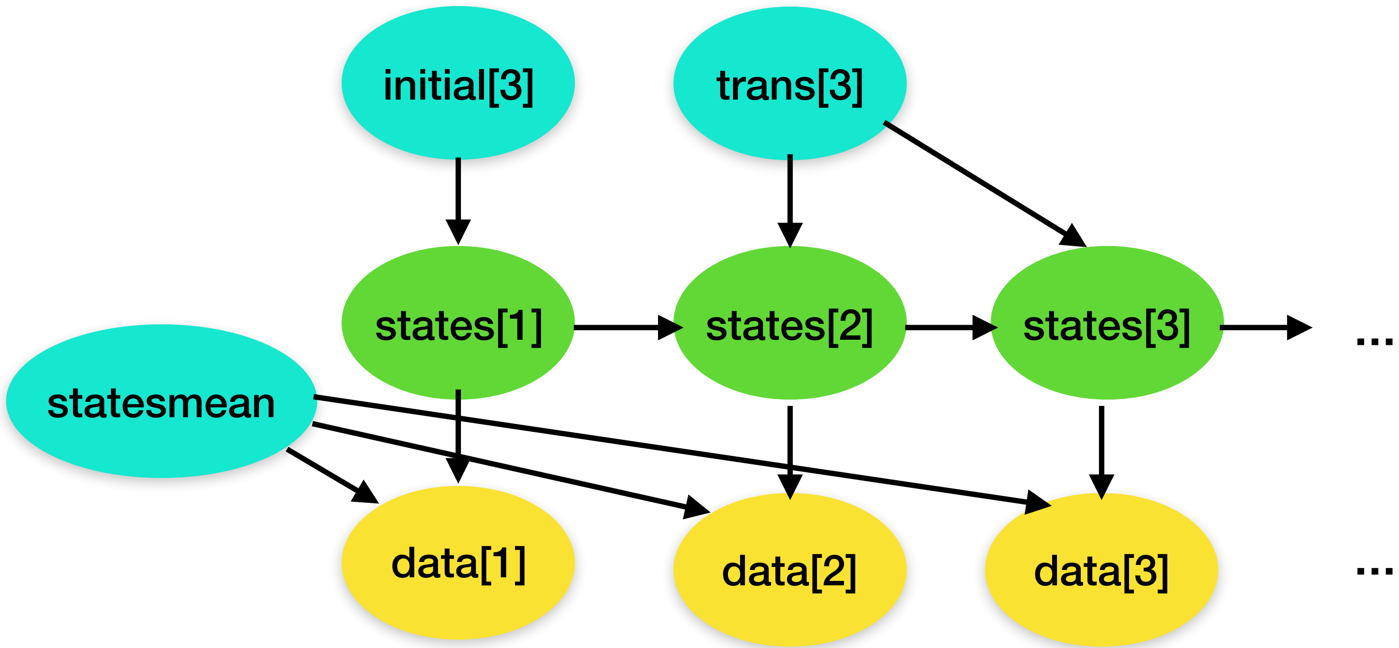
Gradient-based inference

- Sample all variables using a generic gradient guided algorithm, e.g.:
  - *HMC (NUTS)*
  - *Blackbox variational inference*
- *Non-universal*:
  - No stochastic control flows
  - No discrete variables



Compositional inference

- Combine simulation and gradient-based inference
- Generic universal engine



- Gibbs sampling for BayesHMM
  - Sample states using particle Gibbs
  - Sample initial, trans and statesmean using HMC

Basic inference in Turing

```
@model gdemo(x) =  begin
  s ~ InverseGamma(2,3)
  m ~ Normal(0,sqrt(s))
  for i=1:length(x)
    x[i] ~ Normal(m, sqrt(s))
  end
  return(s, m)
end
```

TODO:
- adapt Guide section in Turing's doc
- add a slide on `SampleFromPrior`

Add some details on post-sampling processing with MCMCChain

```
mf = gdemo([1.5, 2])

alg = HMC(2000, 0.1, 10)

chain = sample(mf, alg)
```

1. **By passing data to a compiled model, we get a generated model function `mf`.**
2. **An inference algorithm is defined by its name and corresponding parameters.**
3. **The `sample` function takes a generated model function and a sampling algorithm to perform inference.**
4. **The returned value `chain` stores MCMC samples.**

Compositional inference in Turing

```
# Sampler = HamiltonianMonteCarlo + ParticleGibbs
g1 = Gibbs(500, HMC(1, 0.2, 3, :m), PG(50, 1, :s))
```

1 Gibbs is defined by number of iterations and multiple sampling algorithms as its components.

2 HMC is specified to sample variable m.

3 PG is specified to sample variable s.

Available algorithms in Turing

Move to conclusion section.

Sampler	Support discrete variables?	Require gradients?	Require adaption?	Support universal programs?	MCMC factory operator?
HMC	No	Yes	Yes	No	Yes
NUTS	No	Yes	Yes	No	Yes
IS	Yes	No	No	Yes	No
SMC	Yes	No	No	Yes	No
PG	Yes	No	No	Yes	Yes
PMMH	Yes	No	No	Yes	Yes
IPMCMC	Yes	No	No	Yes	Yes

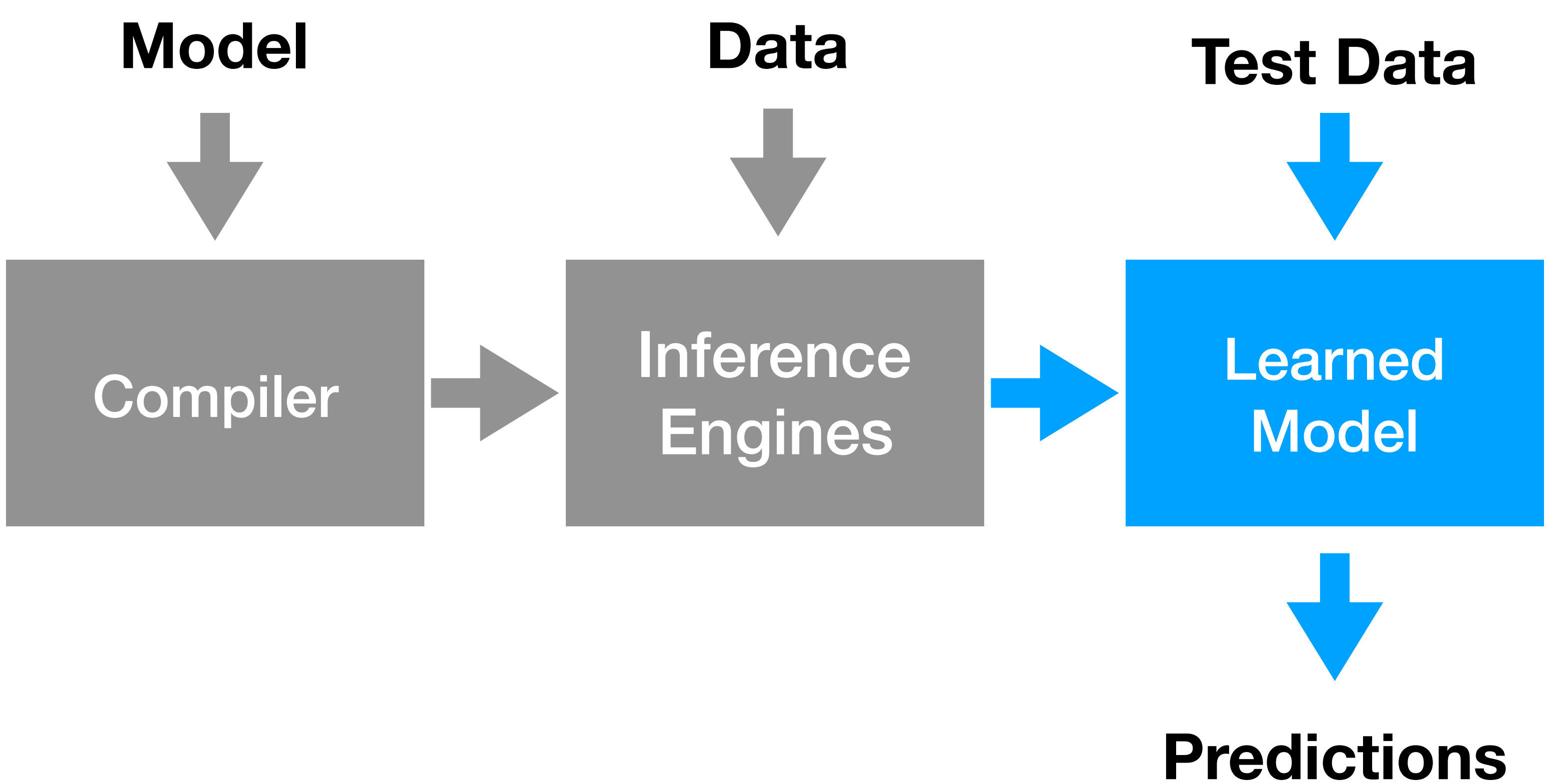
Current supported inference algorithms in Turing

- *Particle Gibbs* in Turing is a re-implementation of Wood (2014), with a more efficient mechanism for copying/forking particles.
- *Compositional inference* is closely related with Vikash (2014).

This should really not be an image but rather a svg or pdf. It's to blurry as is.

Also, I'm not sure how informative this table actually is. Maybe add this as an additional slide at the end?

Talk plan



Model
Data
Test Data
Compiler
Inference Engines
Learned Model
Predictions

Inference results

```julia
g = Gibbs(500, HMC(1, 0.2, 3, :s),
               PG(10, 1, :m))

# Run the sampler
c = sample(gdemo(1.5, 2), g);
```

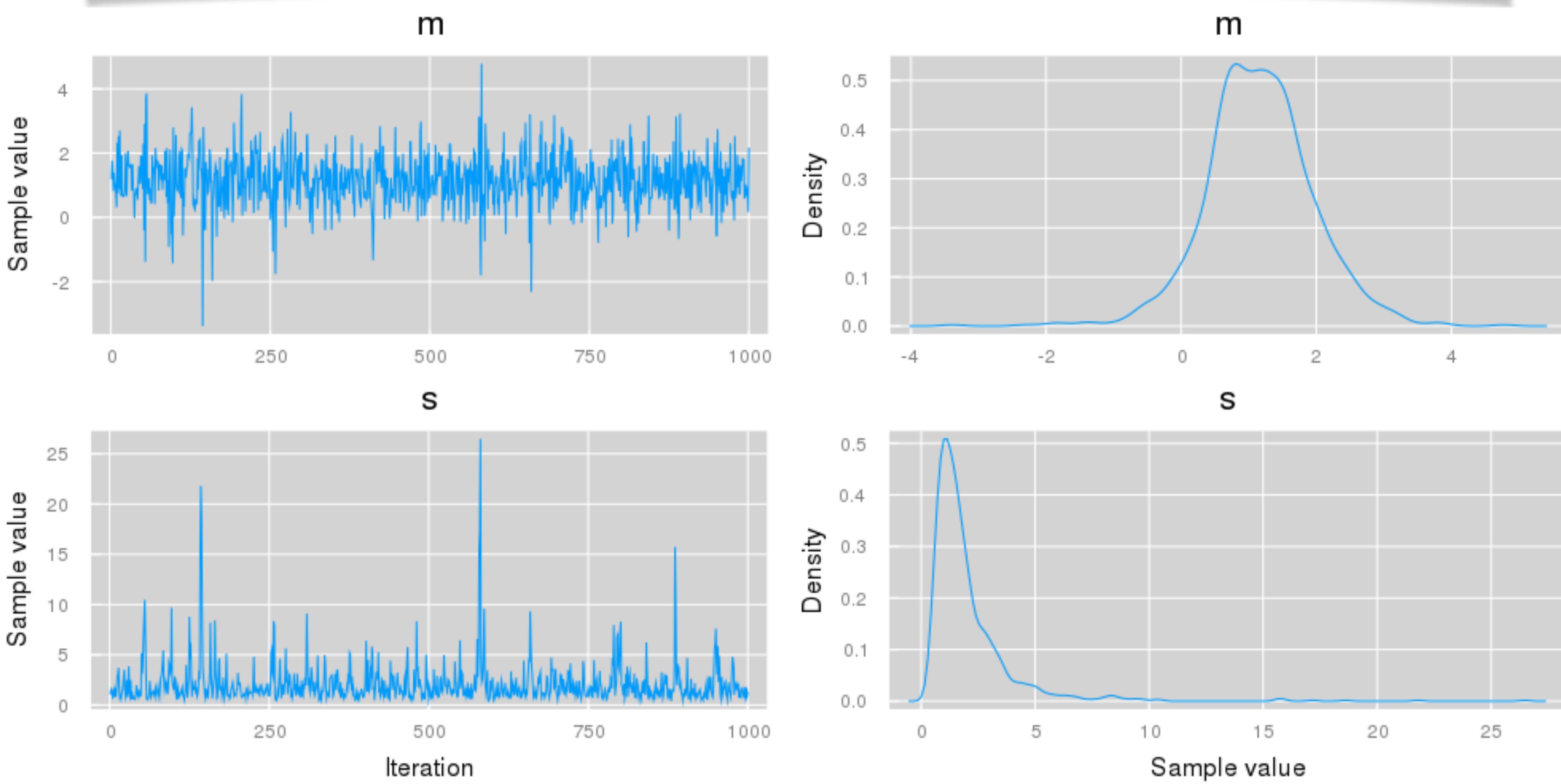
```
julia> c = sample(gdemo(1.5, 2), g)
[ Info:  Assume - `s` is a parameter
[ Info:  Assume - `m` is a parameter
[ Info:  Observe - `x` is an observation
[ Info:  Observe - `y` is an observation
[Gibbs] Sampling...100% Time: 0:00:04
[ Info: [Gibbs] Finished with
[ Info:   Running time    = 4.406516913999995;
Object of type "Turing.Chain{AbstractRange{Int64}}"

Iterations = 1:1000
Thinning interval = 1
Chains = 1
Samples per chain = 1000

[1.19424 0.0 … 0.1 0.0; 1.76147 5.0 … 0.1 -5.04962; … ; 0.16521 5.0 … 0.1 -6.34745;
2.17485 5.0 … 0.1 -5.78878]
```

I would drop this slide.

Maybe it would be more interesting to have a GP example? This way we could highlight how Turing interacts with other packages.

Inference results

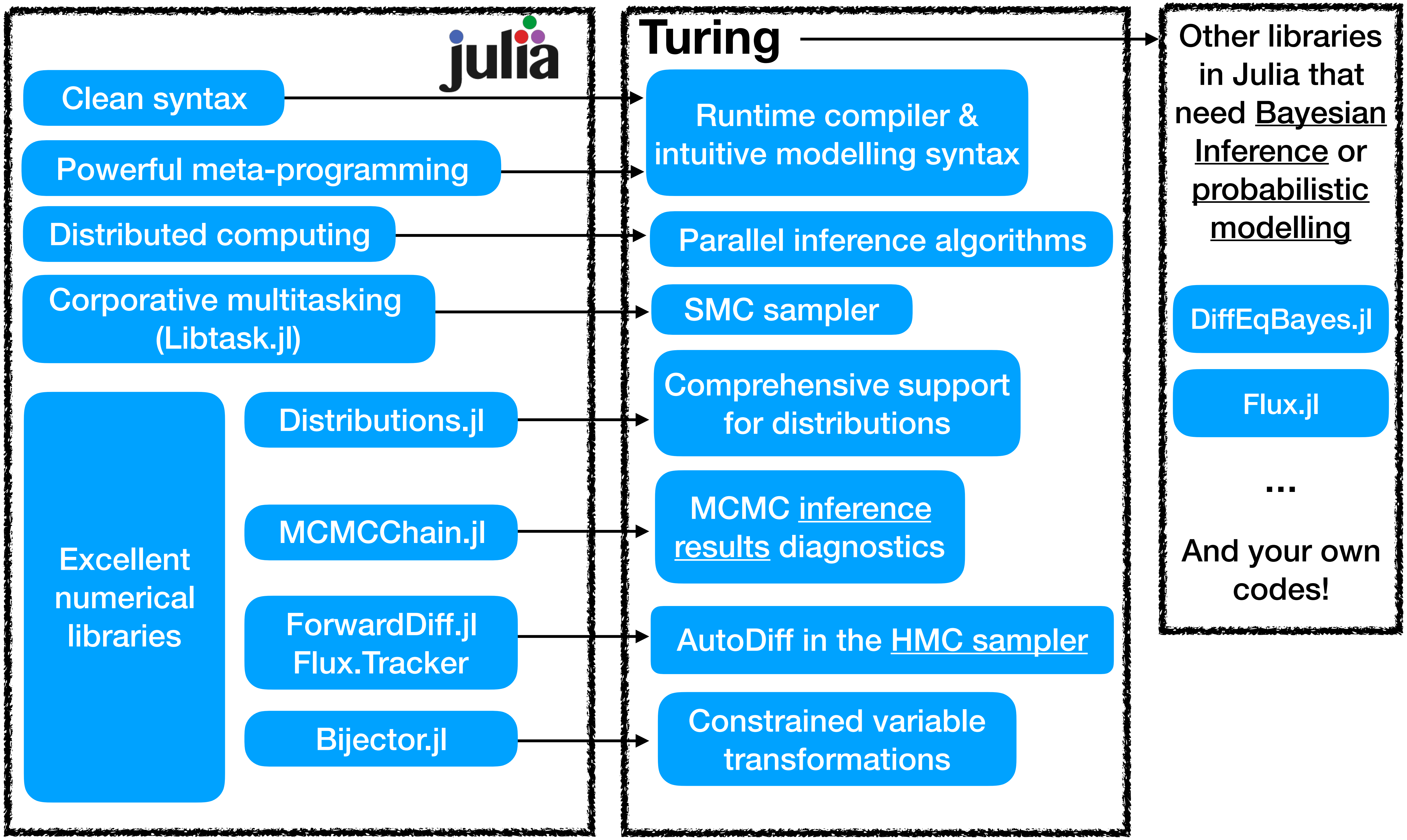
```
julia> plot(c)
```



Inference results

```
julia> describe(c)
Iterations = 1:1000
Thinning interval = 1
Chains = 1
Samples per chain = 1000

Empirical Posterior Estimates:
          Mean            SD                   Naive SE                 MCSE           ESS
      m  1.159092440 0.80245398600286965695716 0.02537582313249944071520400 0.018075292012779801270605 1000.00000
 lf_num  4.995000000 0.15811388300841913712169 0.005000000000000044408921 0.004999999999999827568486 1000.00000
      s  2.074526995 2.06522391559965390328557 0.06530811451546250645527890 0.090128941322170613714703 8  525.05615
elapsed  0.004406517 0.04087875130208082352645 0.001292699620181492333945 2 0.001808159452742373694139 6  511.11872
epsilon  0.100000000 0.000000000000000013884732 0.000000000000000043907378 0.000000000000000046259293  900.90090
     lp -5.751843644 1.16059043841524767159967 0.036701092160055633073501 0 0.047209064773478755239200 0  604.37596

Quantiles:
          2.5%        25.0%        50.0%        75.0%        97.5%
      m -0.4156437838  0.6751308439  1.1591866244  1.6194797994  2.797401556
 lf_num  5.0000000000  5.0000000000  5.0000000000  5.0000000000  5.000000000
      s  0.5592247213  1.0143592798  1.5257070660  2.3944259281  6.980157267
elapsed  0.0019876127  0.0022259065  0.0024100215  0.0026112668  0.007310805
epsilon  0.1000000000  0.1000000000  0.1000000000  0.1000000000  0.100000000
     lp -8.7877142810 -6.1846725212 -5.4480569586 -4.9603088831 -4.636854812
```

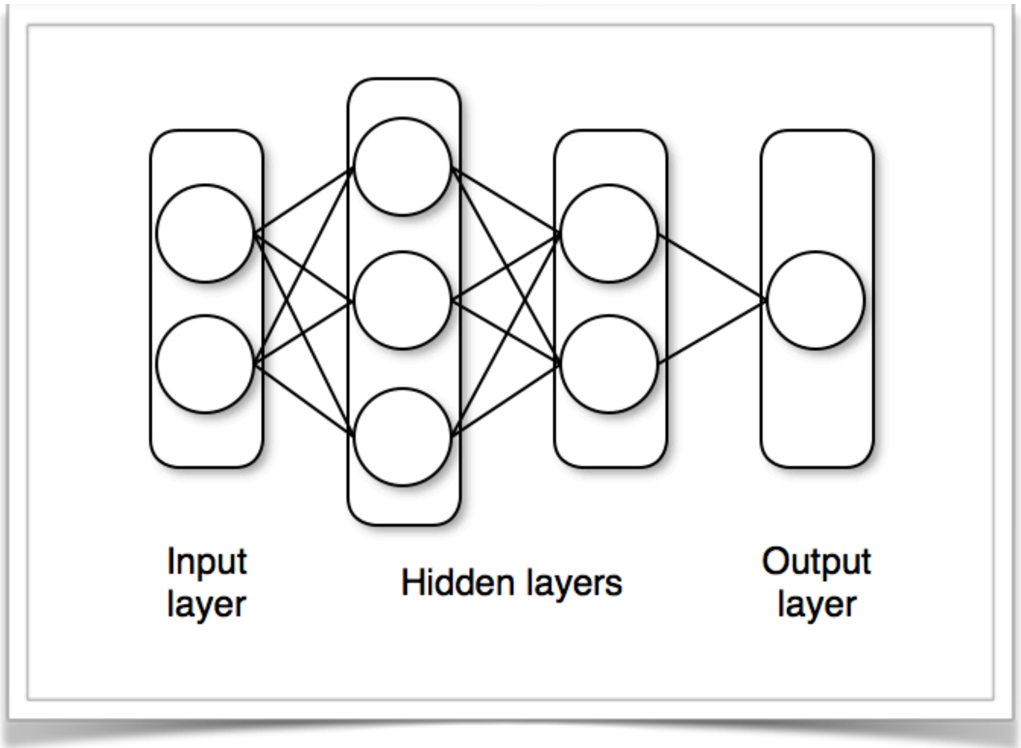
Probabilistic programming in Julia



Bayesian Deep Learning

```julia
function nn_forward(x, theta::AbstractVector)
    W₁, b₁, W₂, b₂, Wₒ, bₒ = unpack(theta)
    nn = Chain(Dense(W₁, b₁, tanh),
               Dense(W₂, b₂, tanh),
               Dense(Wₒ, bₒ, σ))
    return nn(x)
end
```

Flux.jl




```julia
alpha = 0.09              # regularizatin term
sig = sqrt(1.0 / alpha) # variance of the Gaussian prior

@model bayes_nn(xs, ts) = begin
    theta ~ MvNormal(zeros(20), sig .* ones(20))

    preds = nn_forward(xs, theta)
    for i = 1:length(ts)
        ts[i] ~ Bernoulli(preds[i])
    end
end
```

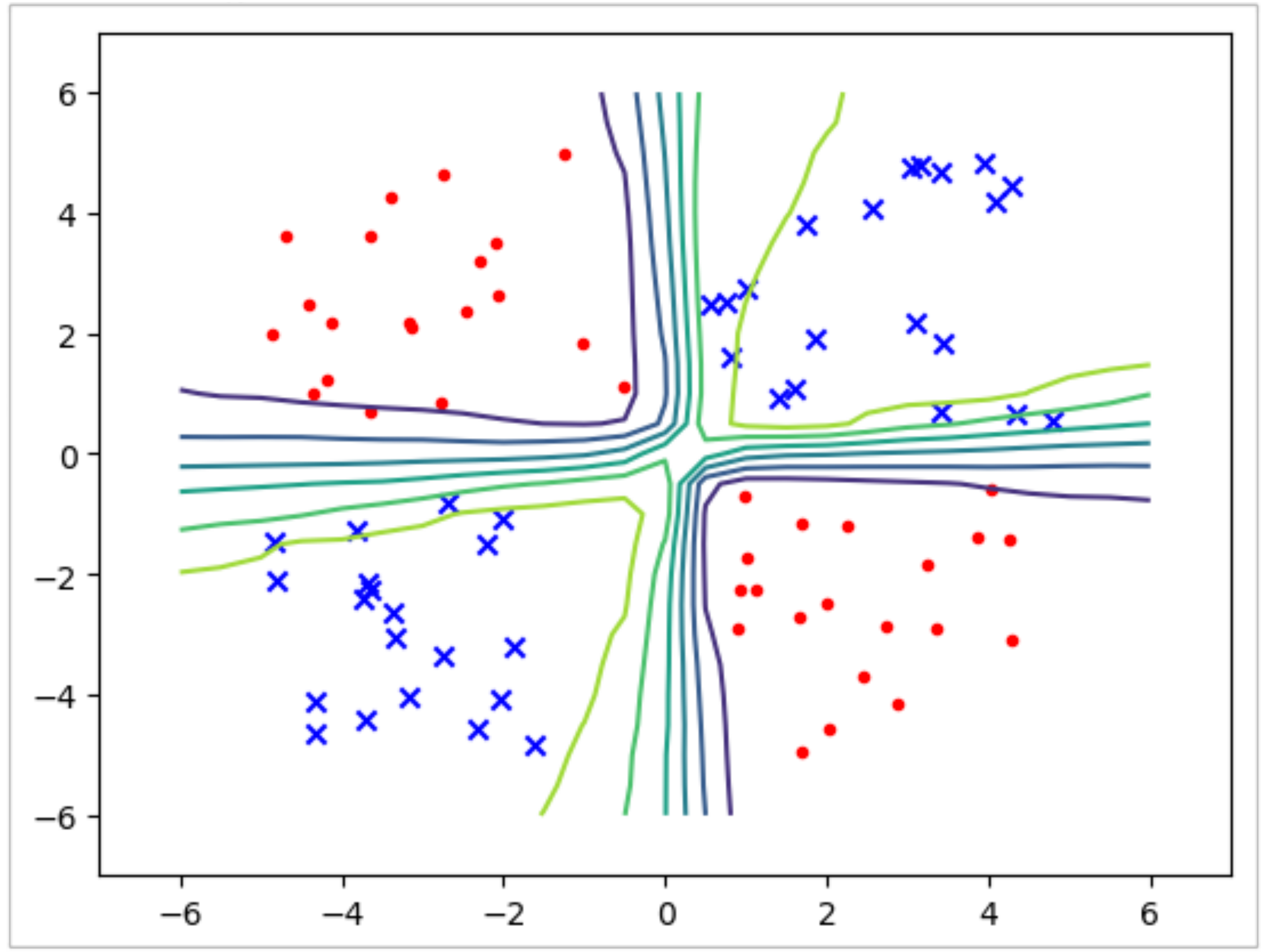
Turing.jl

Bayesian Deep Learning - Inference

```
N = 5000
ch = sample(bayes_nn(hcat(xs...), ts), HMC(N, 0.05, 4))
```

```
[HMC] Sampling... 98%  ETA: 0:00:01
  ϵ:          0.05
  α:          0.9999819814494996

[HMC] Finished with
  Running time        = 60.61580677799989;
  Accept rate         = 0.9206;
  #lf / sample        = 3.9992;
  #evals / sample     = 5.999;
  pre-cond. diag mat  = [1.0, 1.0, 1.0, 1.0, 1.0

[HMC] Sampling...100% Time: 0:01:01
```

Replace 'Inference results' with 'Inference'



Takeaways

* Instead of next steps: Next milestones
* tighter integration instead of integration
* Maybe mention that we will support

- A powerful probabilistic programming language
  - *Intuitive* modelling syntax
  - Support both black-box and *compositional* inference
  - Pure *Julia* code, fully hackable
- Next milestones:
  - Compositional modelling
  - Tighter integration with deep learning packages
  - Scaling up to bigger problems

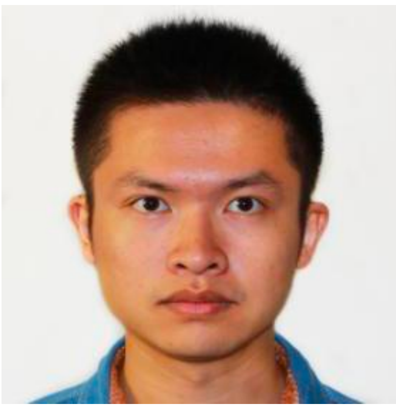
Bibliography

1. Ge, Hong, Kai Xu, and Zoubin Ghahramani. "Turing: Composable inference for probabilistic programming." In *International Conference on Artificial Intelligence and Statistics*, pp. 1682-1690. 2018.
2. Carpenter, Bob, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. "Stan: A probabilistic programming language." *Journal of statistical software* 76, no. 1 (2017).
3. Wood, Frank, Jan Willem Meent, and Vikash Mansinghka. "A new approach to probabilistic programming inference." In *Artificial Intelligence and Statistics*, pp. 1024-1032. 2014.
4. Winn, John, and Tom Minka. "Probabilistic programming with Infer .NET." Machine Learning Summer School lecture notes, available at http://research. microsoft. com/~ minka/papers/mlss2009 (2009).
5. Lunn, David J., Andrew Thomas, Nicky Best, and David Spiegelhalter. "WinBUGS-a Bayesian modelling framework: concepts, structure, and extensibility." *Statistics and computing* 10, no. 4 (2000): 325-337.
6. Tran, Dustin, Matthew D. Hoffman, Rif A. Saurous, Eugene Brevdo, Kevin Murphy, and David M. Blei. "Deep Probabilistic Programming." (2016).
7. Mansinghka, Vikash, Daniel Selsam, and Yura Perov. "Venture: a higher-order probabilistic programming platform with programmable inference." *arXiv preprint arXiv:1404.0099*(2014).
8. Hoffman, Matthew D., and Andrew Gelman. "The No-U-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo." *Journal of Machine Learning Research* 15, no. 1 (2014): 1593-1623.

Contributors

web: turing.ml

Please get in touch in you want to contribute!



Kai Xu[3]

Hong Ge[1]

Emile Mathieu[2]

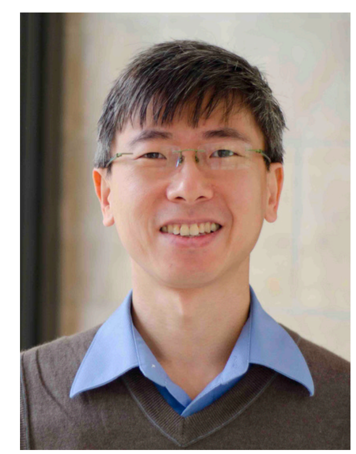
Martin Trapp[4]

many others …

Zoubin Ghahramani[1]

Will Tebbutt[1]

Wessel Bruinsma[1]



Yee Whye Teh[2]

1: Department of Engineering, University of Camb
2: Department of Statistics, University of Oxfor
3. School of Informatics, University of Edinburg
4: Austrian Research Institute for Artificial Intellige

Change my affiliation to:
SPSC Lab, Graz University of Technology

or

Austrian Research Institute for Artificial Intelligence