How to perform correct and efficient Bayesian inference

- Previous methods focussed on either *correct* or *efficient*, not both
    - *correctness*: general purpose probabilistic programming languages
    - *efficiency*: programming from scratch

Blang

- Our effort: Blang, a Bayesian modelling language focusing on supporting correct and efficient combinatorial space sampling
  - Open source
  - Project page: https://www.stat.ubc.ca/~bouchard/blang/

Collaborators: Davor Cubranic, Sahand Hosseini, Matteo Lepur, Kevin Chern, Zihui Ouyang

Declarative model construction

- `model`: a collection of probability distributions (`law`'s) indexed by *parameters* (i.e. a family)

- one way to build a model is to *declare* (conditional) probabilities for each random variable (syntax inspired by WinBUG/JAGS)

```
x | lambda ~ Poisson(lambda)
```

  - motivation: stay close to mathematical notation

```
model SimplePhyloExample {
  random RealVar shape ?: latentReal, rate ?: latentReal
  random SequenceAlignment observations
  random UnrootedTree tree ?: unrootedTree(observations.observedTreeNodes)
  param EvolutionaryModel evoModel ?: kimura(observations.nSites)

  laws {
    shape ~ Exponential(1.0)
    rate ~ Exponential(1.0)
    tree | shape, rate ~ NonClockTreePrior(Gamma::distribution(shape, rate))
    observations | tree, evoModel ~ UnrootedTreeLikelihood(tree, evoModel)
  }
}
```

Model composition

Language	Language used for its standard library
Win / Open BUGS	Pascal
RevBayes	C++
JAGS	C++
Stan	C++
Blang	**Blang**

Model *composition*

- `model`: a collection of distributions indexed by *parameters*
  - can be specified using conditional distributions
  - a conditional distribution can be built from a parameterized collection of distributions (parameter ⇝ conditional)
  - ⇒ can build a `model` using other `model`’s

```
model SimplePhyloExample {
...

  laws {
    shape ~ Exponential(1.0)
    ...
  }
}
```

```
model Exponential {
  random RealVar realization
  param  RealVar rate

  laws {
    realization | rate ~ Gamma(1.0, rate)
  }
}
```

```
model Gamma {
  random RealVar realization
  param  RealVar shape
  param  RealVar rate

  laws {
    logf(shape, rate, realization) {
      if (shape <= 0.0 || rate <= 0)
        return NEGATIVE_INFINITY
      if (realization <= 0.0)
        return NEGATIVE_INFINITY
      ...
    }
    ...
}
```

Composition via parameters

- Pushing this idea a bit further, parameters themselves can be models (distributions)
- Other examples: distribution of branch lengths, Dirichlet process' base measure, etc.

```
model IntMixture {
  param Simplex proportions
  param List<IntDistribution> components
  random IntVar realization

  laws {
    logf(proportions, components, realization) {
      var sum = 0.0
      if (components.size !== proportions.nEntries) throw new RuntimeException
      for (i : 0 ..< components.size) {
        val prop = proportions.get(i)
        if (prop < 0.0 || prop > 1.0) return NEGATIVE_INFINITY
        sum += prop * exp(components.get(i).logDensity(realization))
      }
      return log(sum)
    }
  }

  generate (rand) {
    val category = rand.categorical(proportions.vectorToArray)
    return components.get(category).sample(rand)
  }
}
```

```
...
    observation | proportions, lambda, rho
      ~ IntMixture(
        proportions,
        #[
          Poisson::distribution(lambda),
          YuleSimon::distribution(rho)
        ]
      )
...
```

Usage example

Example: a mixture of discrete distributions, taking a list of distributions as parameter: the mixture components

- Each Blang model is turned into an inference program
  (currently command line, but more interfaces under development; currently working on the r interface)

- This program takes as input observed data and outputs posterior samples for the unobserved variables
  Concretely, inputs are currently command line arguments for each variable (organized hierarchically, most with sensible default values. try `--help`).
  Outputs are *tidy* csv files (Wickham, 2013).

Try at home!
Needed: Oracle Java 8, git, POSIX

```
> git clone https://github.com/UBC-Stat-ML/blangExample.git
[cloning]

> ./gradlew installDist
[downloading dependencies and compiling]

> ./build/install/example/bin/example \
  --model.observations.file data/primates.fasta \
  --model.observations.encoding DNA \
  --engine SCM \
  --engine.nThreads Max \
  --excludeFromOutput observations

Preprocessing started
4 samplers constructed with following prototypes:
RealScalar sampled via: [RealSliceSampler]
UnrootedTree sampled via: [SingleNNI, SingleBranchScaling]
Sampling started
[sampling progress report]
Normalization constant estimate: -1216.1211229417504
Final rejuvenation started
Preprocessing time: 141.4 ms
Sampling time: 2.304 min
executionMilliseconds : 138405
```

Open type system

- Blang allows object-oriented development of custom random variable datatypes (combinatorial objects)

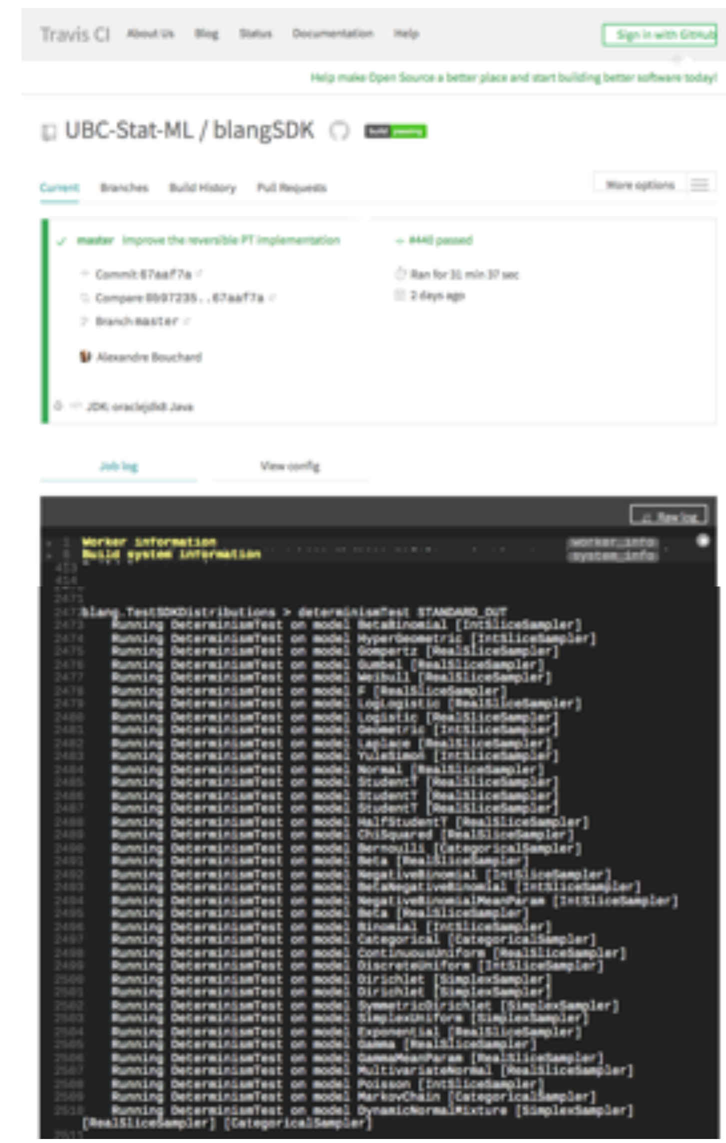
```
/**
 * An unrooted phylogenetic tree.
 * @author Alexandre Bouchard (alexandre.bouchard@gmail.com)
 */
@Samplers(SingleNNI, SingleBranchScaling)
class UnrootedTree {
  val Map<UnorderedPair<TreeNode, TreeNode>, Double> branchLengths
  val UndirectedGraph<TreeNode, UnorderedPair<TreeNode, TreeNode>> topology
  ...
}
```

```
class SingleBranchScaling extends MHSampler {
  @SampledVariable UnrootedTree variable

  override propose(Random rand, Callback callback) {
    val allEdges = newArrayList(variable.topology.edgeSet)
    val edge = sample(allEdges, rand)
    val oldValue = variable.getBranchLength(edge)
    val m = exp(2.0 * log(2.0) * (rand.nextDouble - 0.5))
    callback.setProposalLogRatio(log(m))
    variable.updateBranchLength(edge, m * oldValue)
    if (!callback.sampleAcceptance)
      variable.updateBranchLength(edge, oldValue)
  }
}
```

- For scalability, user may need to write a sampler, but..
  - user can write using the same syntax
  - Bar is lower in terms of efficiency : thanks to advanced posterior simulation methods (beyond MCMC)
  - Blang helps you checking *correctness* of implementation

Correctness

- How to check MCMC code is invariant and/or irreducible?
- Seems hard; surprisingly very good tests can be constructed
- Example: for discrete `model`'s, Blang's `DiscreteMCTest` utility will:
    - enumerate all the execution traces for each associated samplers
    - compute the posterior π by enumeration
    - build an explicit transition matrix *M* for each sample
    - check that $\pi = M\pi$, as well as irreducibility
- Many more tests, existing and novel, including tests for continuous models (e.g., Geweke, 2004)



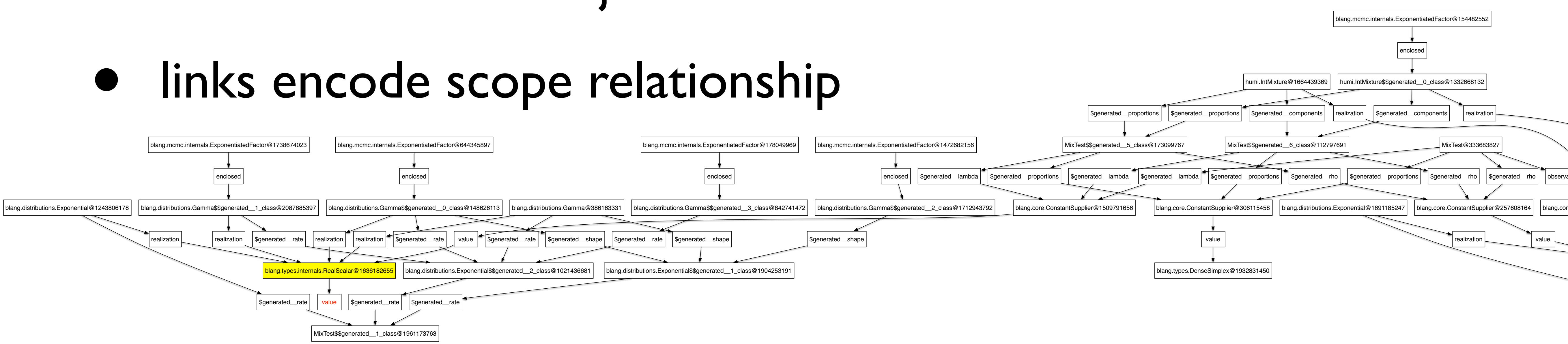
JUnit integration

Posterior inference

- Blang uses the declarative syntax to automatically build a *sequence* of distributions instead of just the posterior distribution
- Key to the advanced *inference engines* available
  - *Non-reversible Parallel Tempering*
  - *Sequential Change of Measure*

Exploiting sparsity

- Samplers can be much faster if they avoid recomputing all model components
    - these savings are formalized by sparse *factor graphs* (Clifford, 1990)
- Idea: building factor graphs using *scoping information*
- Details: get factor graph by running linear time graph algorithms on an *accessibility graph*:
    - in which vertices objects
    - links encode scope relationship



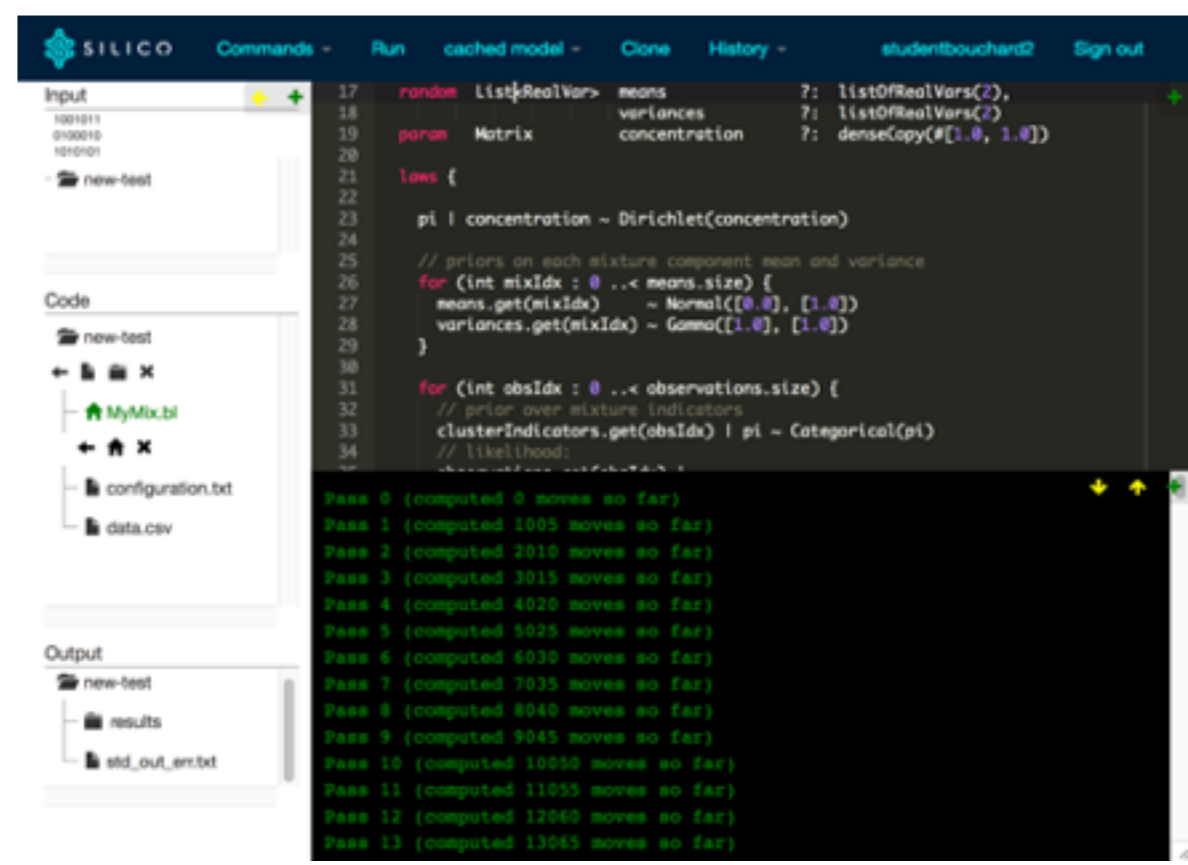
Language details

- Built using Xtext, (framework for designing programming languages)
- Allowed us to "quickly" (3yrs) build an extensive multi-paradigm language
    - in addition to declarative capability, supports functional, generic and object programming, static typing, just-in-time compilation, garbage collection, etc
    - advanced IDE support
- Runs on JVM, interoperates with Java
    - fast (just-in-time compiled)
    - can exploit arbitrarily many cores

```
grammar ca.ubc.stat.blang.BlangDsl with
org.eclipse.xtext.xbase.annotations.XbaseWithAnnotations

generate blangDsl "http://www.ubc.ca/stat/blang/BlangDsl"

import "http://www.eclipse.org/xtext/common/JavaVMTypes" as jvmTypes
import "http://www.eclipse.org/Xtext/Xbase/XAnnotations" as xAnnotations

BlangModel:
      {BlangModel}
      ('package' package=QualifiedName)?
      importSection=XImportSection?
      (annotations += XAnnotation)*
      'model' name=ID '{'
    (variableDeclarations += VariableDeclaration)*
    'laws' '{' (lawNodes += LawNode)* '}'
    ('generate' '(' generationRandom = ValidID ')' generationAlgorithm =
XBlockExpression )?
    (variableDeclarations += VariableDeclaration)*
      '}'
;

...
```

Workflows

- Blang IDE:
  - leverages static types (eg smart links, call/type hierarchies, refactoring)
  - debugging, profiling, code coverage analysis, etc
- Web IDE
- Command line



Reproducibility & dissemination

- Code fully deterministic
  - even in multithread mode
  - checked via test units
- Create versioned packages containing models/samplers that others can import
  - built on Maven
  - smart dependency resolution

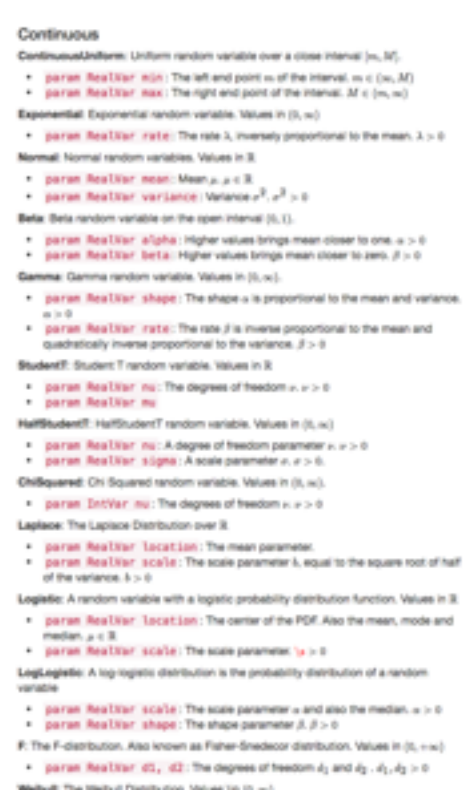
dependencies.txt

```
ca.ubc.stat:nowellpack:1.0.5
ca.ubc.stat:conifer:2.0.4
```

```
+--- ca.ubc.stat.blang:ca.ubc.stat.blang:3.25.2
|     +--- org.eclipse.jdt:org.eclipse.jdt.core:3.10.0
|     |     +--- org.eclipse.core:org.eclipse.core.resources:3.7.100
|     |     |     +--- org.eclipse.core:org.eclipse.core.expressions:3.4.300
|     |     |     |     \--- org.eclipse.core:org.eclipse.core.runtime:3.7.0
|     |     |     |           +--- org.eclipse.osgi:org.eclipse.osgi:3.7.1
|     |     |     |           +--- org.eclipse.equinox:org.eclipse.equinox.common:3.6.0
|     |     |     |           +--- org.eclipse.core:org.eclipse.core.jobs:3.5.100
|     |     |     |           |     \--- org.eclipse.equinox:org.eclipse.equinox.common:3.6.0
|     |     |     |           +--- org.eclipse.equinox:org.eclipse.equinox.registry:3.5.101
|     |     |     |           |     \--- org.eclipse.equinox:org.eclipse.equinox.common:3.6.0
|     |     |     |           +--- org.eclipse.equinox:org.eclipse.equinox.preferences:3.4.1
|     |     |     |           |     \--- org.eclipse.equinox:org.eclipse.equinox.common:3.6.0
|     |     |     |           +--- org.eclipse.core:org.eclipse.core.contenttype:3.4.100
|     |     |     |           |     +--- org.eclipse.equinox:org.eclipse.equinox.preferences:3.4.1 (*)
|     |     |     |           |     +--- org.eclipse.equinox:org.eclipse.equinox.registry:3.5.101 (*)
|     |     |     |           |     \--- org.eclipse.equinox:org.eclipse.equinox.common:3.6.0
|     |     |     |           \--- org.eclipse.equinox:org.eclipse.equinox.app:1.3.100
|     |     |     +--- org.eclipse.core:org.eclipse.core.filesystem:1.3.100
|     |     |     |     +--- org.eclipse.equinox:org.eclipse.equinox.common:3.6.0
|     |     |     |     +--- org.eclipse.equinox:org.eclipse.equinox.registry:3.5.101 (*)
|     |     |     |     \--- org.eclipse.osgi:org.eclipse.osgi:3.7.1
|     |     |     \--- org.eclipse.core:org.eclipse.core.runtime:3.7.0 (*)
|     |     +--- org.eclipse.core:org.eclipse.core.runtime:3.7.0 (*)
|     |     +--- org.eclipse.core:org.eclipse.core.filesystem:1.3.100 (*)
|     |     \--- org.eclipse.text:org.eclipse.text:3.5.101
|     |           +--- org.eclipse.core:org.eclipse.core.commands:3.6.0
|     |           \--- org.eclipse.equinox:org.eclipse.equinox.common:3.6.0
|     +--- org.eclipse.xtext:org.eclipse.xtext:2.12.0
|     |     +--- org.eclipse.xtext:org.eclipse.xtext.util:2.12.0
|     |     |     +--- org.eclipse.xtend:org.eclipse.xtend.lib:2.12.0
|     |     |     |     +--- org.eclipse.xtext:org.eclipse.xtext.xbase.lib:2.12.0
|     |     |     |     |     \--- com.google.guava:guava:[14.0,19) -> 19.0
|     |     |     |     \--- org.eclipse.xtend:org.eclipse.xtend.lib.macro:2.12.0
|     |     |     |           \--- org.eclipse.xtext:org.eclipse.xtext.xbase.lib:2.12.0 (*)
|     |     |     +--- log4j:log4j:1.2.16
|     |     |     +--- org.eclipse.emf:org.eclipse.emf.common:[2.10.1,3) -> 2.15.0
|     |     |     +--- org.eclipse.emf:org.eclipse.emf.ecore:[2.10.2,3) -> 2.15.0
|     |     |     |     \--- org.eclipse.emf:org.eclipse.emf.common:[2.15.0,3.0.0) -> 2.15.0
|     |     |     \--- com.google.inject:guice:3.0
|     |     |           +--- javax.inject:javax.inject:1
|     |     |           \--- aopalliance:aopalliance:1.0
```

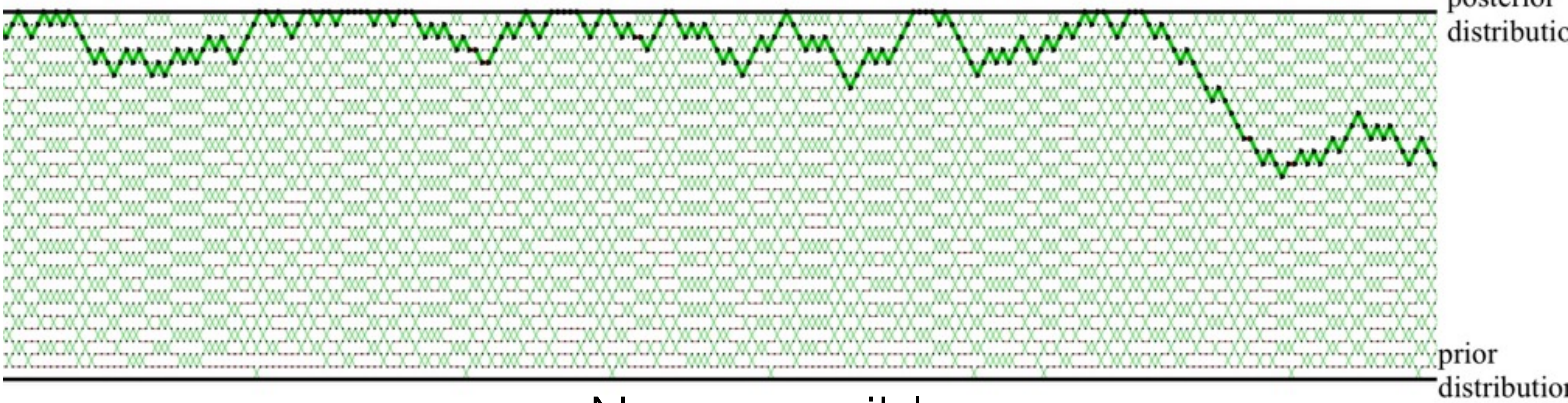
Documentation

- Available at https://www.stat.ubc.ca/~bouchard/blang/
  - Getting started & a few examples
  - Setting up workflows
  - Complete syntax
  - Reference for SDK distributions, engines, tests, io, etc

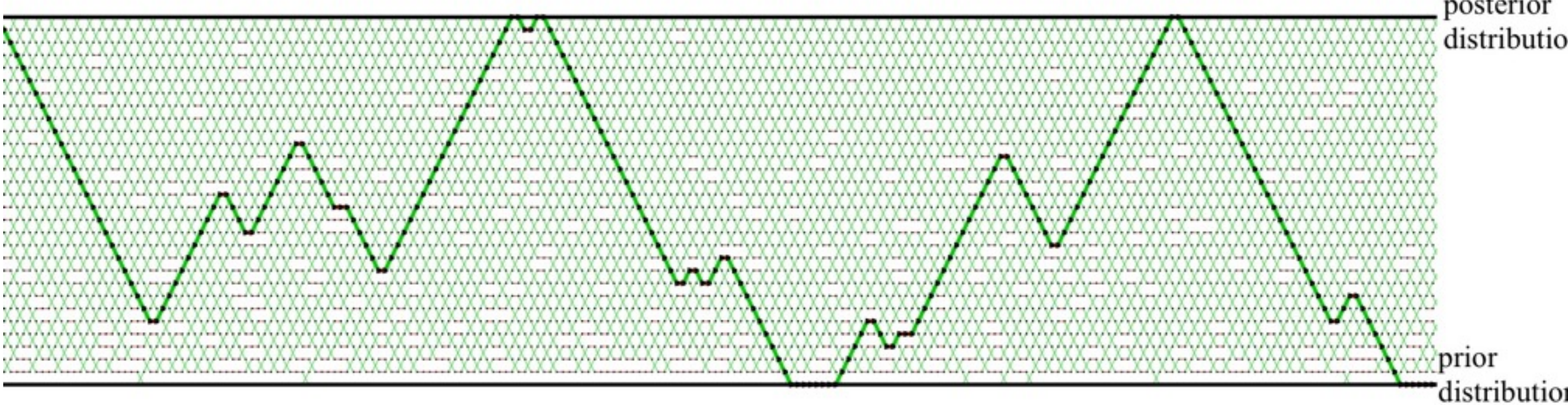


Visualization: paths

Reversible



Non-reversible



Visualization: adaptation