High Performance Computing for Science and Engineering II

25.3.2019 - **Lecture 6: The Korali Framework**

Lecturer: Dr. Sergio Martin

CSElab

Computational Science & Engineering Laboratory

ETH zürich

Schedule (Red = UQ, Blue = HPC)

18.02 (SM) Introduction
25.02 (PK) Bayesian UQ I - Likelihood & Priors
04.03 (PK) Bayesian UQ II - Laplacian Methods, Variational Inference
11.03 (PK) Model Selection / Forward Propagation of Uncertainty
18.03 (GA) Markov Chain Monte Carlo
25.03 (SM) High Throughput Computation for UQ & Optimization: Korali
01.04 (SM) Partitioned Global Address Space programming with UPC++
08.04 << No Class - Holiday: Sechseläuten >>
15.04 (SM) Many-Core Processor Programming (CUDA) I (Basic Concepts)
22.04 << No Class - Easter Week >>
29.04 (SM) Many-Core Processor Programming (CUDA) II (Optimizations)
06.05 (SM) Advanced MPI Concepts, MPI + OpenMP Hybridization
13.05 (SM) MPI+CUDA, Heterogeneous Computing
20.05 (SM) Communication-Tolerant Programming + Pre-Exam Review
27.05 --- Exam

Today's Class

Korali:

- Purpose and Design
- Modules
- Usage Example: The Ackley Function

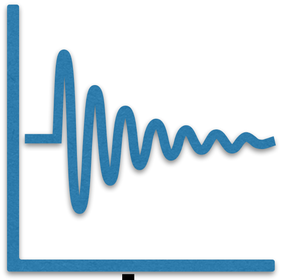
Homework 3 (Part II):

- How to submit your code and write your report.
- Initial Setup.
- Understanding the tasks

Korali

Uncertainty Quantification Engine

How do we optimize/quantify uncertainty?

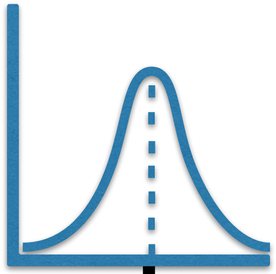


Better understanding of the system and models uncertainties.

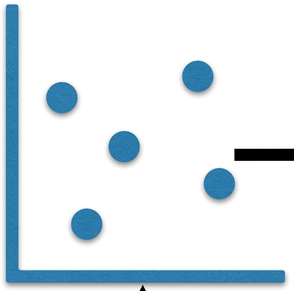
UQ/Optimization Methods
CMA-ES
Markov-Chain Monte-Carlo
TMCMC

Improved

Parameter Guess



Computed Results



Experimental Measurements

Compare

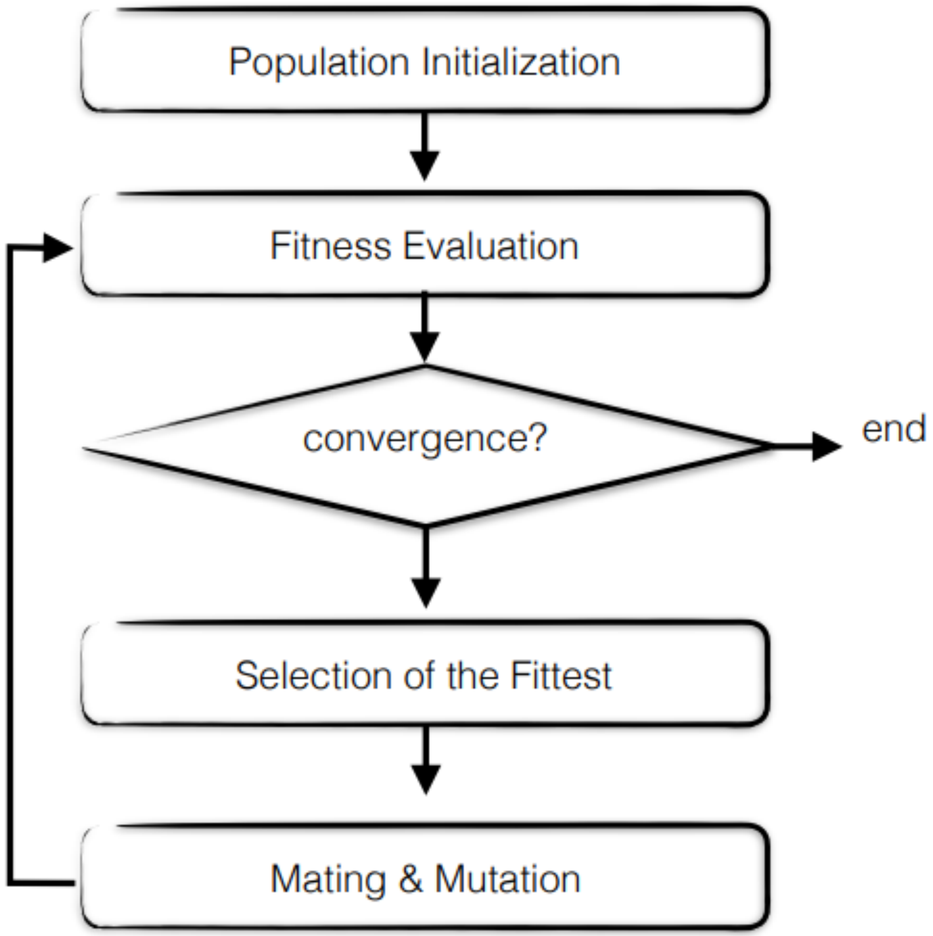
Discretize / Approximate

Mathematical Model

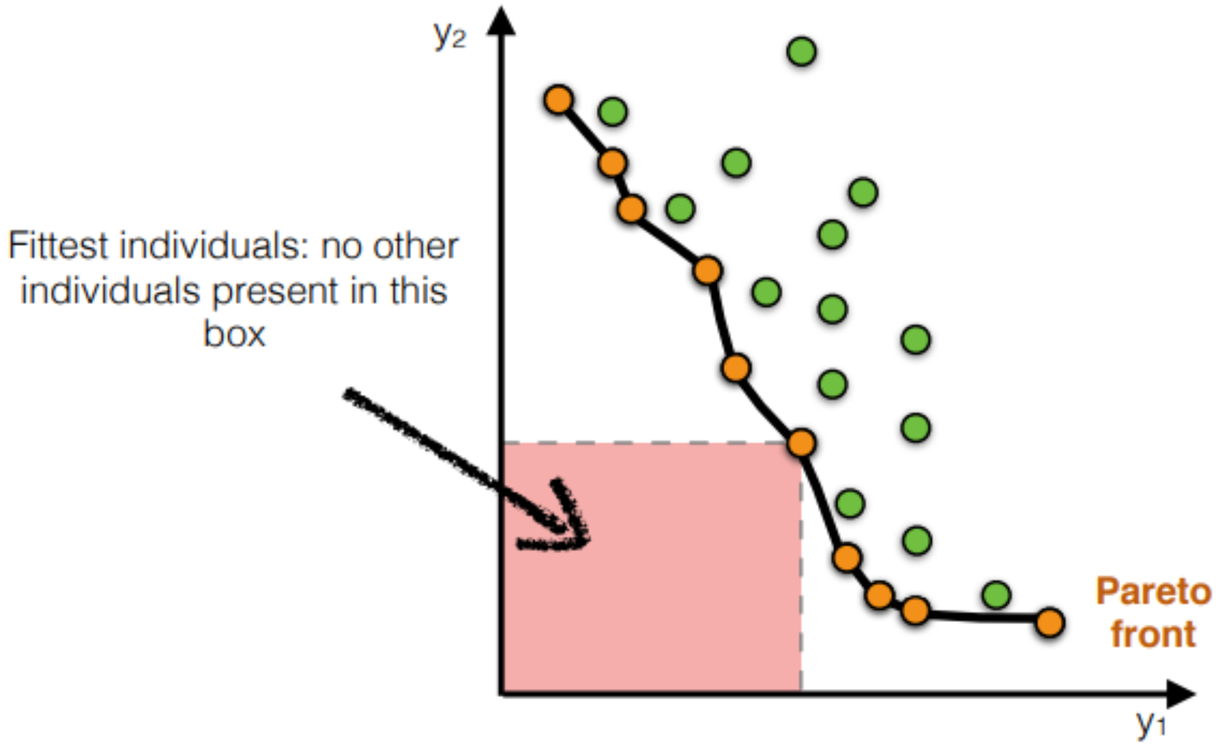
Computational Model

Physical System

Evolutionary Strategy



Multi-Dimensional Selection



- Maintain a diverse approximation of the Pareto front
  - NSGA-II algorithm (Deb *et al.*, 2002)
- Scheduling of evaluations to use comp. resources efficiently
  - TORC library (Hadjidoukas *et al.*, 2012)

UQ / Optimization Cycle

Fact I: Uncertainty Quantification and Optimization methods require many model evaluations.

Fact II: For problems with many parameters, the number of model evaluations can be vast.

Fact III: It can be impractical (or impossible) to drive the evaluation cycle manually.

We (scientists and engineers) need tools to automatically:

- Determine a set of samples to evaluate
- Evaluate the set of samples using a computational model.
- Accumulate results and use them as inputs to a UQ or Optimization Method.
- Detect finalization conditions
  (e.g., Global maximum found, Parameter distribution well-defined.)

One Generation

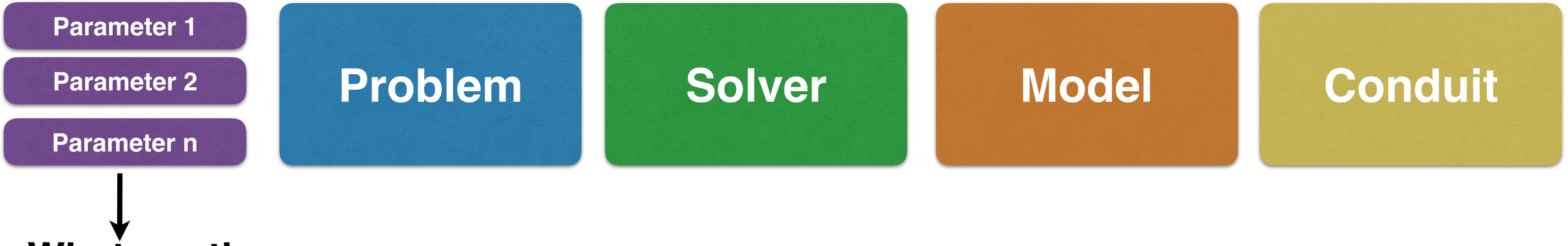
Today, we will learn to use:

Korali

A C++ Modular Framework for Optimization
and Bayesian Uncertainty Quantification

The Korali Framework

A Korali application is built as a combination of 5 Modules



What are the parameters (x)?

How many of them?

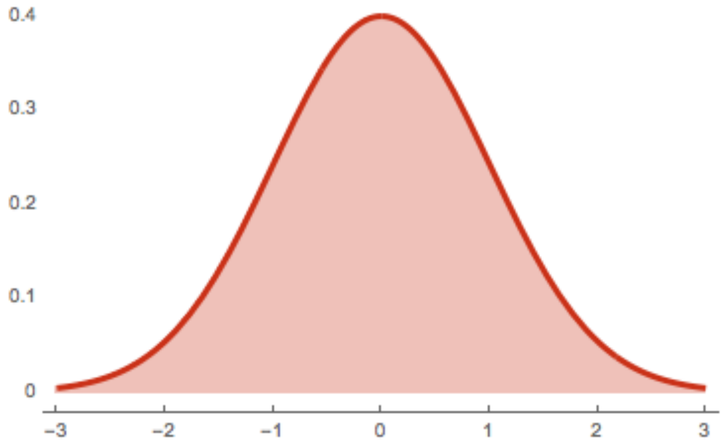
What is their Prior Distribution

What are their bounds?

Module: Parameters

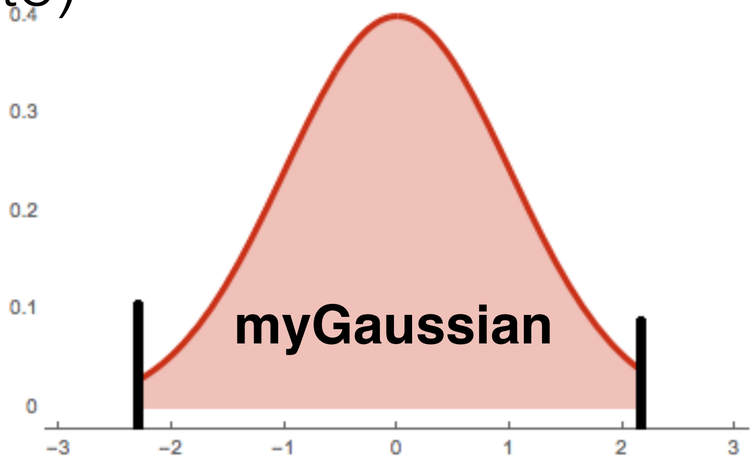
Korali Parameters serve three purposes:

- Determine the number of parameters.
- Determine the bounds of the distribution space.
- Indicate any Prior knowledge about their distribution.



Korali::Parameter::Uniform par(double min, double max)
Korali::Parameter::Gaussian par(double mean, double sigma)
Korali::Parameter::Gamma par(double shape, double rate)
Korali::Parameter::Exponential par(double lambda)

par.**setBounds**(double lowerBound, double upperBound)
par.**setName**("myGaussian")



The Korali Framework

Problem Modules

What are we measuring?

Direct Evaluation of the Model F(x)

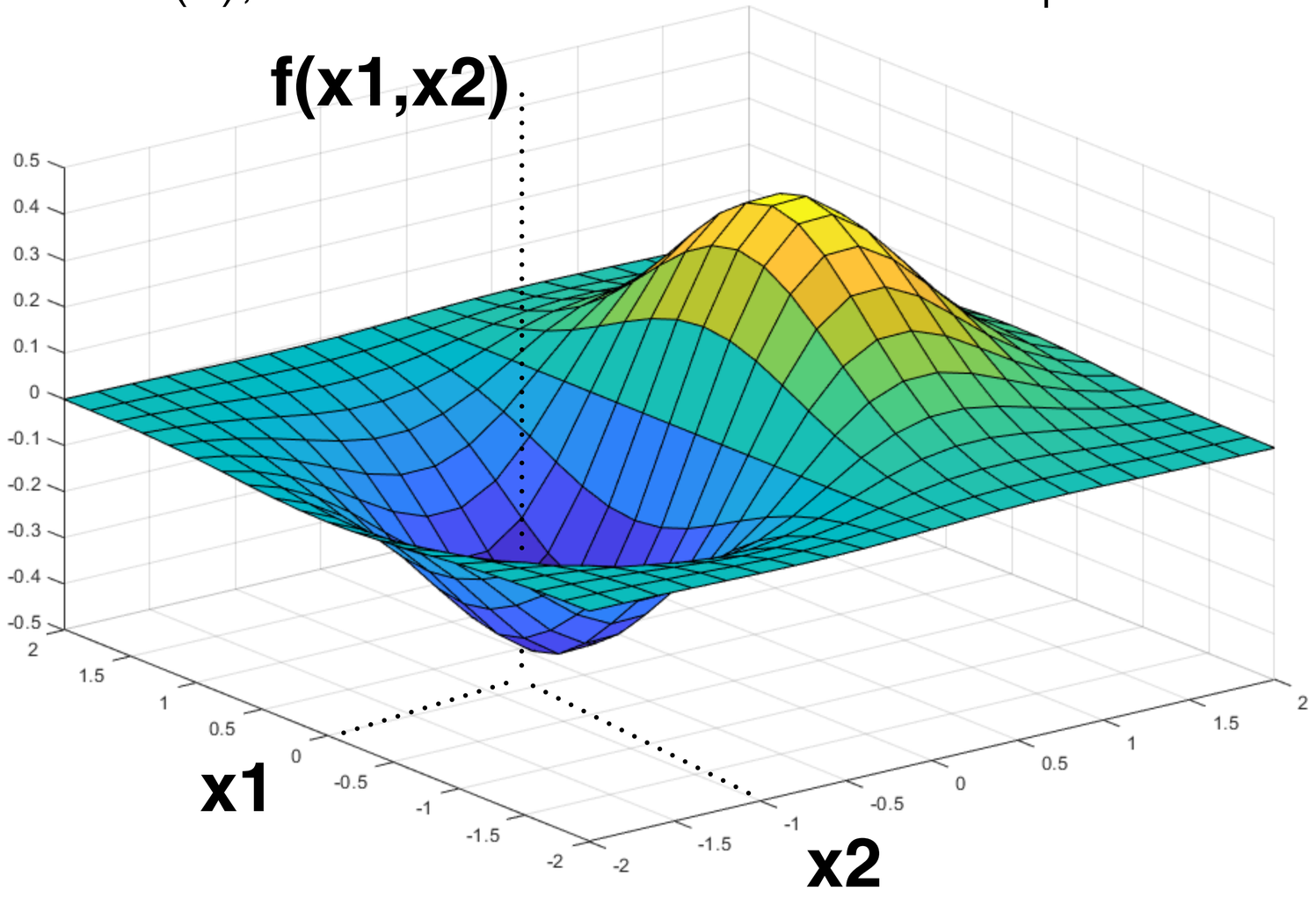
Likelihood Distribution $p_{Y|X}(y|x)$

Posterior Distribution $p_{X|Y}(x|y)$

Module: Direct Problem

Korali::Problem::Direct p(f(x))

Evaluates the value of f(x), where x are the user-defined parameters.



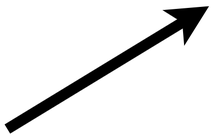
Module: Likelihood Problem

Korali::Problem::Likelihood likelihood(...)

Evaluates the likelihood of the parameters given a set of reference data points.

Uses a **Gaussian** estimator for the likelihood (other distributions will come later)

$$P(data|x) \propto \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{1}{2\sigma^2}\sum_{i=0}^{n}(f(x_i) - data_i)^2\right)$$



Sigma is a **parameter** automatically added by Korali. $x^* = \begin{pmatrix} x \\ \sigma \end{pmatrix}$

To set reference data, execute: p.**setReferenceData**(size_t nPoints, double* data)

The Sigma (σ) Parameter

Represents the estimation error (Uncertainty).

What contributes to σ?

- Structural Uncertainty (are we using the correct model?)
- Computational Uncertainty (fine vs coarse resolution) **HW5!** 😱
- Measurement Uncertainty (Quality and Quantity of experimental data)
- Parametric Uncertainty (How well the parameters fit the data)

Module: Posterior Problem

Korali::Problem::Posterior posterior(...)

Evaluates the posterior distribution given data points and prior information.

Uses a Gaussian estimation for the likelihood times prior probability densities.

$$P(x|data) \propto \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{1}{2\sigma^2}\sum_{i=0}^{n}(f(x_i) - data_i)^2\right)\prod_{i=0}^{n} P(x_i)$$

Same syntax and usage as Likelihood Problem but requires Prior knowledge.

The Korali Framework

Solver Modules

Optimization/Sampling Method

Optimization: **CMA-ES**, Conjugate Gradient, etc...

Sampling: Markov Chain Monte Carlo, **TMCMC**, mTMCMC, ...

Reinforcement Learning: (coming)

Module: CMA-ES Solver

Korali's for Stochastic optimization engine for non-linear & non-convex problems
Finds the values of parameters that maximize the value of Problem *p*.

Korali::Solver::CMAES solver(Korali::Problem p)

Usage:

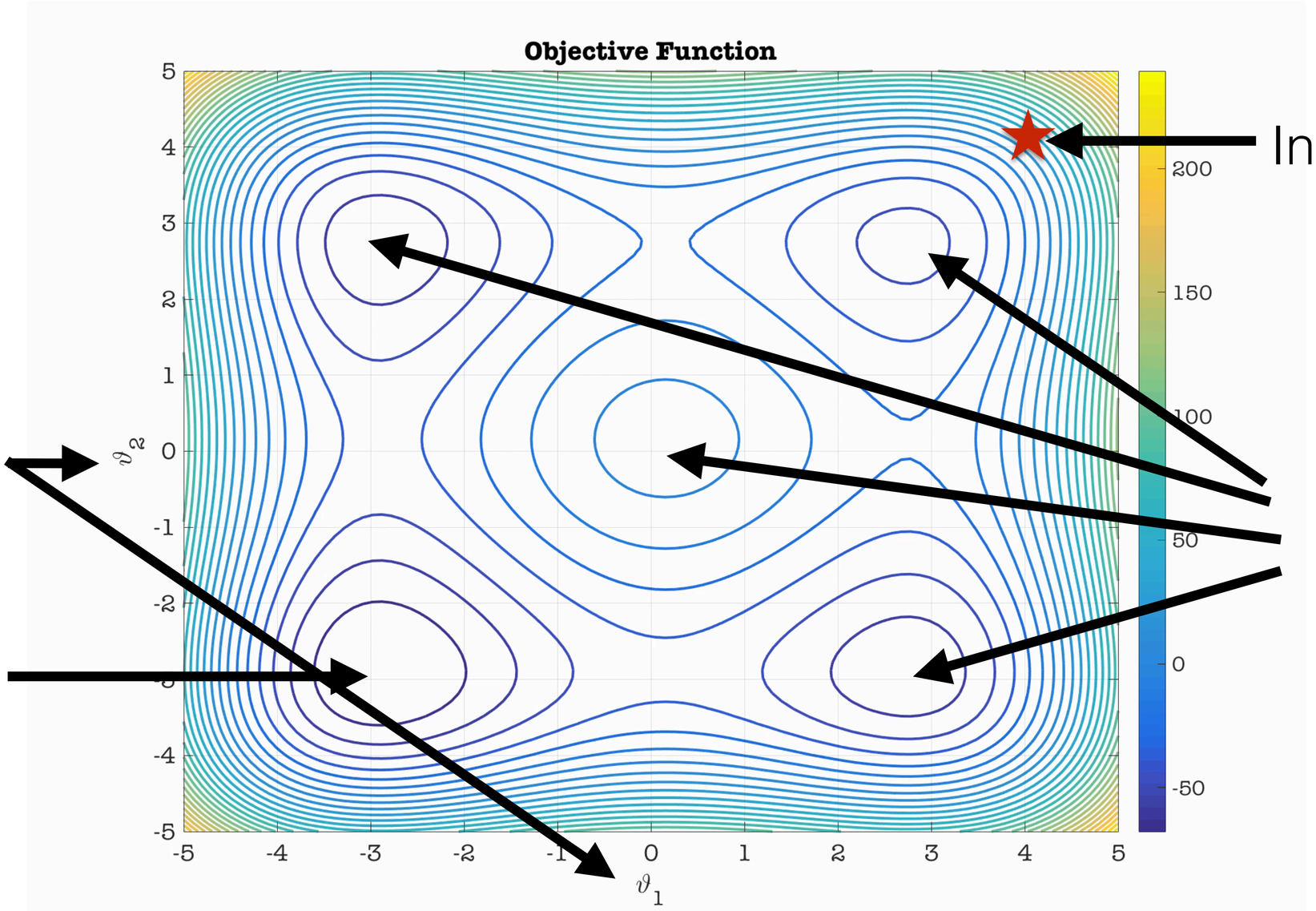
solver.**setMaxGenerations**(int maxGens); - Default 200
  Sets maximum number of generations.

solver.**setPopulationSize**(int nSamples); - Default 1000
  Sets maximum number of samples per generation.

solver.**run**();
  Runs Korali CMA-ES.

Module: CMA-ES Solver

Optimization. Minimizing a two-parameter function.



Module: CMA-ES Solver

```
[Korali] Gen 166 - Elapsed Time: 0.293846, Fitness Change: 9.56e-10
[Korali] Gen 167 - Elapsed Time: 0.294317, Fitness Change: 1.59e-10
[Korali] Gen 168 - Elapsed Time: 0.293199, Fitness Change: 1.59e-10
[Korali] Gen 169 - Elapsed Time: 0.293833, Fitness Change: 1.59e-10
[Korali] Gen 170 - Elapsed Time: 0.294094, Fitness Change: 5.73e-11
[Korali] Finished - Reason: Object variable changes < 1.00e-07
[Korali] Parameter 'Sigma' Value: 0.010227
[Korali] Parameter 'Intensity' Value: 36.742792
[Korali] Parameter 'PosX' Value: 0.305979
[Korali] Parameter 'PosY' Value: 0.802538
[Korali] Total Elapsed Time: 49.934612s
```

Module: TMCMC Solver

Korali's Engine for the Transitional Markov-Chain Monte-Carlo Method
Gives a detailed probability distribution around the fittest parameter values.

Korali::Solver::TMCMC solver(Korali::Problem p)

Usage:

solver.**setMaxGenerations**(int maxGens); - Default 200

solver.**setPopulationSize**(int nSamples); - Default 1000

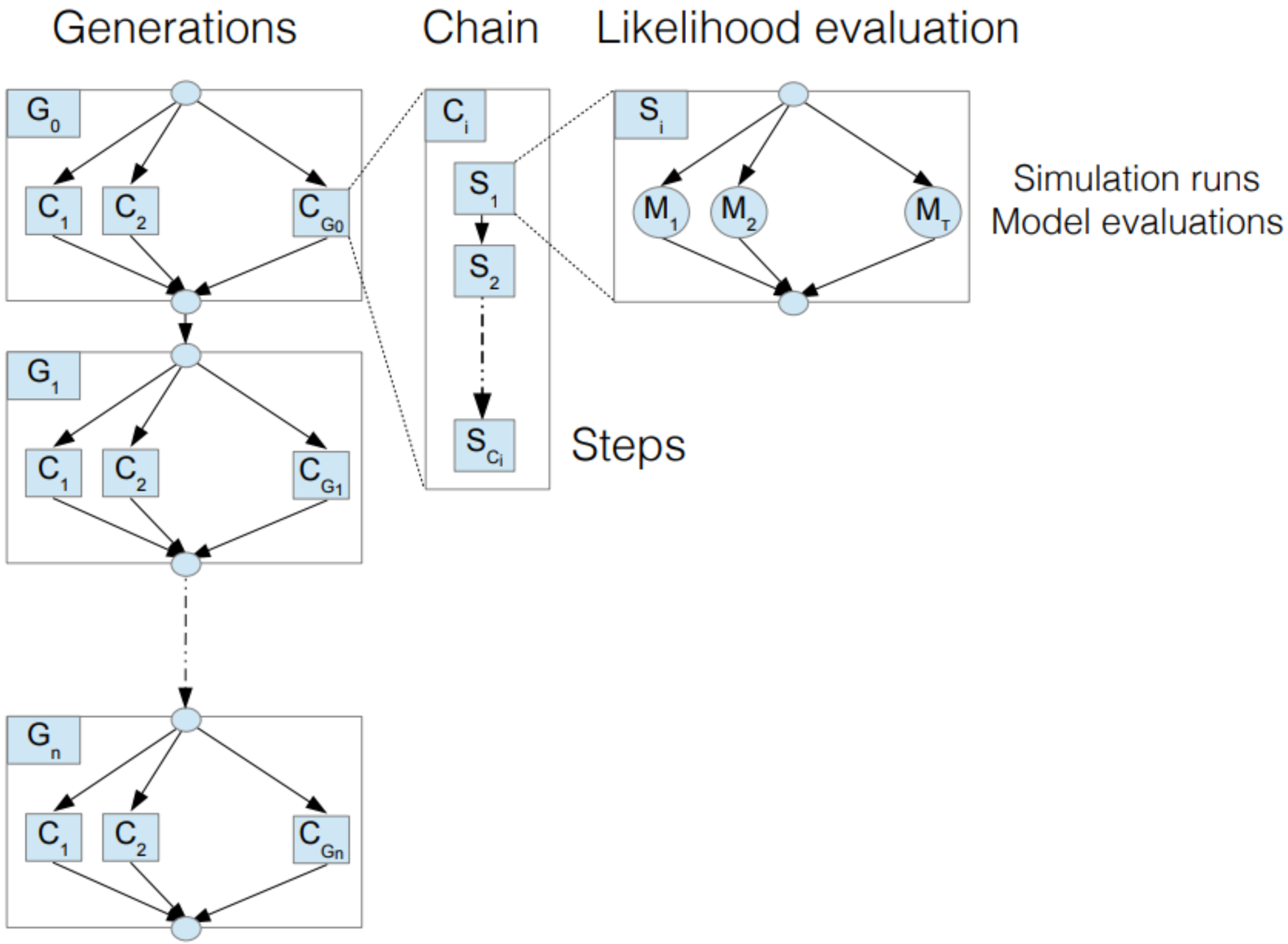
solver.**setCovarianceScaling**(double beta); - Default: 0.005

Affects the sensitivity of re-sampling, smaller for small magnitude parameters.

solver.**run**();

Runs Korali TMCMC. --> Saves results to the tmcmc.txt file

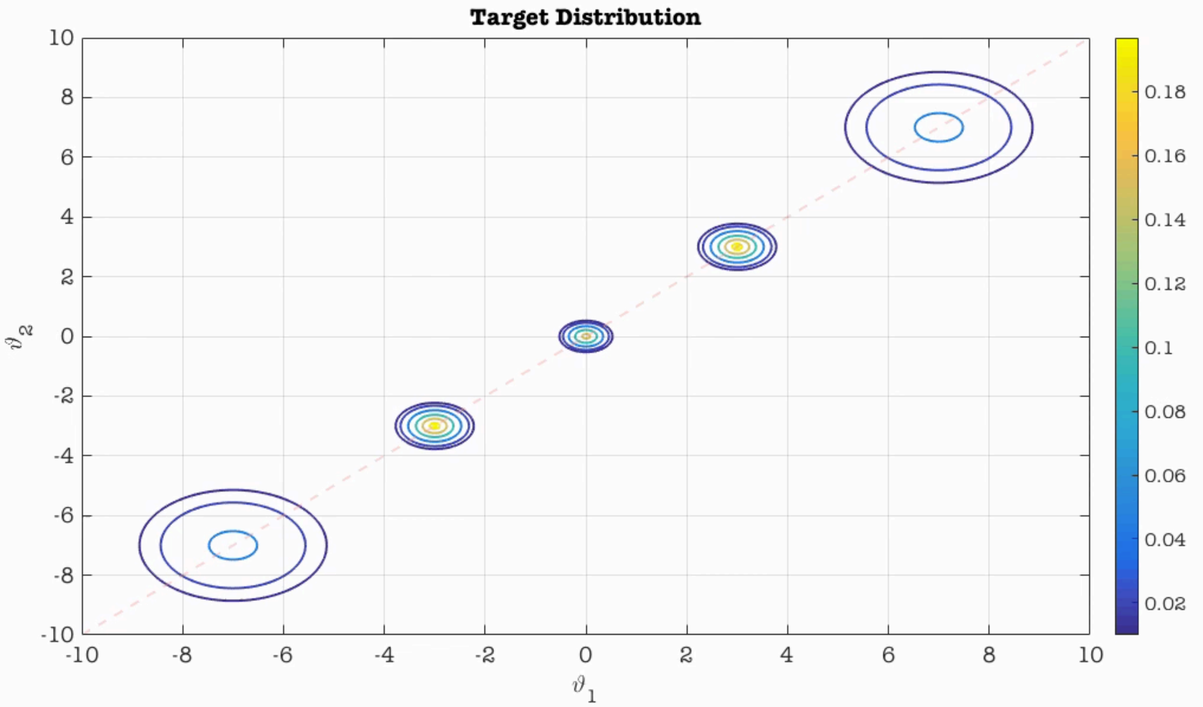
TMCMC Method



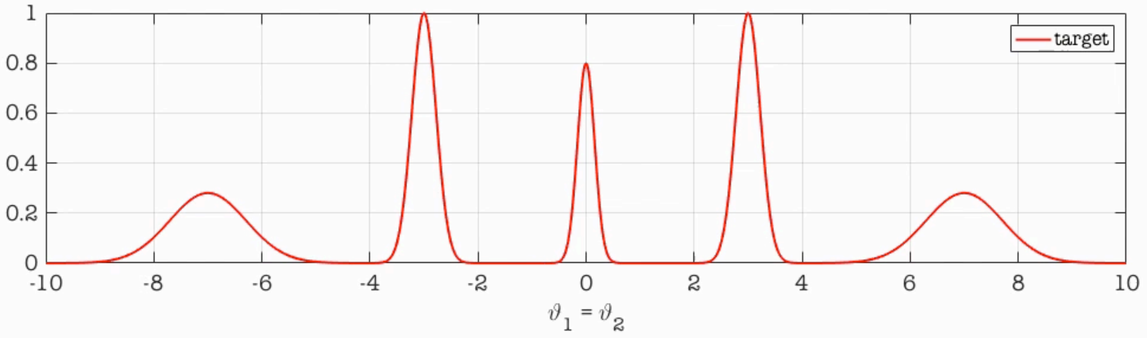
Module: TMCMC Solver

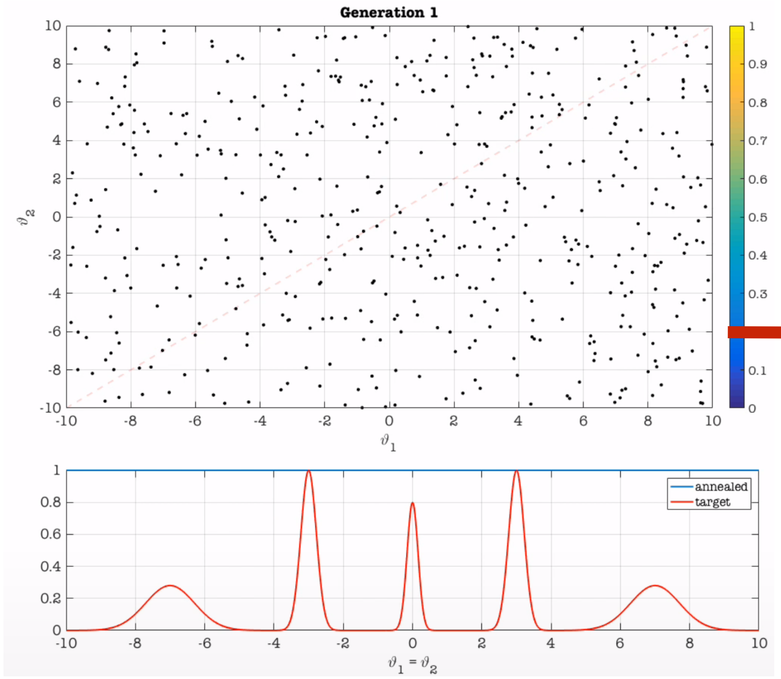
Example: Korali finding the distribution of parameters *v1* and *v2*.

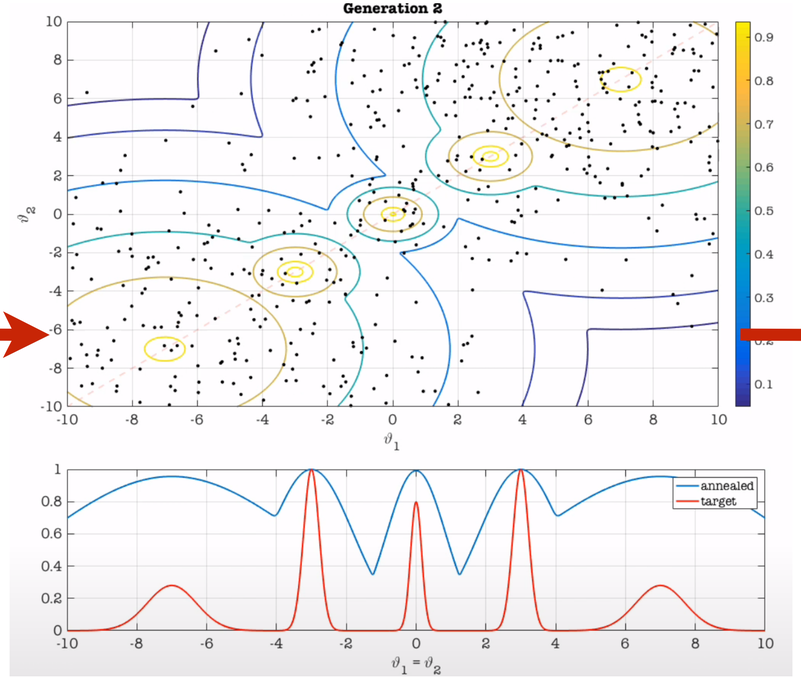
Parameter Space →

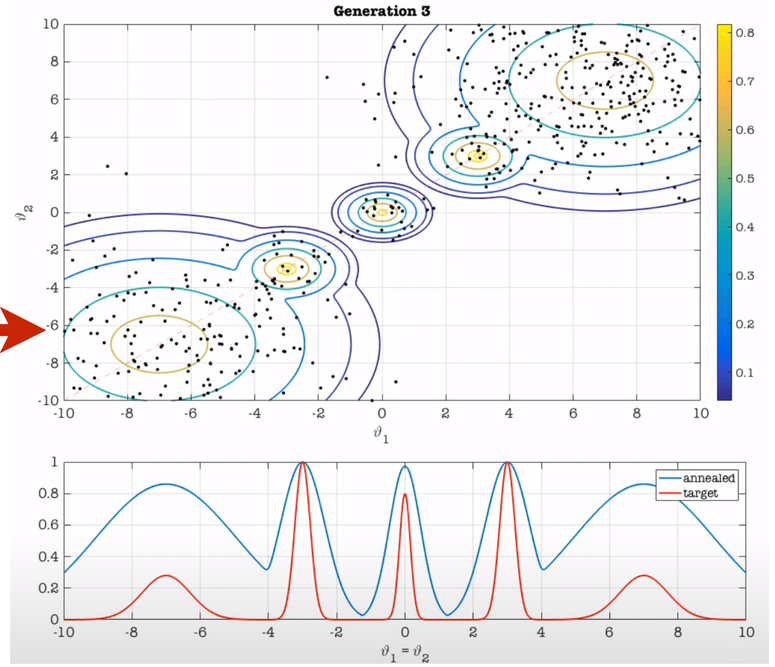


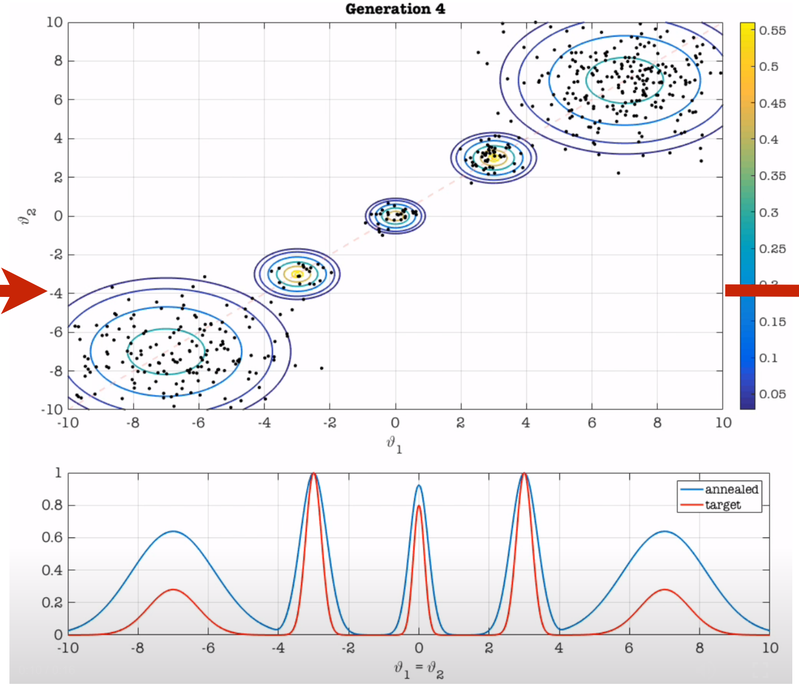
Likelihood *v1=v2*
i.e. Parameters Reflect Experimental Observations →

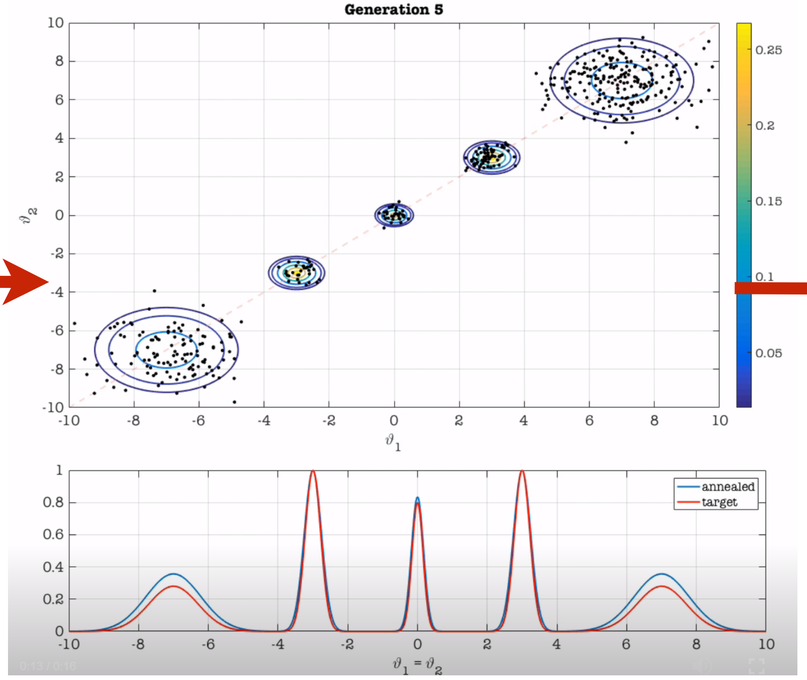


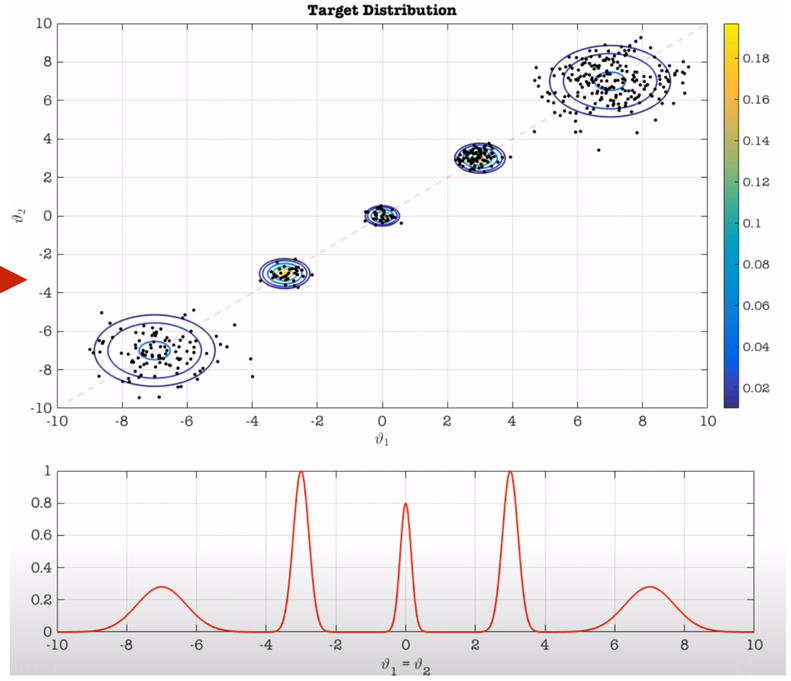

Generation 1
annealed
target


Generation 2
annealed
target


Generation 3
annealed
target


Generation 4
annealed
target

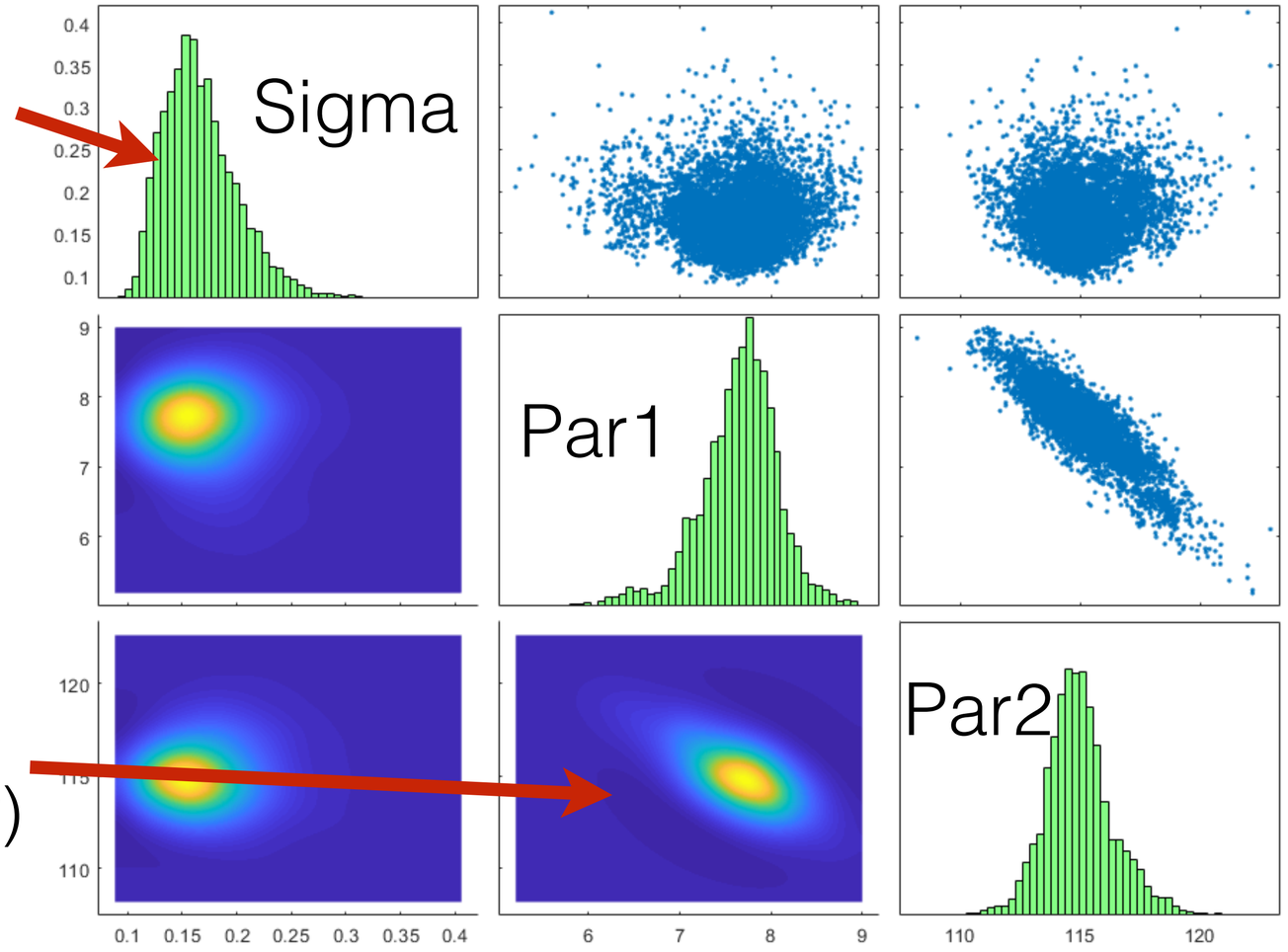

Generation 5
annealed
target


Target Distribution

Module: TMCMC Solver

Example Result: Likelihood Problem

Parameter Histograms



Scatter Plots of Pairwise Samples

Pair Marginal Histograms (Par1 vs. Par2)

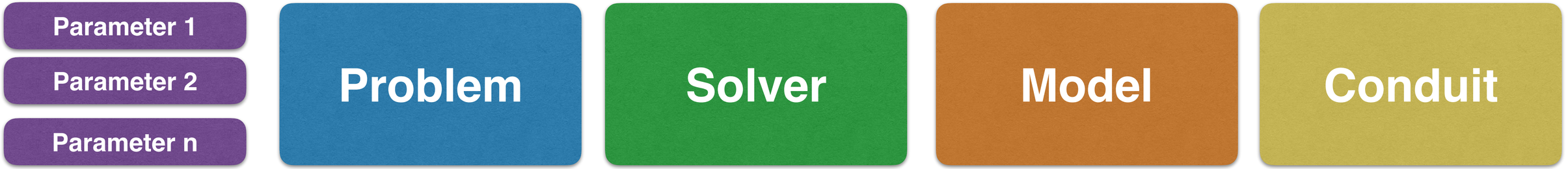
Module: CMA-ES Solver

```
[Korali] Starting TMCMC. Parameters: 3, Seed: 0x5C988932
[Korali] Generation 0 - Time: 0.012616s, Annealing: 0.00%, Acceptance: 100.00%
[Korali] Generation 1 - Time: 0.004292s, Annealing: 4.53%, Acceptance: 54.32%
[Korali] Generation 2 - Time: 0.003958s, Annealing: 8.52%, Acceptance: 53.08%
[Korali] Generation 3 - Time: 0.003906s, Annealing: 12.92%, Acceptance: 50.58%
[Korali] Generation 4 - Time: 0.003889s, Annealing: 20.42%, Acceptance: 47.34%
[Korali] Generation 5 - Time: 0.003814s, Annealing: 41.58%, Acceptance: 47.17%
[Korali] Generation 6 - Time: 0.003803s, Annealing: 97.83%, Acceptance: 48.21%
[Korali] Finished. Evidence: -1.154530.
[Korali] Total Time: 0.201538s - Sampling Time: 0.036278s - Engine Time: 0.165186s.
[Korali] Saving results to file: tmcmc.txt.
```

Represents the logEvidence
(the higher, the better)

The Korali Framework

Model Interfaces



Computational Model to Run

Blood Flow Simulation
Airplane Wing Drag Simulation
Heat Distribution Model (heat2d)

Model Interface: Direct Evaluation

Interface: double f(double* x)

Input: An array with parameter values(x)

Output: A double with the value of f(x).

Example: Evaluate the **Ackley** Function

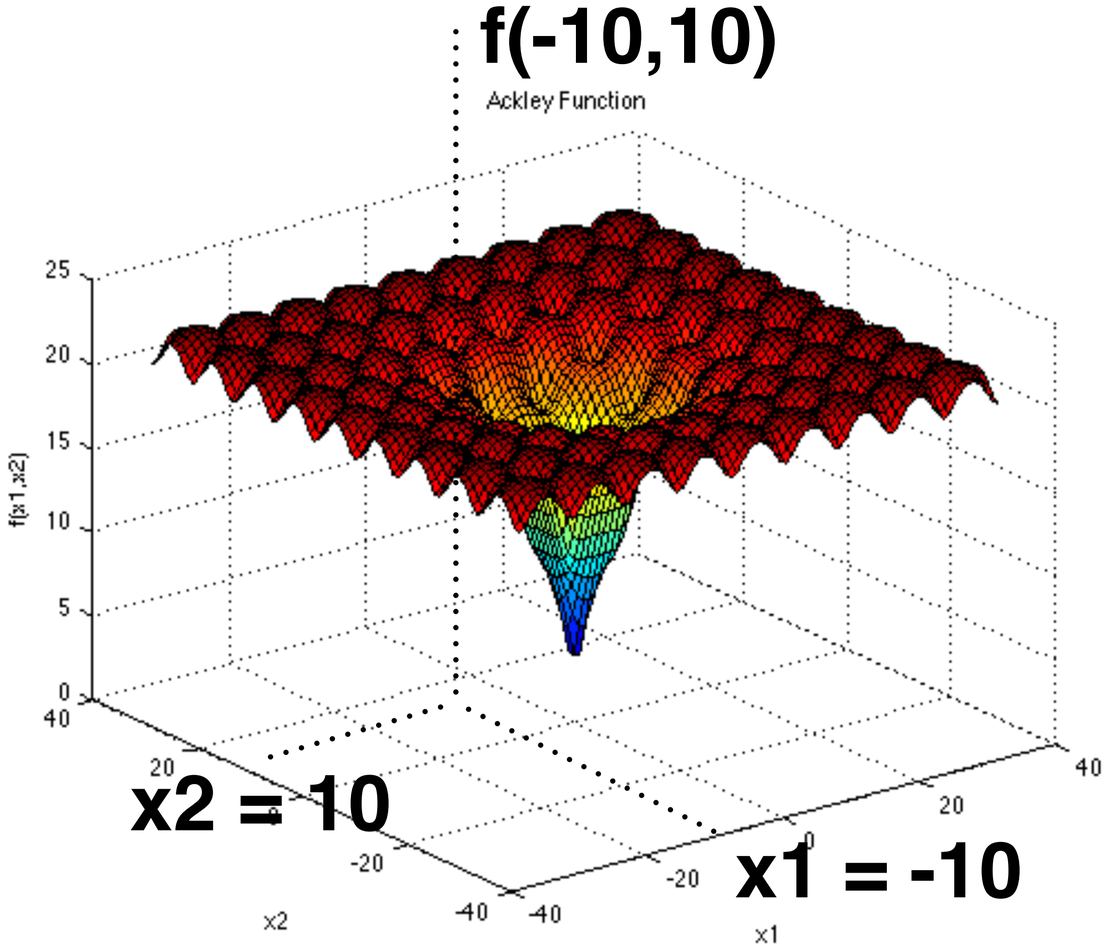
```
double ackley(double *x)
{
  const double a = 20, b = 0.2, c = 2π;

  double s1 = x[0]*x[0] + x[1]*x[1];
  double s2 = cos(c*x[0]) + cos(c*x[1]);

  return -a*exp(-b*sqrt(s1/2)) - exp(s2/2) + a + exp(1.);
}
```

Parameter x1: [-40; 40]

Parameter x2: [-40; 40]



Model Interface: Likelihood / Posterior

Interface: void f(double* x, double* fx)

Input: An array with parameter values(x)

Output: An array with the value of f(x).

Korali will automatically calculate:

$$P(data|x) \propto \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{1}{2\sigma^2}\sum_{i=0}^{n}(f(x_i) - data_i)^2\right)$$

Example: Infer Ackley's a/b/c parameters that best fit the data.

X1	X2	d(x1,x2)	f(x1,x2) / a,b,c
-27.37	1.89	-26.19	
26.98	35.85	-53.79	
38.78	-29.02	-24.04	
34.93	-32.20	-0.07	
24.91	36.68	-17.06	
-7.52	-2.24	-57.63	
-38.23	-18.36	-13.78	
31.68	29.36	-73.60	
26.52	-1.01	-67.16	
-25.32	7.97	-68.12	

Data Provided Calculated

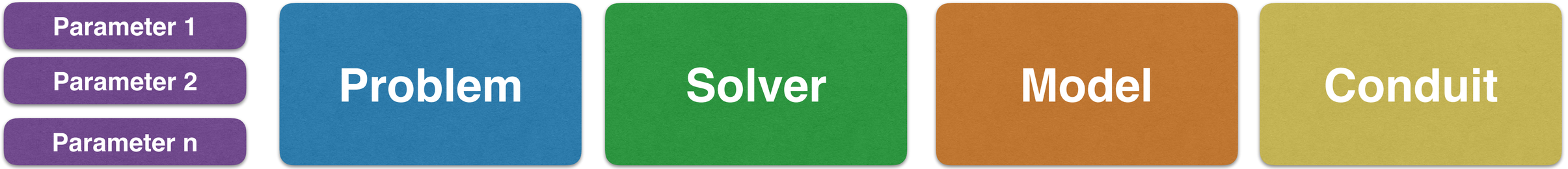
Parameters: a: [10;30] b:[0;0.5] c:[0;4π]

```
void fit_ackley(double *x, double*fx)
{
  double a = x[0], b = x[1], c = x[2];

  for (p in dataPoints)
      fx = ackley(a,b,c, p.x, p.y);
}
```

The Korali Framework

Conduit Modules



How do we run the Solver/Model?

Single-Core

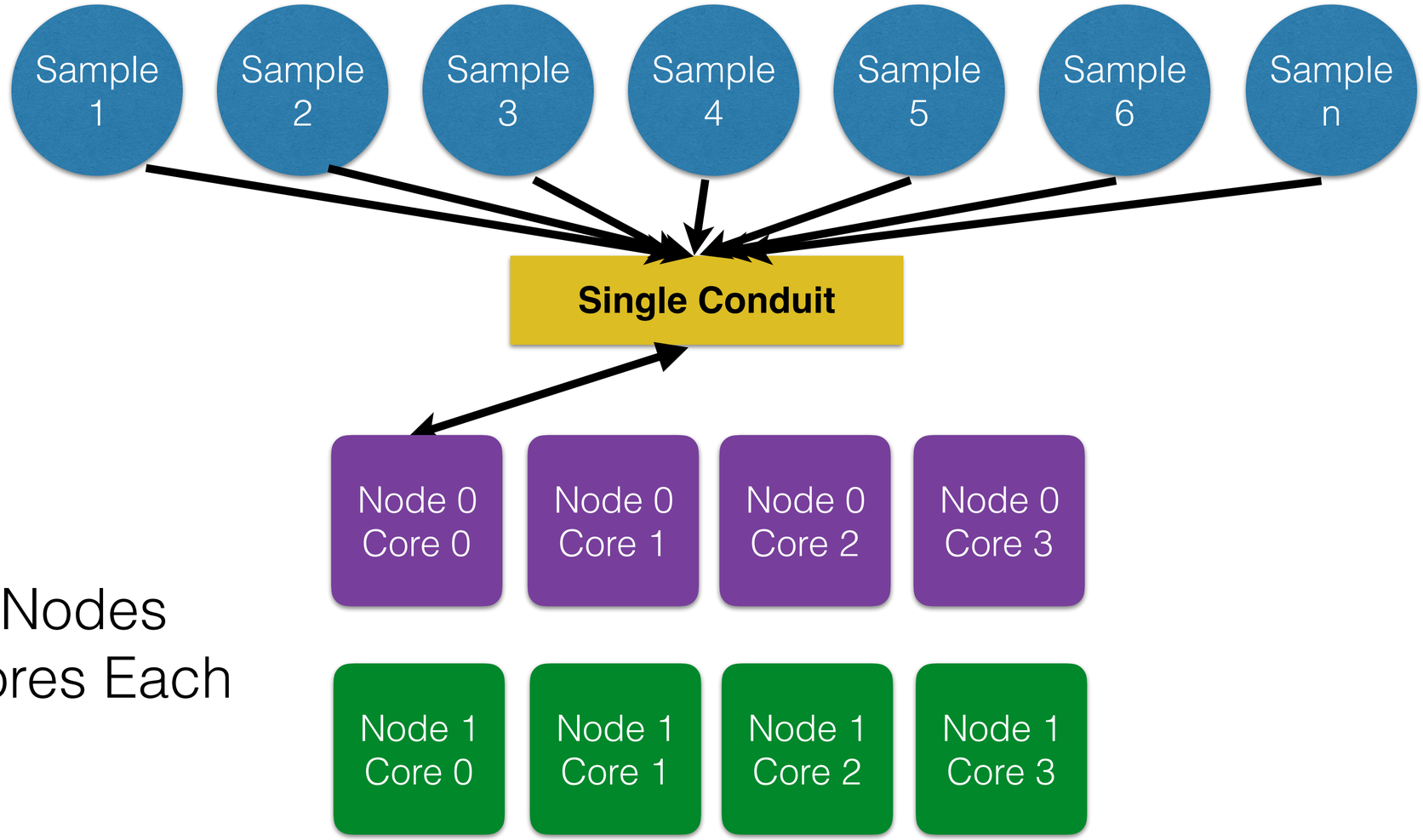
Multi-Core

Distributed Memory

GPU

Module: Single-Core Conduit

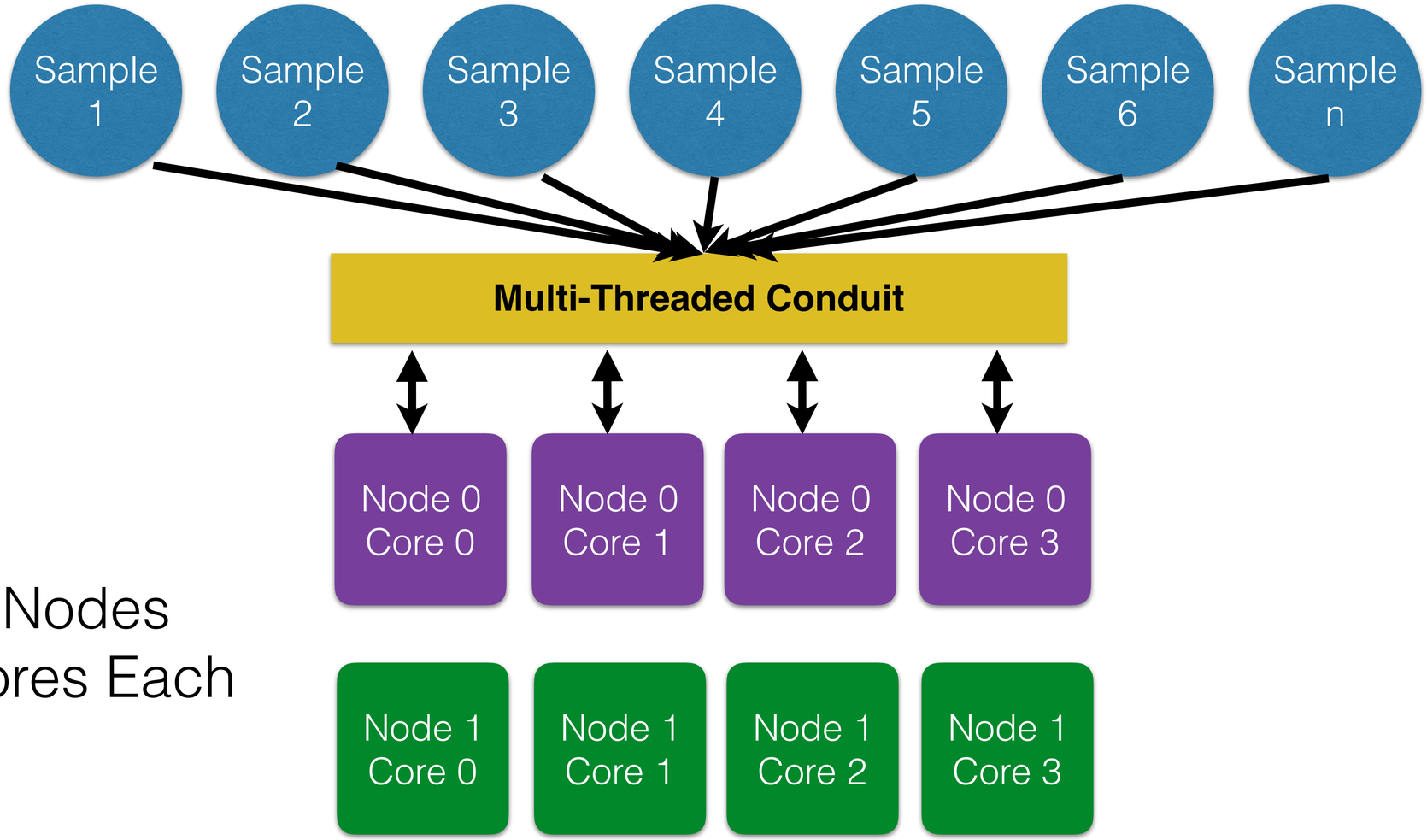
Sample Population (Generation 0)



2 Nodes
4 Cores Each

Module: Multi-Threaded Conduit

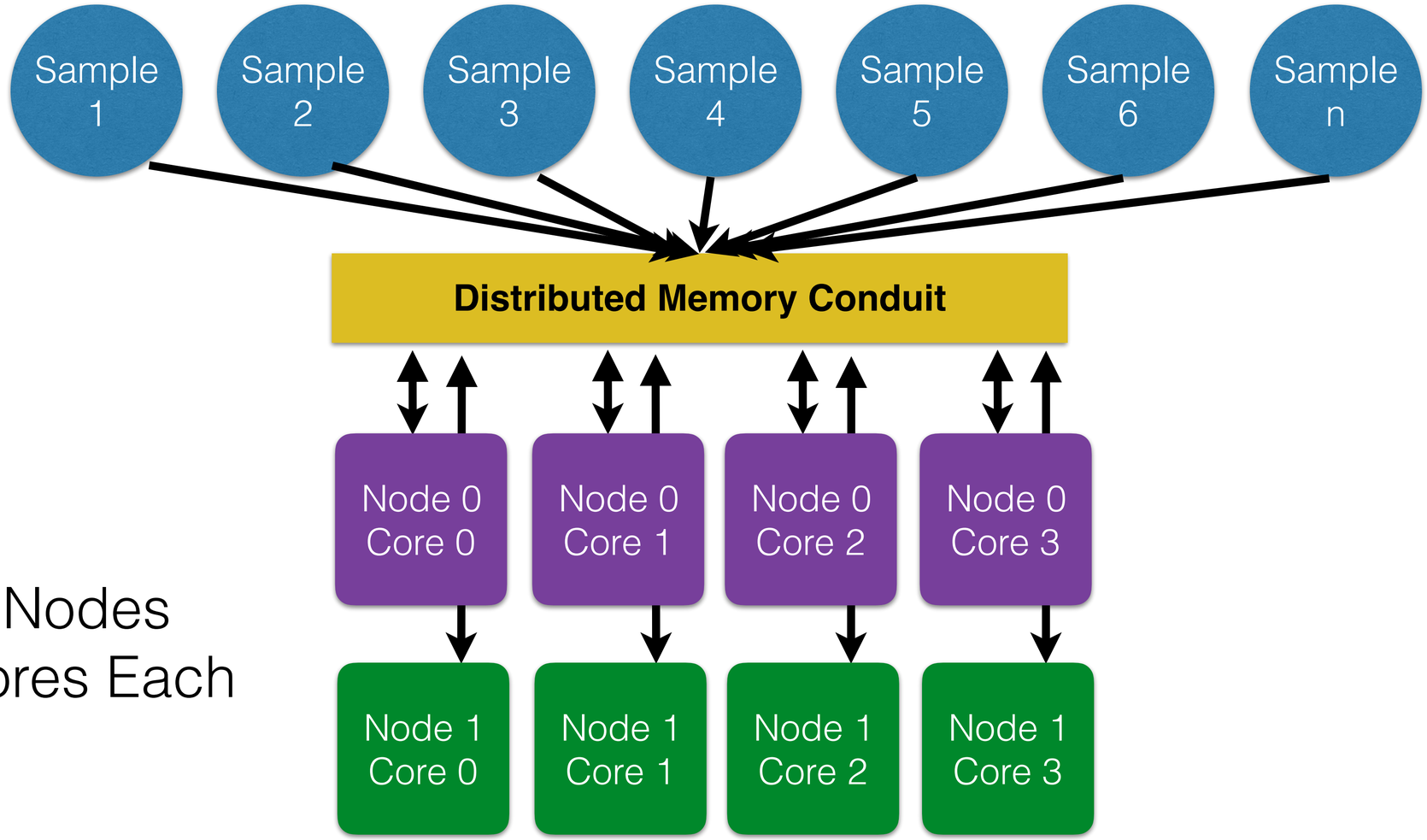
Sample Population (Generation 0)



2 Nodes
4 Cores Each

Module: Distributed Memory Conduit

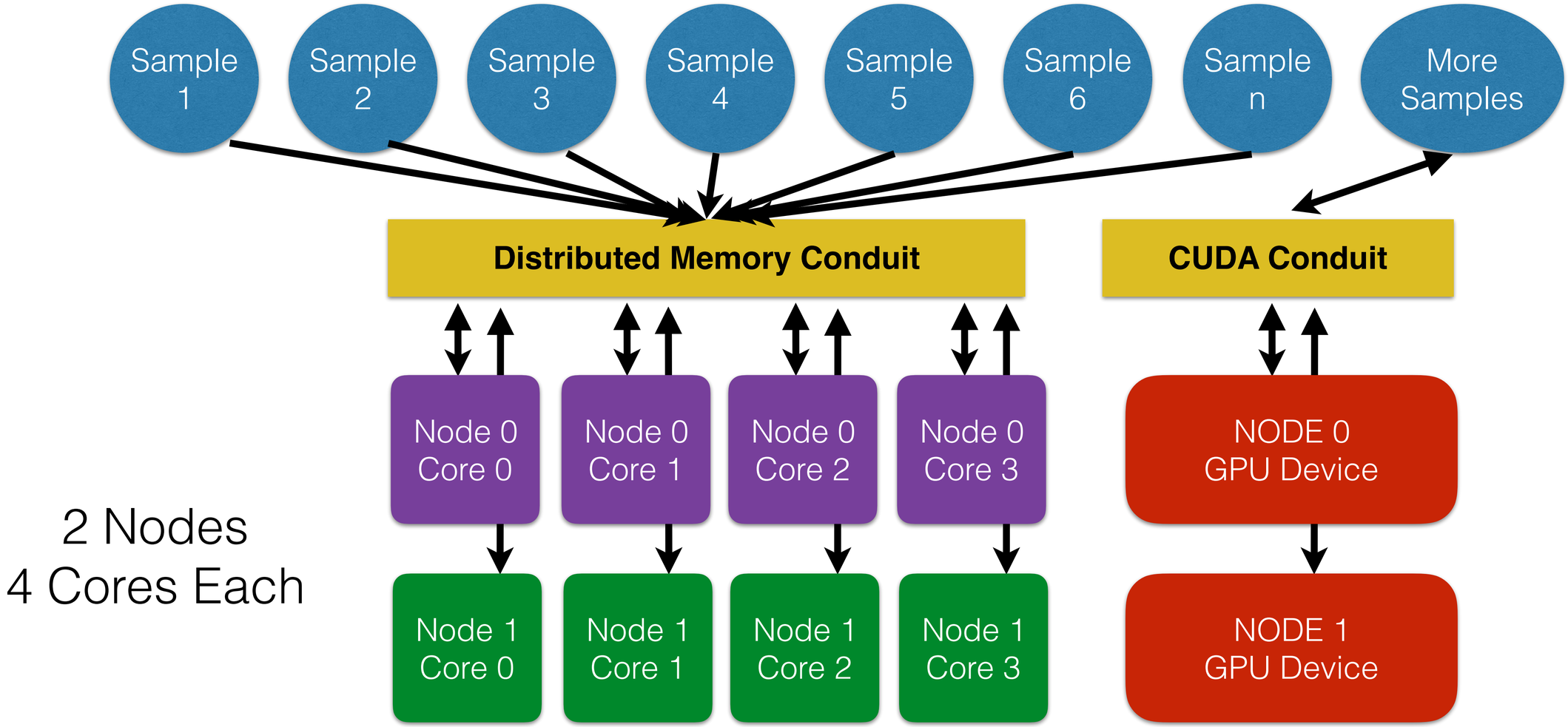
Sample Population (Generation 0)



2 Nodes
4 Cores Each

Module: CUDA Conduit

Sample Population (Generation 0)



2 Nodes
4 Cores Each

Selecting Korali's Conduit

Set by an Environmental Variable: KORALI_CONDUIT

Single-Core Conduit:
$export KORALI_CONDUIT=**single** ← Only one available for HW3

Multi-Threaded Conduit (OpenMP):
$export KORALI_CONDUIT=**openMP**

Multi-Threaded Conduit (Pthreads):
$export KORALI_CONDUIT=**pthreads**

Distributed Memory Conduit (UPC++):
$export KORALI_CONDUIT=**upcxx** ← You will program this one for HW4!

Distributed Memory Conduit (MPI):
$export KORALI_CONDUIT=**mpi**

GPU Conduit (cuda):
$export KORALI_CONDUIT=**cuda**

Putting it all together!

Parameter 1

Parameter 2

Parameter n

Problem

Solver

Model

Conduit

2D Ackley Function