The Current State and Evolution of Stan

Rok Češnovar

Stan Development Team

Faculty of Computer and Information Science, University of Ljubljana

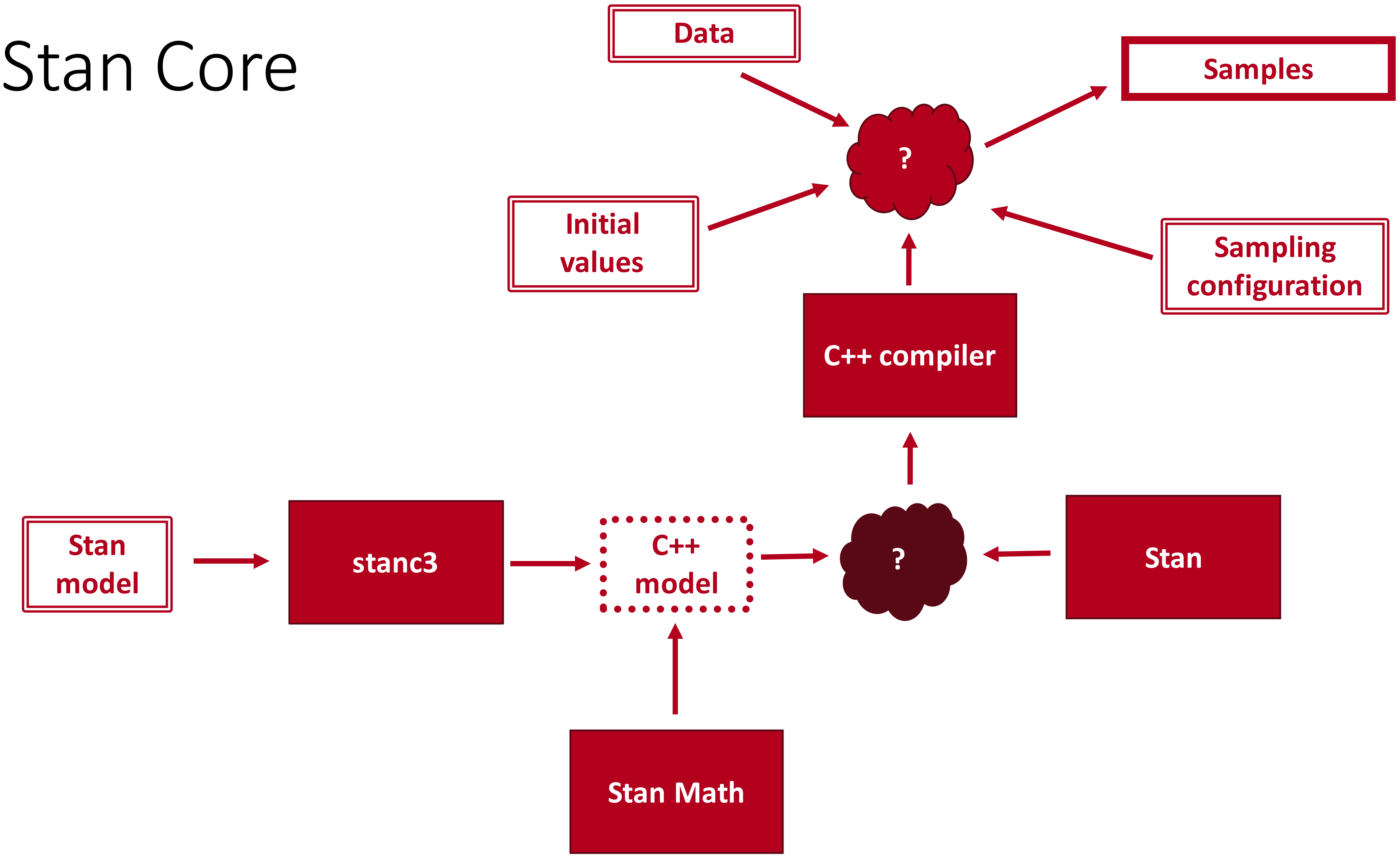
Outline

- Core modules of Stan
- Interfaces
- Feature highlights
- Tips and tricks
- Features in the pipeline

Requirements for a Bayesian Inference Framework

- Domain specific language
  - Stan language - **stanc3**
- Sampling algorithm
  - Dynamic HMC - **Stan**
- Automatic differentiation
  - **Stan Math**

Stan Core



Data
Samples
?
Initial values
Sampling configuration
C++ compiler
Stan model
stanc3
C++ model
?
Stan
Stan Math

Stan Math

- Automatic differentiation with operator overloading in C++

```
double a = 5; int b = 4; double c = 6; double f;
f = a * a * b + b * sqrt(c) * a;
```

Stan Math

- Function parameters -> `stan::math::var`
  - Value
  - Adjoint

```
var a = 5; int b = 4; var c = 6;
var f = a * a * b + b * sqrt(c) * a;

f.grad()

f.val(), a.adj(), c.adj()
```

`stan::math::var`

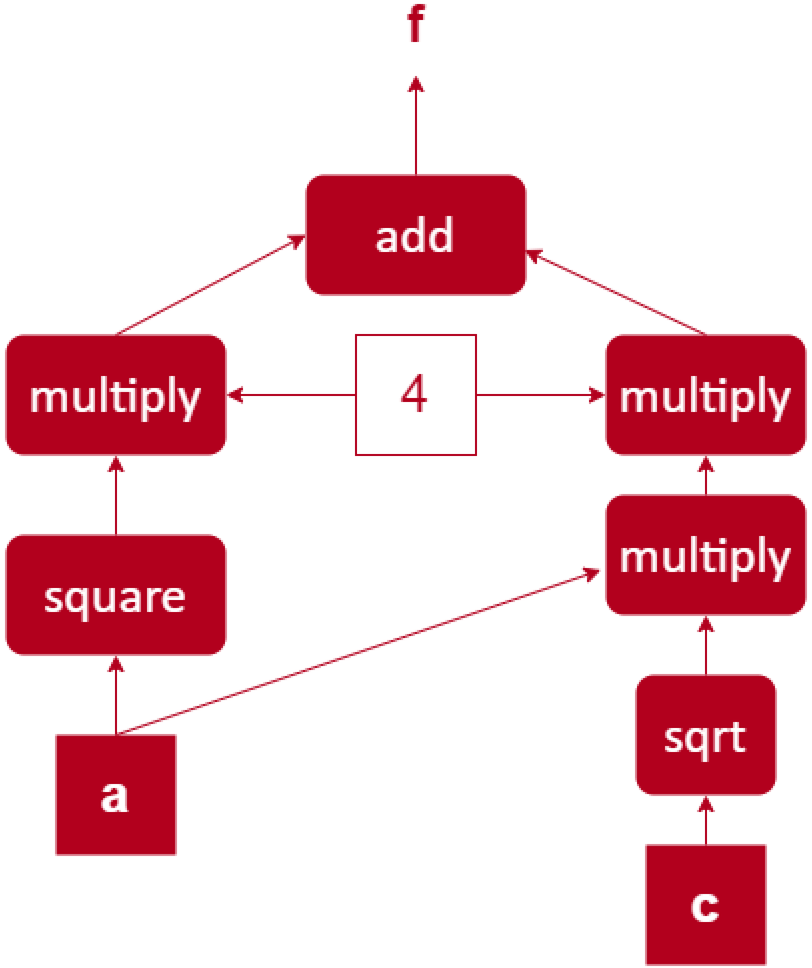
- all functions must support `var` arguments and defined derivatives

```
multiply(double, double)
multiply(var&, var&)
multiply(var&, double)
multiply(double, var&)
```

- data and parameter containers
  - `std::vector<T1>`
    - T1 = int, double, var
  - `Eigen::Matrix<T2, -1, -1>`
    - T2 = double, var

Reverse mode automatic differentiation

```
var a = 5; int b = 4; var c = 6; var f;
f = square(a) * b + a * sqrt(c) * b;
f.grad()
```



stanc3

- Translates the Stan model to a C++ class representing the model
- Type translation - data

Stan data type	C++ type
int	int
real	double
vector	Eigen::Matrix<double, -1, 1>
row_vector	Eigen::Matrix<double, 1, -1>
matrix	Eigen::Matrix<double, -1, -1>
T []	std::vector<T>

stanc3

- Translates the Stan model to a C++ class representing the model
- Type translation - parameters

Stan parameter type	C++ type
int	N / A
real	var
vector	Eigen::Matrix<var, -1, 1>
row_vector	Eigen::Matrix<var, 1, -1>
matrix	Eigen::Matrix<var, -1, -1>
T []	std::vector<T>

stanc3

- Written in OCaml
- Data variables are translated to private member of the class

```
data {
  int N;
  int n_redcards[N];
  int n_games[N];
  vector[N] rating;
}
```

```
class redcard_model final :
 public model_base_crtp<test_model> {

 private:
  int N;
  std::vector<int> n_redcards;
  std::vector<int> n_games;
  Eigen::Matrix<double, -1, 1> rating;
```

- The data is read and optionally transformed in in the class constructor

Stanc3

- Parameters, transformed parameters and the model block is implemented in the `log_prob` member function of the C++ class
- Uses functions implemented in Stan Math

```
parameters {
  vector[2] beta;
}
```

```
stan::math::accumulator<T__> lp_accum__;
stan::io::deserializer<T__> in__(params_r__, params_i__);

Eigen::Matrix<T__, -1, 1> beta
                        = Eigen::Matrix<T__, -1, 1>(2);

beta = in__.template read<Eigen::Matrix<T__, -1, 1>>(2);
```

Stanc3

```
model {
  target += normal_lpdf(beta | 0, 1);
  n_redcards ~ binomial_logit(n_games, beta[1] + beta[2] * rating);
}
```

```
lp_accum__.add(normal_lpdf<false>(beta, 0, 1));
lp_accum__.add(
    binomial_logit_lpmf<propto__>(
                n_redcards, n_games,
                add(beta[0], multiply(beta[1], rating))
    )
);
return lp_accum__.sum();
```

Stanc3

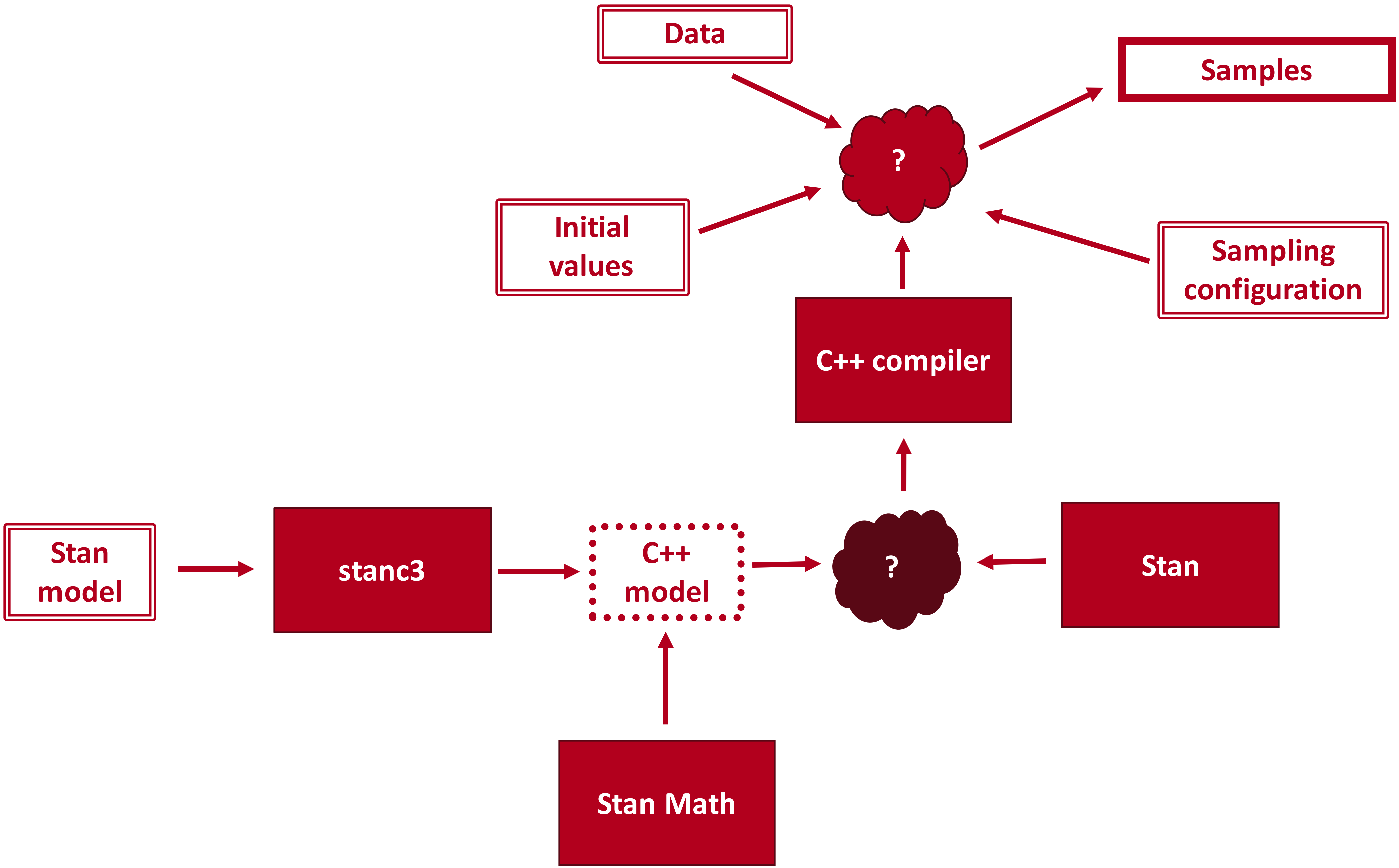
```
model {
  target += normal_lpdf(beta | 0, 1);
  n_redcards ~ binomial_logit(n_games, beta[1] + beta[2] * rating);
}
```

```
lp_accum__.add(normal_lpdf<false>(beta, 0, 1));
lp_accum__.add(
    binomial_logit_lpmf<propto__>(
                n_redcards, n_games,
                add(rvalue(beta, "beta", index_uni(1)),
                    multiply( rvalue(beta, "beta", index_uni(2)),  rating) )
    )
);
return lp_accum__.sum();
```

Stan

- Algorithms
  - Dynamic HMC & HMC
  - ADVI
  - Optimization
- Model helper functions
  - rvalue, assign
  - serialization/deserialization

Interfaces



Data
Samples
?
Initial values
Sampling configuration
C++ compiler
Stan model
stanc3
C++ model
?
Stan
Stan Math

CmdStan

Data	JSON or RDump files
Initial values	Command line argument or JSON/Rdump files
Sampling configuration	Command line arguments
Samples	CSV files
Stan model	A .stan file
stanc3	A binary executable shipped with CmdStan releases
Combining core modules	Standard C++
Compiled model	Executable file (.exe)

CmdStan

- Relases available at https://github.com/stan-dev/cmdstan/releases

- Fastest compile times
- Easiest to fine tune performance with C++ compiler flags
- Only interface in CI/CD
- Slower I/O
  - Use JSON input: ~3s for reading 100MB of floating point values
  - Output: ~20s for 1000 samples of 50k parameters/generated quantities

RStan 2.26 and beyond

Data	R list passed to Stan C++ via Rcpp
Initial values	R arguments passed to C++ functions via Rcpp
Sampling configuration	R function arguments passed to C++ functions via Rcpp
Samples	R vectors
Stan model	R string
stanc3	Javascript version of stanc3 (translated from OCaml)
Combining core modules	Rcpp and C++
Compiled model	Object file that is linked with R

RStan 2.26 and beyond

- Not currently available on CRAN, only on stan-dev R packages repository https://mc-stan.org/r-packages/

```
remove.packages(c("rstan", "StanHeaders"))
install.packages("rstan", repos = c("https://mc-stan.org/r-packages/", getOption("repos")))
```

- A significant overhaul due to the change in the Stan-to-C++ compiler
- Additional features not supported via CmdStan:

`log_prob`, `grad_log_prob`, `unconstrain_pars`, `expose_stan_functions`

PyStan 3

- PyStan was split in two parts:
  - Httpstan
    - HTTP-based REST interface
    - Serves as a PyStan3 backend for the Stan C++
  - PyStan3
    - Python-only package, easier to develop and maintain the user-facing functionalities
- Additional features like `log_prob`, `unconstrain_pars` also available
- Currently supports the default dynamic HMC sampler
- Supported OS: Linux, MacOS with Python 3.7+

httpstan

Data	JSON object in HTTP request body
Initial values	JSON object in HTTP request body
Sampling configuration	JSON object in HTTP request body
Samples	JSON in HTTP response
Stan model	String in HTTP body request
stanc3	A binary executable shipped with CmdStan releases
Combining core modules	Cython and C++
Compiled model	Object file that is linked with Python

PyStan

Data	Python objects
Initial values	Python object
Sampling configuration	Arguments to Python function
Samples	Python object
Stan model	Python string
stanc3	N/A - compiled with httpstan
Combining core modules	N/A - delegated to httpstan
Compiled model	N/A - the model is compiled with httpstan

CmdStanR/CmdStanPy

- Light-weight wrappers for CmdStan for R an Python
- Benefits/drawbacks of CmdStan with ease of use via R/Python
- Less tight coupling of R/Python and Stan C++
  - Easier to detect and automatically handle installation issues
  - Seamless version updates
  - Access to MPI/OpenCL capabilites (via CmdStan)
  - Feature like log_prob, unconstrain_pars are not available

CmdStanR/Py

Data	R/Python objects
Initial values	R/Python objects
Sampling configuration	Arguments to R/Python function
Samples	CSV files read to R/Python objects
Stan model	Stan files
stanc3	Binaries in the CmdStan releases
Combining core modules	Standard C++ in CmdStan
Compiled model	CmdStan produced executables

Feature highlights

Expression templates

- Stan Math optimization – `vector/row_vector/matrix` functions return expressions not containers of values

$$\underbrace{\underbrace{\texttt{x + x}}_{\texttt{temp1}} + \underbrace{\texttt{y * y}}_{\texttt{temp2}}}_{\texttt{temp3}}$$

- Speedup for functions that use data and functions in the generated quantities
- Speedups for all log probability mass/density functions

GLM functions

- a more efficient way of writing generalized linear models (up to

```
y ~ bernoulli_logit(alpha + beta * x);
```

↓

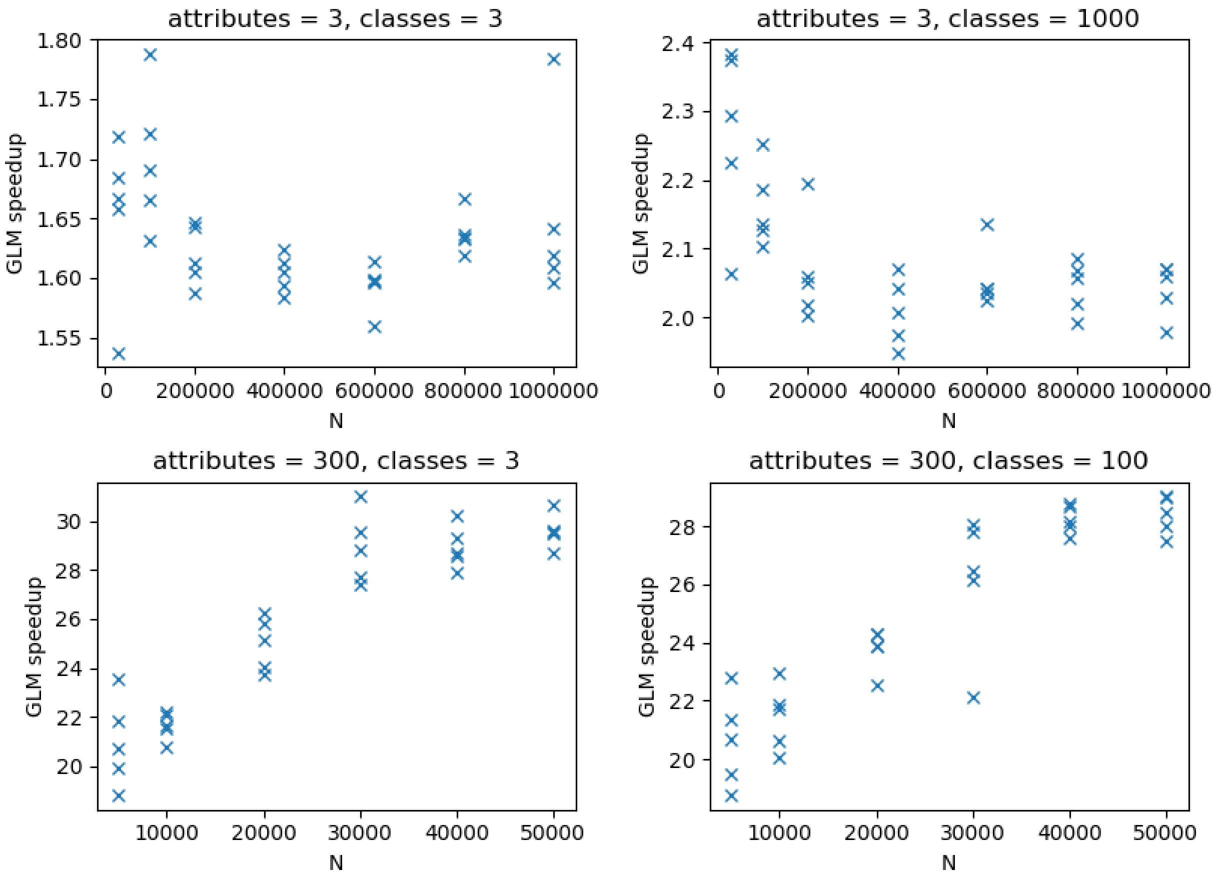
```
y ~ bernoulli_logit_glm (x, alpha, beta);
```

bernoulli_logit_glm_lpmf, neg_binomial_2_log_glm_lpmf, normal_id_glm_lpdf, poisson_log_glm_lpmf,

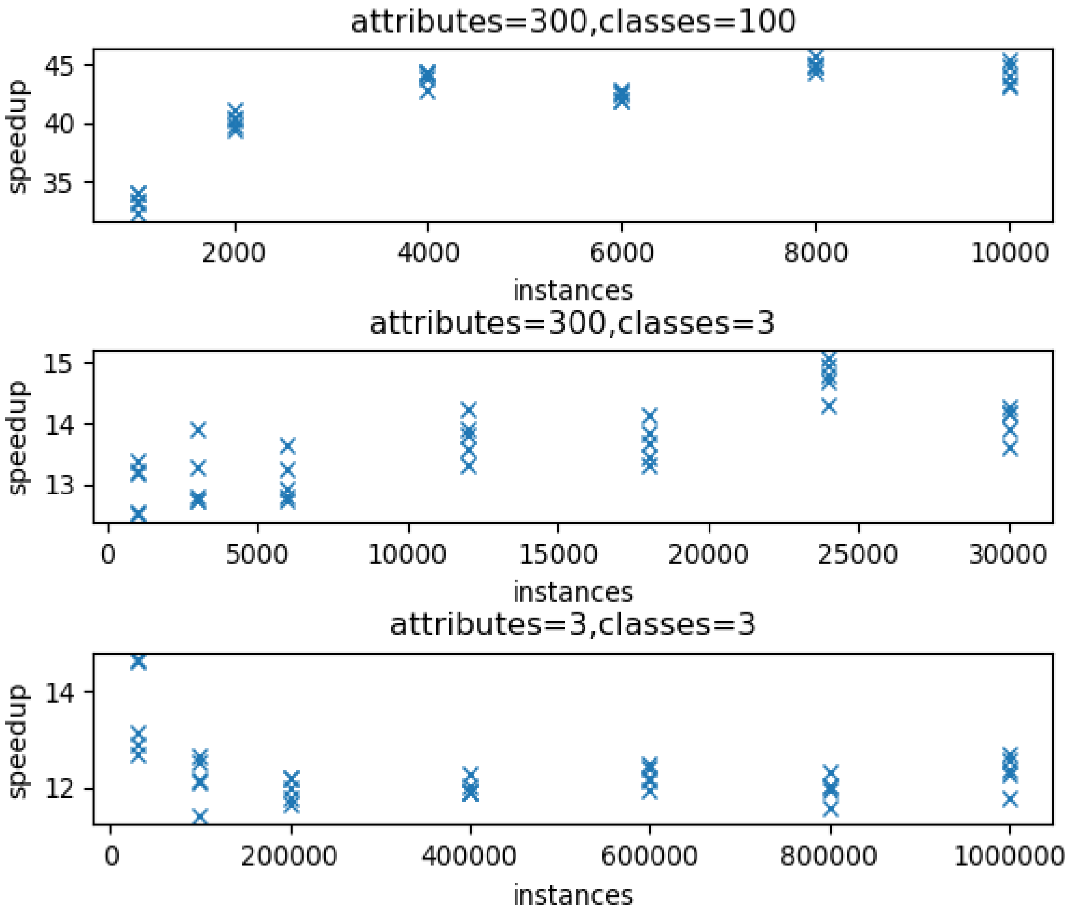
categorical_logit_glm_lpmf, ordered_logistic_glm_lpmf

GLM functions

- Ordinal regression GLM



- Softmax regression GLM



Profiling

- Identifying bottlenecks in your model
- Evaluating parallelization or use of different functions / parametrization

```
model {
    profile("priors") {
        target += std_normal_lpdf(beta);
        target += std_normal_lpdf(alpha);
    }
    profile("likelihood") {
        target += bernoulli_logit_lpmf(y | X * beta + alpha);
    }
}
```

Profiling

- Reports total time, time spent in the forward and reverse pass and the number of gradient evaluations

name	total_time	forward_time	reverse_time	autodiff_calls
likelihood	1.00471000	0.85333200	0.15137400	17607
priors	0.00732542	0.00603501	0.00129041	17607

Language features

- New ODE interface
- New array syntax
- reduce_sum and within-chain parallelization

Within-chain parallelization

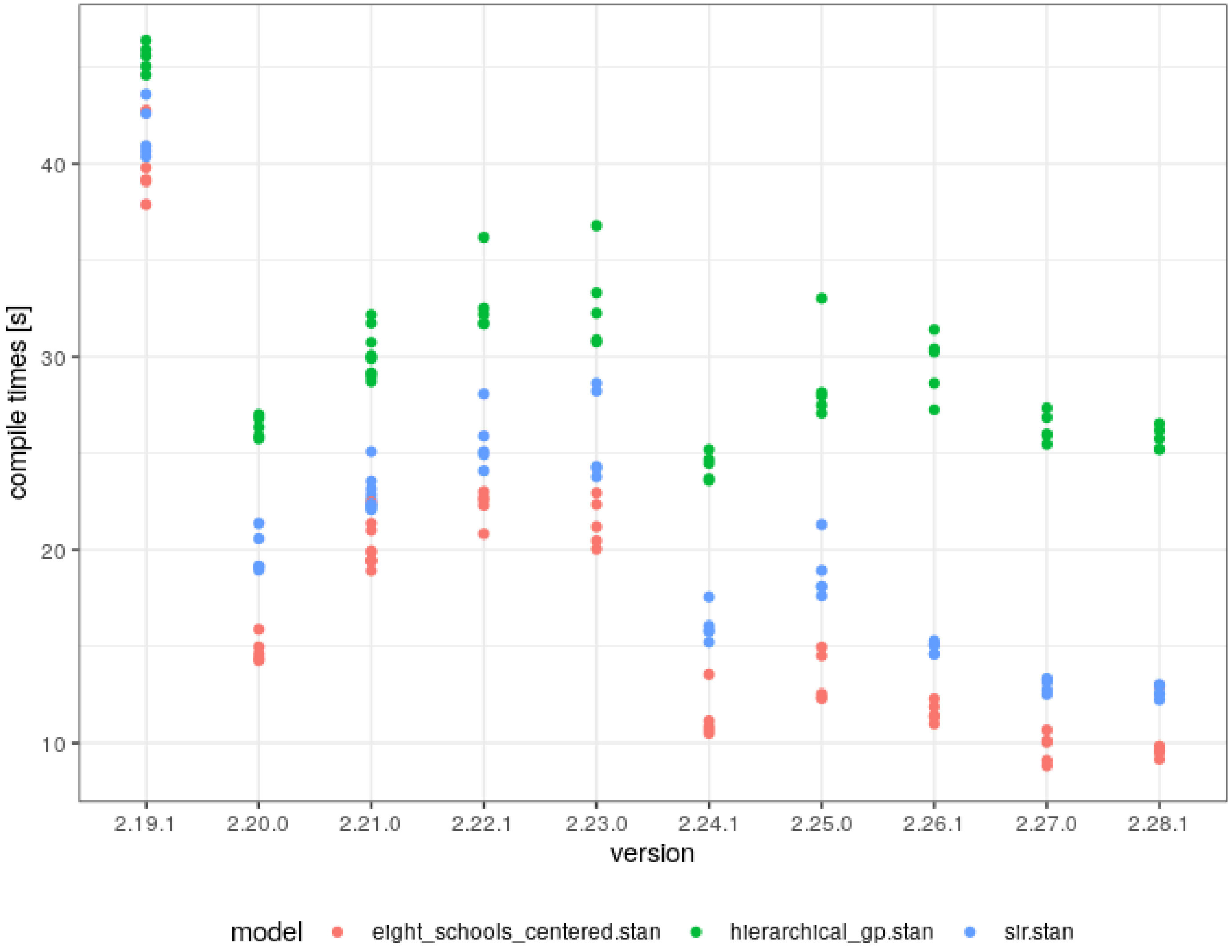
```
target += binomial_logit_lupmf(n_redcards | n_games, beta[1] + beta[2] * rating);

real partial_sum_lpmf(
        array[] int n_redcards, array[] int n_games, vector rating, vector beta) {
  return binomial_logit_lupmf(n_redcards | n_games, beta[1] + beta[2] * rating);
}

real partial_sum_lpmf(array[] int slice_n_redcards,
                      int start, int end,
                      array[] int n_games, vector rating, vector beta) {
return binomial_logit_lupmf(slice_n_redcards|n_games[start:end],beta[1] + beta[2] *
rating[start:end]);}
```

Within-chain parallelization

```
target += reduce_sum(
   partial_sum_lupmf, n_redcards, grainsize, n_games, rating, beta
);
```

- no additional parallelization frameworks required
  - Intel TBB - bundled with Stan Math

Faster compilation



Tips and tricks

- `vector` instead of `array` wherever applicable
  - Expression templating, faster functions, support for immutable matrices/vectors
- Write vectorized code – future performance proof
- ARM vs Intel
  - Up to 3x speedups with multiple chains or within-chain parallelization
  - Amazon EC2 c6g vs c5
- Windows – WSL
  - 1.5x – 2.5x speedups

Future

https://github.com/stan-dev/design-docs

New language features

- Closures (N. Huurre, B. Carpenter)
- Tuples (R. Bernstein, B. Carpenter)
- Ragged arrays (B. Carpenter)
- Complex numbers (B. Ward, B. Carpenter, S. Bronder)

Immutable matrices

S. Bronder, T. Ciglarič, B. Bales, R. Češnovar, B. Carpenter

```
Eigen::Matrix<var, -1, -1>
```

```
var<Eigen::Matrix<double, -1, -1>>
```

- A new templated stan::math::var class -> var_value<T>
  - stan::math::var = stan::math::var<double>

```
stan::math::var_value<Eigen::Matrix<var, -1, -1>>
```

Immutable matrices/vectors

S. Bronder, T. Ciglarič, B. Bales, R. Češnovar, B. Carpenter

- Function evaluations and gradient evaluations are up to 3 times faster with immutable containers (some even 10 times)
- Math supported is mostly complete
- Requires careful stanc3 optimization
  - Slower for indexed code
- Write vectorized code wherever possible

OpenCL

R. Češnovar, T. Ciglarič, S. Bronder

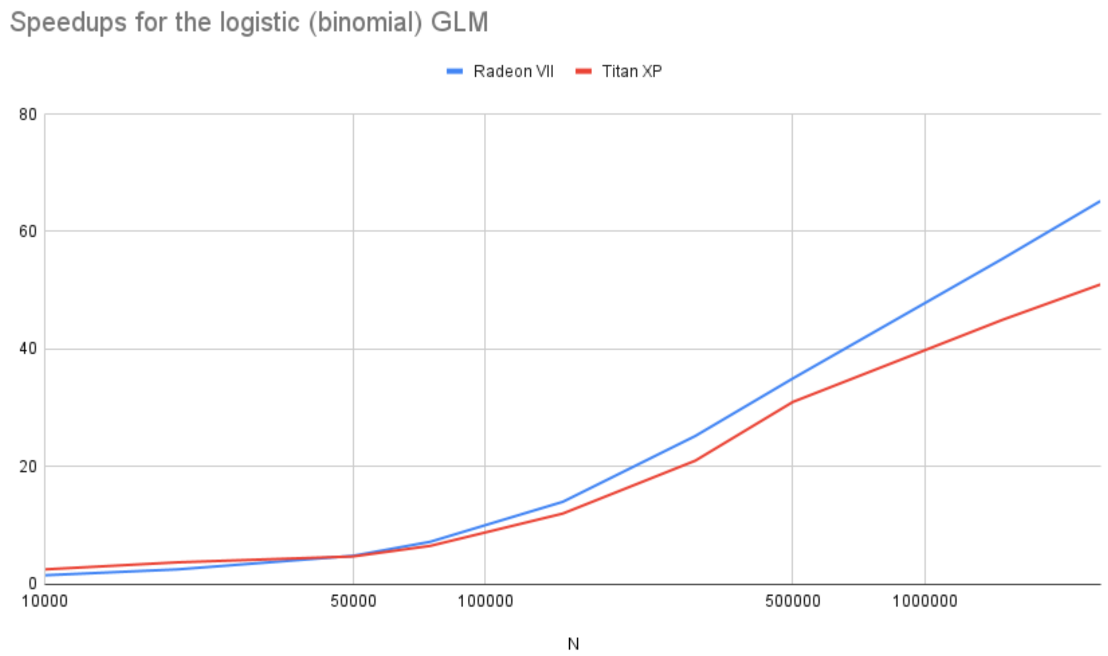
- `matrix_cl<T>` - data vector/matrix/arrays on GPUs
- `var_value<matrix_cl<double>>` parameters vector/matrix/array on GPUs

```
using stan::math;
Eigen::Matrix<double, -1, -1> A = Eigen::Matrix<double, -1, -1>::Random(N, M);
Eigen::Matrix<double, -1, -1> B = Eigen::Matrix<double, -1, -1>::Random(M, N);

var_value<matrix_cl<double>> A_cl = to_matrix_cl(A);
var_value<matrix_cl<double>> B_cl = to_matrix_cl(B);
var_value<matrix_cl<double>> C_cl = A_cl * B_cl;
var C_sum = sum(C_cl);
C_sum.grad();
```

OpenCL

R. Češnovar, T. Ciglarič, S. Bronder

- Support in Math for most functions
  - No HOF support
- Partial support in stanc3
  - All lpdf/lpmf functions can use GPUs as of 2.26



Embedded Laplace approximation

C. Margossian, A. Vehtari, D. Simpson, R. Agrawal

- Marginalize out some of the parameters and run MCMC on a reduced, better behaved parameter space.
- The marginalized-out parameters are later recovered in the generated quantities block.
- Laplace approximation is used for marginalization.

- https://github.com/charlesm93/StanCon2020
- https://arxiv.org/abs/2004.12550

Embedded Laplace approximation

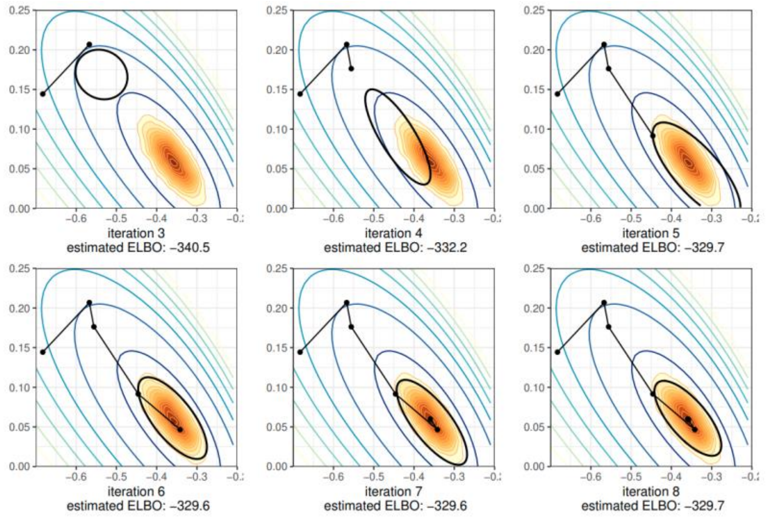
C. Margossian, A. Vehtari, D. Simpson, R. Agrawal

```
model {
  rho ~ inv_gamma(rho_location_prior, rho_scale_prior);
  alpha ~ inv_gamma(alpha_location_prior, alpha_scale_prior);
  target += laplace_marginal_poisson_log_lpmf(
                 y | n_samples, ye, K, phi, x, delta,delta_int, theta_0);
}

generated quantities {
  vector[n_obs] theta
    = laplace_poisson_log_rng(y, n_samples, ye, K,
                              phi, x, delta, delta_int, theta_0);
}
```

Pathfinder

L. Zhang, B. Carpenter, A. Gelman, A. Vehtari, S. Bronder

- Locates low-rank normal approximations to the target density along a quasi-Newton optimization path.
- Evaluates the ELBO in parallel for each normal approximation and returns draws from the approximation that maximizes the ELBO.
- Requires one to two orders of magnitude fewer log density and gradient evaluations.
- Can be used to replace parts of warmup in HMC
- Paper: https://arxiv.org/abs/2108.03782



Thank you!

- A big thanks to all our developers, users and sponsors!
  - https://mc-stan.org/about/team/
  - https://github.com/sponsors/stan-dev
  - https://numfocus.org/donate-to-stan

- Help us out by contributing code, documentation or ideas at https://github.com/stan-dev/