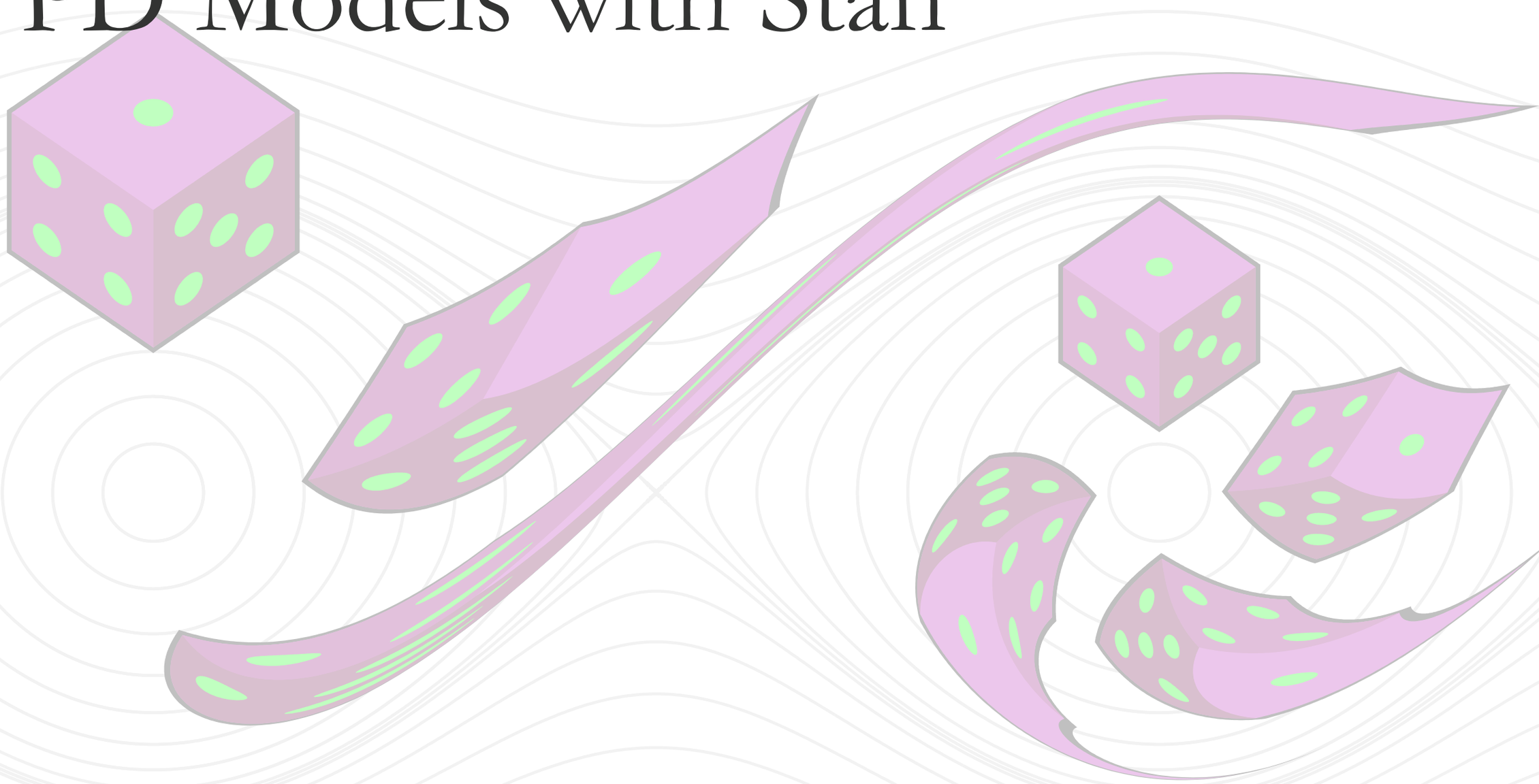
Fitting Complex PK/PD Models with Stan



Michael Betancourt
@betanalpha, @mcmc_stan
University of Warwick

Bayes Pharma
Novartis Basel Campus
May 20, 2015

Because of the clinical applications, precise PK/PD modeling is incredibly important.

PK/PD Model

θ

Because of the clinical applications, precise PK/PD modeling is incredibly important.

PK/PD Model

θ

Treatment

$R(T, \theta)$

Robust treatment decisions requires modeling both the latent PK/PD process and the measurement.

PK/PD Model

θ

Treatment

$R(T, \theta)$

Robust treatment decisions requires modeling both the latent PK/PD process and the measurement.

PK/PD Model

θ

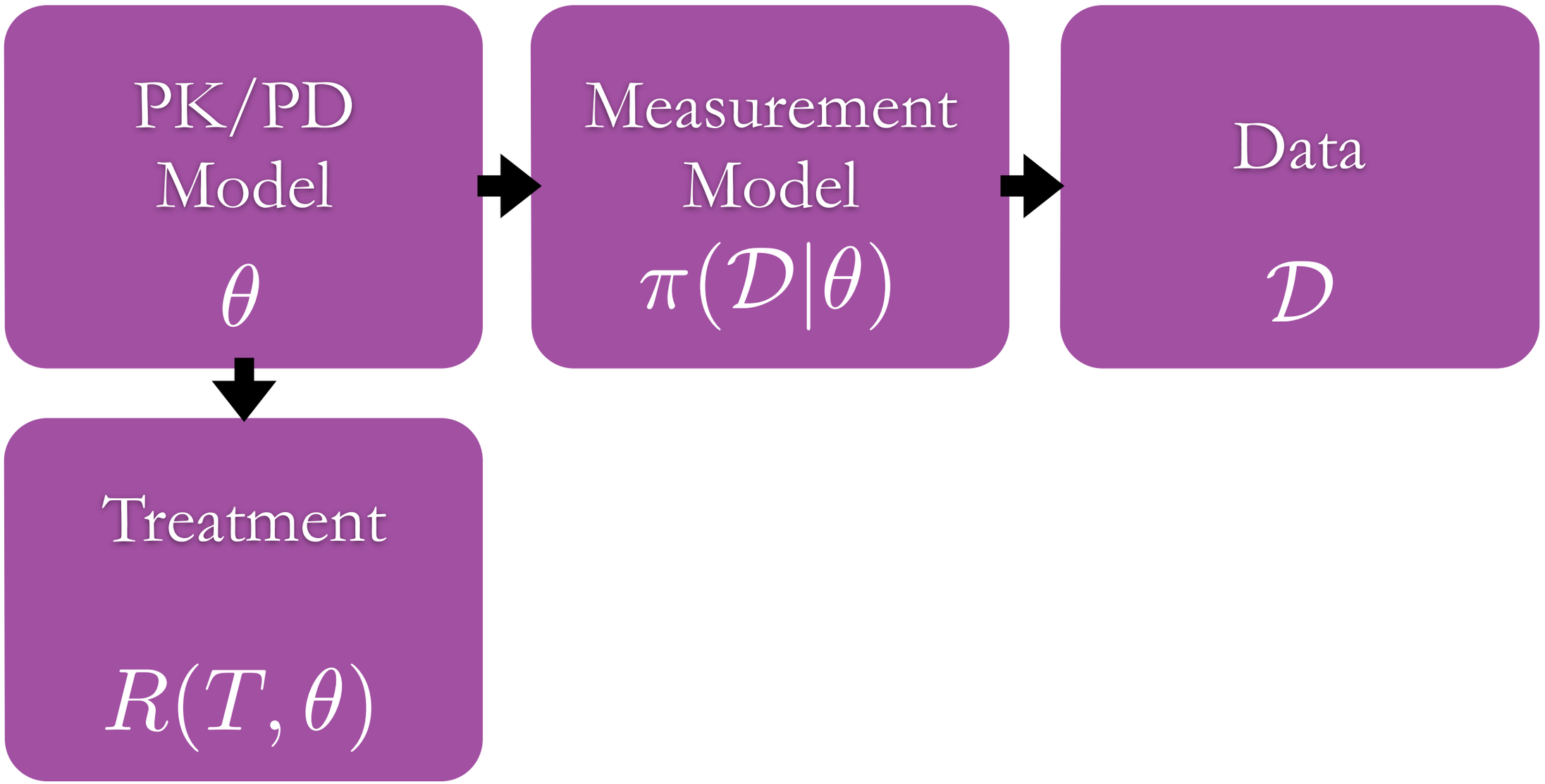
Data

$\mathcal{D}$

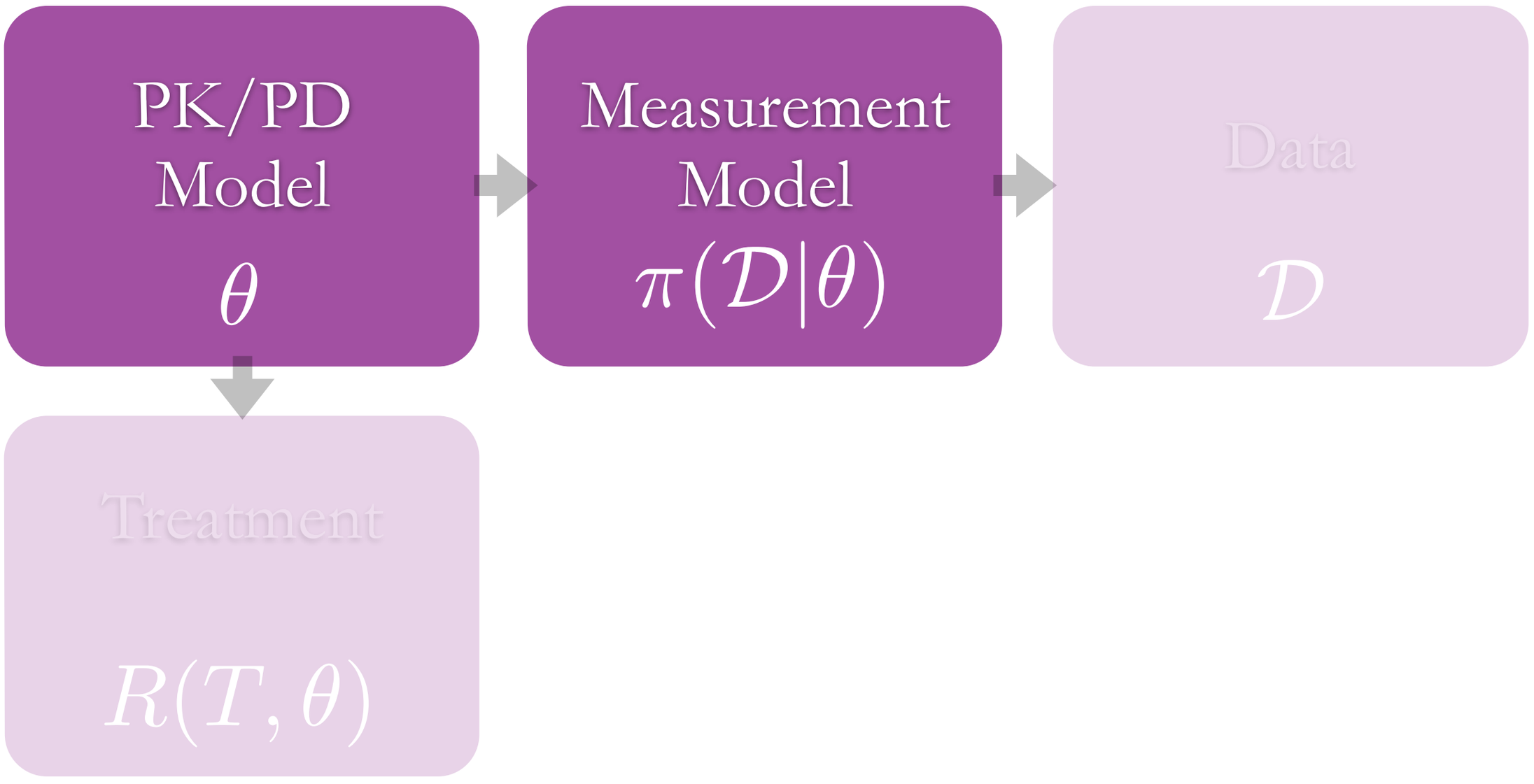
Treatment

$R(T, \theta)$

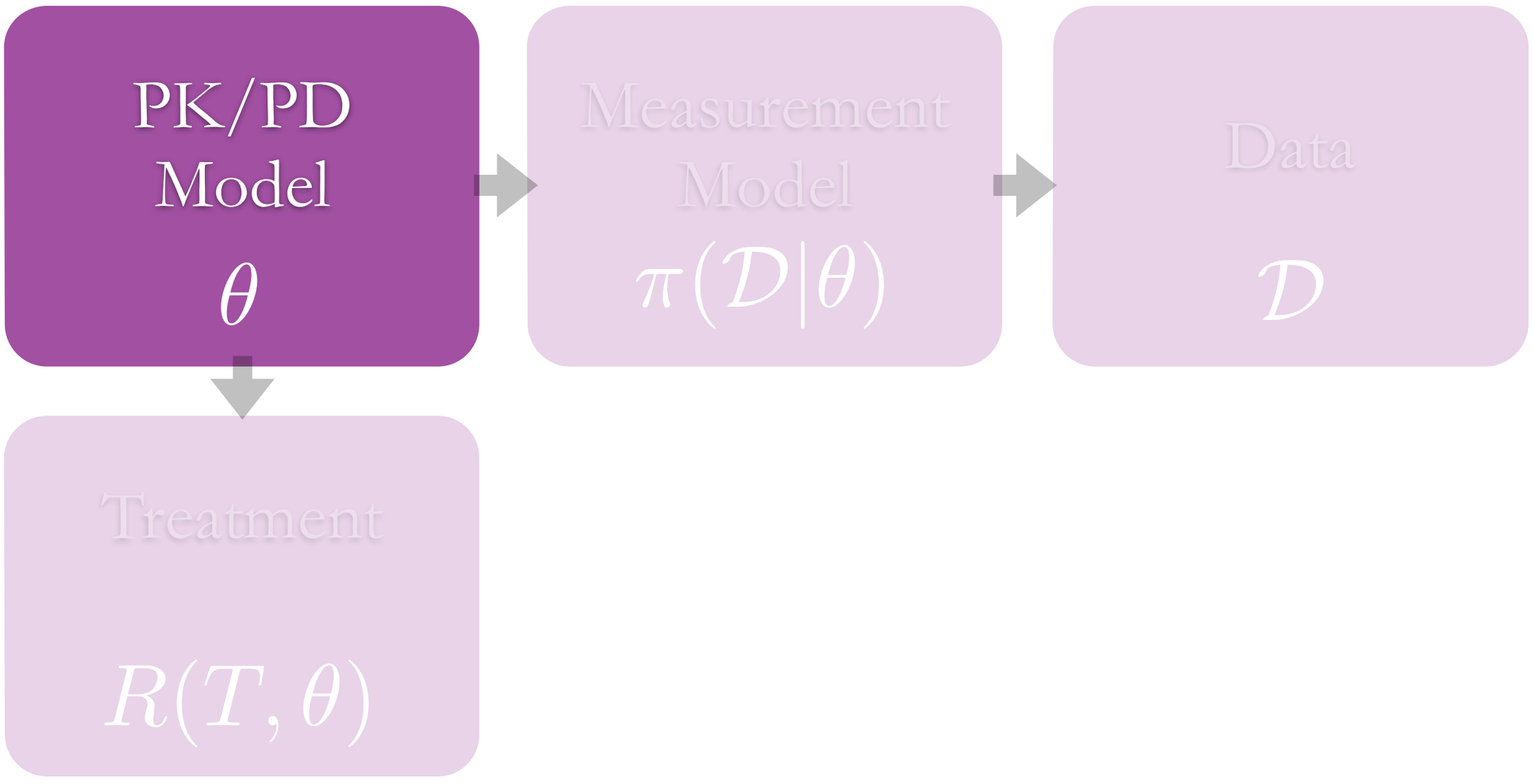
Robust treatment decisions requires modeling both the latent PK/PD process and the measurement.



Analytic models are common due to their computational convenience but ultimately too restrictive.



Analytic PK/PD models, for example, are exactly solvable.



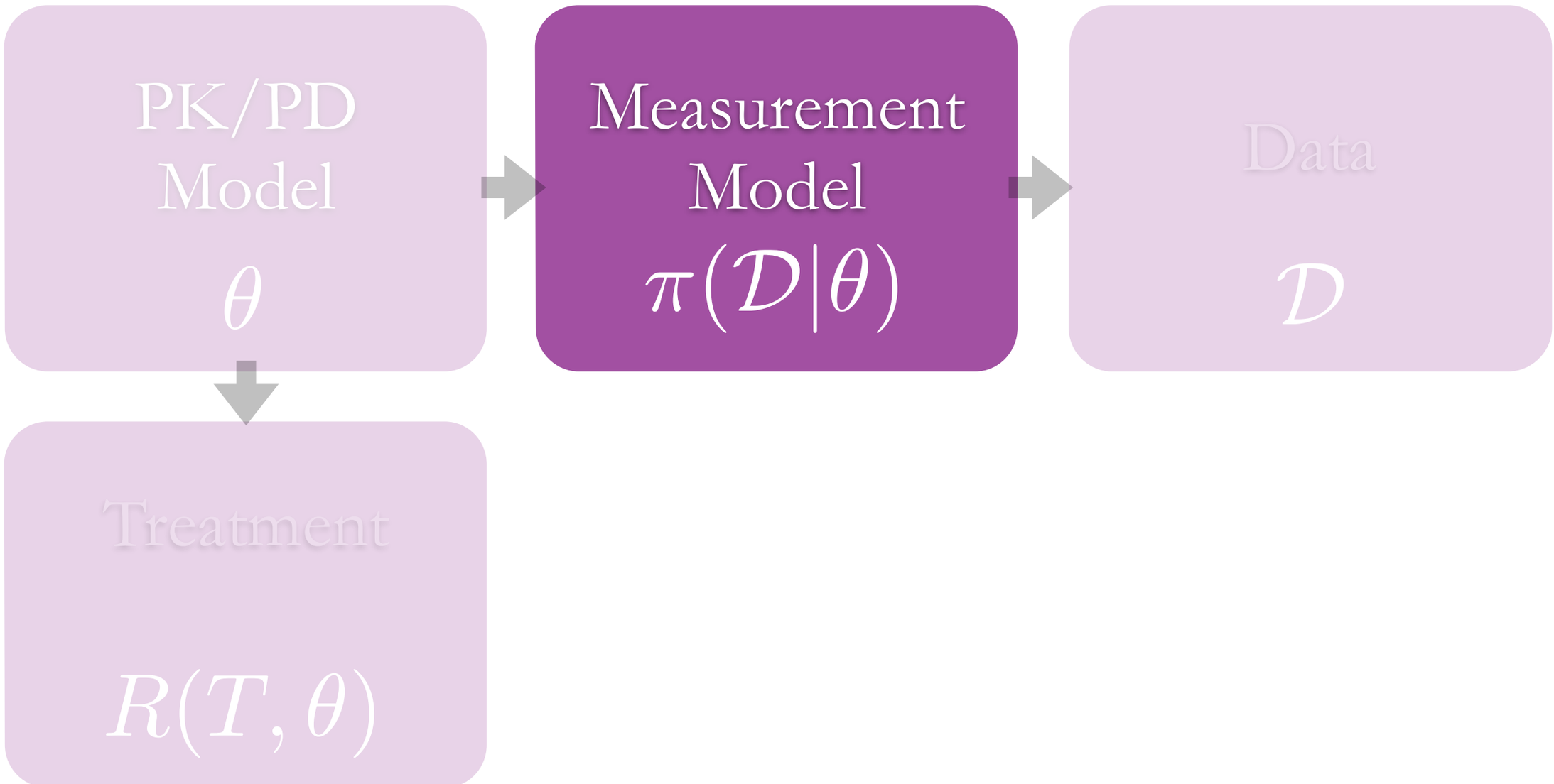
Analytic PK/PD models, for example, are exactly solvable.

$$\frac{\mathrm{d}C}{\mathrm{d}t} = -\frac{V_m}{V}\frac{C}{K_m + C}$$

These analytic models, however, can't capture many nonlinear effects in more realistic PK/PD systems.

$$\frac{\mathrm{d}C}{\mathrm{d}t} = -\frac{V_m}{V}\frac{C}{K_m + C} + \frac{k}{V}e^{-kt}$$

Similarly, analytic measurement models are straightforward to fit.



Similarly, analytic measurement models are straightforward to fit.

$$\widehat{C}_{t,n}(V, V_m) = \mathcal{N}(C(t, V, V_m), \sigma)$$

Similarly, analytic measurement models are straightforward to fit.

$$\widehat{C}_{t,n}(V, V_m) = \mathcal{N}\left(C(t, V, V_m)\,, \sigma\right)$$

$$C(t, V, V_m) \approx \frac{1}{N} \sum_{n=1}^{N} \hat{C}_{t,n}(V, V_m)$$

Similarly, analytic measurement models are straightforward to fit.

$$\widehat{C}_{t,n}(V, V_m) = \mathcal{N}\left(C(t, V, V_m), \sigma\right)$$

$$C(t, V, V_m) \approx \frac{1}{N} \sum_{n=1}^{N} \hat{C}_{t,n}(V, V_m)$$

$$V, V_m \approx C^{-1}\left(\frac{1}{N} \sum_{n=1}^{N} \hat{C}_{t,n}(V, V_m)\right)$$

But for population data analytic measurement models either introduce bias or suffer from large uncertainties.

$$\frac{\mathrm{d}C}{\mathrm{d}t} = -\frac{V_m}{V}\frac{C}{K_m + C}$$

But for population data analytic measurement models either introduce bias or suffer from large uncertainties.

$$\frac{\mathrm{d}C}{\mathrm{d}t} = -\frac{V_m^n}{V^n} \frac{C}{K_m + C}$$

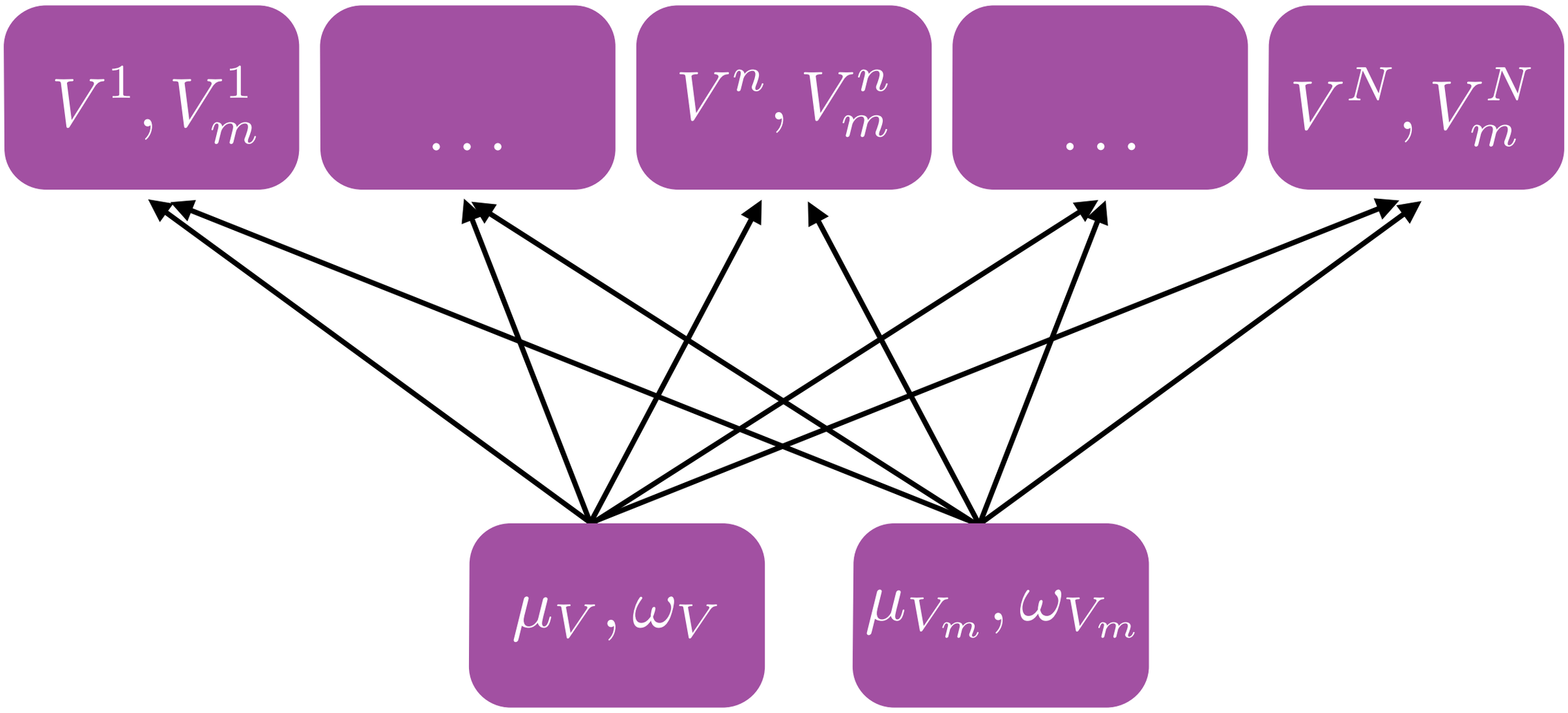
But for population data analytic measurement models either introduce bias or suffer from large uncertainties.

$$\frac{\mathrm{d}C}{\mathrm{d}t} = -\frac{V_m^n}{V^n}\frac{C}{K_m + C}$$

But for population data analytic measurement models either introduce bias or suffer from large uncertainties.

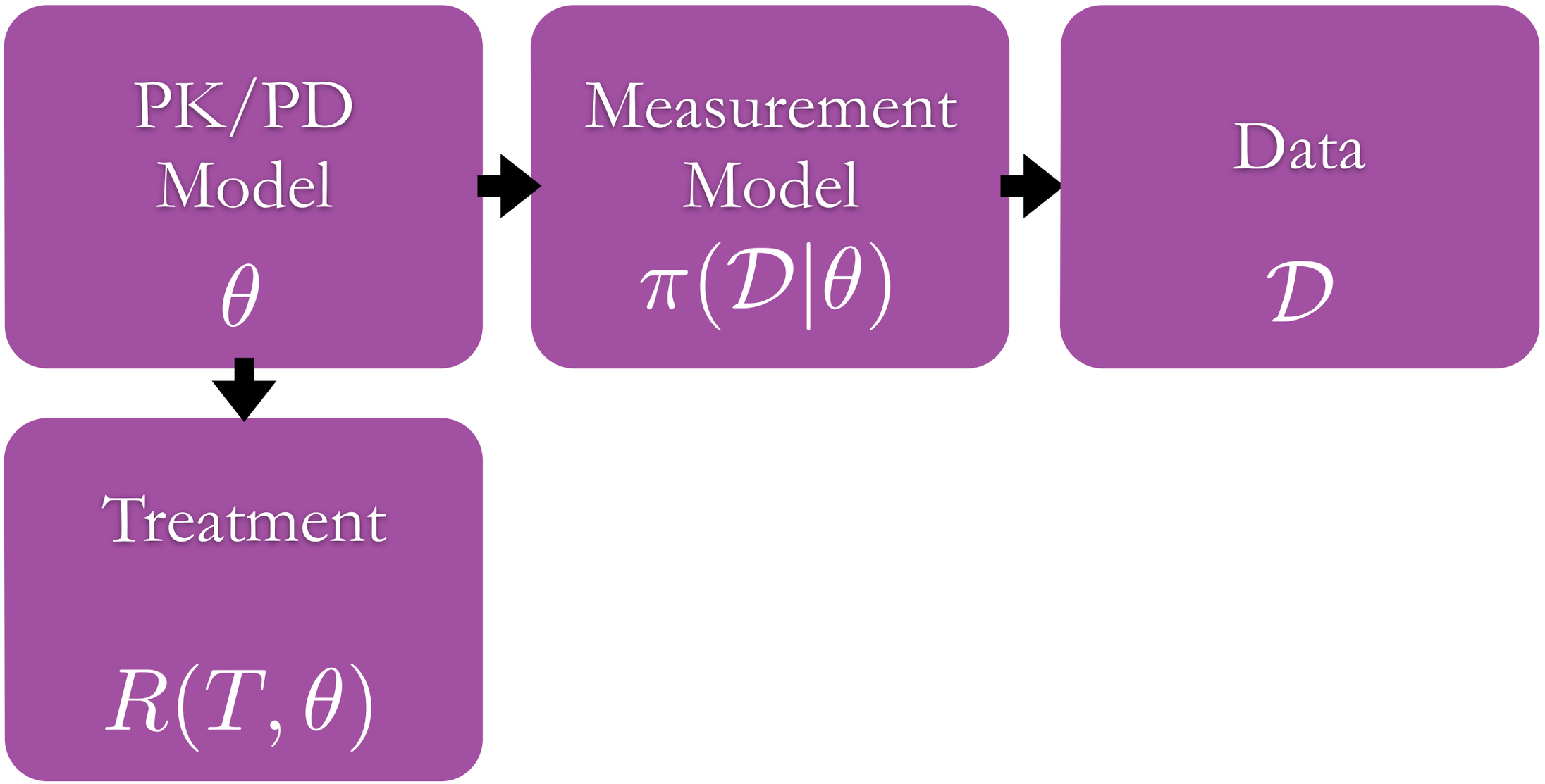
V^1, V_m^1 | ... | V^n, V_m^n | ... | V^N, V_m^N

If we want to maximize the utility of the data then we need non-analytic measurement models.



$$V^n \sim \log \mathcal{N}(\mu_V, \omega_V) \qquad V_m^n \sim \log \mathcal{N}(\mu_{V_m}, \omega_{V_m})$$

Non-analytic models are practically difficult because they are computationally demanding to fit.



Recall that in Bayesian inference our complete model is specified with a posterior distribution.

Recall that in Bayesian inference our complete model is specified with a posterior distribution.

$$\pi(\theta)$$

Recall that in Bayesian inference our complete model is specified with a posterior distribution.

$$\pi(\mathcal{D}|\theta)\,\pi(\theta)$$

Recall that in Bayesian inference our complete model is specified with a posterior distribution.

$$\pi(\theta|\mathcal{D}) \propto \pi(\mathcal{D}|\theta)\,\pi(\theta)$$

And all well-posed statistical manipulations are expectations with respect to the posterior.

$$\mathbb{E}[f] = \int \mathrm{d}\theta \, \pi(\theta|\mathcal{D}) \, f(\theta)$$

Expectations, however, are computationally demanding when the model is complex.

$$\mathbb{E}[f] = \int \mathrm{d}\theta \, \pi(\theta|\mathcal{D}) \, f(\theta)$$

The problem is that probability concentrates on a nonlinear surface called the *typical set*.

In order to estimate expectations we can use Markov chain Monte Carlo to find and explore the typical set.

In order to estimate expectations we can use Markov chain Monte Carlo to find and explore the typical set.

Given the complexity of the PK/PD model, this exploration has to be extremely efficient to be practical.

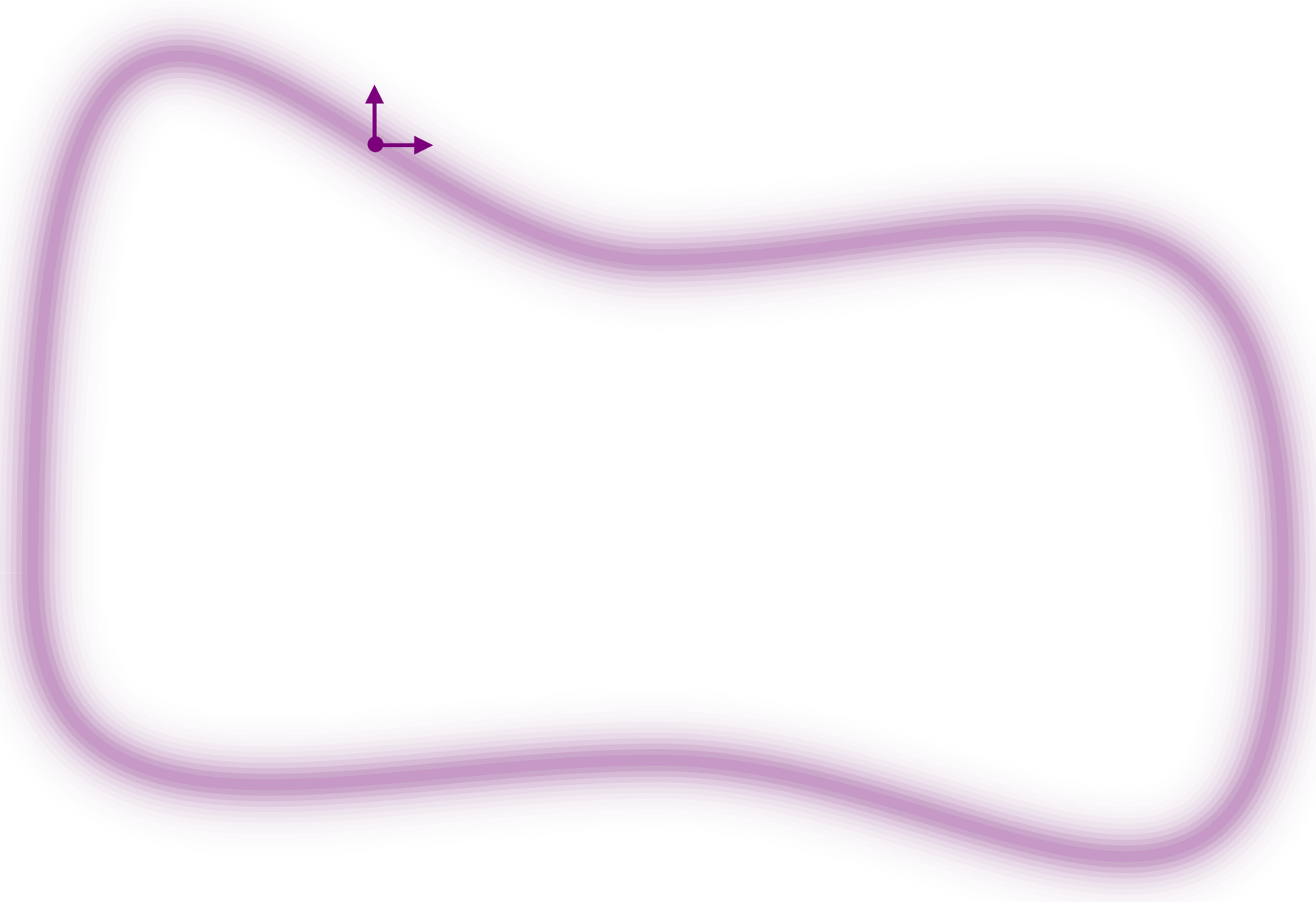
Random Walk Metropolis explores with
an ineffective “guided” diffusion.

Random Walk Metropolis explores with
an ineffective “guided” diffusion.

Random Walk Metropolis explores with
an ineffective “guided” diffusion.

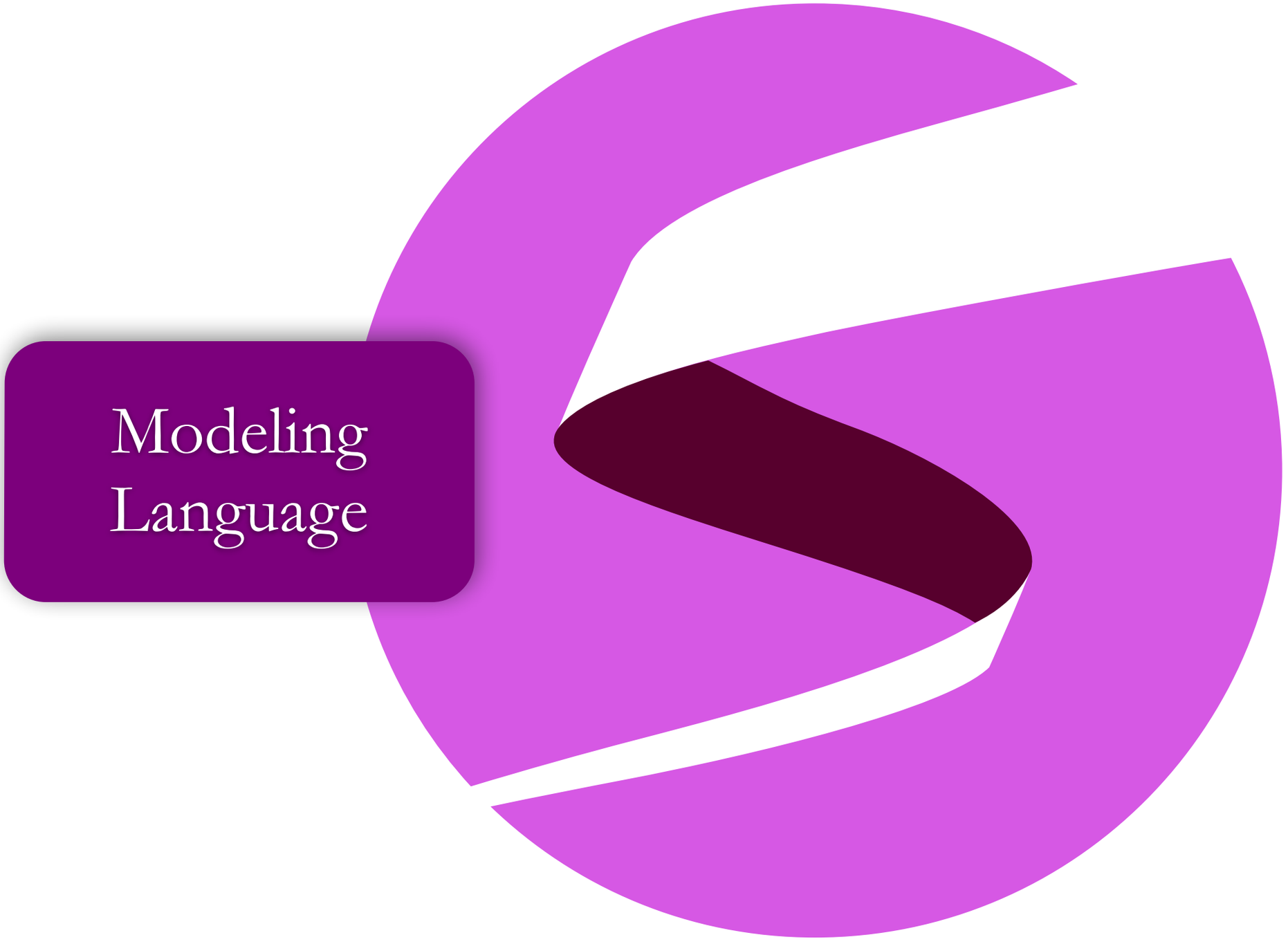
The Gibbs sampler updates each parameter
one-at-a-time to no greater success.

The Gibbs sampler updates each parameter one-at-a-time to no greater success.

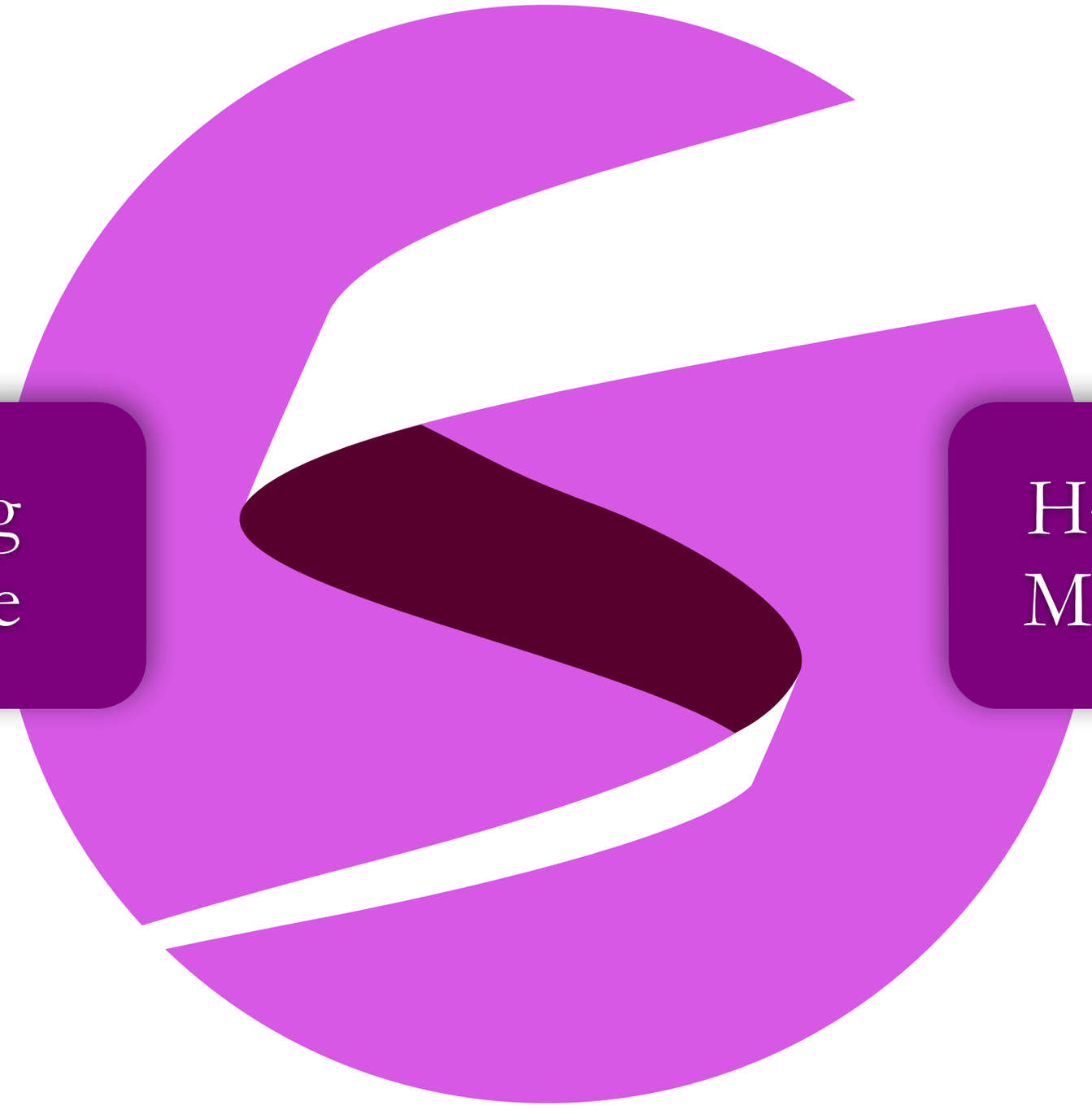


In order to explore efficiently
we need *coherent* exploration.

In order to explore efficiently
we need *coherent* exploration.


Modeling
Language

Modeling Language



Hamiltonian Monte Carlo

Modeling Language

Hamiltonian Monte Carlo

A strongly typed modeling language allows users to specify complex models with minimal effort.

```
data {
  int<lower=1> N;
  real x[N];
}
transformed data {
  vector[N] mu;
  cov_matrix[N] Sigma;
  for (i in 1:N)
    mu[i] <- 0;
  for (i in 1:N)
    for (j in 1:N)
      Sigma[i,j] <- exp(-pow(x[i] - x[j],2))
                    + if_else(i==j, 0.1, 0.0);
}
parameters {
  vector[N] y;
}
```

Stan not only admits the specification of a system of ordinary differential equations…

```
functions {
  real[] onecomp_mm_infusion(real t,
                             real[] y,
                             real[] theta,
                             real[] x_r,
                             int[] x_i) {

    real dydt[1];

    if (t < x_r[2])
      dydt[1] <- - theta[2] * y[1] / ( theta[1] * (theta[3] + y[1]) )
                 + x_r[1] / (theta[1] * x_r[2]);
    else
      dydt[1] <- - theta[2] * y[1] / ( theta[1] * (theta[3] + y[1]) );

    return dydt;

  }
}
```

But also high-performance integrators.

```
transformed parameters {
  …
  for (p in 1:N_patients) {
    V[p] <- exp(mu_V + eta_V[p] * omega_V);
    V_m[p] <- exp(mu_V_m + eta_V_m[p] * omega_V_m);

    theta[1] <- V[p];
    theta[2] <- V_m[p];
    theta[3] <- K_m;

    C <- integrate_ode(onecomp_mm_infusion, C0, t0, t, theta, x_r, x_i);
  …
}
```

Consequently, even complex statistical modeling of PK/PD systems is straightforward in Stan.

```
model {

  eta_V ~ normal(0, 1);
  mu_V ~ normal(log(5), 1);
  sigma_V ~ cauchy(0, 1);

  eta_V_m ~ normal(0, 1);
  mu_V_m ~ cauchy(0, 1);
  sigma_V_m ~ cauchy(0, 1);

  K_m ~ cauchy(0, 1);

  for (p in 1:N_patients)
    for (n in 1:N_t)
      C_hat[p, n] ~ lognormal(log(C[p, n]), 0.15);
}
```


Modeling
Language
Hamiltonian
Monte Carlo

Once a posterior has been specified, Stan implements Hamiltonian Monte Carlo to estimate expectations.

Let's consider a one-compartment Michaelis-Menten clearance model with a constant infusion.

$$\frac{\mathrm{d}C}{\mathrm{d}t} = -\frac{V_m}{V}\frac{C}{K_m + C} + I(t)$$

$$I(t) = \begin{cases} 0, & t \leq 0 \\ \frac{D}{VT}, & 0 < t < T \\ 0, & T \leq t \end{cases}$$

Data is simulated for ten patients, with each patient given their own V and V_m.

$$\frac{\mathrm{d}C^n}{\mathrm{d}t} = -\frac{V_m^n}{V^n}\frac{C^n}{K_m + C^n} + I(t)$$

$$\hat{C^n}(t) \sim \log \mathcal{N}(C^n(t), \sigma)$$

Data is simulated for ten patients, with each patient given their own V and V_m.

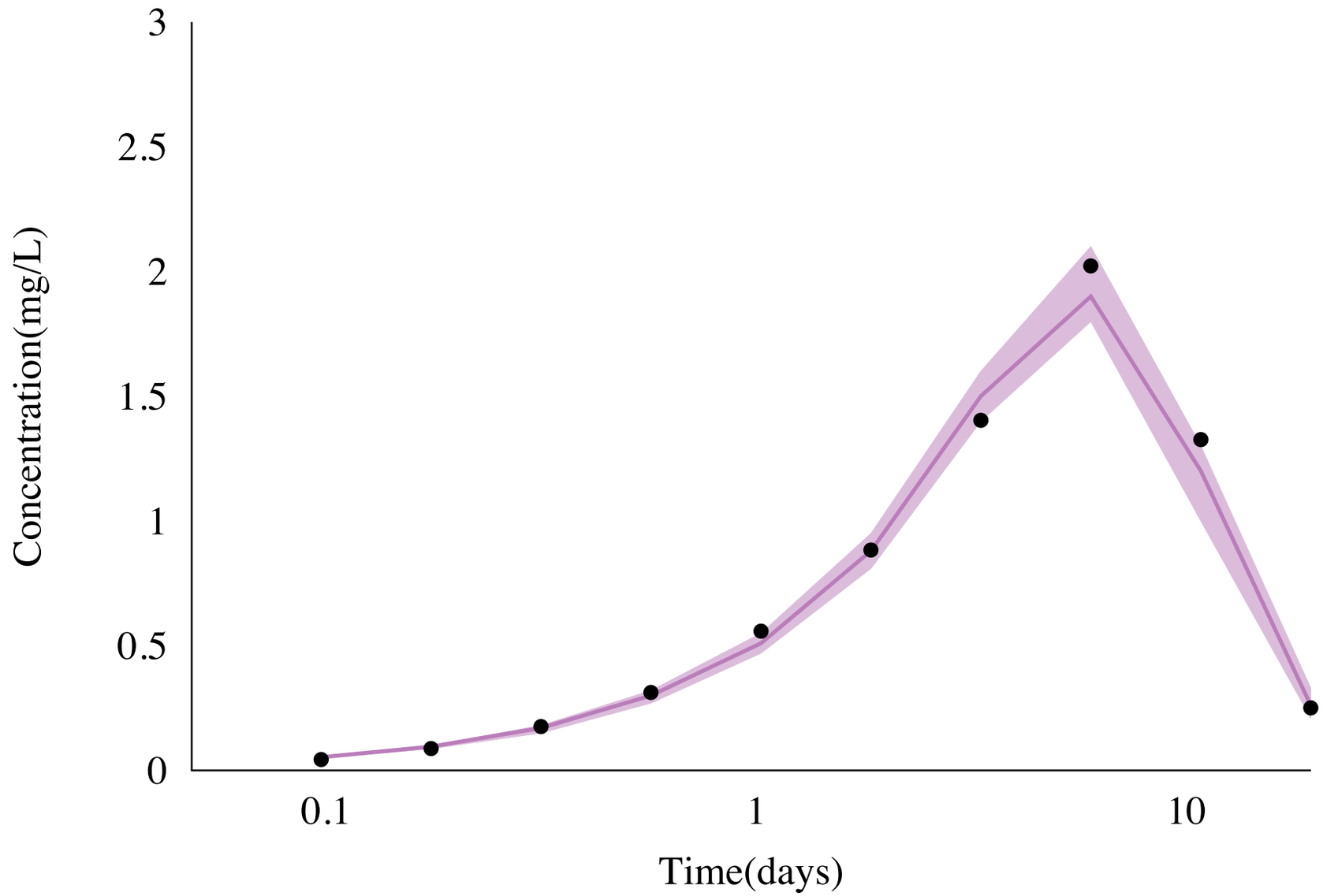
$$\frac{\mathrm{d}C^n}{\mathrm{d}t} = -\frac{V_m^n}{V^n}\frac{C^n}{K_m + C^n} + I(t)$$

$$\hat{C^n}(t) \sim \log\mathcal{N}(C^n(t), \sigma)$$

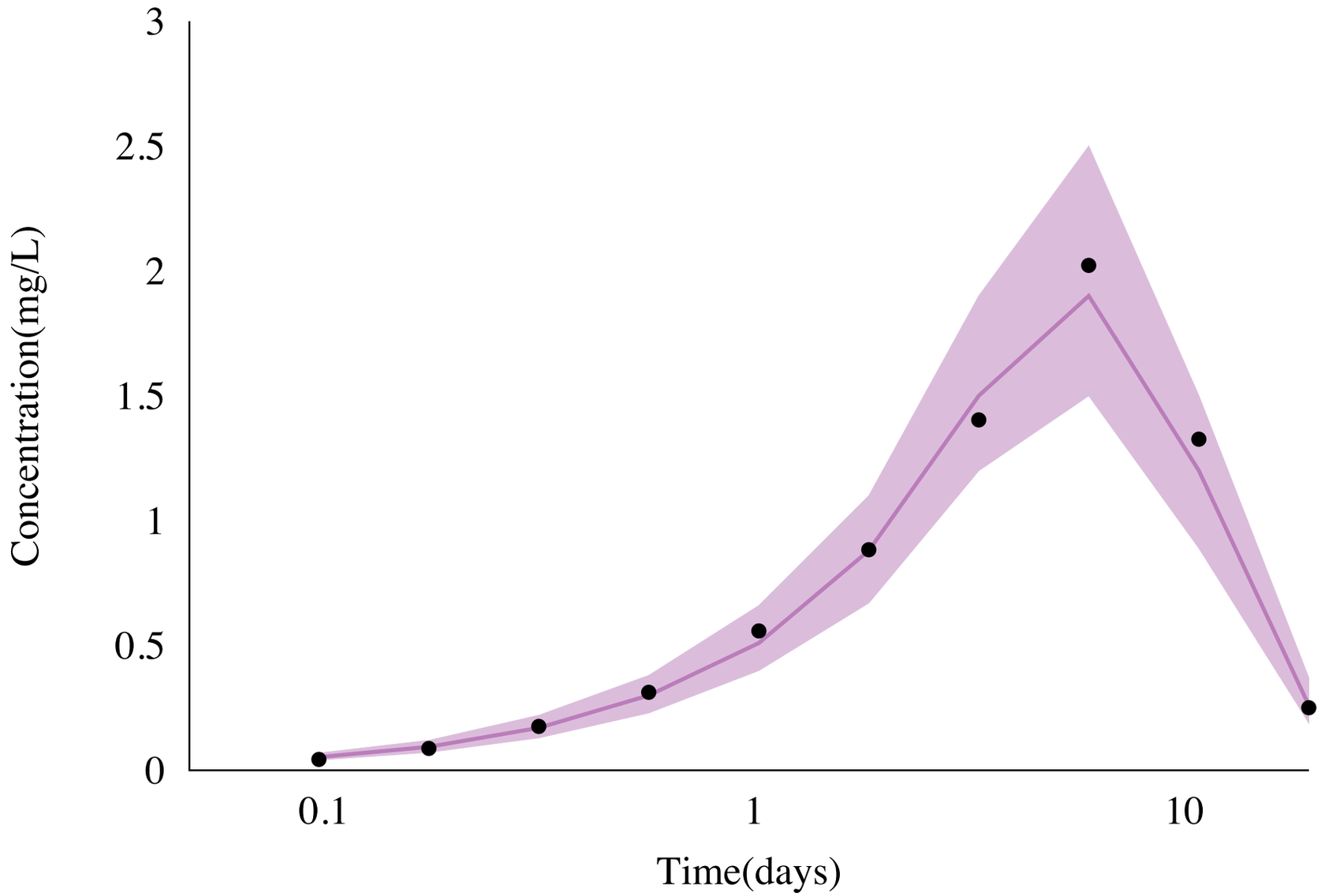
$$V^n \sim \log\mathcal{N}(\mu_V, \omega_V)$$

$$V_m^n \sim \log\mathcal{N}(\mu_{V_m}, \omega_{V_m})$$

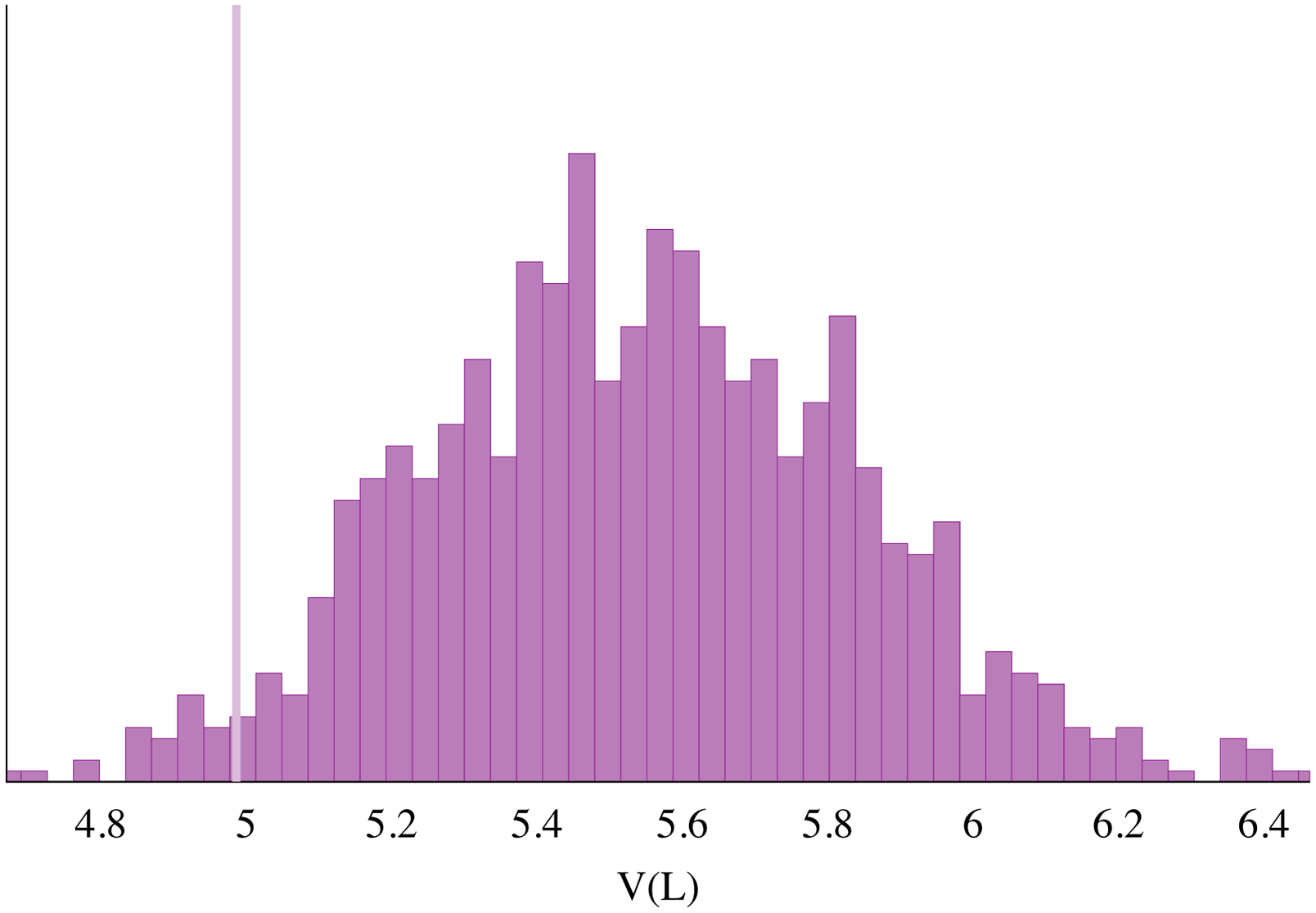
Within a few minutes Stan generates all of the samples we need to infer the individual patient concentrations.



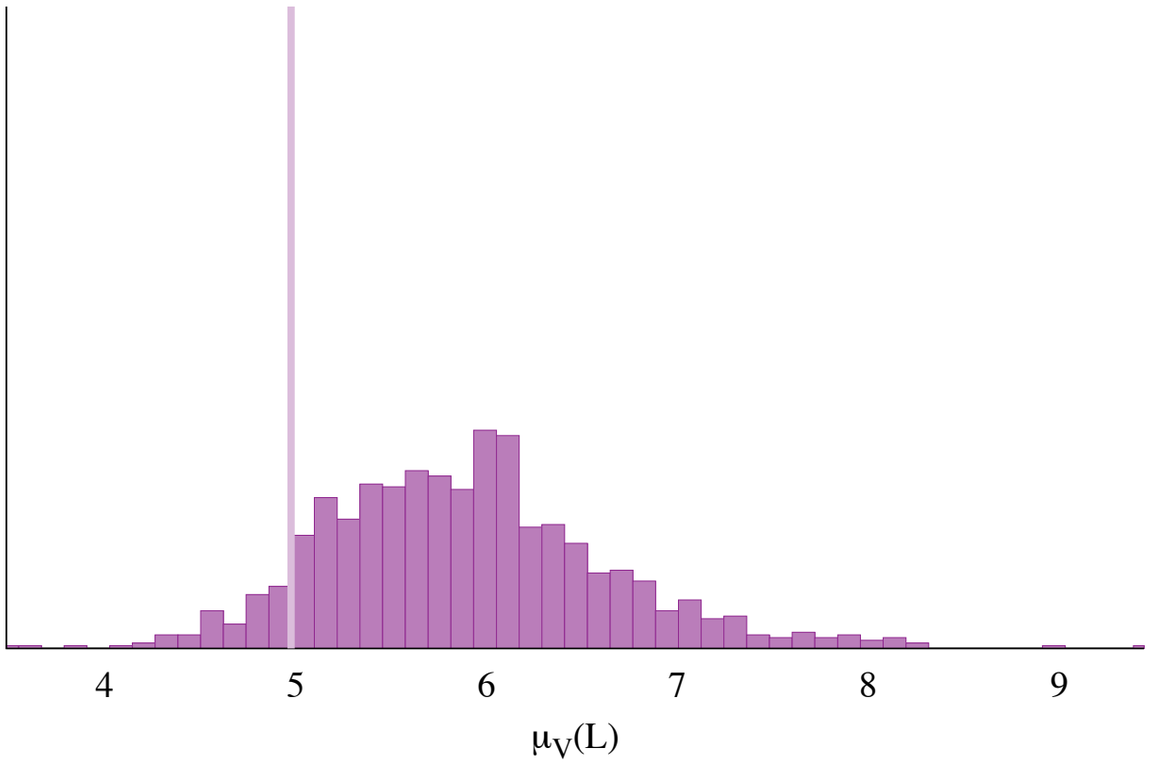
As well as simulated data for constructing posterior predictive checks.

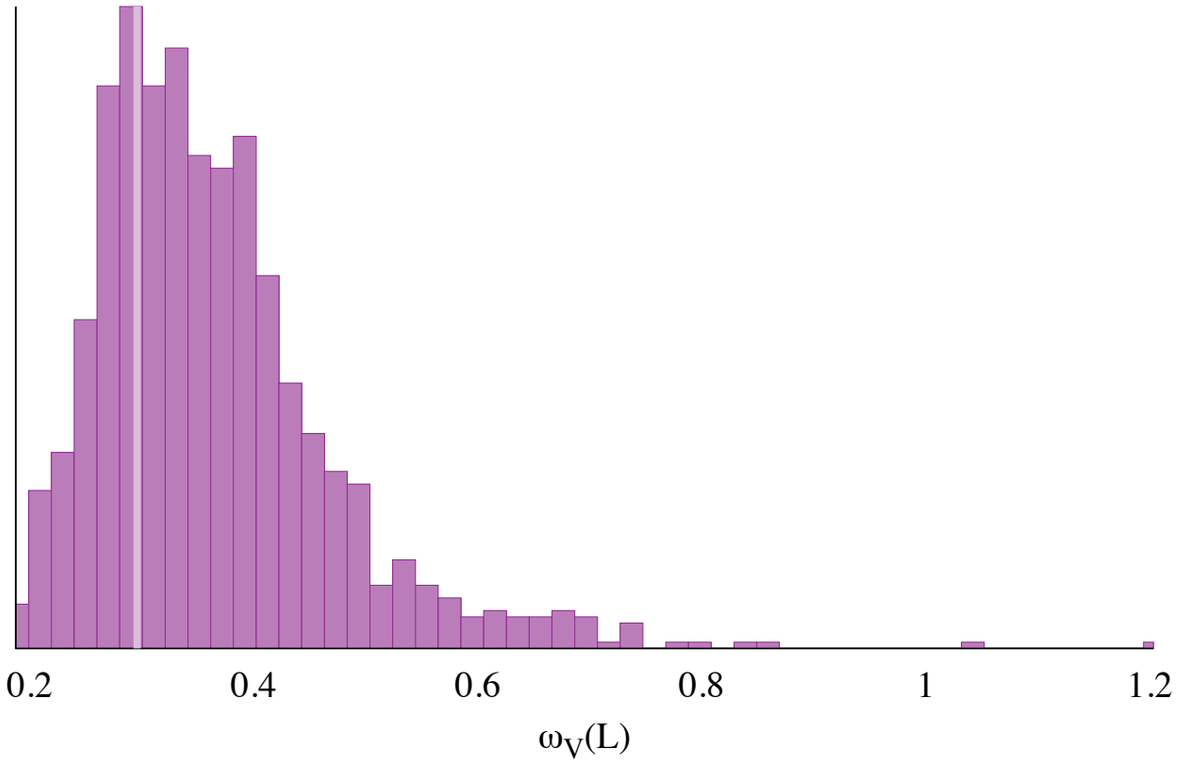


We can also analyze the system parameters for each individual patient.

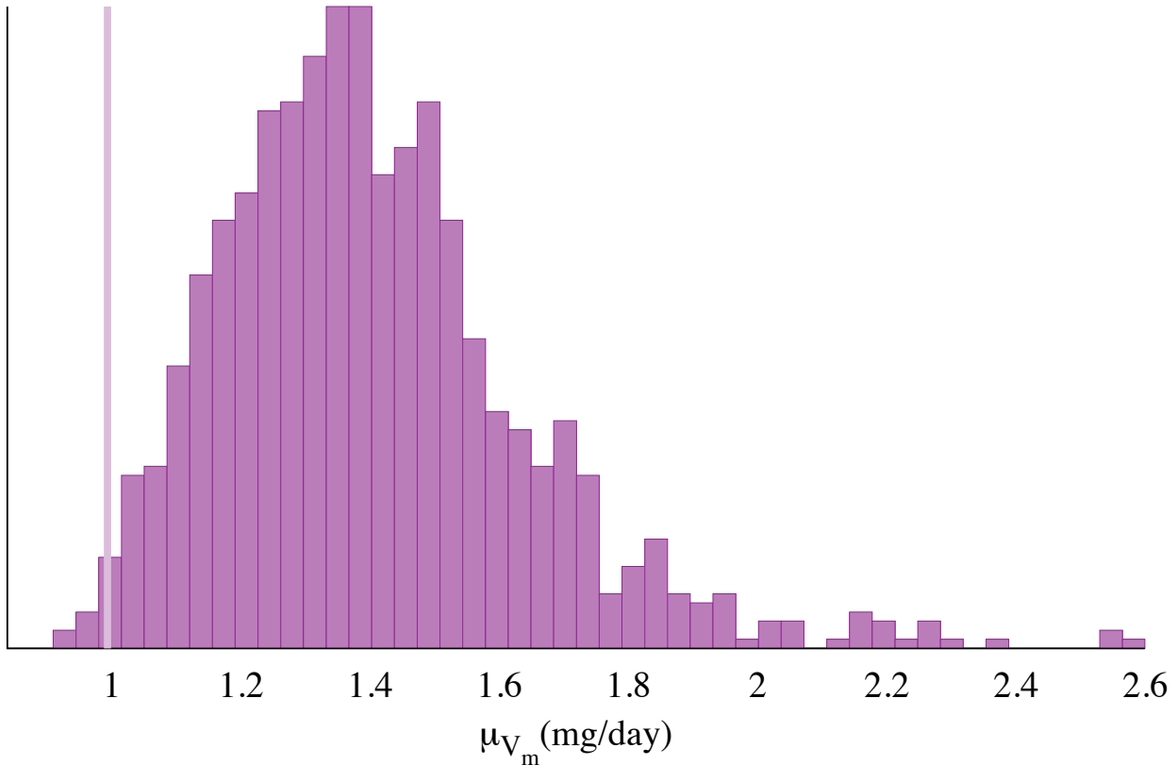


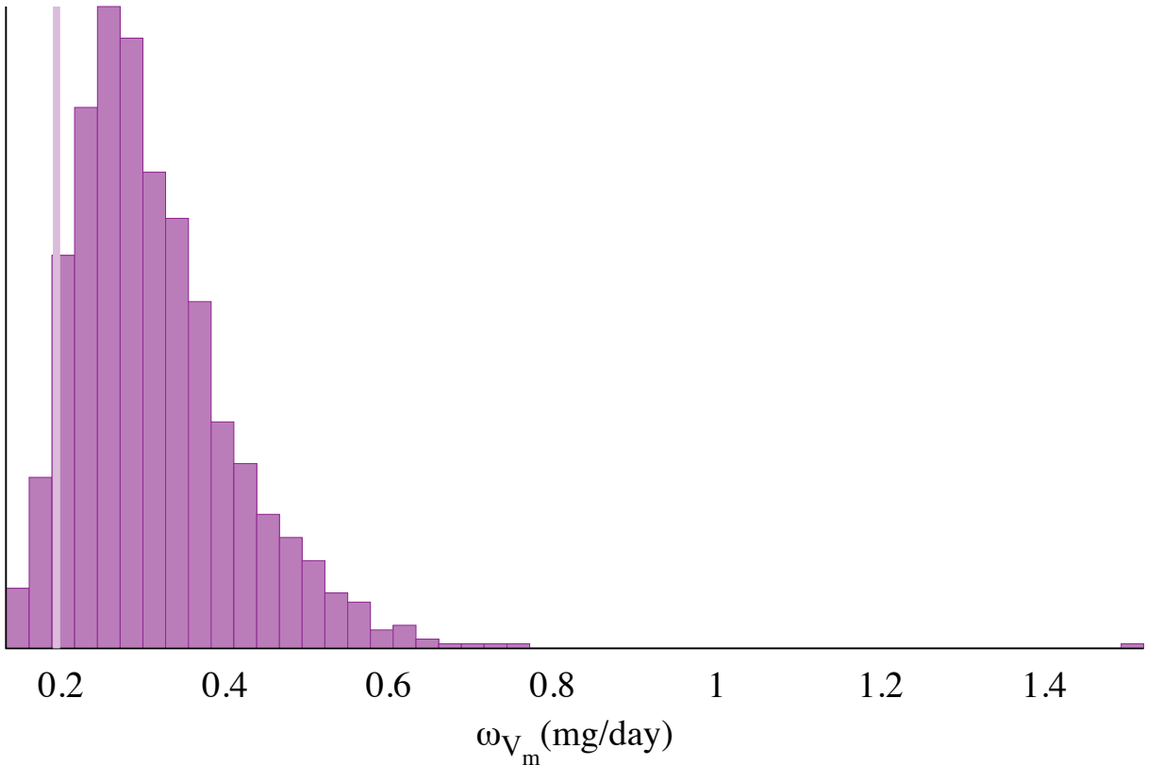
As well as the population parameters.





As well as the population parameters.





Stan

Stan is a probabilistic programming language implementing full Bayesian statistical inference with

- MCMC sampling (NUTS, HMC)

and penalized maximum likelihood estimation with

- Optimization (L-BFGS)

Stan is coded in C++ and runs on all major platforms (Linux, Mac, Windows).

Stan is freedom-respecting, open-source software (new BSD core, GPLv3 interfaces).

Interfaces

Download and getting started instructions, organized by interface:

- RStan v2.6.0 (R)
- PyStan v2.6.0 (Python)
- CmdStan v2.6.0 (shell, command-line terminal)
- MatlabStan (MATLAB)
- Stan.jl (Julia)

Manual & Examples