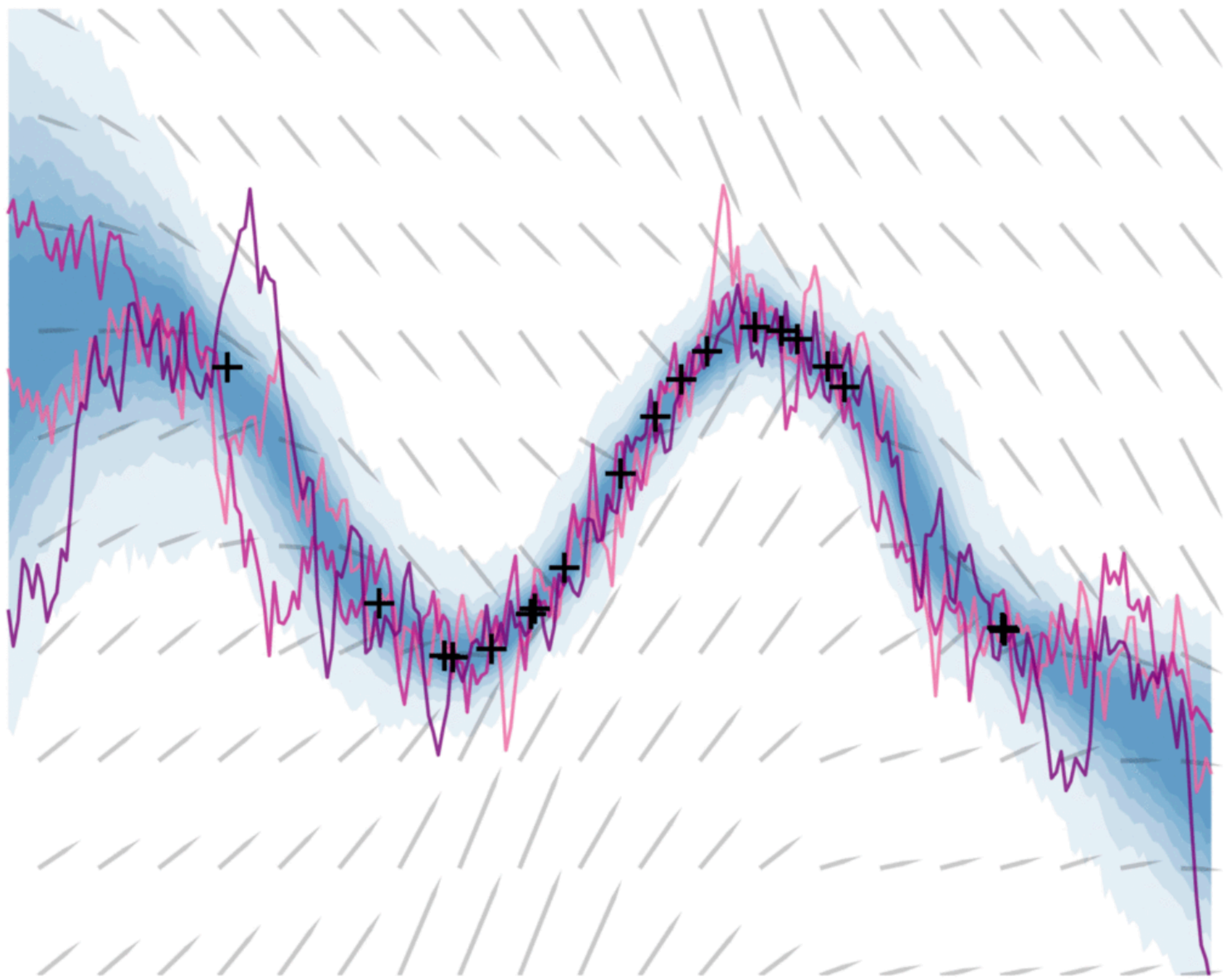
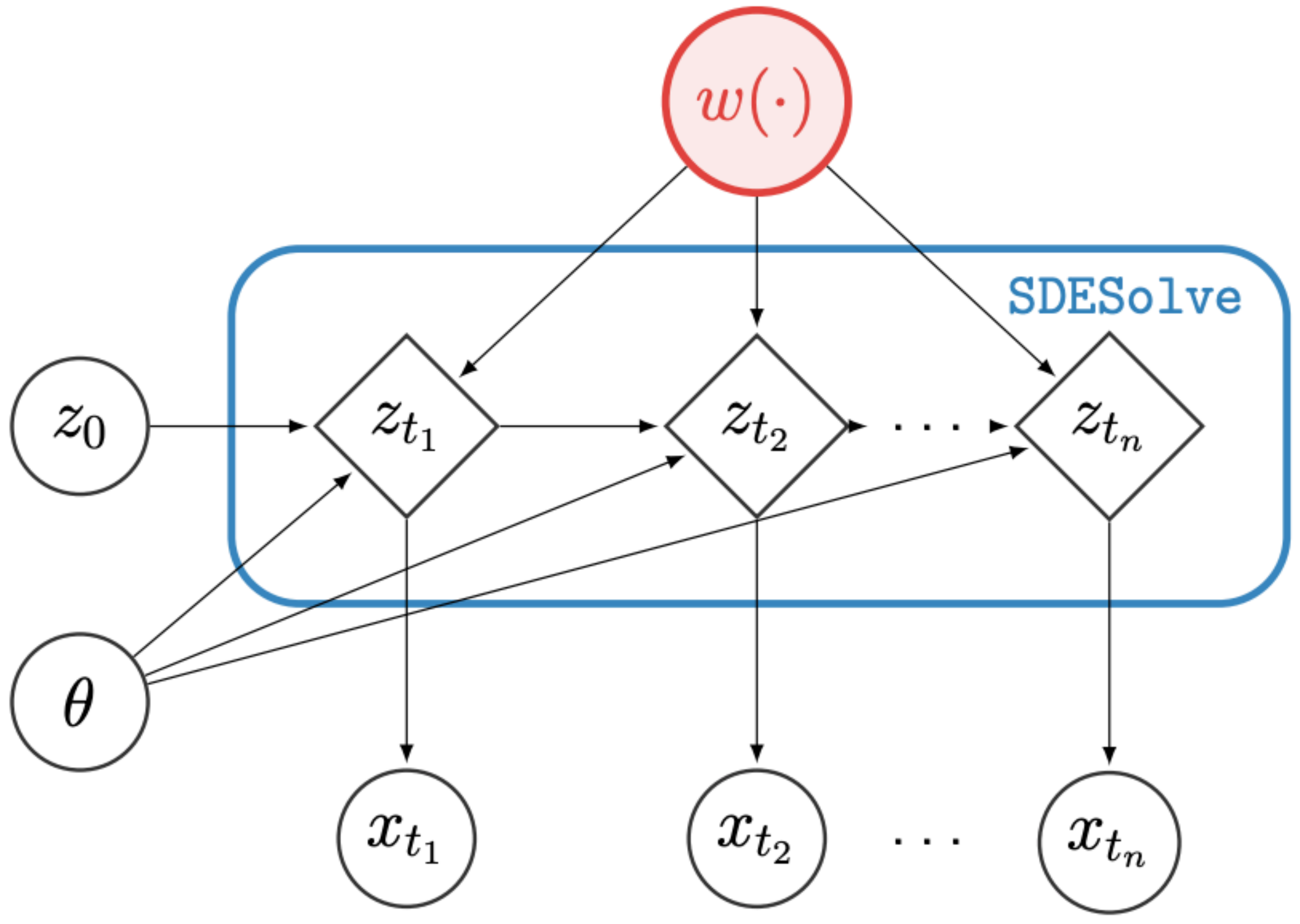
Latent Stochastic Differential Equations



Xuechen Li, Leonard Wong, Ricky Chen, David Duvenaud
University of Toronto, Vector Institute,
Google Brain Toronto

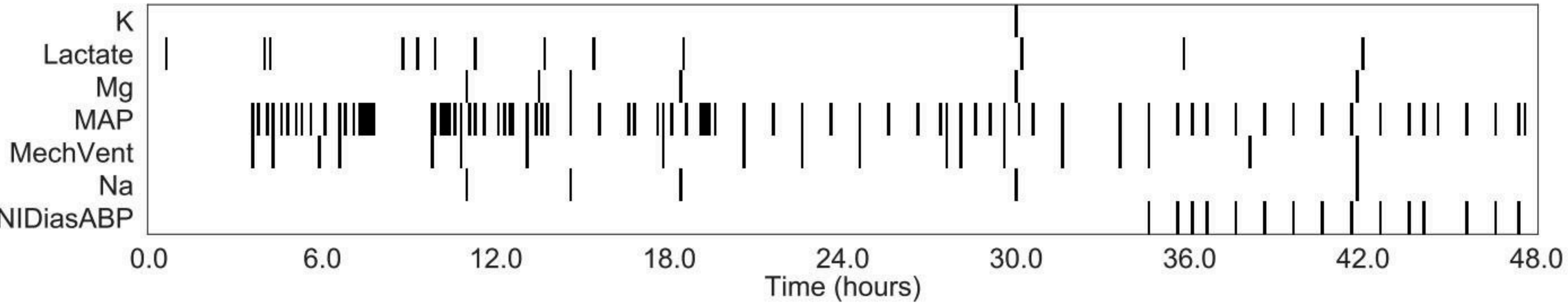
Summary

- New-ish model class for continuous-time generative models:
  - Neural net dynamics and likelihoods
  - Well-defined model, tractable marginal likelihood estimates
- 2020: Adjoint sensitivity method for SDEs.
  - O(1) training memory cost, adaptive compute
- 2021: Asymptotically-zero variance gradient estimator
- Exploring this model class: Time series, BNNs, multi-scale



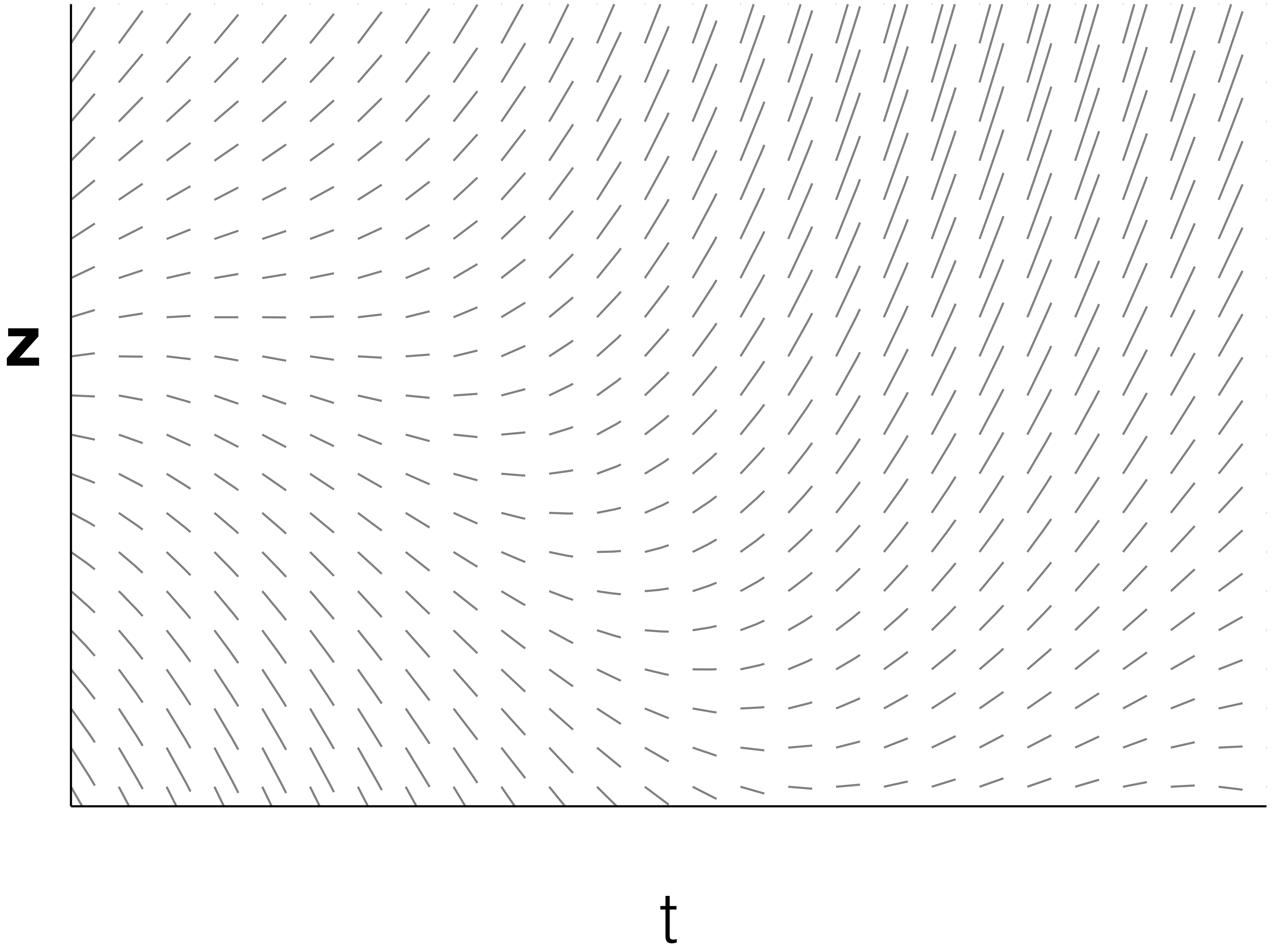
(a) Generation

Motivation: Irregularly-timed datasets



- Most patient data, business data irregularly sampled through time.
- Most large parametric models in ML are discrete time: RNNs, HMMs, DKFs
- How to handle these data without binning?

Ordinary Differential Equations



- Vector-valued **z** changes in time
- Time-derivative: $\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$
- Initial-value problem: given $\mathbf{z}(t_0)$, find:

$$\mathbf{z}(t_1) = \mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt$$

- Euler approximates with small steps:

$$\mathbf{z}(t + h) = \mathbf{z}(t) + hf(\mathbf{z}, t)$$

Autoregressive continuous-time

Standard RNN:

$$h'_i = h_{i-1}$$

$$h_i = \mathrm{RNNCell}(h'_i, x_i)$$

ODE-RNN:

$$h'_i = \mathrm{ODESolve}(f_\theta, h_{i-1}, (t_{i-1}, t_i))$$

$$h_i = \mathrm{RNNCell}(h'_i, x_i)$$

Limitations of RNN-based models

- Hidden state h represents model's belief about system's future, not the same thing as system state.
- Not a well-defined generative model.
- No explicit use of Bayes' rule, just makes predictions (but robust to mis-specification!)

$$h_i' = \text{ODESolve}(f_\theta, h_{i-1}, (t_{i-1}, t_i))$$

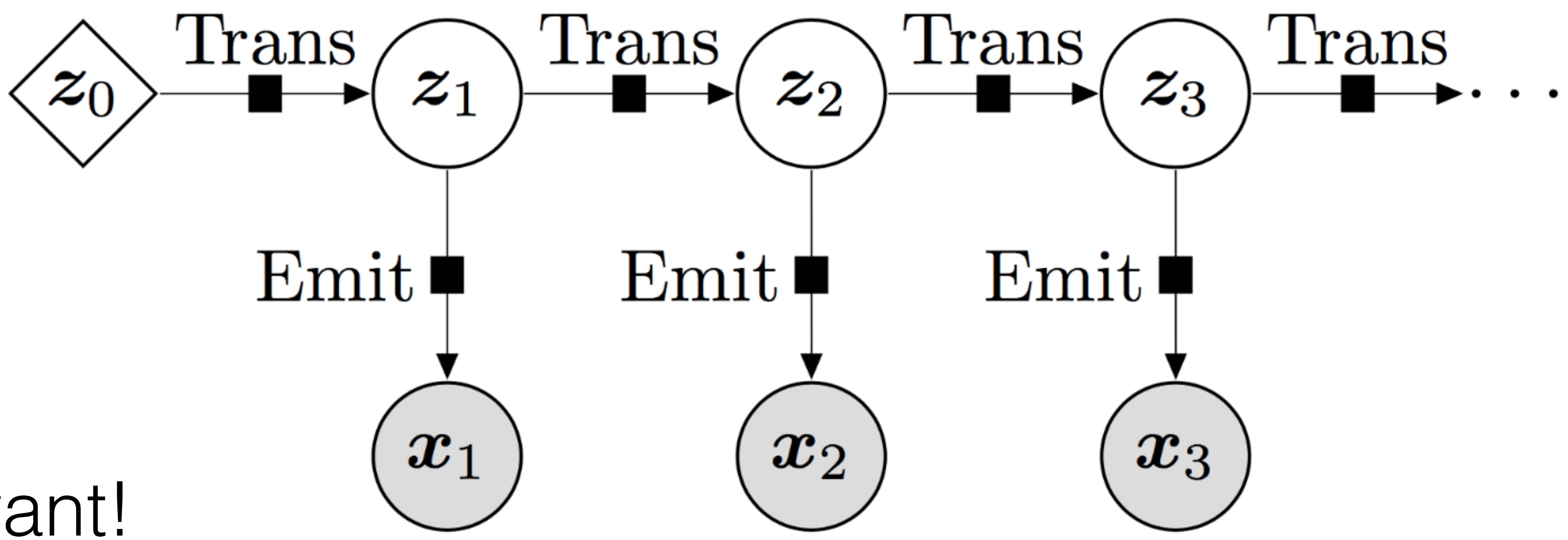
$$h_i = \text{RNNCell}(h_i', x_i)$$

Latent variable models

- Kalman Filters, Hidden Markov Models, Deep Markov Models
  - specify p(z), p(x|z)

$$p(x) = \int p(x \mid z) p(z) dz$$

- Can integrate out z however you want!
  - Recognition net can give approx. posterior

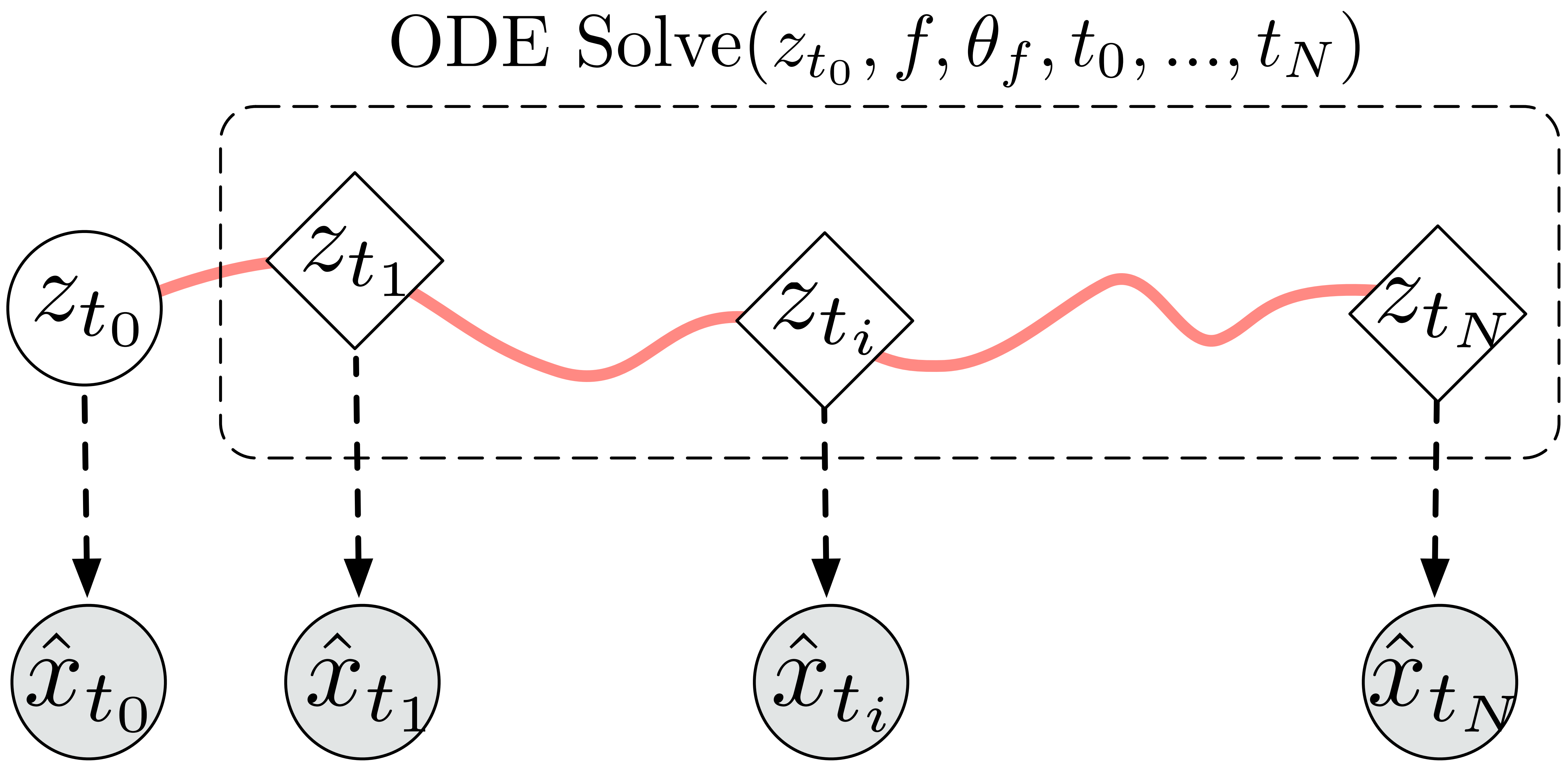


https://pyro.ai/examples/dmm.html

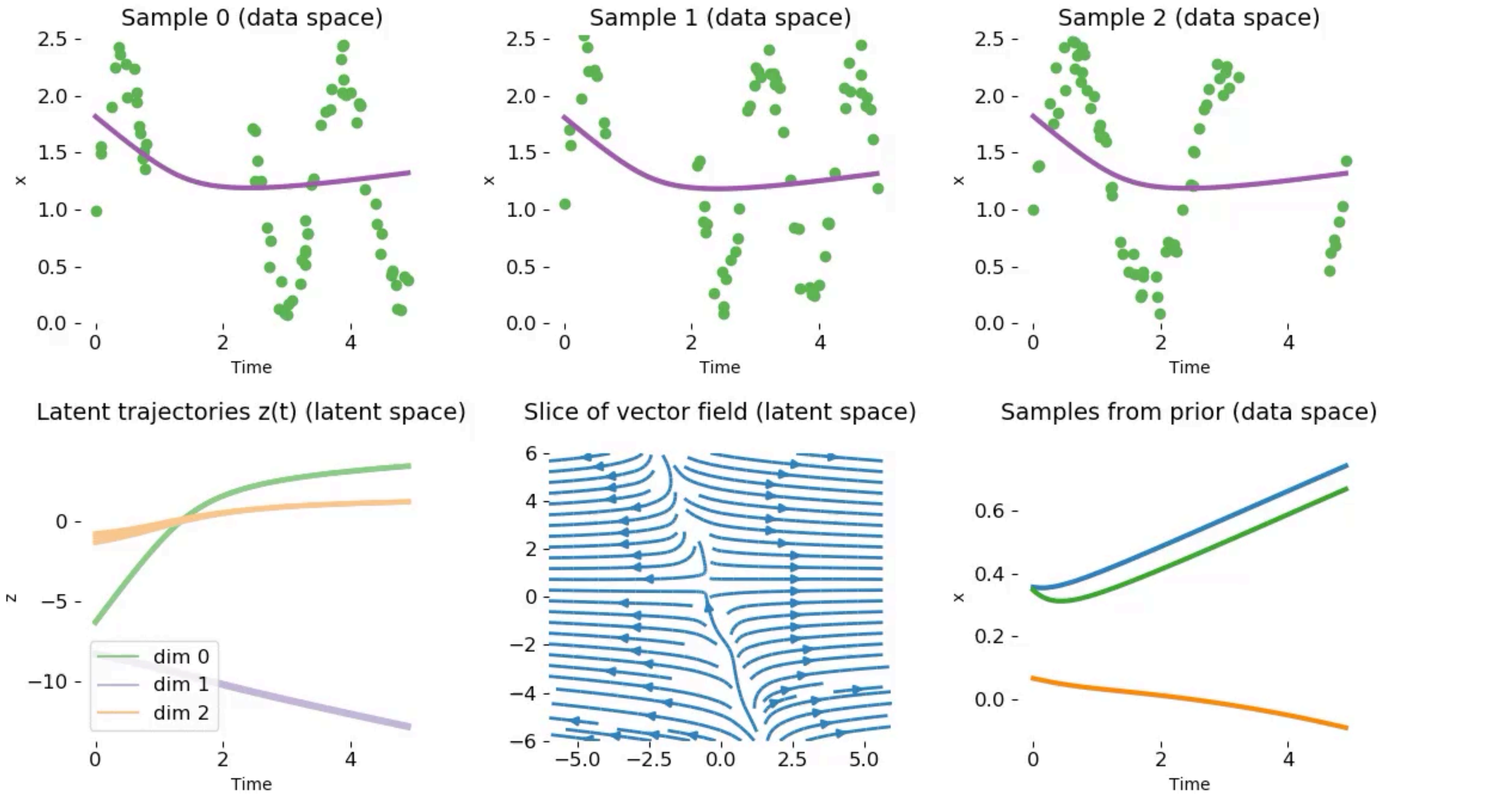
[Krishnan, Shalit & Sontag '15]

ODE latent-variable model

- z(t) is state of system at time t
- Need to approximate posterior p(z_t0 | x_t1…)
- Well-defined state at all times, dynamics separate from inference



$$\mathbf{z}_{t_0} \sim p(\mathbf{z}_{t_0})$$
$$\mathbf{z}_{t_1}, \mathbf{z}_{t_2}, \ldots, \mathbf{z}_{t_N} = \text{ODESolve}(\mathbf{z}_{t_0}, f, \theta_f, t_0, \ldots, t_N)$$
$$\text{each} \quad \mathbf{x}_{t_i} \sim p(\mathbf{x}|\mathbf{z}_{t_i}, \theta_{\mathbf{x}})$$


Sample 0 (data space)
Sample 1 (data space)
Sample 2 (data space)
Latent trajectories z(t) (latent space)
Slice of vector field (latent space)
Samples from prior (data space)
x
z
Time
dim 0
dim 1
dim 2

Physionet: Predictive accuracy

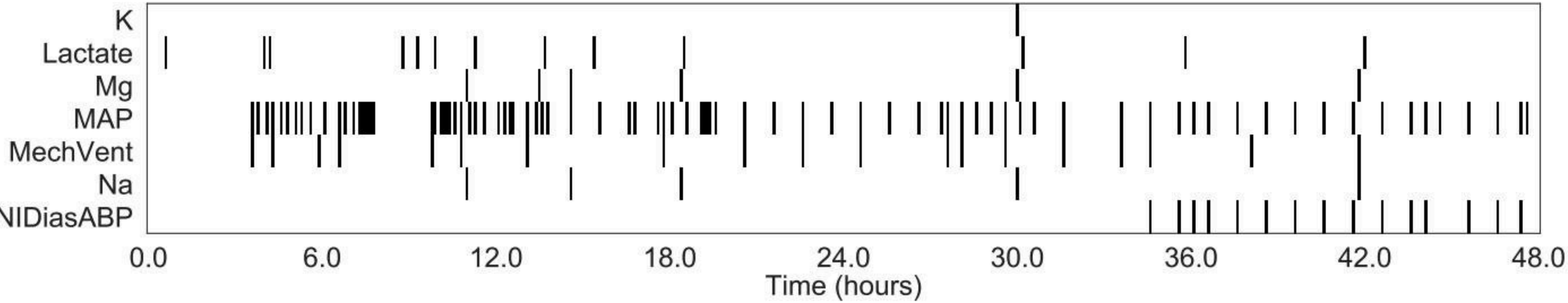


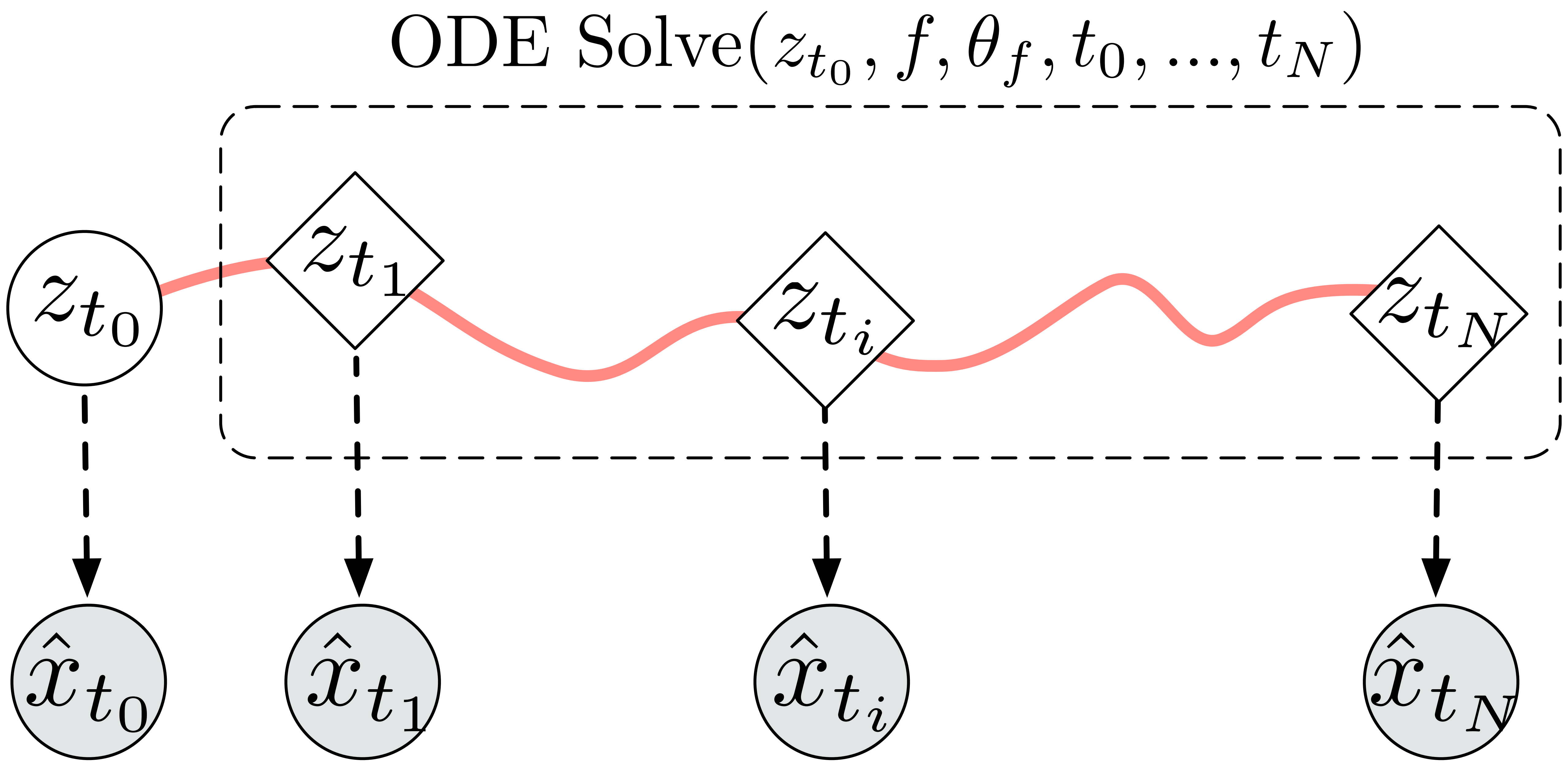
Table 4: Test MSE (mean $\pm$ std) on PhysioNet. **Autoregressive** models.

Model	Interp ($\times 10^{-3}$)
RNN Δ_t	3.520 ± 0.276
RNN-Impute	3.243 ± 0.275
RNN-Decay	3.215 ± 0.276
RNN GRU-D	3.384 ± 0.274
ODE-RNN (Ours)	$\mathbf{2.361} \pm \mathbf{0.086}$

Table 5: Test MSE (mean $\pm$ std) on PhysioNet. **Encoder-decoder** models.

Model	Interp ($\times 10^{-3}$)	Extrap ($\times 10^{-3}$)
RNN-VAE	5.930 ± 0.249	3.055 ± 0.145
Latent ODE (RNN enc.)	3.907 ± 0.252	3.162 ± 0.052
Latent ODE (ODE enc)	$\mathbf{2.118} \pm \mathbf{0.271}$	$\mathbf{2.231} \pm \mathbf{0.029}$
Latent ODE + Poisson	2.789 ± 0.771	$\mathbf{2.208} \pm \mathbf{0.050}$

Limitations of Latent ODEs



- Deterministic dynamics!
  - State size grows with sequence length
- Special time t0

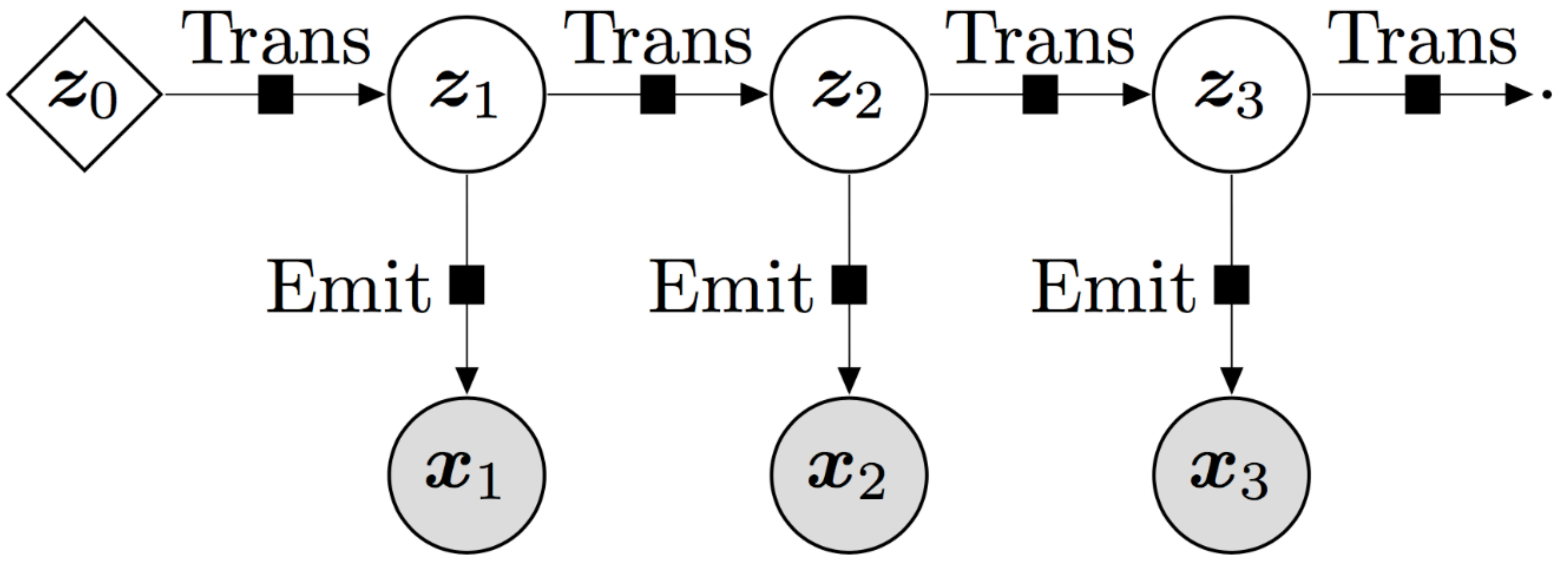
Stochastic transition dynamics

- Nonlinear latent variable with noise at each step:

$$z_{t+1} = z_t + f_\theta(z_t) + \epsilon$$

- Could add more steps between observations.

- Infinitesimal limit some sort of stochastic ODE…?



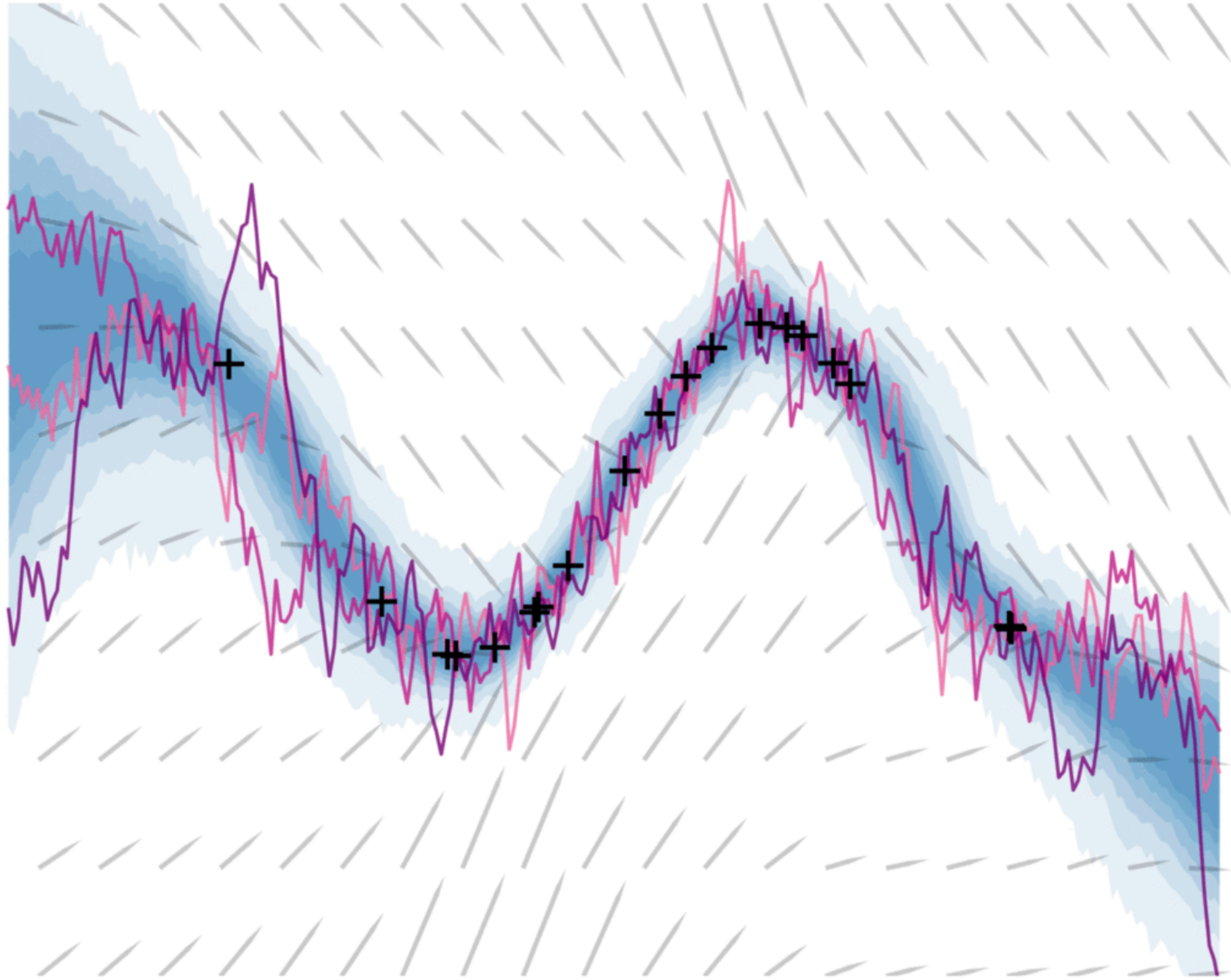
https://pyro.ai/examples/dmm.html

Stochastic Differential Equations

$$\frac{dz}{dt} = f(z(t)) \;+ \text{“}\epsilon\text{”}$$

$$dz = f(z(t))dt \;+\; \sigma(z(t))dB(t)$$

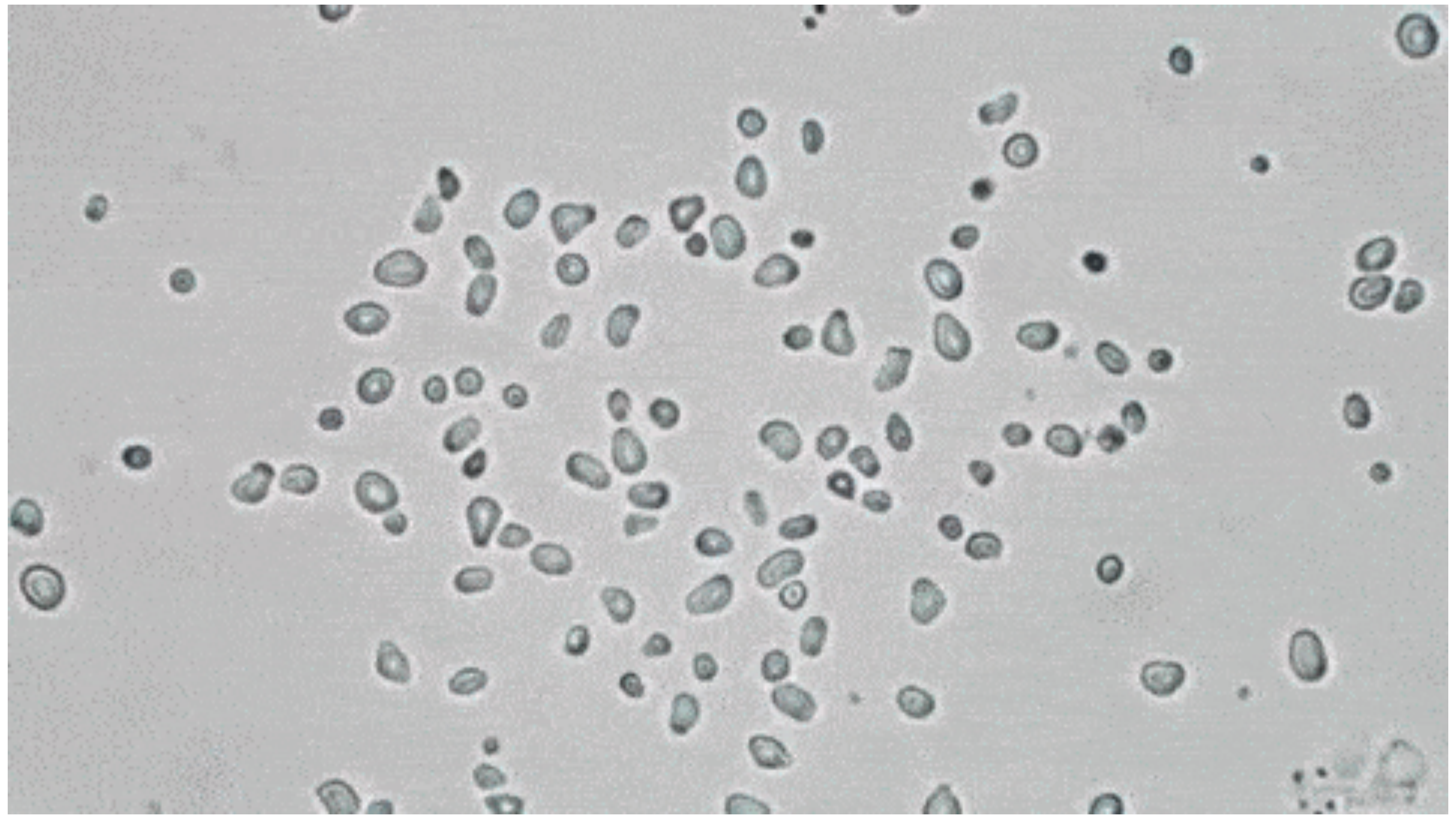
Drift Diffusion



- Implicit distribution over functions.

What are SDEs good for?

- natural fit for many small, unobserved interactions:
  - motion of molecules in a liquid
  - allele frequencies in a gene pool
  - prices in a market



- Interactions don’t need to be Gaussian if CLT kicks in
- Let’s put neural nets in SDE dynamics and fit to data!

$$dz = f_\theta(z(t))dt + \sigma_\theta(z(t))dB(t)$$

How to fit ODE params?

$$L(\theta) = L\left(\int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right)$$

$$\frac{\partial L}{\partial \theta} = ?$$

- Don't backprop through solver: High memory cost, numerical error
- Alexey Radul: Approximate the derivative, don't differentiate the approximation!

Continuous-time Backpropagation

Standard Backprop:

- Can build adjoint dynamics with autodiff, compute gradients with second ODE solve:

```
def f_aug([z, a, d], t):
    return [f, -a*df/dz, -a*df/dθ]

[z0, dL/dz(t0), dL/dθ] =
    ODESolve(f_aug, [z(t1), dL/dz(t1), 0], t1, t0)
```

$$\frac{\partial L}{\partial \mathbf{z}_t} = \frac{\partial L}{\partial \mathbf{z}_{t+1}} \frac{\partial f(\mathbf{z}_t, \theta)}{\partial \mathbf{z}_t}$$

$$\frac{\partial L}{\partial \theta} = \sum_t \frac{\partial L}{\partial \mathbf{z}_t} \frac{\partial f(\mathbf{z}_t, \theta)}{\partial \theta}$$

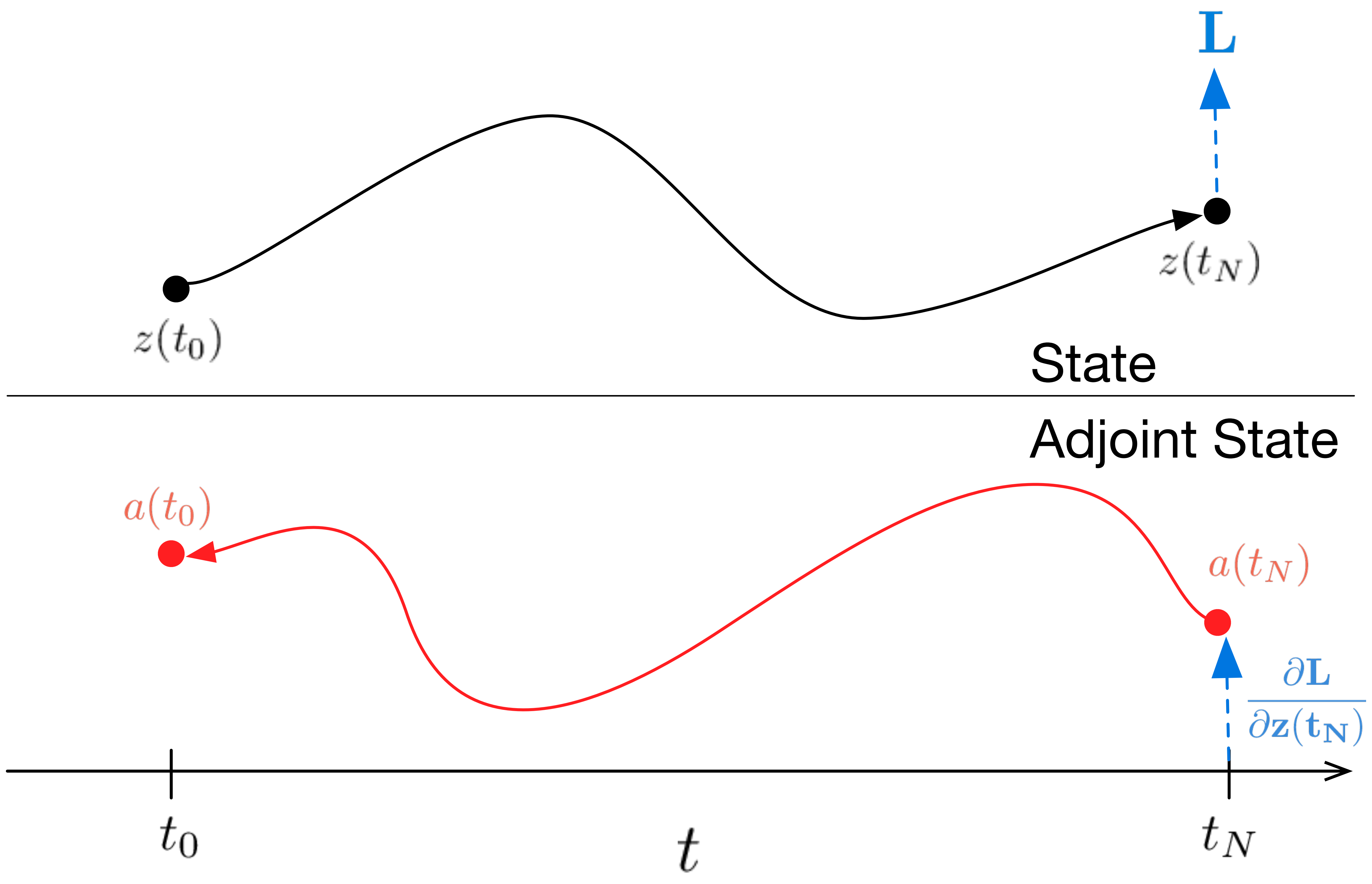
Adjoint sensitivities:
(Pontryagin et al., 1962):

$$\frac{\partial}{\partial t} \frac{\partial L}{\partial \mathbf{z}(t)} = \frac{\partial L}{\partial \mathbf{z}(t)} \frac{\partial f(\mathbf{z}(t), \theta)}{\partial \mathbf{z}}$$

$$\frac{\partial L}{\partial \theta} = \int_{t_1}^{t_0} \frac{\partial L}{\partial \mathbf{z}(t)} \frac{\partial f(\mathbf{z}(t), \theta)}{\partial \theta} dt$$

O(1) Memory Gradients

- No need to store activations, just run dynamics backwards from output.
- Easy to run ODE backwards, just run negate dynamics and time:
    - back_f(z, t) = -f(z, -t)



Algorithm 1 ODE Adjoint Sensitivity

Input: Parameters θ, start time t_0, stop time t_1, final state z_{t_1}, loss gradient $\partial \mathcal{L}/z_{t_1}$, dynamics $f(z, t, \theta)$.

def $\overline{f}([z_t, a_t, \cdot],\ t,\ \theta)$: ▷ Augmented dynamics
  $v = f(z_t, -t, \theta)$
  return $[-v,\ a_t \partial v/\partial z,\ a_t \partial v/\partial \theta\,]$

$$\begin{bmatrix} z_{t_0} \\ \partial \mathcal{L}/\partial z_{t_0} \\ \partial \mathcal{L}/\partial \theta \end{bmatrix} = \texttt{odeint}\left(\begin{bmatrix} z_{t_1} \\ \partial \mathcal{L}/\partial z_{t_1} \\ \mathbf{0}_p \end{bmatrix}, \overline{f}, -t_1, -t_0 \right)$$

return $\partial \mathcal{L}/\partial z_{t_0}, \partial \mathcal{L}/\partial \theta$

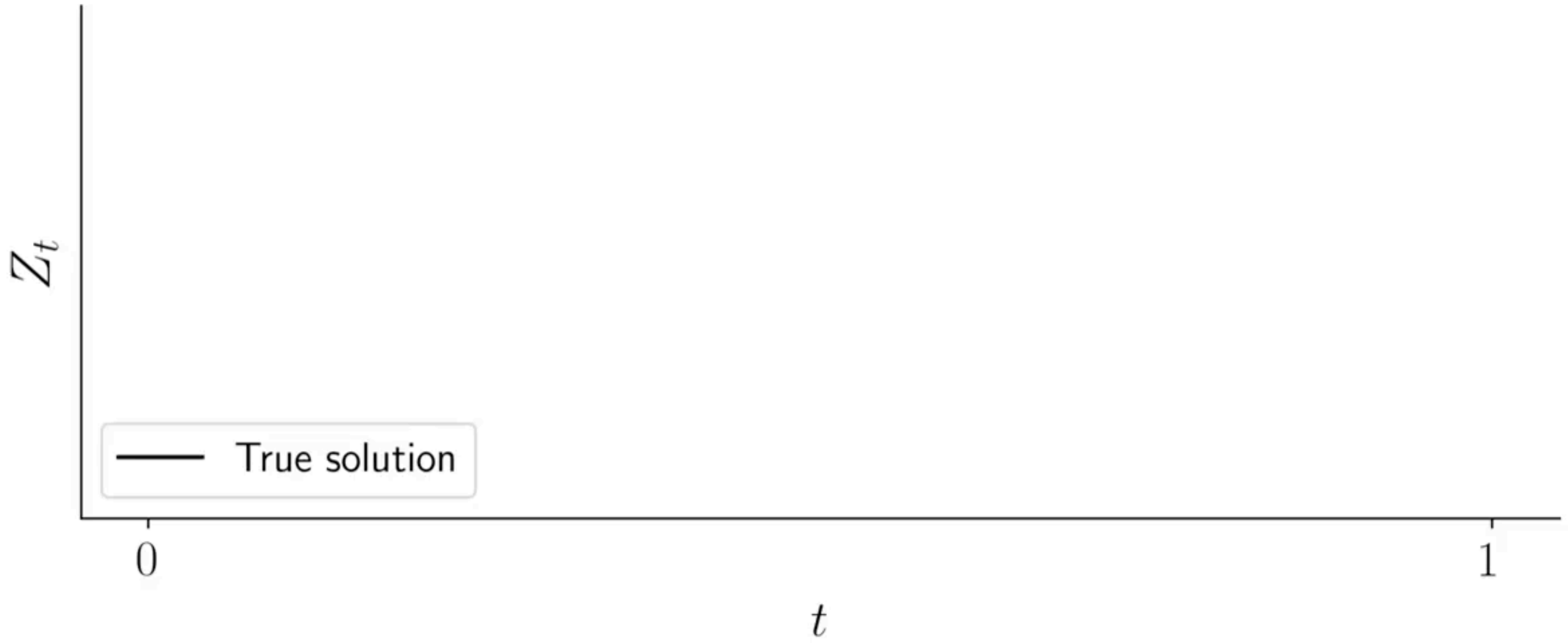
- Final algorithm for ODE grads: Solve one big augmented system backwards in time.
- Mostly worked out by Pontryagin (1961)

Why not repeat same trick?

- If an SDE is just “an ODE with noise”, why not use same adjoint method?
- "Unfortunately, there is no straightforward way to port this construction to SDEs." - Tzen & Raginsky (2019)
- (alternative: Rough path theory. ask later)

What is “running an SDE backwards”?

- Me: Let’s just slap negative signs on everything and hope for the best
- Xuechen Li and Leonard Wong: What does that even mean?
    - Much later: Nvm that’s correct.
- Builds on Kunita (2019)



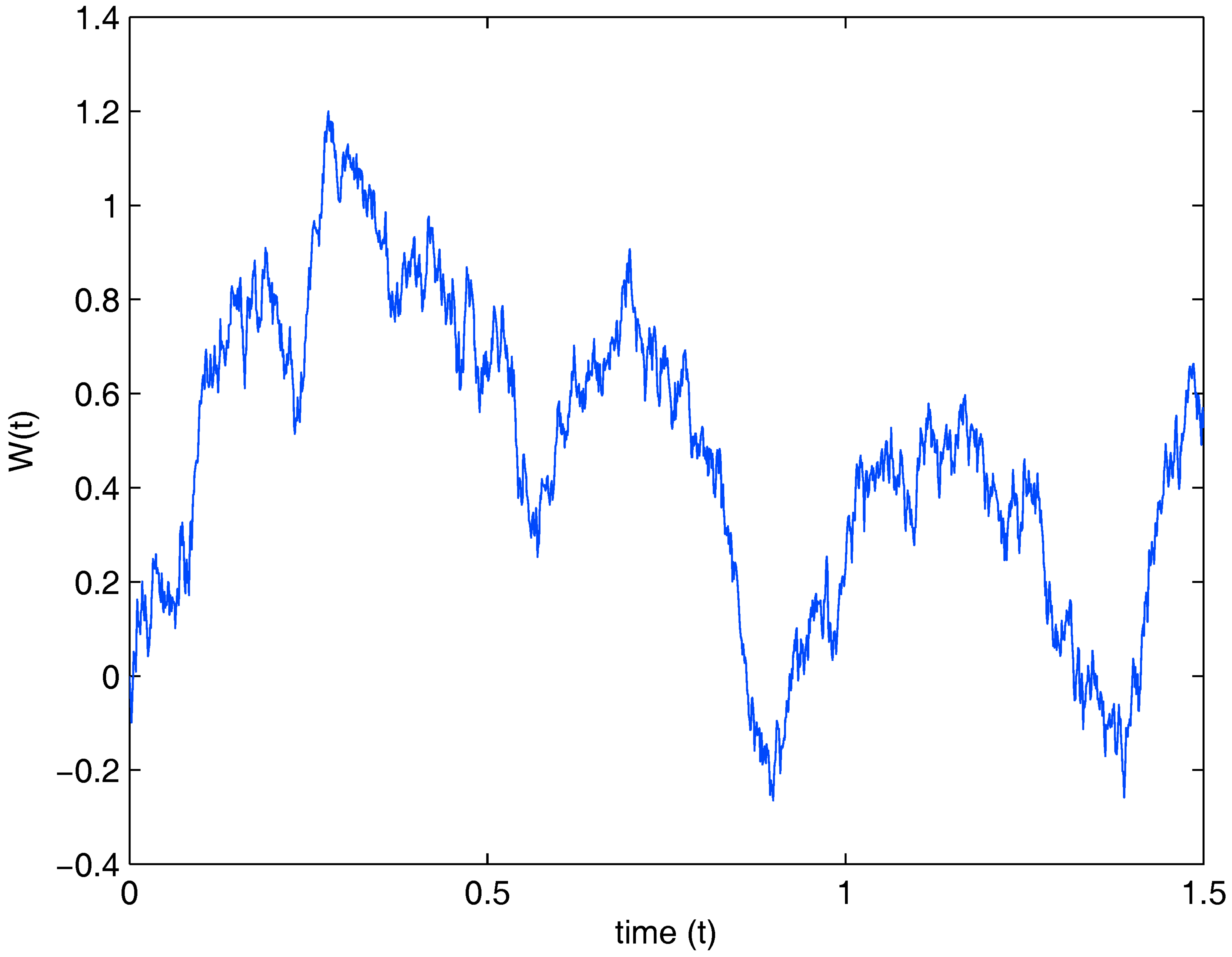
$$dz = -f(z(-t))dt + \sigma(z(-t))dB(-t)$$

Need to store noise

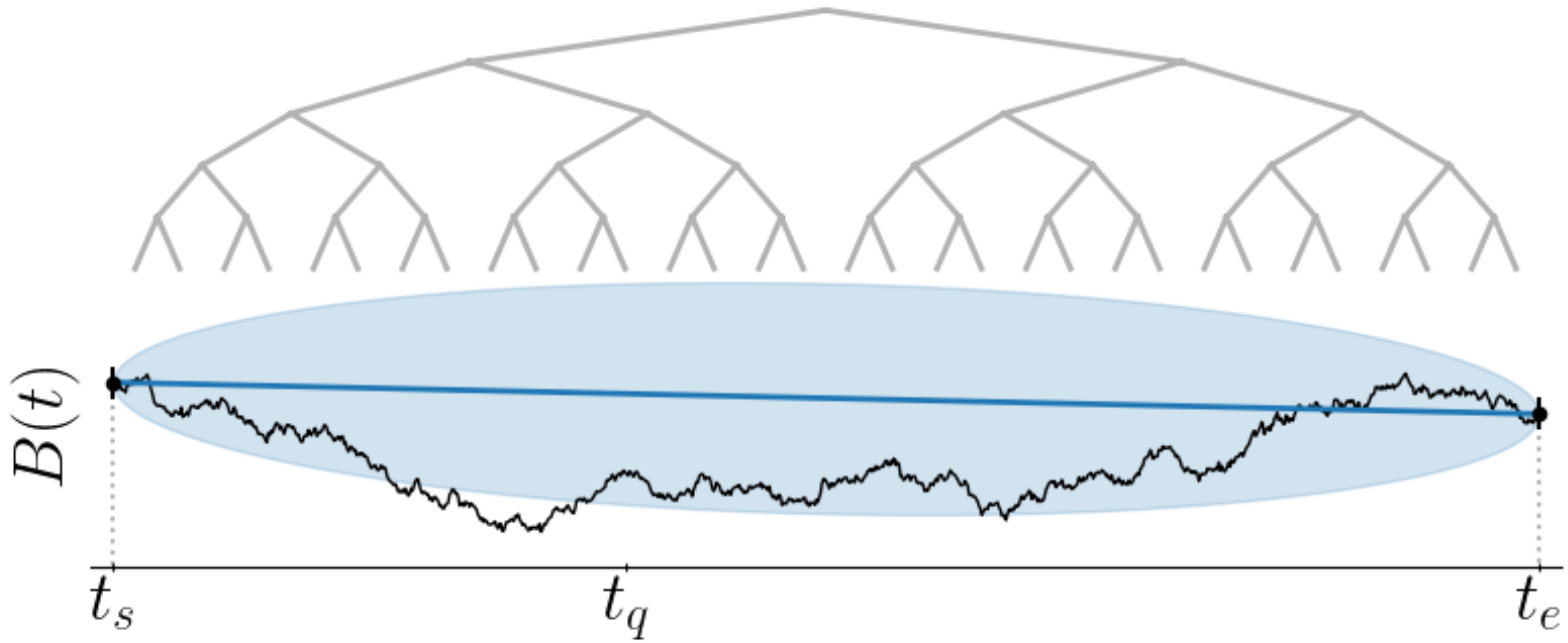
- Reparameterization trick: Use same noise from forward pass on reverse pass

- Infinite reparameterization trick: Use same Brownian motion sample on forward and reverse passes.

- Need to sample entire function



Virtual Brownian Tree



- Can 'zoom in' arbitrarily close at any point.
  - O(1) memory, O(log(1/eps)) time
- splittable random seed ensures all entire sample is consistent

Algorithm 1 ODE Adjoint Sensitivity

Input: Parameters θ, start time t_0, stop time t_1, final state z_{t_1}, loss gradient $\partial\mathcal{L}/z_{t_1}$, dynamics $f(z, t, \theta)$.

def $\overline{f}([z_t, a_t, \cdot],\ t,\ \theta)$: ▷ Augmented dynamics
  $v = f(z_t, -t, \theta)$
  return $[-v,\ a_t \partial v/\partial z,\ a_t \partial v/\partial\theta\,]$

$$\begin{bmatrix} z_{t_0} \\ \partial\mathcal{L}/\partial z_{t_0} \\ \partial\mathcal{L}/\partial\theta \end{bmatrix} = \texttt{odeint}\left(\begin{bmatrix} z_{t_1} \\ \partial\mathcal{L}/\partial z_{t_1} \\ \mathbf{0}_p \end{bmatrix}, \overline{f}, -t_1, -t_0\right)$$

return $\partial\mathcal{L}/\partial z_{t_0}, \partial\mathcal{L}/\partial\theta$

Algorithm 1 ODE Adjoint Sensitivity

Input: Parameters θ, start time t_0, stop time t_1, final state z_{t_1}, loss gradient $\partial\mathcal{L}/z_{t_1}$, dynamics $f(z, t, \theta)$.

def $\overline{f}([z_t, a_t, \cdot],\ t,\ \theta)$: ▷ Augmented dynamics
  $v = f(z_t, -t, \theta)$
  return $[-v,\ a_t\partial v/\partial z,\ a_t\partial v/\partial\theta]$

$$\begin{bmatrix} z_{t_0} \\ \partial\mathcal{L}/\partial z_{t_0} \\ \partial\mathcal{L}/\partial\theta \end{bmatrix} = \texttt{odeint}\left(\begin{bmatrix} z_{t_1} \\ \partial\mathcal{L}/\partial z_{t_1} \\ \mathbf{0}_p \end{bmatrix}, \overline{f}, -t_1, -t_0\right)$$

return $\partial\mathcal{L}/\partial z_{t_0}, \partial\mathcal{L}/\partial\theta$

Algorithm 2 SDE Adjoint Sensitivity (Ours)

Input: Parameters θ, start time t_0, stop time t_1, final state z_{t_1}, loss gradient $\partial\mathcal{L}/z_{t_1}$, drift $f(z, t, \theta)$, diffusion $\sigma(z, t, \theta)$, Wiener process sample $w(t)$.

def $\overline{f}([z_t, a_t, \cdot],\ t,\ \theta)$: ▷ Augmented drift
  $v = f(z_t, -t, \theta)$
  return $[-v,\ a_t\partial v/\partial z,\ a_t\partial v/\partial\theta]$

def $\overline{\sigma}([z_t, a_t, \cdot],\ t,\ \theta)$: ▷ Augmented diffusion
  $v = \sigma(z_t, -t, \theta)$
  return $[-v, a_t\partial v/\partial z, a_t\partial v/\partial\theta]$

def $\overline{w}(t)$: ▷ Replicated noise
  return $[-w(-t), -w(-t), -w(-t)]$

$$\begin{bmatrix} z_{t_0} \\ \partial\mathcal{L}/\partial z_{t_0} \\ \partial\mathcal{L}/\partial\theta \end{bmatrix} = \texttt{sdeint}\left(\begin{bmatrix} z_{t_1} \\ \partial\mathcal{L}/\partial z_{t_1} \\ \mathbf{0}_p \end{bmatrix}, \overline{f}, \overline{\sigma}, \overline{w}, -t_1, -t_0\right)$$

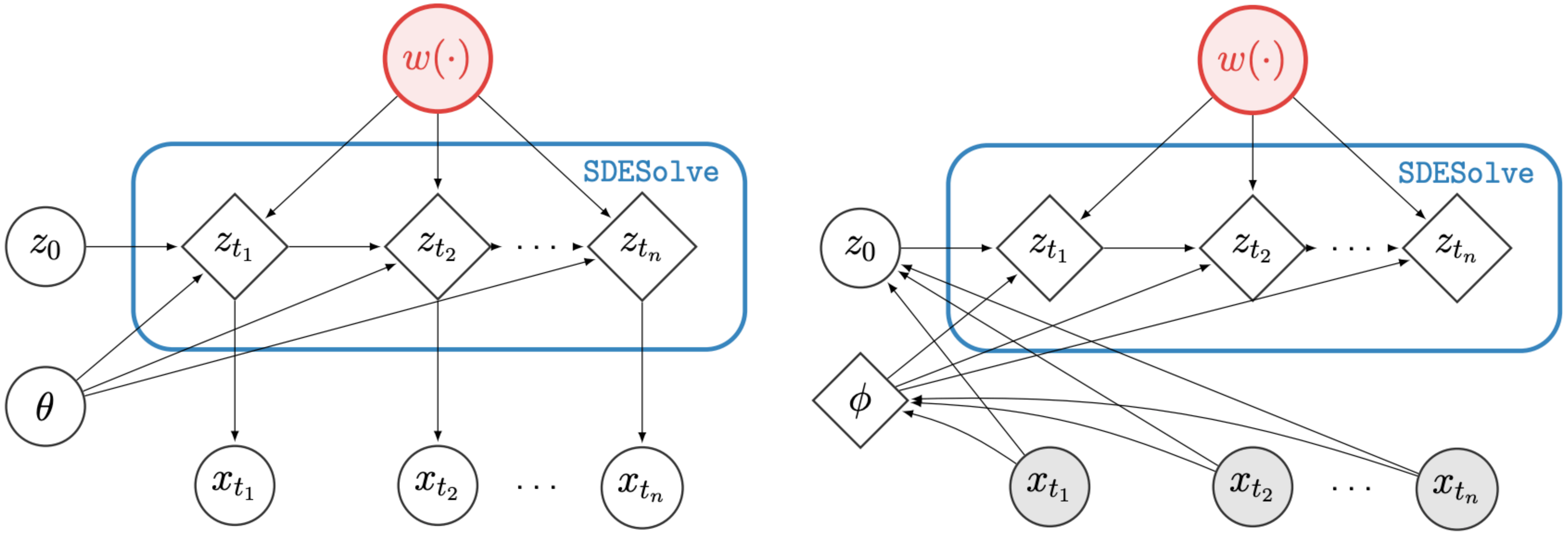
return $\partial\mathcal{L}/\partial z_{t_0}, \partial\mathcal{L}/\partial\theta$

Time and memory cost

Method	Memory	Time
Forward pathwise [14, 60]	$\mathcal{O}(1)$	$\mathcal{O}(LD)$
Backprop through solver [11]	$\mathcal{O}(L)$	$\mathcal{O}(L)$
Stochastic adjoint (ours)	$\mathcal{O}(1)$	$\mathcal{O}(L \log L)$

- Time more like O(L) when dynamics are expensive
- Can now fit large SDE models by gradient descent!

Latent-variable model

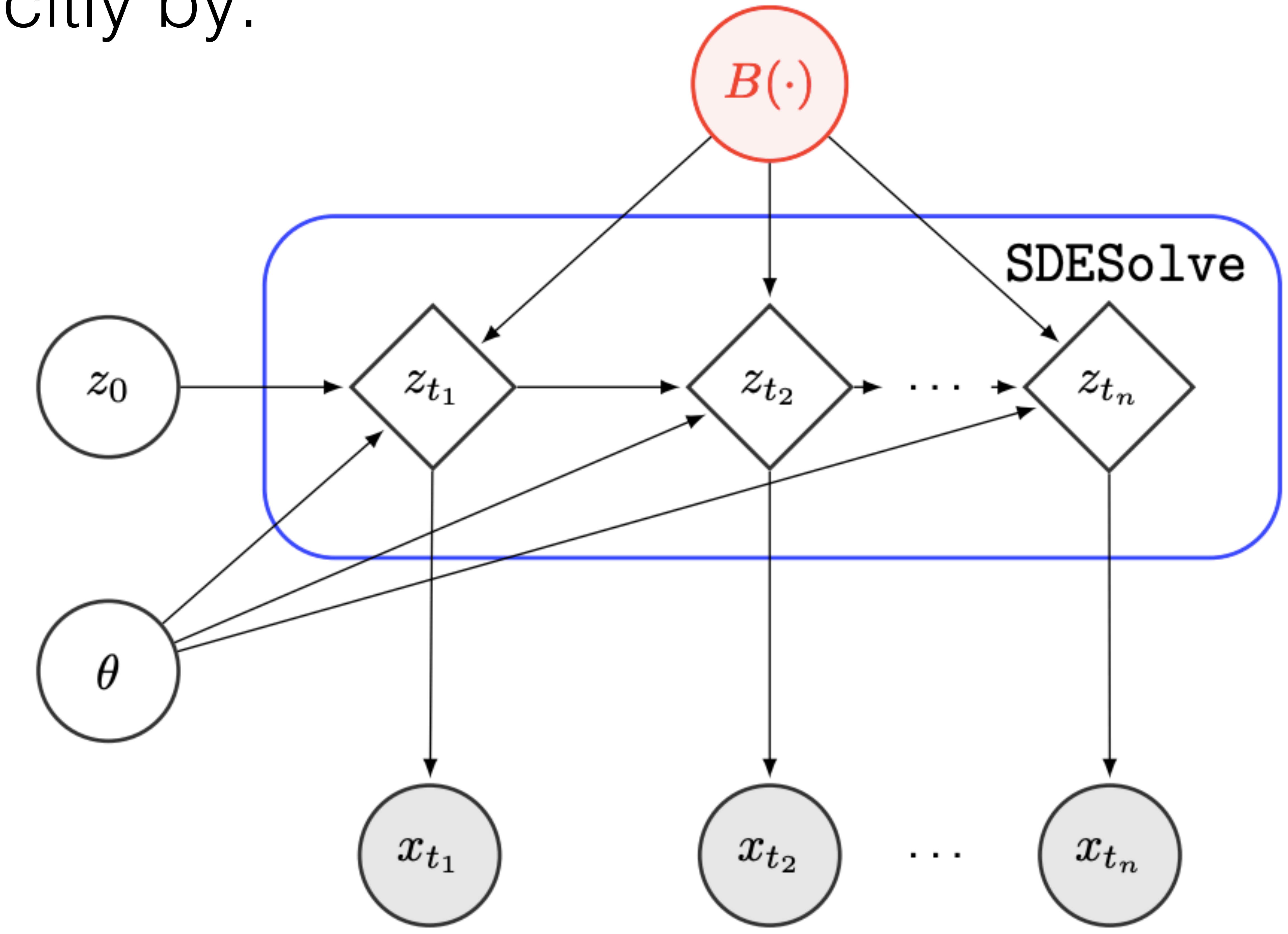


(a) Generation (b) Recognition

- Can handle arbitrary likelihoods. Infinite-dimensional VAE.

Latent SDE Model

- Generative model (decoder) defined implicitly by:
  - an SDE $dz_p = f_\theta(z(t))dt + \sigma_\theta(z(t))dB(t)$
  - A likelihood (noise model) p(x_t | z_t)

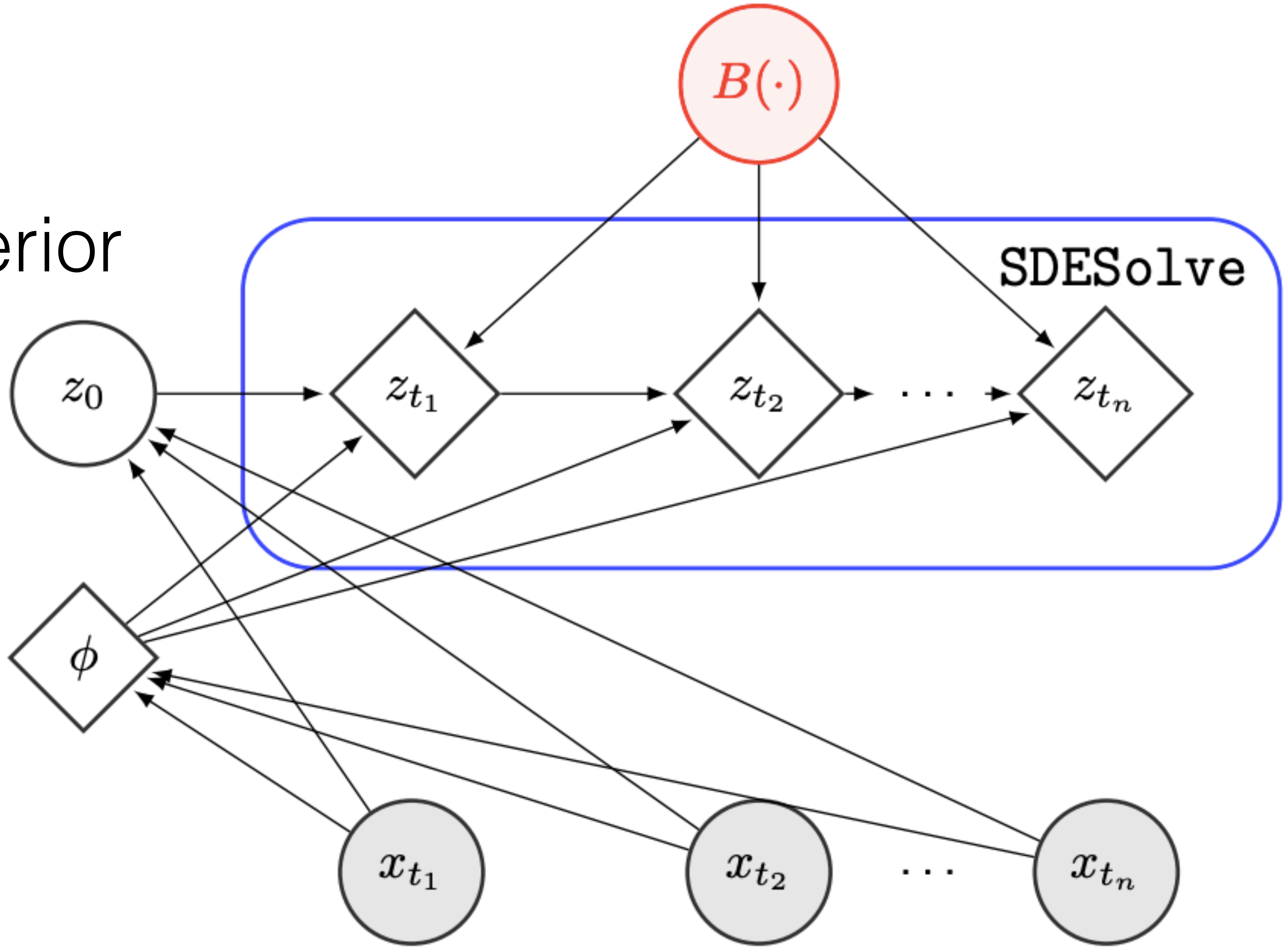


(b) Generation

Variational inference

- Recognition model (encoder) takes in data, outputs:
  - Distribution over initial state q(z0 | x1.. xN)
  - Params of SDE defining approximate posterior

$$dz_q = f_\phi(z(t))dt + \sigma_\theta(z(t))dB'(t)$$



(a) Inference

- Like Neural Processes, but actually a well-defined probabilistic model

Variational inference

- To optimize ELBO, need unbiased estimate of KL divergence between
- prior: $dz_p = f_\theta(z(t))dt \ + \ \sigma_\theta(z(t))dB(t)$
- approximate posterior: $dz_q = f_\phi(z(t))dt \ + \ \sigma_\theta(z(t))dB'(t)$

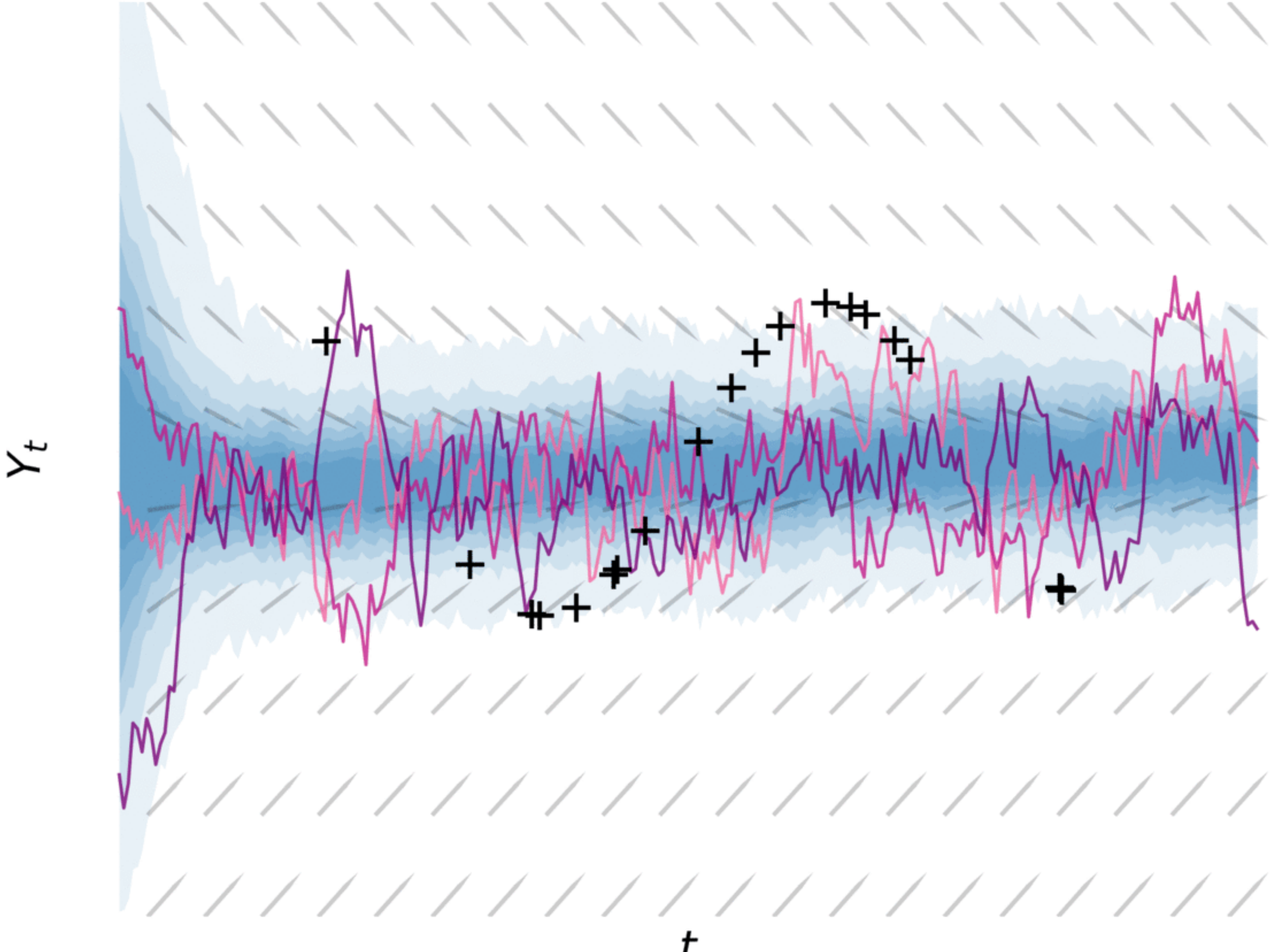
$$\mathcal{L}_{\mathrm{VI}} = \mathbb{E}\Big[\tfrac{1}{2}\int_0^T |u(Z_t, t)|^2 \ \mathrm{d}t - \sum_{i=1}^{N} \log p(y_{t_i}|z_{t_i})\Big]$$

$$u(t) = \left|\frac{f_\theta(z(t)) - f_\phi(z(t))}{\sigma_\theta(z(t))}\right|_2^2$$

1D Latent SDE

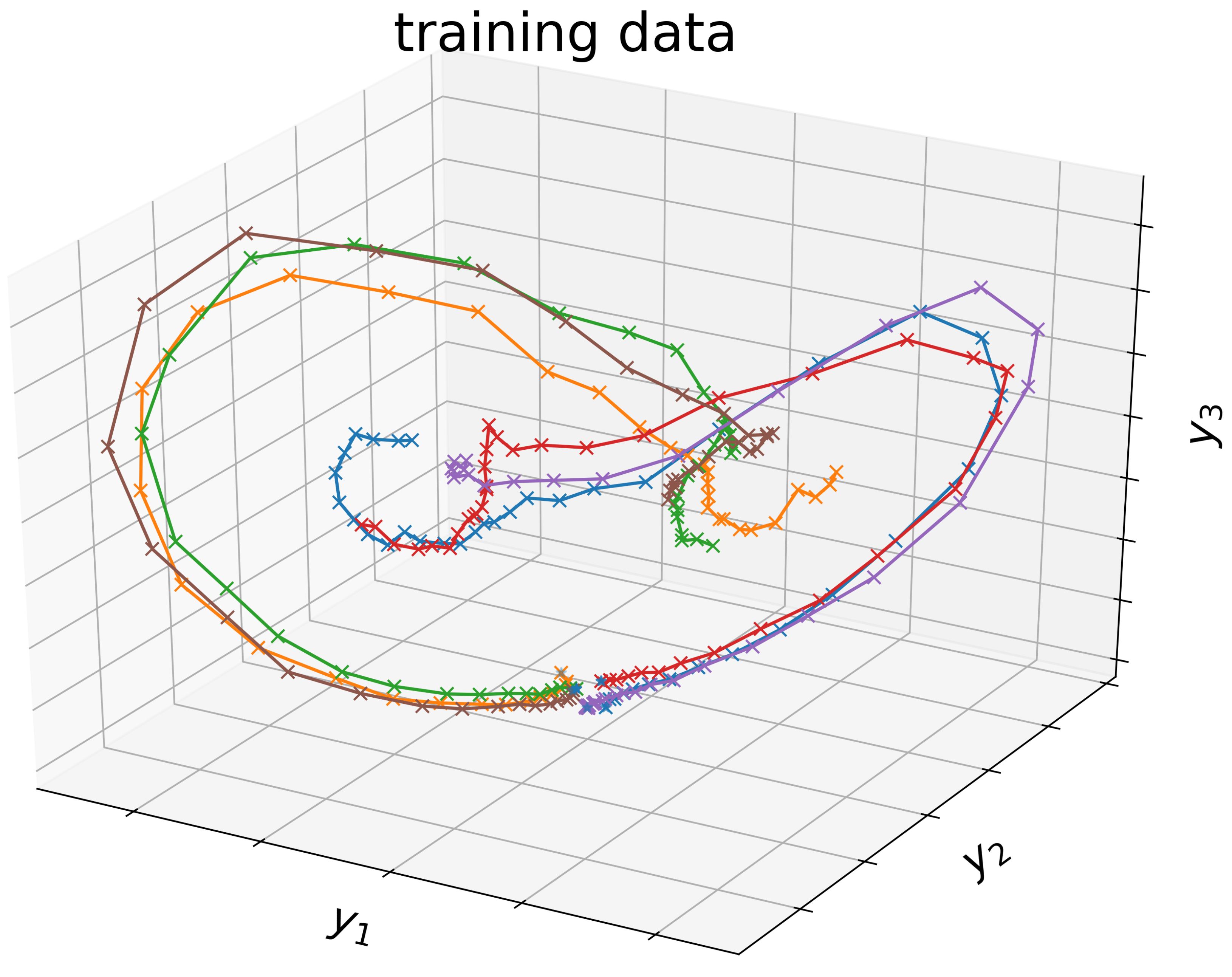
- Ornstein-Uhlenbeck prior, Laplace likelihood

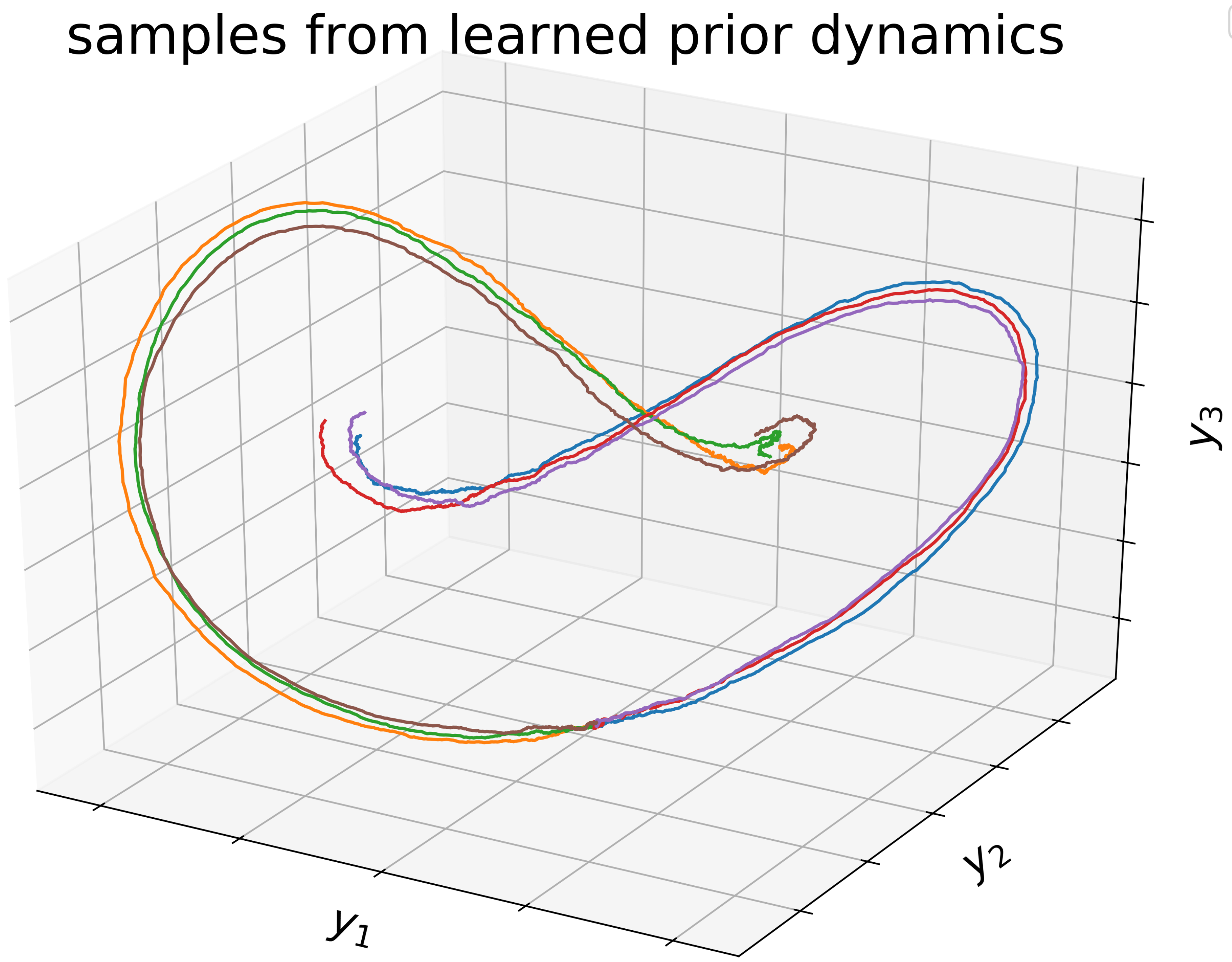
- Posterior SDE steers sample paths to data



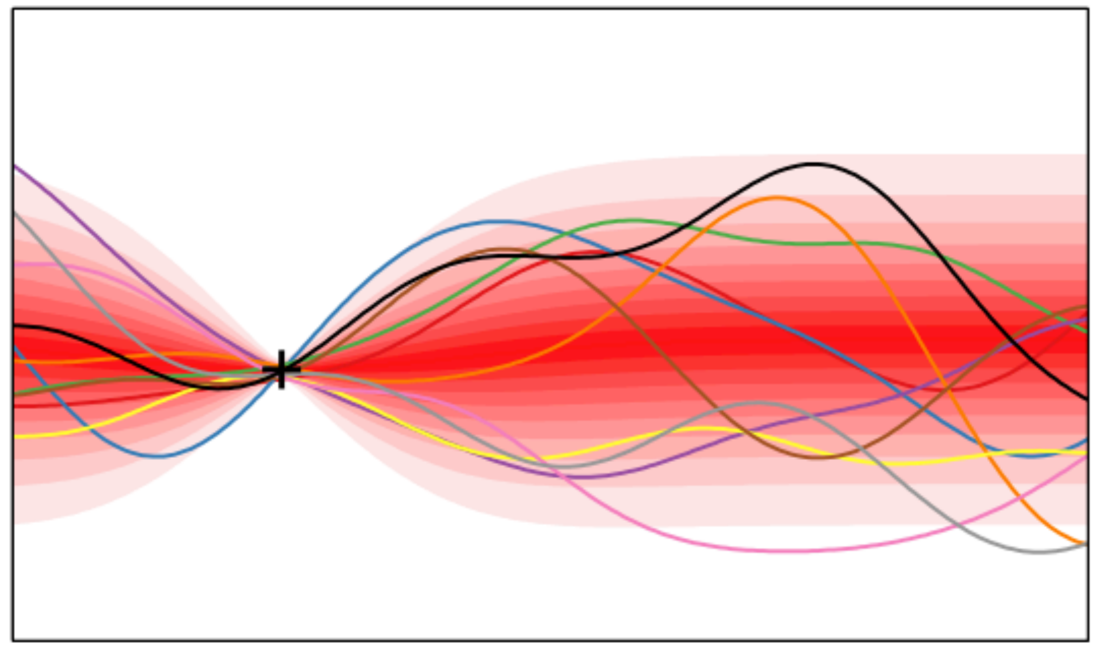
Latent SDEs:
An unexplored model class

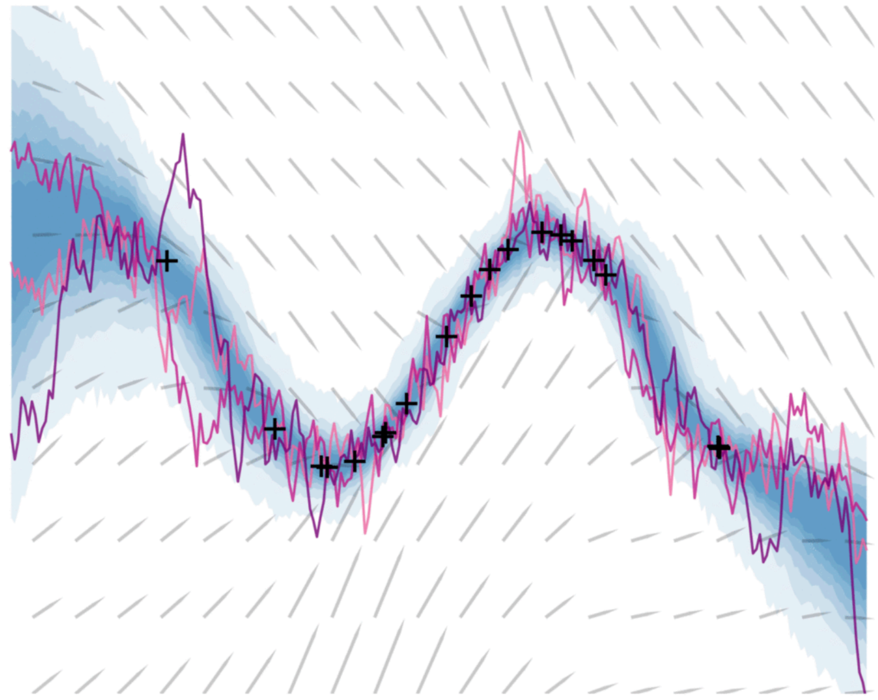
- Define implicit prior + posterior over functions
- Define observation likelihoods. Anything differentiable wrt latent state (e.g. text models!)
- Train everything with stochastic variational inference.
- Can use adaptive-step SDE solvers.
- Should scale to millions of params, huge states. Can use adaptive Milstein solver (only diagonal noise).





GPs vs Markov SDEs



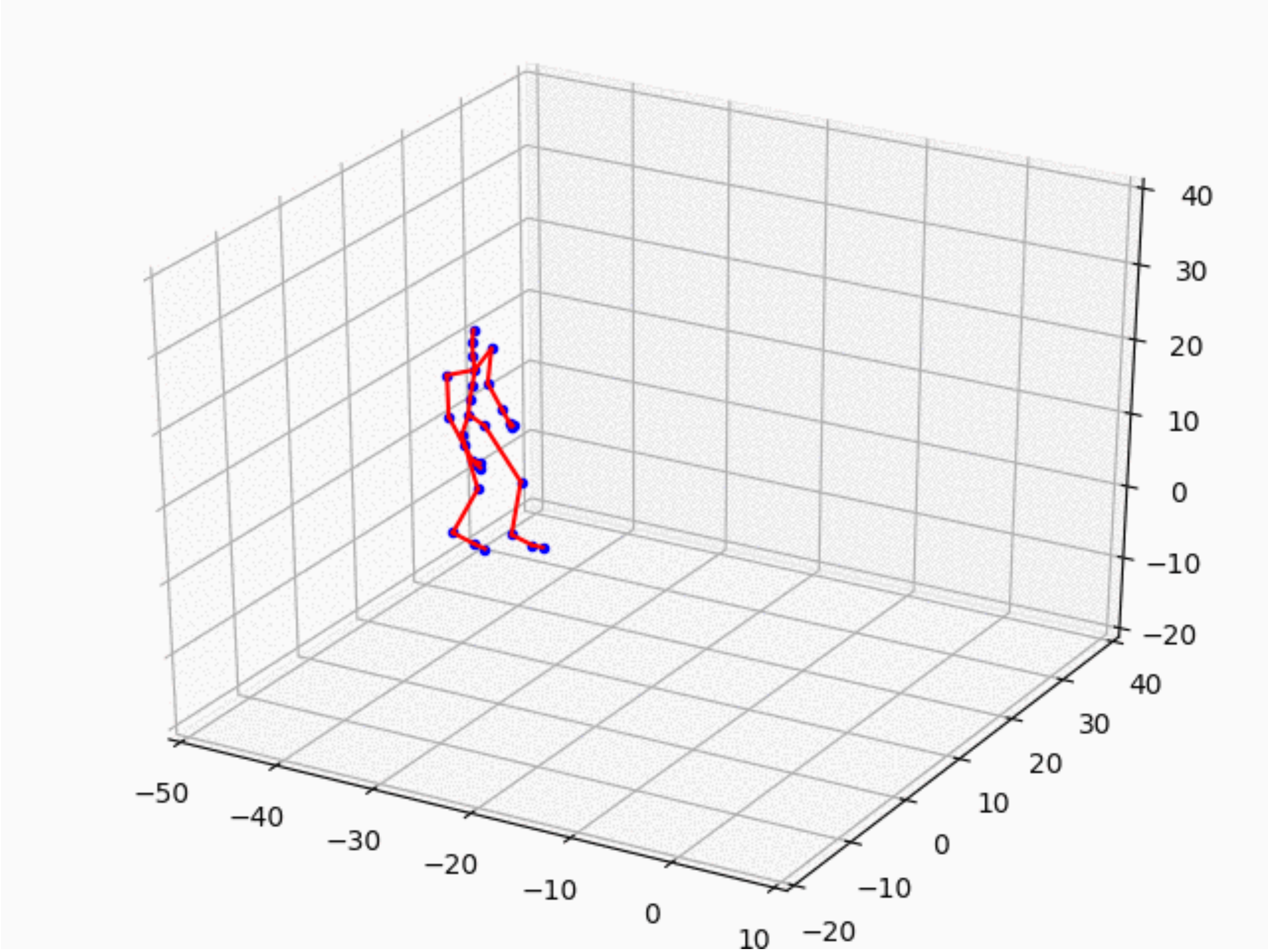


- mean and kernel funcs
- Not closed over marginal transforms.
  exp(f(x) ~ GP)
  not a GP
- Multi-dim input fine

- Drift and diffusion funcs
- Closed over marginal transforms.
  exp(f(x) ~ SDE)
  still an SDE
- Only single-dim input

Early latent SDE results: Mocap

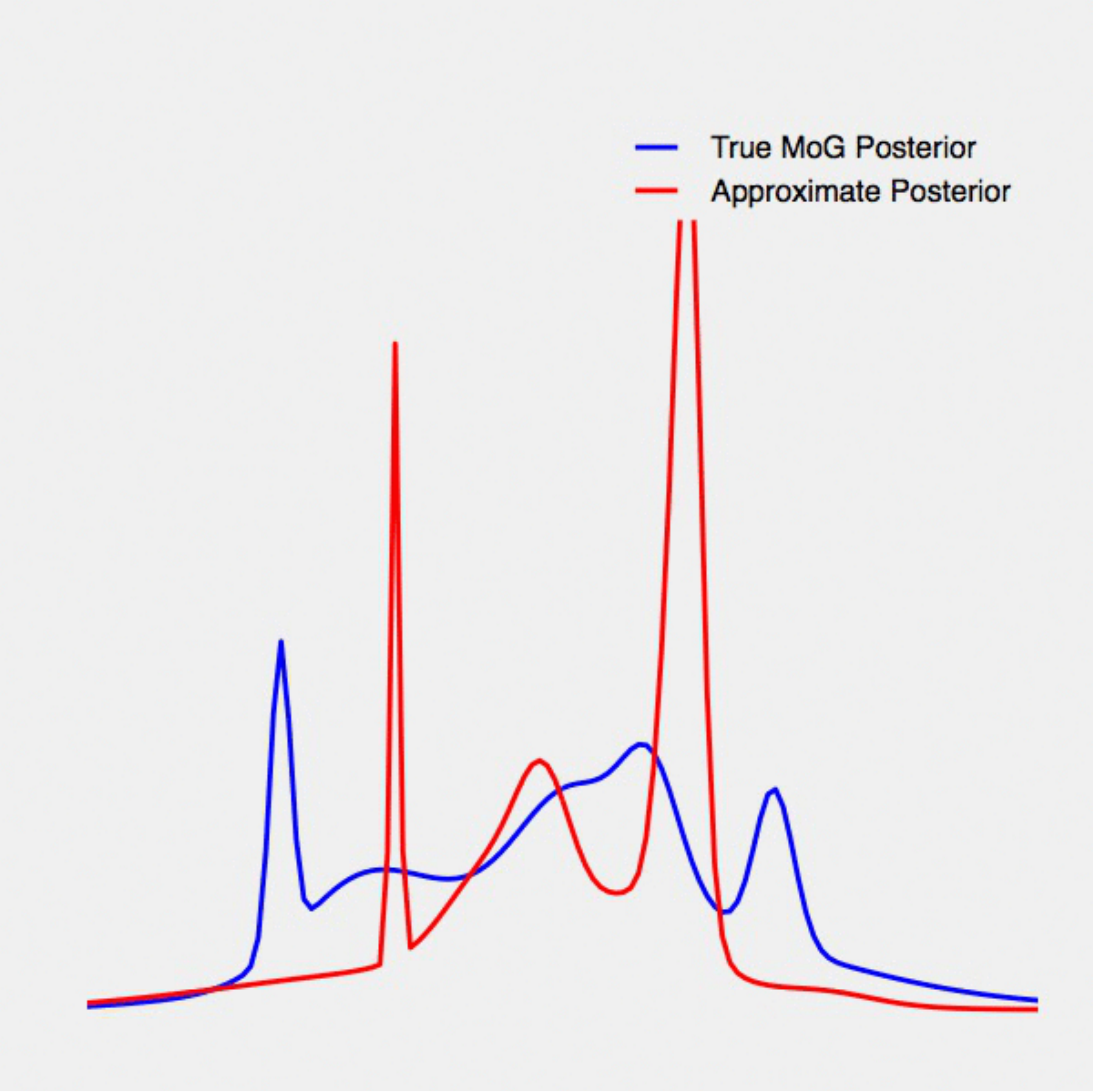
- 50D data, 6D latent space, sharing dynamics and recognition params across time series (11000 params)

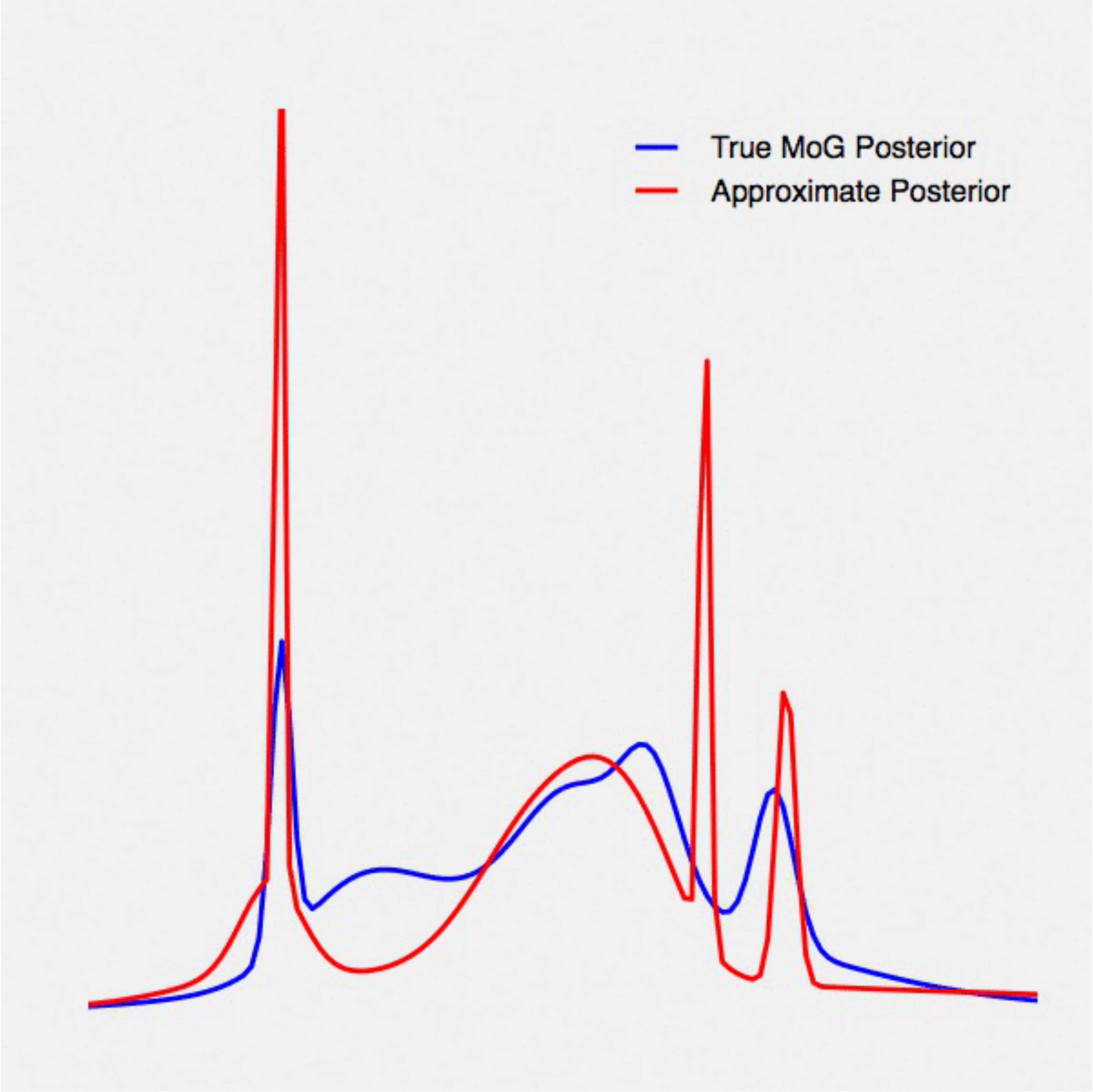


Method	Test MSE
npODE [18]	22.96†
NeuralODE [4]	$22.49 \pm 0.88^\dagger$
ODE^{2}VAE [61]	$10.06 \pm 1.4^\dagger$
ODE^{2}VAE-KL [61]	$8.09 \pm 1.95^\dagger$
Latent ODE [4, 50]	5.98 ± 0.28
Latent SDE (this work)	$\mathbf{4.03 \pm 0.20}$

SVI Gradient variance

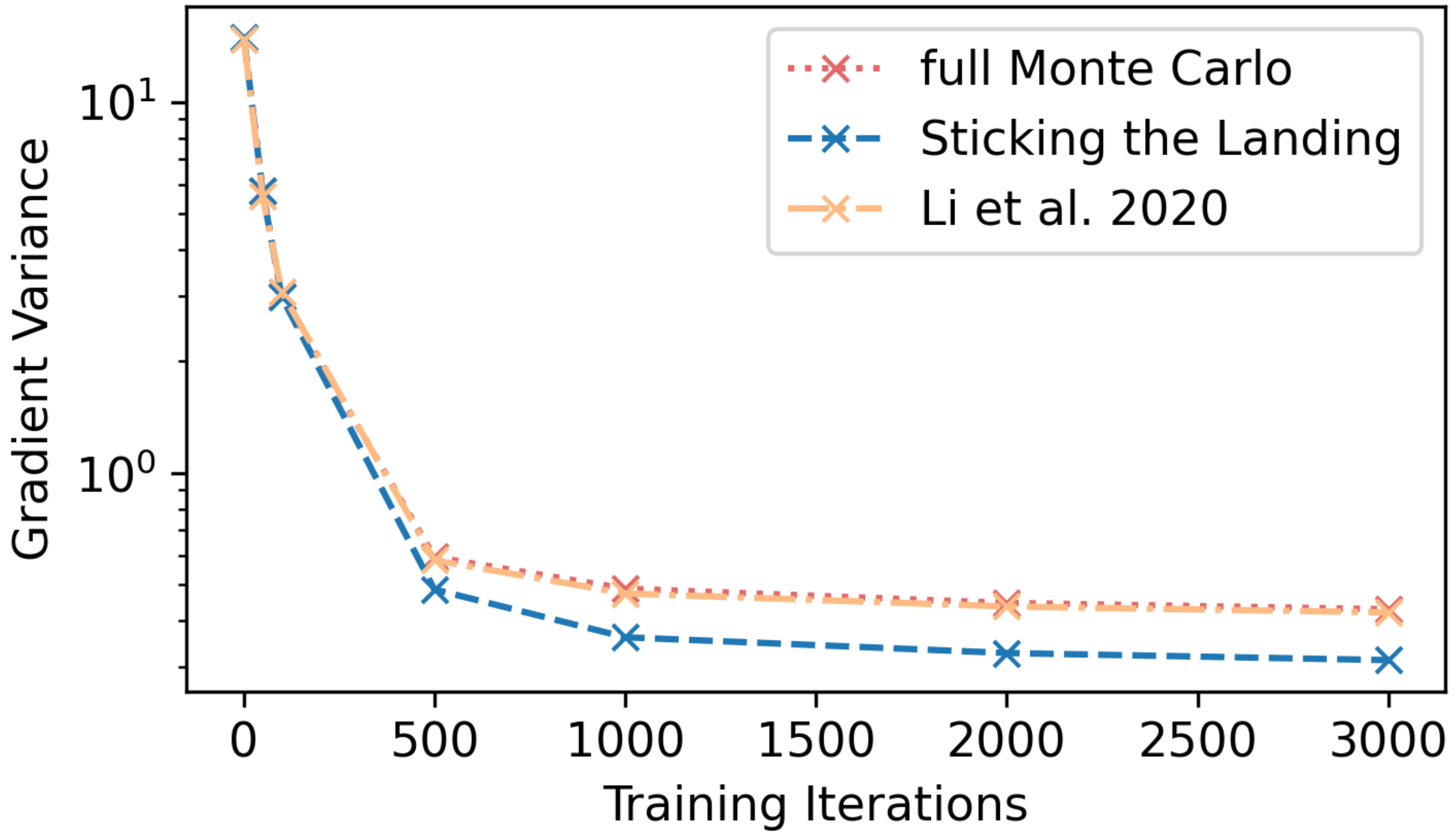
- Sticking the landing [Roeder, Wu, Duvenaud, NIPS 2017]
  SVI gradient estimator whose variance goes to zero as q(z) -> p(z|x)

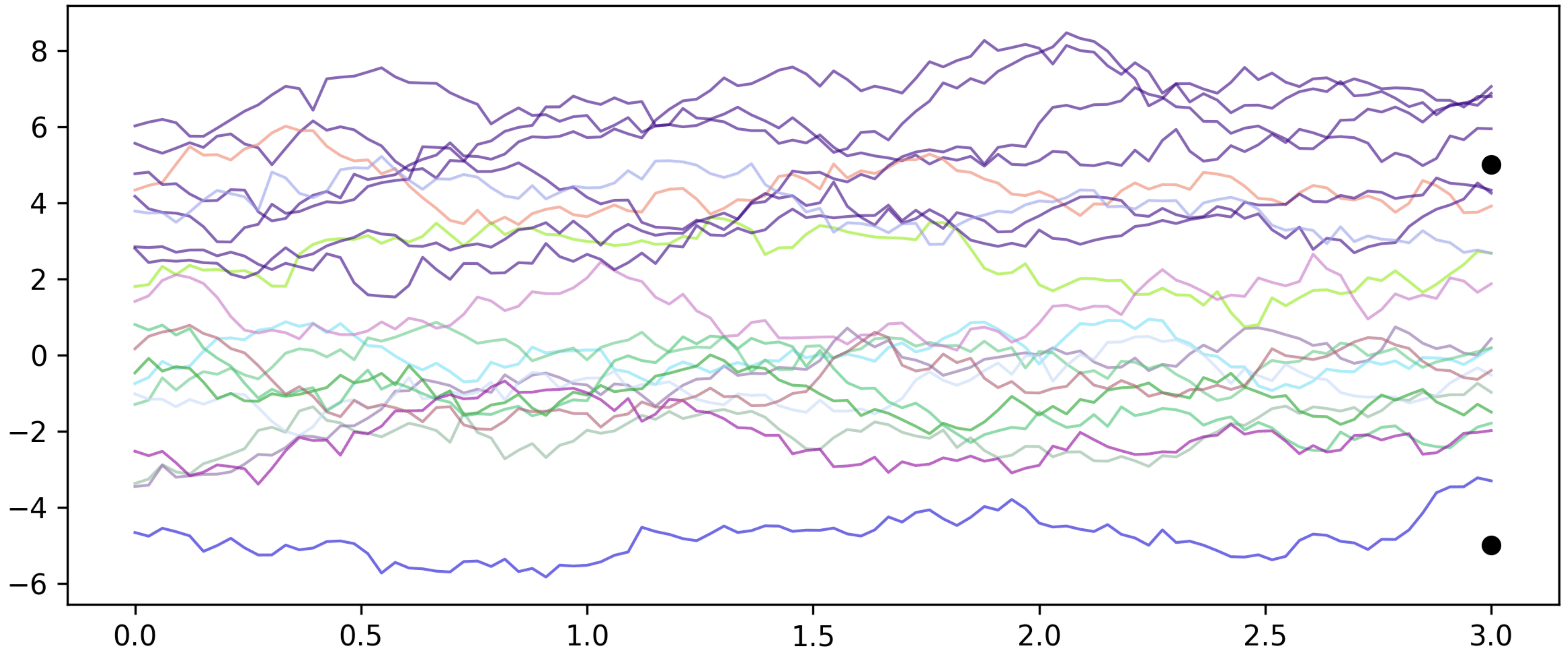


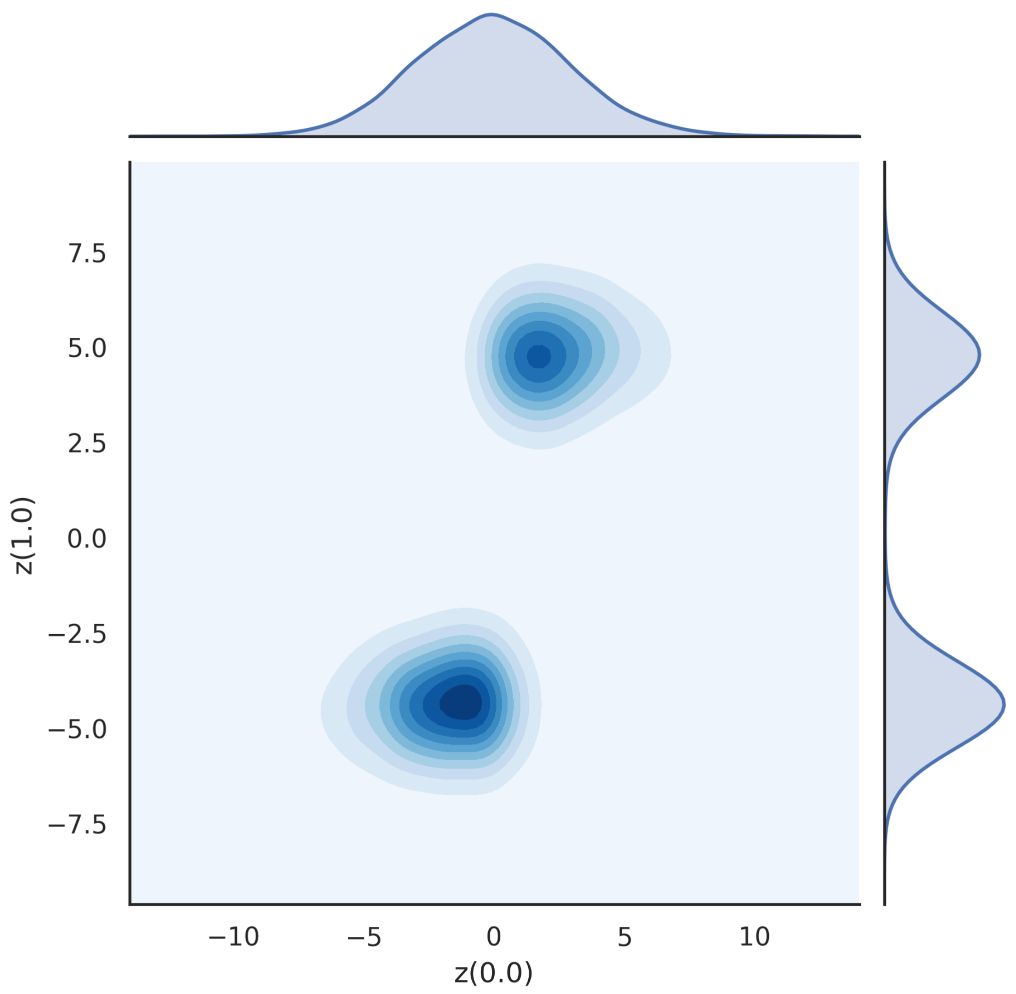


SVI Gradient variance

- New for ICML 2021: We extended "Sticking the Landing" to SDEs
- Reminder: Approx posterior can be arbitrarily close to true posterior!







Takeaway

- Large SDE-based latent-variable models now practical-ish
- Should handle real irregularly-sampled time series!
  - Can condition on time of observations
  - Can answer any query, not just forward prediction
- In practice, start with an RNN!
- Code: https://github.com/google-research/torchsde

Next steps

- Modeling:
  - Multi-timescale SDE - skip low-level details
  - Jump processes, SPDEs
- Applications:
  - Population genetics, finance, epidemiology? User traces? Let's talk!
  - Infinitely deep Bayesian neural networks

Building an infinitely-deep BNN

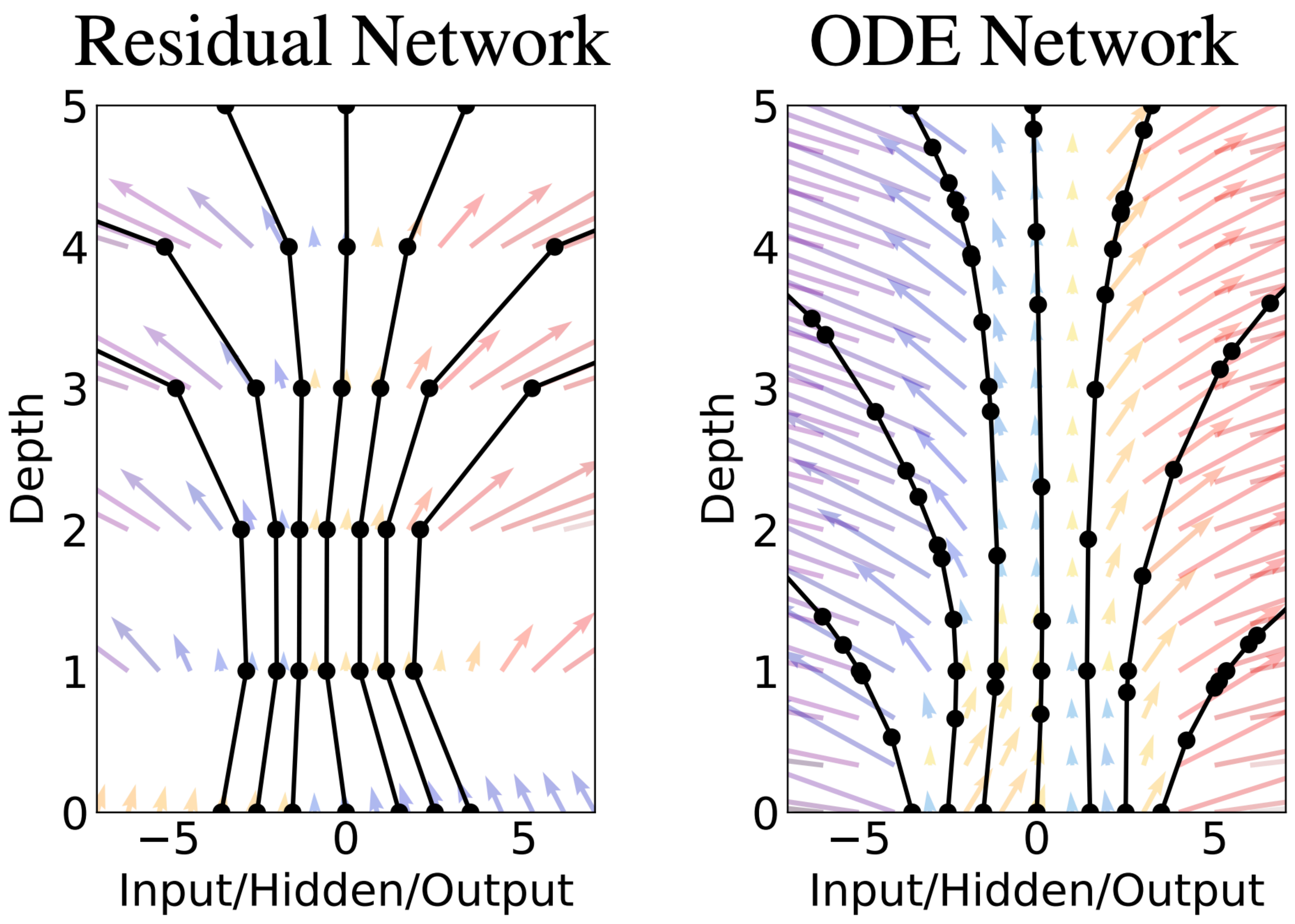
- Start with a ResNet:

$$h_{t+\epsilon} = h_t + \epsilon f_h(h_t, w_t)$$

- Take limit as eps -> 0, number of layers grows.

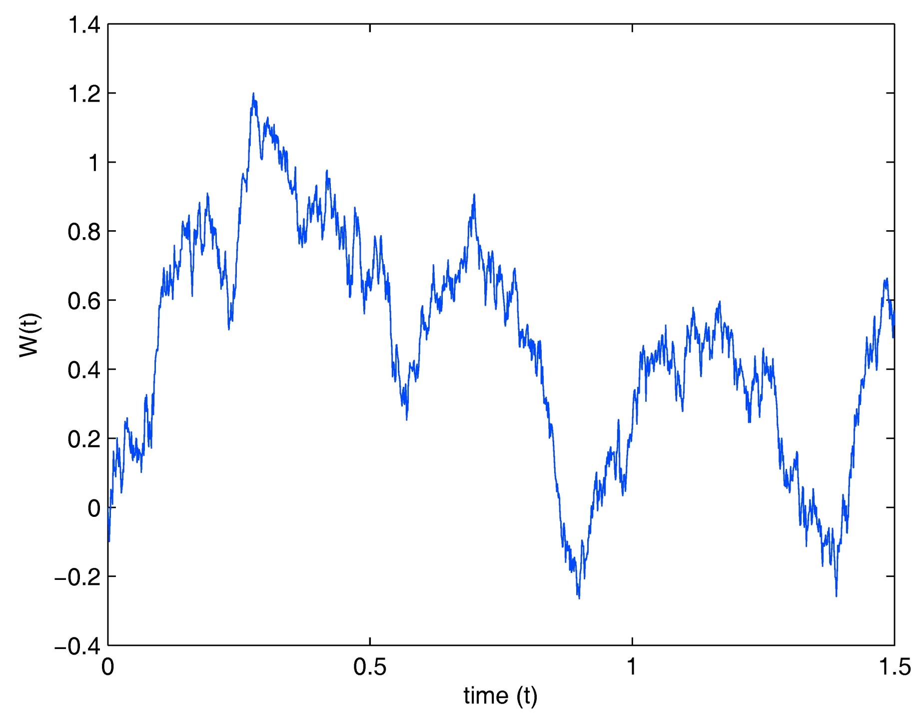
- Given a process over weights, activations h follow a random ODE:

$$dh_t = f_h(h_t, w_t)$$



Building an infinitely-deep BNN

- Prior on weights is a OU process



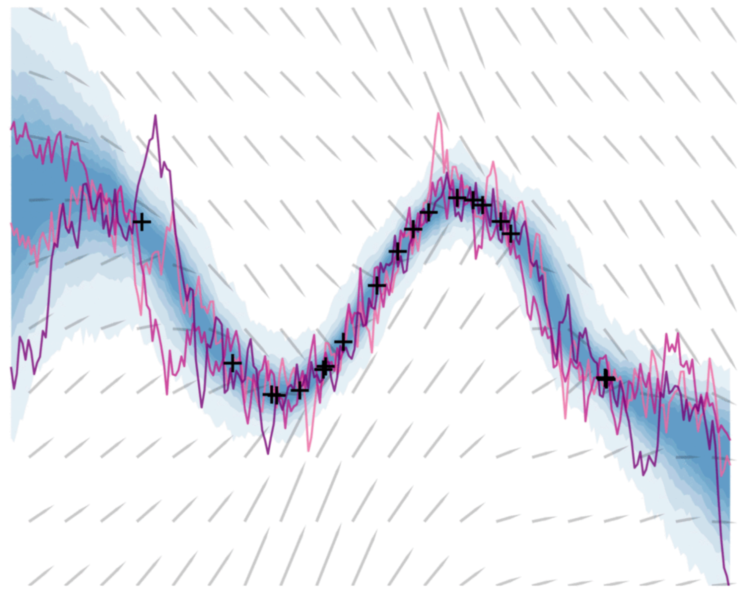
$$dw_t = -w_t dt + dB_t$$

- Likelihood depends on activation at time 1:

$$p(y \mid x, w) = \mathcal{N}(y \mid h_1, w)$$

- Define approximate posterior on weights:

$$dw_t = f_w(w_t, \phi)dt + dB_t$$



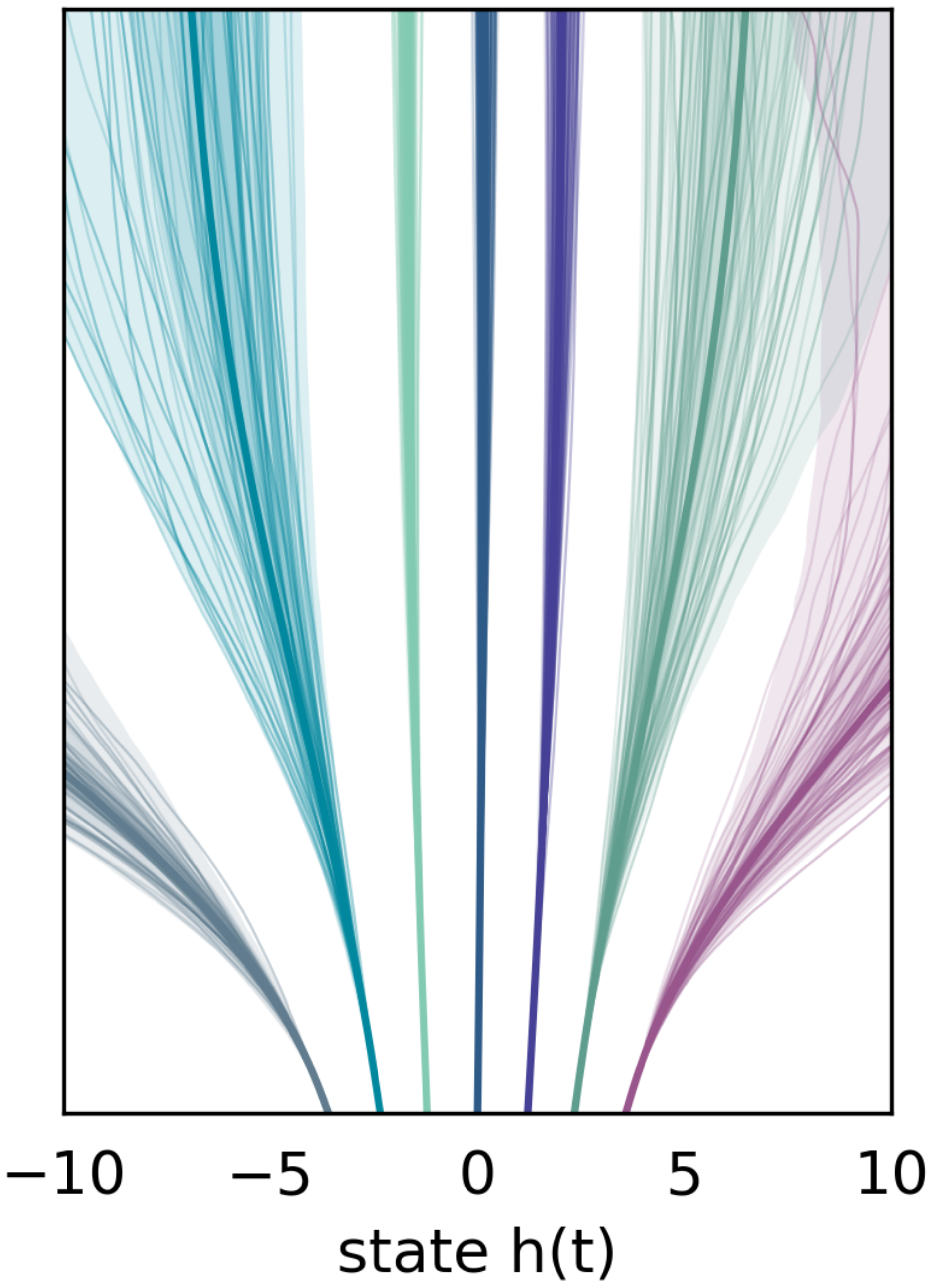
Building an infinitely-deep BNN

- Can sample weights from approx posterior and evaluate network output in one SDE solve:

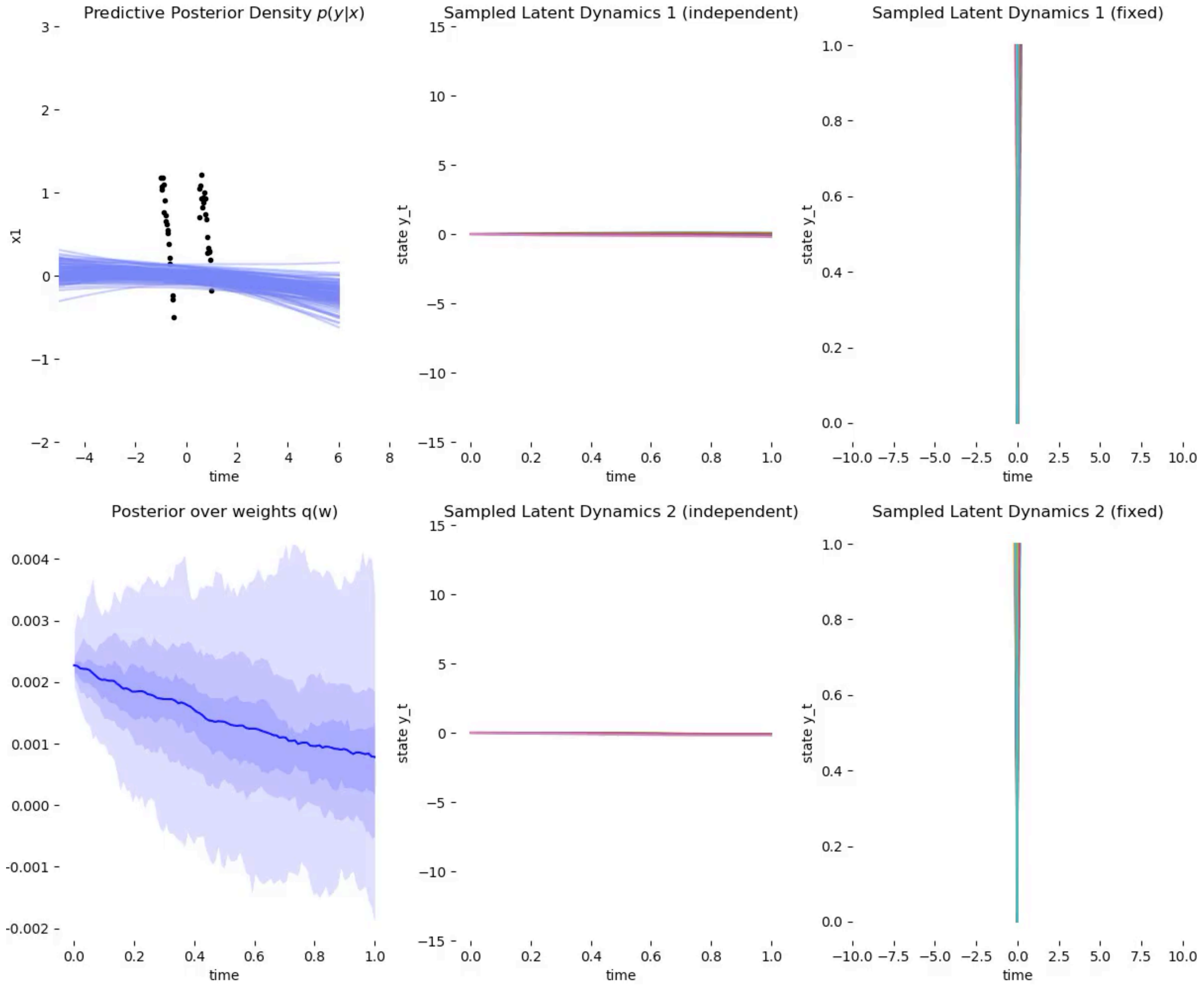
$$d \begin{bmatrix} w_t \\ h_t \end{bmatrix} = \begin{bmatrix} f_w(w_t, {\color{red}\phi}) \\ f_h(h_t, w_t) \end{bmatrix} dt + \begin{bmatrix} I \\ \mathbf{0} \end{bmatrix} dB_t$$

- Start h_0 at input to neural network x.

- h_1 is output of neural network

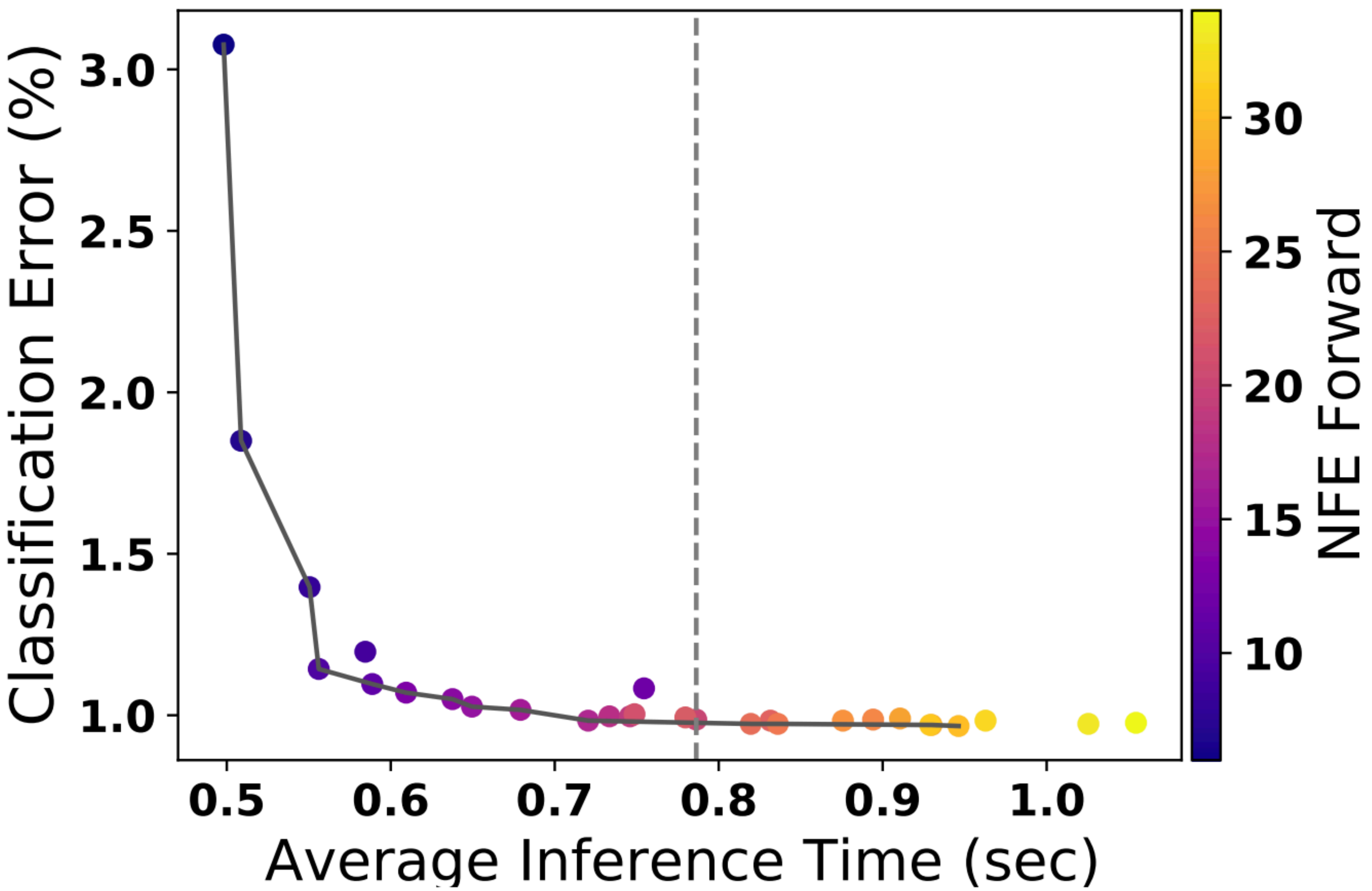


Training an infinitely-deep BNN



Practical Advantages (in theory)

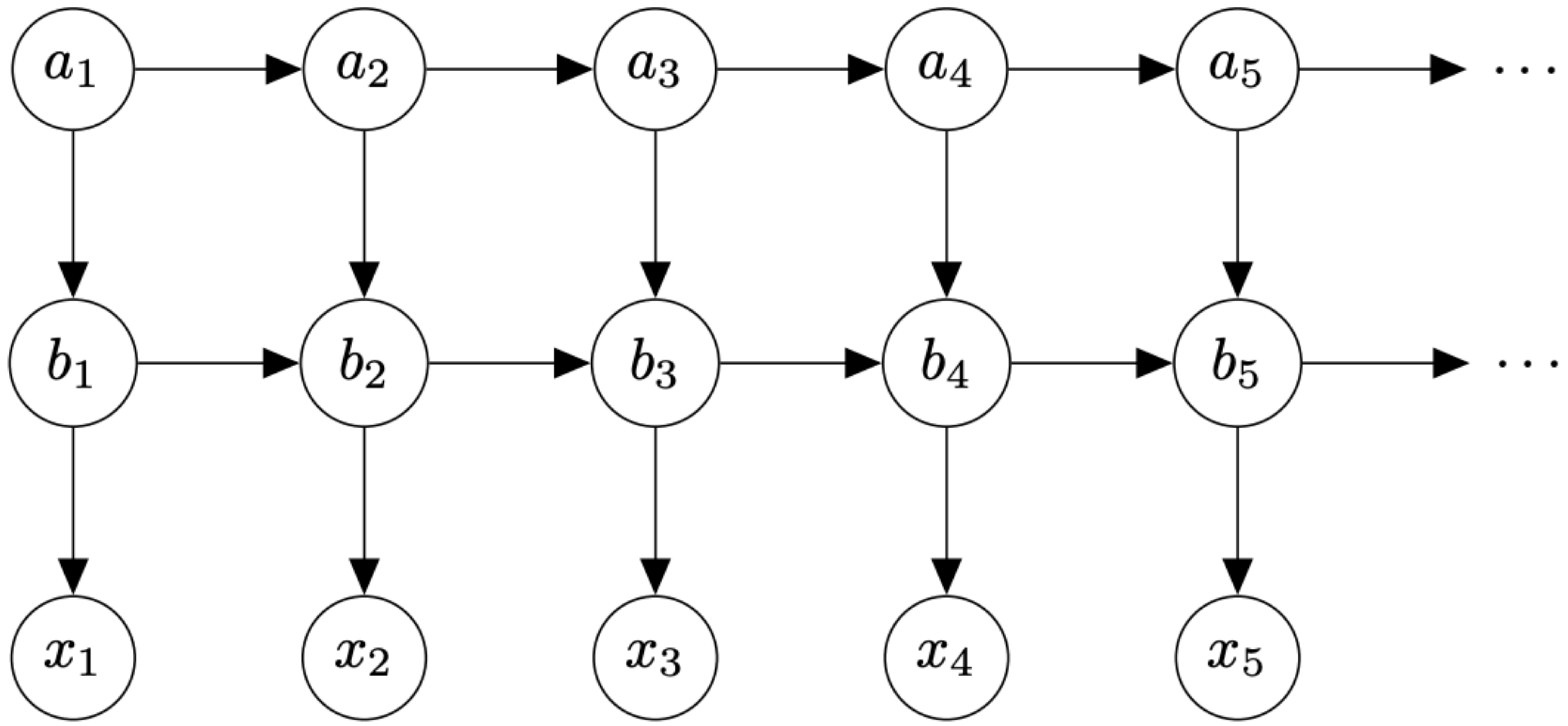
- Continuous-time formulation allows use of adaptive SDE solvers.
- Can adjust adaptive solver tolerance at test time, trades off speed vs precision
- Arbitrarily-flexible approx posterior with no O(D^3) scaling.
  - (True scaling unknown!?)



At least scales to CIFAR10

	MNIST		CIFAR-10	
Model	Accuracy (%)	ECE ($\times 10^{-2}$)	Accuracy (%)	ECE ($\times 10^{-2}$)
ResNet32	99.46 ± 0.00	2.88 ± 0.94	87.35 ± 0.00	8.47 ± 0.39
ODENet	98.90 ± 0.04	1.11 ± 0.10	88.30 ± 0.29	8.71 ± 0.21
HyperODENet	99.04 ± 0.00	1.04 ± 0.09	87.92 ± 0.46	15.86 ± 1.25
Mean Field ResNet32	99.44 ± 0.00	2.76 ± 1.28	86.97 ± 0.00	3.04 ± 0.94
Mean Field ODENet	98.81 ± 0.00	2.63 ± 0.31	81.59 ± 0.01	3.62 ± 0.40
Mean Field HyperODENet	98.77 ± 0.01	2.82 ± 1.34	80.62 ± 0.00	4.29 ± 1.10
SDE BNN	99.30 ± 0.09	0.63 ± 0.10	88.98 ± 0.94	7.60 ± 0.37
SDE BNN (+ STL)	99.10 ± 0.09	0.78 ± 0.12	89.10 ± 0.45	7.97 ± 0.51

Multi-Scale Continuous-time



$$da = f_a(a(t))\ \mathrm{d}t \quad + \quad \sigma_a(a(t))\ \mathrm{d}W(t)$$

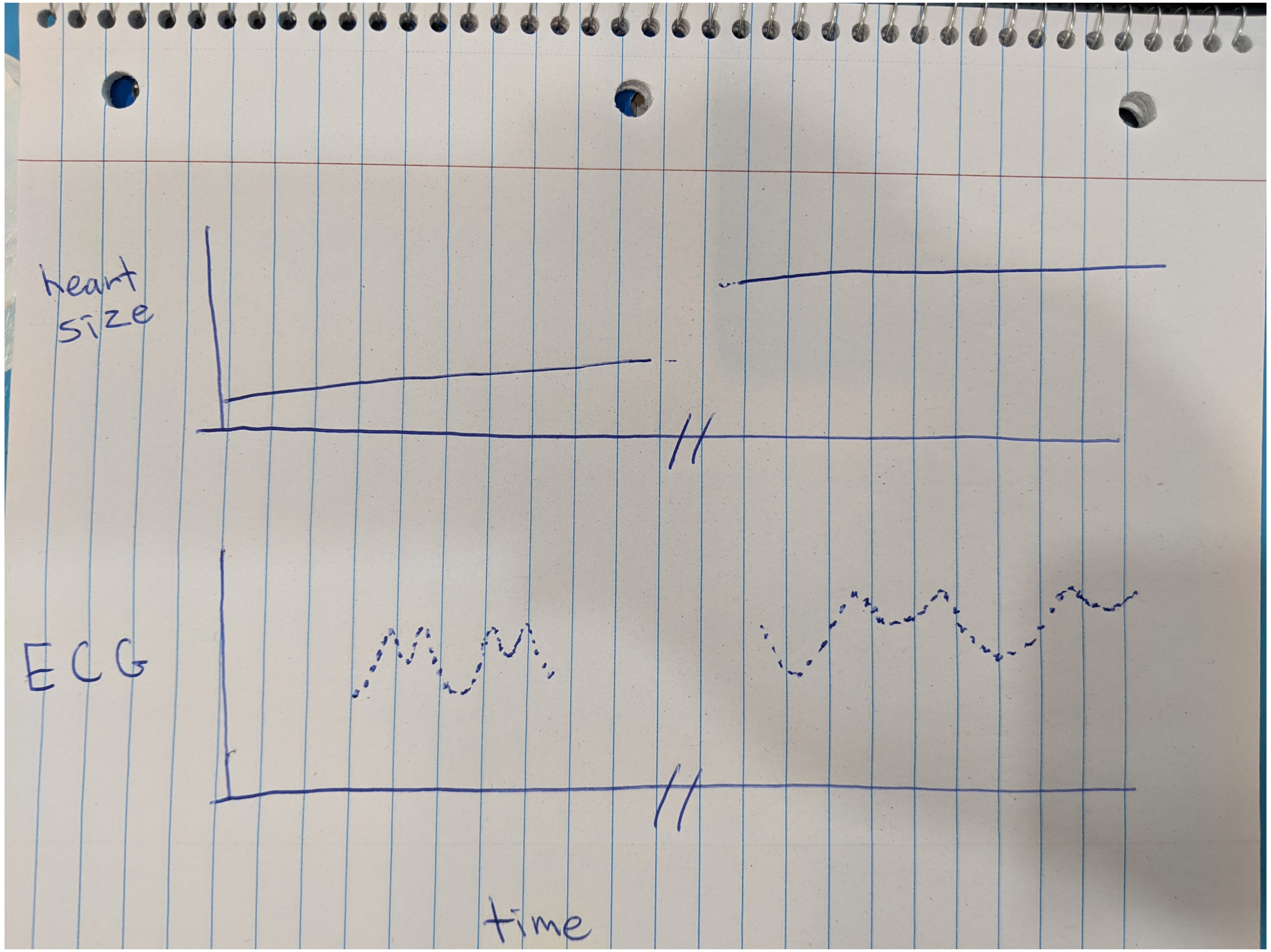
$$db = f_b(a(t), b(t))\ \mathrm{d}t + \sigma_b(a(t), b(t))\ \mathrm{d}W(t)$$

- Conjecture: Infinitesimal time limit of Markov models give SDE with variable dependence same as parents in graph.

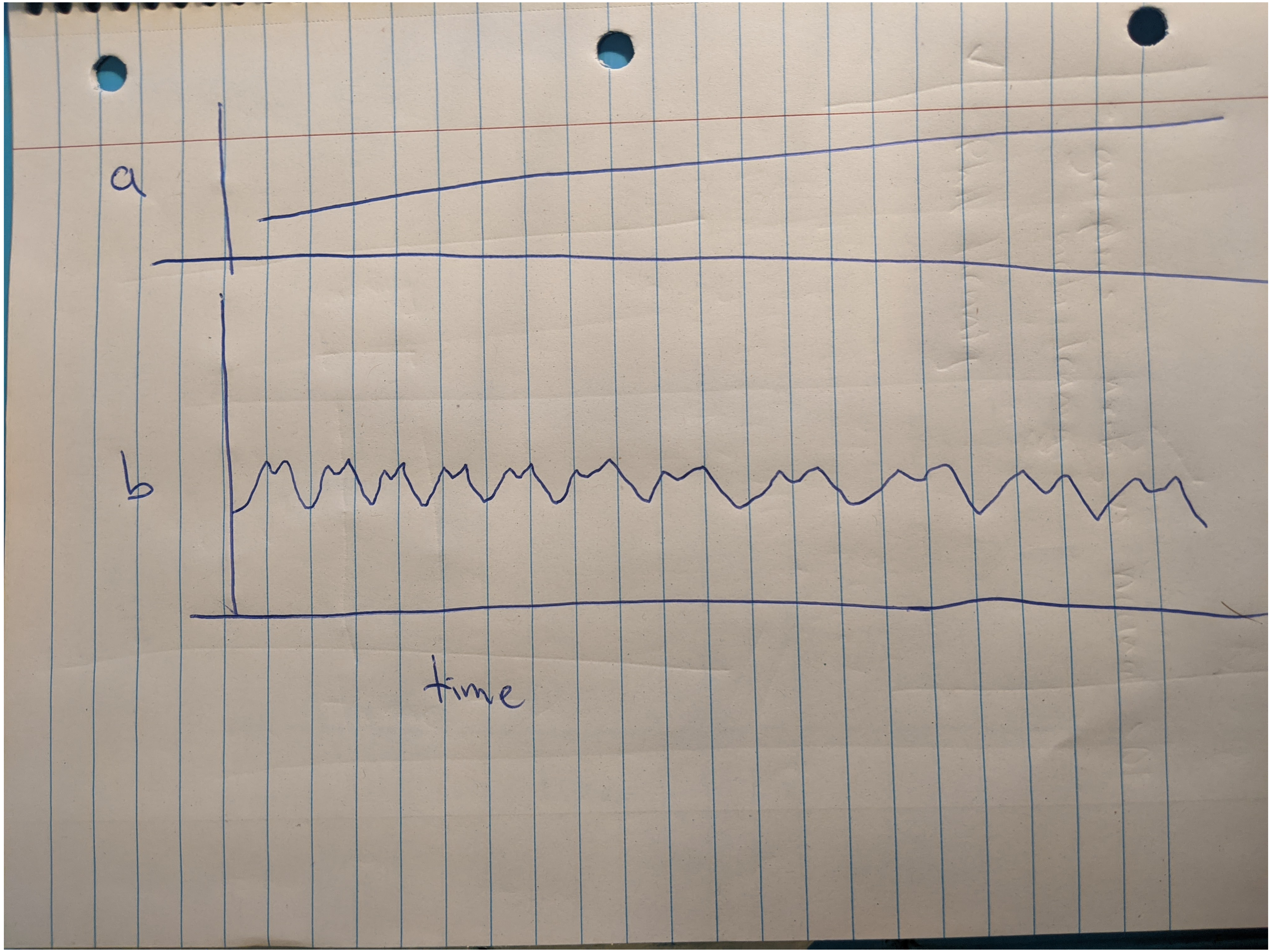
Example: Infant Electrocardiograms

- Jointly compute p(ECG of time N, ECG of time N + 10000)

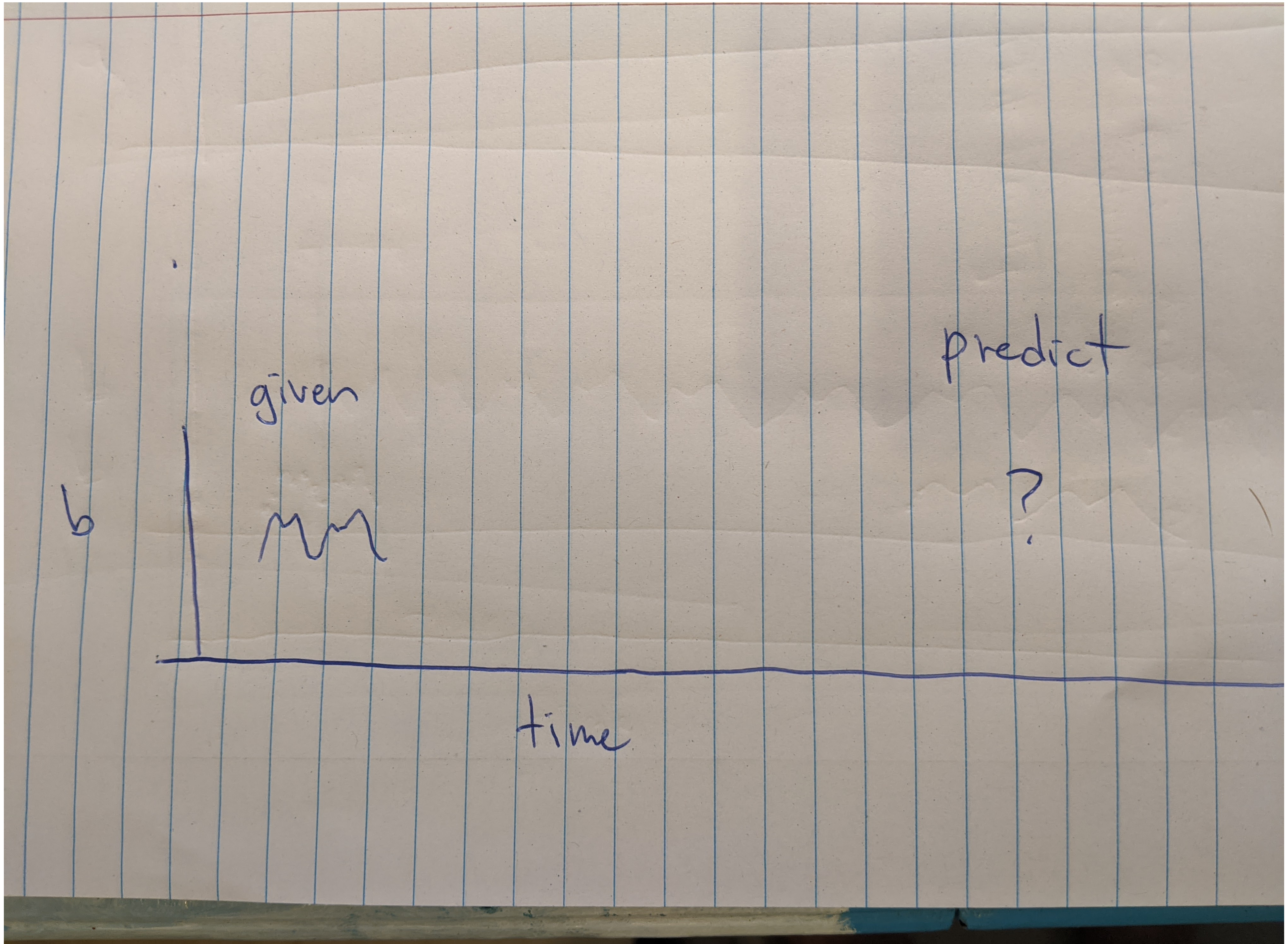
- Phase of ECG in between are irrelevant, heart size is sufficient statistic



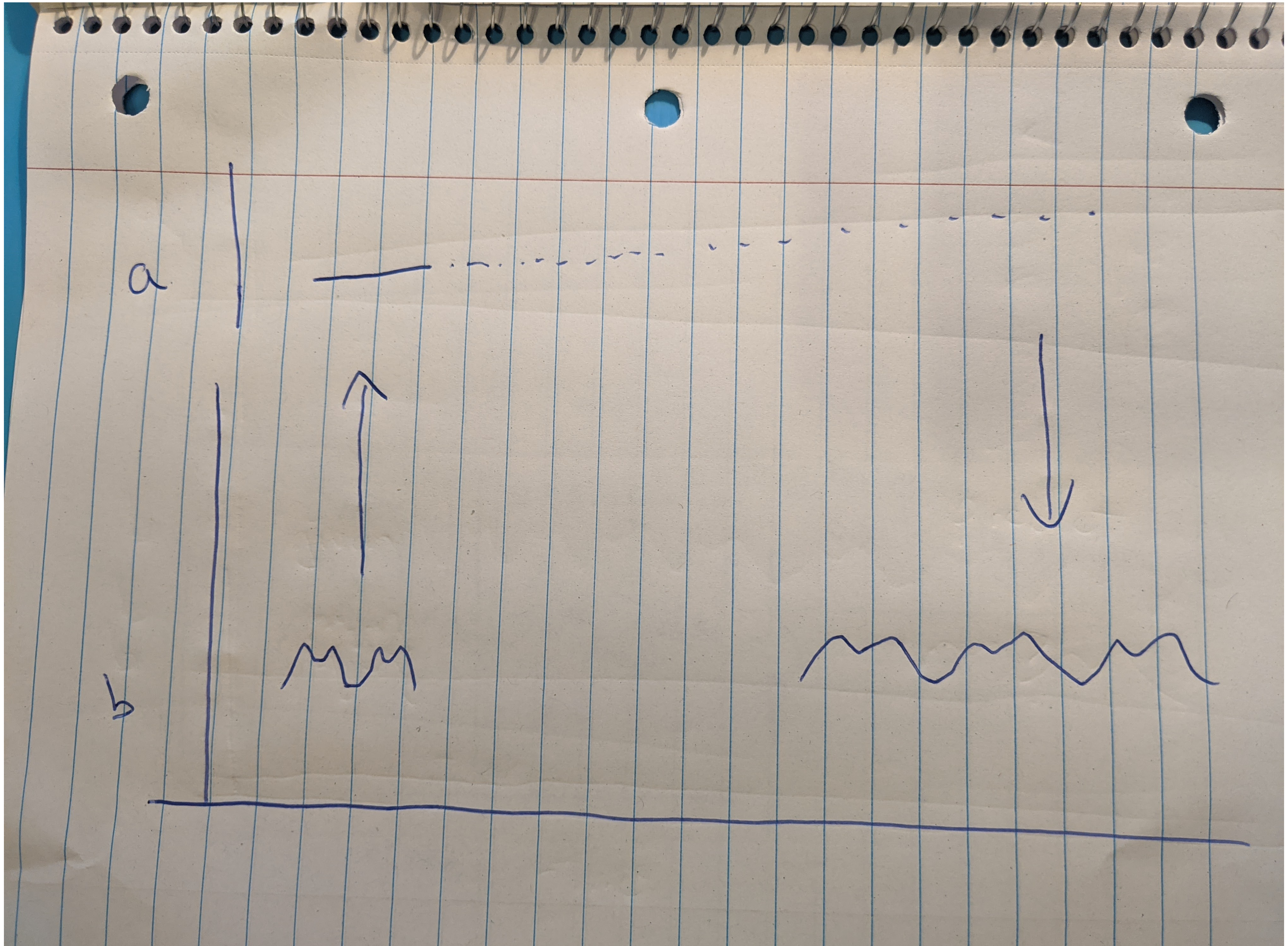
Example: Infant Electrocardiograms



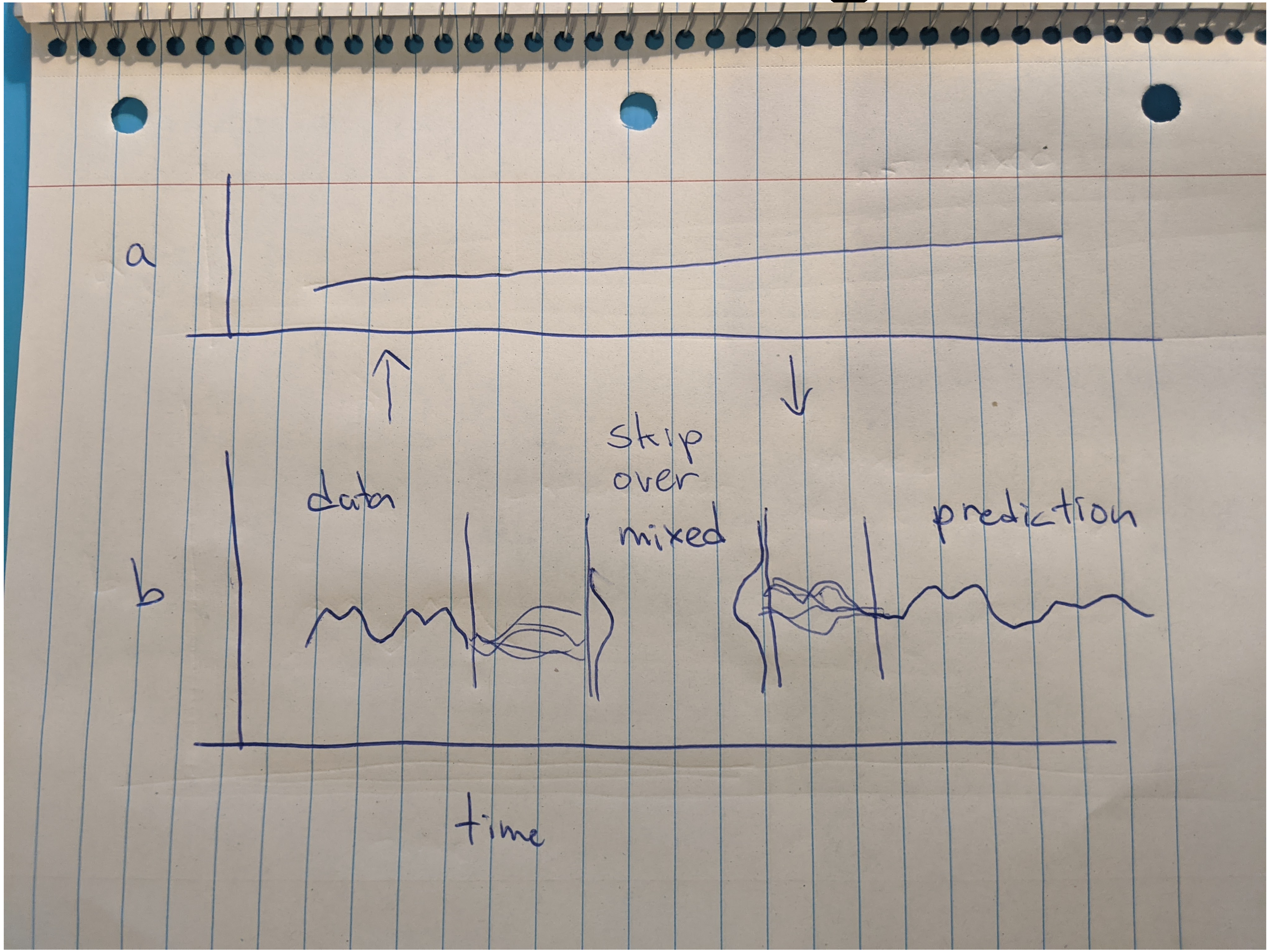
Example: Infant Electrocardiograms



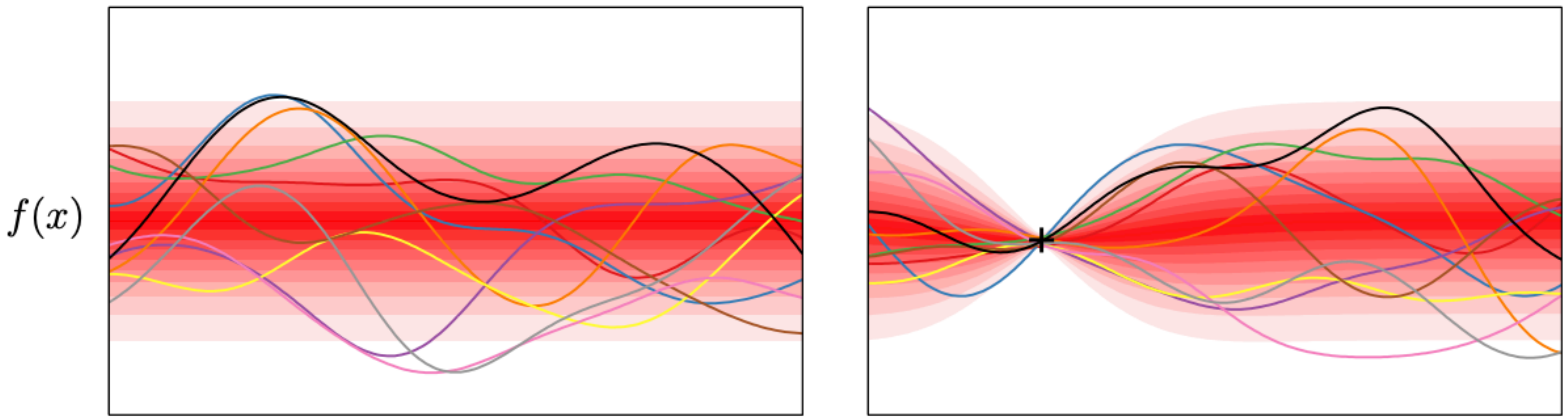
Example: Infant Electrocardiograms



Example: Infant Electrocardiograms



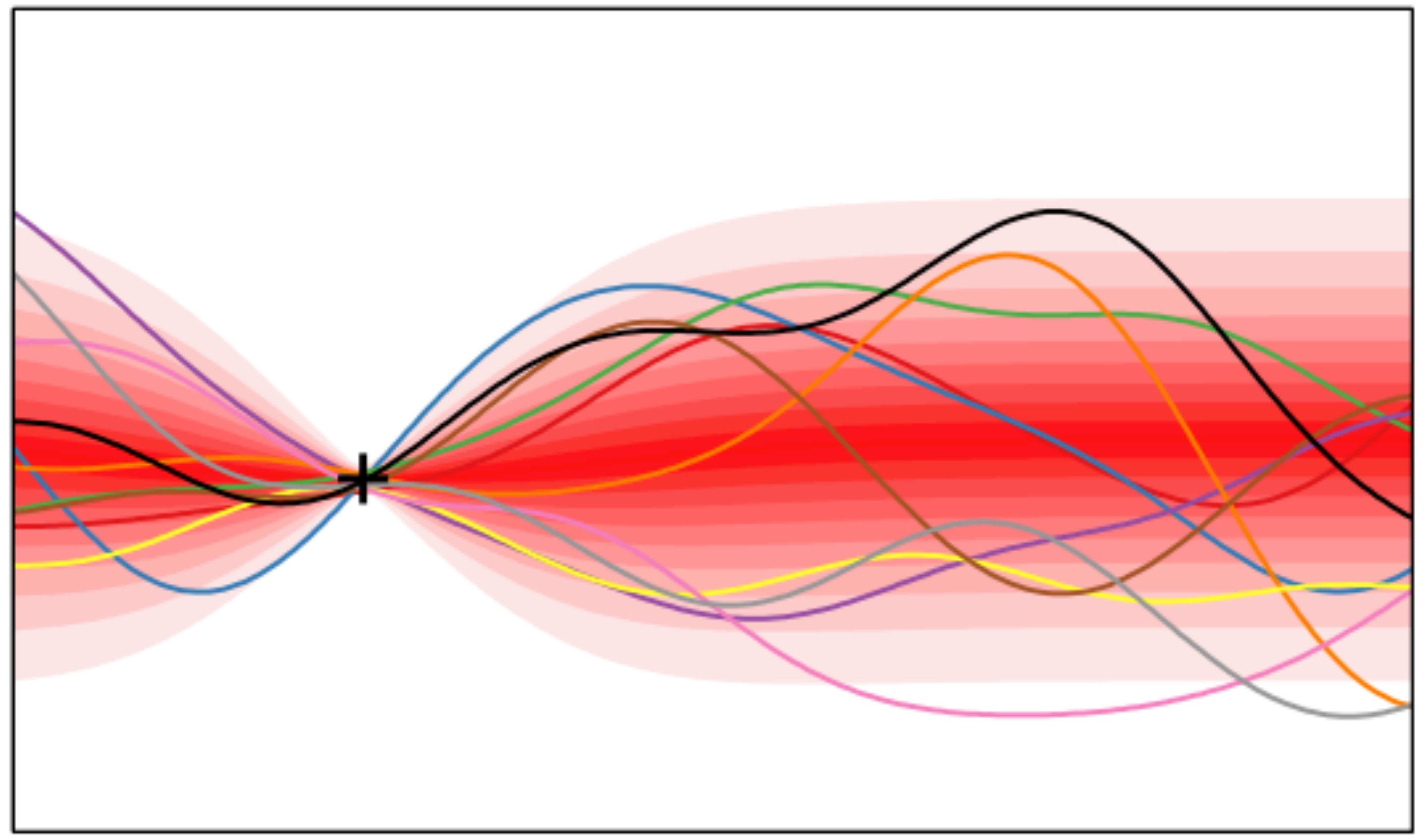
Hope #1: Low levels Mix Fast



- Away from observations, fine-grained details usually uncorrelated given high-level properties. I.e. conditional independence of fine given coarse
- E.g. in some GPs, we mix back to prior away from data
- Not always true (e.g. in computers) but that situation is always hard

Skipping over irrelevant details

- Expensive part is simulation of finer levels
- Away from data, these variables have KL of 0 given coarse grained trajectories!

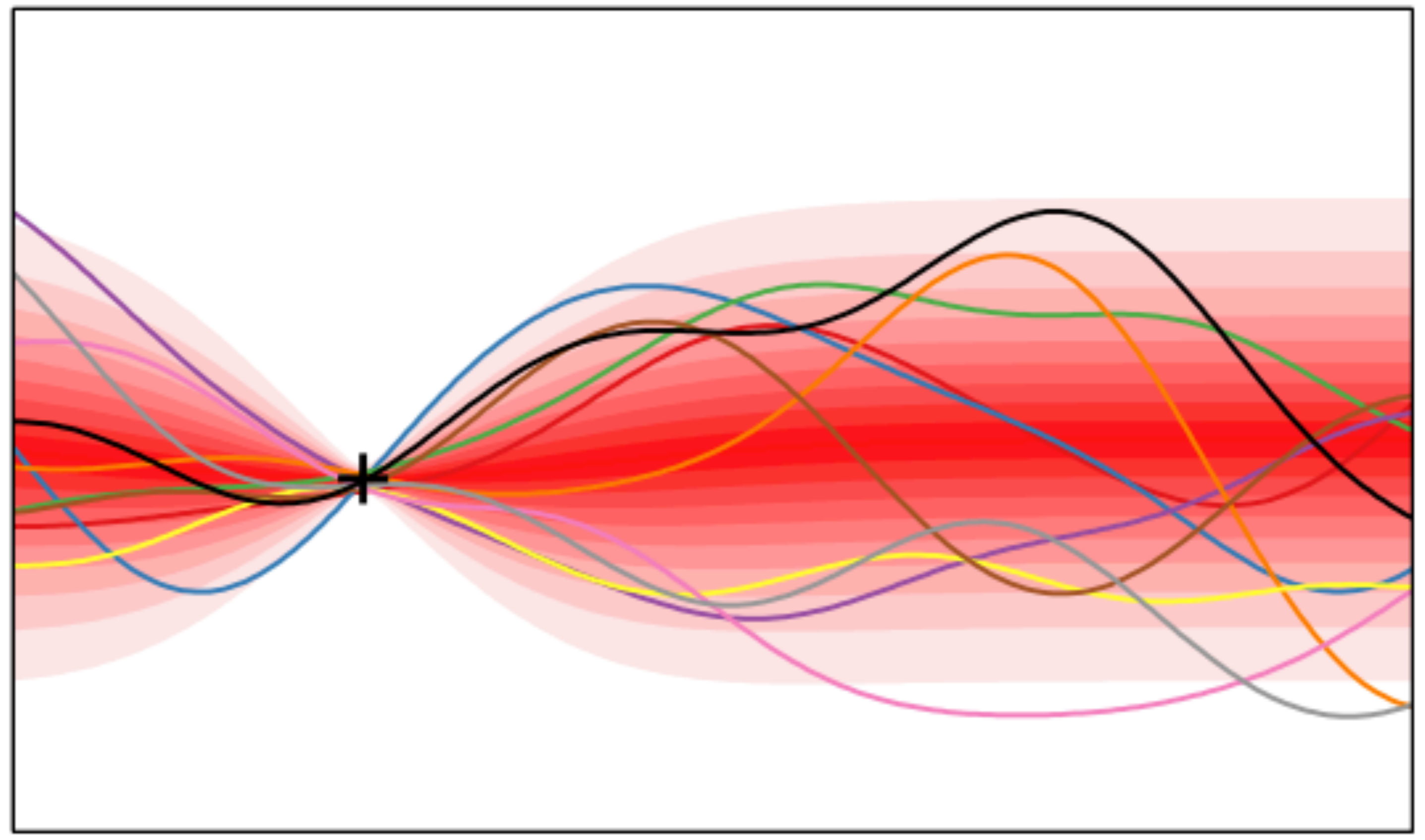


$$KL(q||p) = \mathbb{E}_{a(\cdot),b(\cdot)|a(\cdot)}\left[\underbrace{\int_0^T u(a(t))\ \mathrm{d}W_t}_{\text{coarse grained}} + \underbrace{\int_0^T u(a(t),b(t))\ \mathrm{d}W_t}_{\text{fine grained}}\right]$$

Skipping over irrelevant details

- Expensive part of ELBO is detailed simulation of finer levels away from data

- But these variables have KL of 0 given coarse grained trajectories!

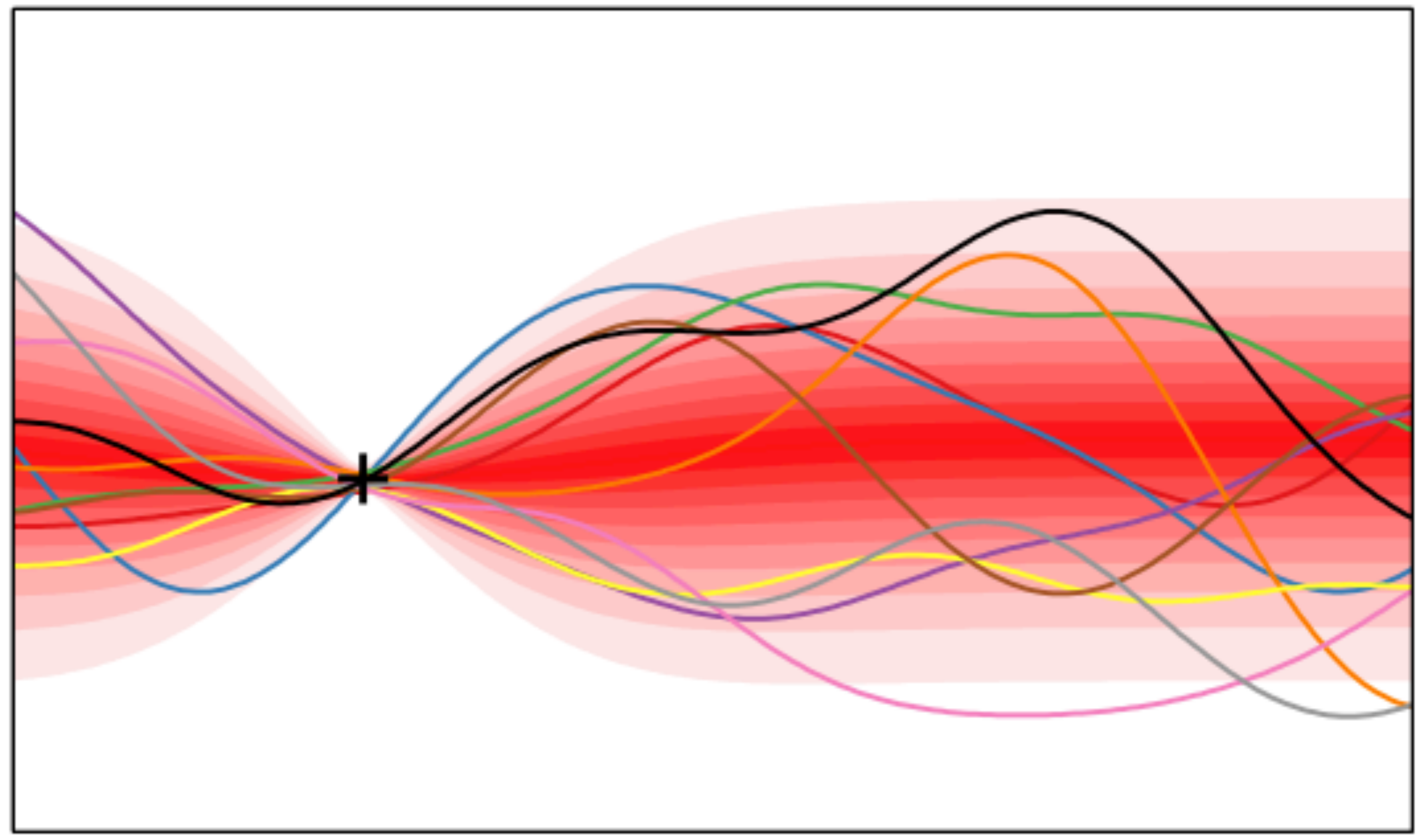


$$KL(q||p) = \underbrace{\mathbb{E}_{a(\cdot)}\left[\int_0^T u(a(t))\ \mathrm{d}W_t\right]}_{KL(q_a||p_a)} + \underbrace{\mathbb{E}_{a(\cdot),b(\cdot)|a(\cdot)}\left[\int_0^T u(a(t),b(t))\ \mathrm{d}W_t\right]}_{KL(q_b||p_b)}$$

Skipping over irrelevant details

- Expensive part of ELBO is detailed simulation of finer levels away from data

- But these variables have KL of 0 given coarse grained trajectories!

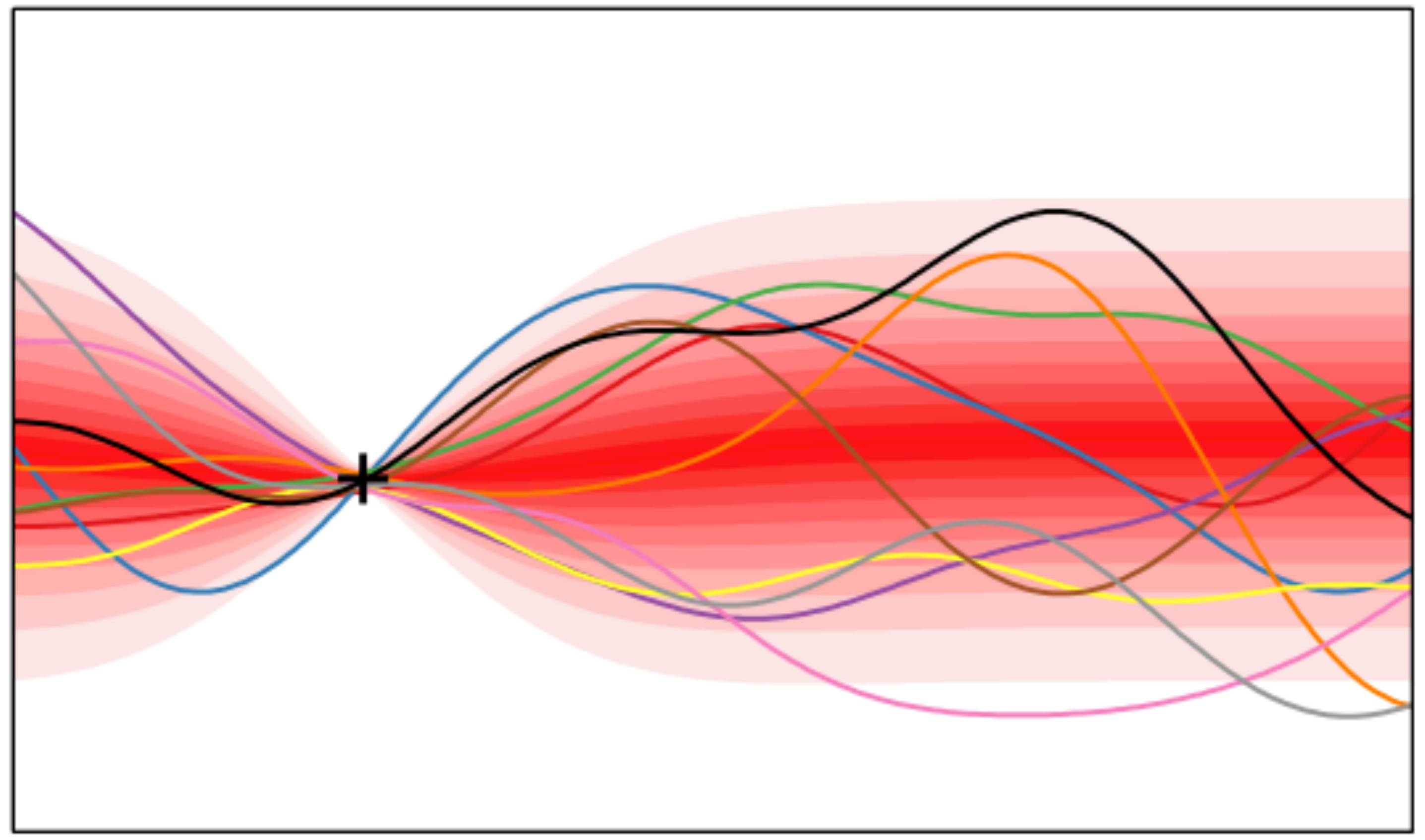


$$KL(q_b||p_b) = \mathbb{E}_{a(\cdot),b(\cdot)|a(\cdot)}\left[\underbrace{\int_{\text{unmixed}} u(a(t),b(t))\ \mathrm{d}W_t}_{\text{near data}}\right] + \mathbb{E}_{a(\cdot),b(\cdot)|a(\cdot)}\left[\underbrace{\int_{\text{mixed}} u(a(t),b(t))\ \mathrm{d}W_t}_{\text{away from data}}\right]$$

Skipping over irrelevant details

- Expensive part of ELBO is detailed simulation of finer levels away from data

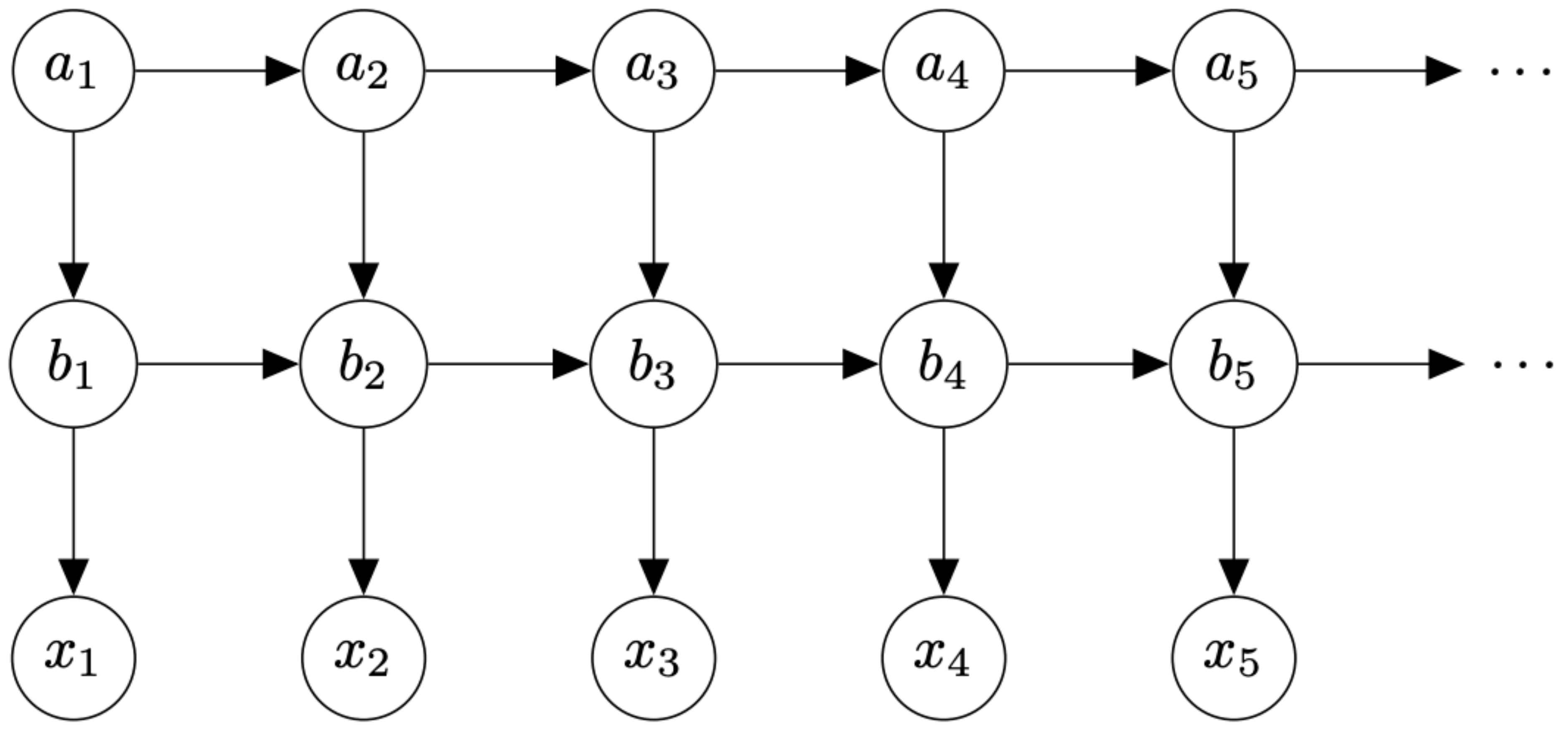
- But these variables have KL of 0 given coarse grained trajectories!



$$KL(q_b||p_b) = \mathbb{E}_{a(\cdot),b(\cdot)|a(\cdot)}\left[\underbrace{\int_{\text{unmixed}} u(a(t),b(t))\,\mathrm{d}W_t}_{\text{near data}}\right] + \cancel{\mathbb{E}_{a(\cdot),b(\cdot)|a(\cdot)}\left[\underbrace{\int_{\text{mixed}} u(a(t),b(t))\,\mathrm{d}W_t}_{\text{away from data}}\right]}$$

Refinement: Auxiliary Variables

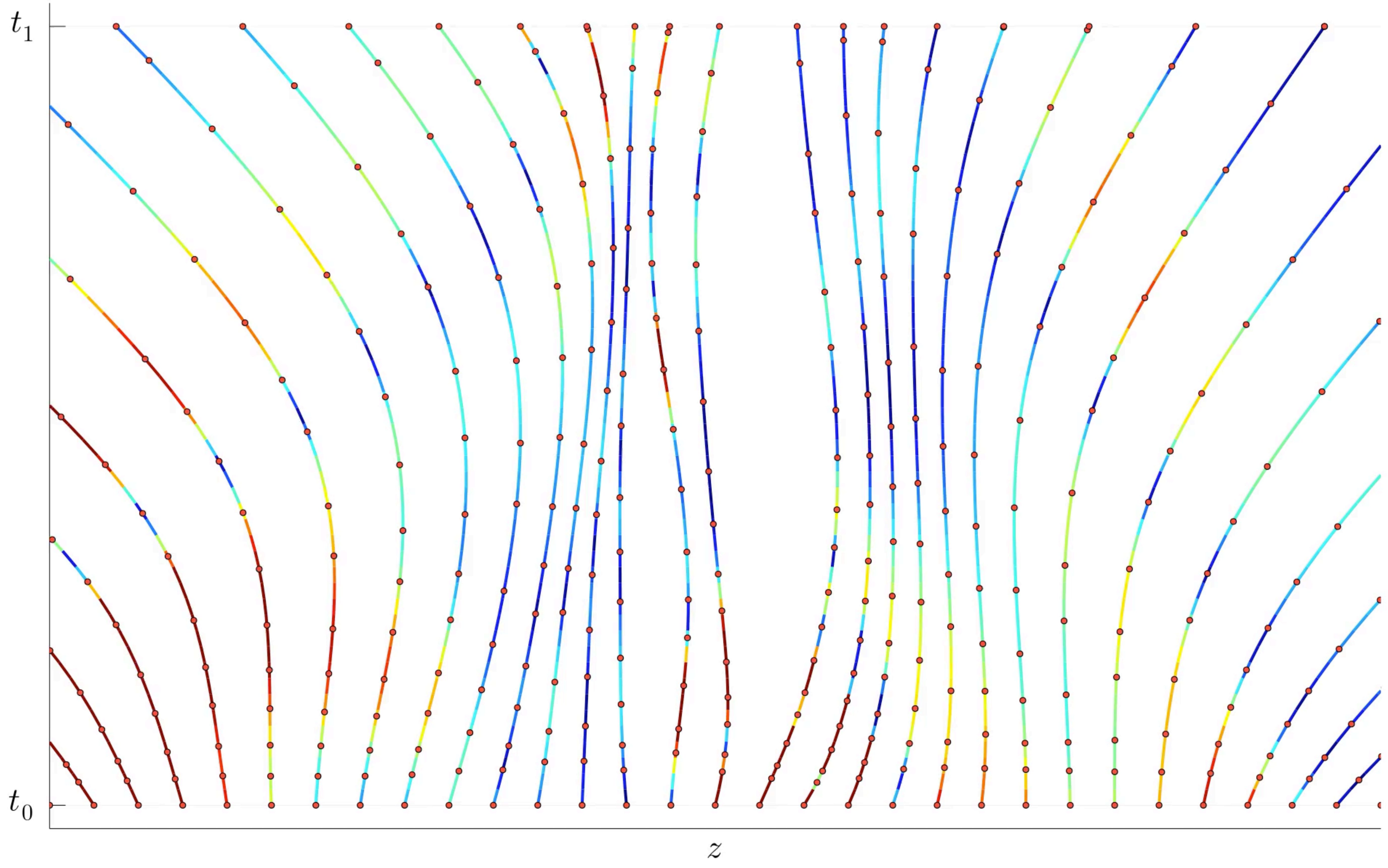
- Why are there extraneous coarse-grained variables in our model?
  - Should only exist in approximation.
  - E.g. In Ising model, temperature isn't a separate variable in model
- Answer: Put only fine-grained variables in model p, both sets in approx q
  - Standard trick in variational inference (auxiliary vars in variational dist)
- Can have as many time scales as we want.



Unsolved Problems

- How to estimate marginals to sample from when we ``fade back in''

- How to regularize approx. posterior dynamics to be fast to mix?



Learning Differential Equations that are Easy to Solve
Jacob Kelly*, Jesse Bettencourt*, Matthew Johnson, David Duvenaud

Xuechen Li, Winnie Xu, Leonard Wong, Ricky Chen, Yulia Rubanova,
David Duvenaud

Thanks!

VECTOR INSTITUTE

Connections to BNN theory

- "Liberty or Depth" (Farquhar, Smith, Gal, 2020): Infinite depth mean-field gives arbitrarily good predictive posteriors?

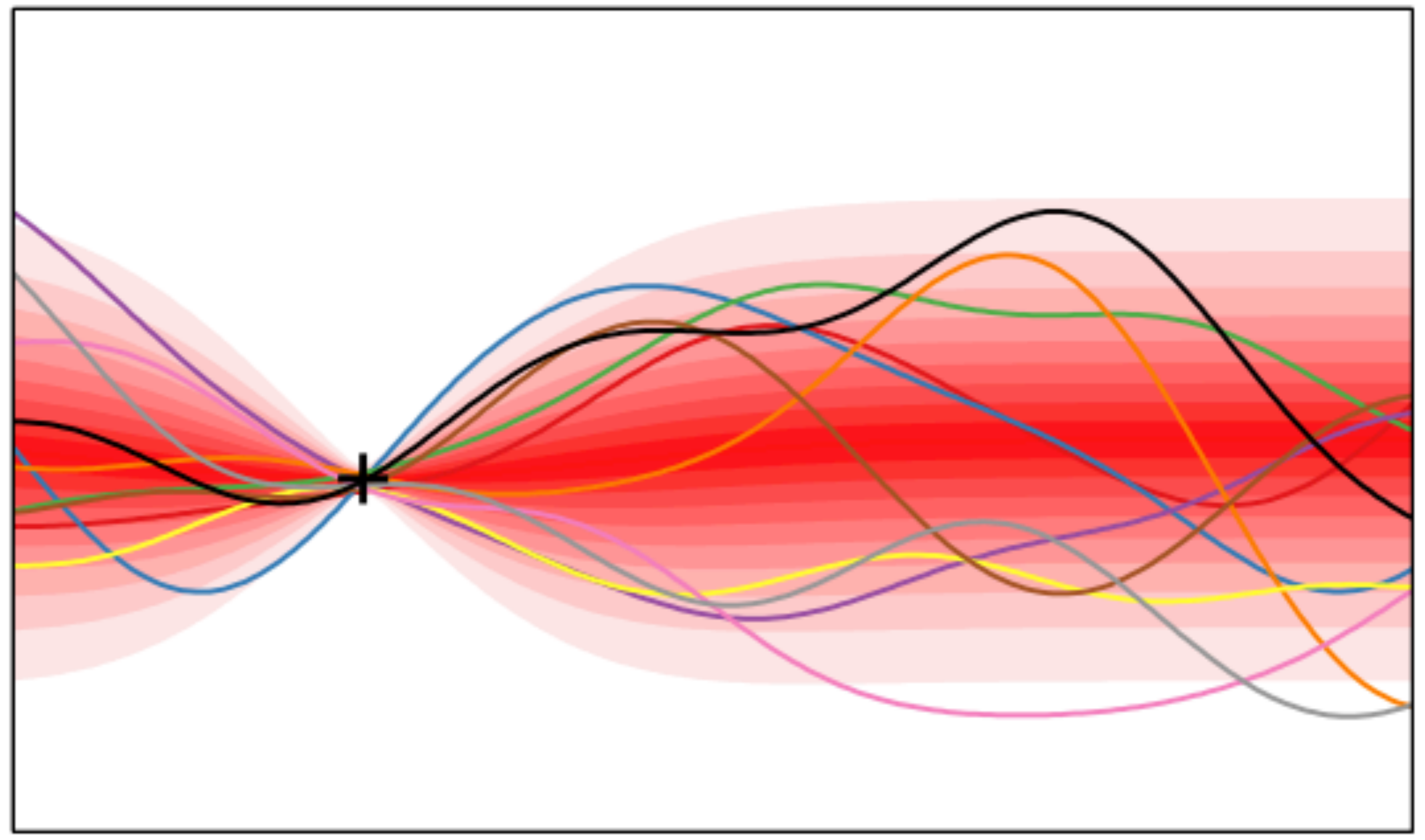
- Mean-field (Brownian motion) sufficient in SDEs for arbitrary expressiveness. But true and approximate posterior not Gaussian.

$$dz = f(z(t))dt + \sigma(z(t))dB(t)$$

Putting it all together:

- Break model into coarse (slow) and fine (fast) vars.
- When sampling:
  - Recognition nets look at local data and give posterior over coarse and fine variables
  - Sample entire coarse trajectory (only using approximate dynamics, never real ones!)
  - Sample fine trajectory starting just before and ending just after areas with data
  - Gives (almost) unbiased estimates of ELBO and predicted trajectories

$$
\begin{aligned}
da &= f_a(a(t))\,\mathrm{d}t \quad + \quad \sigma_a(a(t))\,\mathrm{d}W(t) \\
db &= f_b(a(t), b(t))\,\mathrm{d}t + \sigma_b(a(t), b(t))\,\mathrm{d}W(t)
\end{aligned}
$$



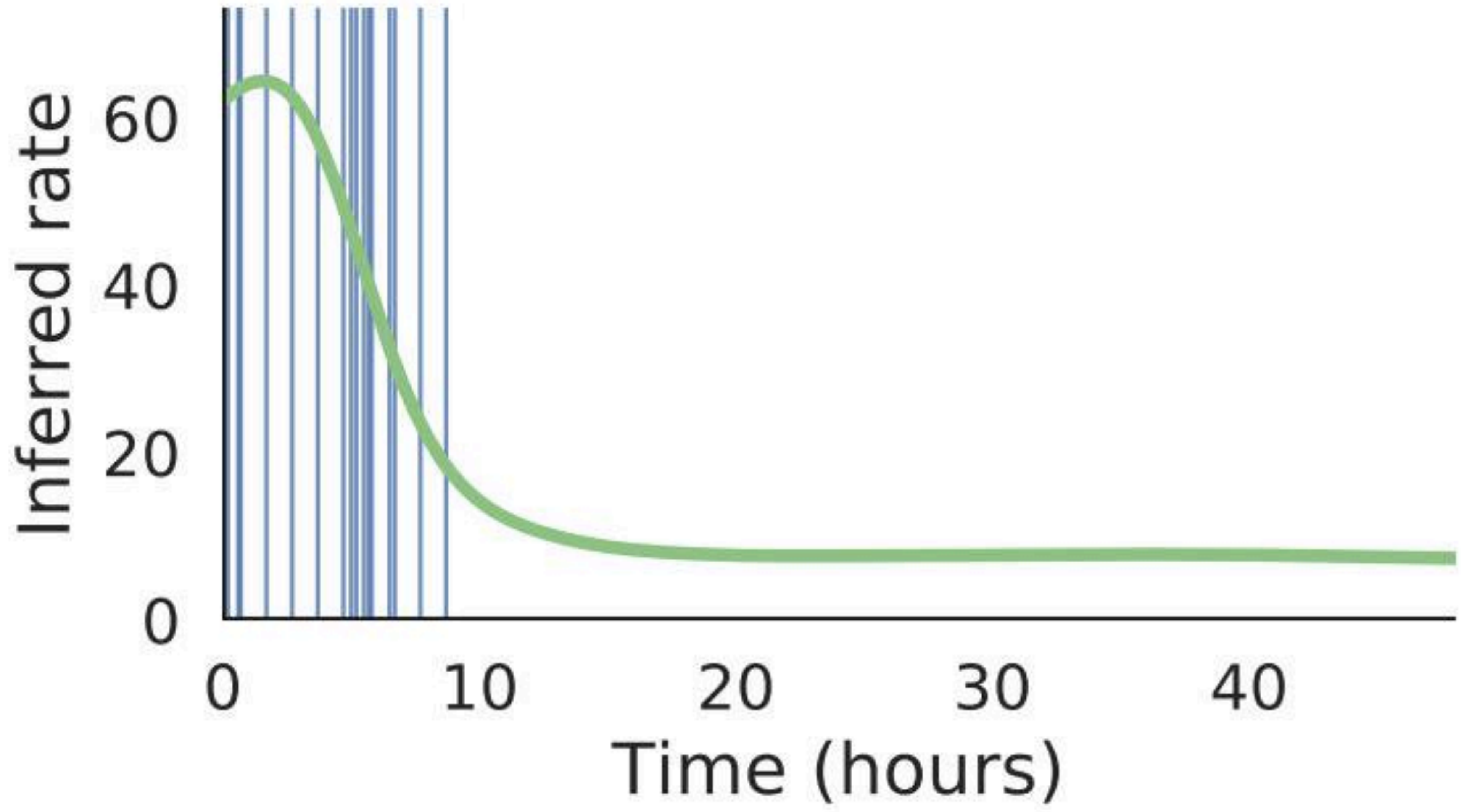
Poisson Process Likelihoods

$$\log p(t_1, \ldots, t_N | t_{\text{start}}, t_{\text{end}})$$

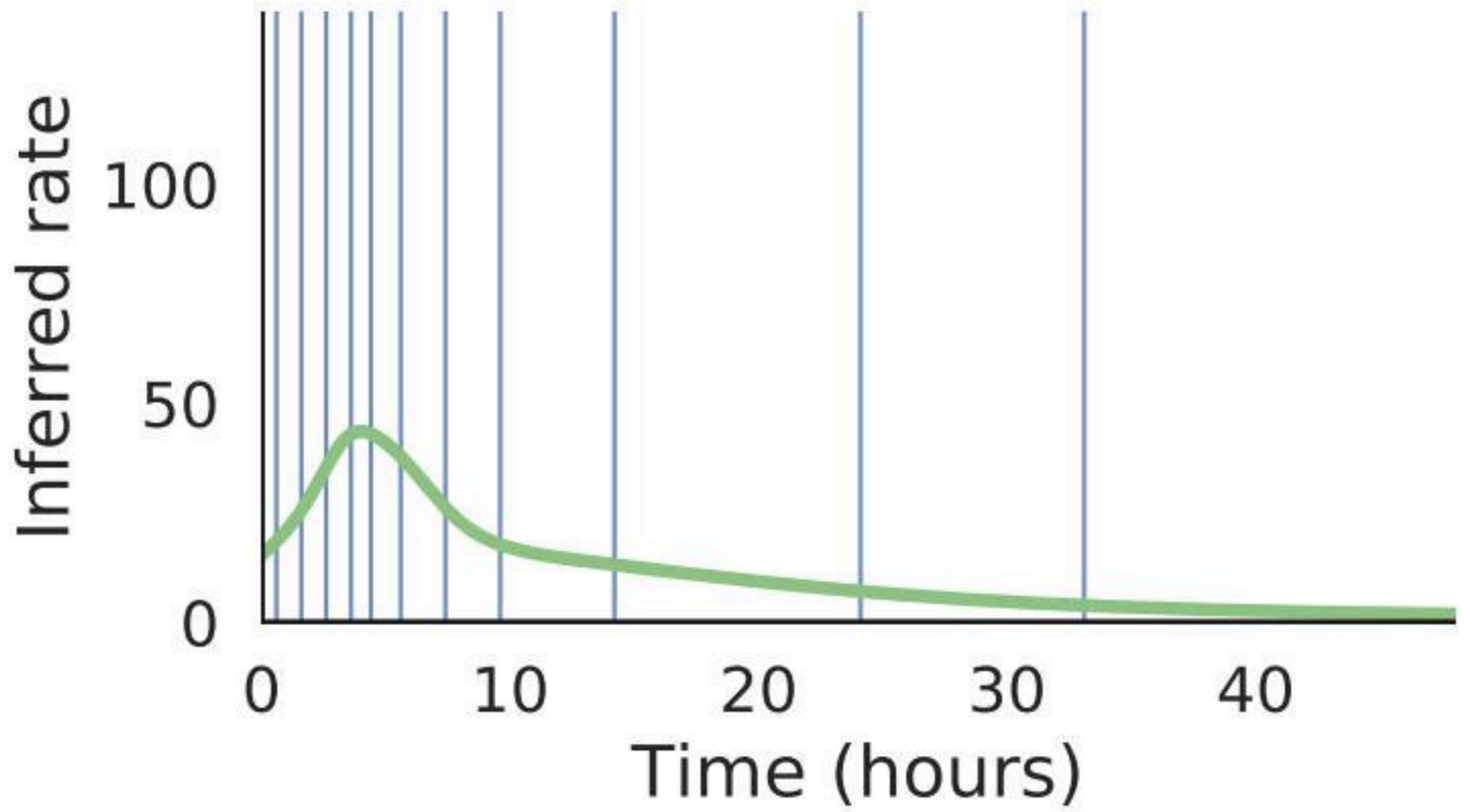
$$= \sum_{i=1}^{N} \log \lambda(\mathbf{z}(t_i)) - \int_{t_{\text{start}}}^{t_{\text{end}}} \lambda(\mathbf{z}(t)dt$$

- Model p(obs, time) instead of p(obs | time)
- Non-intervention model
- E.g. hurricanes

Diastolic arterial blood pressure



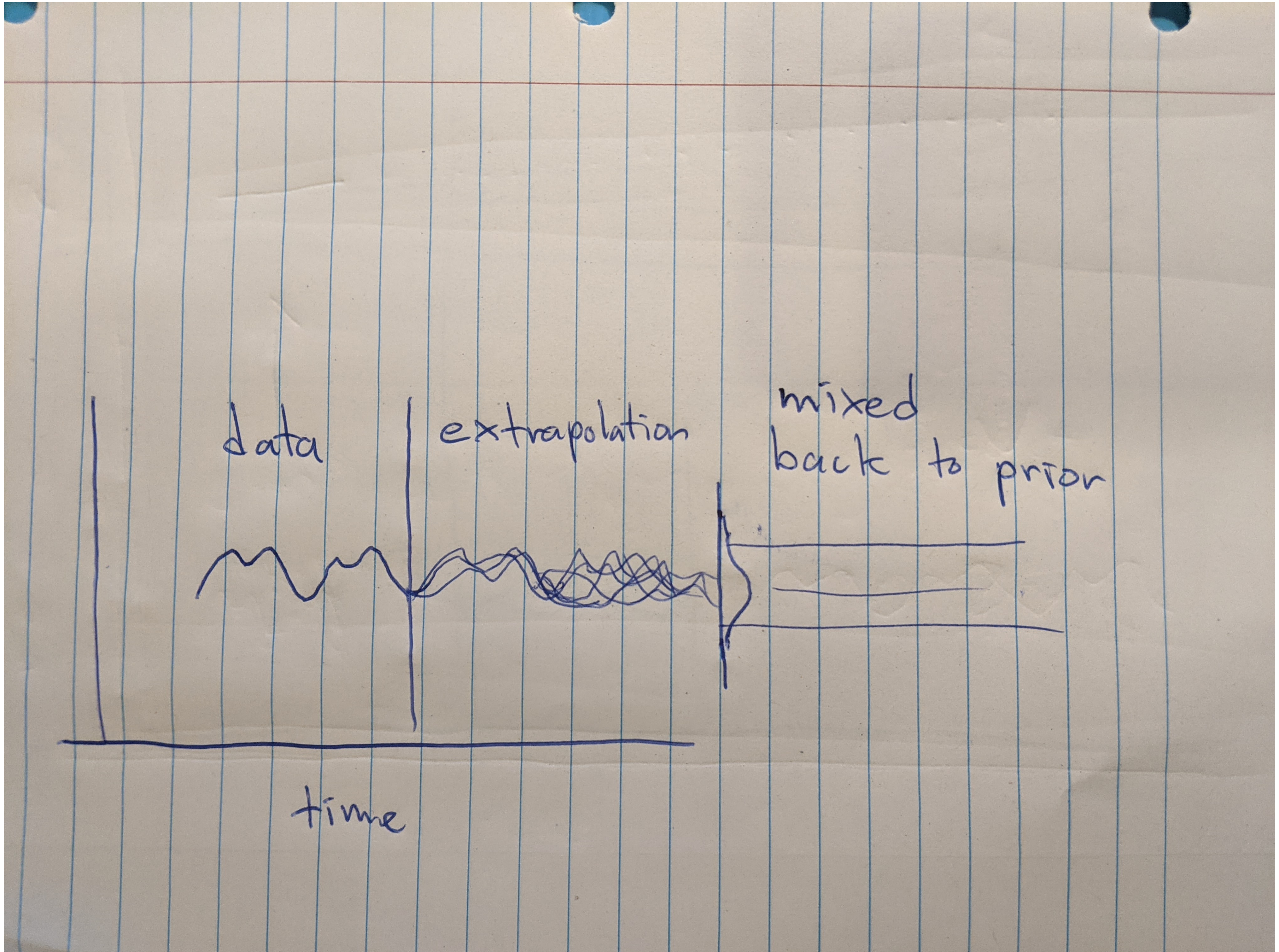
Partial pressure of arterial O2



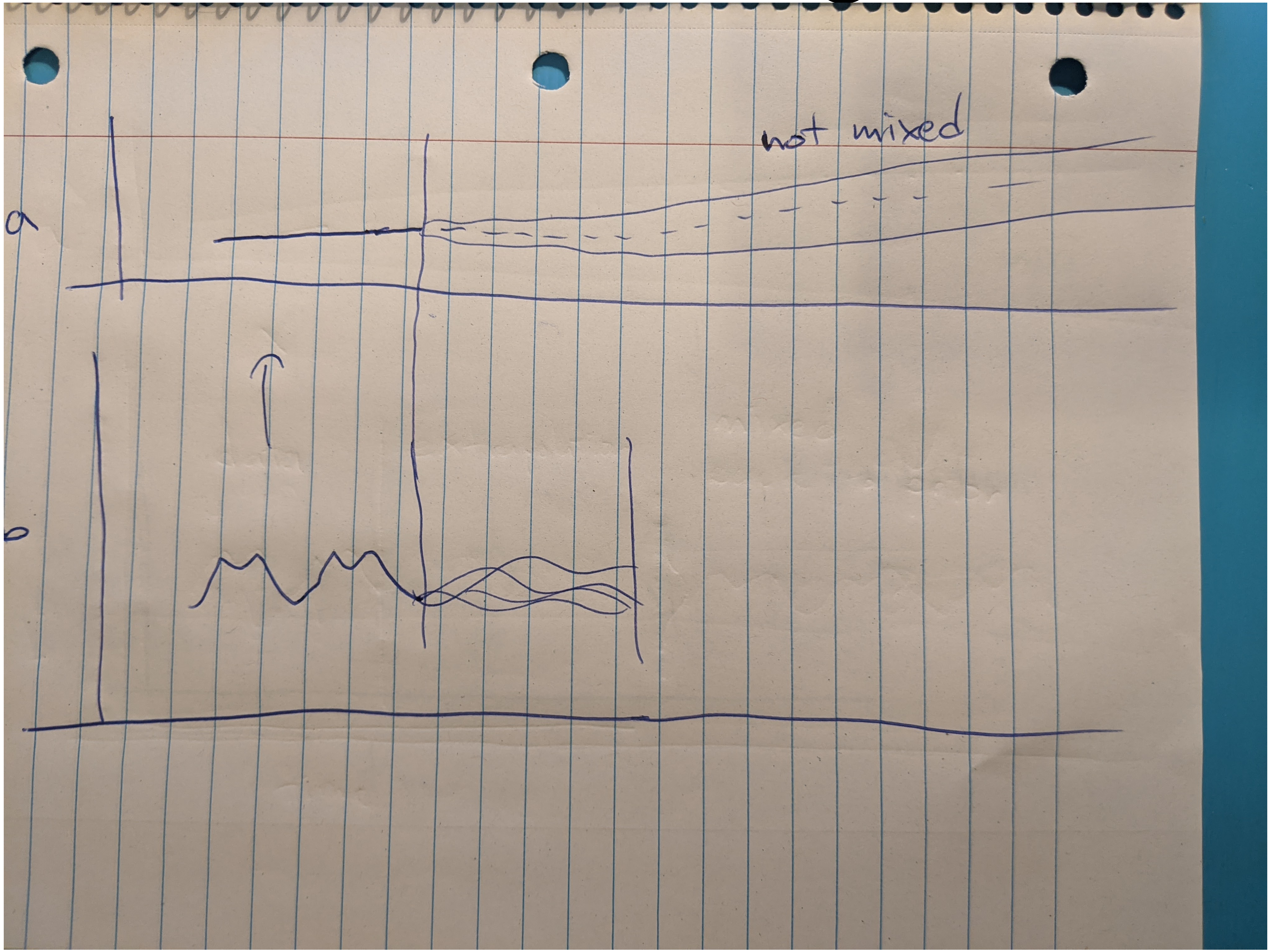
Achieving the dream

- Jointly learn true fine-grained expensive model and flexible approximation strategy from raw fine-grained data.
- Auxiliary coarse-grained variables might be interpretable
- Can combine high-level and low-level info automatically?

Example: Infant Electrocardiograms



Example: Infant Electrocardiograms



Dex: a typed array language built for speed

```
def map (f : a->b) (xs : n=>a) : n=>b =
  for i. f x.i
```

Flexibility

- Ragged and sparse arrays
- Algebraic data types (e.g. `Value|NaN|Missing`)

Correctness

- Dependent types for compile-time debugging (e.g. shape checking)
- Composable, zero-cost abstractions (e.g. run on any vector space)

Performance

- Fast nested loops + gradients (e.g. CTC loss)
- CPU, GPU, TPU backends, JAX interop

Ray tracer written in Dex
google-research.github.io/dex-lang/raytrace.html

Related work 1

- Tzen + Raginski: Deep LVMs become SDEs in the limit. Variational inf framework. Forward-mode autodiff.

- Peluchetti + Favaro: Worked out SDE corresponding to infinitely-deep convnets with uncertain weights

- Jia + Benson: Added countably many discrete jumps to latent ODEs

Related work 2

- Thomas Ryder, Andrew Golightly, A Stephen Mc-Gough, and Dennis Prangle. *Black-box variational inference for stochastic differential equations.*
- Pashupati Hegde, Markus Heinonen, Harri Lähdesmäki, and Samuel Kaski. *Deep learning with differential gaussian process flows.*
- Markus Heinonen, Cagatay Yildiz, Henrik Mannerström, Jukka Intosalmi, and Harri Lähdesmäki. *Learning unknown ODE models with gaussian processes.*
- *C. Garcıa, A. Otero, P. Felix, J. Presedo, and D. Marquez. Nonparametric estimation of stochastic differential equations with sparse Gaussian processes.*

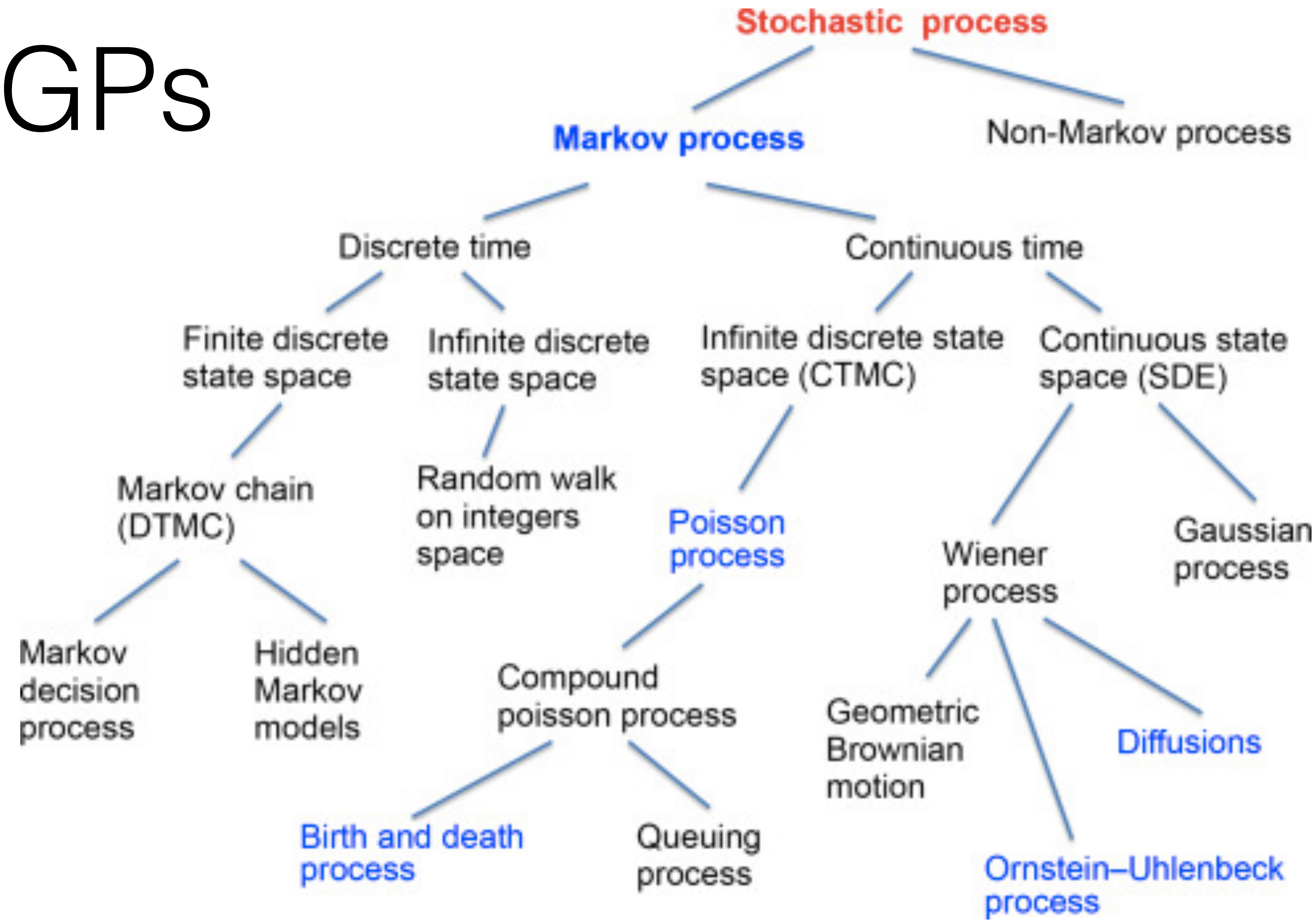
- All use Euler discretizations. Not clear what limiting algorithm is (e.g. enforces invariants?), and not memory-efficient.
- Not even going to discuss methods that require solving a PDE - not scalable.
- We want to use adaptive, (high-order?) SDE solvers.

Limitations

- SDE solvers generally lower-order convergence than ODE solvers
  - (e.g. Milstein order 1 vs RK4)
- Non-diagonal noise requires Levy areas
  - Diagonal noise requires funny parameterization
- Need jump-style noise? (e.g. hit by a car)
- Only one input dimension (unlike GPs)

SDEs vs GPs

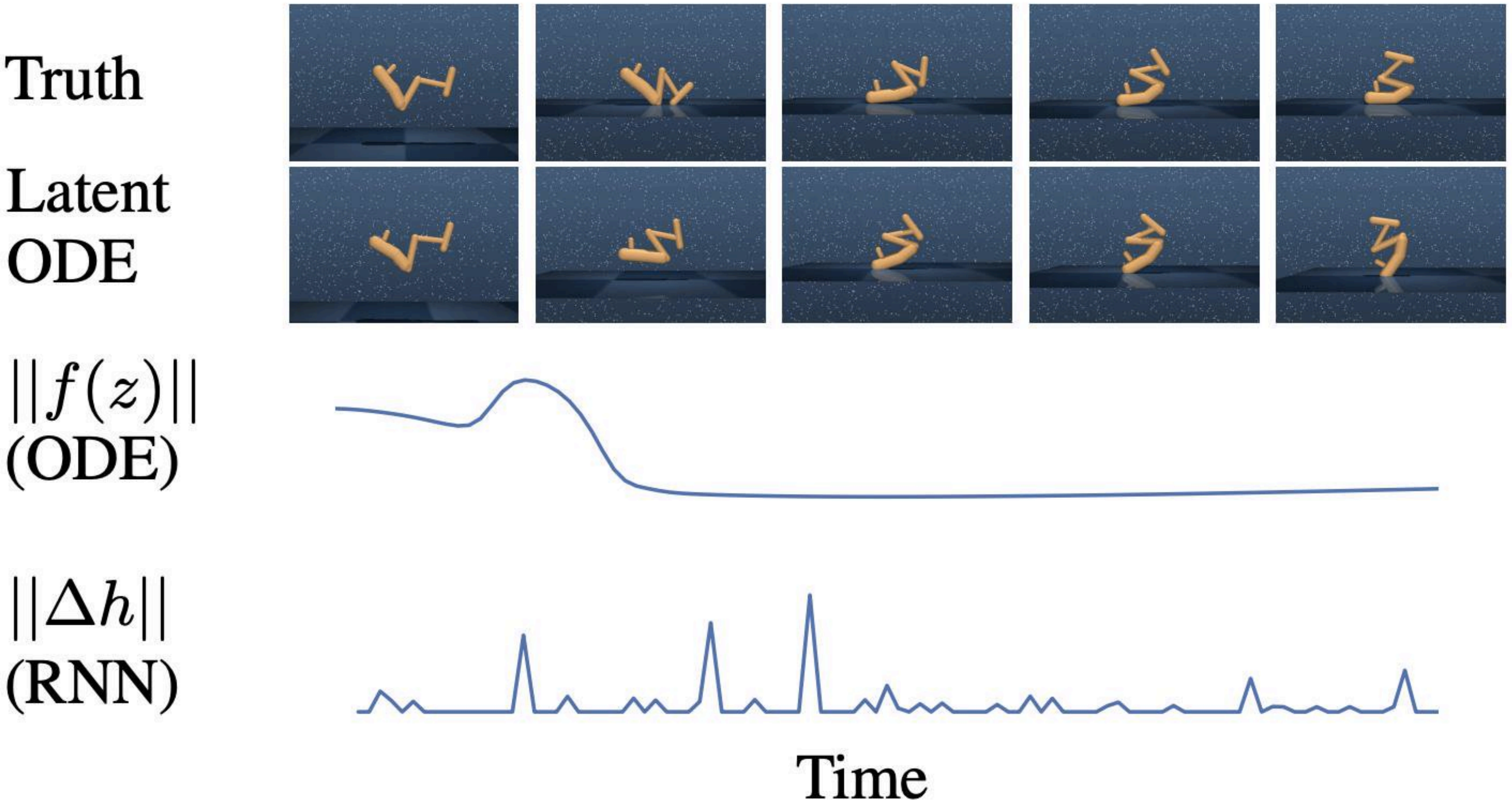
- Distinct sets of priors over functions
- Easy to construct non-Gaussian SDE



From “Handbook of Statistics”, Mubayi et al, 2019.
Line means "can be used to construct", but not "contains"

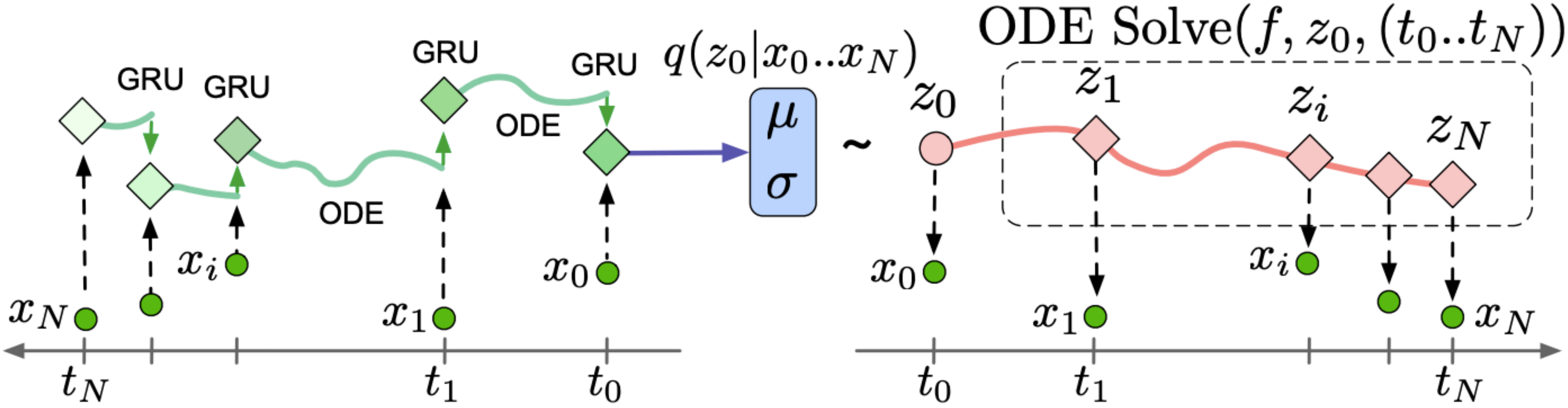
Mujoco: State versus Belief states

- States are more interpretable than belief states
- True dynamics are deterministic



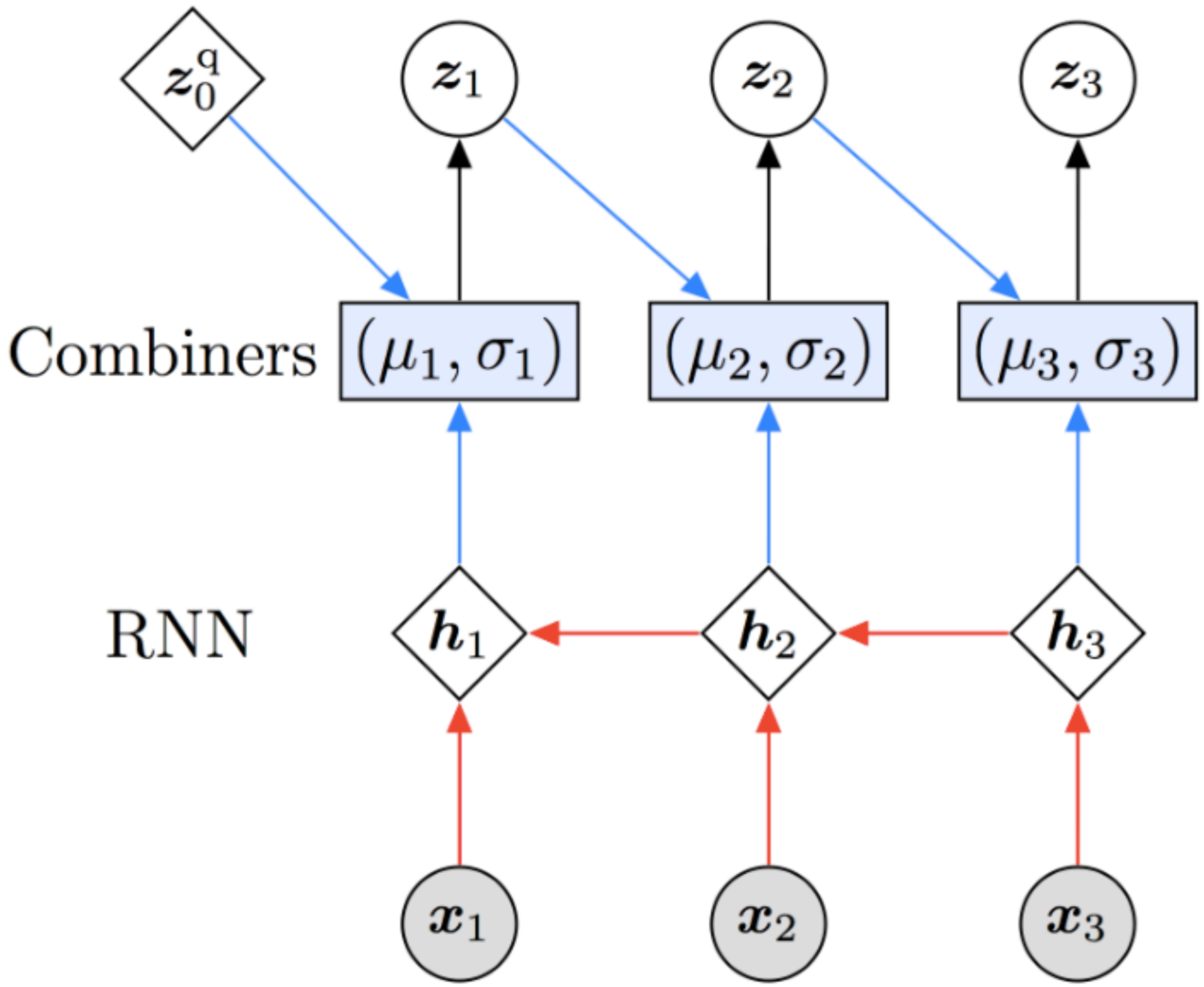
An ODE latent-variable model

- Can do VAE-style inference with an RNN encoder



Latent variable models

- Can use a neural net to guess optimal variational params from data
- Structure of recognition net an implementation detail
  - Only there to speed things up.
  - Just needs to output a normalized distribution over z

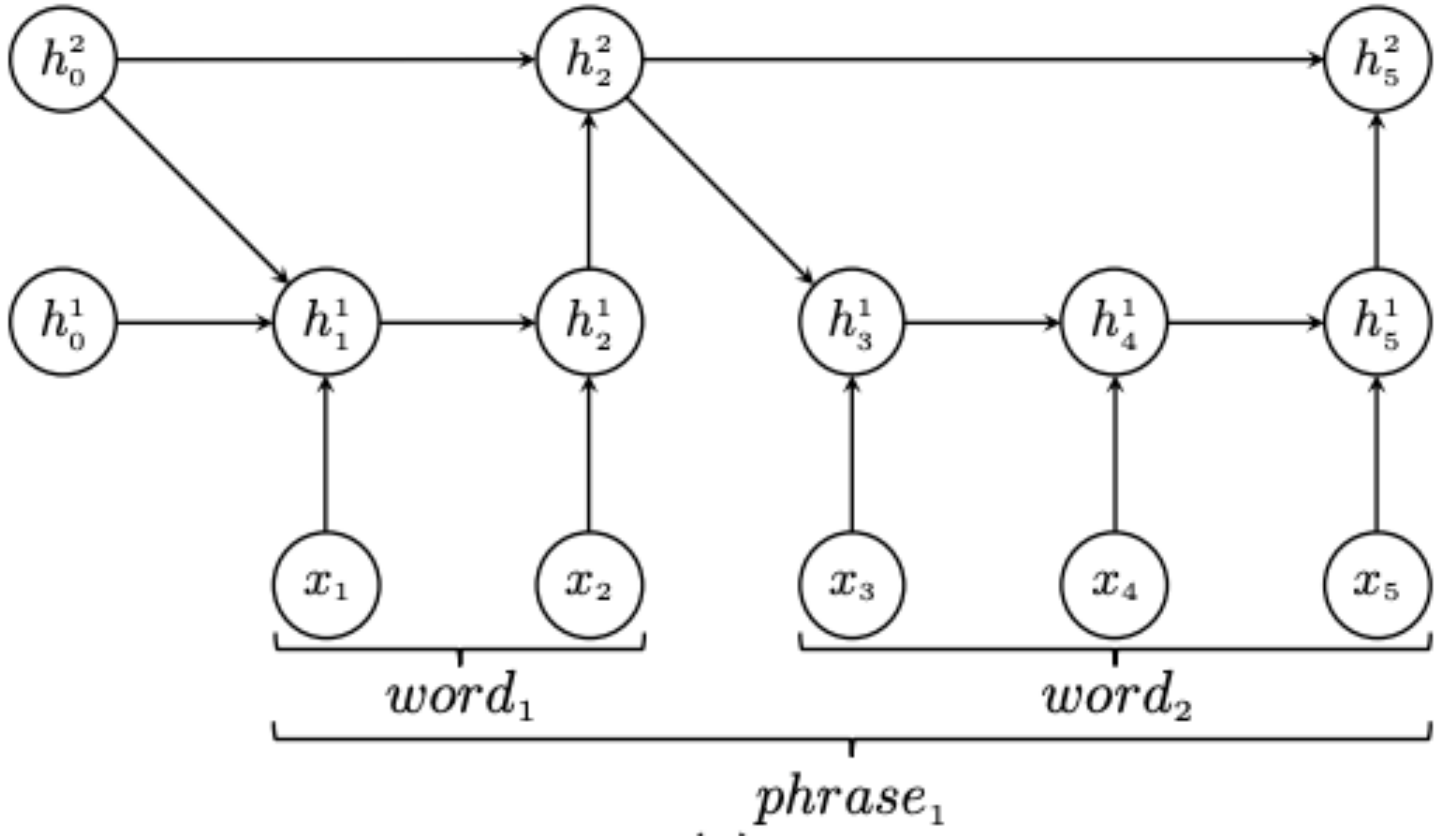


https://pyro.ai/examples/dmm.html

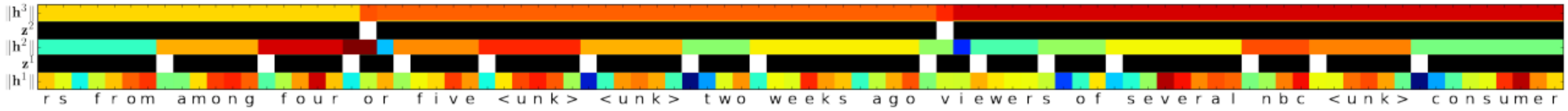
Multi-scale RNNs: 2016

HIERARCHICAL MULTISCALE RECURRENT NEURAL NETWORKS

Junyoung Chung, Sungjin Ahn & Yoshua Bengio *



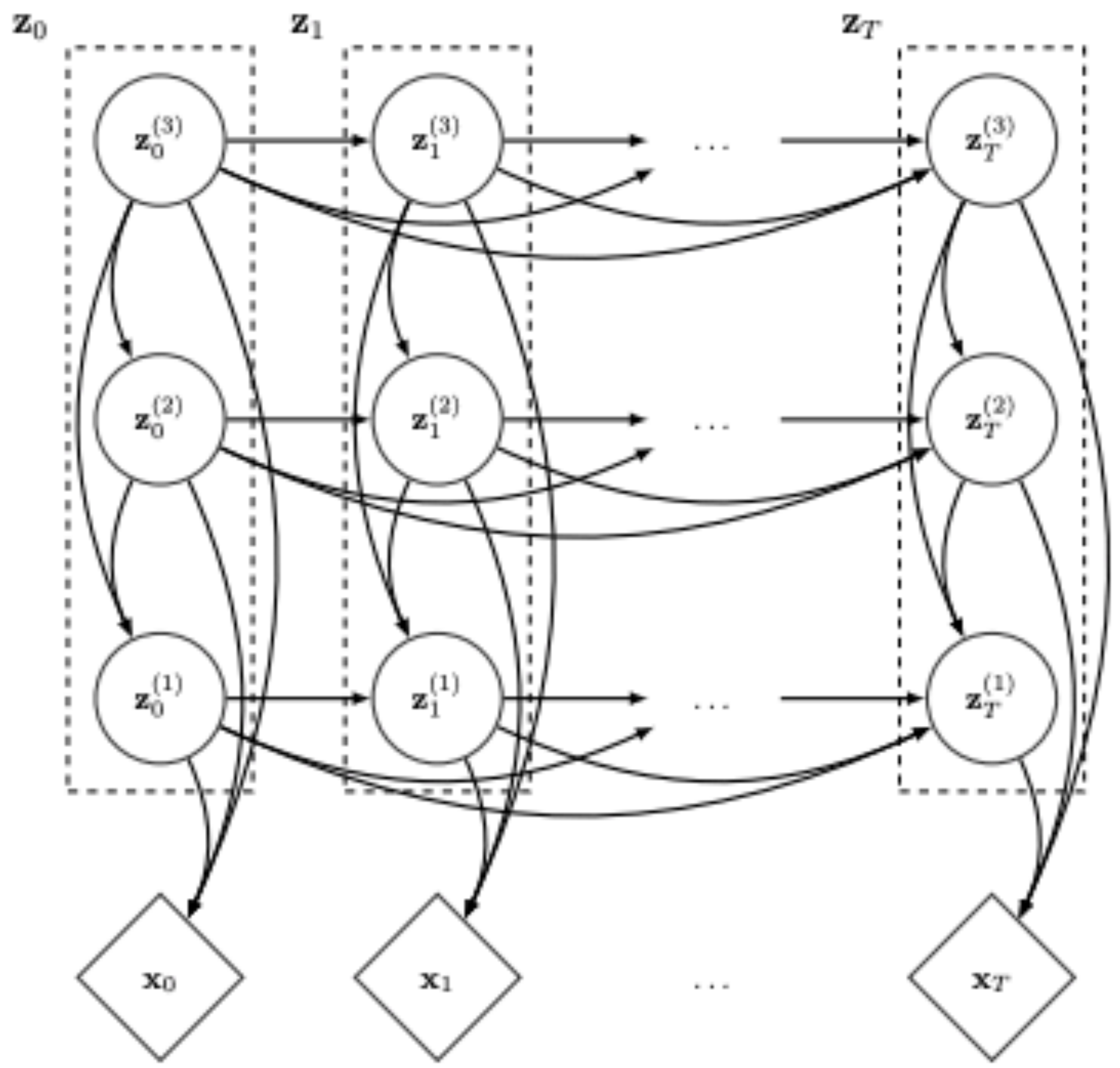
Penn Treebank Line 3



Multi-scale Markov models: 2020

VIDEOFLOW: A CONDITIONAL FLOW-BASED MODEL FOR STOCHASTIC VIDEO GENERATION

Manoj Kumar*, Mohammad Babaeizadeh, Dumitru Erhan, Chelsea Finn, Sergey Levine, Laurent Dinh, Durk Kingma
Google Research, Brain Team



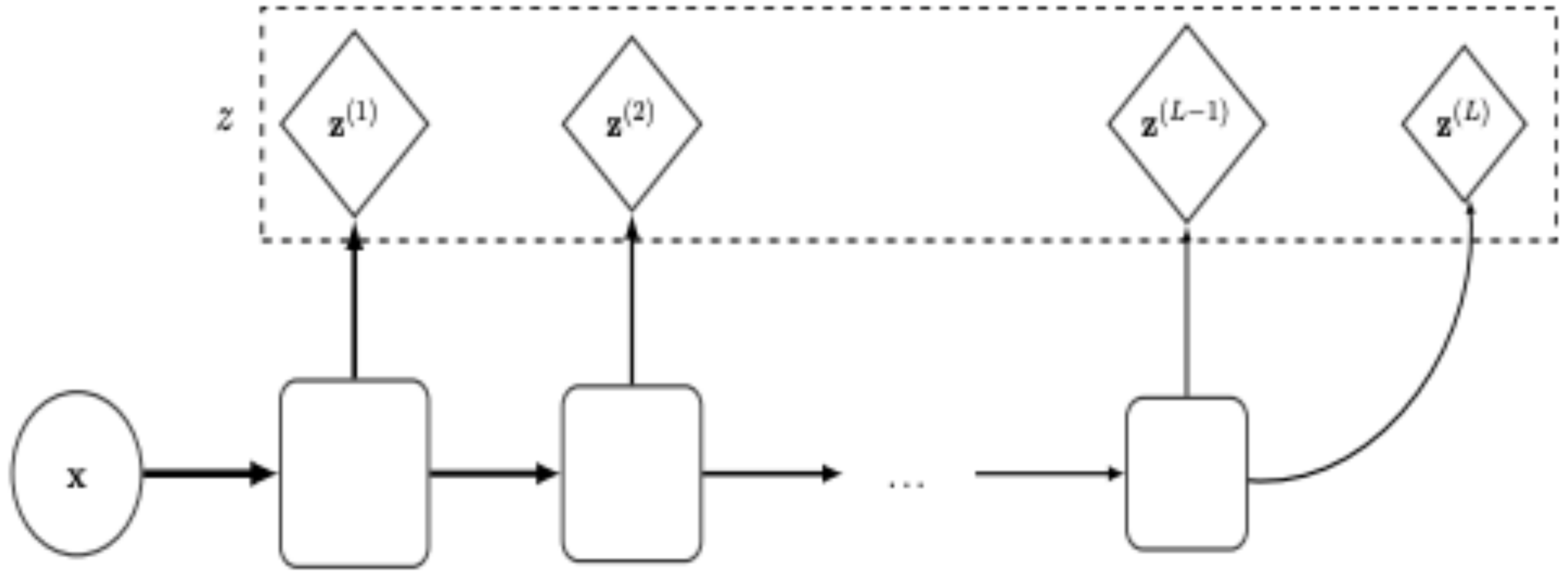


Figure 1: Left: Multi-scale prior The flow model uses a multi-scale architecture using several levels of stochastic variables. **Right: Autoregressive latent-dynamic prior** The input at each timestep $\mathbf{x}_t$ is encoded into multiple levels of stochastic variables $(\mathbf{z}_t^{(1)}, \ldots, \mathbf{z}_t^{(L)})$. We model those levels through a sequential process $\prod_t \prod_l p(\mathbf{z}_t^{(l)} \mid \mathbf{z}_{<t}^{(l)}, \mathbf{z}_t^{(>l)})$.

Problems with Discrete Time

- Need to choose discretizations, mixing times without gradients
  - Probably want state-dependent step sizes
- Finest scale determined by sampling rate
- Can't apply to irregularly sampled data easily