Flow Matching (Diffusion Models)

CS 180 Fall 2025
Angjoo Kanazawa and Alexei Efros

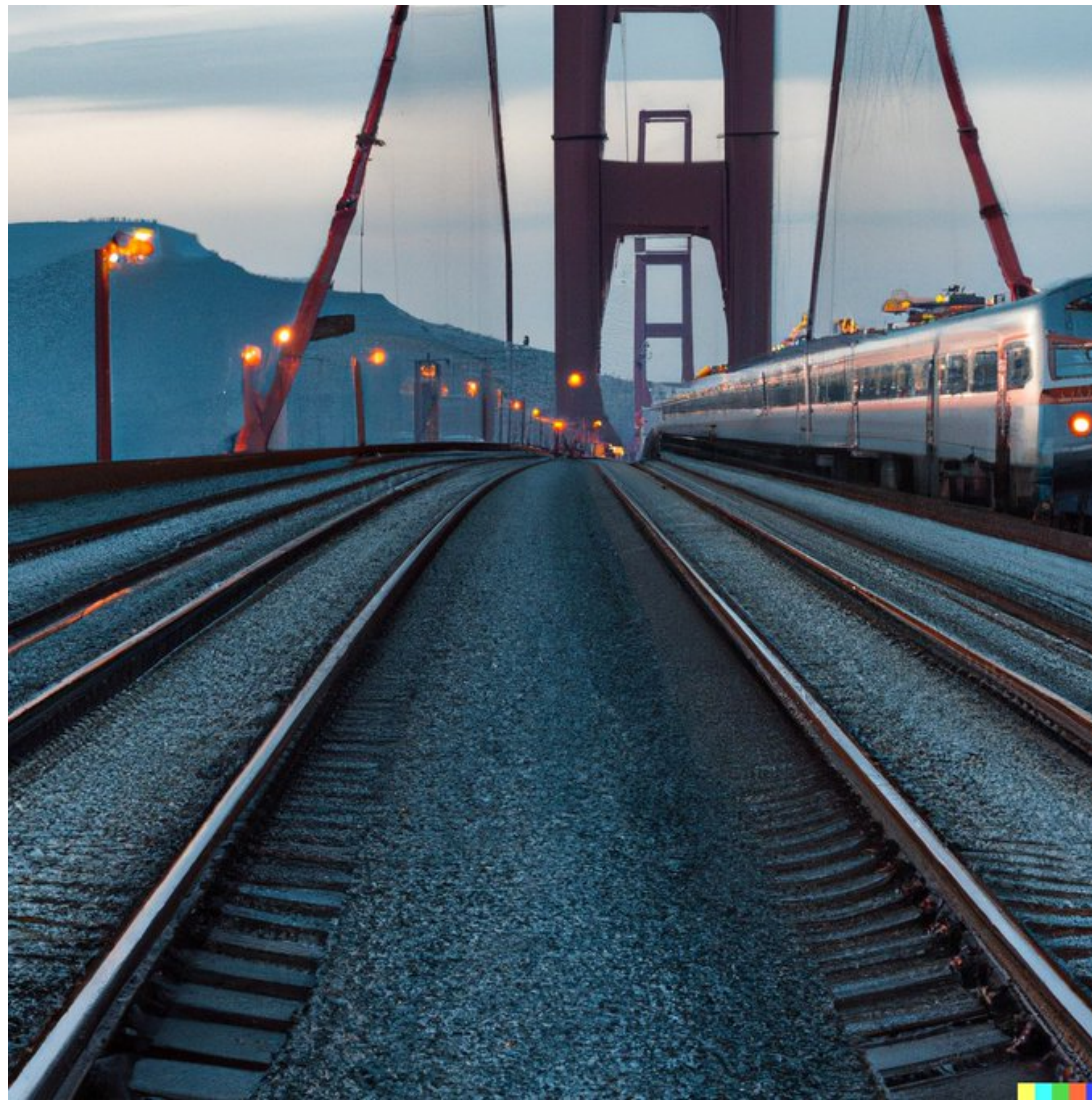
Thanks to Yaron Limpan & Steve Seitz for amazing slides, Songwei Ge, and David McAllister for discussion

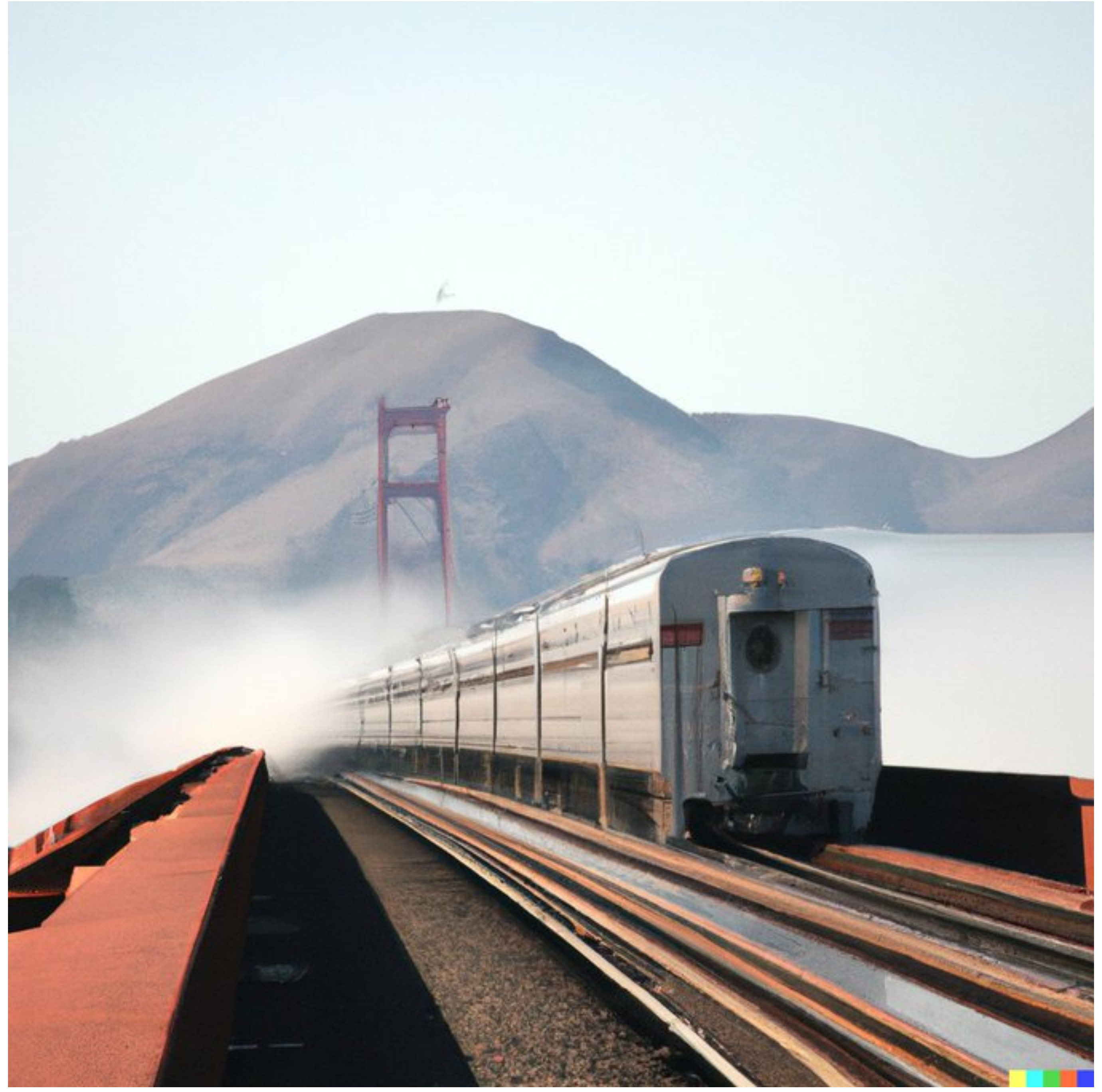
Sora, Open AI February 15, 2024

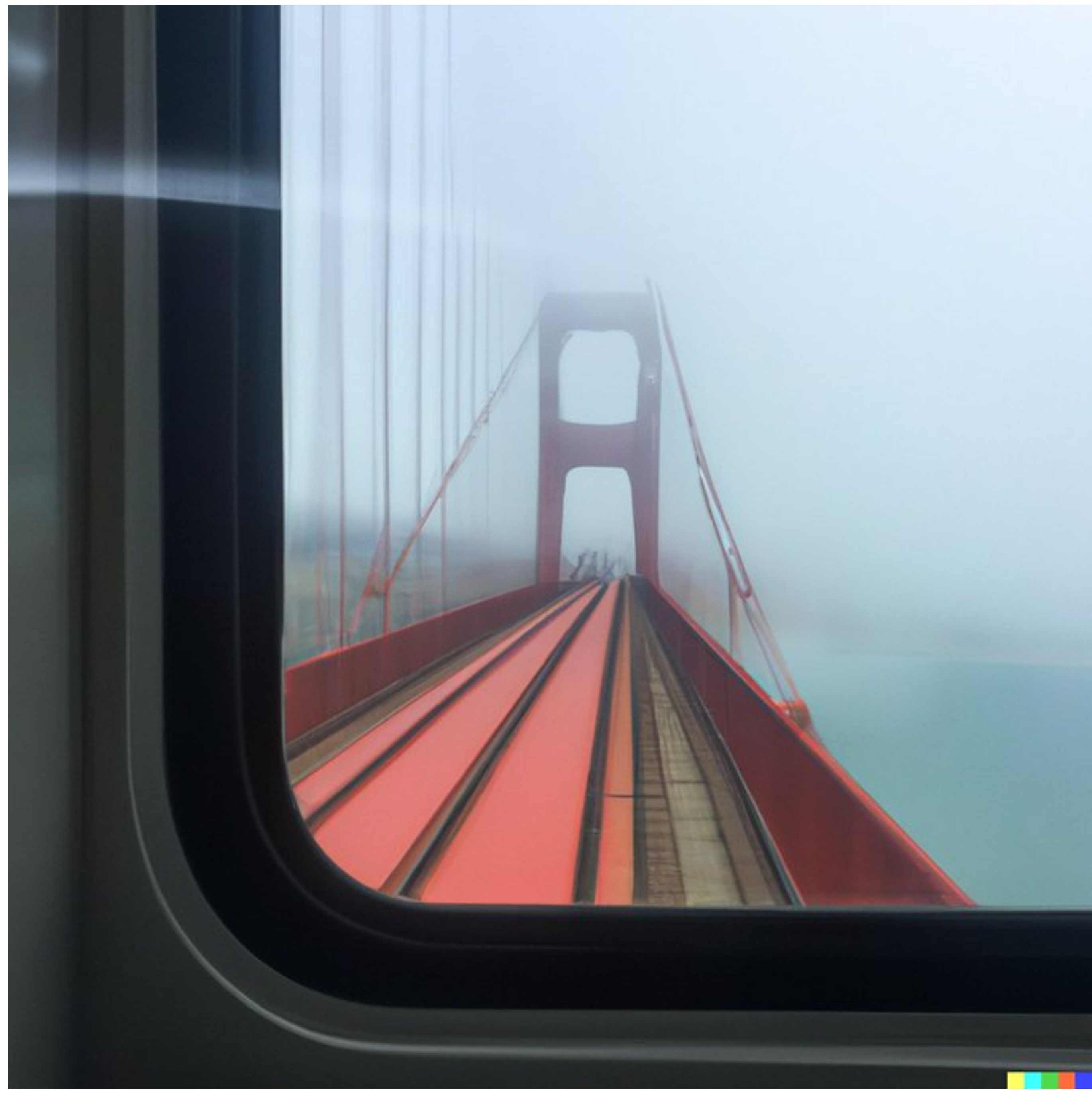
An astronaut riding a horse in a photorealistic style (Dall-E 2)

slide from Steve Seitz's [video]

Impressive compositionality:







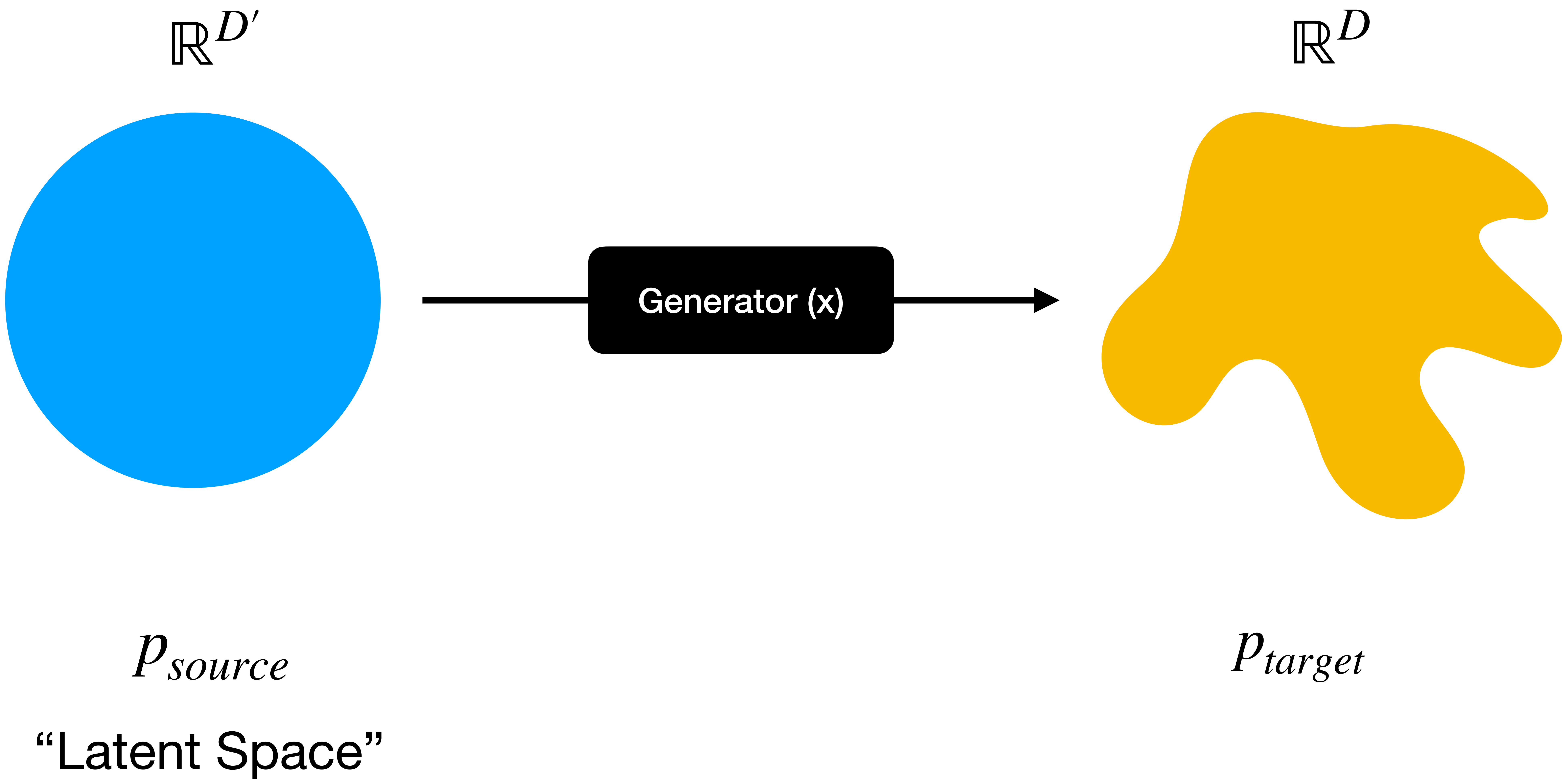
DALL-E + Danielle Baskin

Generative Models

Goal: Modeling the space of Natural Images

- Want to estimate $P(x)$ the probability distribution of natural images
- Why? Many reasons

The generative story



The generative story: Ex. PCA

- A Generative Model has the process of sampling (generating) an image

- For ex, here’s the generative story for PCA in its probabilistic interpretation:

Generative Story

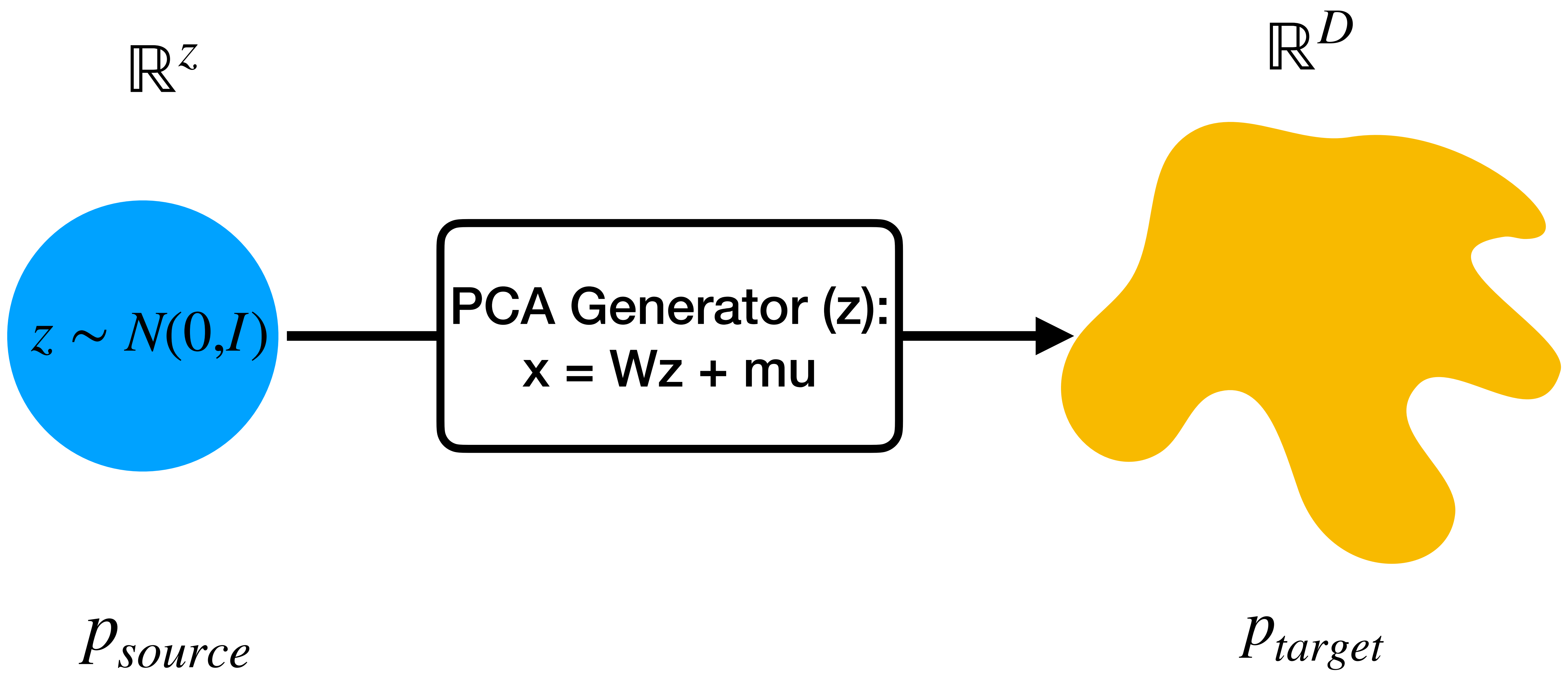
- Any Generative Model has a process of sampling an image
- For ex, here's the generative story for PCA in its probabilistic interpretation:

1. Sample from a Gaussian Distribution

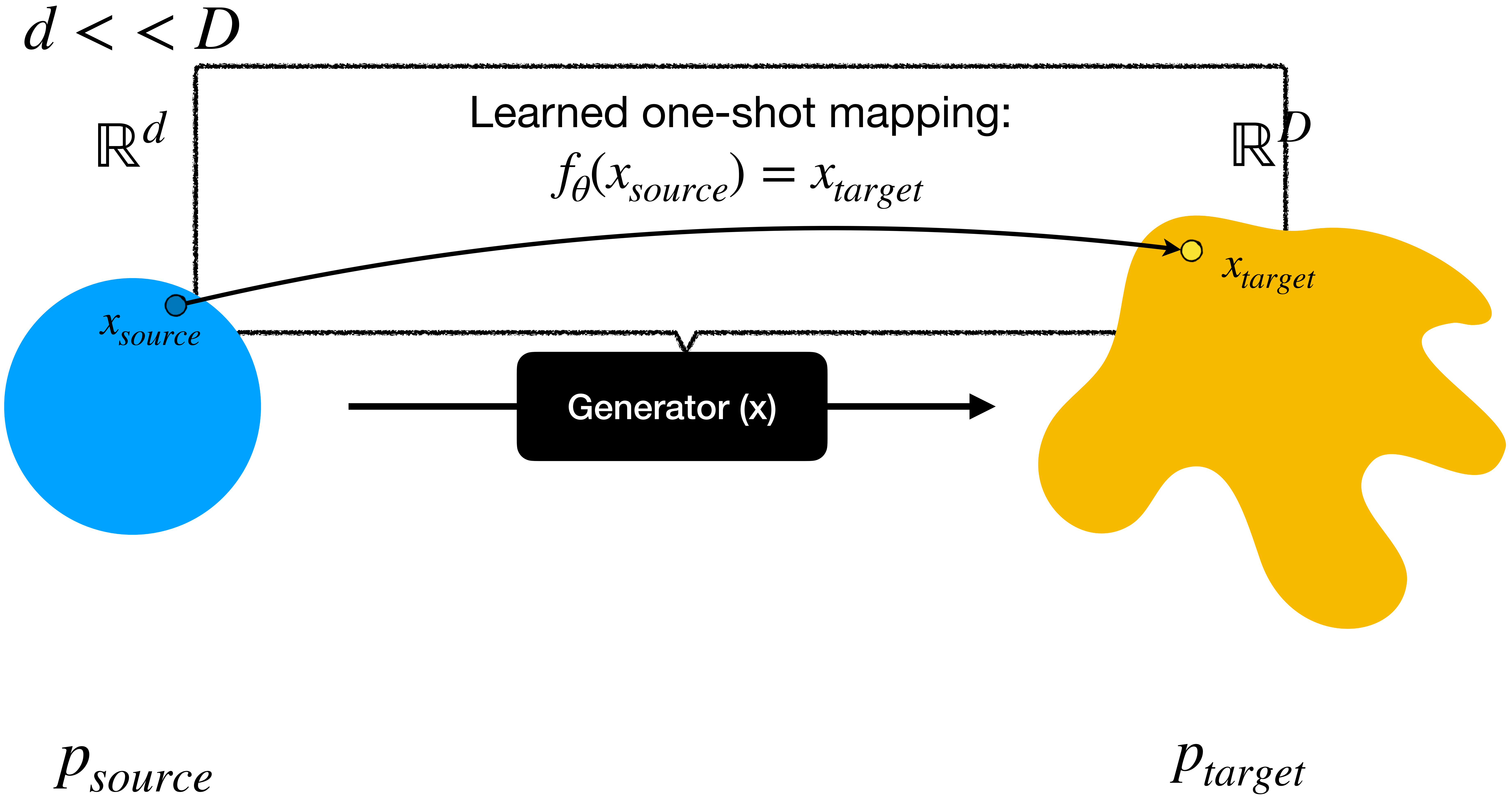
$$z \sim N(0, I)$$

2. Project to Images (W = Eigenvectors, Mu = avg datapoint)

$$x = Wz + \mu$$



Example: GANs / VAEs



“No More GANs” Movement

- GANs really opened up the possibility of image generation
- But people didn’t like it for many reasons
    - Severe mode collapse
    - Unstable training mechanics
- Flow/Diffusion is a reactionary movement against GANs,
  next natural evolution

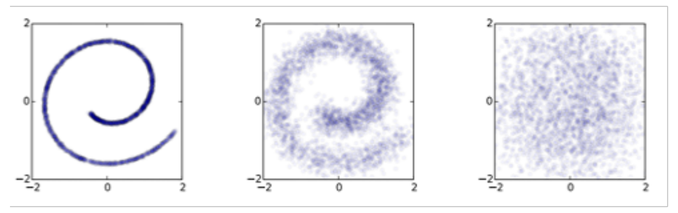
History

GAN, Goodfellow 2014
DCGAN 2015..

StyleGAN 2018

DALL-E1 Open AI 2020

DALL-E2 Open AI 2023
StableDiffusion, Stability 2023



Sohl-Dickstein et al. 2015
Deep unsupervised learning using non equilibrium thermodynamics

DDPM, Ho et al. 2020

Song et al. Score-based Generative Models, DDIM 2021

Rectified Flow, Liu et al. 2022

NICE Dinh et al.
Normalizing Flows 2015

RealNVP, Dinh et al. 2017

Glow, Kingma & Dhaliwal 2018

**Flow Matching,
Lipman et al. 2022**

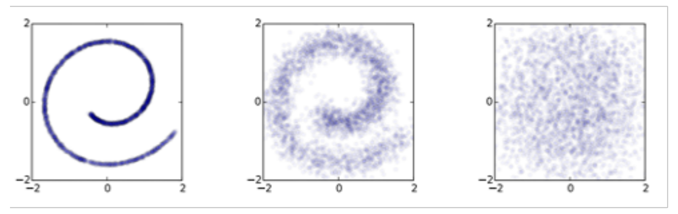
Neural ODE Chen et al. 2018…

Flow Matching Tutorial
NeurIPS 2024

MovieGen late 2024~

History

Diffusion Arc



Sohl-Dickstein et al. 2015
Deep unsupervised learning using non equilibrium thermodynamics

DALL-E1 Open AI 2020

DDPM, Ho et al. 2020

Song et al. Score-based Generative Models, DDIM 2021

DALL-E2 Open AI 2023
StableDiffusion, Stability 2023

Rectified Flow, Liu et al. 2022

NICE Dinh et al.
Normalizing Flows 2015

RealNVP, Dinh et al. 2017

Glow, Kingma & Dhaliwal 2018

Neural ODE Chen et al. 2018…

Normalizing Flow Arc

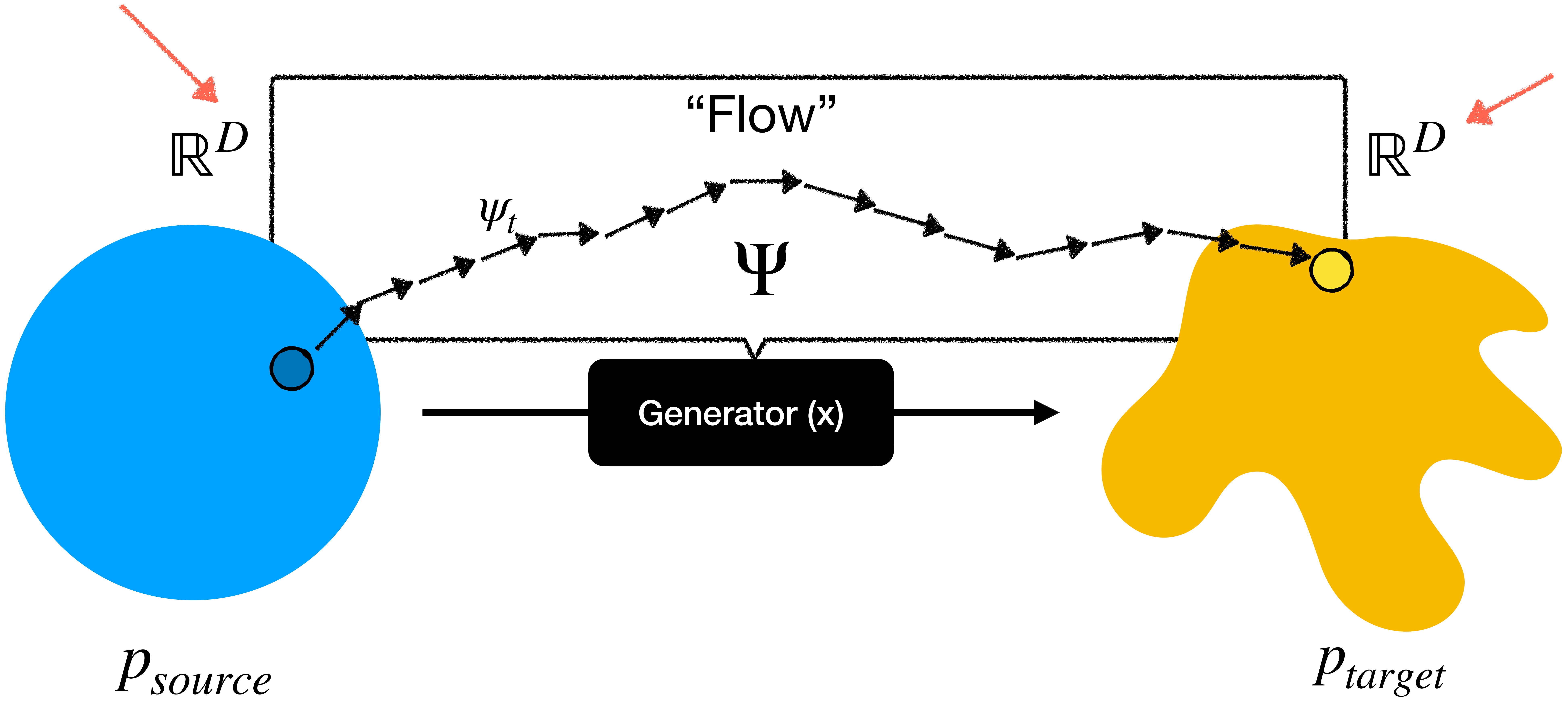
Flow Matching, Lipman et al. 2022

Unification / Simplification

Flow Matching Tutorial NeurIPS 2024

MovieGen late 2024~

Flow based Generative Models

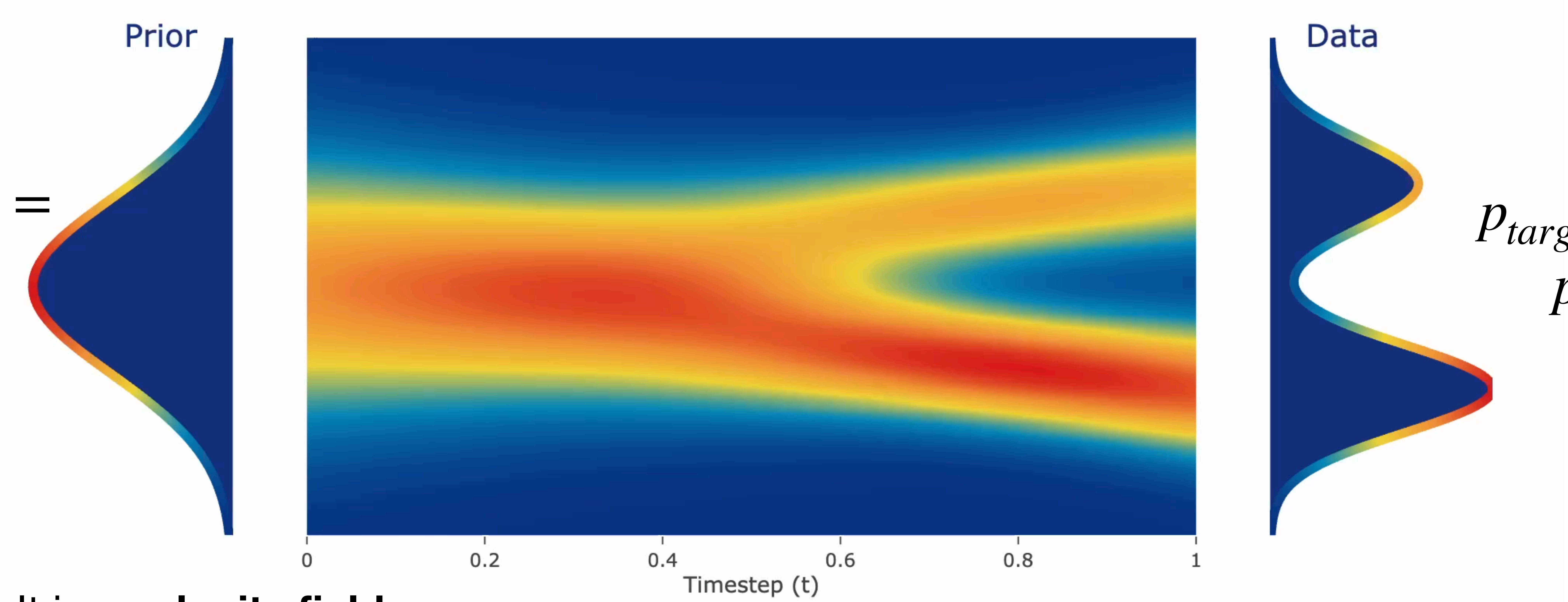


1. Latent space dim is same as the target!

2. Takes T steps to go from src to tgt

What is Flow?

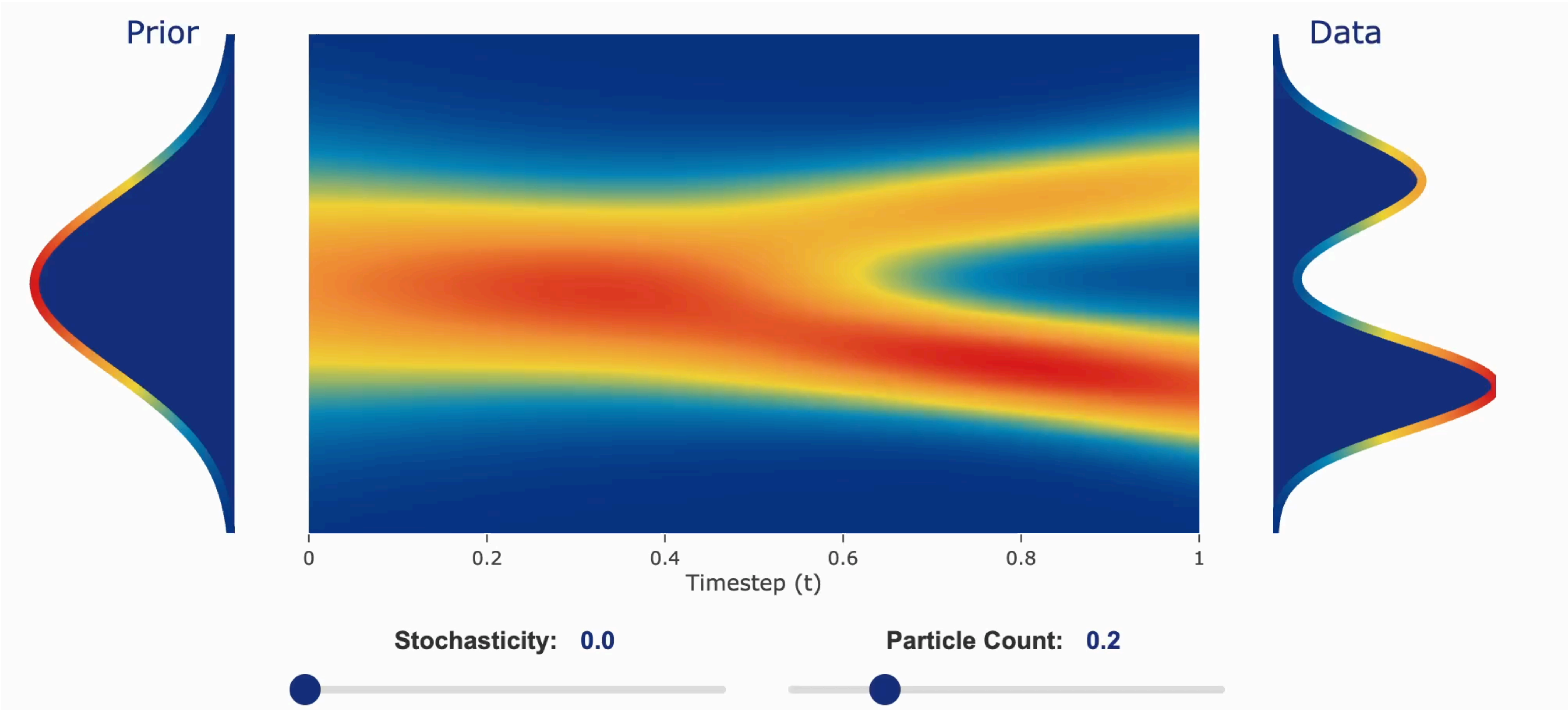
$p_{source} = p_0$



$p_{target} = p_1$

- It is a **velocity field.**
- It’s like a river with some currents, every point defines how fast you move (velocity)
- You ride this river to go from one distribution to next

Riding the river = Integration

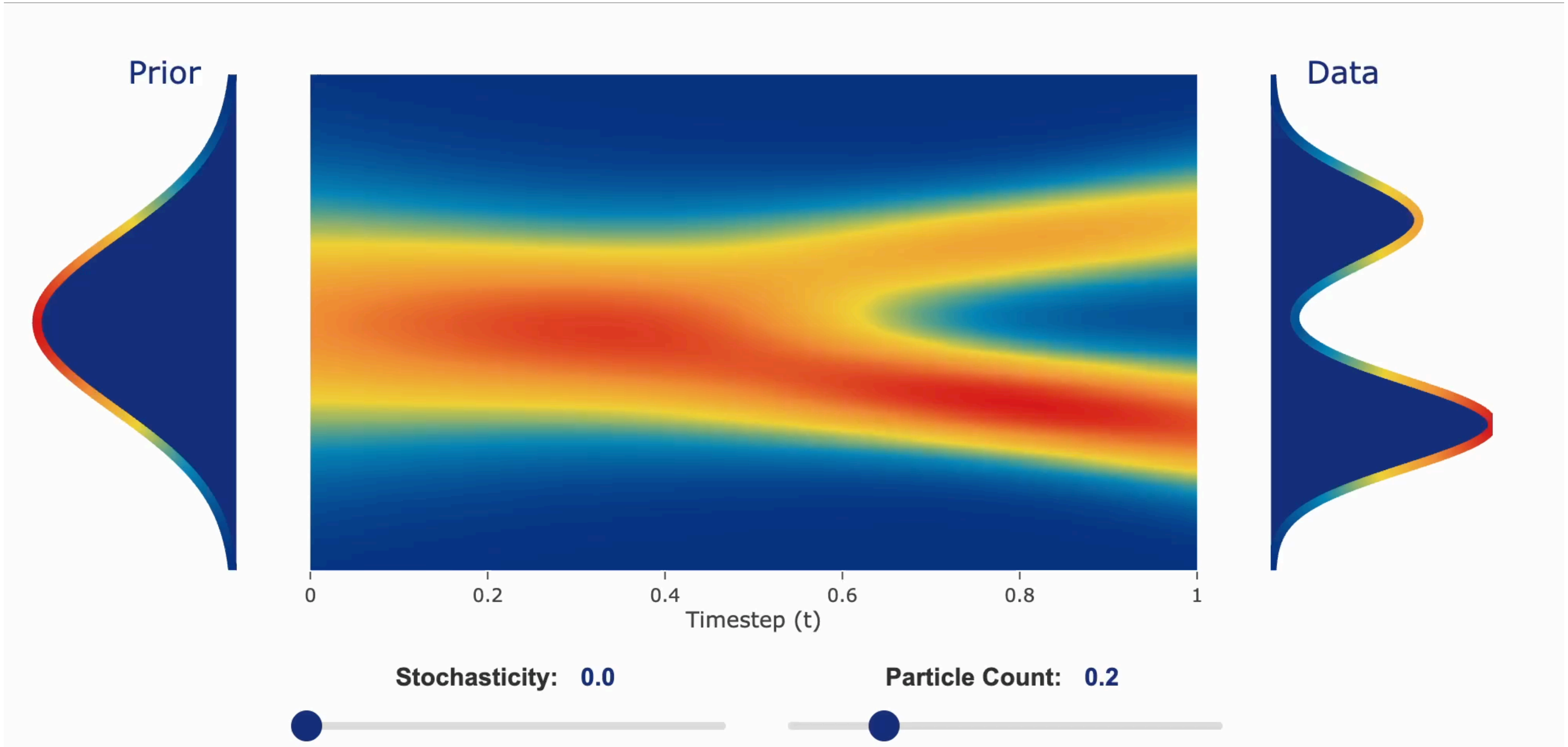


Simplest "Euler Integration":

$$x_{t+\Delta t} = x_t + v_\theta(x_t, t)\Delta t$$

- Riding this rive means you add little bits of velocity defined at each location
- This is called "Integration", also called solving the Ordinary Differential Equation (ODE) with initial state x_0, through some differential parametrized by a network: $\dfrac{dx}{dt} = v_\theta(x, t)$
- You can add stochasticity when riding it, then it becomes SDE (more next lecture)

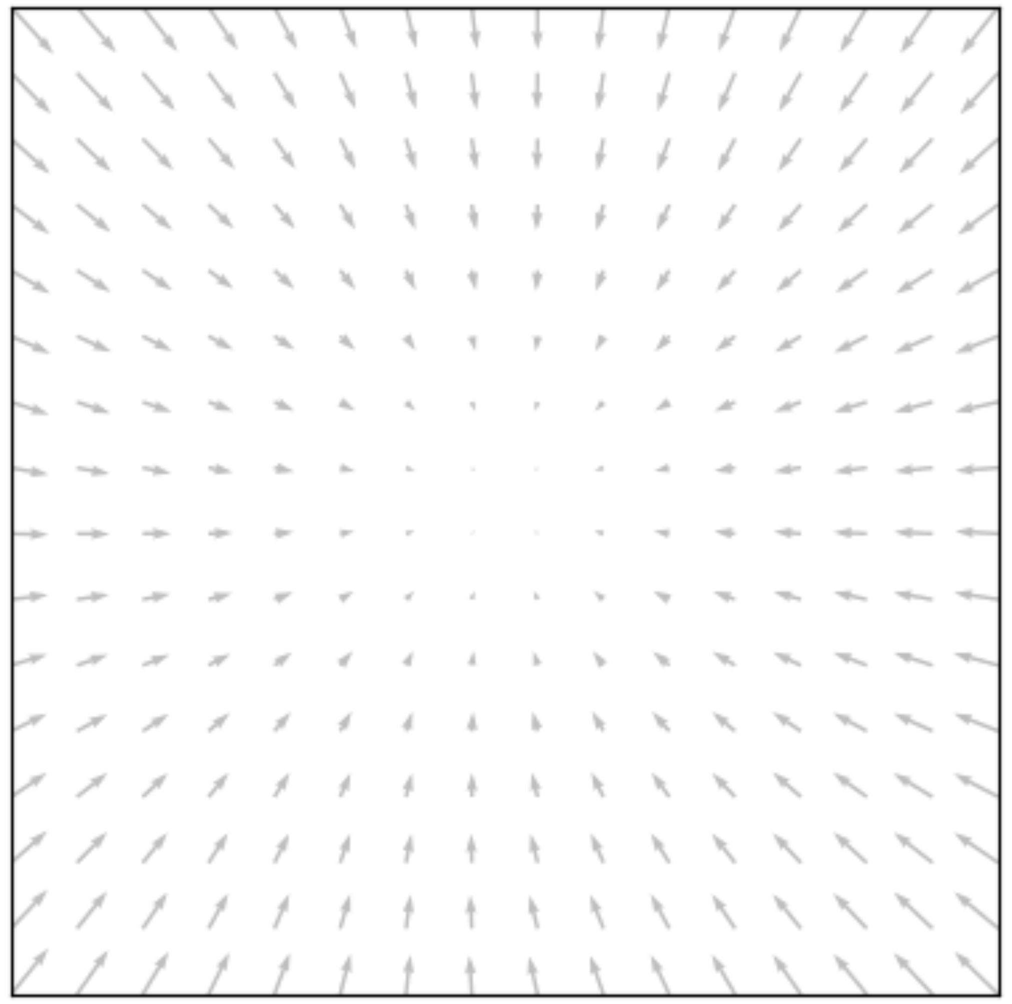
How can we learn this flow?

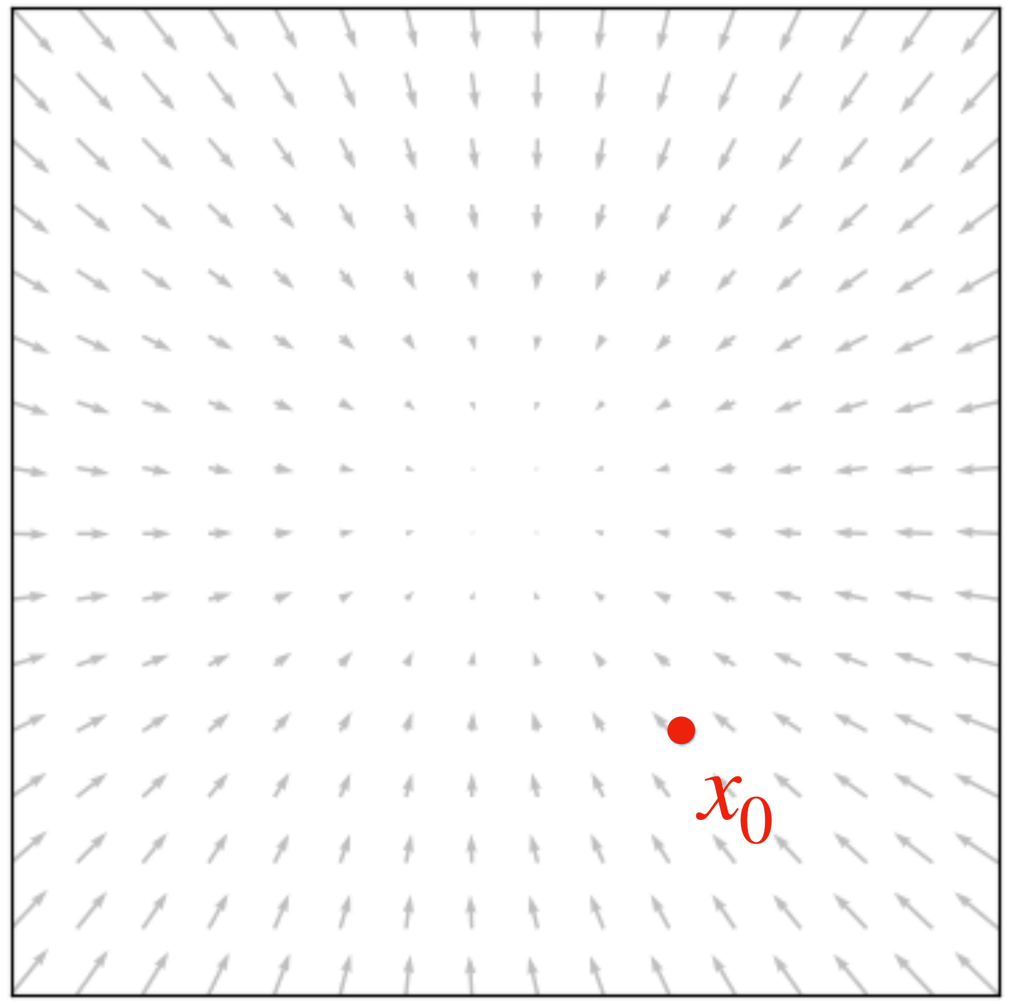


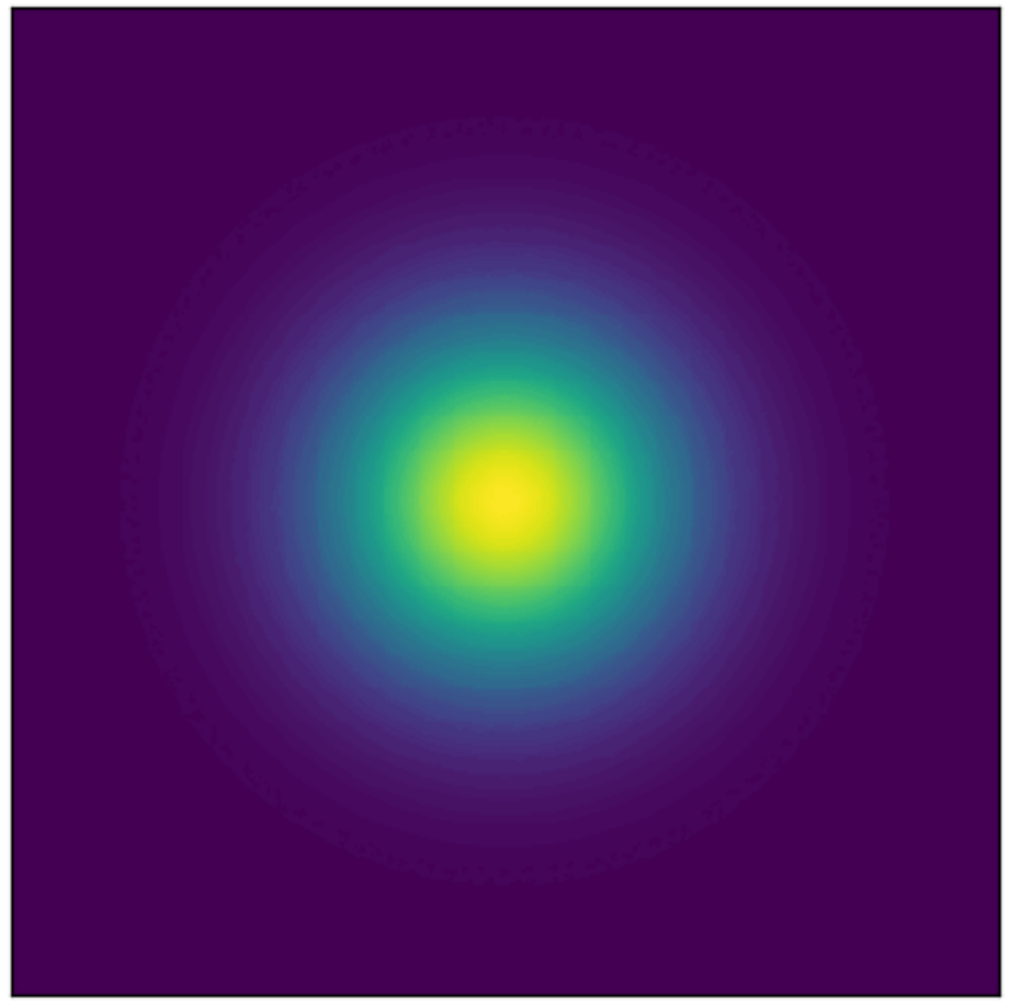
In 2D

$x_0 \sim p$

$v_t(x)$ $\quad$ $x_t = \Psi_t(x_0)$ $\quad$ $x_t \sim p_t$







Flow ODE

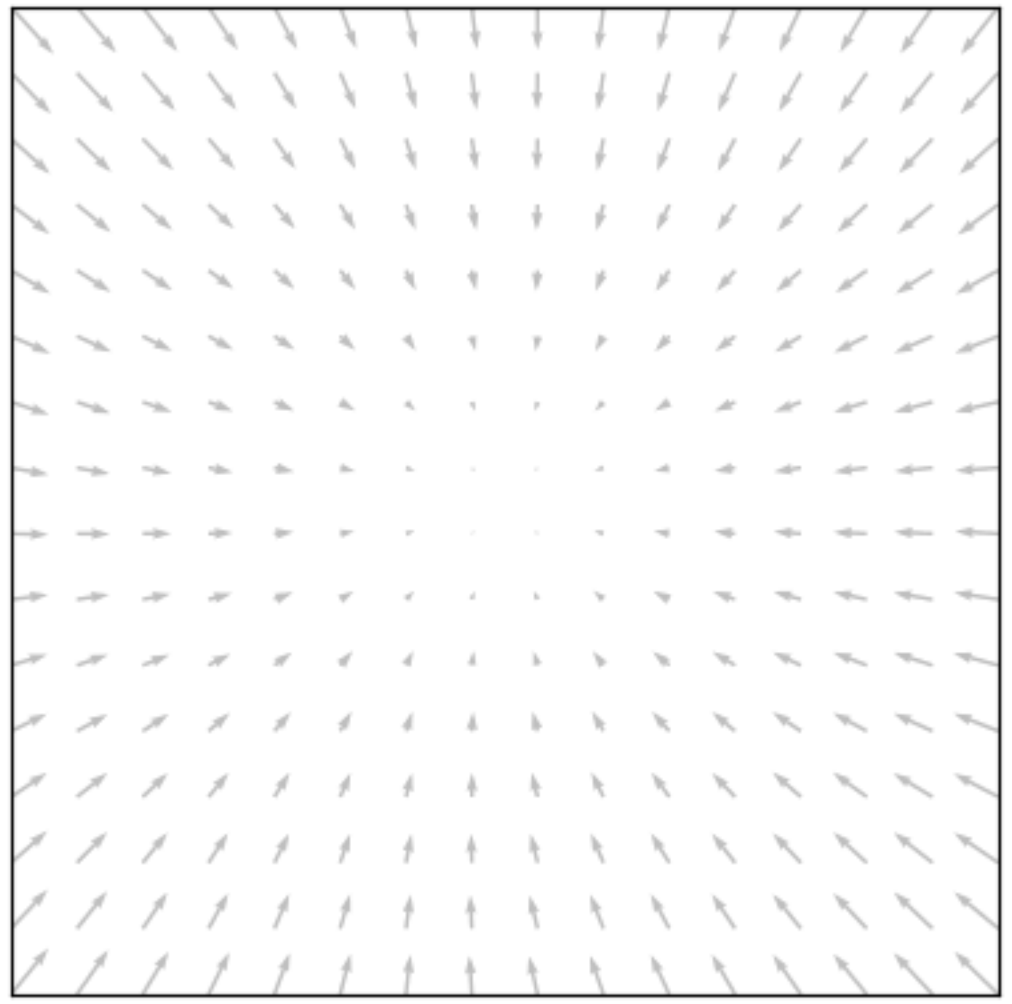
$$\dot{x}_t = v_t(x_t)$$

Slides from Yaron Lipman

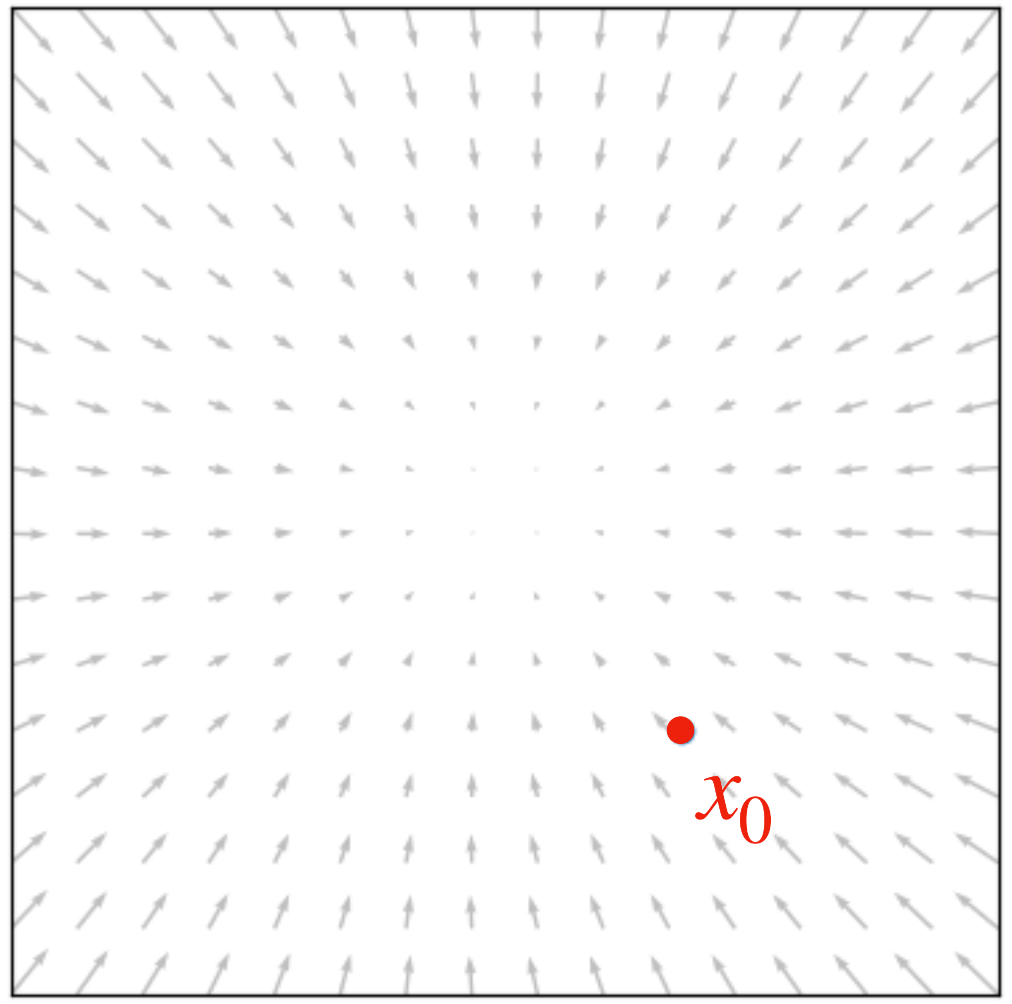
[Chen et al. 2018]

What do we have to train this?

$v_t(x)$



$x_t = \Psi_t(x_0)$

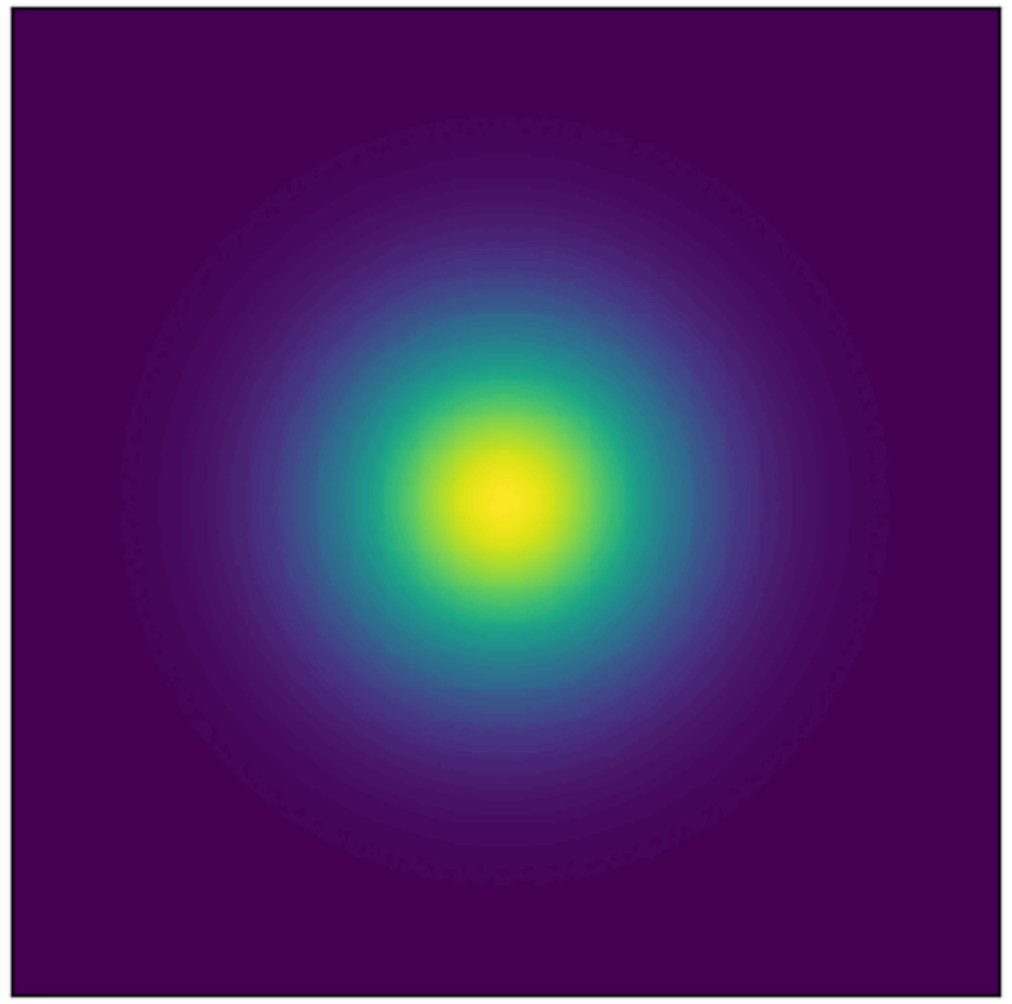


Flow ODE

$$\dot{x}_t = v_t(x_t)$$

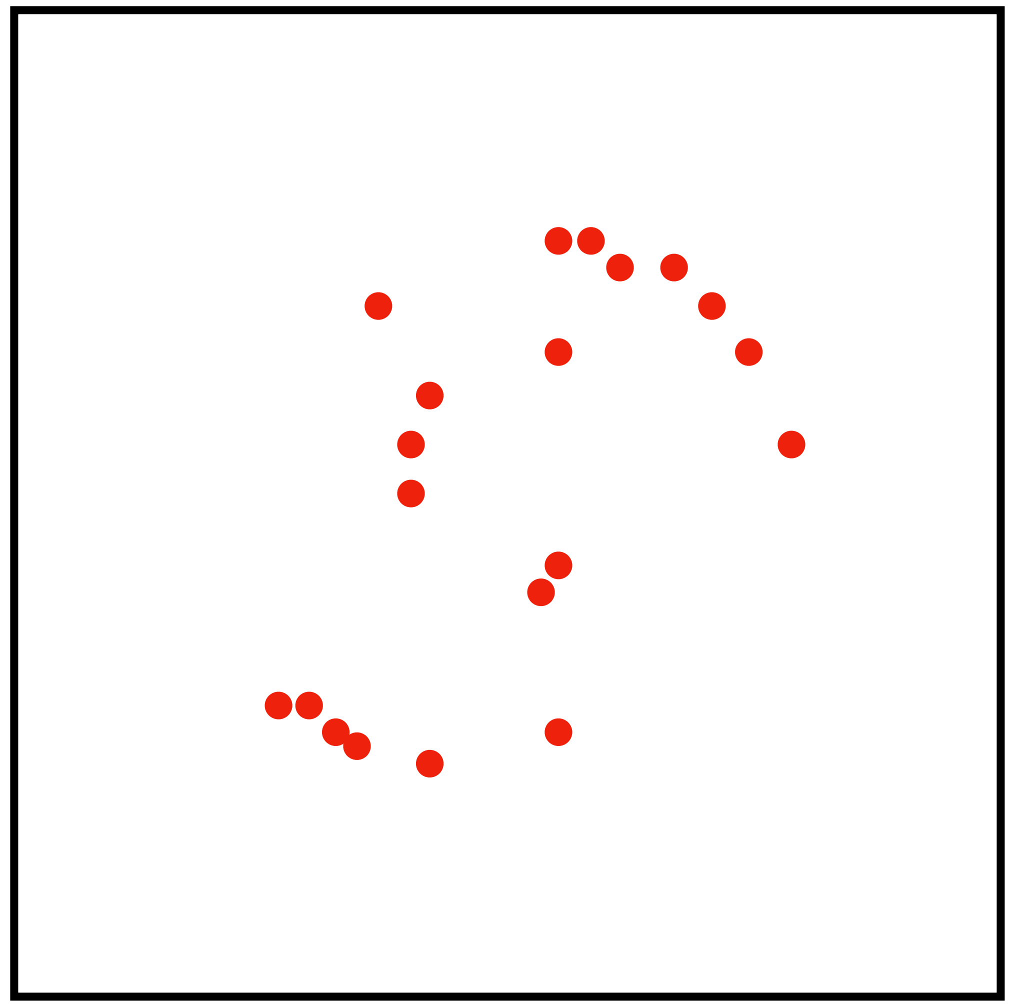
$x_0 \sim p$

$x_t \sim p_t$



Samples of p_1

$q(x)$



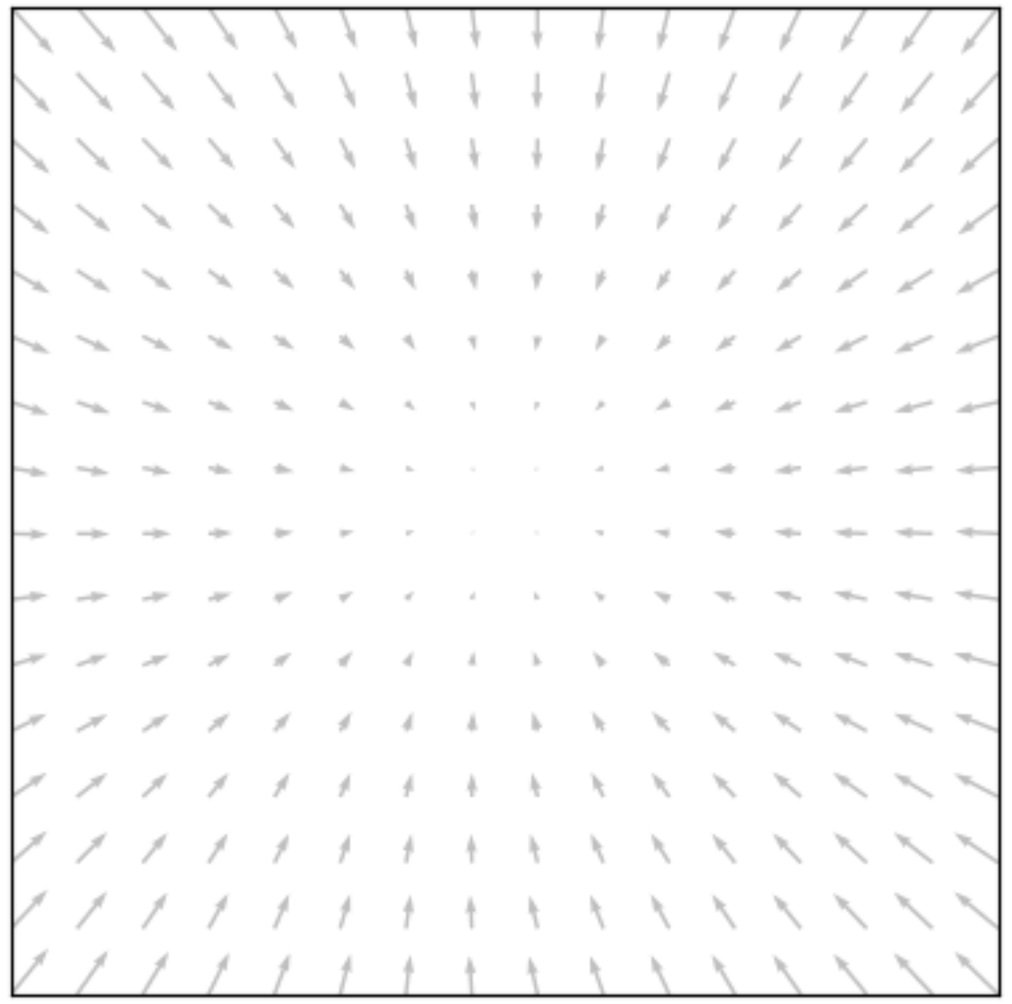
Slides from Yaron Lipman

[Chen et al. 2018]

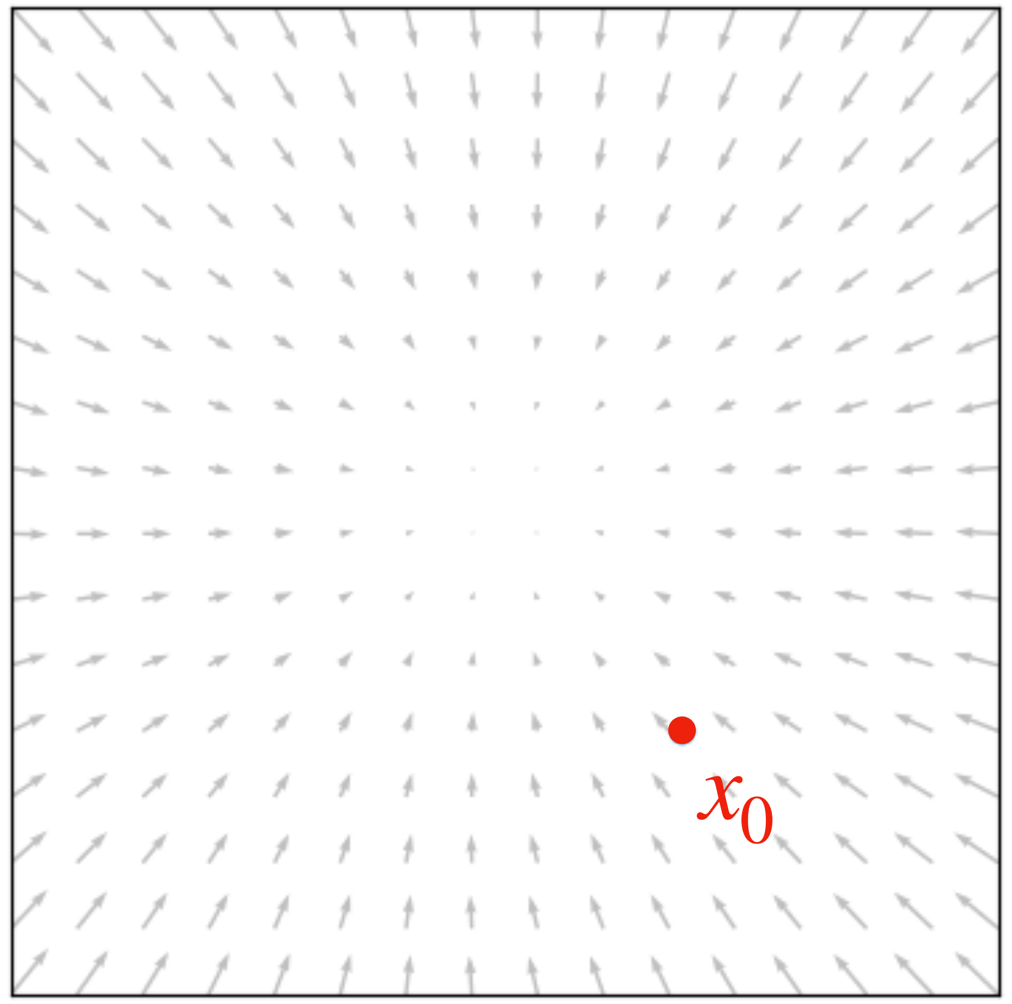
Important Caveat: Continuity Eq

$$x_0 \sim p$$

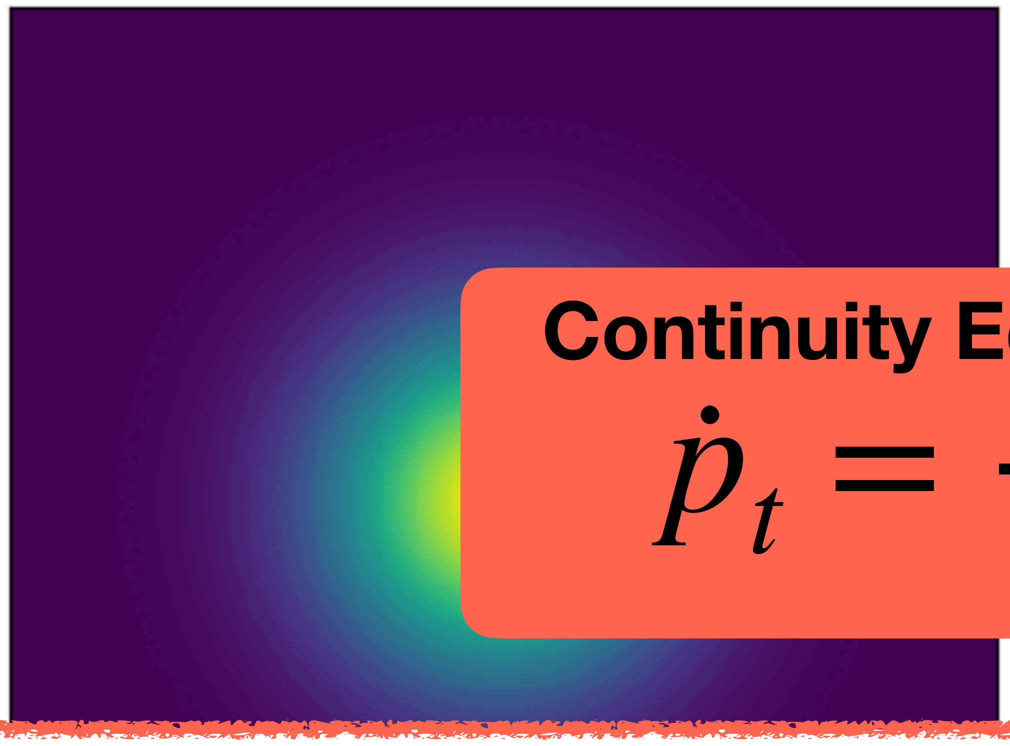
$$v_t(x)$$



$$x_t = \Psi_t(x_0)$$



$$x_t \sim p_t$$



Continuity Equation PDE (*fixed x*)

$$\dot{p}_t = -\operatorname{div}(p_t v_t)$$

- Need to conserve probability mass
- In the river, analogy you cannot add or remove water
- It has to come from somewhere and go somewhere

Flow ODE

$$\dot{x}_t = v_t(x_t)$$

Slides from Yaron Lipman

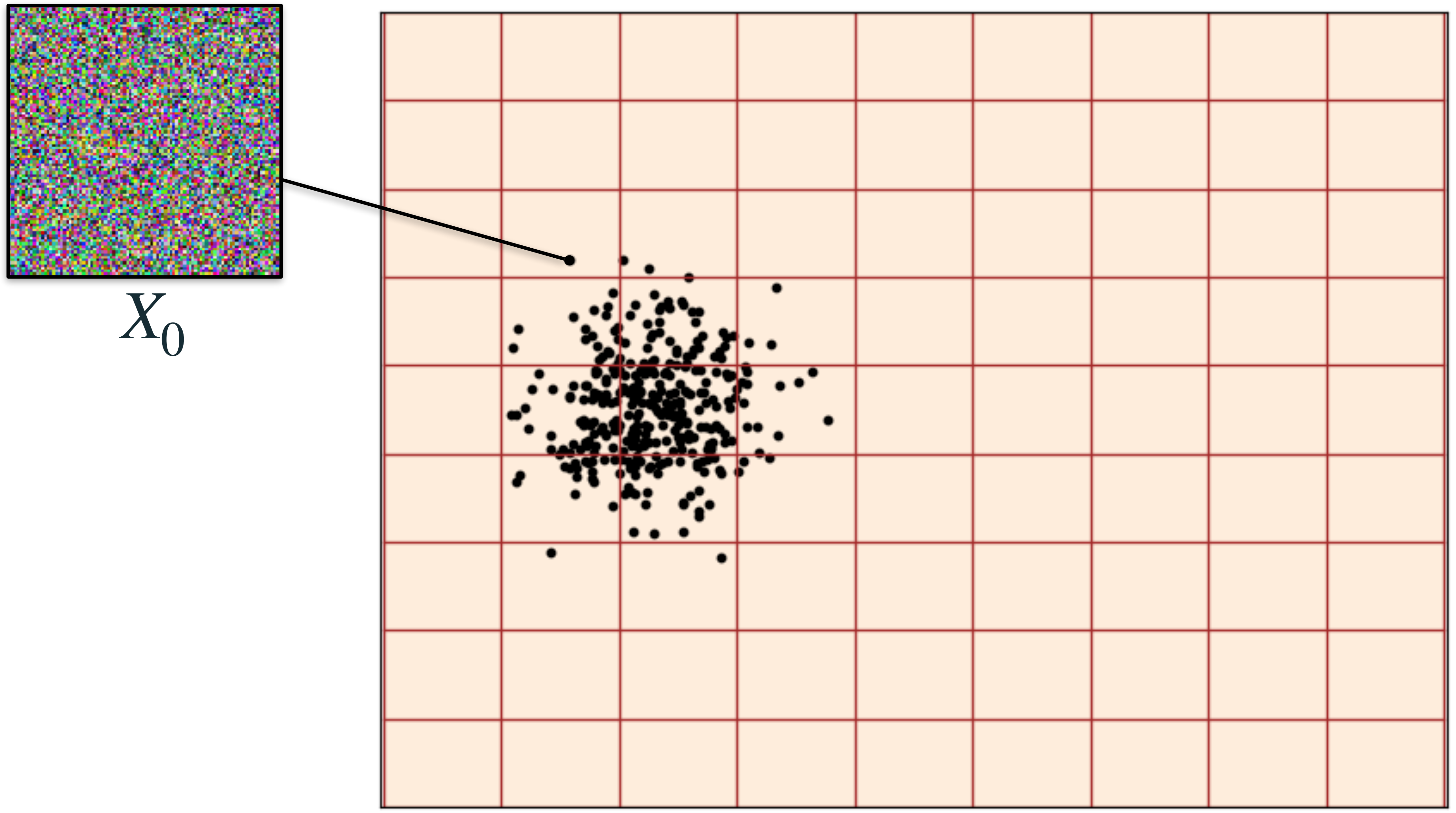
[Chen et al. 2018]

The flow can be though about learning a warping function

$$X_t = \psi_t(X_0)\ , \quad t \in [0,1]$$

Warping

Source $X_0 \sim p$



Slides from Yaron Lipman

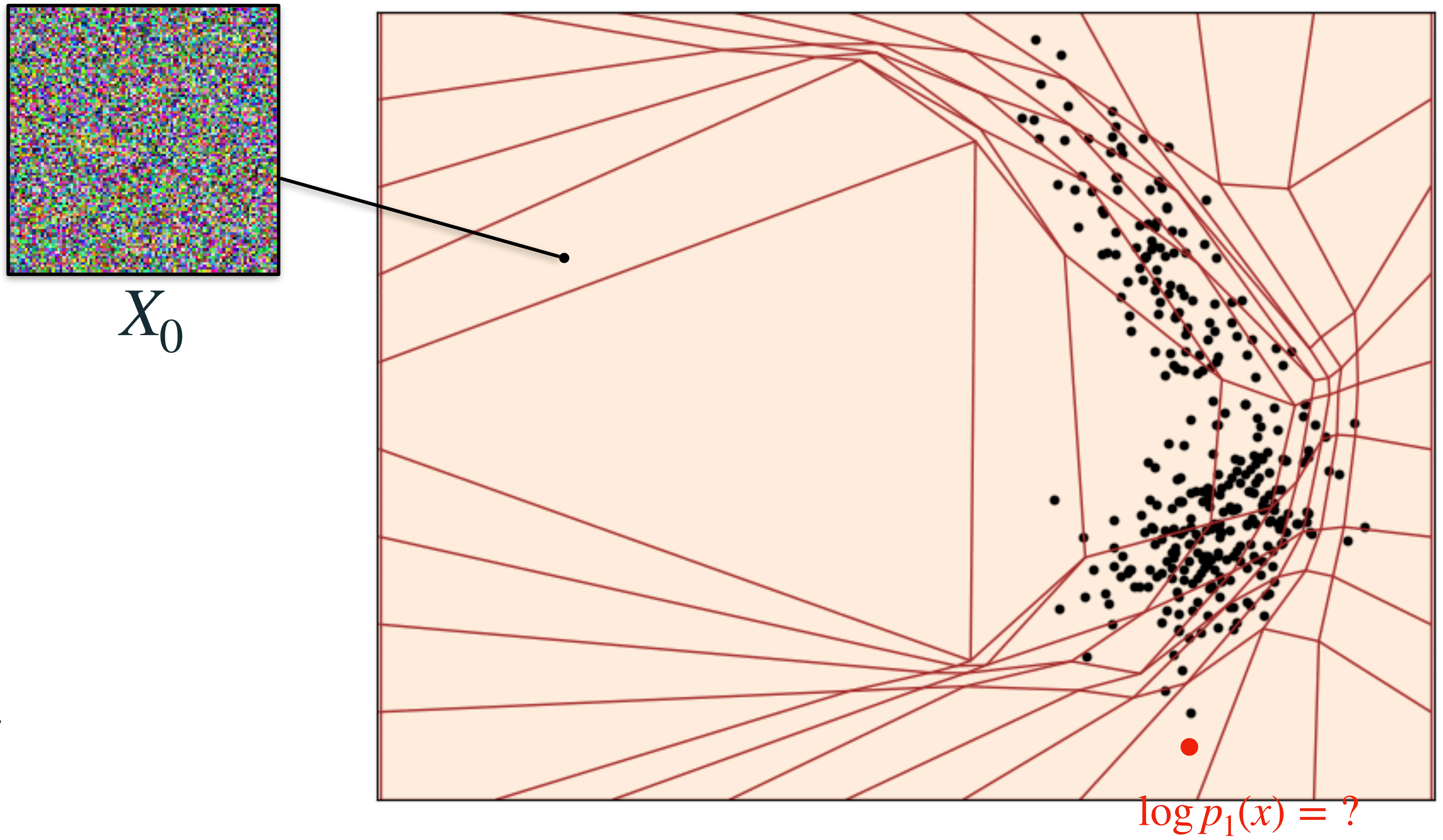
Initial approach trained flow with Maximum Likelihood

$$D_{\mathrm{KL}}(q \,\|\, p_1) = -\,\mathbb{E}_{x\sim q} \log p_1(x) + c$$

$$X_t = \psi_t(X_0)\,, \quad t \in [0,1]$$

Warping

Source $X_0 \sim p$



- Normalizing Flow, Continuous Normalizing Flow
- Chaining ψ_t needs to satisfy the continuity equation!!!!!
- This requires ODE integration DURING training with invertible neural networks

Slide adapted from Yaron Lipman

Previous Normalizing Flow works

- Tries to directly deal with this continuity equation constraint
- Very slow to train (need to integrate while training)
- Other constraints like invertibility of ψ_t
- Nice idea with promising results but limited capability + not practical to train

Instead, model Flow with Velocity

$$\frac{\mathrm{d}}{\mathrm{d}t}\psi_t(x) = u_t(\psi_t(x))$$

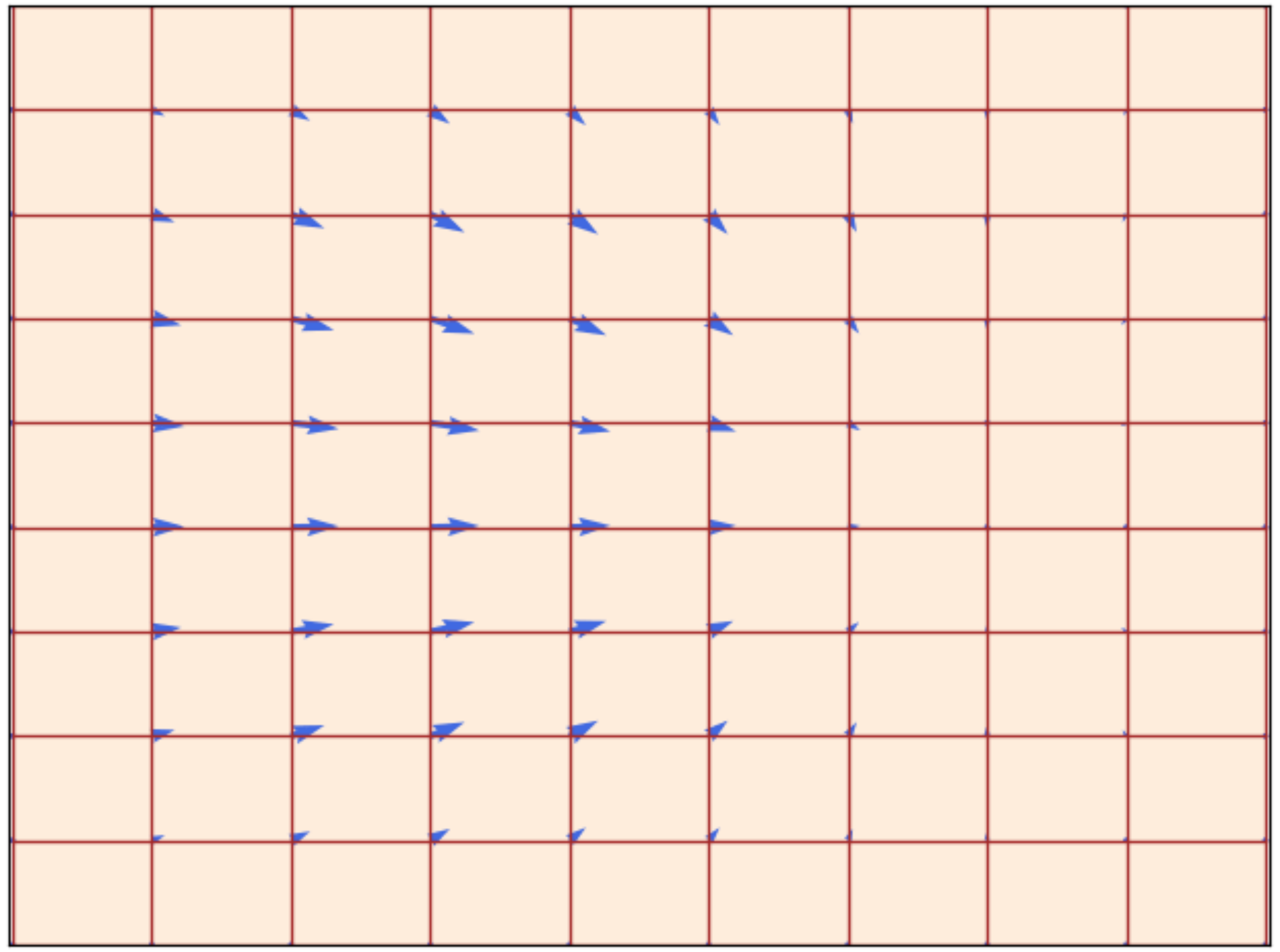
Flow

$$\psi_t(x)$$

Solve ODE **Differentiate**

$$u_t(x)$$

Velocity



- **Pros**: velocities are ***linear***
- **Cons**: simulate to sample

Slides from Yaron Lipman

Flow Matching [Lipman et al. ’22]

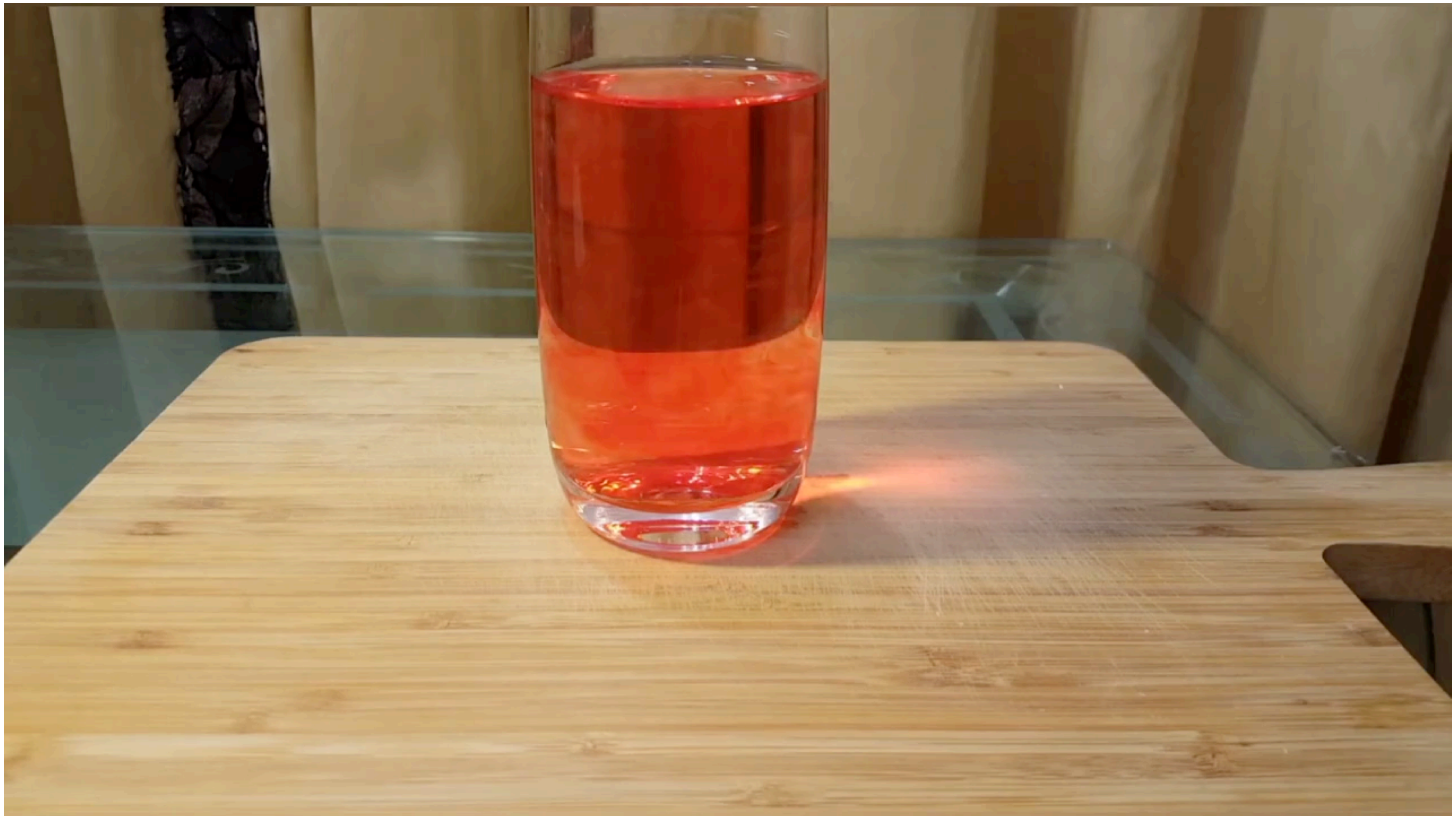
- Directly learn the velocity field!
- Don’t have to worry bout the continuity equation because velocity fields can’t add / subtract mass. You only re-distribute
- Continuity equation satisfied by construction.
- Basically just learn velocity fields for each data sample (conditional velocity field), all will be fine!

Diffusion: Physics Interpretation

Heat Diffusion

Diffusion: Physics Interpretation

Reversing the process



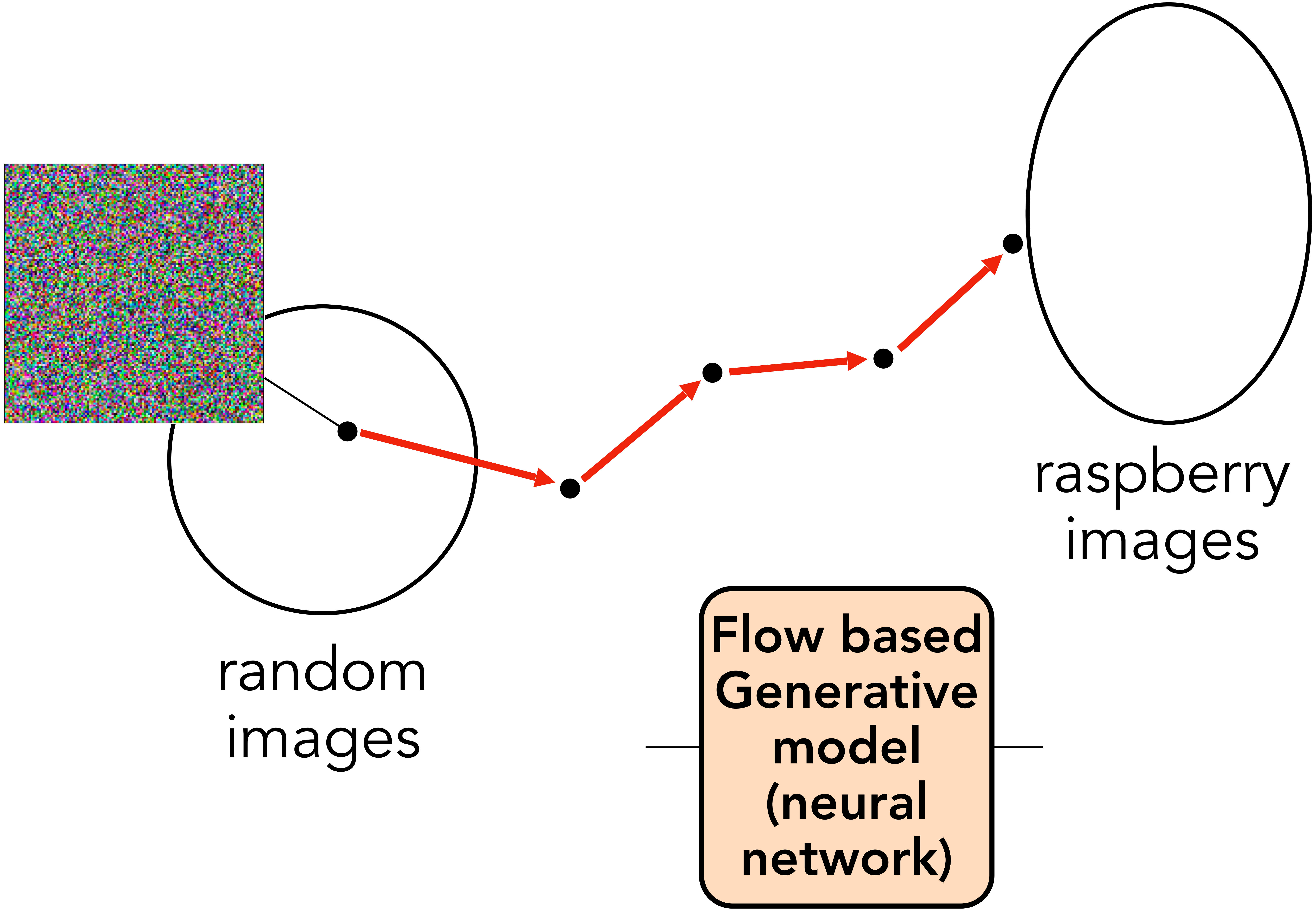
Idea in Song et al. Score-Based Generative Modeling through Stochastic Differential Equations 2021

Diffusion vs Flow matching

- They both end up learning flow, but diffusion poses the problem as learning a specific noising process and learning to denoise it.
- Diffusion: spread out like heat diffusion, learn how to undo it
- Flow matching: More general, just directly learn a velocity field from one to another

Ok so how to train the flow??

Quick Recap:



slide from Steve Seitz's [video]

Training

1. Take real data, corrupt it to left the distribution somehow

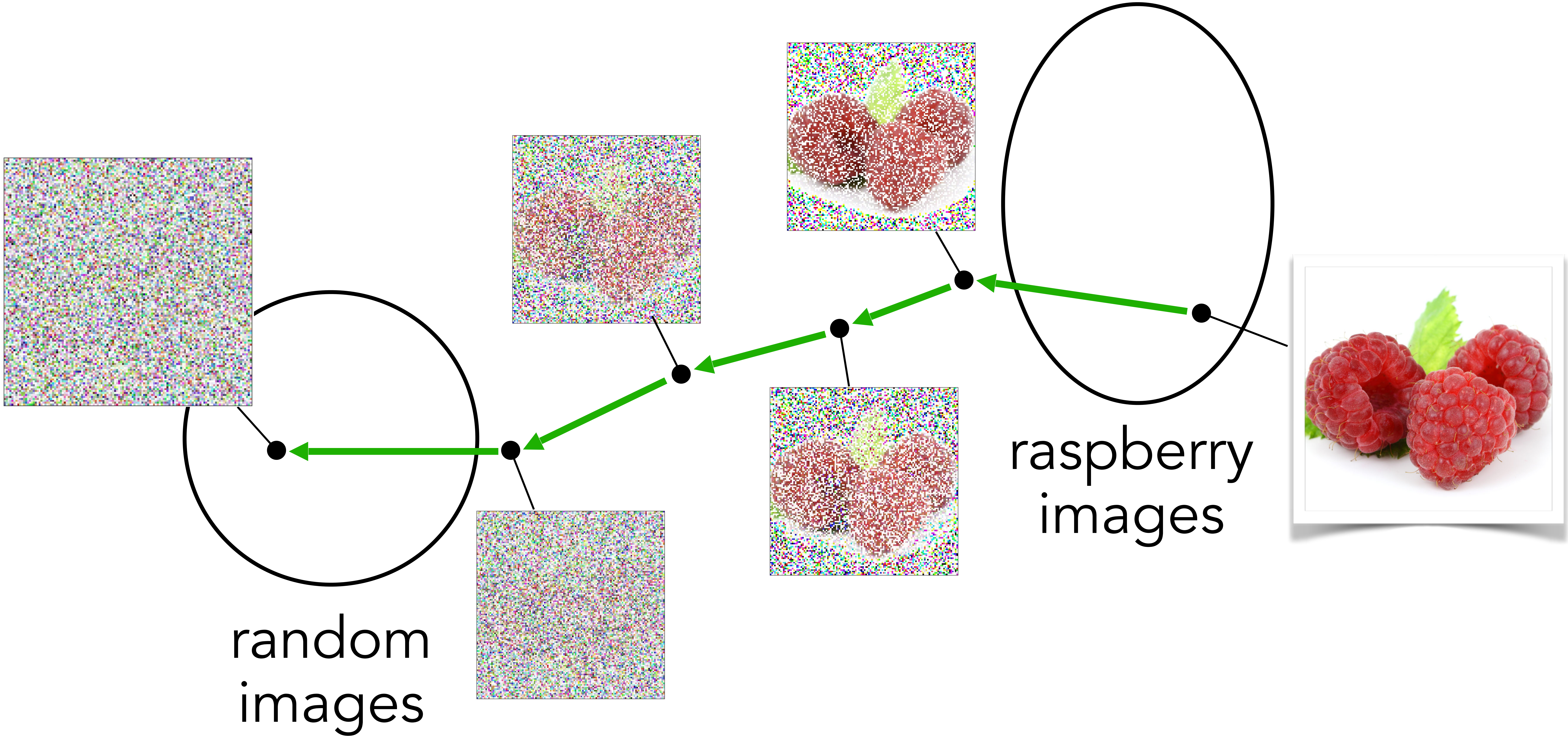


Figure from Steve Seitz's [video]

Training

1. Take real data, corrupt it to left distribution somehow
2. Learn to undo the process!

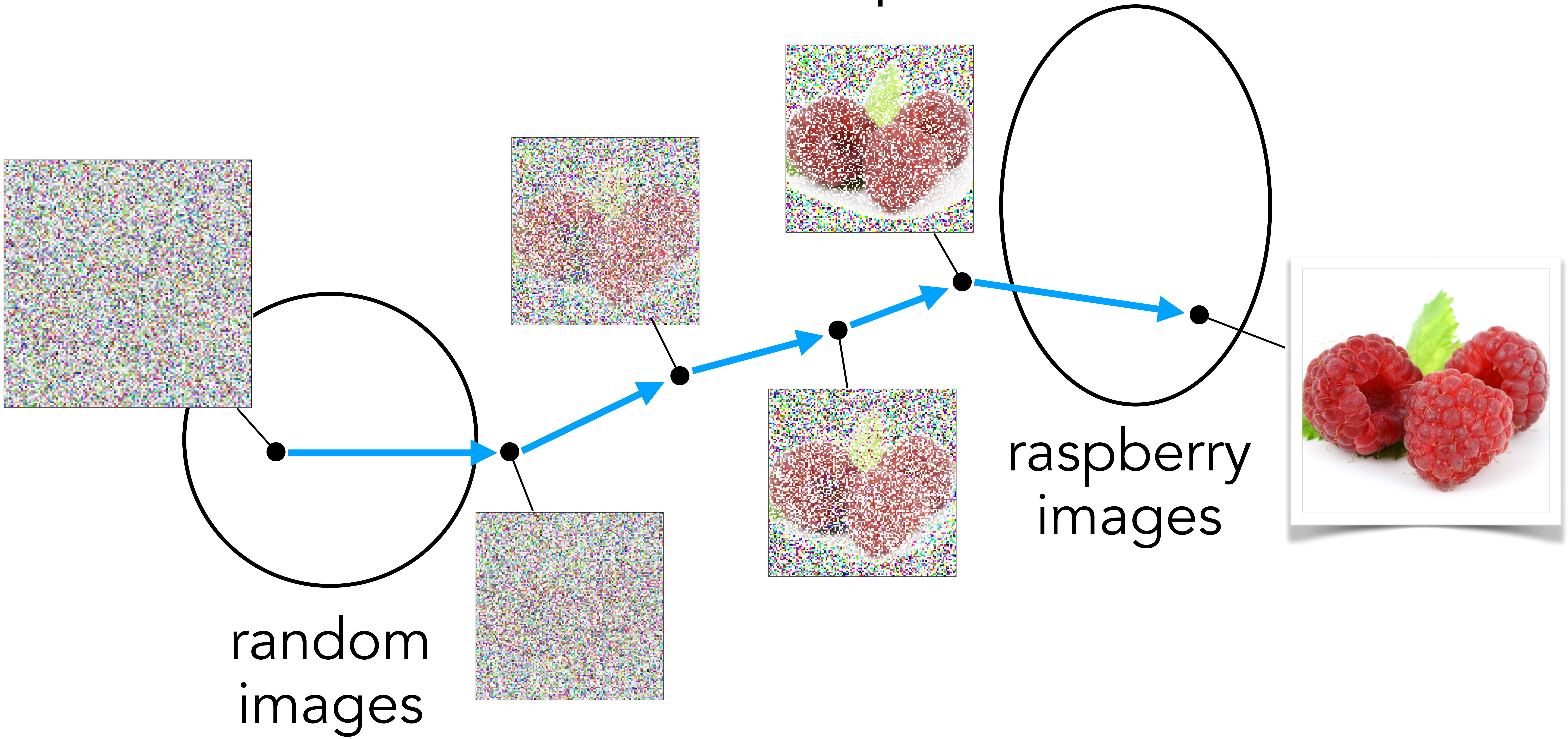
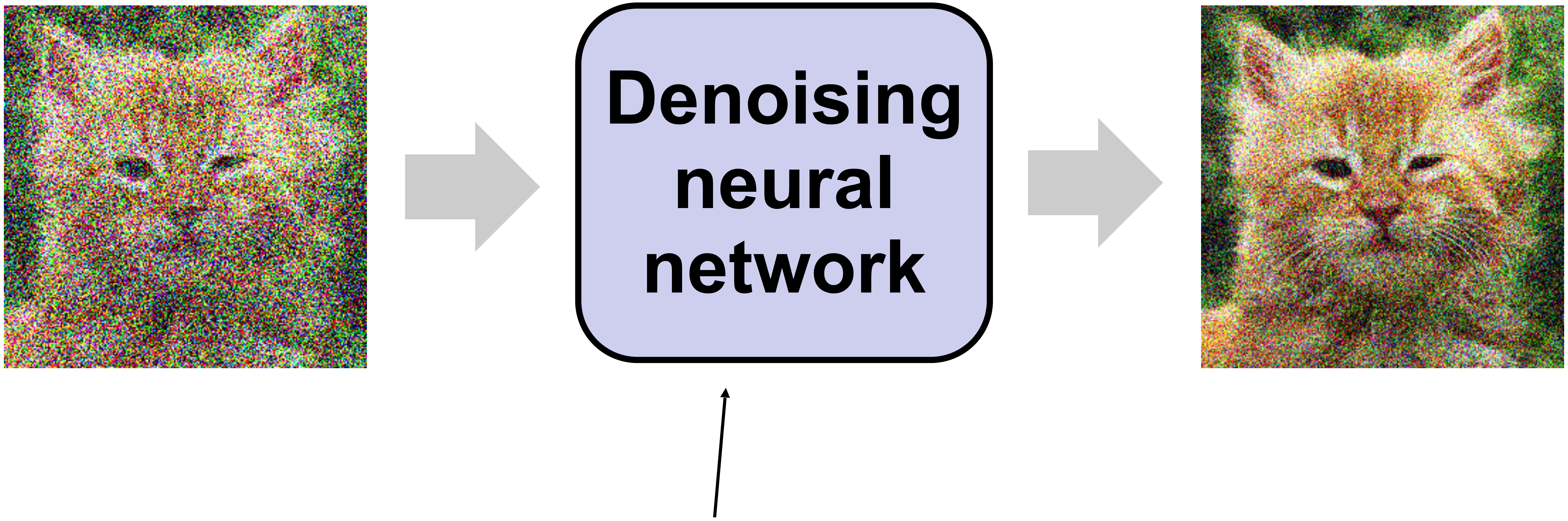


Figure from Steve Seitz's [video]

Denoising with a neural network



This network can be a U-Net or other
suitable image-to-image network

Slide source: Steve Seitz

$$$ question, how to pick the intermediate path?

How to generate this Green path?

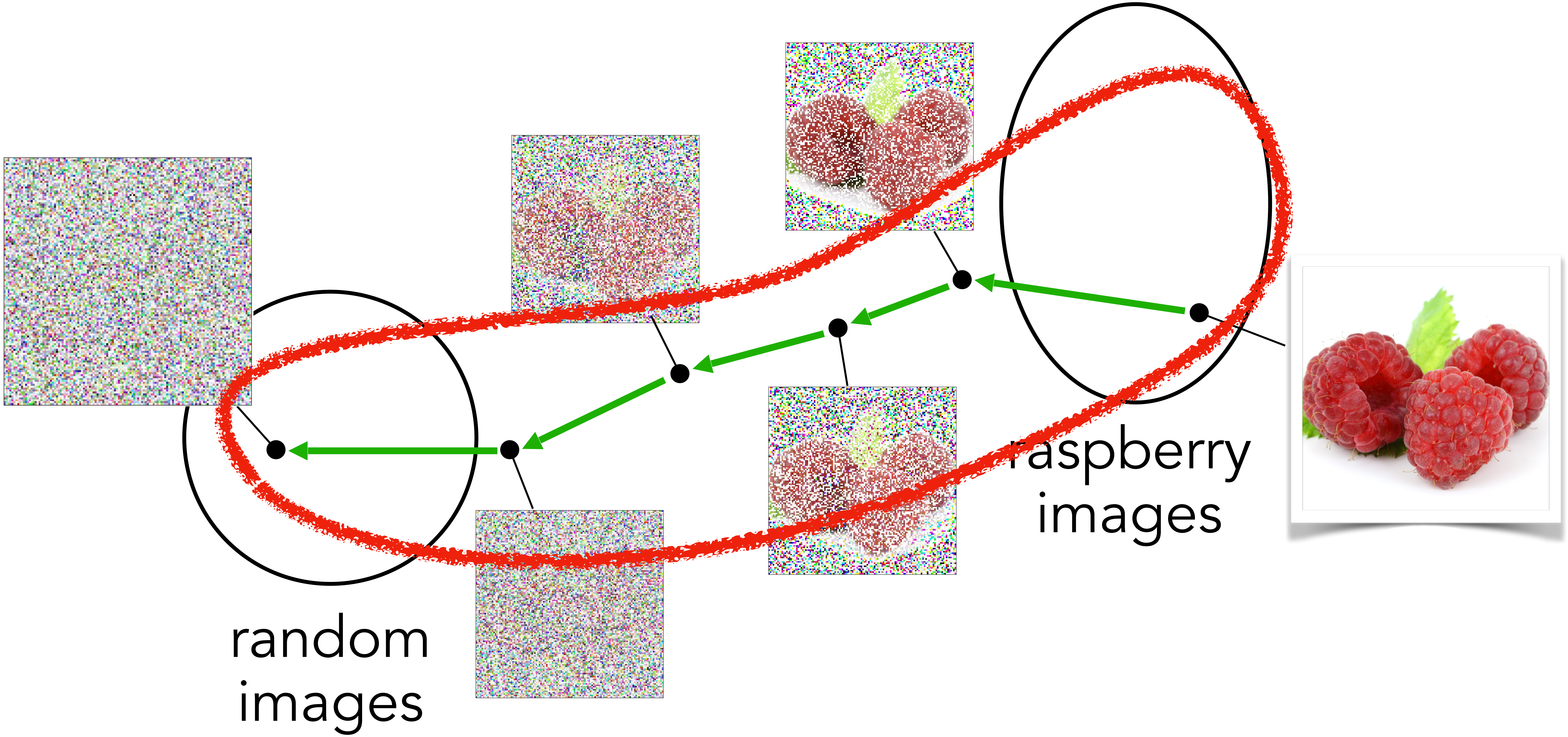


Figure from Steve Seitz's [video]

What is the path?

- How to add noise? What kind of noise?? What schedule to add them???
- This is complicated in the diffusion literature (lots of math).
- Can we keep it simple?

Bc it has to follow physical diffusion process.. Every time step some gaussian has to be added. But in the end you want it to be a N(0,1) etc..

Flow Matching [Lipman et al. 2022] !

Flow matching basically says, you can add noise however you like!

How to construct x_t

TLDR: Sample noise, add it, then reconstruct the data

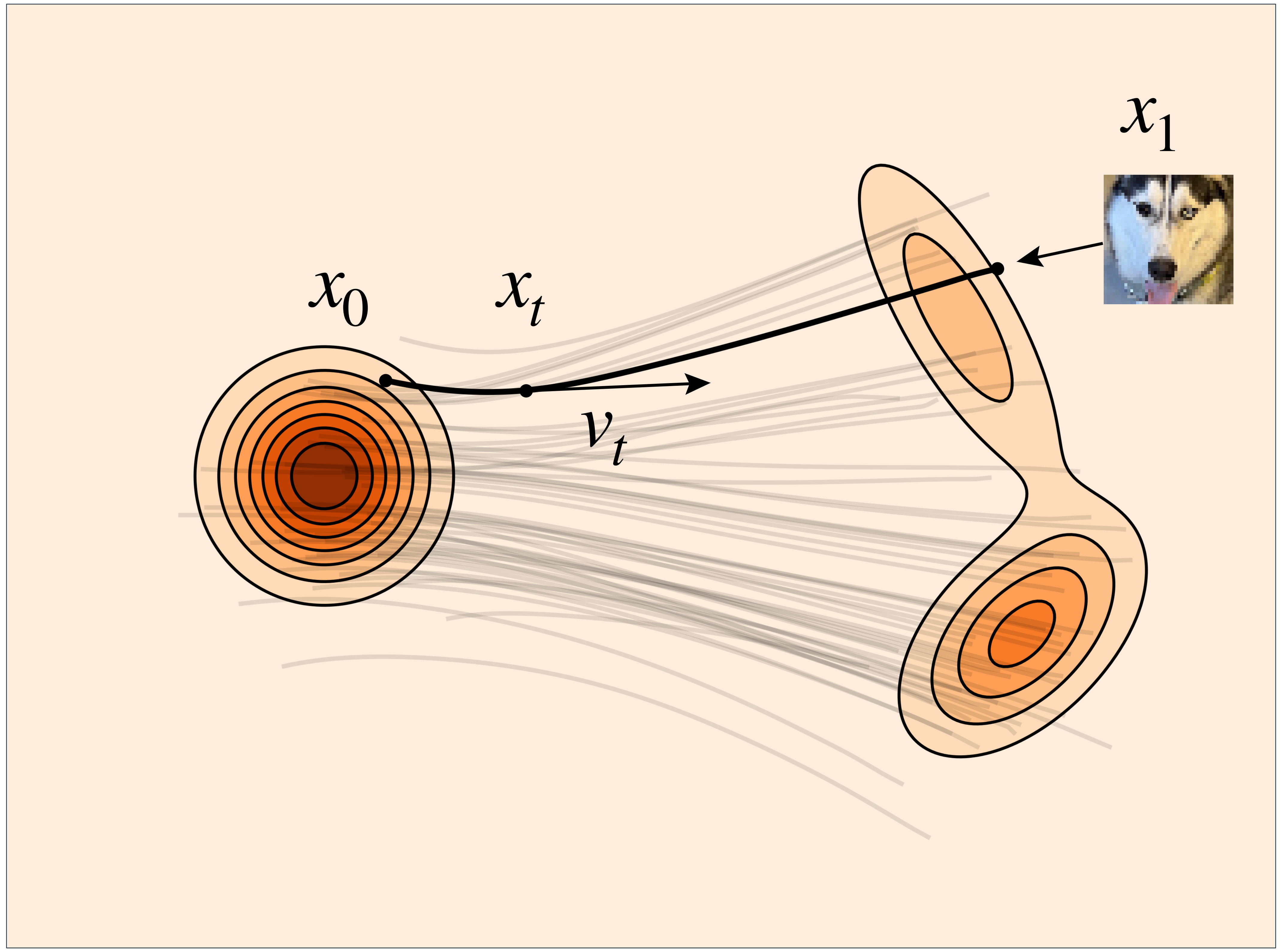
Flow matching says you can **pick any combination**, as long as it starts from a sample in the source (e.g. gaussian) and ends with a sample in the target distribution (image)

$$x_t = \alpha_t x_0 + \sigma_t x_1$$

$$x_0 \sim p_0(x) \qquad\qquad x_1 \sim p_1(x)$$

Flow training

- For each data x_1
  - Sample some noise x_0
  - Combine it however you want to get x_t
  - Now learn to predict the velocity at x_t
    - What is the velocity? Depends on how you got x_t

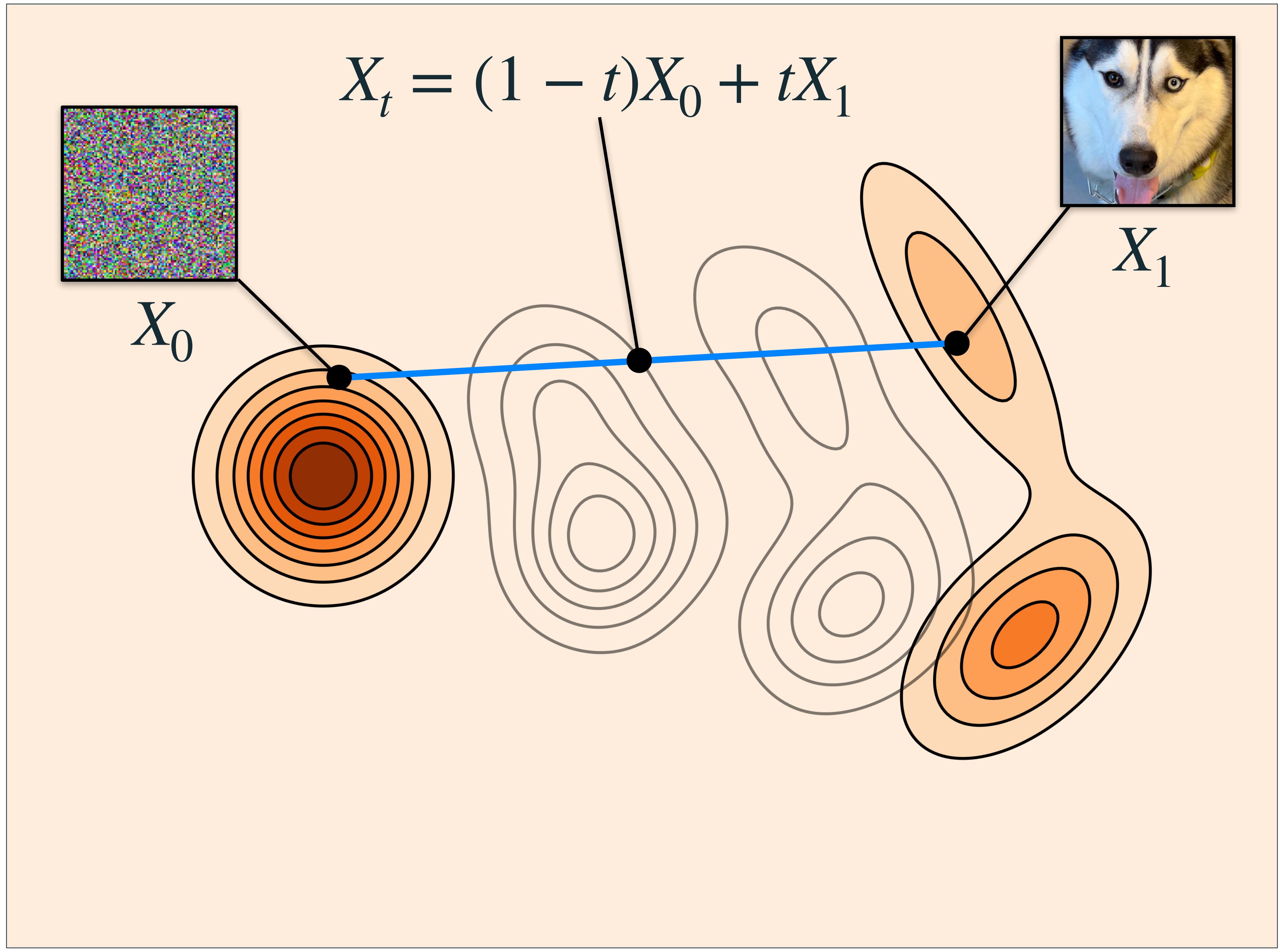


A Very Simple Way

Linear interpolation!

$$x_t = \alpha_t x_0 + \sigma_t x_1$$

$$x_t = (1-t)x_0 + t x_1$$



What is the velocity supervision?

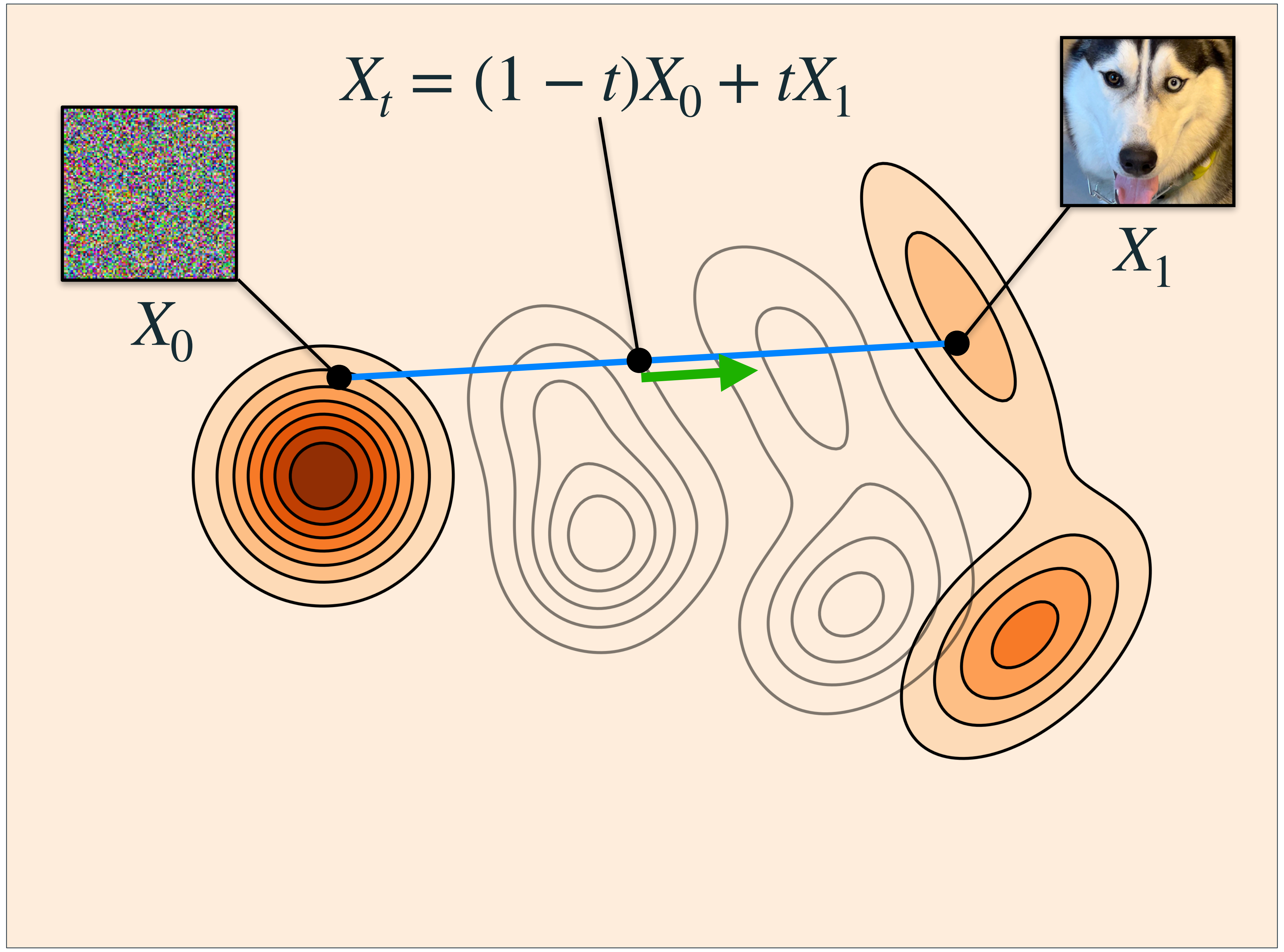
$x_t = \alpha_t x_0 + \sigma_t x_1$

$$\mathbb{E}_{t,X_0,X_1} \left\| u_t^\theta(X_t) - (X_1 - X_0) \right\|^2$$

$$x_t = (1-t)x_0 + t x_1$$

$$\frac{dx_t}{dt} = -x_0 + x_1$$

$$= x_1 - x_0$$



*Conditioned on a single sample

Inside a Training Loop

Flow Matching

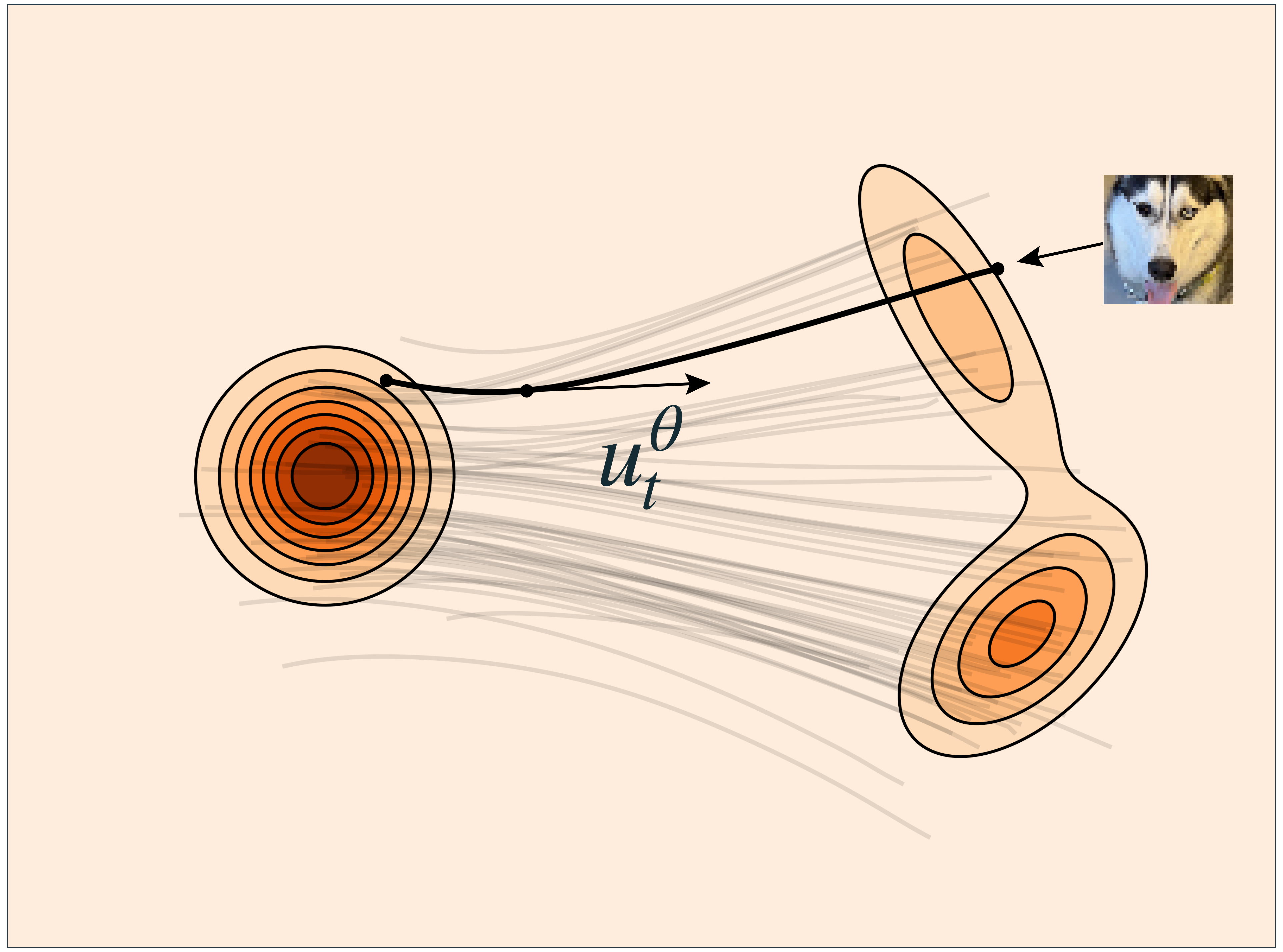
```
x = next(dataset)
t = torch.rand(1) # Sample timestep (0,1)
noise = torch.randn_like(x) # Sample noise
x_t = (1-t) * noise + (t) * x # Get noisy x_t

flow_pred = model(x_t, t) # Predict noise in x_t
flow_gt = x - noise # ground truth flow (w/ linear sched)
loss = F.mse_loss(flow_pred, flow_gt) # Update model
loss.backward()
optimizer.step()
```

Test-time sampling

- Just take a small step in the velocity

- Use any ODE Solver, i.e. integration you like, like Euler integration:

$$x_{t+\Delta t} = x_t + \Delta t \cdot \frac{dx}{dt}\bigg|_{x_t, t}$$



Sample
from $X_0 \sim p$

Inside Sampling Loop

```
velocity = model(x_t, t) # Predict noise in x_t
x_t = x_t + dt * velocity # Step in velocity
```

Model Parametrization

- Simplest — Just make your NN predict the velocity, which with the simple linear interpolation is always just x1 - x0
- Other options: Make it output the noise added or the clean image. Possible with some arithmetics
- But will have some 1/t or 1/(1-t) terms, which is annoying at the edges

Training: Model parameterization

- You can make your network output undo the noise in many different ways, predicting x, v, noise, or flow

$$v_t = \alpha_t x_1 - \sigma_t x_0 \qquad \begin{aligned} u_t &= x_1 - x_0 = \epsilon - x_0 \\ u_t &= x_t - x_0 \end{aligned}$$

- These are all equivalent because of the linear relationship with x_t. You can derive all of these as long as you know one of them

$$x_t = \alpha_t x_0 + \sigma_t x_1$$

- For example

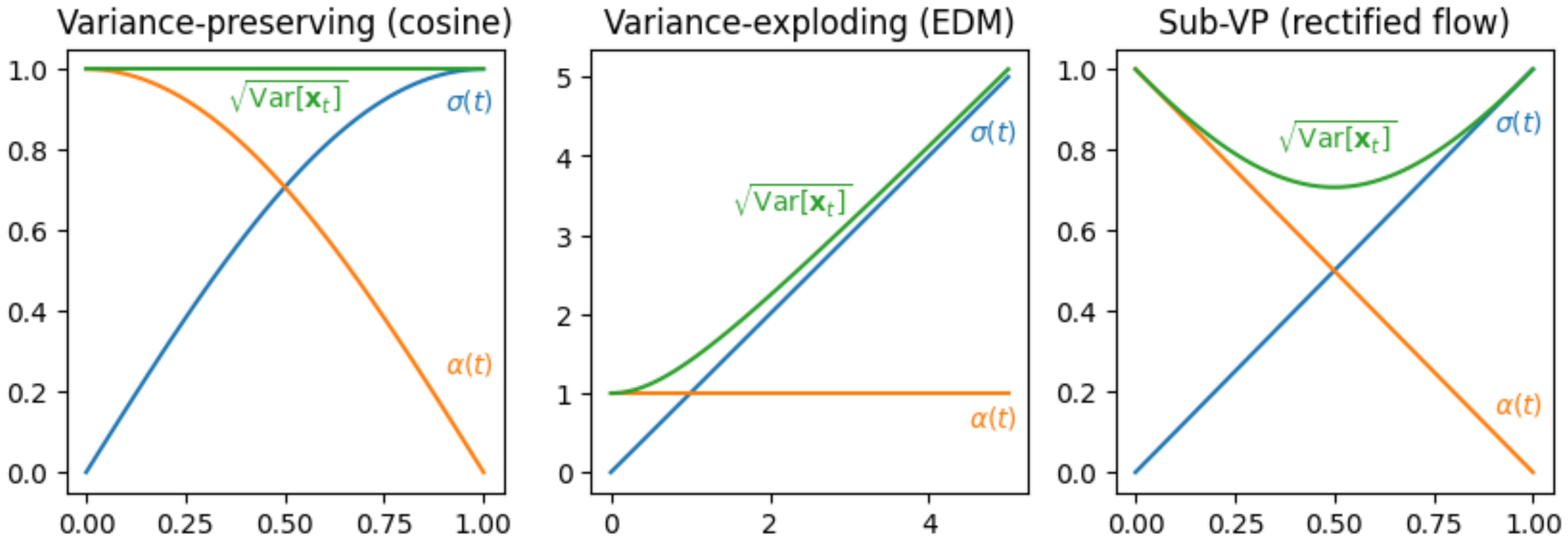
$$\mathbb{E}[(\hat{\mathbf{x}}_0 - \mathbf{x}_0)^2] = \mathbb{E}\left[\left(\frac{\mathbf{x}_t - \sigma(t)\hat{\varepsilon}}{\alpha(t)} - \frac{\mathbf{x}_t - \sigma(t)\varepsilon}{\alpha(t)}\right)^2\right] = \mathbb{E}\left[\frac{\sigma(t)^2}{\alpha(t)^2}\left(\hat{\varepsilon} - \varepsilon\right)^2\right].$$

Slides from Yaron Lipman

Other options lead to prior works

$$x_t = \alpha_t x_0 + \sigma_t x_1$$

- Other choices:
  - Preserve variance (VP-ODE) - DDPM
  - Exploding variance (VE-ODE) - Score Matching/DDIM
  - Linear interpolation (Flow Matching, Rectified Flow)



Training: Flow Matching vs. Diffusion

Algorithm 1: Flow Matching training.

Input : dataset q, noise p
Initialize v^θ
while *not converged* **do**
- $t \sim \mathcal{U}([0,1])$ ▷ sample time
- $x_1 \sim q(x_1)$ ▷ sample data
- $x_0 \sim p(x_0)$ ▷ sample noise
- $x_t = \Psi_t(x_0|x_1)$ ▷ conditional flow
- Gradient step with $\nabla_\theta \|v_t^\theta(x_t) - \dot{x}_t\|^2$

Output: v^θ

Algorithm 2: Diffusion training.

Input : dataset q, noise p
Initialize s^θ
while *not converged* **do**
- $t \sim \mathcal{U}([0,1])$ ▷ sample time
- $x_1 \sim q(x_1)$ ▷ sample data
- $x_t = p_t(x_t|x_1)$ ▷ sample conditional prob
- Gradient step with $\nabla_\theta \|s_t^\theta(x_t) - \nabla_{x_t} \log p_t(x_t|x_1)\|^2$

Output: v^θ

$p_t(x_t | x_1)$ general
$p(x_0)$ is general

$p_t(x_t | x_1)$ closed-form from of SDE $dx_t = f_t dt + g_t dw$

- *Variance Exploding*: $p_t(x | x_1) = \mathcal{N}(x | x_1, \sigma_{1-t}^2 I)$
- *Variance Preserving*: $p_t(x | x_1) = \mathcal{N}(x | \alpha_{1-t} x_1, (1 - \alpha_{1-t}^2) I)$

$$\alpha_t = e^{-\frac{1}{2}T(t)}$$

$p(x_0)$ is Gaussian
$p_0(\cdot | x_1) \approx p$

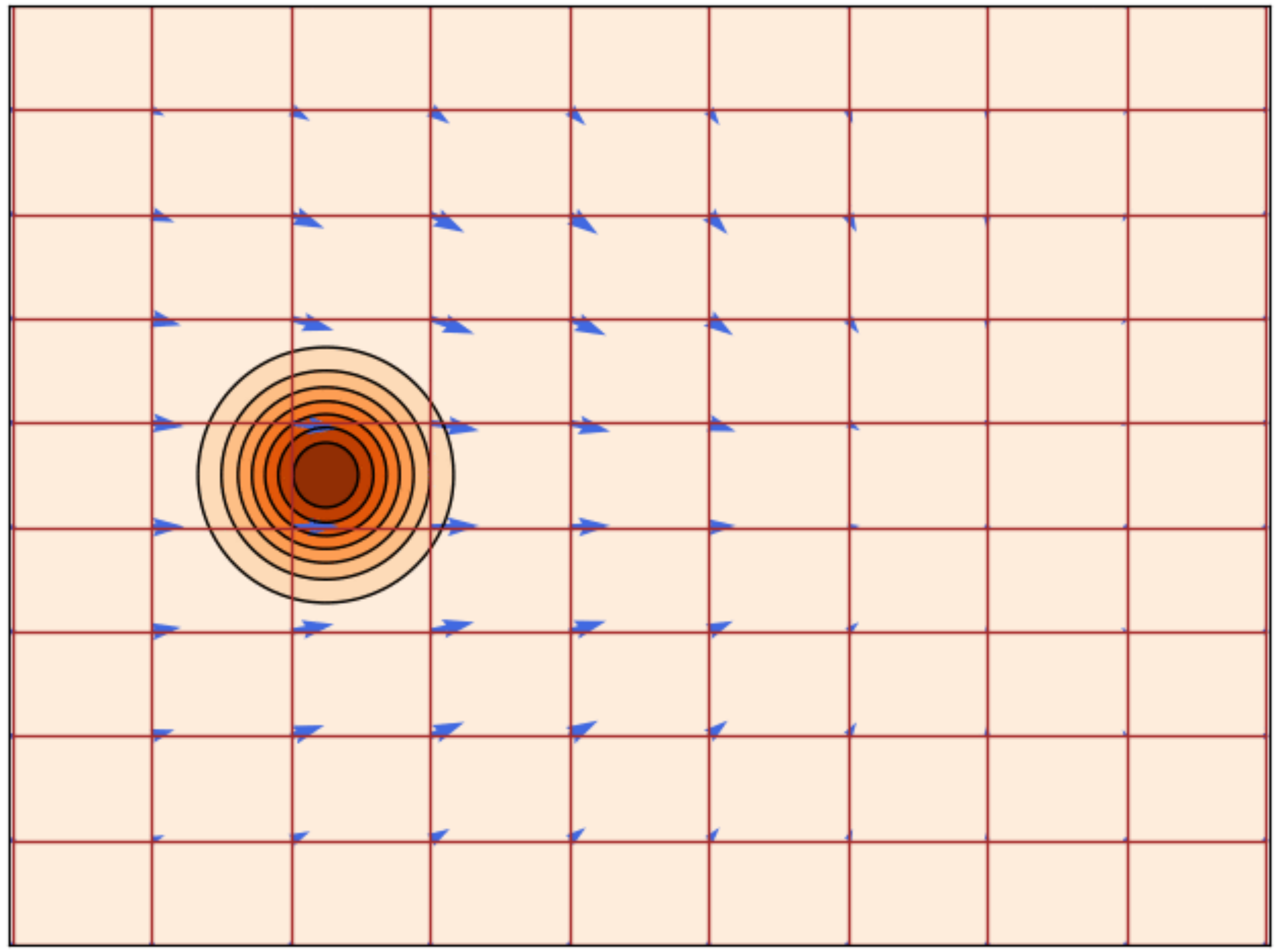
Why does Flow Matching work?

FM —> predict the velocity conditioned on a single sample

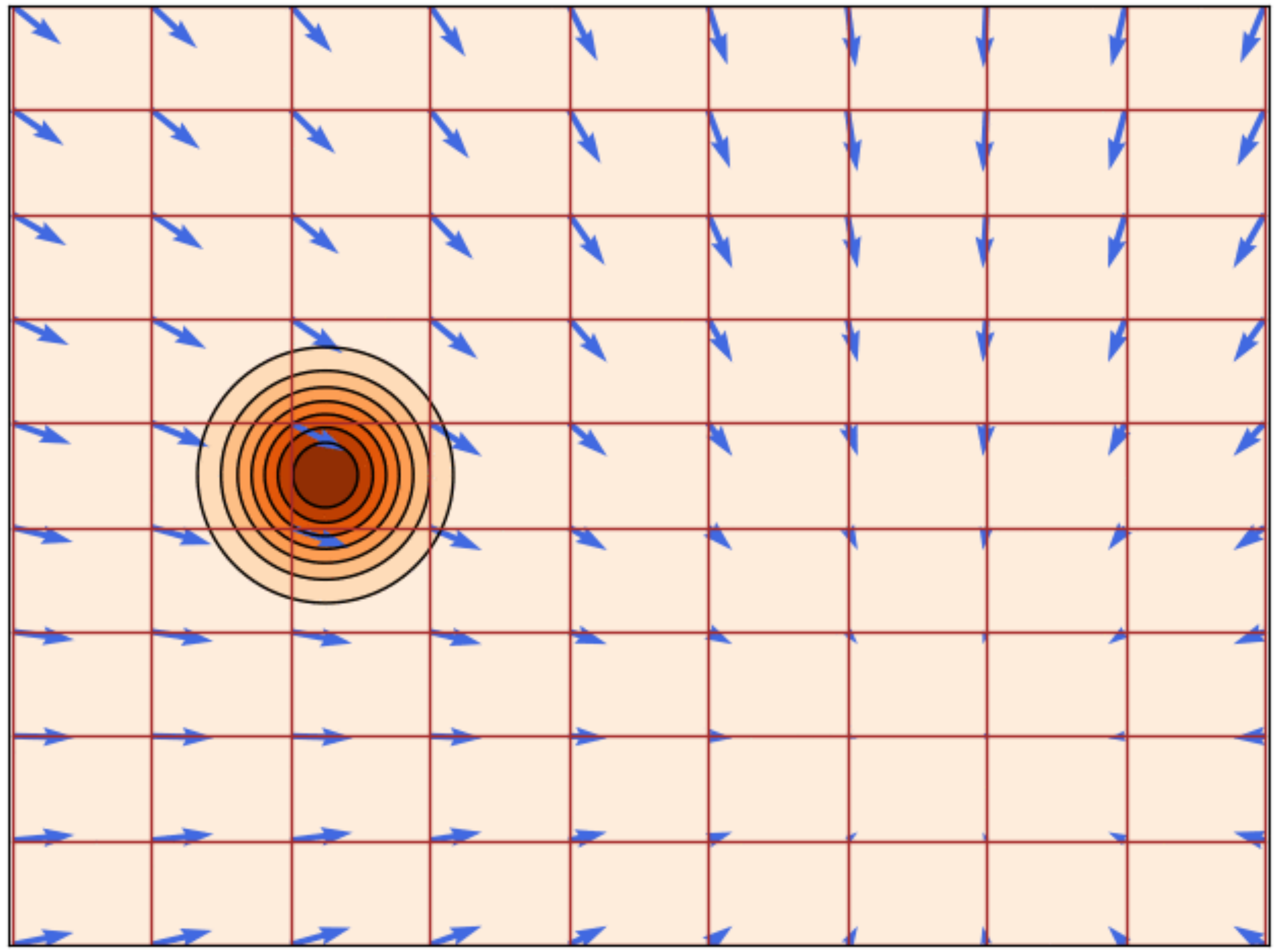
Recap

- What we want is the flow (velocity field) that takes samples from p_0 to p_1 when integrated (called Marginal Flow)

- But what we did was to train flow for each sample (Conditional Flow)



Marginal Flow (what we want)



Conditional Flow (what we trained)

Turns out Gradient is the same!

- **Flow Matching loss:**

$$\mathscr{L}_{\mathrm{FM}}(\theta) = \mathbb{E}_{t,X_t} \left\| u_t^\theta(X_t) - u_t(X_t) \right\|^2$$

We can't do this bc we don't know what ground truth flow is

- **Conditional Flow Matching loss**:

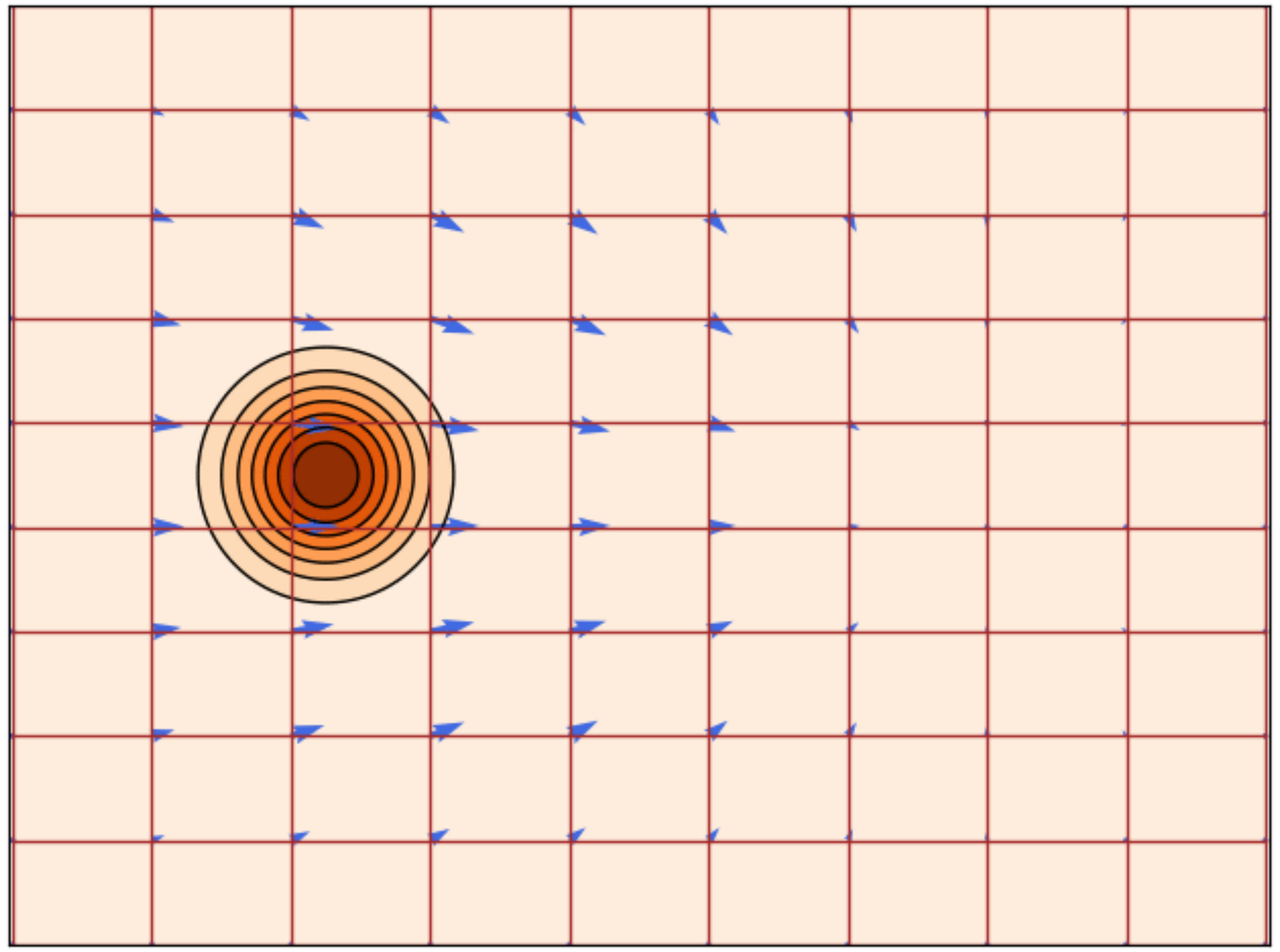
$$\mathscr{L}_{\mathrm{CFM}}(\theta) = \mathbb{E}_{t,X_1,X_t} \left\| u_t^\theta(X_t) - u_t(X_t \mid X_1) \right\|^2$$

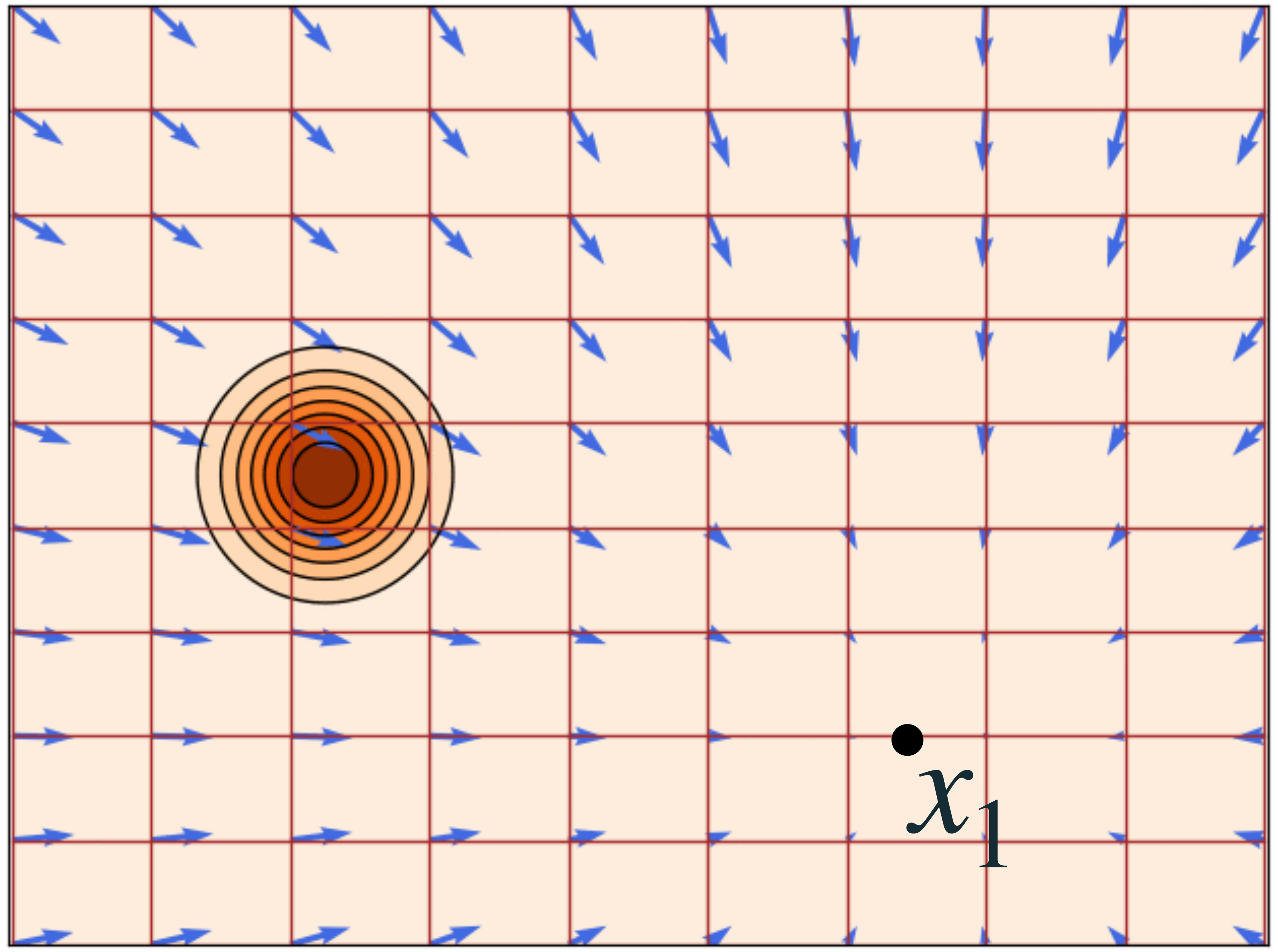
Theorem: Gradient of the losses are equivalent (!!!),

$$\nabla_\theta \mathscr{L}_{\mathrm{FM}}(\theta) = \nabla_\theta \mathscr{L}_{\mathrm{CFM}}(\theta)$$

Build flow from conditional (per-sample) flows

Generate a single target point





$$X_t = \psi_t(X_0 \mid x_1) = (1-t)X_0 + tx_1$$

$$p_t(x) = \mathbb{E}_{X_1} p_{t|1}(x \mid X_1) \longleftarrow p_{t|1}(x \mid x_1)$$ conditional probability

average

$$u_t(x) = \mathbb{E}\left[u_t(X_t \mid X_1) \,\middle|\, X_t = x\right] \longleftarrow u_t(x \mid x_1)$$ conditional velocity

Why does this work?

- High level: You can average conditional flow (marginalization)

$$u_t(x) = \mathbb{E}\left[u_t(X_t \mid X_1) \mid X_t = x\right]$$

Marginal flow
(what we want)

Conditional flow
(easy to obtain from each sample)

$$= \int u_t(x \mid x_1) p(x_1 \mid X_t = x) dx_1$$

$$u_t(x) = \int u_t(x \mid x_1) \frac{p_t(x \mid x_1) q(x_1)}{p_t(x)} dx_1$$

Bayes rule with notation:

$$p(x_1) = q(x_1)$$

It's all just weighted average

$$u_t(x) = \int u_t(x \mid x_1) \frac{p_t(x \mid x_1) q(x_1)}{p_t(x)} dx_1$$

Weight of sample x0 = 1

$$u_t(x_t) = \sum u_t(x_t \mid x_1) \frac{p_t(x_t \mid x_1)}{p_t(x)} q(x_1)$$

Marginal flow

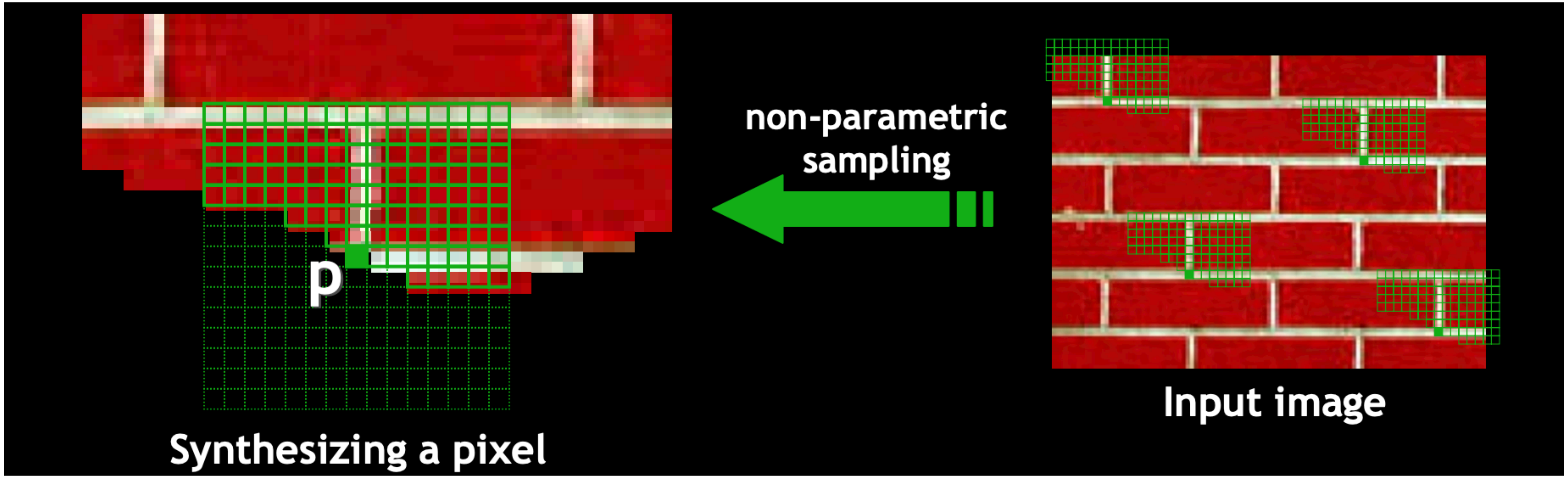
Path from x_1 to x_t

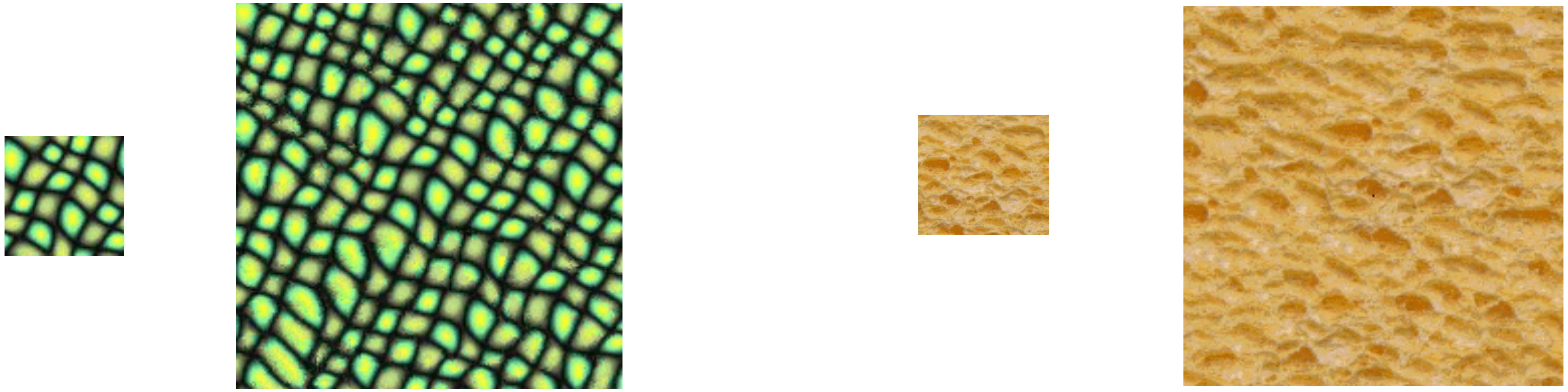
Path weight

Just a weighted average of the flow to each data sample!!!!!
You can actually do this non-parametrically.

See interactive visualization at https://decentralizeddiffusion.github.io/

Efros & Leung ICCV'99





- Non-parametric patch-based NN sampling to fill in missing details & generate textures

Scene Completion Using Millions of Photographs

James Hays Alexei A. Efros
Carnegie Mellon University

Original Image Input Scene Matches Output

Figure 1: Given an input image with a missing region, we use matching scenes from a large collection of photographs to complete the image.

- Non-parametric patch-based NN approach to fill in missing details with **lots of Data!**

Key message

- **One can minimize the diffusion objective (marginal flow) non-parametrically and perfectly minimize the loss.**

- But there is no learning! No ability to generate new images!

- i.e. Exactly minimizing this objective does not guarantee interpolation/compositionally, learning of the image manifold!

- Parametrizing it with neural networks results in magic smoothing to generate new images and interpolate between them. Exactly what makes this possible still active area of research