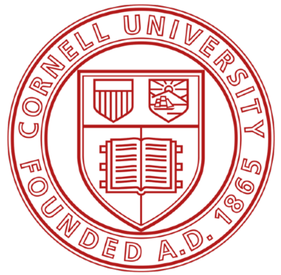
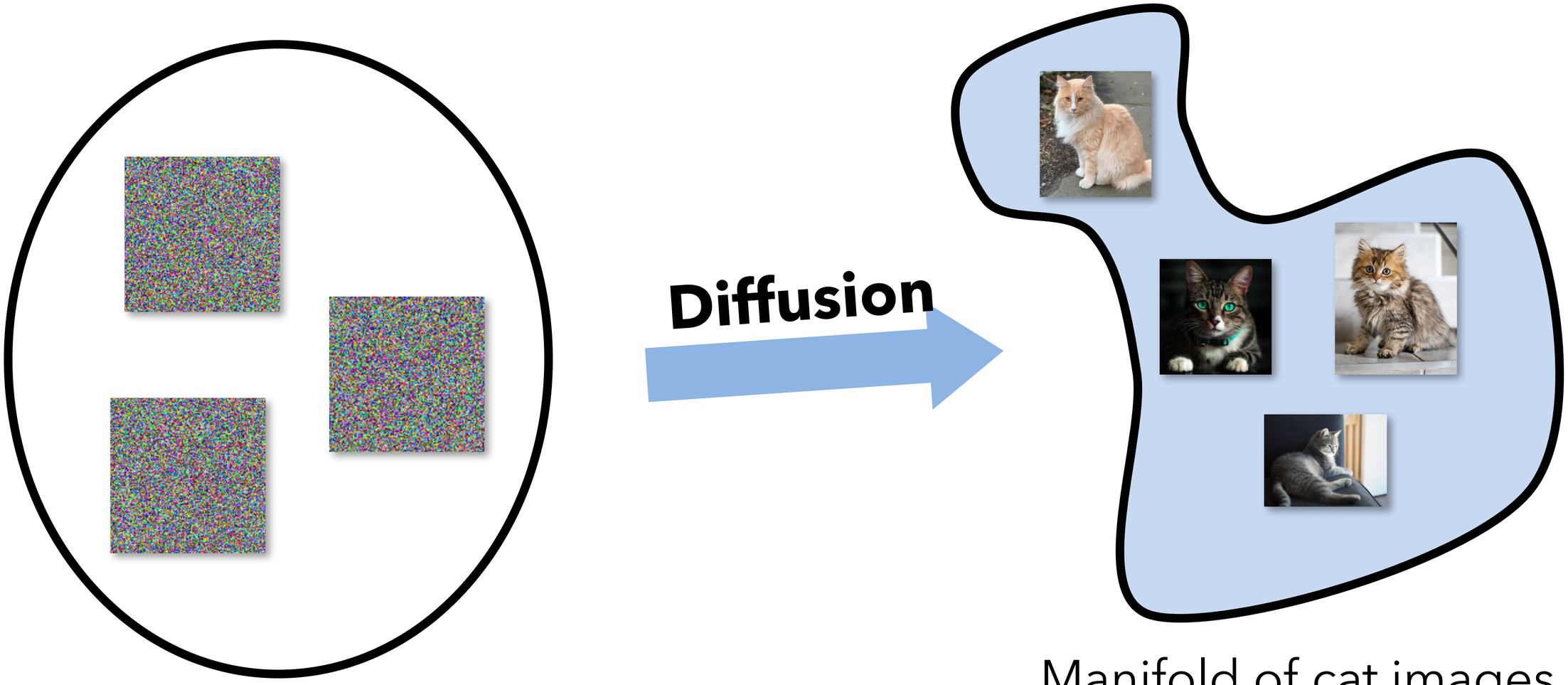
# Lecture 16: Flow matching

CS 5788: Introduction to Generative Models

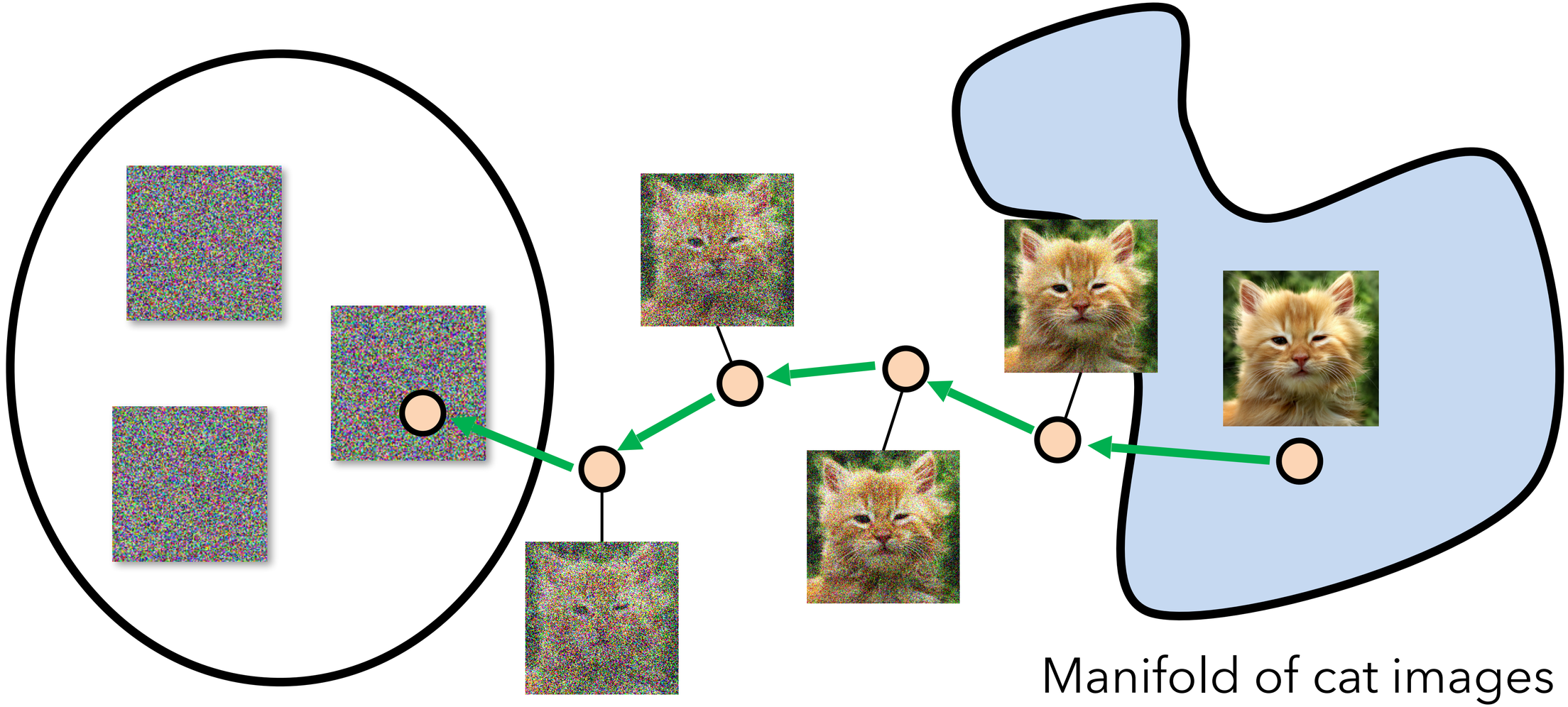



Most slides from Angjoo Kanazawa (also includes slides from Yaron Lipman and Steve Seitz)

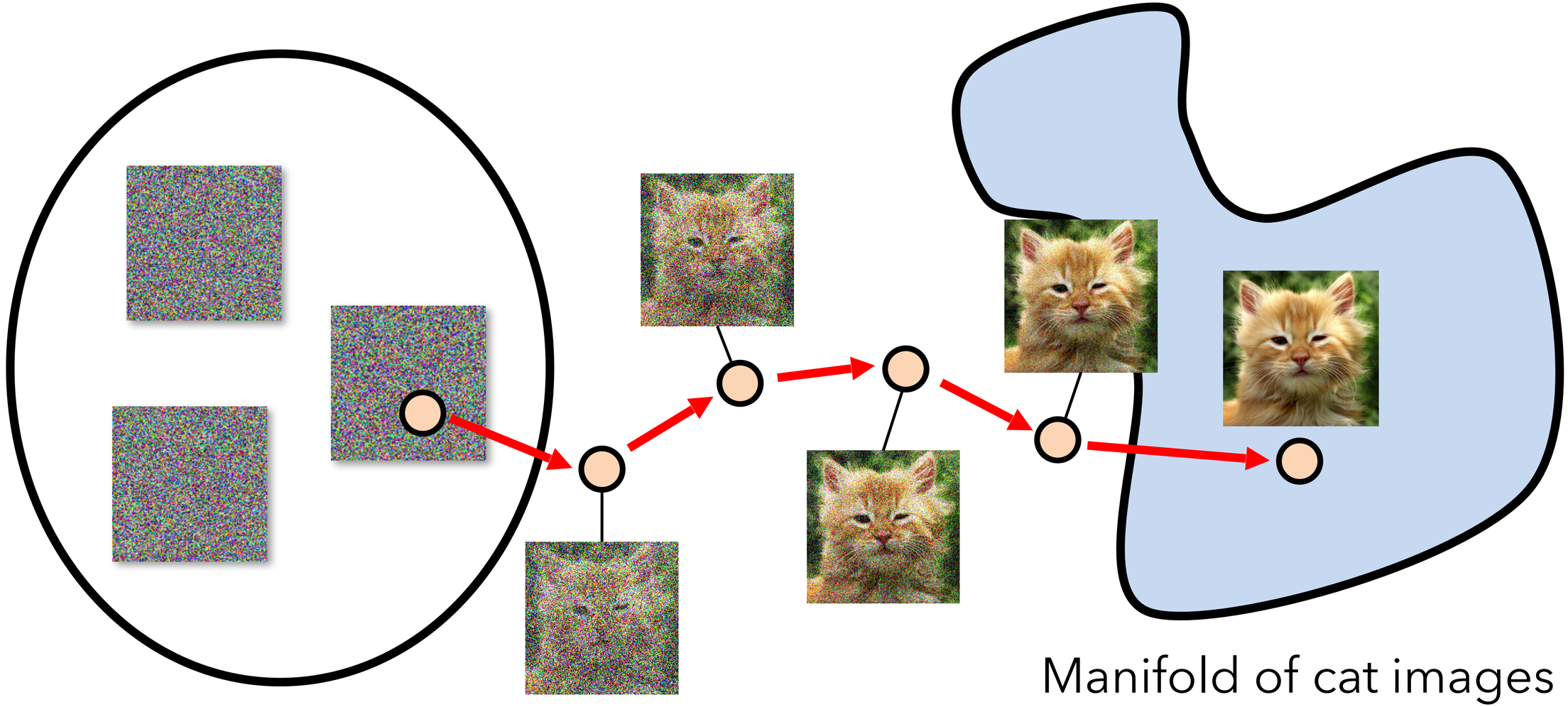
# Recall: diffusion models




Slide source: N. Snavely & S. Seitz




Slide source: N. Snavely & S. Seitz




Slide source: N. Snavely & S. Seitz

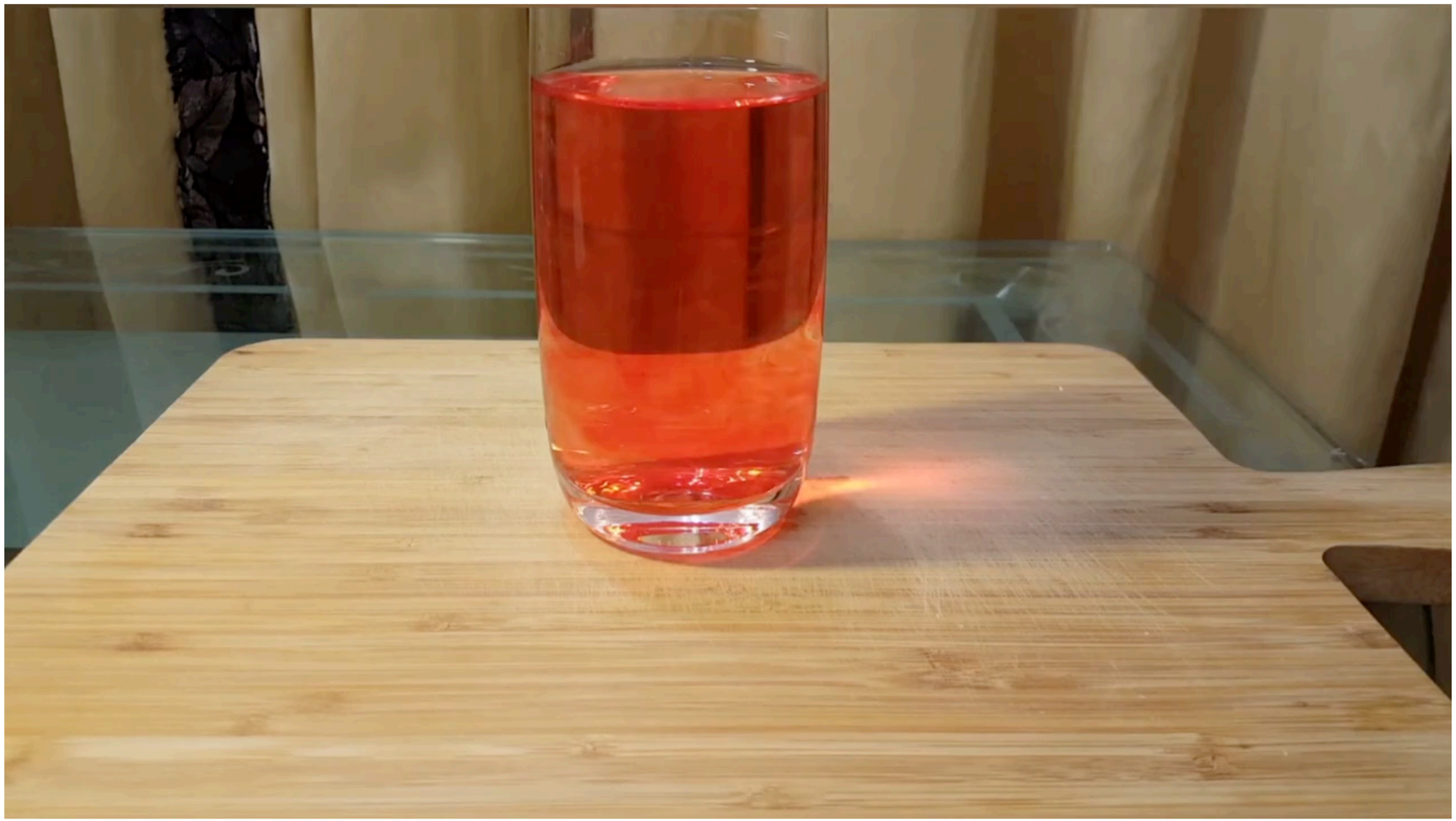
# Physical motivation for diffusion

Heat Diffusion

Source: Angjoo Kanazawa

# Physical motivation for diffusion

Reversing the process



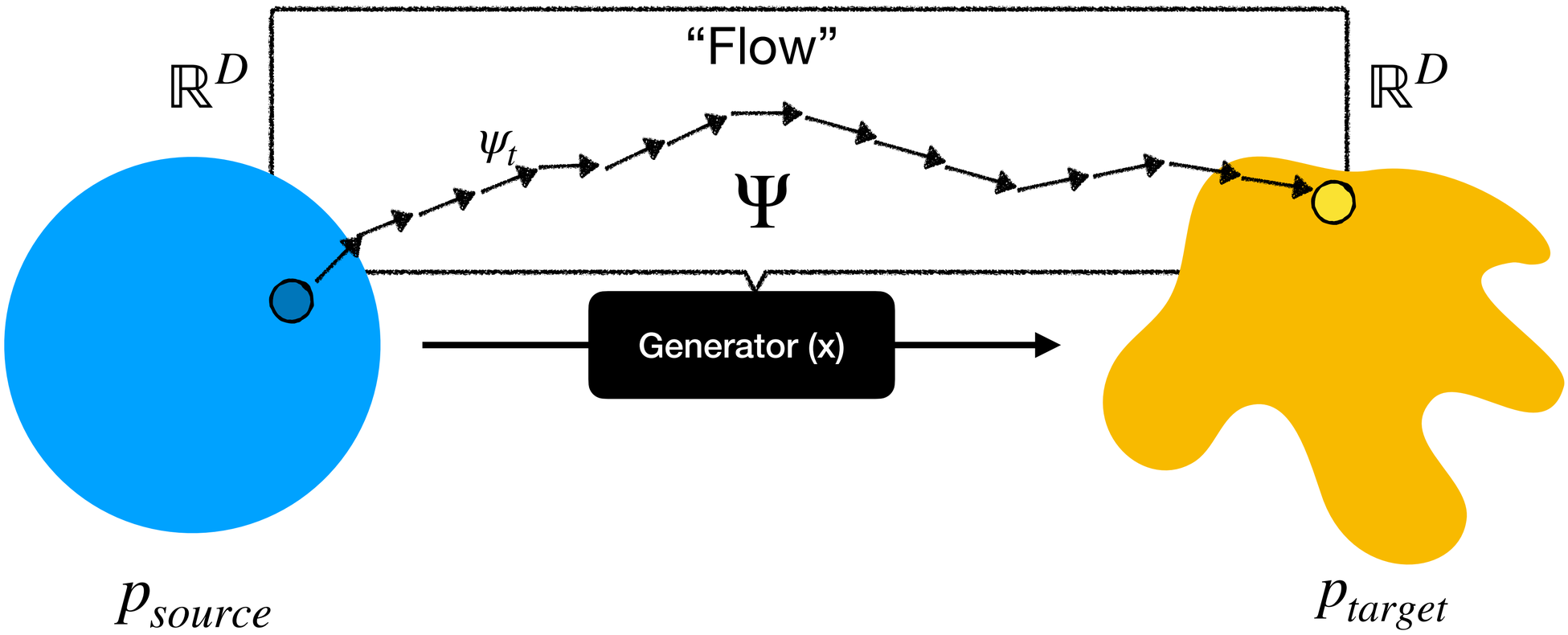
Source: Angjoo Kanazawa

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

# Flow matching

- Another perspective on generative modeling
- Widely used in state-of-art image and video generators
- Training and sampling process is quite similar to diffusion models.
  - In fact, many popular formulations are equivalent [Gao et al., “Diffusion Meets Flow Matching”, 2024].
- This formulation makes a number of things simpler, though!

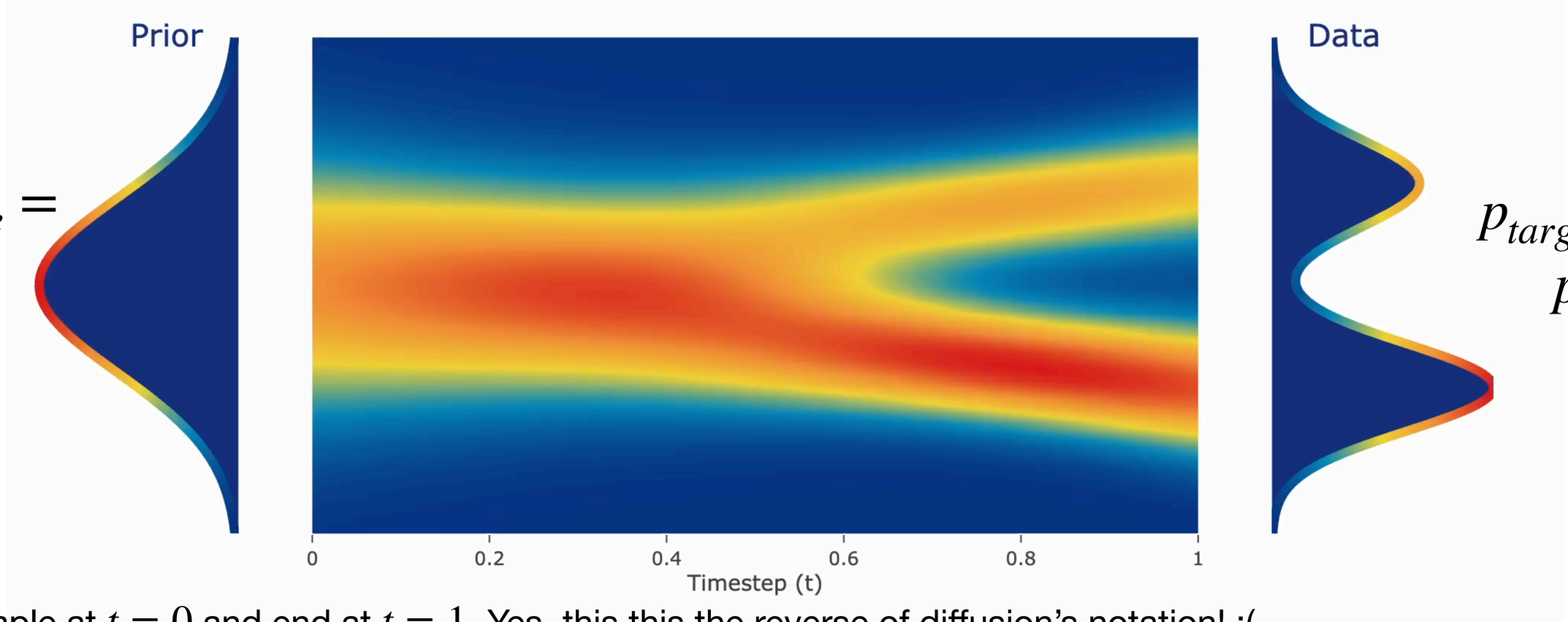
# Flow matching models




Adapted from: Angjoo Kanazawa

# What is a Flow?

$p_{source} = p_0$

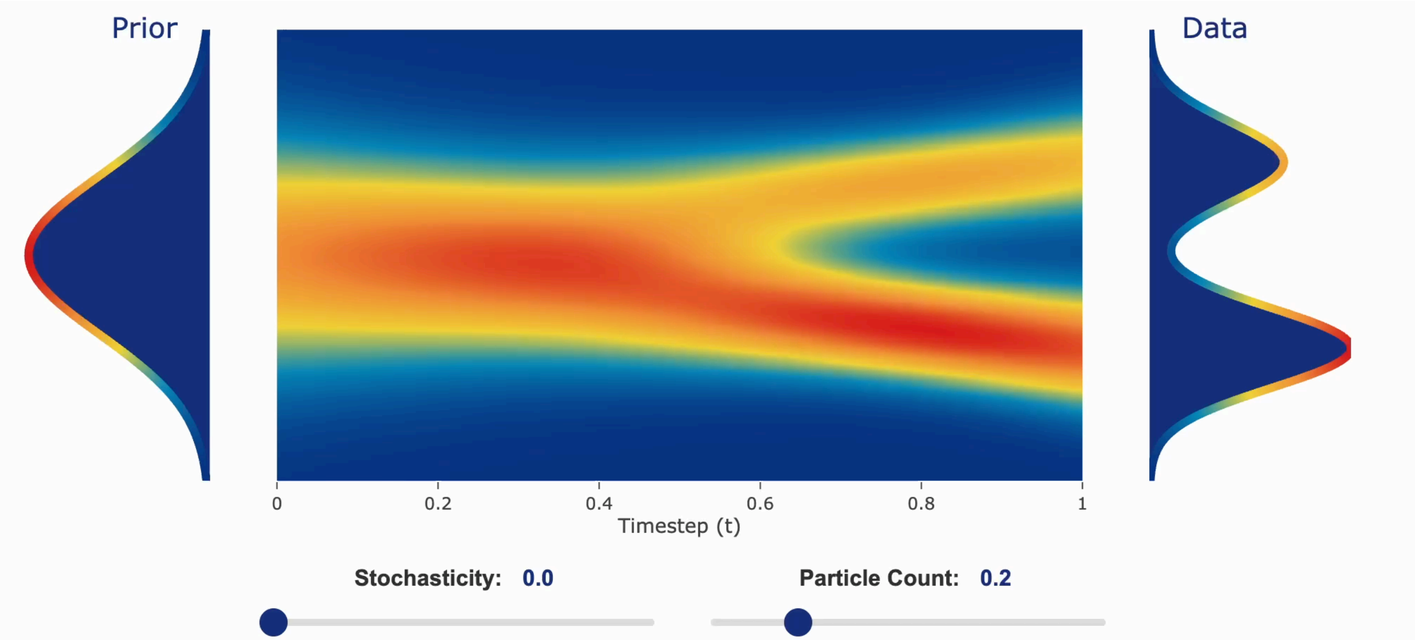



$p_{target} = p_1$

- Sample at $t = 0$ and end at $t = 1$. Yes, this this the reverse of diffusion's notation! :(
- 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

Adapted from: Angjoo Kanazawa

# Riding the river = integration



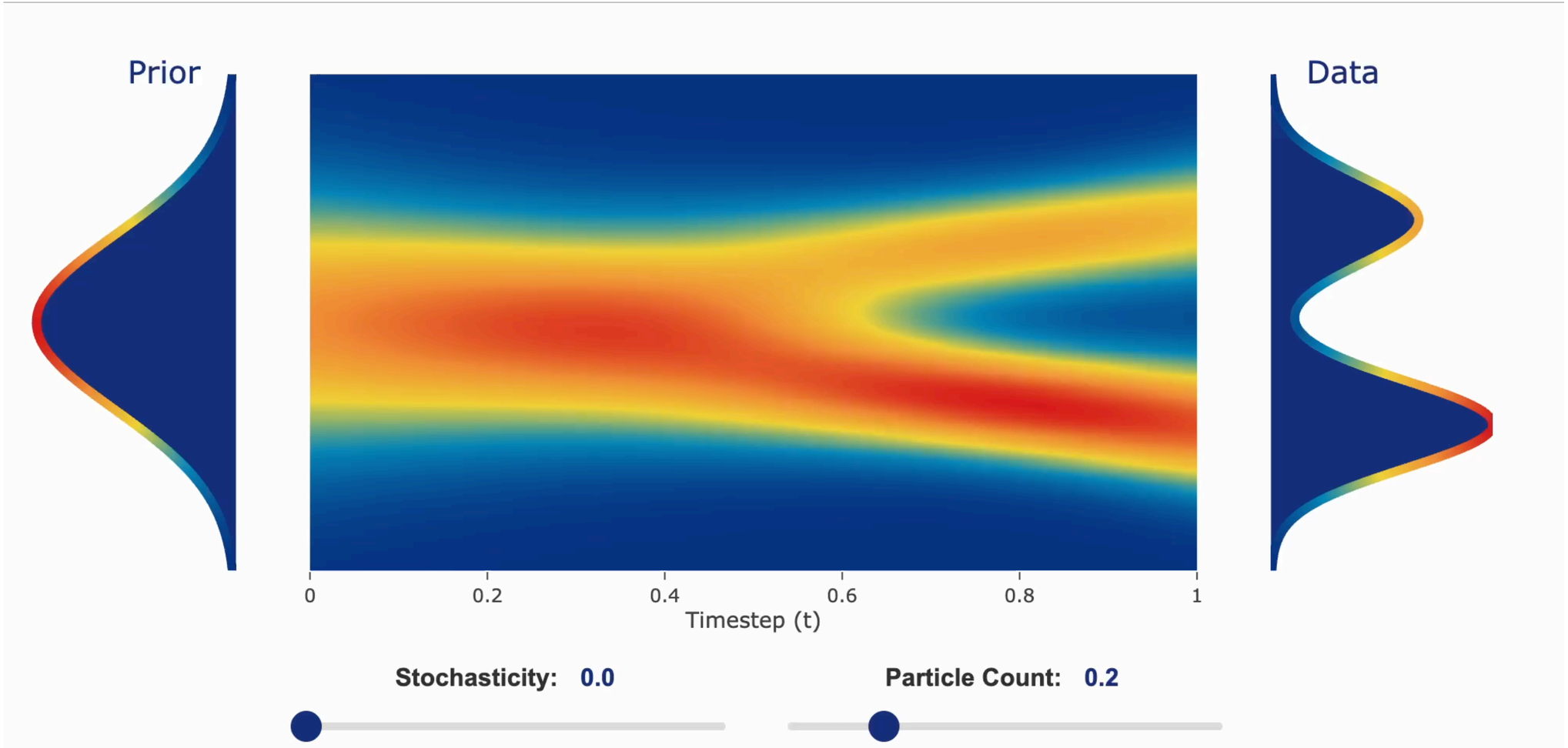

Simplest *Euler Integration*:

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

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

Source: Angjoo Kanazawa

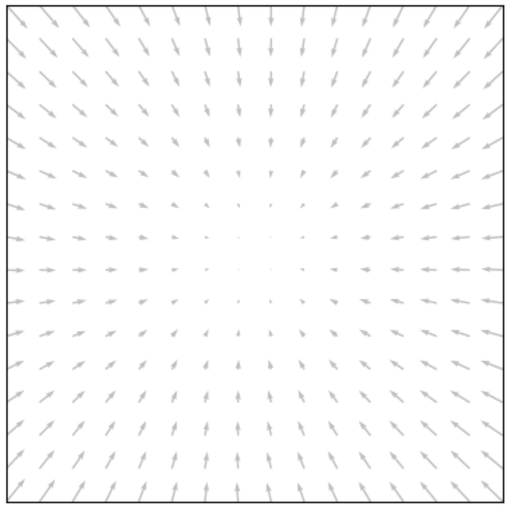
# How can we learn this flow?



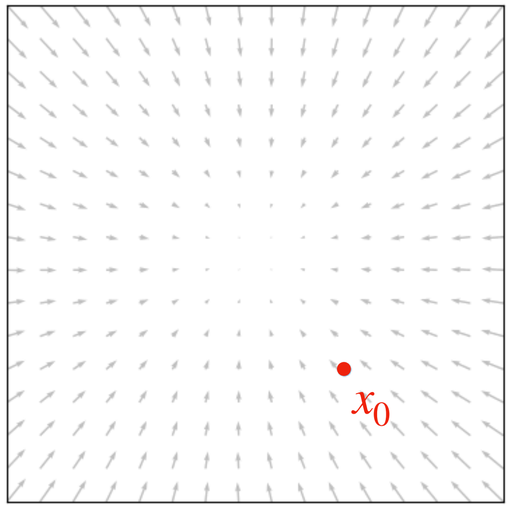

Source: Angjoo Kanazawa

# In 2D

$v_t(x)$

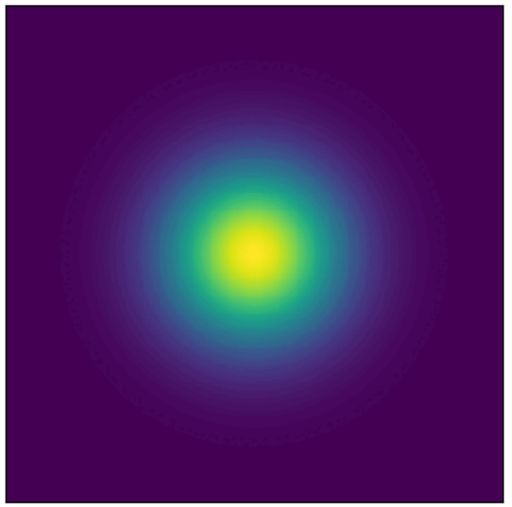


$x_t = \Psi_t(x_0)$




$x_0 \sim p$

$x_t \sim p_t$



**Flow ODE**

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

*Slides from Yaron Lipman and Angjoo Kanazawa*

[Chen et al. 2018]

# How can we train this?

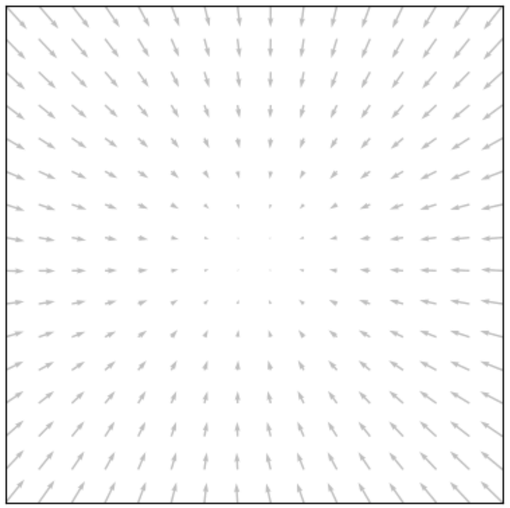
$x_0 \sim p$

$\downarrow$

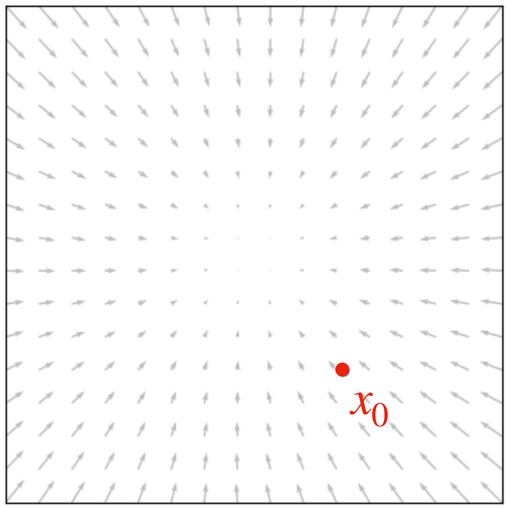
$x_t \sim p_t$

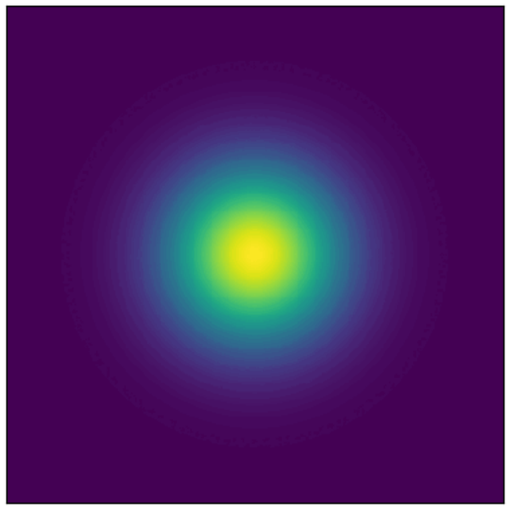
Samples of $p_1$

$v_t(x)$

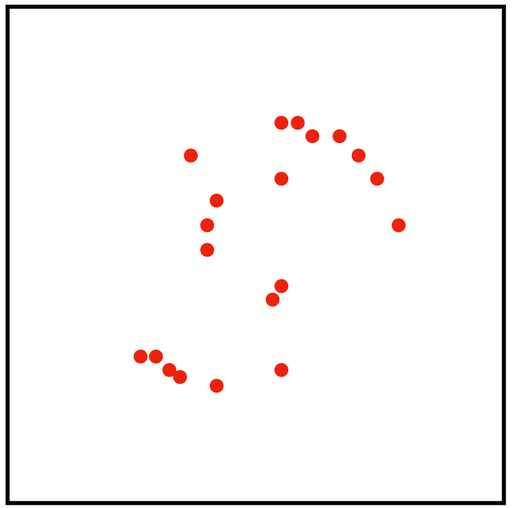


$x_t = \Psi_t(x_0)$



$q(x)$



**Flow ODE**

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

*Slides from Yaron Lipman and Angjoo Kanazawa*

**[Chen et al. 2018]**

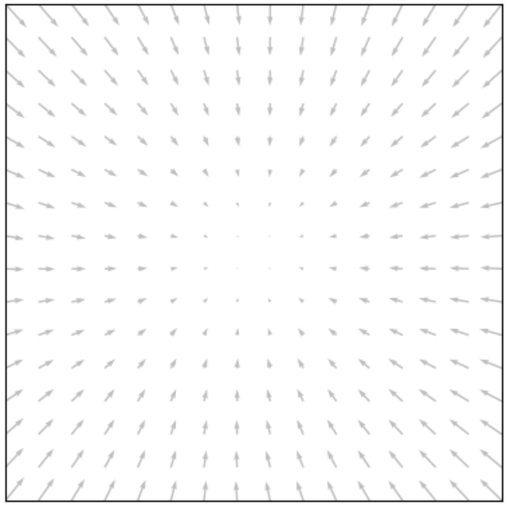
# Caveat: Continuity equation

$$x_0 \sim p$$

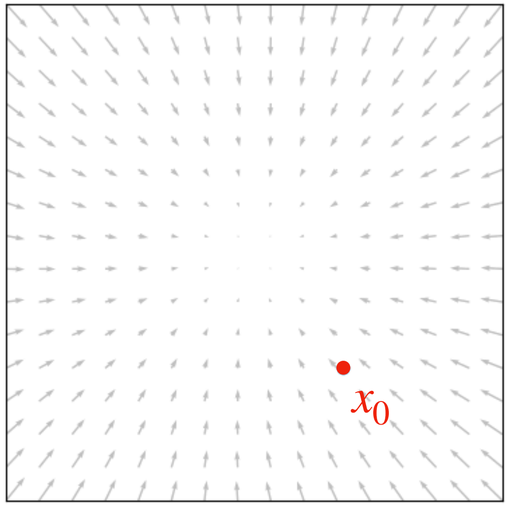
$$\downarrow$$

$$x_t \sim p_t$$

$v_t(x)$



$x_t = \Psi_t(x_0)$

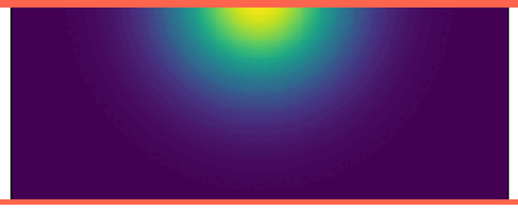



**Flow ODE**

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

**Continuity Equation PDE (*fixed x*)**

$$\frac{d}{dt} p_t(x) = -\operatorname{div}(p_t v_t)$$



where $\operatorname{div} v(x) = \sum_{i=1}^{d} \frac{\partial}{\partial x_i} v_i(x)$

- For probability path to be valid, it needs to conserve probability mass.
- In the river, analogy you cannot add or remove water
- It has to come from somewhere and go somewhere

*Slides from Yaron Lipman and Angjoo Kanazawa*

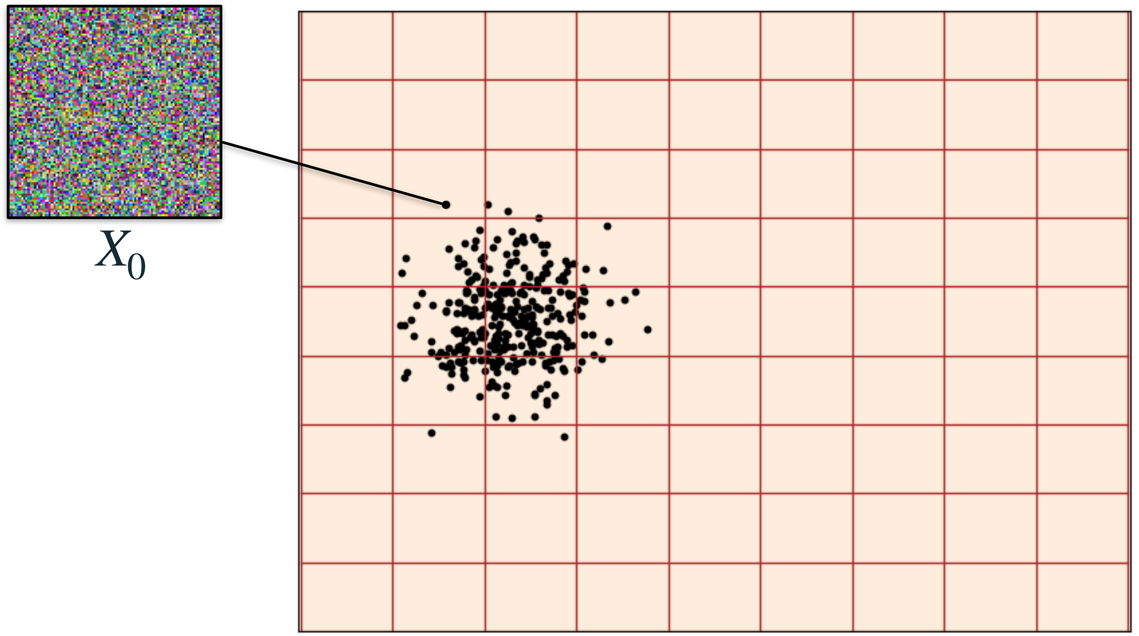
**[Chen et al. 2018]**

# Treating the flow as a warping function

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

Warping

Source $X_0 \sim p$




*Slides from Yaron Lipman and Angjoo Kanazawa*

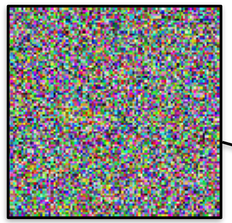
# Early work trained flow by maximum likelihood

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

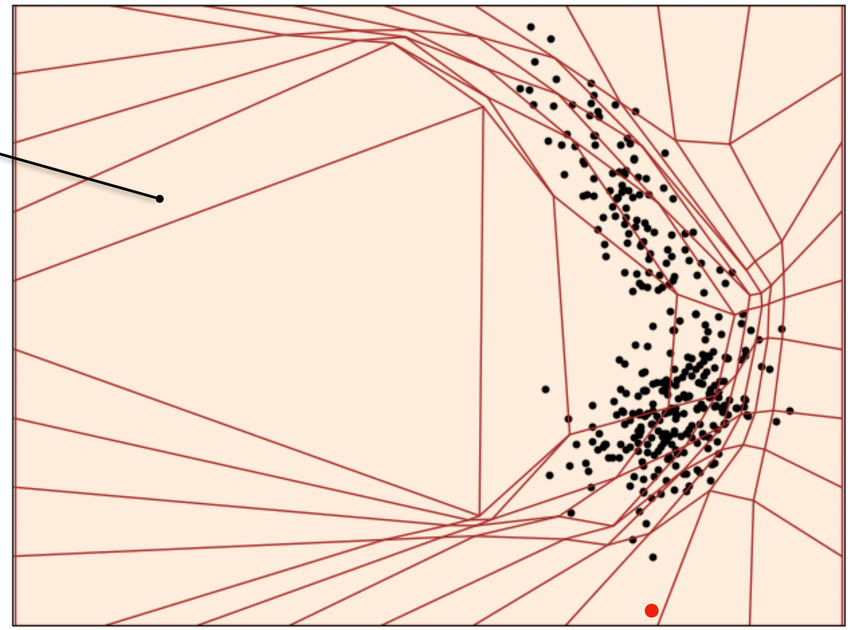
$$X_t = \psi_t(X_0)\,, \quad t \in [0,1]$$

Warping

Source $X_0 \sim p$



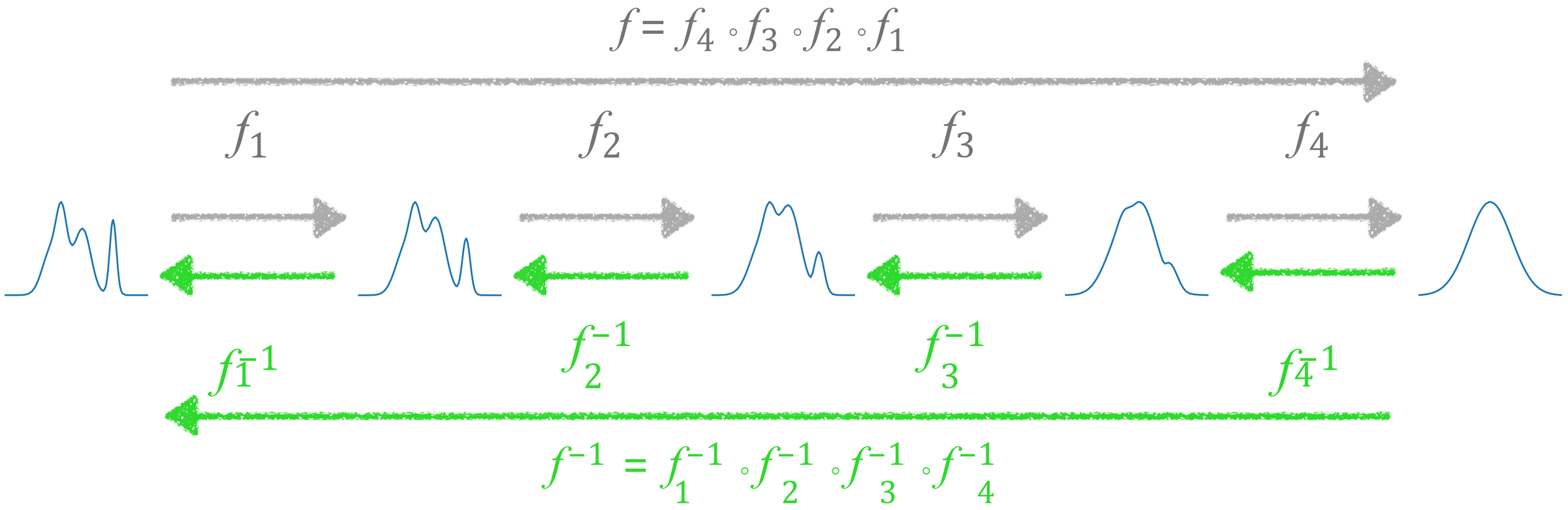
$X_0$




- Chaining $\psi_t$ needs to satisfy the continuity equation.
- This requires ODE integration *during training* with invertible neural networks.
- This is a *Continuous* Normalizing Flow (CNF) model!

*Slide adapted from Yaron Lipman and Angjoo Kanazawa*

# Recall: normalizing flow (with discrete layers)



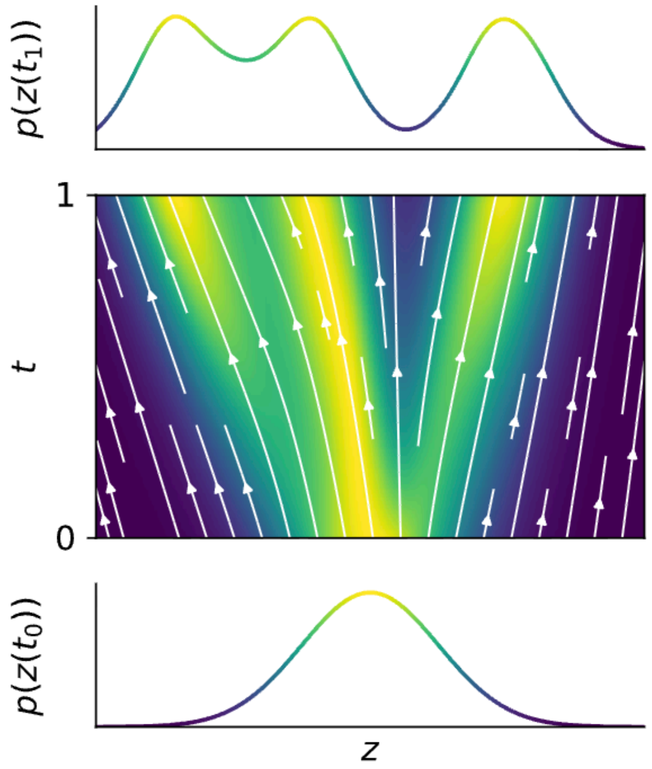

Source: M. Brubaker & Köthe

# Early flow learning methods

- Treat like a Continuous Normalizing Flow (CNF) model (e.g., [Chen et al., “Neural ODE”, 2018])
- Tries to directly deal with this continuity equation constraint
- Very slow to train (need to integrate while training)
- Requires other constraints like invertibility of the $\psi_t$ network.
- Due to these limitations, they are hard to train and require special architectures.

Source: Angjoo Kanazawa

# Continuous normalizing flow example





[Grathwohl et al., “FFJORD”, 2018]

# Instead, model flow with velocity

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

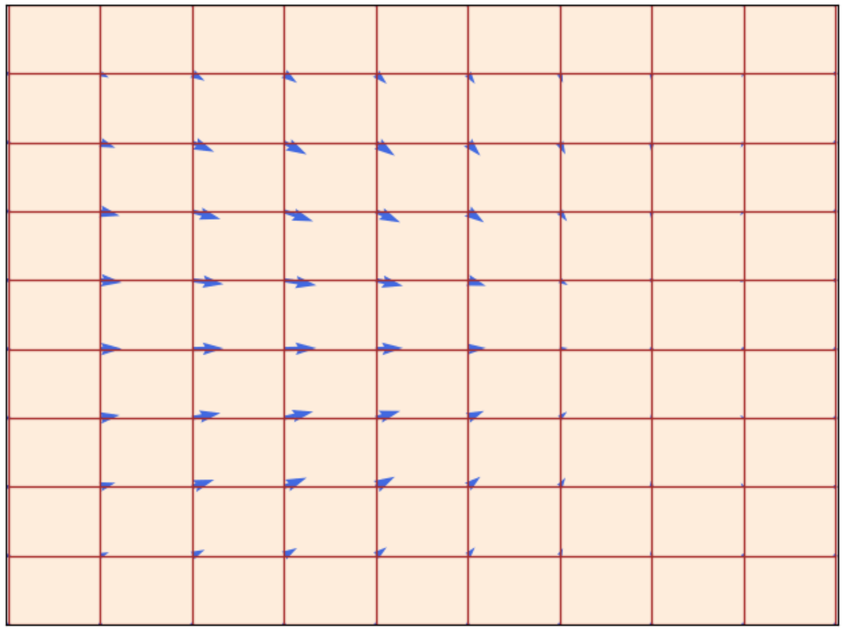
Flow

$$\psi_t(x)$$

**Solve ODE** ↑ ↓ **Differentiate**

$$u_t(x)$$

Velocity



*Slides from Yaron Lipman*

Source: Angjoo Kanazawa

# Flow Matching

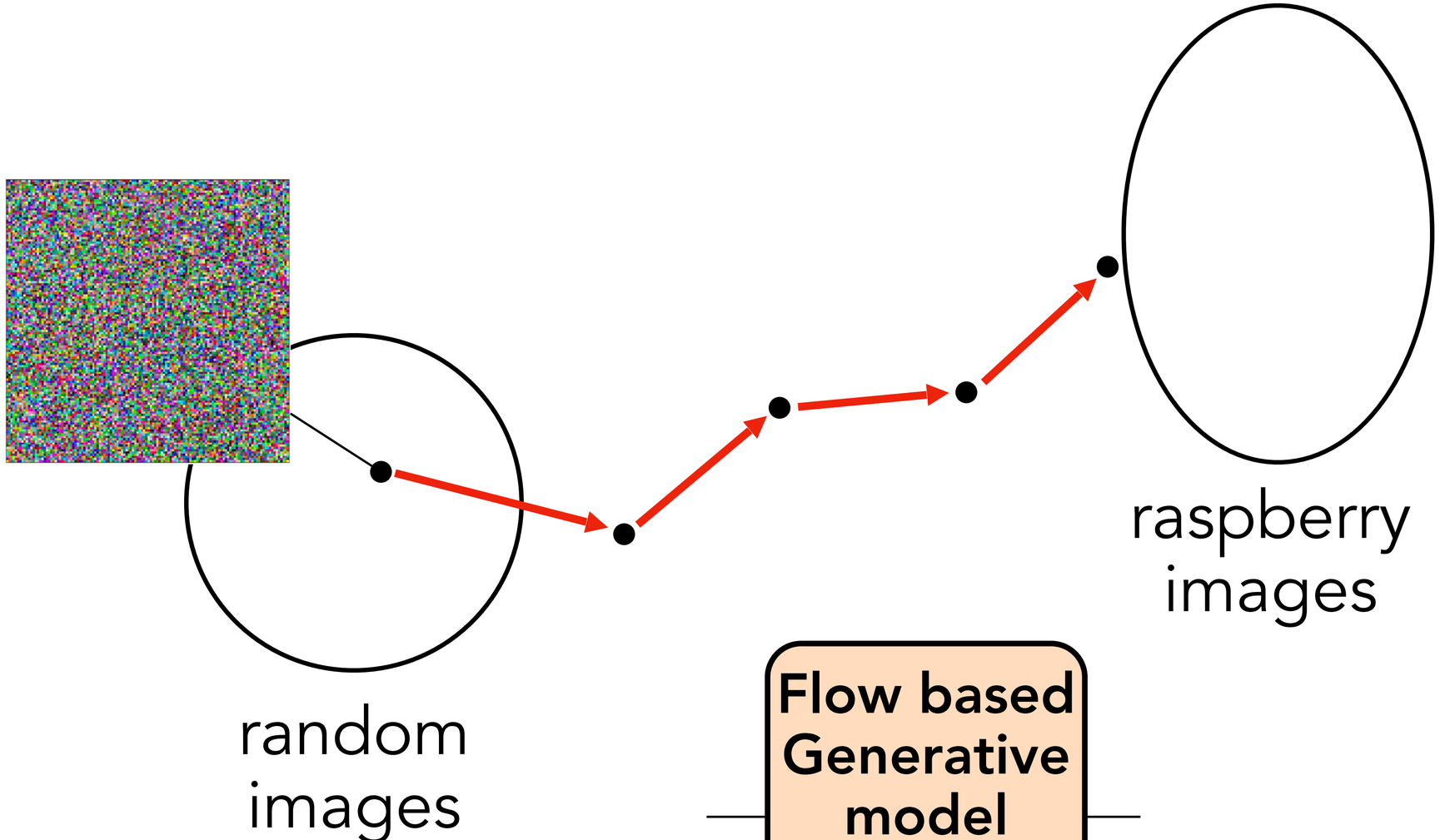
- Directly learn the velocity field [Lipman et al. 2022]
- Don’t have to worry bout the continuity equation because velocity fields can’t add / subtract mass. You only re-distribute
- Continuity equation satisfied by construction.
- Can learn velocity fields for each data sample (conditional velocity field), and everything works fine.

Source: Angjoo Kanazawa

# How do you train the flow?

Adapted from Angjoo Kanazawa

Just like diffusion:

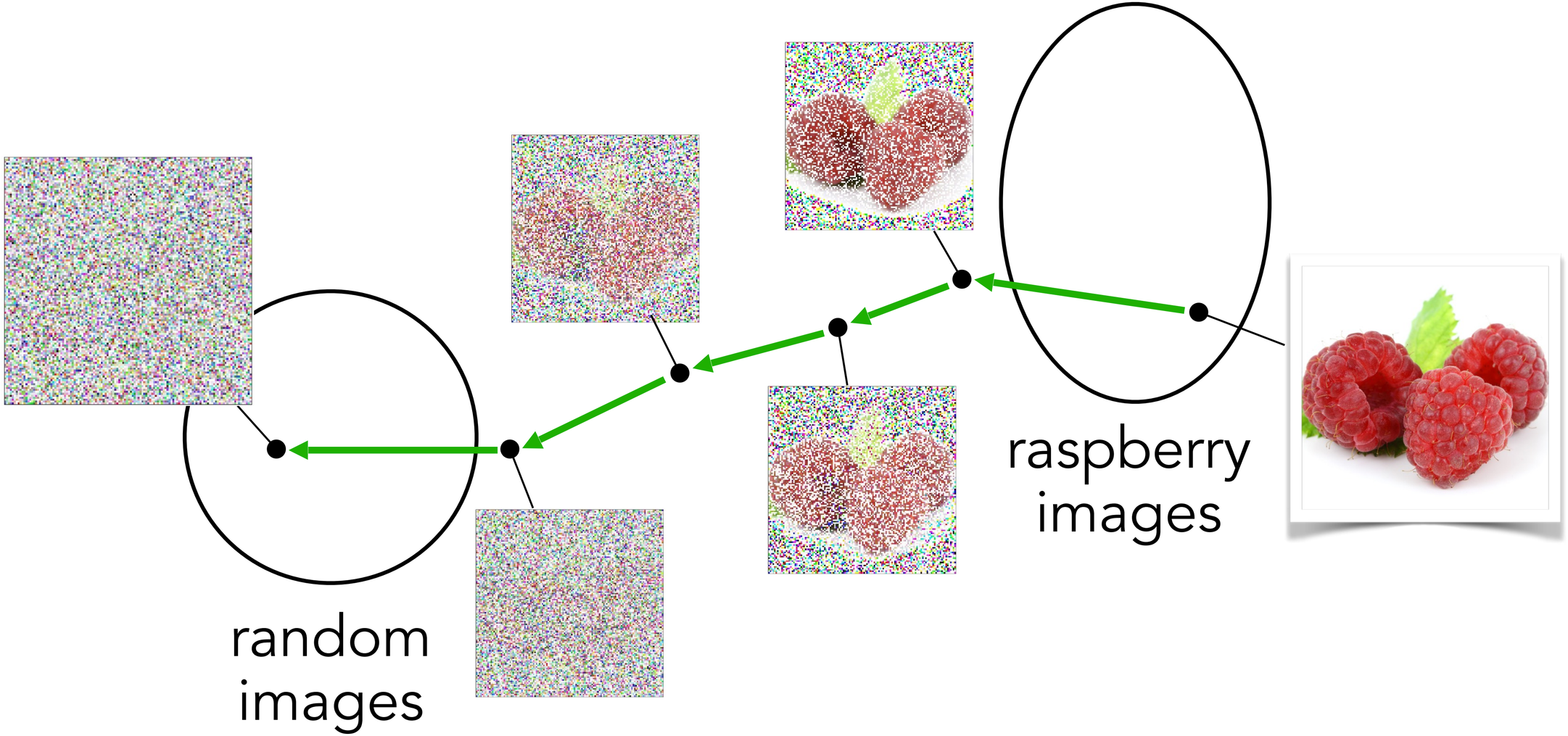



Source: Angjoo Kanazawa

slide from Steve Seitz's video

# Training

1. Take real data and corrupt it

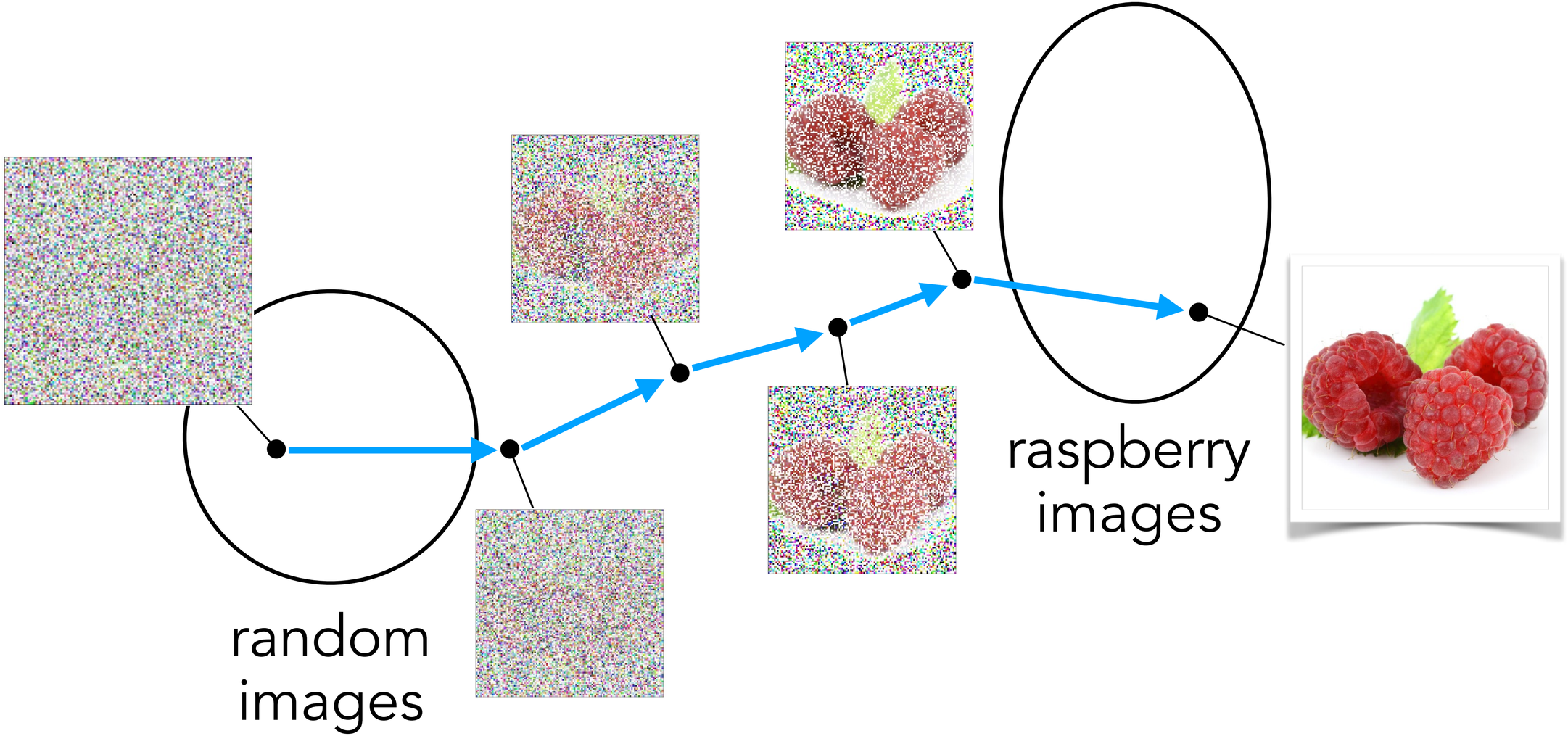



Source: Angjoo Kanazawa

Figure from Steve Seitz's video

# Training

1. Take real data and corrupt it
2. Learn to undo the process!




Source: Angjoo Kanazawa

Figure from Steve Seitz's video

# Denoising with a neural network

**Denoising neural network**

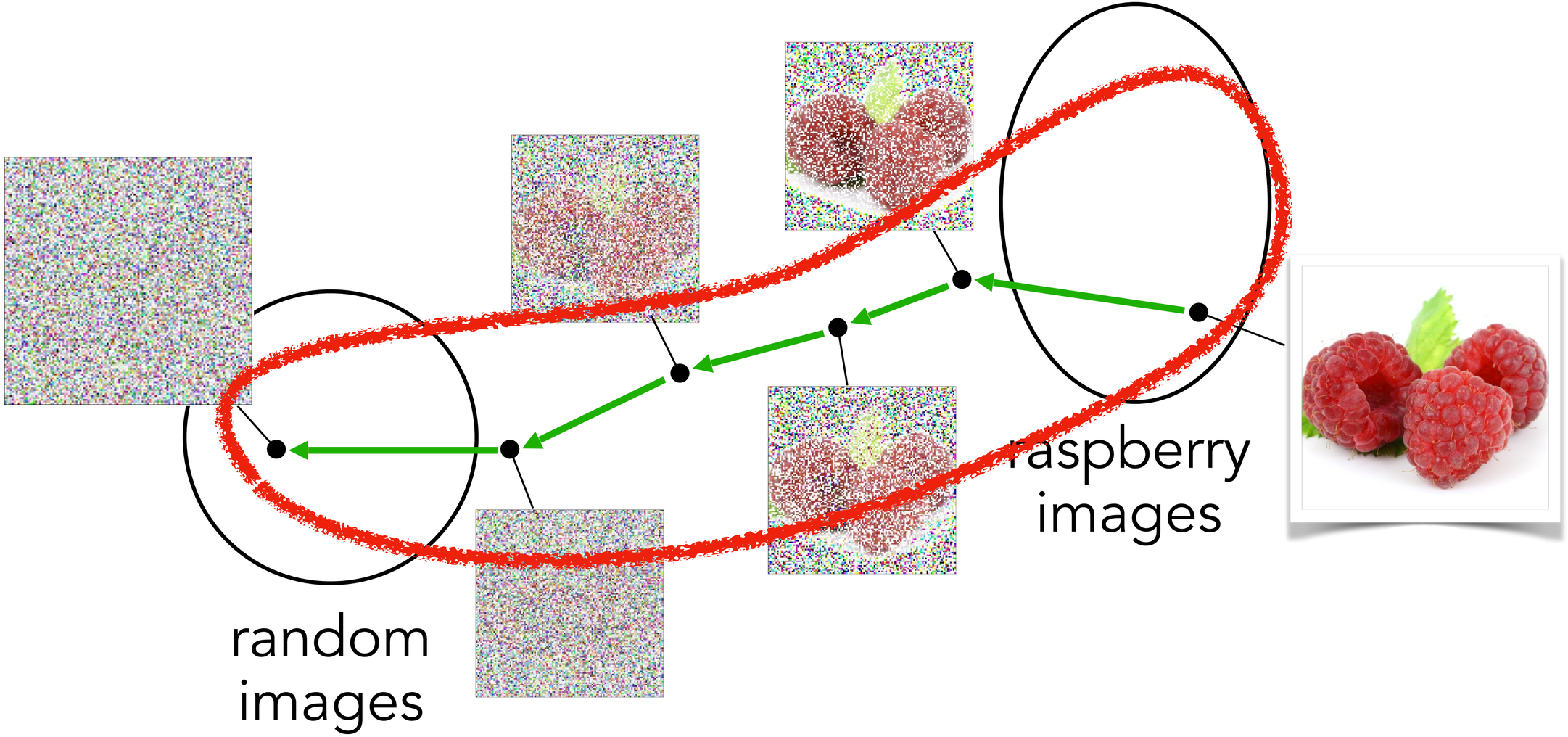
Just like in diffusion, can be a U-Net or transformer (DiT)

Adapted from Angjoo Kanazawa

**Slide source: Steve Seitz**

# How do we pick the intermediate path?

How to generate the path?




Adapted from Angjoo Kanazawa

Figure from Steve Seitz's video

# What is the path?

- How to add noise? What kind of noise? What schedule do we use?
- This is a bit complicated in diffusion. Noise process needed to have a specific form to make the math work.
- Can we make it simpler?
- Flow matching says that you can (essentially) add noise however you like!

Adapted from Angjoo Kanazawa

# How do we construct $x_t$?

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

- Flow matching is quite general! For now, just consider time-dependent weighted combinations.
- Flow matching says we can choose any weighted combination, as long we start from a sample in the source (e.g., Gaussian) and end with a sample in the target distribution (image).

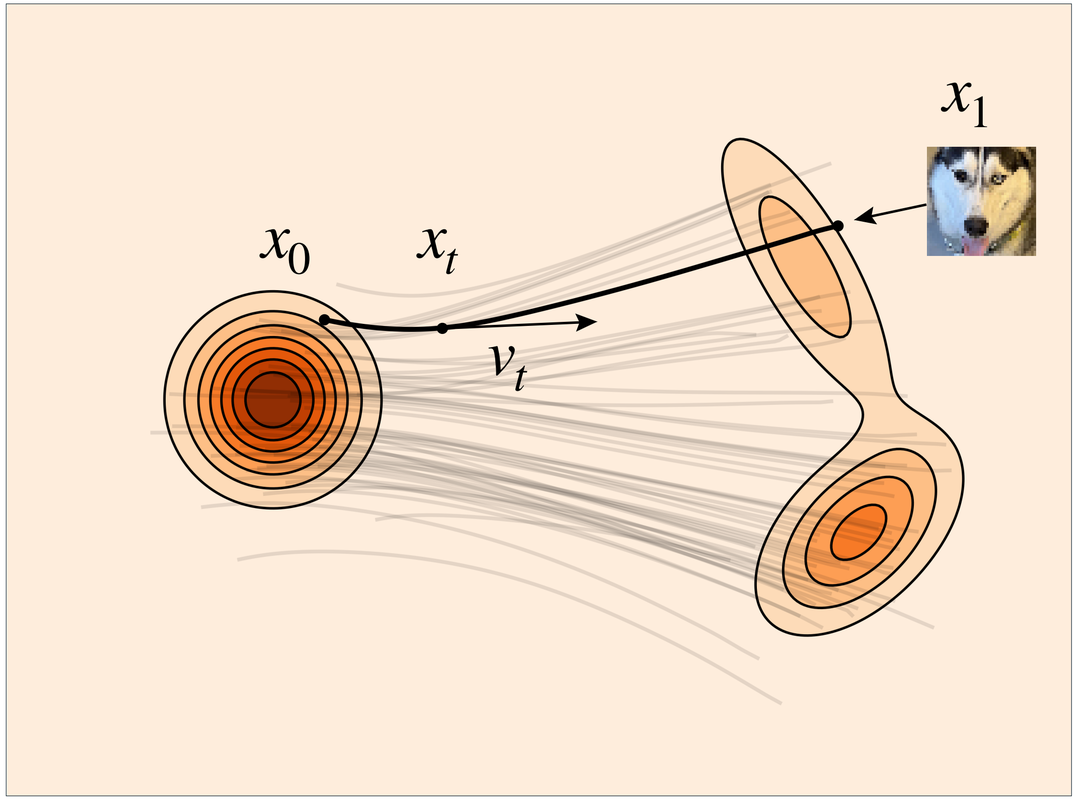
$$x_t = \alpha_t x_0 + \sigma_t x_1$$

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

Adapted from Angjoo Kanazawa

# Flow training

- For each image $x_1$
  - Sample some noise $x_0$
  - Combine them however you'd like to get $x_t$
  - Now learn to predict the velocity at $x_t$
    - What is the velocity? It depends on how you got $x_t$.



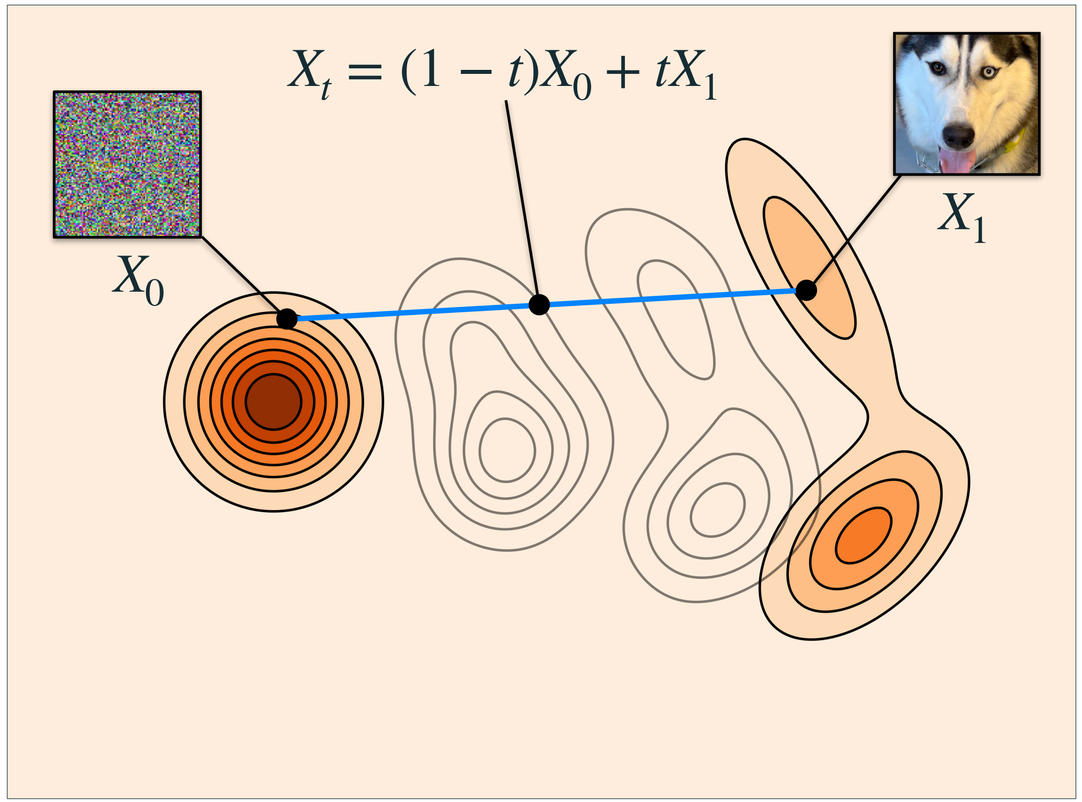

Adapted from Angjoo Kanazawa

# A Very Simple Way

## Linear interpolation!

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

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




Source: Angjoo Kanazawa

# What is the velocity supervision?

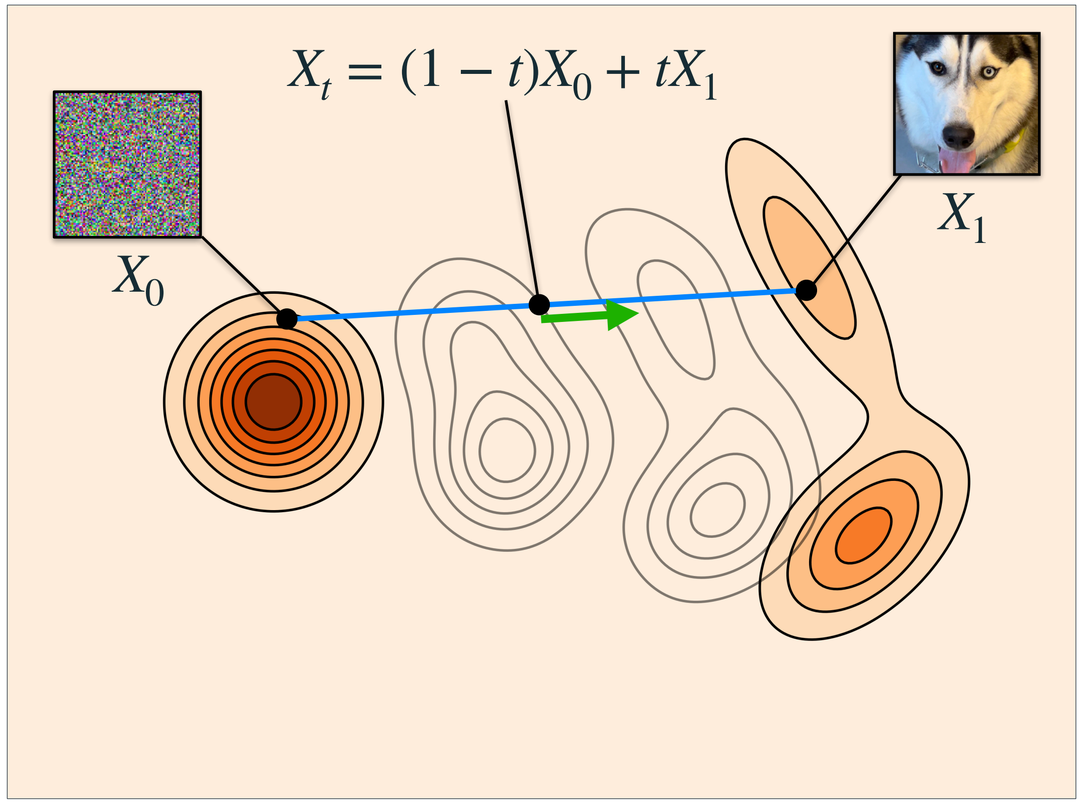
$$x_t = \alpha_t x_0 + \sigma_t x_1$$

$$x_t = (1 - t)x_0 + tx_1$$

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

$$= x_1 - x_0$$

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




*Conditioned on a single sample

Source: Angjoo Kanazawa

# Inside a Training Loop

## Flow Matching

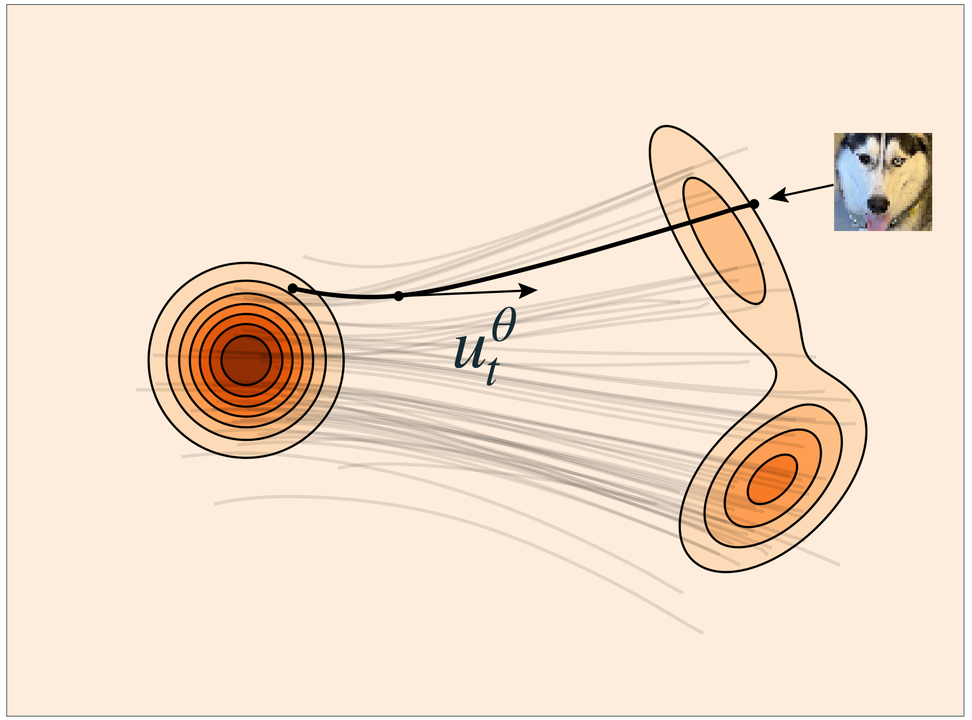
```
x = next(dataset)
t = torch.rand(1) # Sample timestep (0,1)
noise = torch.randn_like(x) # Sample noise
x_t = (1-t) * noise + (t) * x # Get noisy x_t

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()
```

Source: Angjoo Kanazawa

# Test-time sampling

- Just take a small step in the velocity
- Use any ODE Solver, i.e. integration you like, e.g., Euler integration:

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




**Sample**
from $X_0 \sim p$

Adapted from: Angjoo Kanazawa

# Inside Sampling Loop

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

Source: Angjoo Kanazawa

# Training: Model parameterization

- You can make your network output undo the noise in many different ways, predicting $x_1$, v, noise, or flow

$$v_t = \alpha_t x_1 - \sigma_t x_0 \qquad u_t = x_1 - x_0 = \epsilon - x_0$$
$$u_t = x_t - x_0$$

- 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:

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

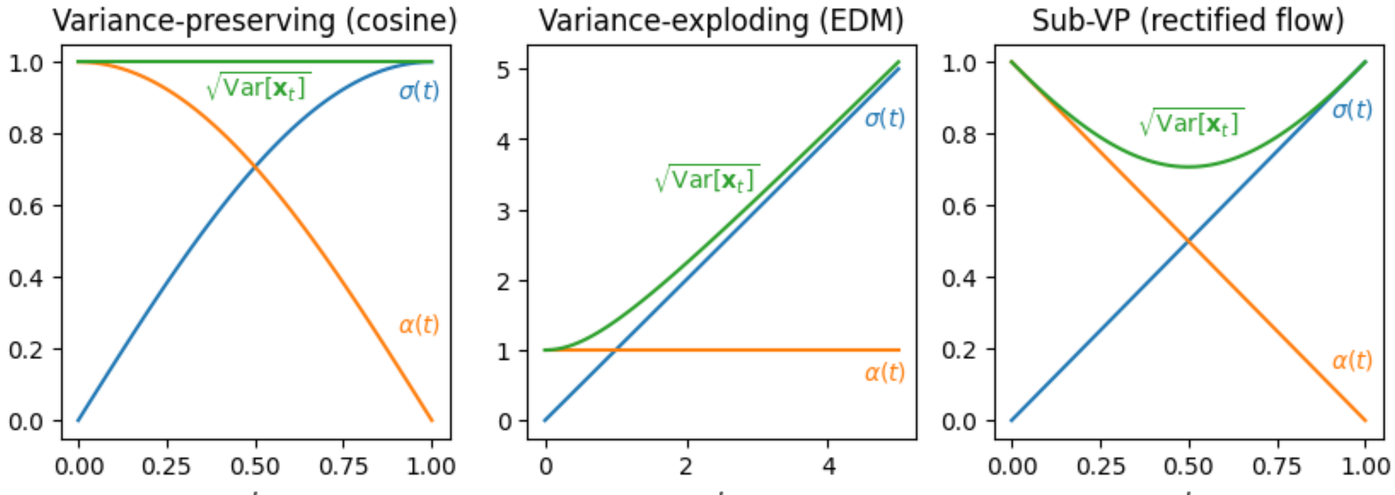
*Slides from Yaron Lipman*

Source: Angjoo Kanazawa

# Weighting schemes

Different choices for: $x_t = \alpha_t x_0 + \sigma_t x_1$

- "Variance preserving": DDPM (the main diffusion formulation you've seen already):
  - i.e., $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$
- Linear interpolation (Flow Matching variant that we've discussed so far, which is also known as Rectified Flow)




Adapted from: Angjoo Kanazawa

# Training: Flow Matching vs. Diffusion

**Algorithm 1:** Flow Matching training.

**Input** : dataset $q$, noise $p$
Initialize $v^\theta$
**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:** $v^\theta$

**Algorithm 2:** Diffusion training.

**Input** : dataset $q$, noise $p$
Initialize $s^\theta$
**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:** $v^\theta$

$p_t(x_t \mid x_1)$ general
$p(x_0)$ is general

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

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

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

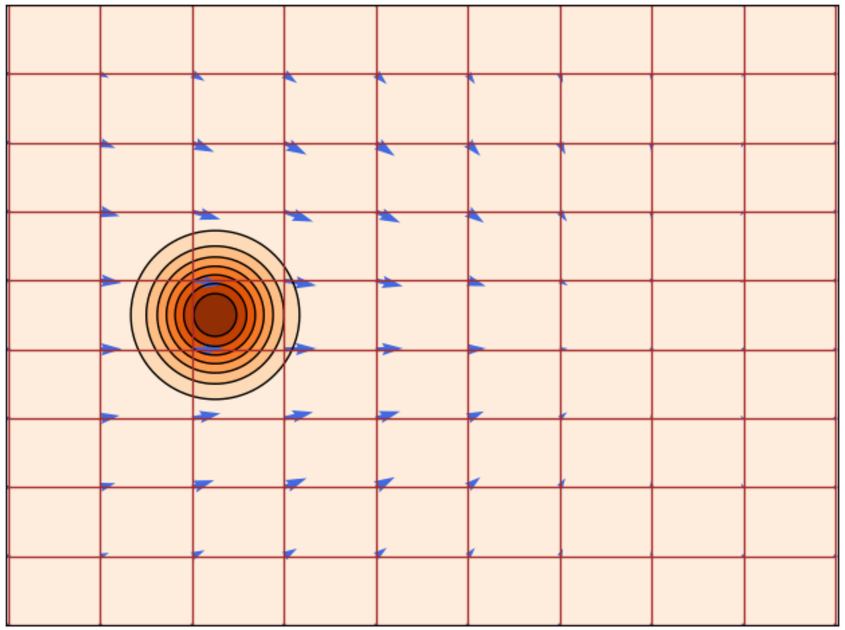
$p(x_0)$ is Gaussian
$p_0(\cdot \mid x_1) \approx p$

Source: Angjoo Kanazawa

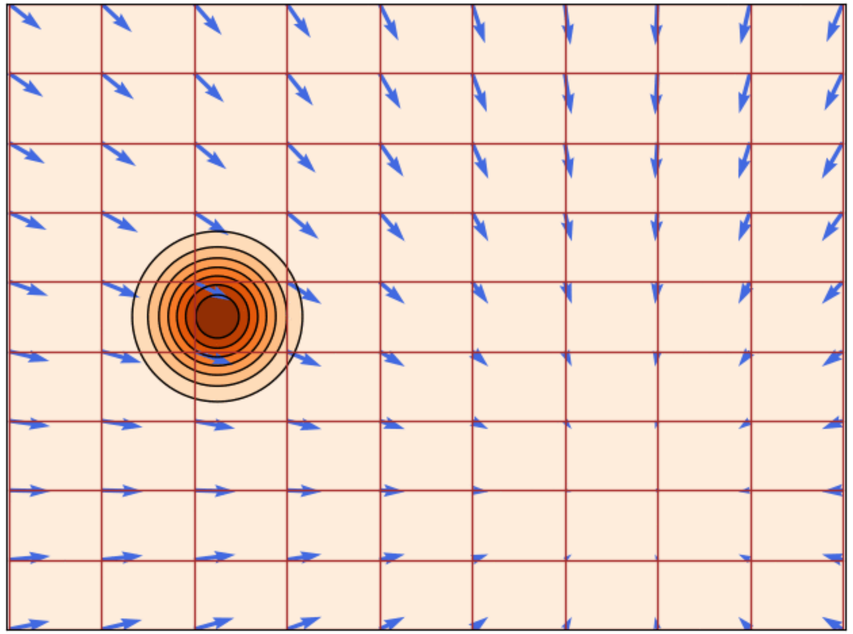
# Why does this work?

- 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)

Source: Angjoo Kanazawa

# It turns out that the gradient is the same!

- **Flow Matching loss:**

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

where $u_t(X_t)$ is the target (marginal) velocity field

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

- **Conditional Flow Matching loss**:

$$\mathscr{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t,X_1,X_t} \left\| u_t^\theta(X_t) - u_t(X_t \mid X_1) \right\|^2 \qquad u_t(X_t \mid X_1) = X_1 - X_0$$

**Theorem:** Gradient of the losses are equivalent:

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

Source: Angjoo Kanazawa

# 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:

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

Source: Angjoo Kanazawa

# It’s all just weighted average

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

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

Weight of sample x0 = 1

**Marginal flow**

**Path from $x_1$ to $x_t$**

**Path weight (how relevant $x_1$ is to $x_t$)**

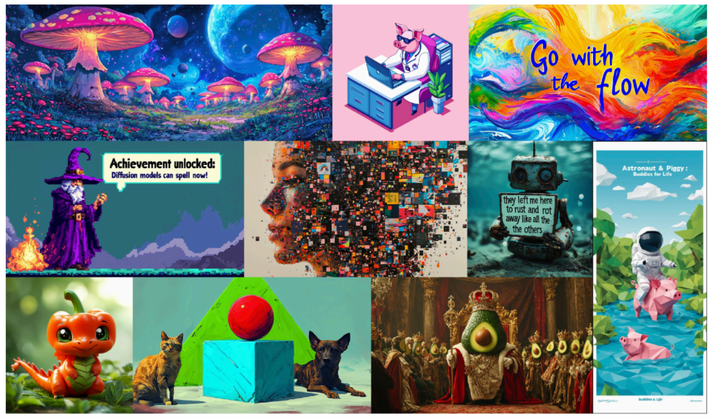
Just a weighted average of the flow to each data sample!

Source: Angjoo Kanazawa

# Widely used in state-of-the-art generators

**Scaling Rectified Flow Transformers for High-Resolution Image Synthesis**

Patrick Esser * Sumith Kulal Andreas Blattmann Rahim Entezari Jonas Müller Harry Saini Yam Levi
Dominik Lorenz Axel Sauer Frederic Boesel Dustin Podell Tim Dockhorn Zion English
Kyle Lacey Alex Goodwin Yannik Marek Robin Rombach *
Stability AI




**FLUX.1 Kontext: Flow Matching for In-Context Image Generation and Editing in Latent Space**

Black Forest Labs*
Stephen Batifol Andreas Blattmann Frederic Boesel Saksham Consul Cyril Diagne
Tim Dockhorn Jack English Zion English Patrick Esser Sumith Kulal
Kyle Lacey Yam Levi Cheng Li Dominik Lorenz Jonas Müller
Dustin Podell Robin Rombach Harry Saini Axel Sauer Luke Smith

**Abstract**

We present evaluation results for *FLUX.1 Kontext*, a generative flow matching model that unifies image generation and editing. The model generates novel output views by incorporating semantic context from text and image inputs. Using a simple

**WAN: OPEN AND ADVANCED LARGE-SCALE VIDEO GENERATIVE MODELS**

**Wan Team, Alibaba Group**

ABSTRACT

This report presents Wan, a comprehensive and open suite of video foundation models designed to push the boundaries of video generation. Built upon the mainstream diffusion transformer paradigm, Wan achieves significant advancements in generative capabilities through a series of innovations, including our novel spatio-temporal variational autoencoder (VAE), scalable pre-training strategies, large-scale data curation, and automated evaluation metrics. These contributions collectively enhance the model's performance and versatility. Specifically, Wan is characterized by four key features: *Leading Performance*: The 14B model of Wan, trained on a vast dataset comprising billions of images and videos, demonstrates the scaling laws of video generation with respect to both data and model size. It consistently outperforms the existing open-source models as well as state-of-

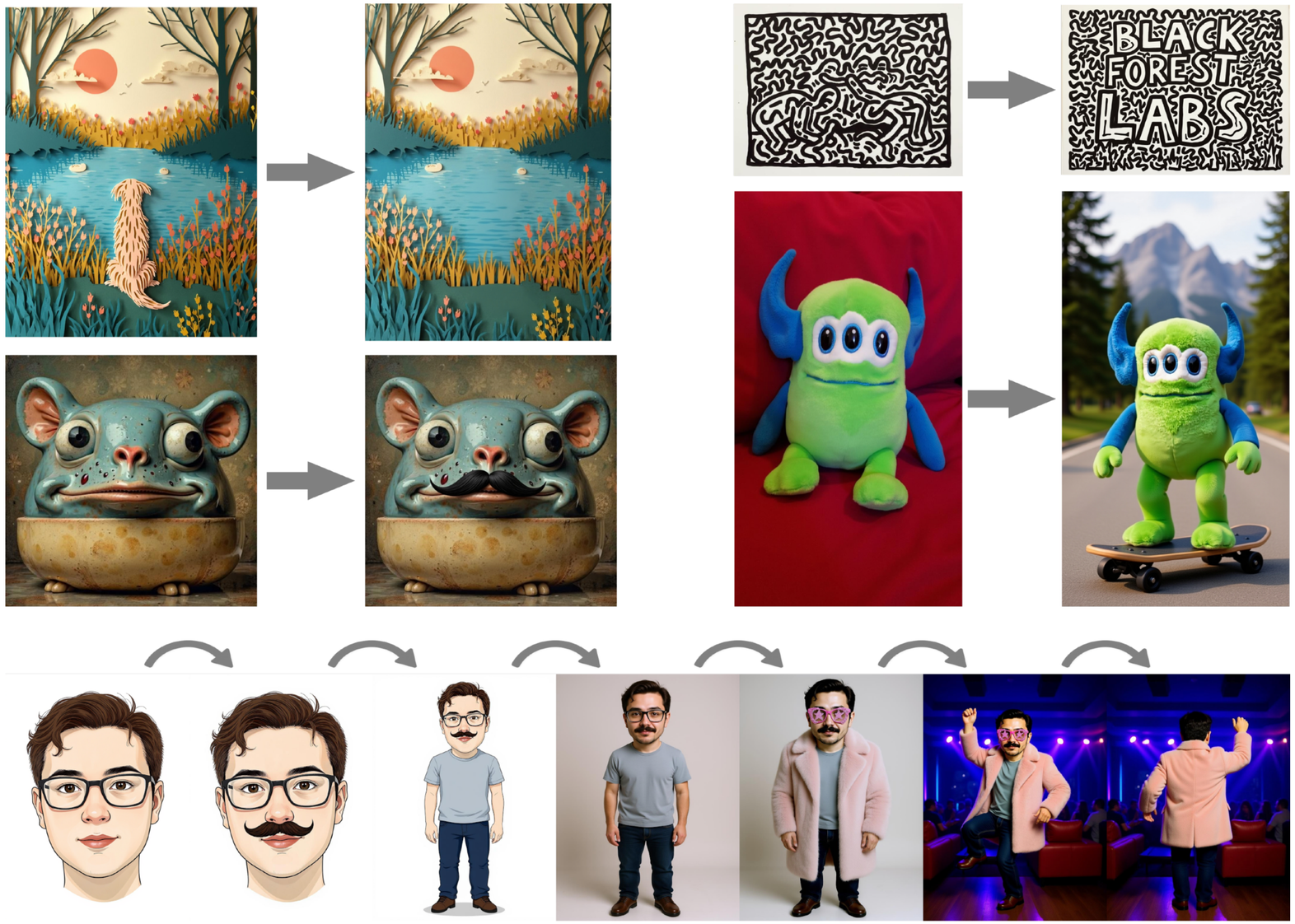
**FRIEREN: Efficient Video-to-Audio Generation Network with Rectified Flow Matching**

**Yongqi Wang‡, Wenxiang Guo‡, Rongjie Huang, Jiawei Huang, Zehan Wang, Fuming You, Ruiqi Li, Zhou Zhao***
Zhejiang University
cyanbox@zju.edu.cn

**Abstract**

Video-to-audio (V2A) generation aims to synthesize content-matching audio from silent video, and it remains challenging to build V2A models with high generation quality, efficiency, and visual-audio temporal synchrony. We propose FRIEREN, a V2A model based on rectified flow matching. FRIEREN regresses the conditional transport vector field from noise to spectrogram latent with straight paths and

# Flow matching in state-of-art generators




Source: FLUX.1-Kontext

# 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
- The flow matching perspective is very general: just directly learn a velocity field from one distribution to another.

Source: Angjoo Kanazawa

**Next class:** image manipulation with diffusion models