

Mathematics of Diffusion Models and their application for Image Inverse Problems

Samuel Hurault

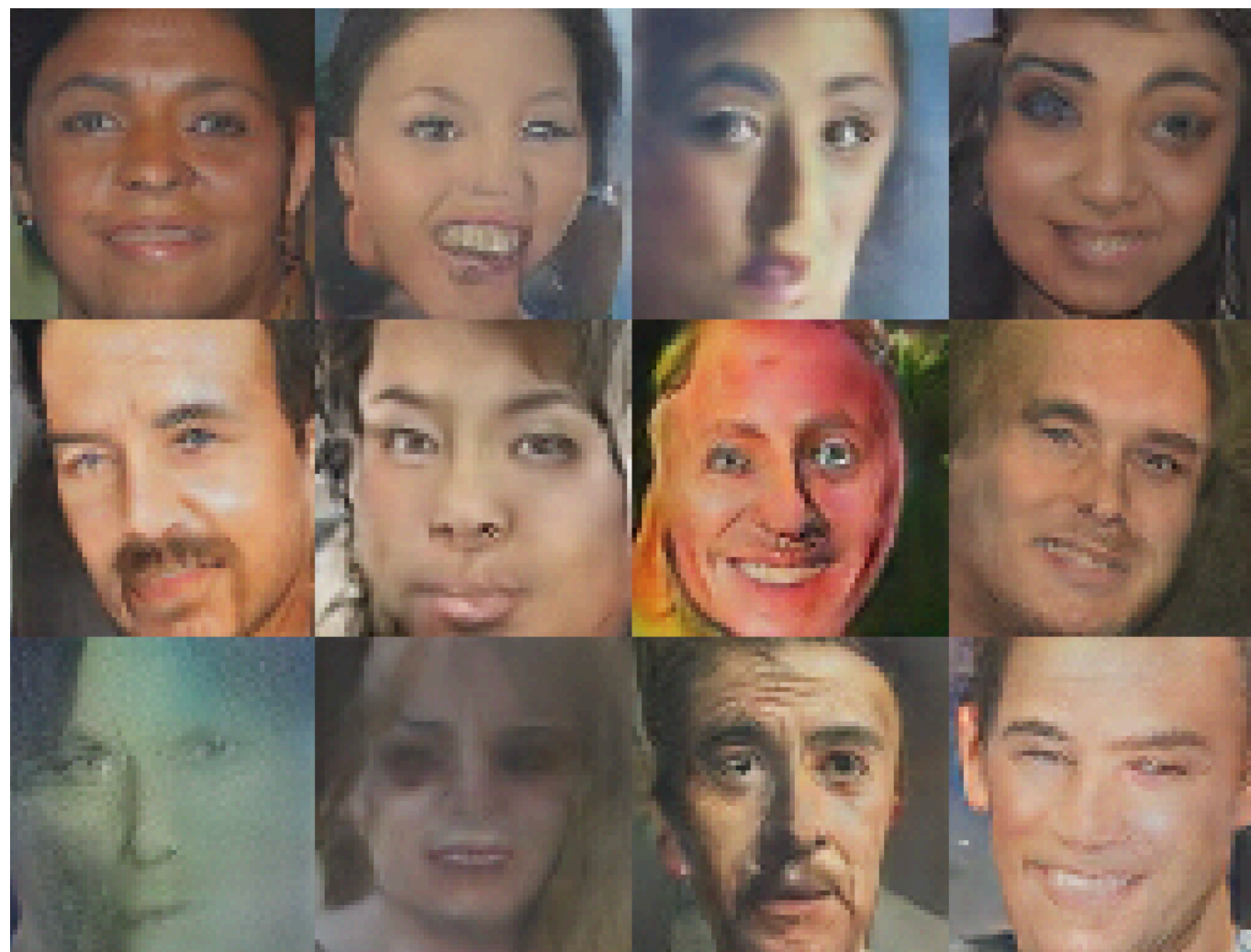
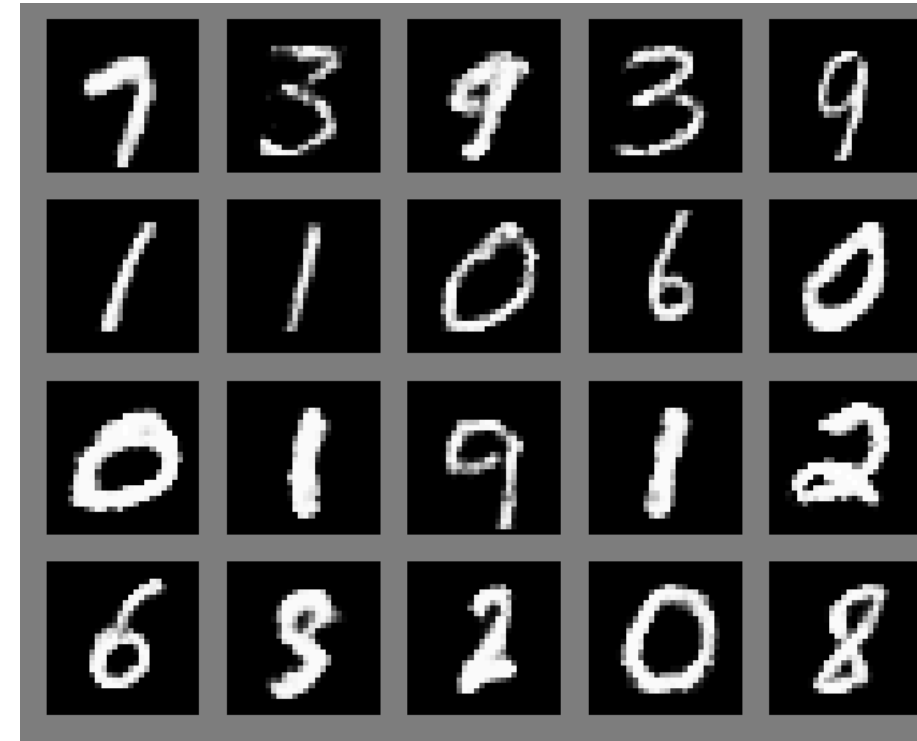
Univ. Gustave Eiffel, CNRS



Introduction to Diffusion Models and Flow Matching

Evolution of image generative models

Generative Adversarial Networks
Goodfellow et al, NIPS 2014



DCGAN
Radford et al, ICLR 2016



Normalizing Flows
Kingma & Dhariwal, NIPS 2018



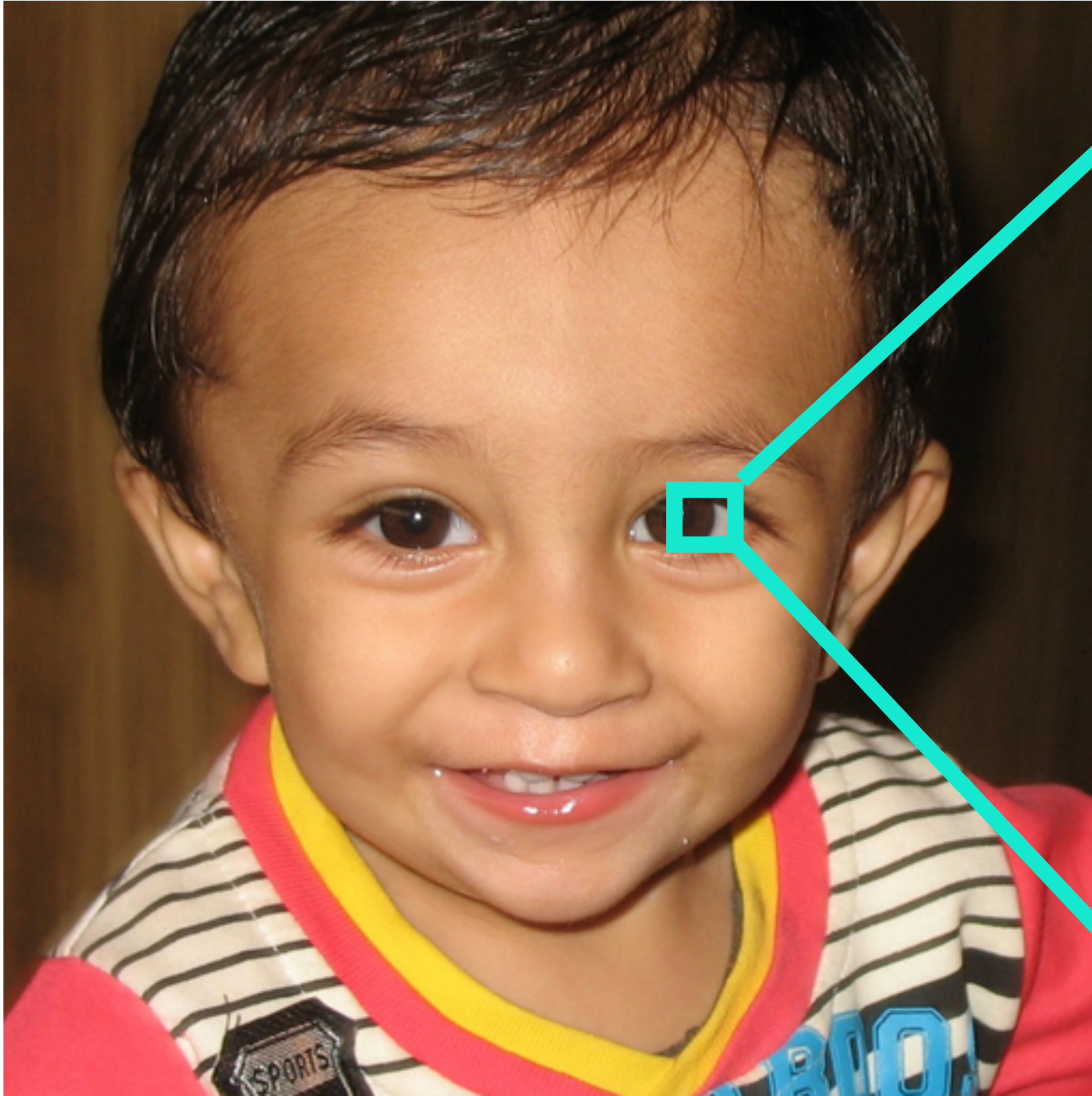
Diffusion Models (DDPM)
Ho et al, NeurIPS 2020

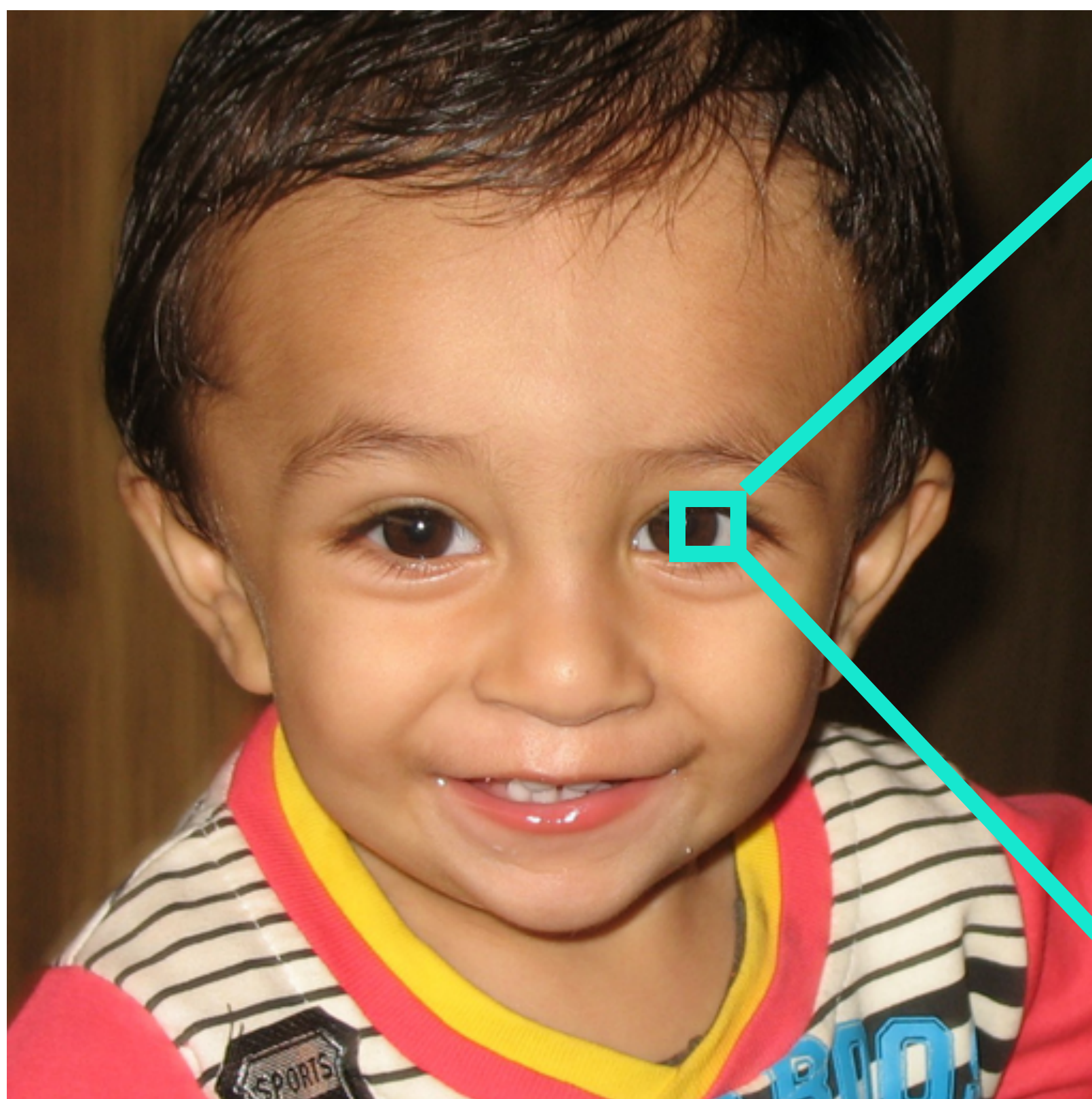
Evolution of image generative models

GPT Image 2

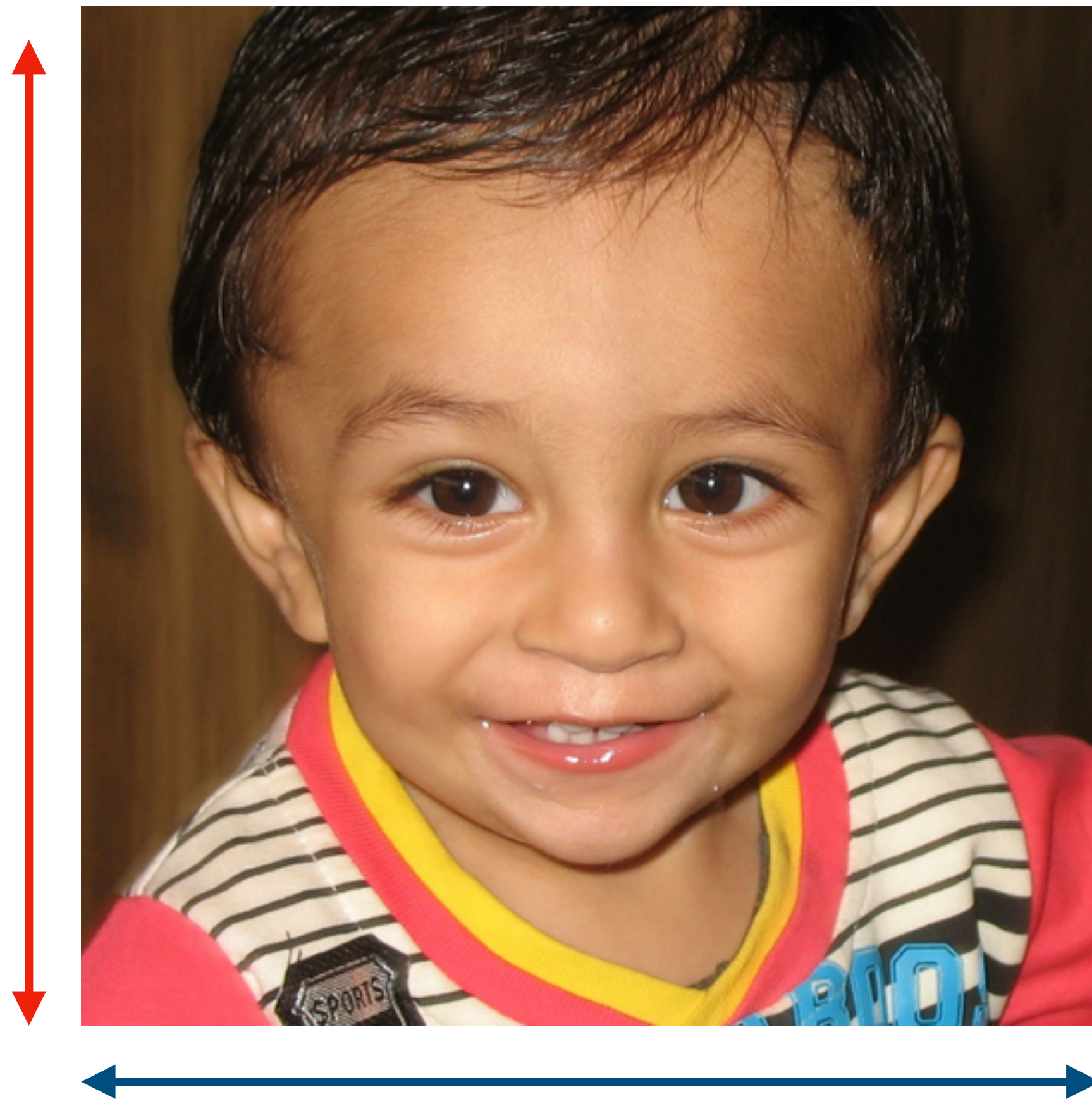
« Image of a researcher in Machine Learning talking about generative models to an audience in a conference »



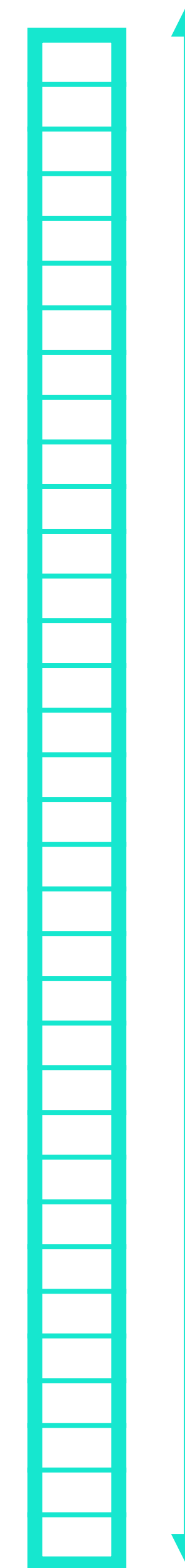


[illegible]

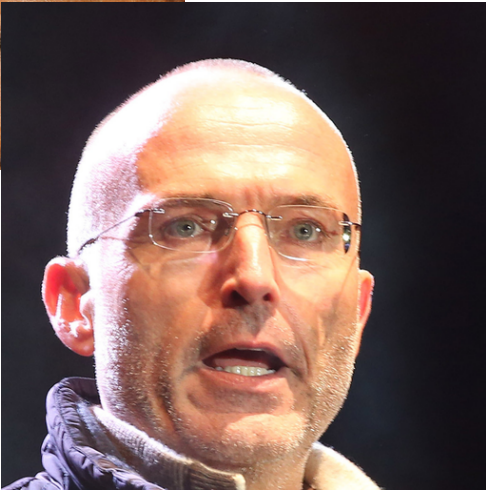
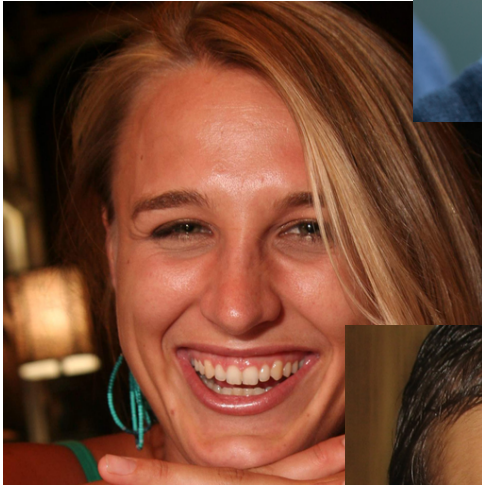
An image is a vector in $\mathbb{R}^d$ with
 $d = \text{nb pixel width} \times \text{nb pixel height} \times \text{nb channels}$



Stacking



$\in \mathbb{R}^d$



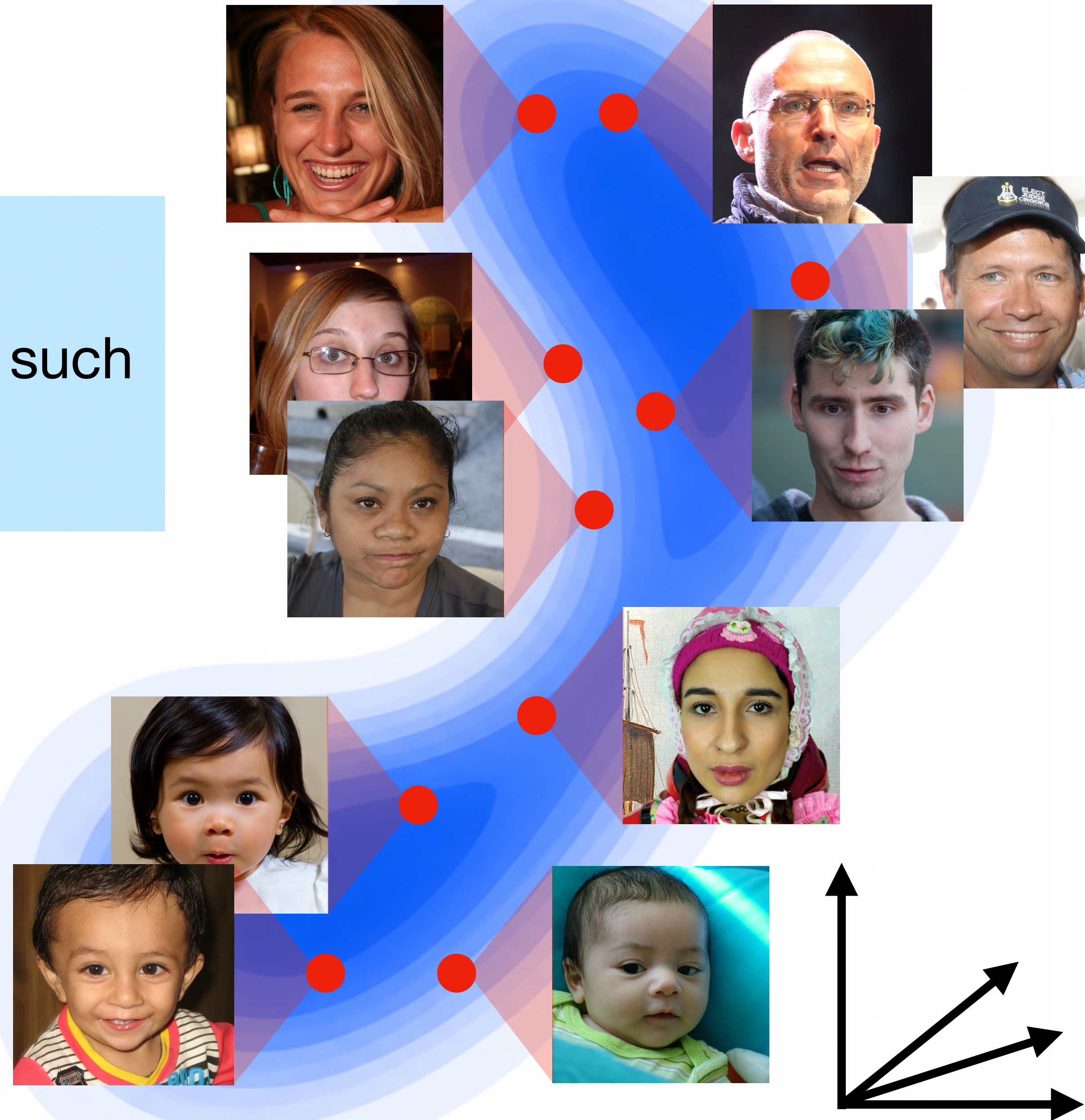
Each image is a point in $\mathbb{R}^d$

Assumption:

There exists a probability distribution p_{data} such that each image $x_i \sim p_{\text{data}}$

Objective:

Approximate p_{data} with a parametric distribution p_{θ} « easy to sample »

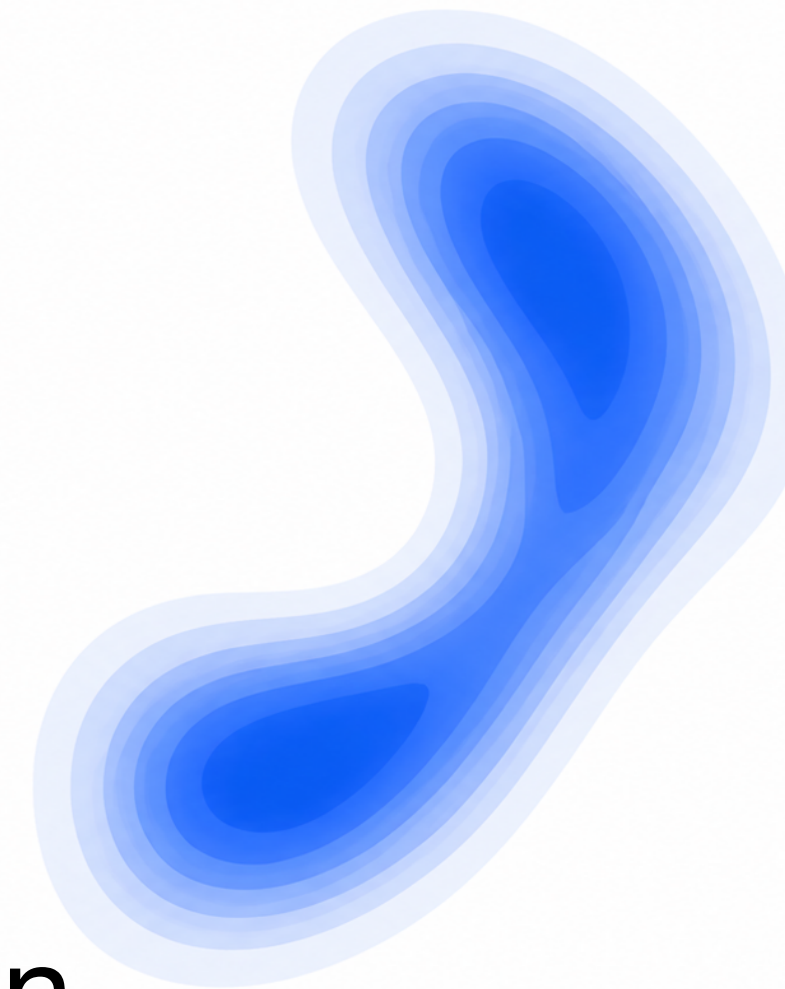


Generative models

From a dataset, generate novel data



Learned
representation



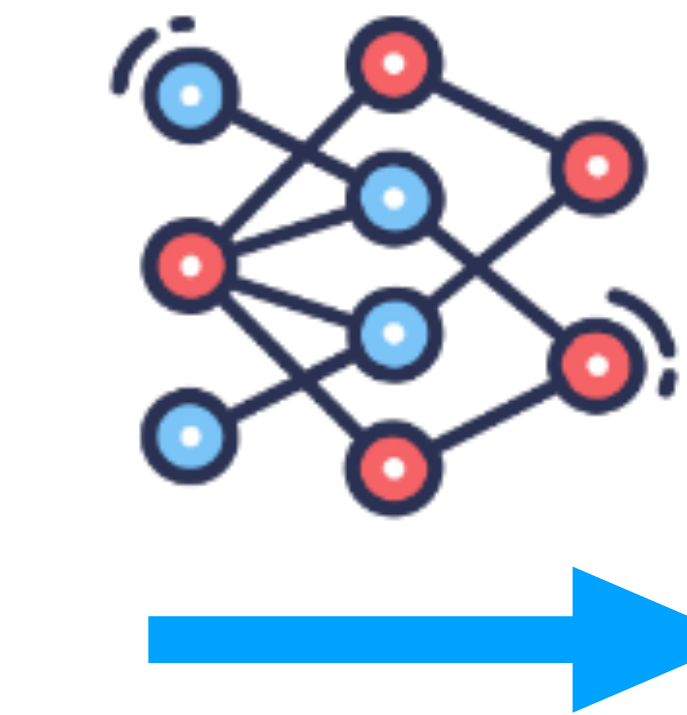
$$p_{\theta}(x)$$

Generation

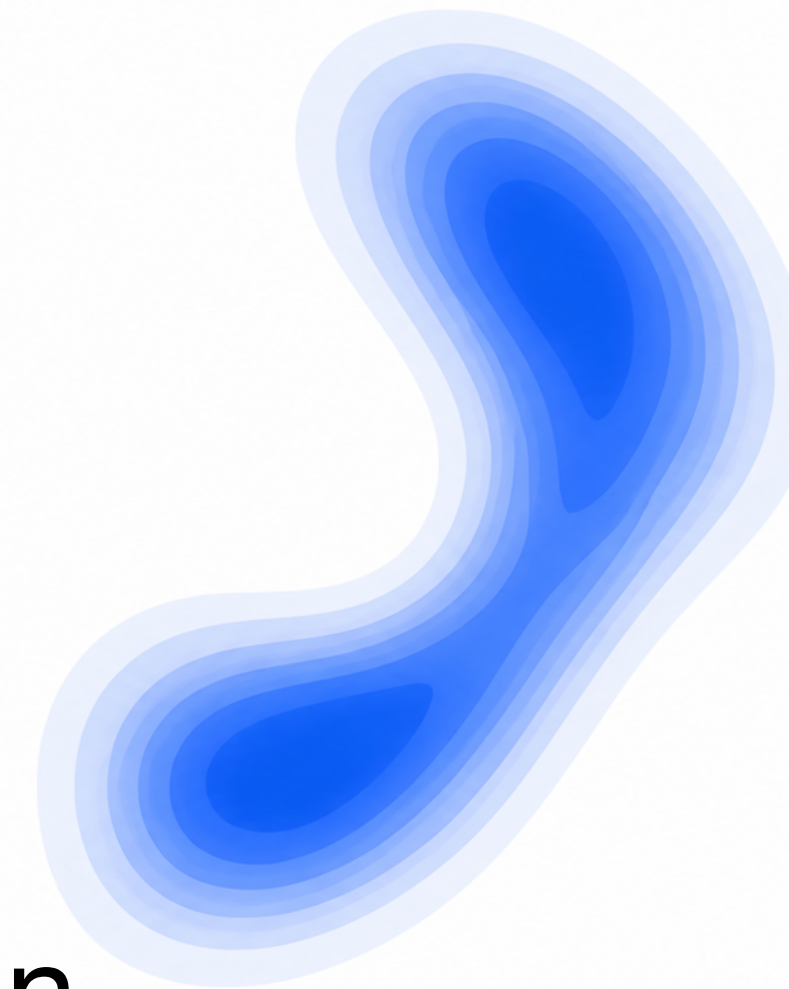


Generative models

From a dataset, generate novel data



Learned
representation



$$p_{\theta}(x)$$

Generation

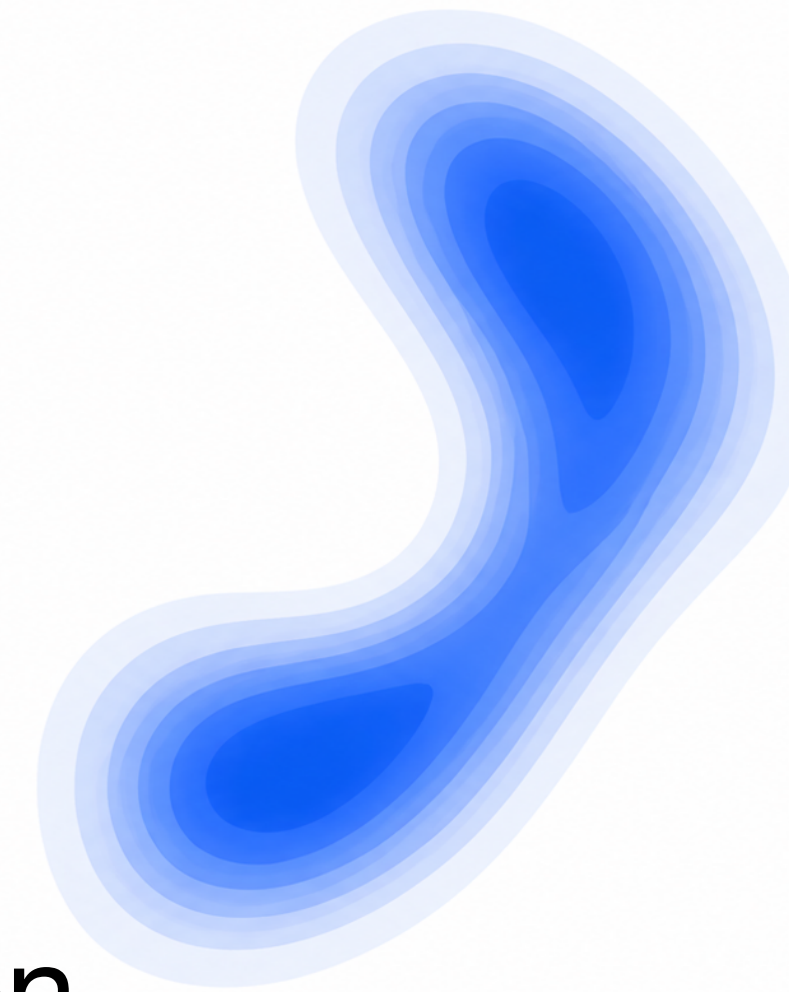
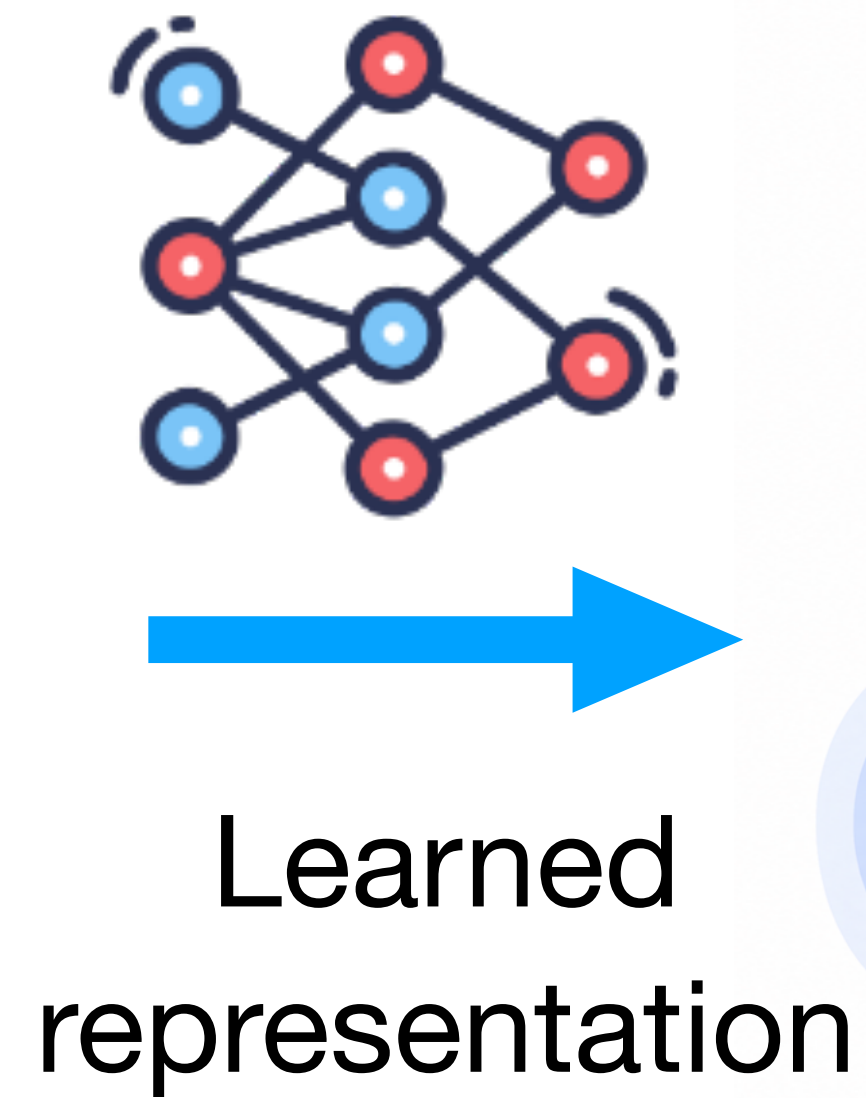
Condition

« A blond child with
curly hair »

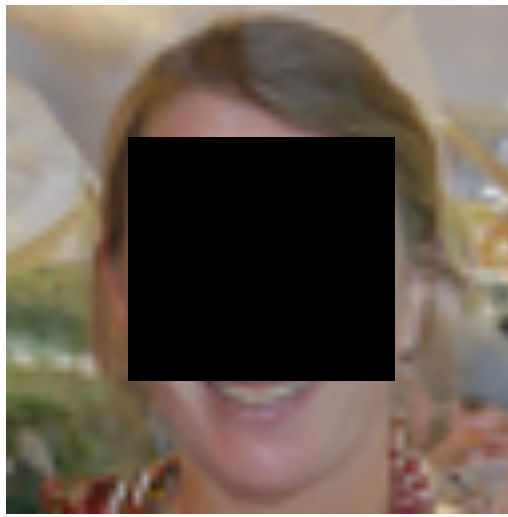


Generative models

From a dataset, generate novel data



Measurement



Generation



Generative models

given samples of p_{data} , generate new samples of p_{data}

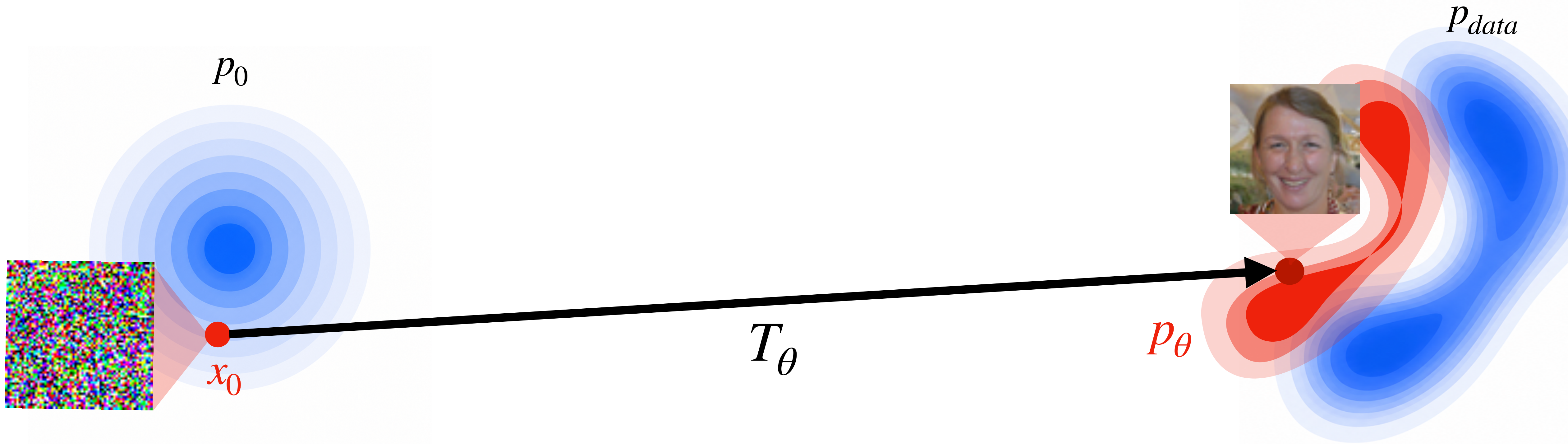
Flow-based methods:

- Choose p_0 easy to sample
- Find a map $T_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^d$ such that
 $x_0 \sim p_0, x = T_\theta(x_0)$ samples p_{data}

$$p_\theta := T_\theta \# p_0 \approx p_{data}$$

Train a neural network T_θ to minimize

$$\min_{\theta} D(p_{data}, p_\theta)$$



Generative models

given samples of p_{data} , generate new samples of p_{data}

Flow-based methods:

- Choose p_0 easy to sample
- Find a map $T_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^d$ such that
 $x_0 \sim p_0, x = T_\theta(x_0)$ samples p_{data}

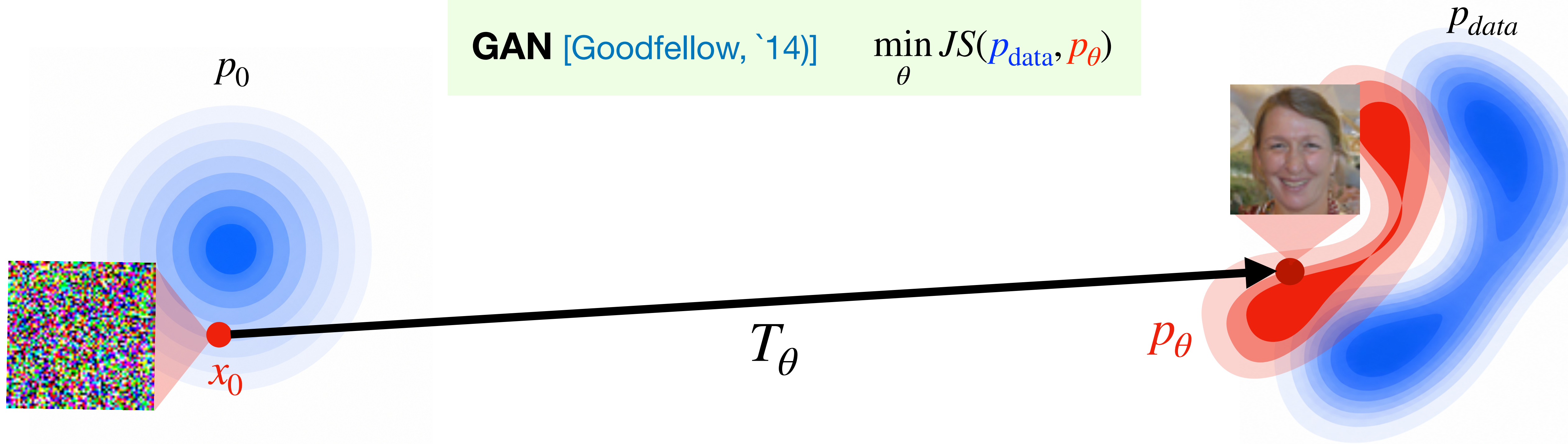
$$p_\theta := T_\theta \# p_0 \approx p_{data}$$

Train a neural network T_θ to minimize

$$\min_{\theta} D(p_{data}, p_\theta)$$

GAN [Goodfellow, '14]

$$\min_{\theta} JS(p_{data}, p_\theta)$$



Generative models

given samples of p_{data} , generate new samples of p_{data}

Flow-based methods:

- Choose p_0 easy to sample
- Find a map $T_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^d$ such that
 $x_0 \sim p_0, x = T_\theta(x_0)$ samples p_{data}

$$p_\theta := T_\theta \# p_0 \approx p_{data}$$

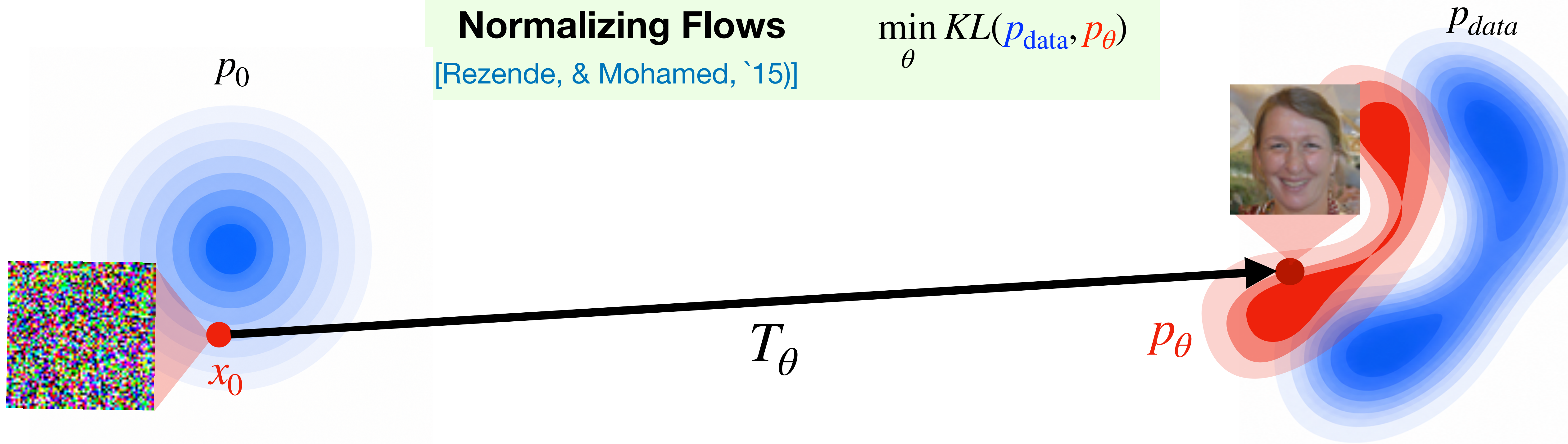
Train a neural network T_θ to minimize

$$\min_{\theta} D(p_{data}, p_\theta)$$

Normalizing Flows

[Rezende, & Mohamed, '15)]

$$\min_{\theta} KL(p_{data}, p_\theta)$$



Generative models

given samples of p_{data} , generate new samples of p_{data}

Flow-based methods:

- Choose p_0 easy to sample
- Find a map $T_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^d$ such that
 $x_0 \sim p_0, x = T_\theta(x_0)$ samples p_{data}

$$p_\theta := T_\theta \# p_0 \approx p_{data}$$

Train a neural network T_θ to minimize

$$\min_{\theta} D(p_{data}, p_\theta)$$

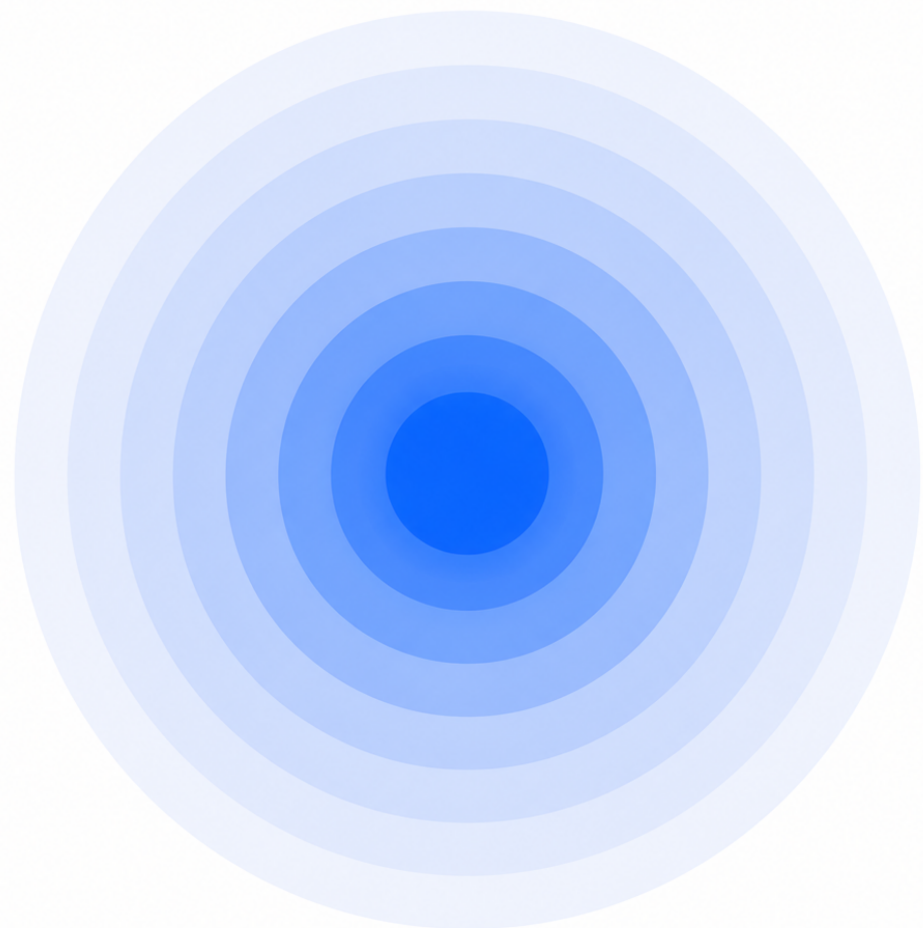
Normalizing Flows

[Rezende, & Mohamed, '15)]

$$\min_{\theta} KL(p_{data}, p_\theta)$$

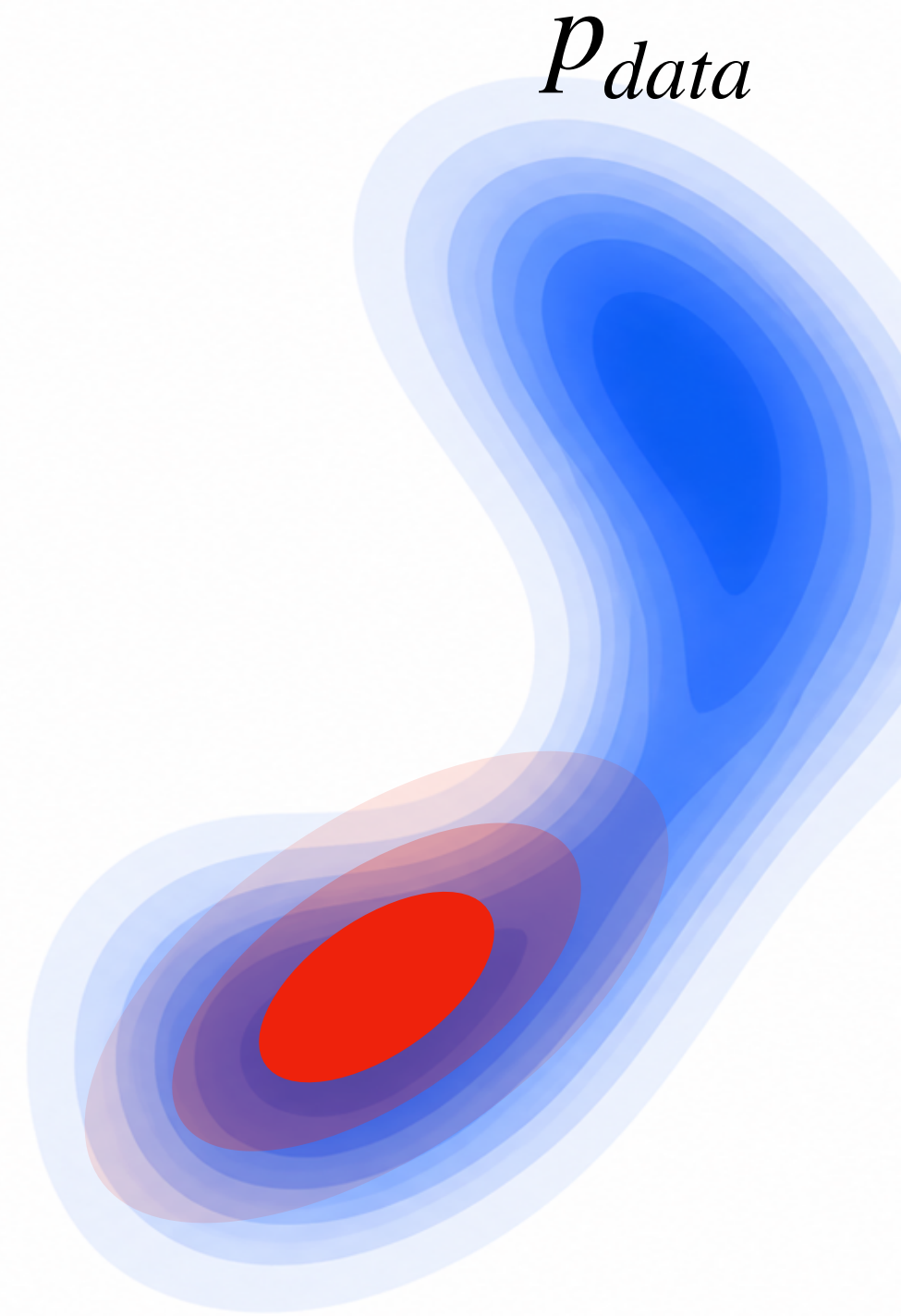
✗ Hard to train a single-step model:
unstable training, mode collapse...

p_0



p_{data}

p_θ



Generative models

given samples of p_{data} , generate new samples of p_{data}

Flow-based methods:

- Choose p_0 easy to sample
- Find a map $T_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^d$ such that
 $x_0 \sim p_0, x = T_\theta(x_0)$ samples p_{data}

$$p_\theta := T_\theta \# p_0 \approx p_{data}$$

Train a time-dependent NN v_θ to minimize

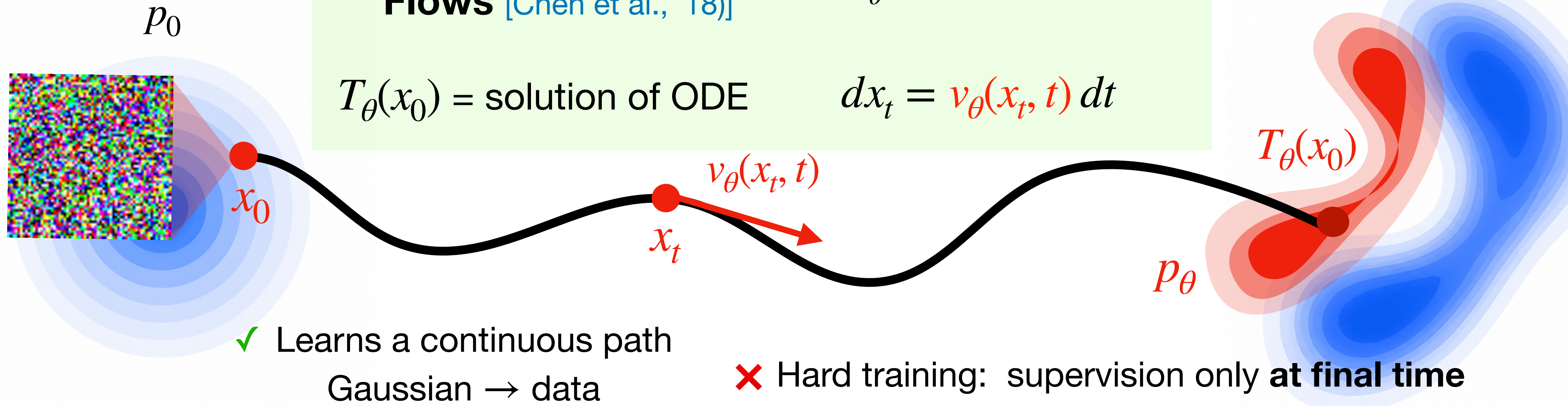
$$\min_{\theta} D(p_{data}, p_\theta)$$

**Continuous Normalizing
Flows** [Chen et al., '18]

$$\min_{\theta} KL(p_{data}, p_\theta)$$

$T_\theta(x_0)$ = solution of ODE

$$dx_t = v_\theta(x_t, t) dt$$



Generative models

given samples of p_{data} , generate new samples of p_{data}

Flow-based methods:

→ Choose p_0 easy to sample

→ Find a map $T_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^d$ such that

$x_0 \sim p_0$, $x = T_\theta(x_0)$ samples p_{data}

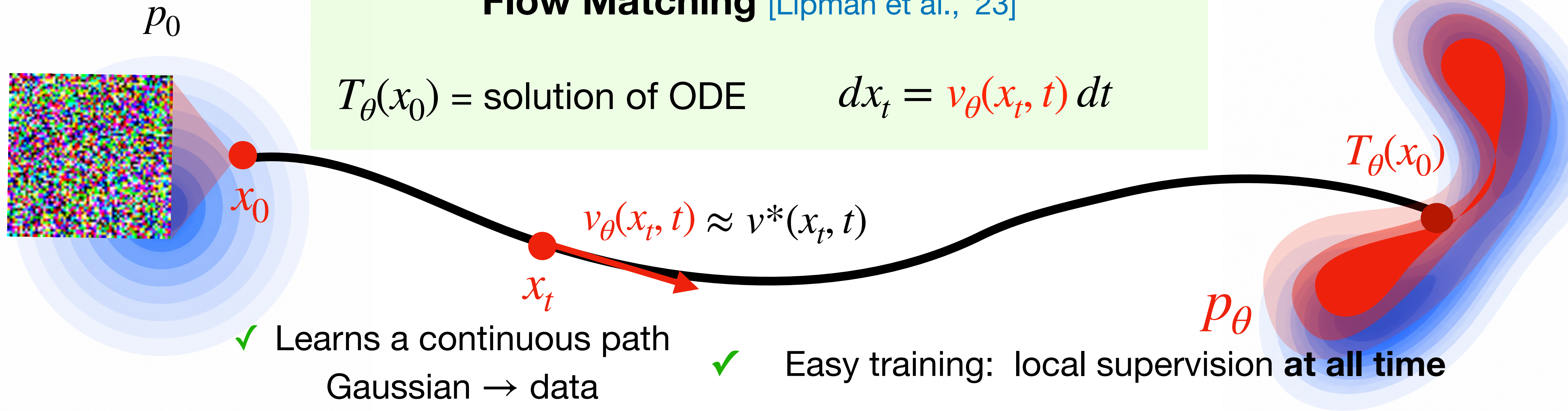
$$p_\theta := T_\theta \# p_0 \approx p_{data}$$

Train a time-dependent NN v_θ to minimize

$$\min_{\theta} \int_t \|v_\theta(x, t) - v^*(x, t)\|$$

Diffusion Models [Song et al., '21] /
Flow Matching [Lipman et al., '23]

$$T_\theta(x_0) = \text{solution of ODE} \quad dx_t = v_\theta(x_t, t) dt$$



✓ Learns a continuous path
Gaussian $\rightarrow$ data

✓ Easy training: local supervision **at all time**

Generative models

given samples of p_{data} , generate new samples of p_{data}

Flow-based methods:

→ Choose p_0 easy to sample

→ Find a map $T_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^d$ such that

$x_0 \sim p_0, x = T_\theta(x_0)$ samples p_{data}

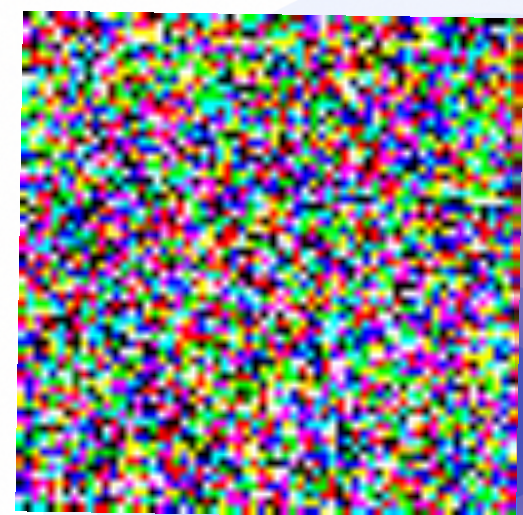
$$p_\theta := T_\theta \# p_0 \approx p_{data}$$

Train a time-dependent NN v_θ to minimize

$$\min_{\theta} \int_t \|v_\theta(x, t) - v^*(x, t)\|$$

Diffusion Models [Song et al., '21] /
Flow Matching [Lipman et al., '23]

$$T_\theta(x_0) = \text{solution of SDE} \quad dx_t = v_\theta(x_t, t) dt + \sqrt{2a_t} dW_t$$



p_0

x_0

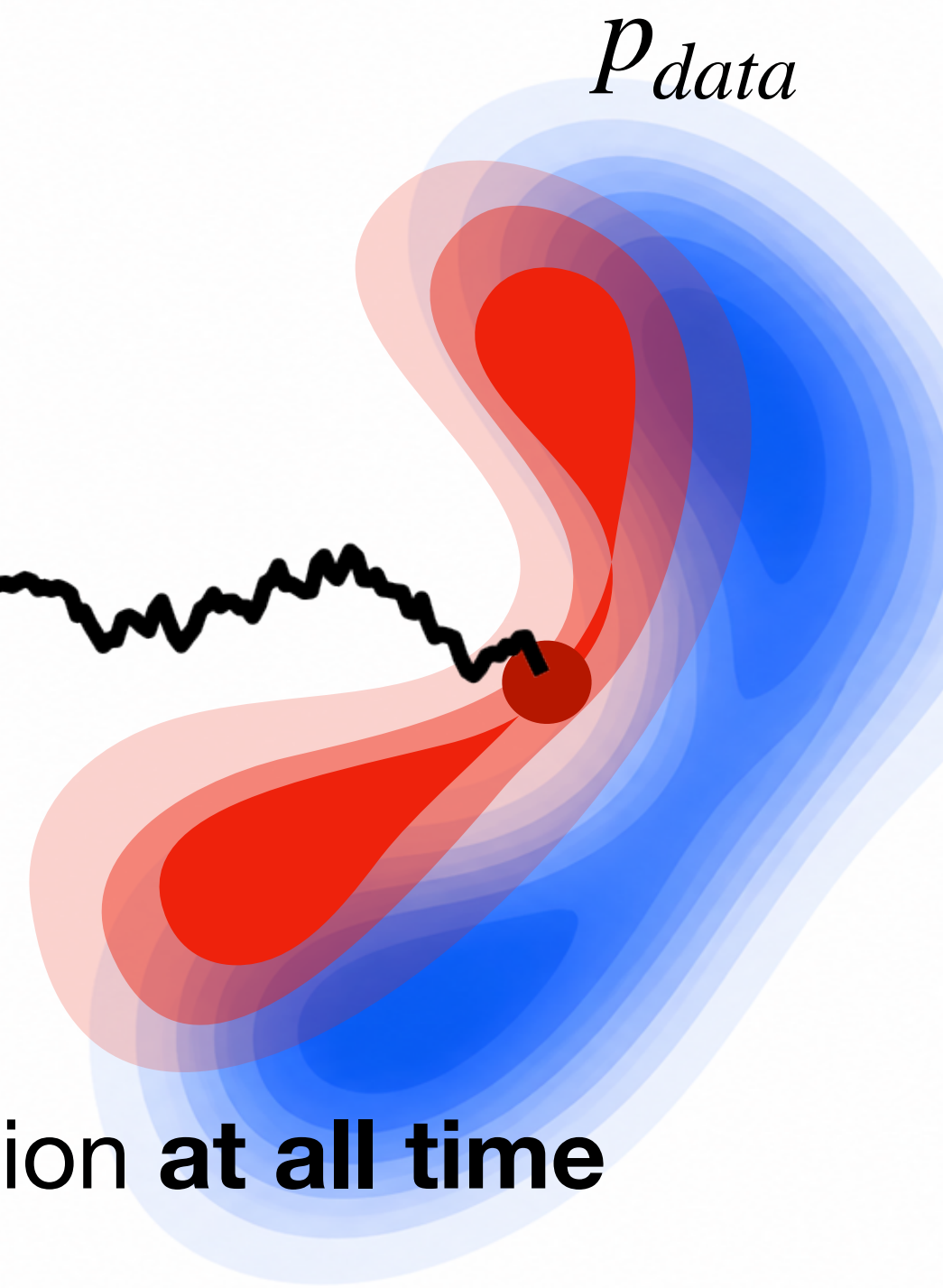
✓ Learns a continuous path
Gaussian $\rightarrow$ data



Easy training: local supervision **at all time**

p_{data}

p_θ



Generative models given samples of p_{data} , generate new samples of p_{data}

Flow-based methods:

→ Choose p_0 easy to sample

→ Find a map $T_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^d$ such that

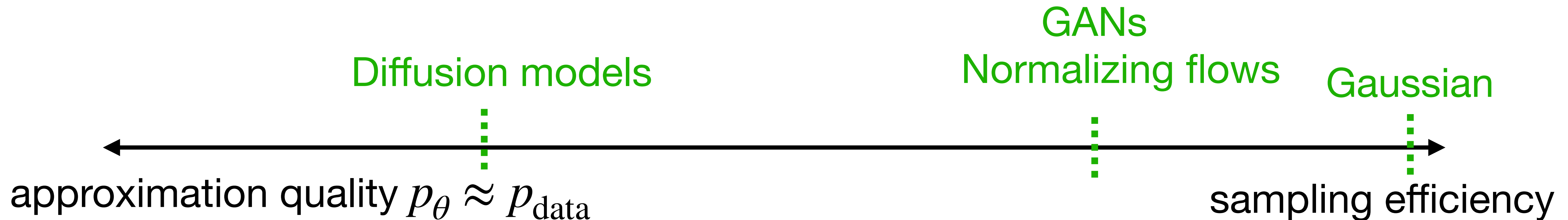
$x_0 \sim p_0, x = T_\theta(x_0)$ samples p_{data}

$$p_\theta := T_\theta \# p_0 \approx p_{data}$$

Challenges :

→ Very high dimension: $d \approx 10^6$ for images

→ Trade-off between training quality and sampling efficiency



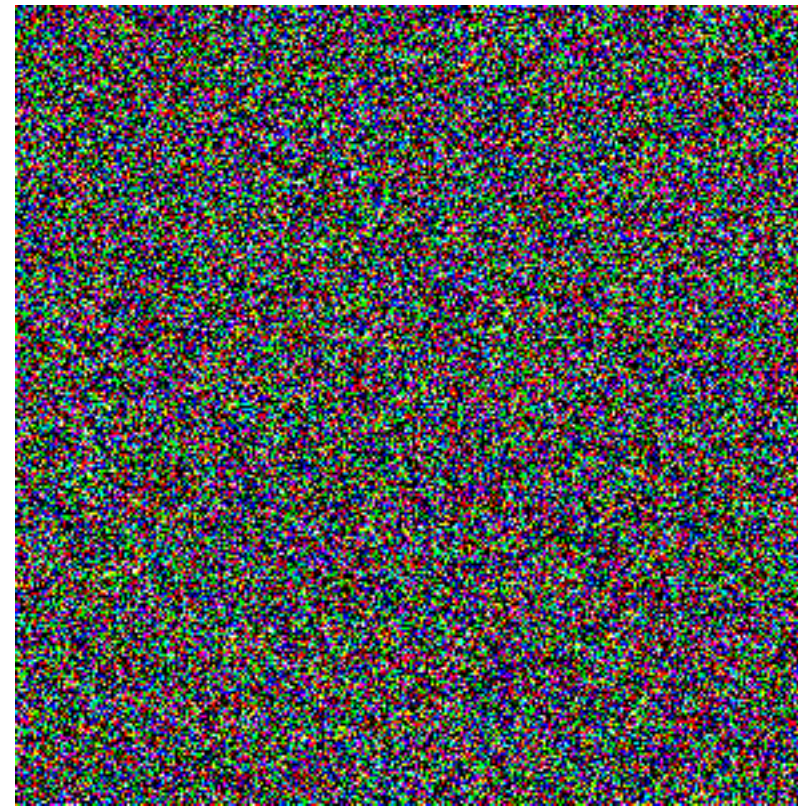
→ *What about recent 1-step generative models ?* Very expensive training

Denoising: the building block of diffusion models



$$x \sim p_{\text{data}}$$

$+$ $\sigma \cdot$

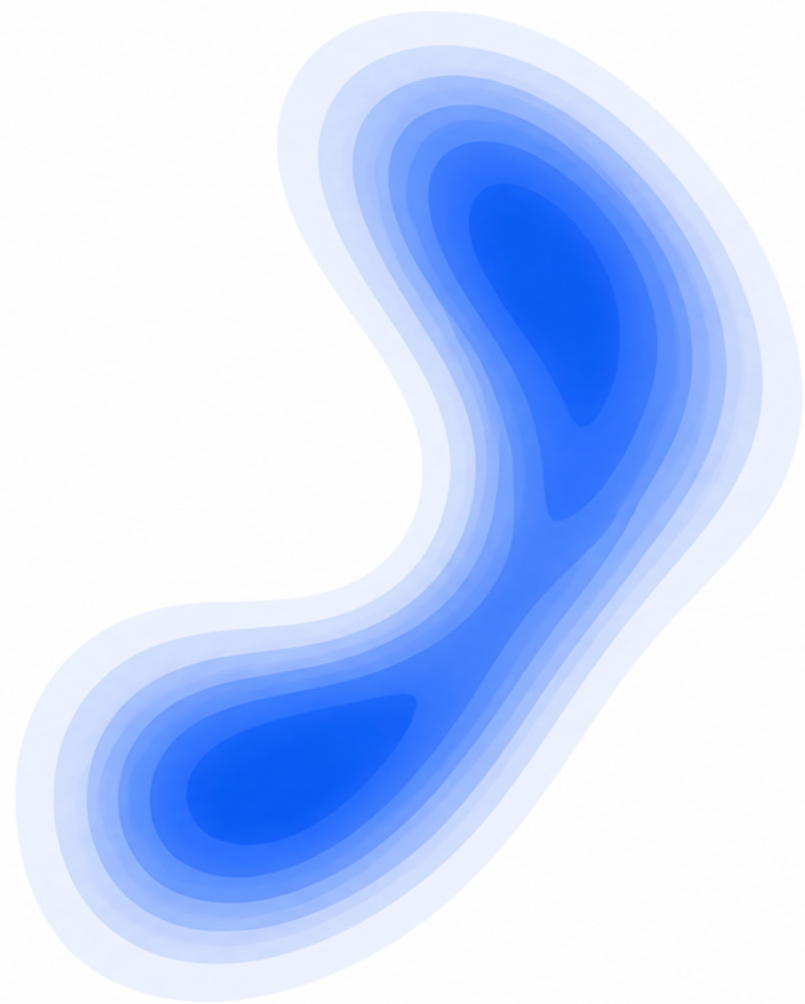


$$w \sim \mathcal{N}(0, \text{Id})$$

$=$

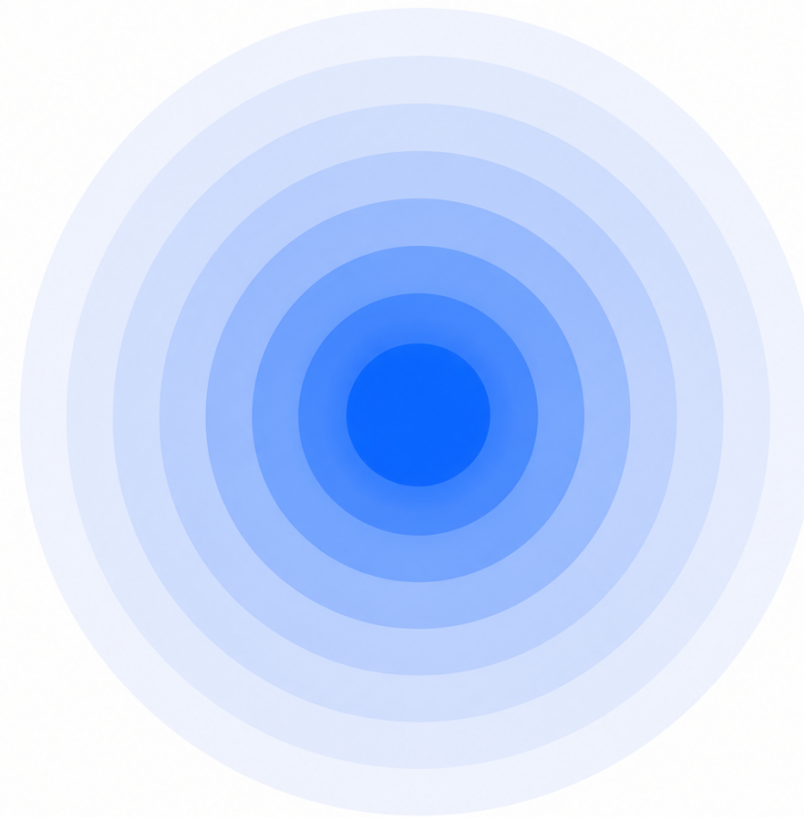


$$y = x + \sigma w$$



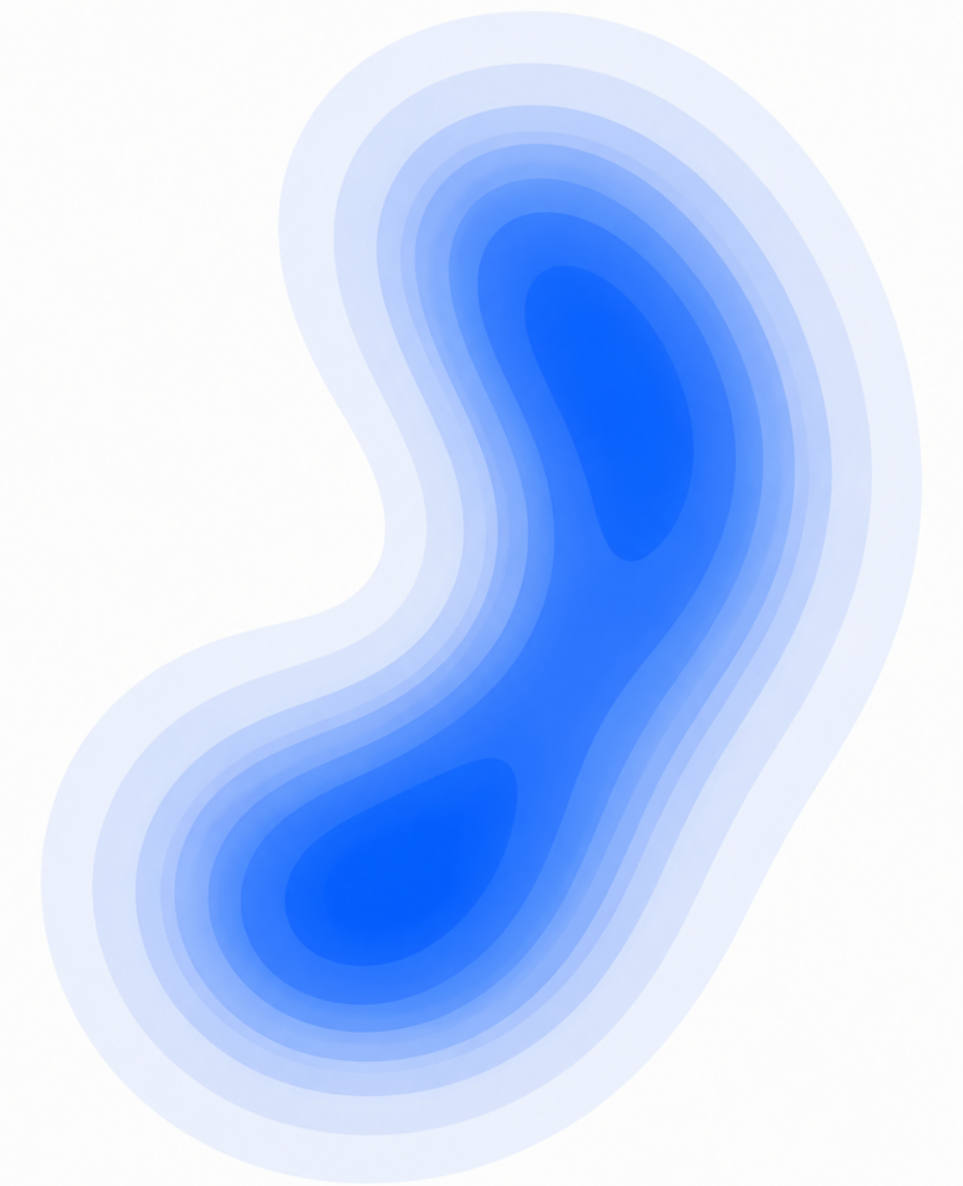
$$p_{\text{data}}$$

$\star$



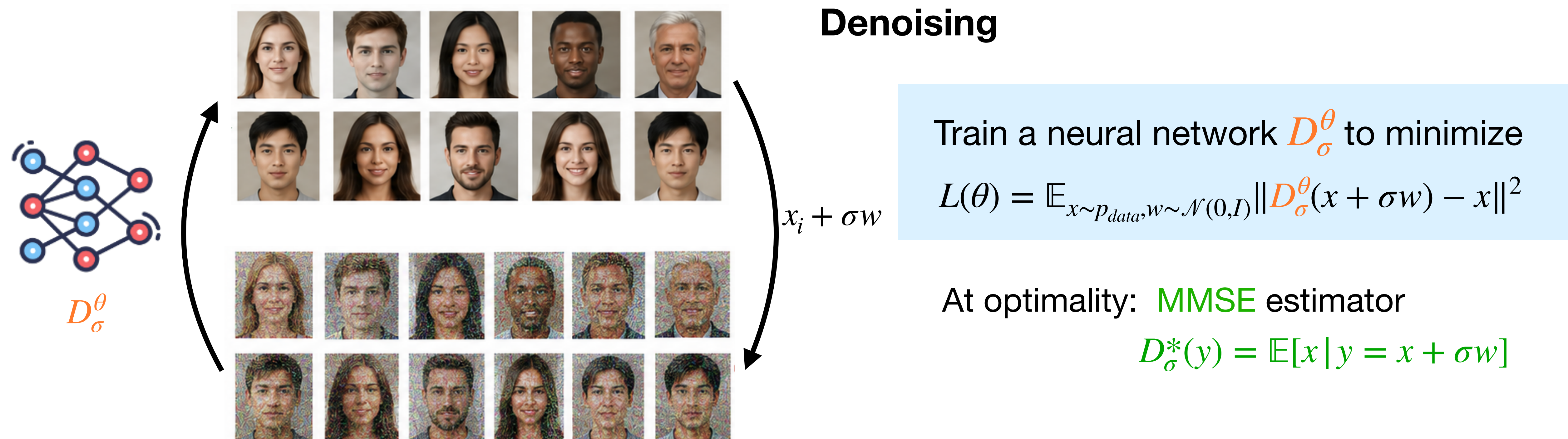
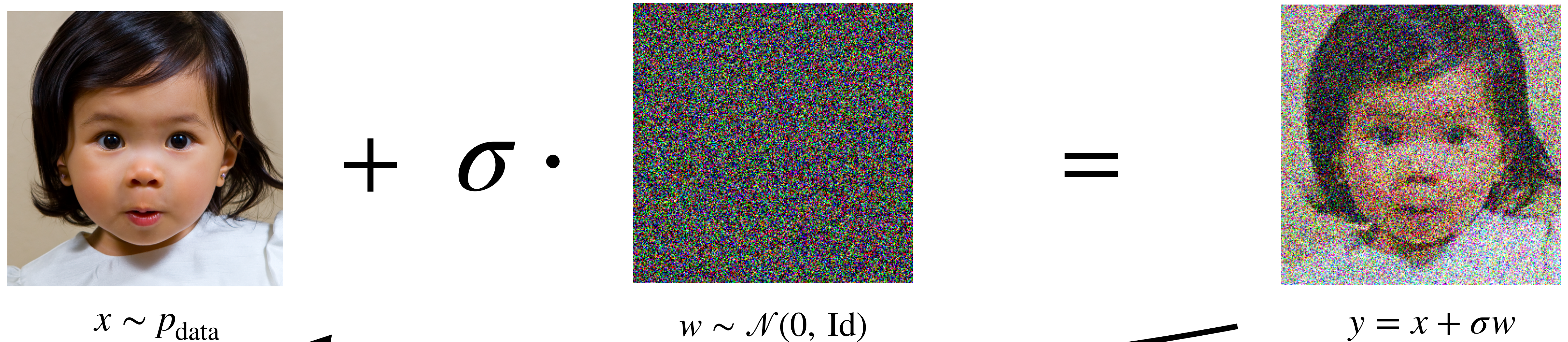
$$\mathcal{N}(0, \sigma^2 \text{Id})$$

$=$



$$p_{\sigma} = p \star \mathcal{N}(0, \sigma^2 \text{Id})$$

Denoising: the building block of diffusion models

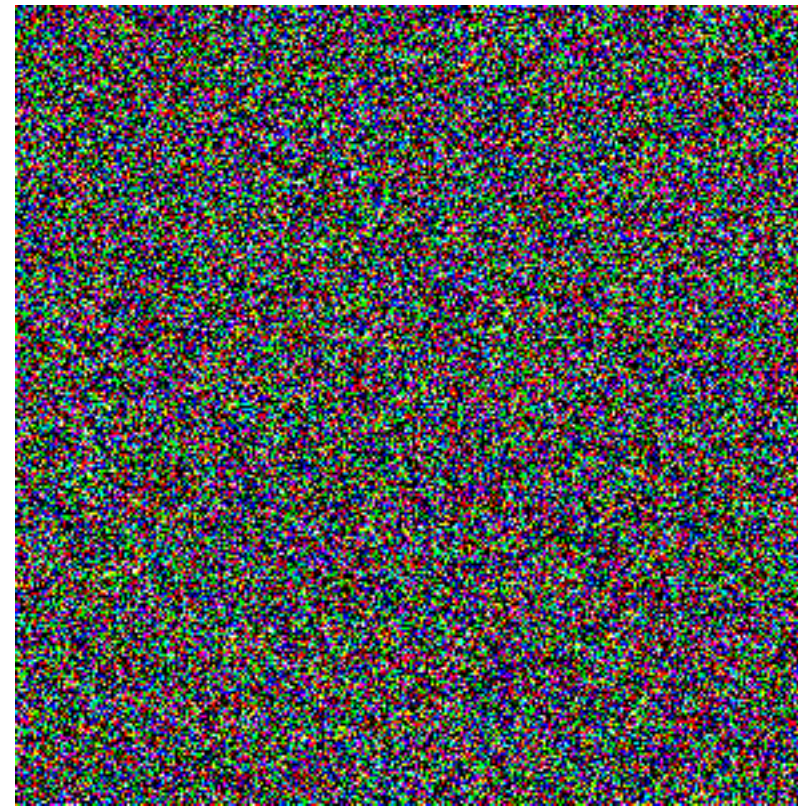


Denoising: the building block of diffusion models



$$x \sim p_{\text{data}}$$

+ $\sigma \cdot$



$$w \sim \mathcal{N}(0, \text{Id})$$

=



$$y = x + \sigma w$$

Denoising

Tweedie formula: [Efron, '11]

$$D_{\sigma}^*(y) = \mathbb{E}[x | y] = y + \sigma^2 \nabla \log p_{\sigma}(y)$$

Exercise: proof

trained **denoiser** $\Rightarrow$ approximation of $\mathbb{E}[x | y]$
 $\Rightarrow$ approximation of $\nabla \log p_{\sigma}$

Train a neural network D_{σ}^{θ} to minimize

$$L(\theta) = \mathbb{E}_{x \sim p_{\text{data}}, w \sim \mathcal{N}(0, I)} \|D_{\sigma}^{\theta}(x + \sigma w) - x\|^2$$

At optimality: **MMSE** estimator

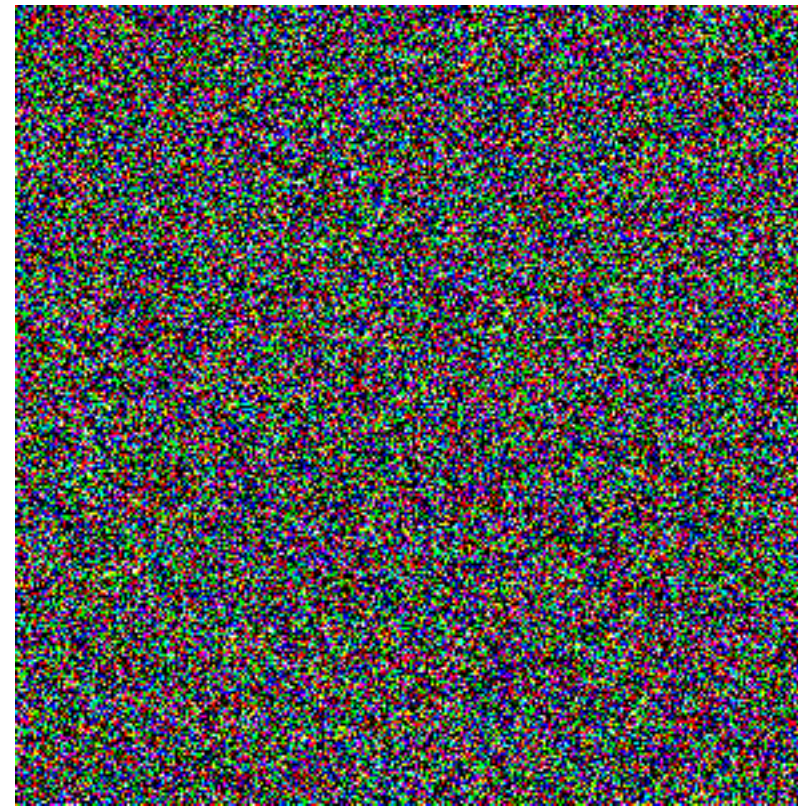
$$D_{\sigma}^*(y) = \mathbb{E}[x | y = x + \sigma w]$$

Denoising: the building block of diffusion models



$$x \sim p_{\text{data}}$$

+ $\sigma \cdot$



$$w \sim \mathcal{N}(0, \text{Id})$$

=



$$y = x + \sigma w$$

Denoising

Tweedie formula: [Efron, '11]

$$D_{\sigma}^*(y) = \mathbb{E}[x|y] = y + \sigma^2 \nabla \log p_{\sigma}(y)$$

Train a neural network D_{σ}^{θ} to minimize

$$L(\theta) = \mathbb{E}_{x \sim p_{\text{data}}, w \sim \mathcal{N}(0, I)} \|D_{\sigma}^{\theta}(x + \sigma w) - x\|^2$$

trained **denoiser** $\Leftrightarrow$ approximation of $\mathbb{E}[x|y]$

$\Leftrightarrow$ approximation of $\nabla \log p_{\sigma}$

*It's an **equivalence**!*

Denoising Score Matching [Vincent, '11]

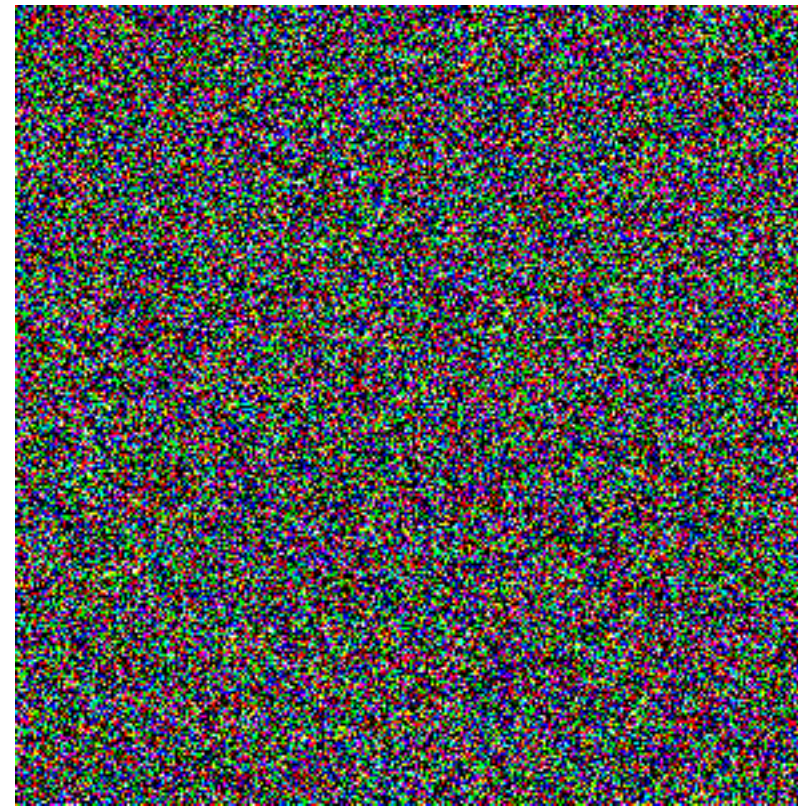
$$L(\theta) = \mathbb{E}_{y \sim p_{\sigma}} \|D_{\sigma}^{\theta}(y) - (y + \sigma^2 \nabla \log p_{\sigma}(y))\|^2 + cst$$

Denoising: the building block of diffusion models



$$x \sim p_{\text{data}}$$

$+$ $\sigma \cdot$



$$w \sim \mathcal{N}(0, \text{Id})$$

$=$



$$y = x + \sigma w$$

Denoising

Tweedie formula: [Efron, '11]

$$D_{\sigma}^{*}(y) = \mathbb{E}[x | y] = y + \sigma^2 \nabla \log p_{\sigma}(y)$$

« 2nd order Tweedie » formula:

$$\text{Cov}[x | y] = \sigma^2 \nabla_y D_{\sigma}^{*}(y)$$

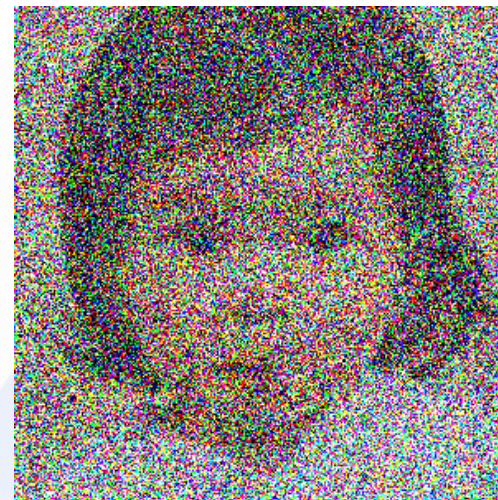
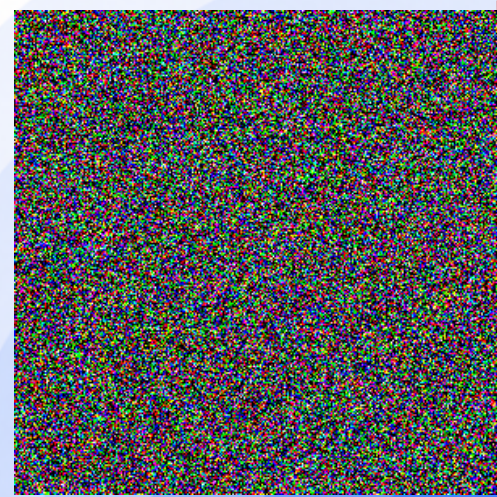
Exercise: proof

trained **denoiser** $\Leftrightarrow$ approximation of $\mathbb{E}[x | y]$

∇ (trained **denoiser**) $\Rightarrow$ approximation of $\text{Cov}[x | y]$

Diffusion models

Noising process

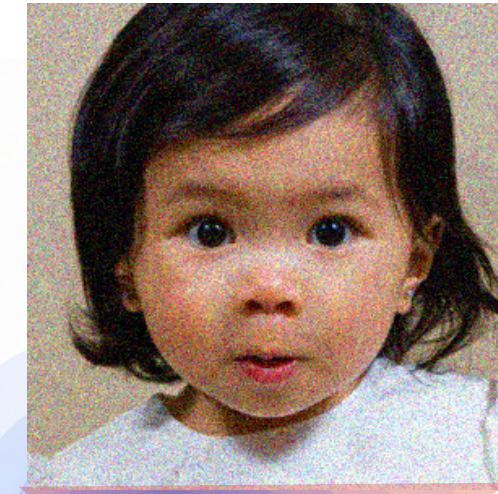
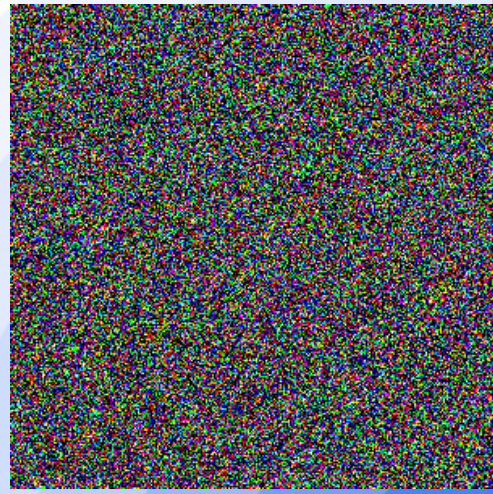


Generative process



Diffusion models

Noising process

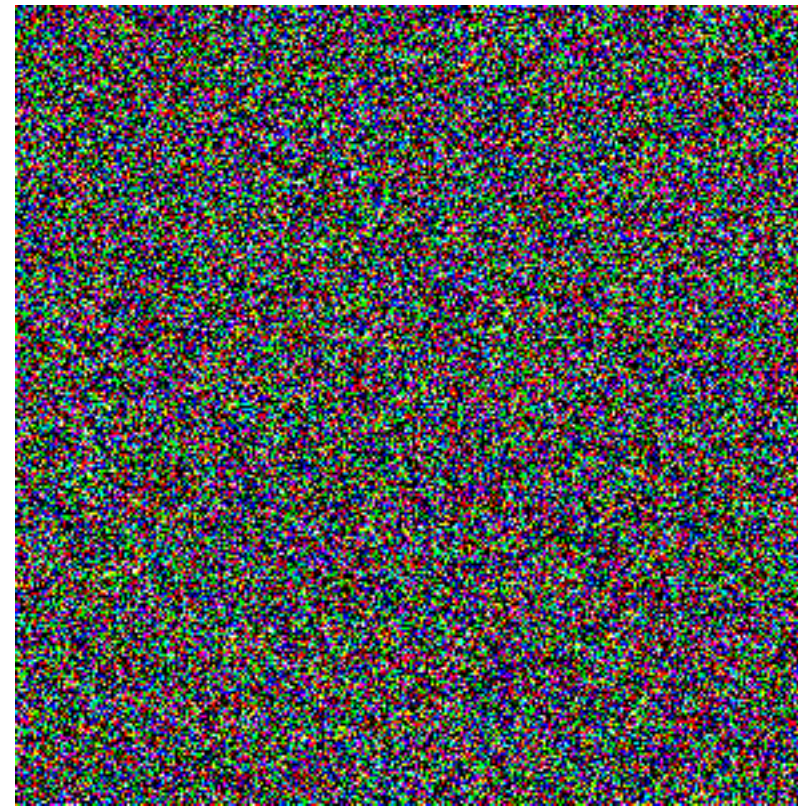


Forward noising process: from data to Gaussian



$$x \sim p_{\text{data}}$$

$+$ $\sigma_1 \cdot$

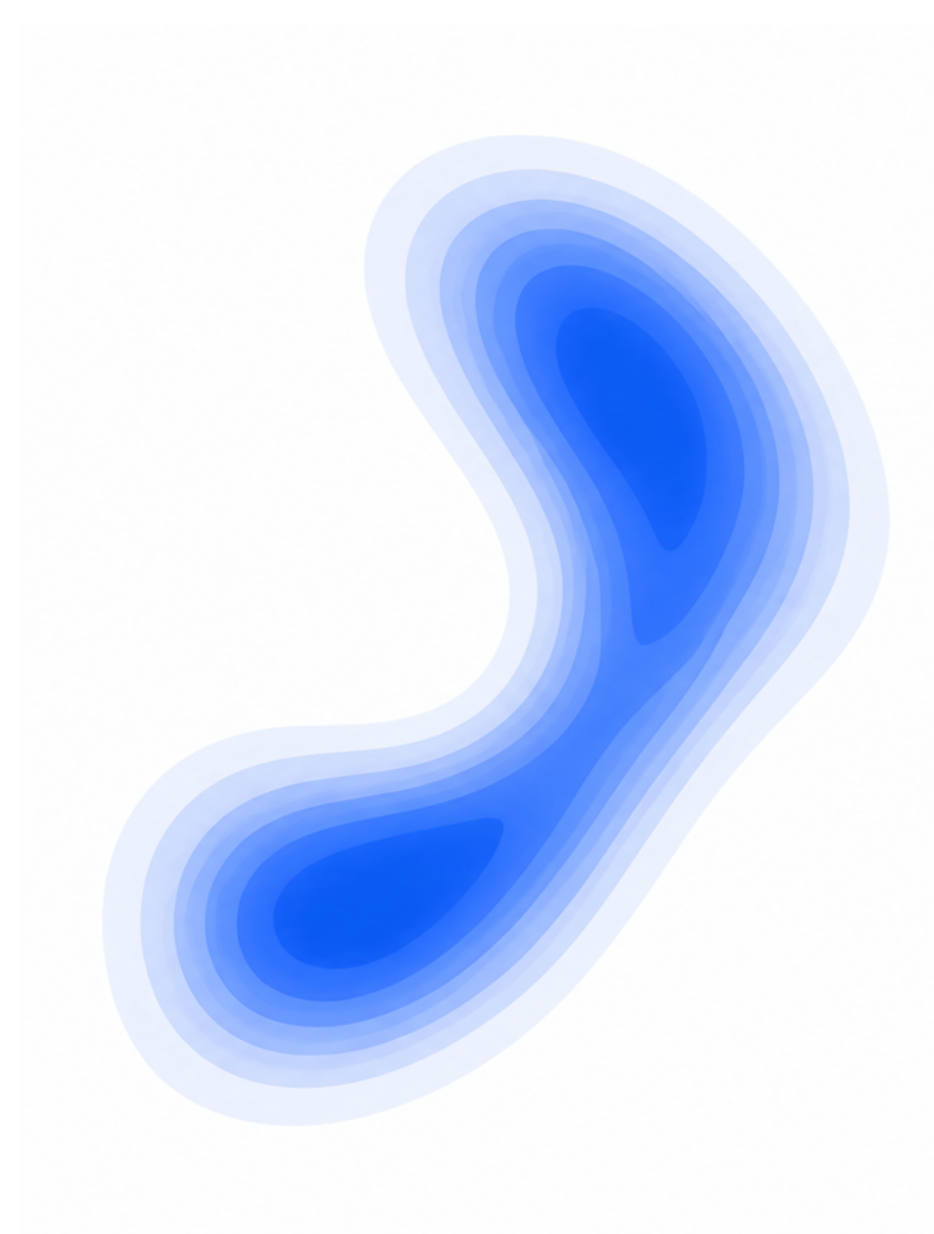


$$w \sim \mathcal{N}(0, \text{Id})$$

$=$

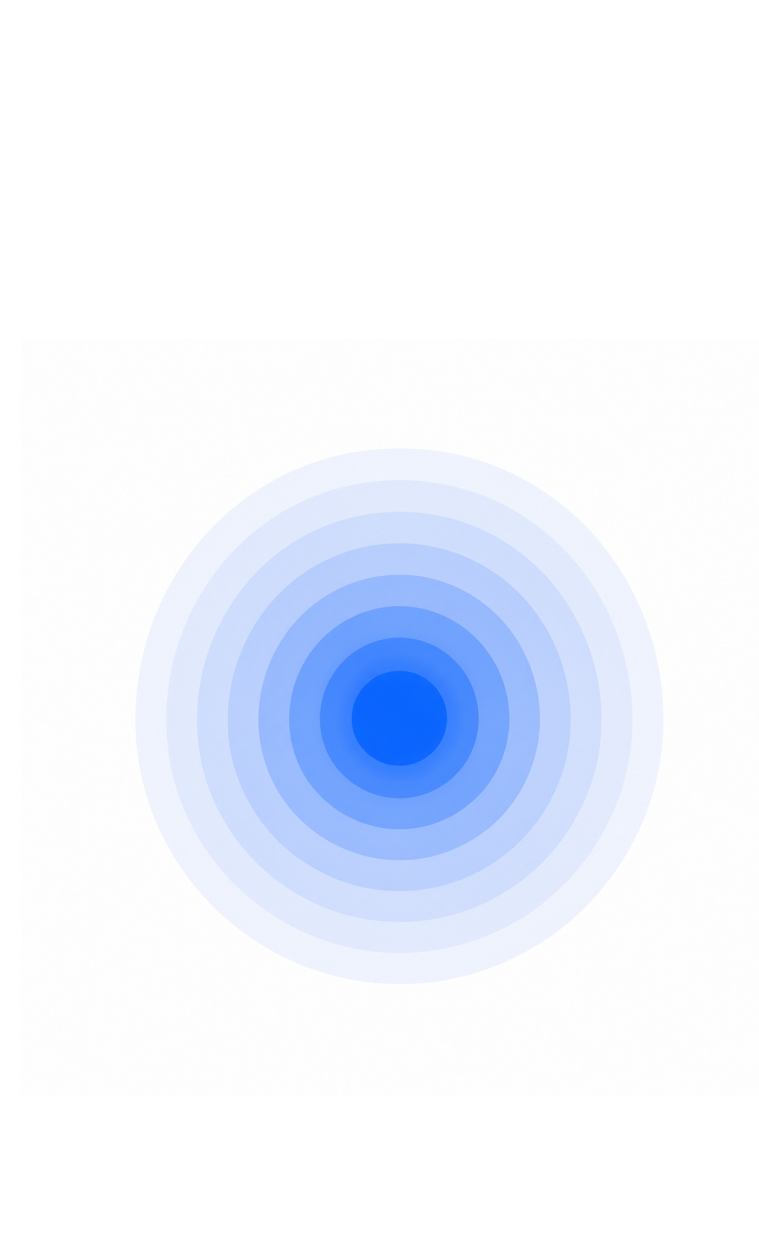


$$y = x + \sigma_1 w$$



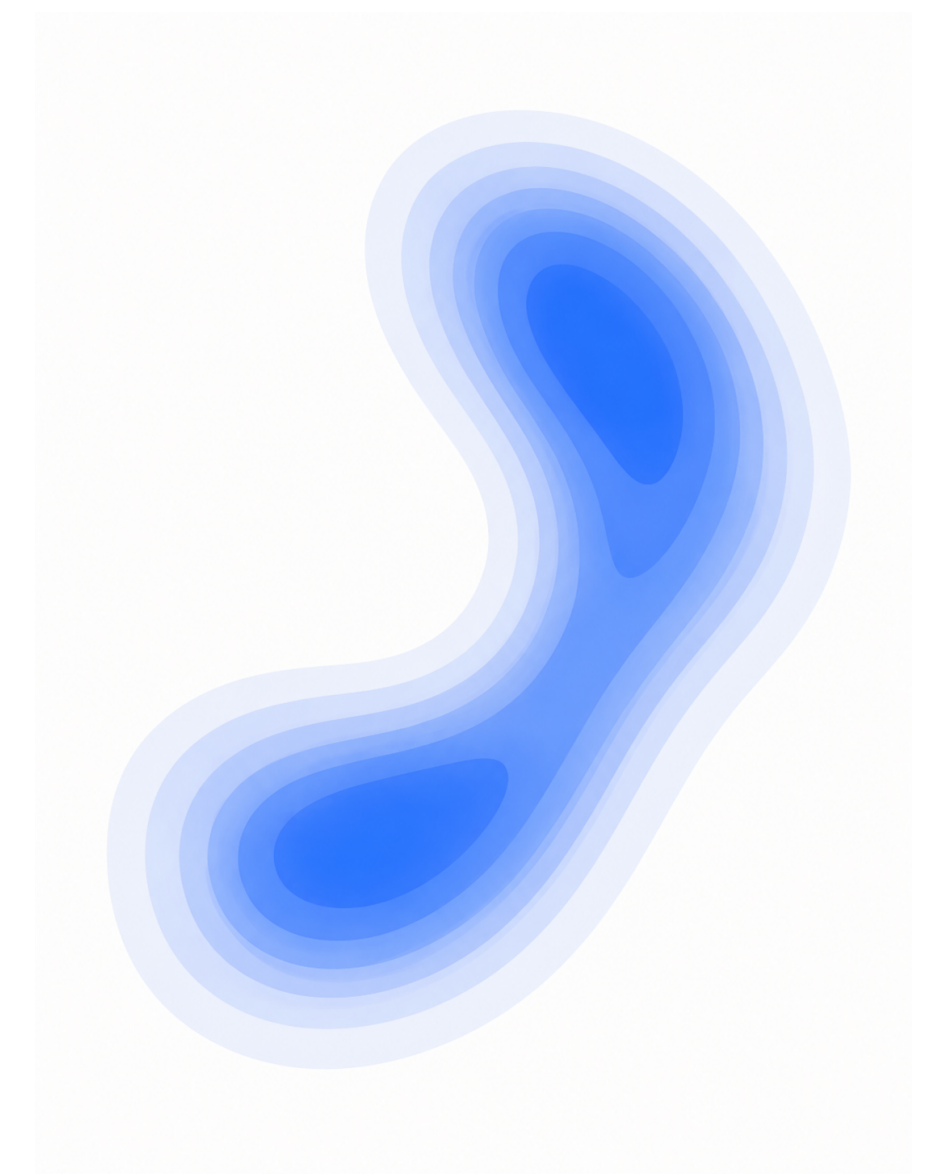
$$p_{\text{data}}$$

$\star$



$$\mathcal{N}(0, \sigma_1^2 \text{Id})$$

$=$



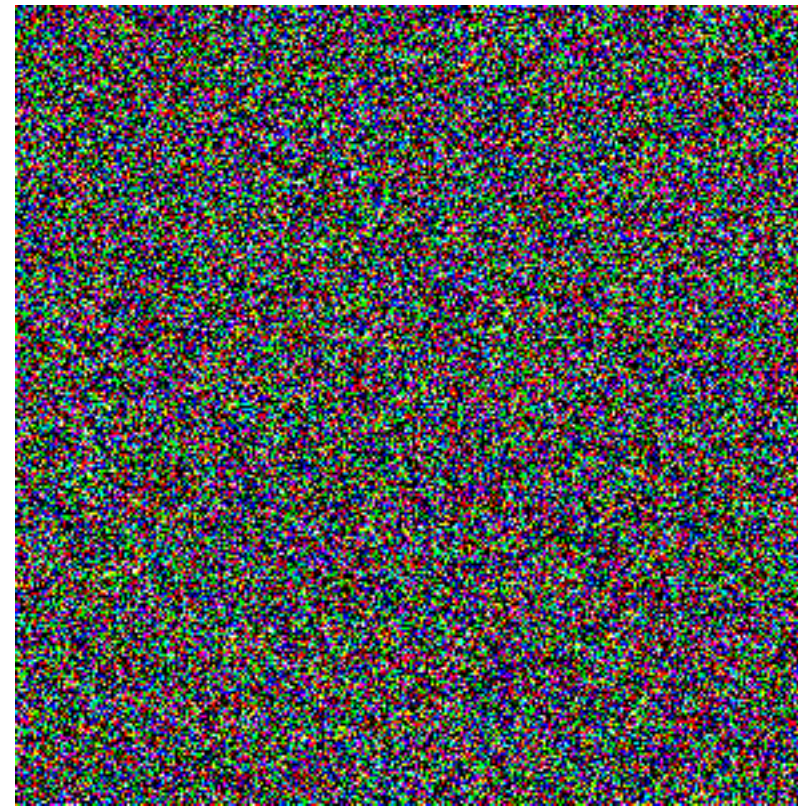
$$p_{\sigma_1} = p_{\text{data}} \star \mathcal{N}(0, \sigma_1^2 \text{Id})$$

Forward noising process: from data to Gaussian



$$x \sim p_{\text{data}}$$

$+$ $\sigma_2 \cdot$

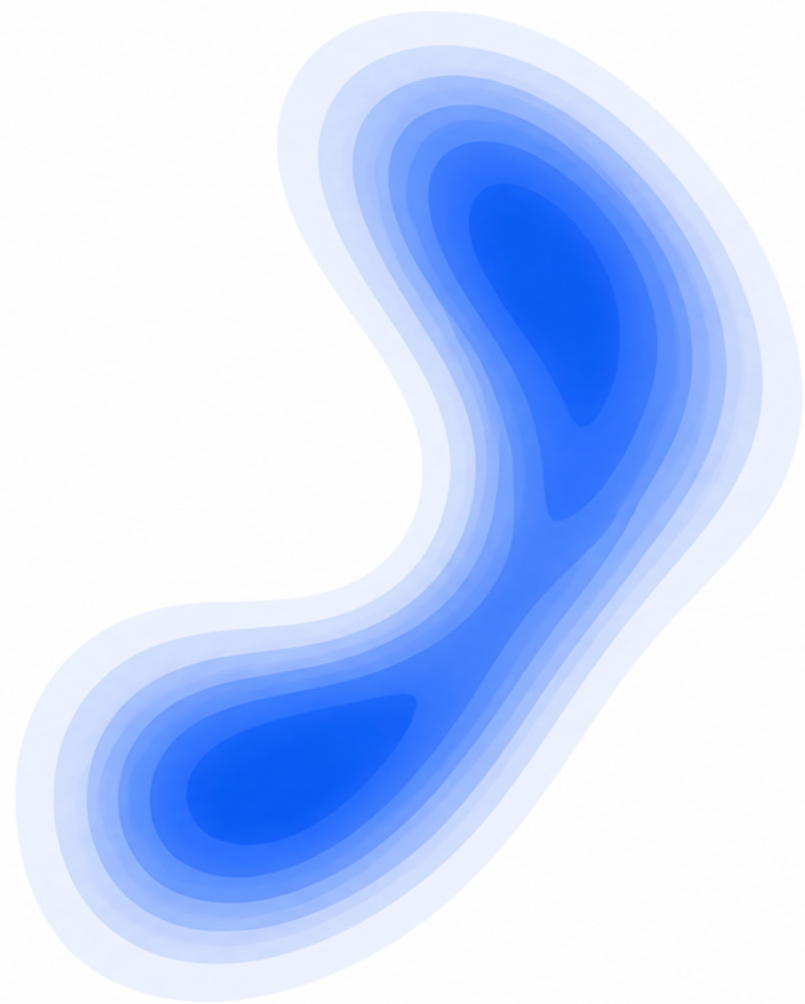


$$w \sim \mathcal{N}(0, \text{Id})$$

$=$

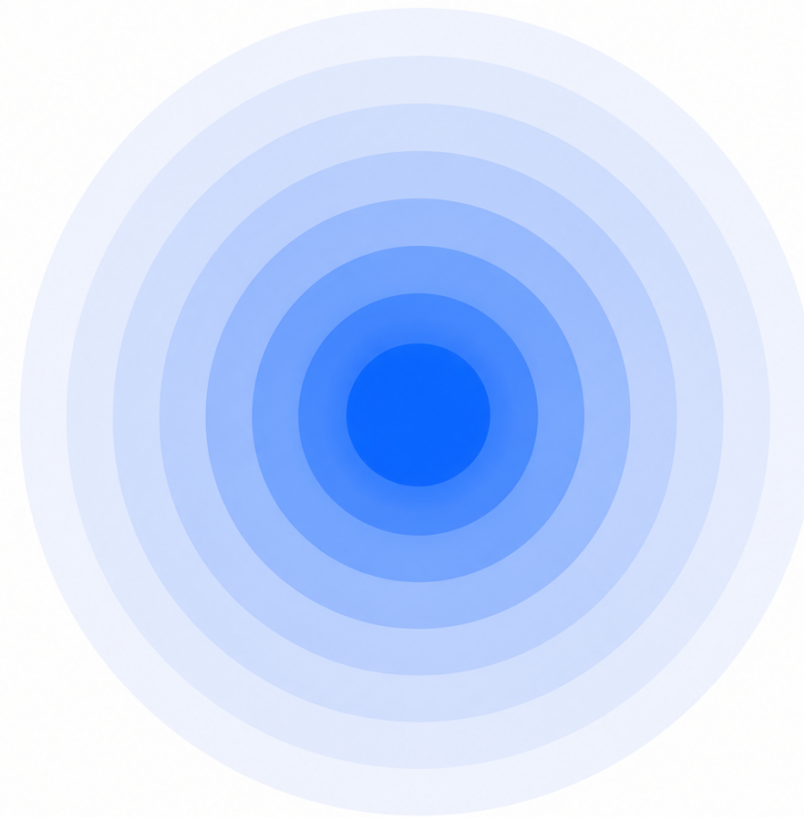


$$y = x + \sigma_2 w$$



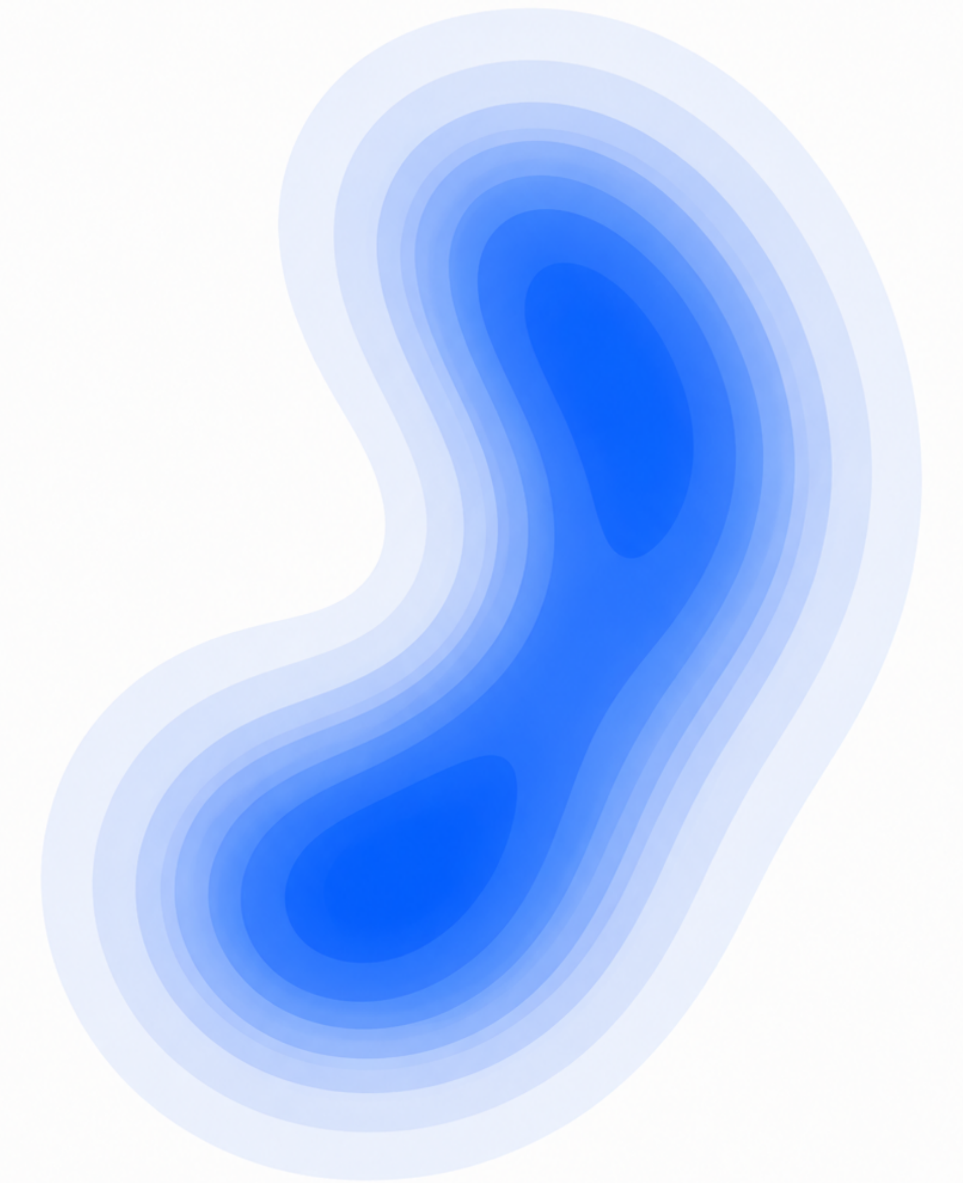
$$p_{\text{data}}$$

$\star$



$$\mathcal{N}(0, \sigma_2^2 \text{Id})$$

$=$



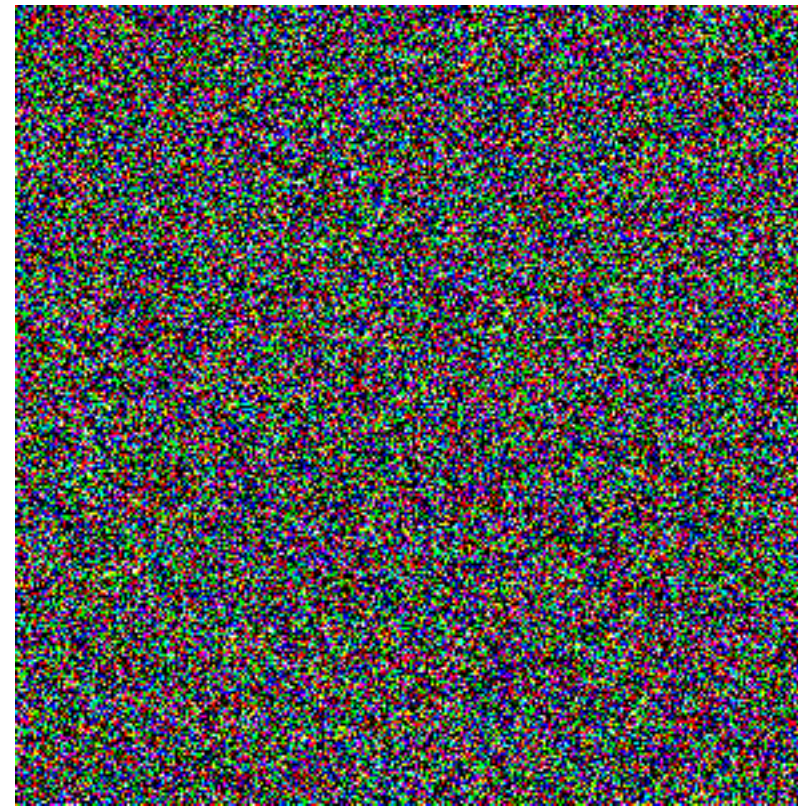
$$p_{\sigma_2} = p_{\text{data}} \star \mathcal{N}(0, \sigma_2^2 \text{Id})$$

Forward noising process: from data to Gaussian



$$x \sim p_{\text{data}}$$

$+$ $\sigma_3 \cdot$

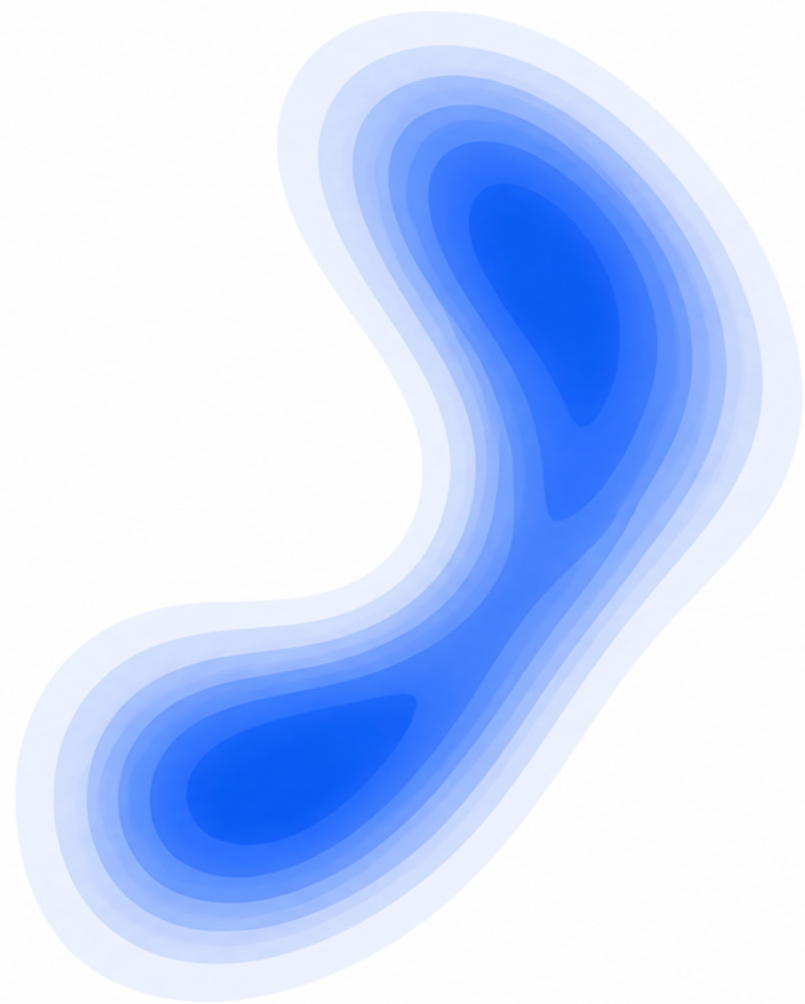


$$w \sim \mathcal{N}(0, \text{Id})$$

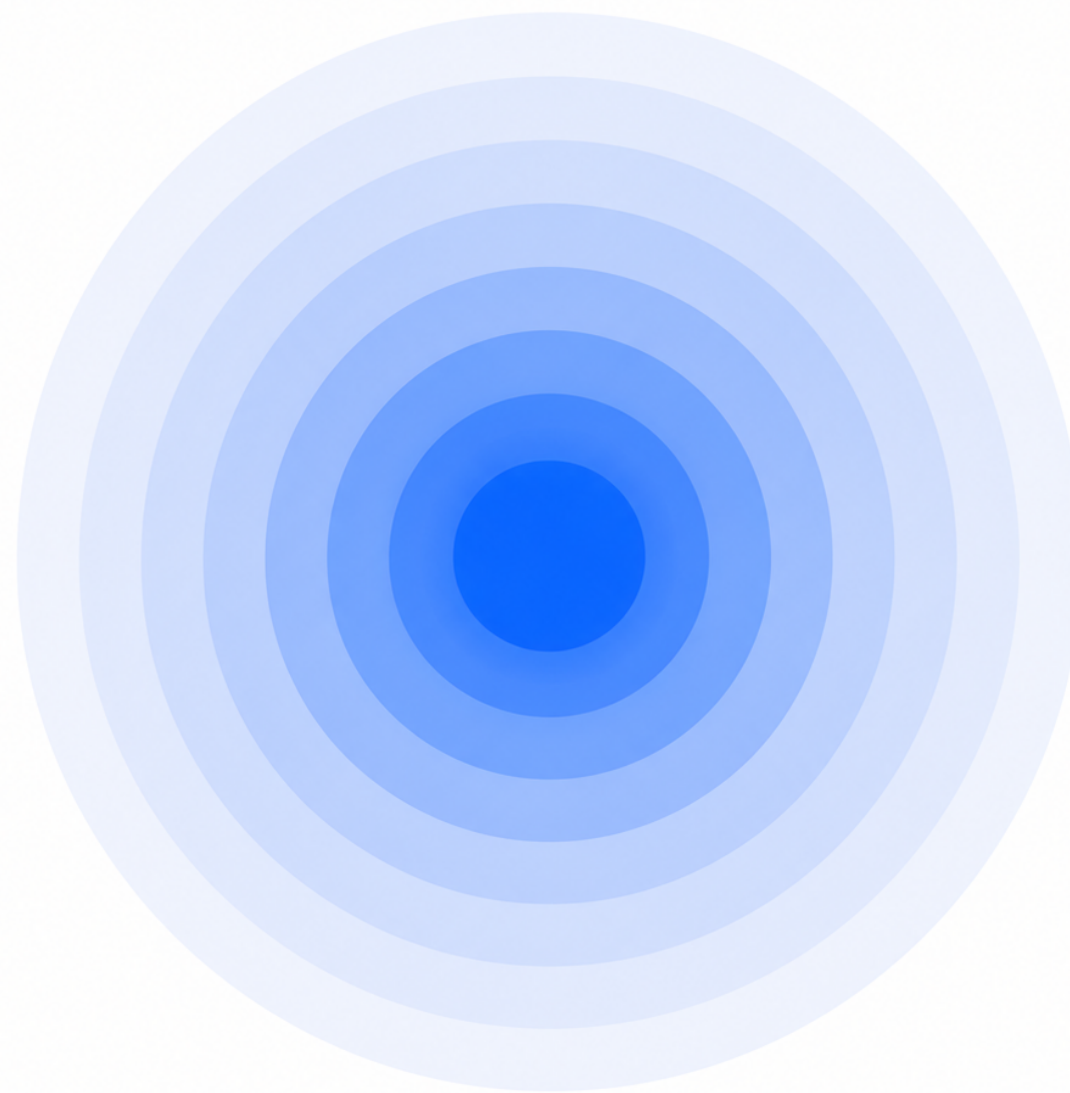
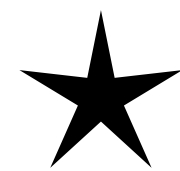
$=$



$$y = x + \sigma_3 w$$

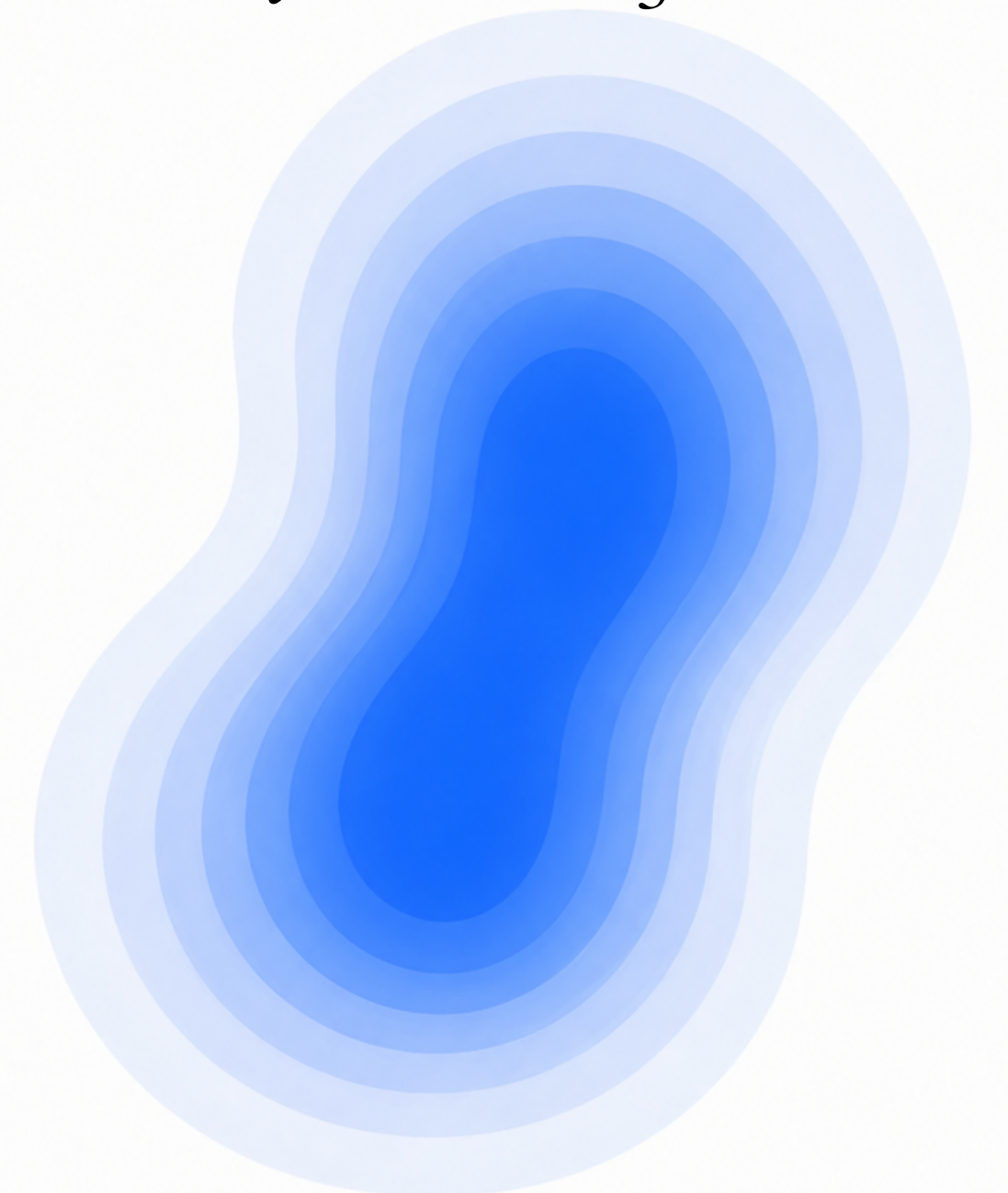


$$p_{\text{data}}$$



$$\mathcal{N}(0, \sigma_3^2 \text{Id})$$

$=$



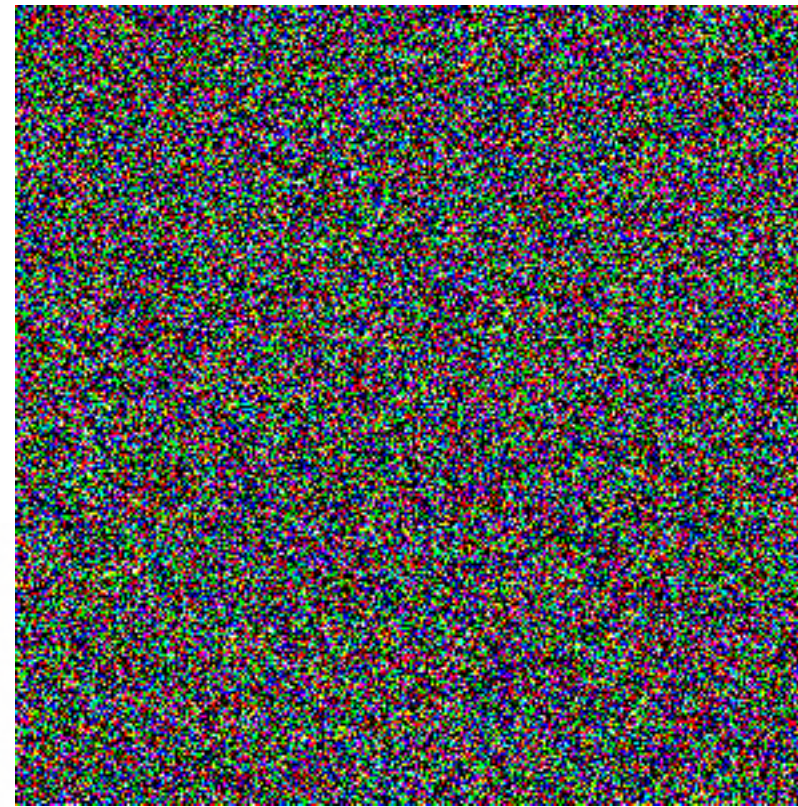
$$p_{\sigma_3} = p_{\text{data}} \star \mathcal{N}(0, \sigma_3^2 \text{Id})$$

Forward noising process: from data to Gaussian



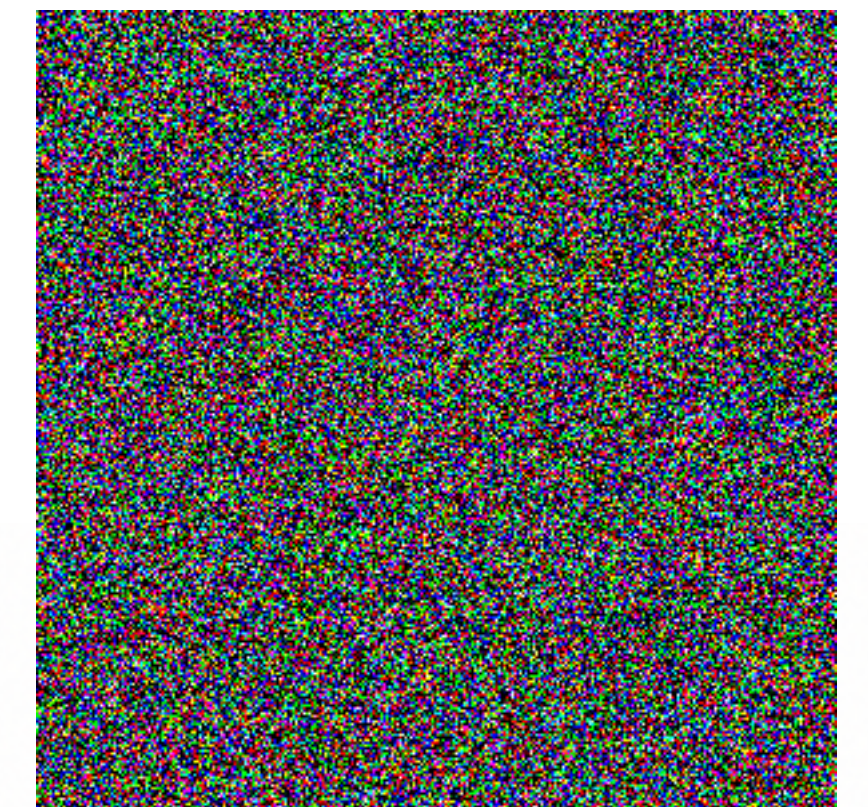
$$x \sim p_{\text{data}}$$

$+ \sigma_4 \cdot$

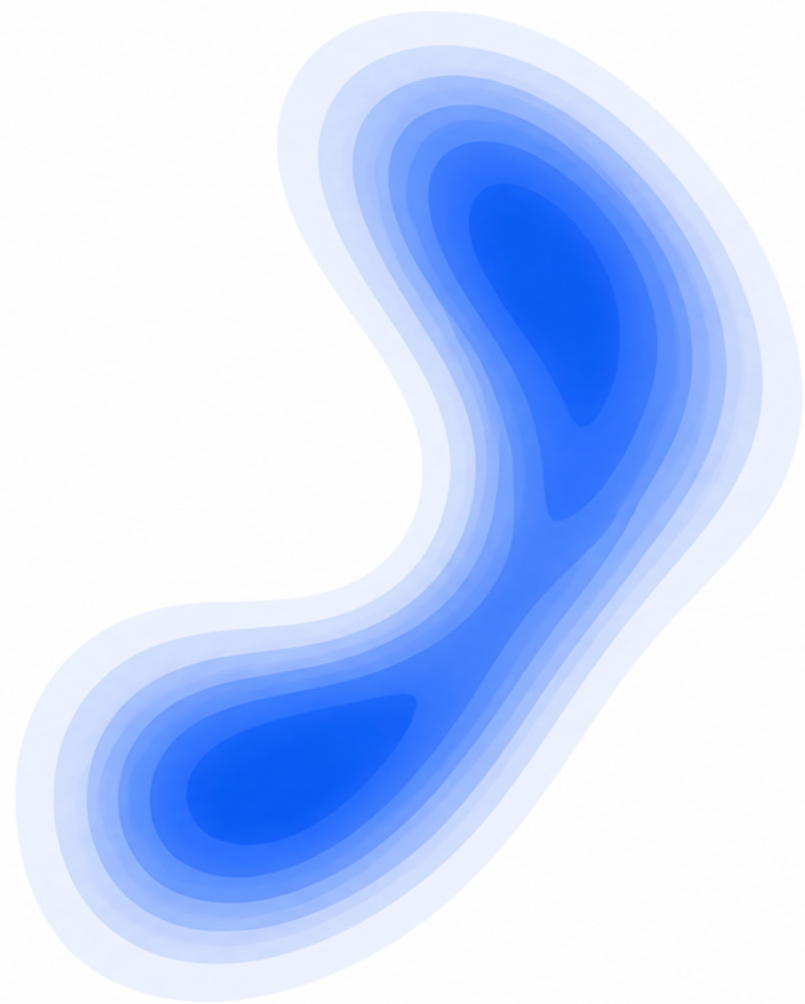


$$w \sim \mathcal{N}(0, \text{Id})$$

$=$

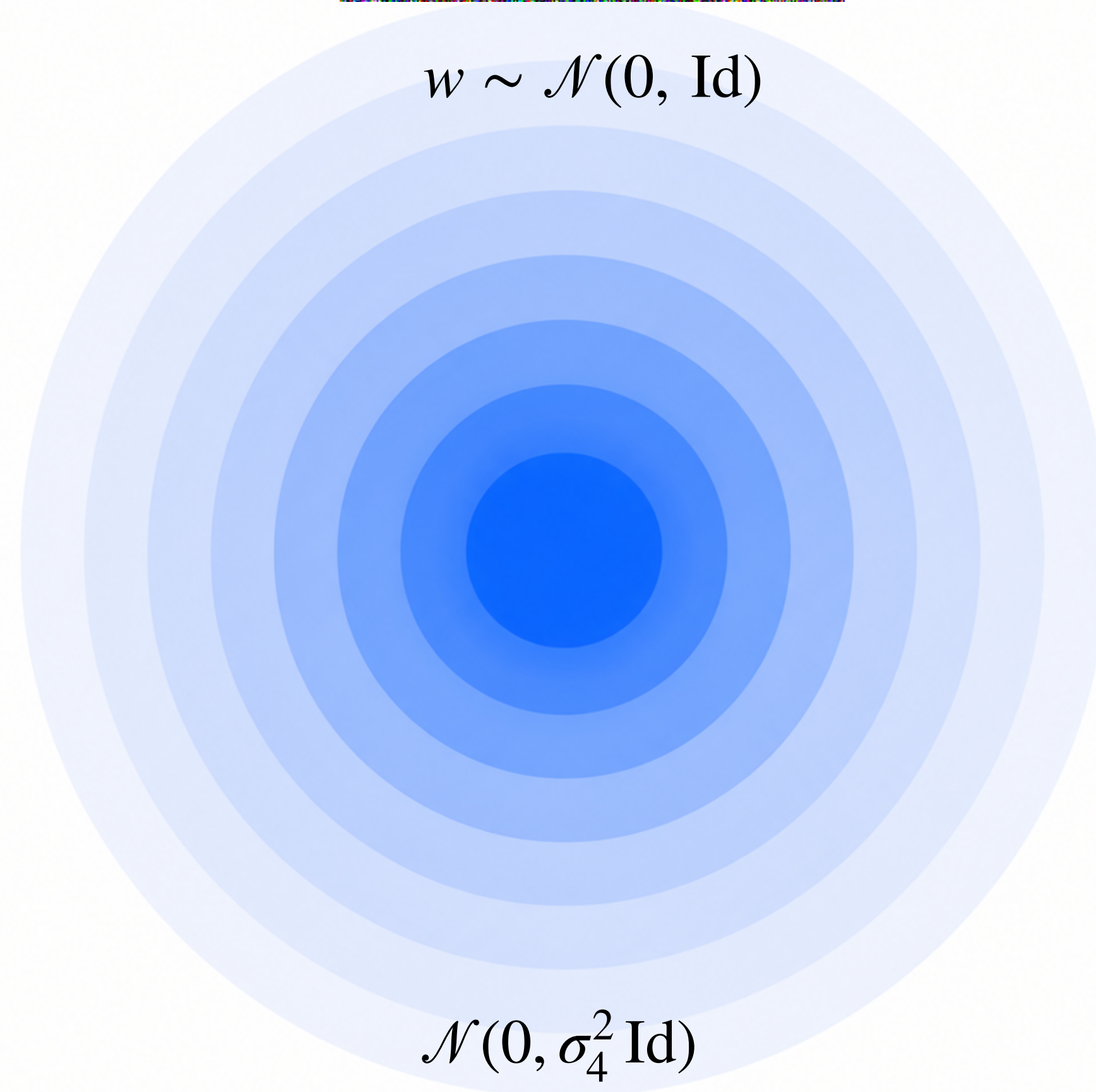


$$y = x + \sigma_4 w$$



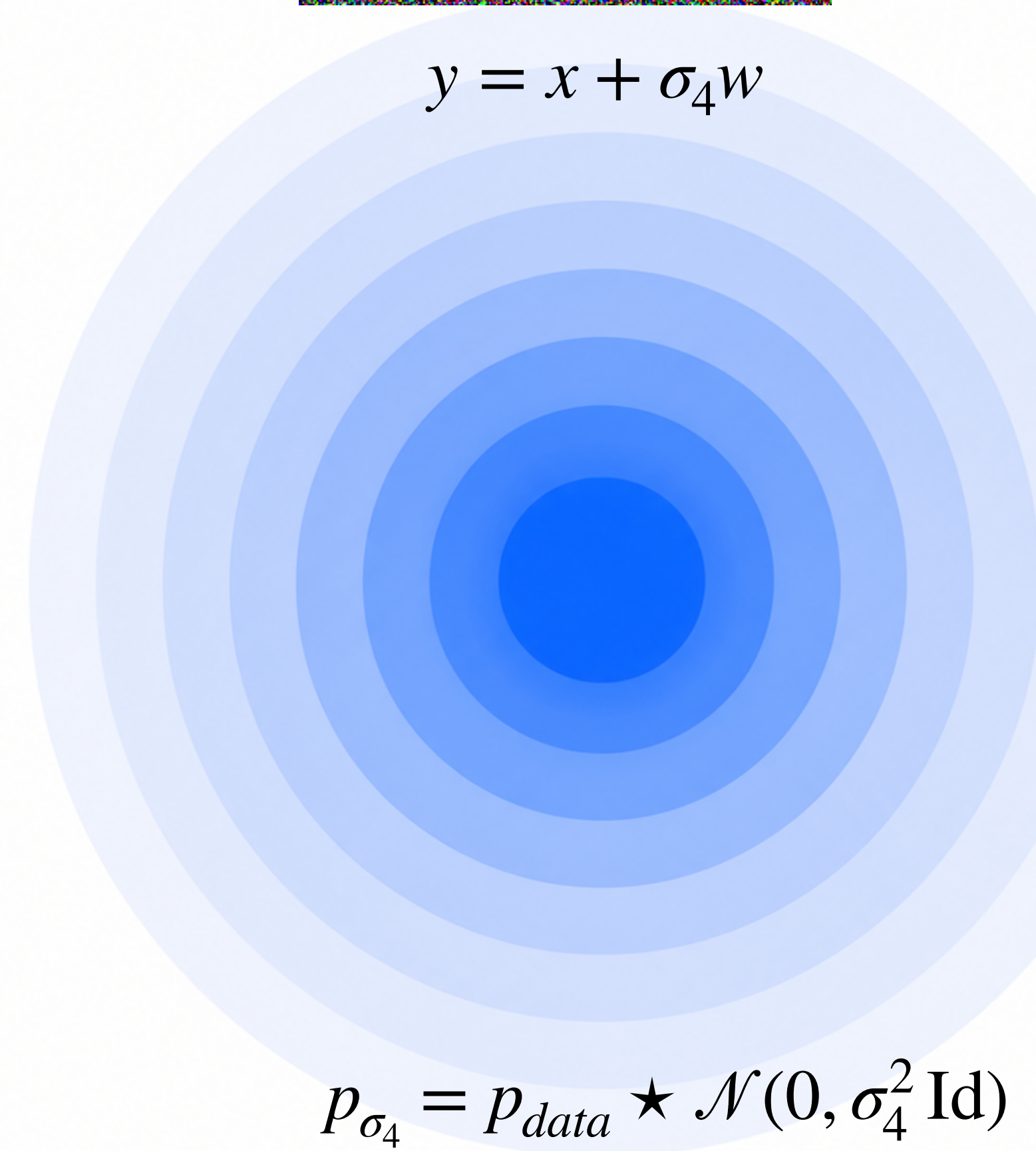
$$p_{\text{data}}$$

$\star$



$$\mathcal{N}(0, \sigma_4^2 \text{Id})$$

$=$

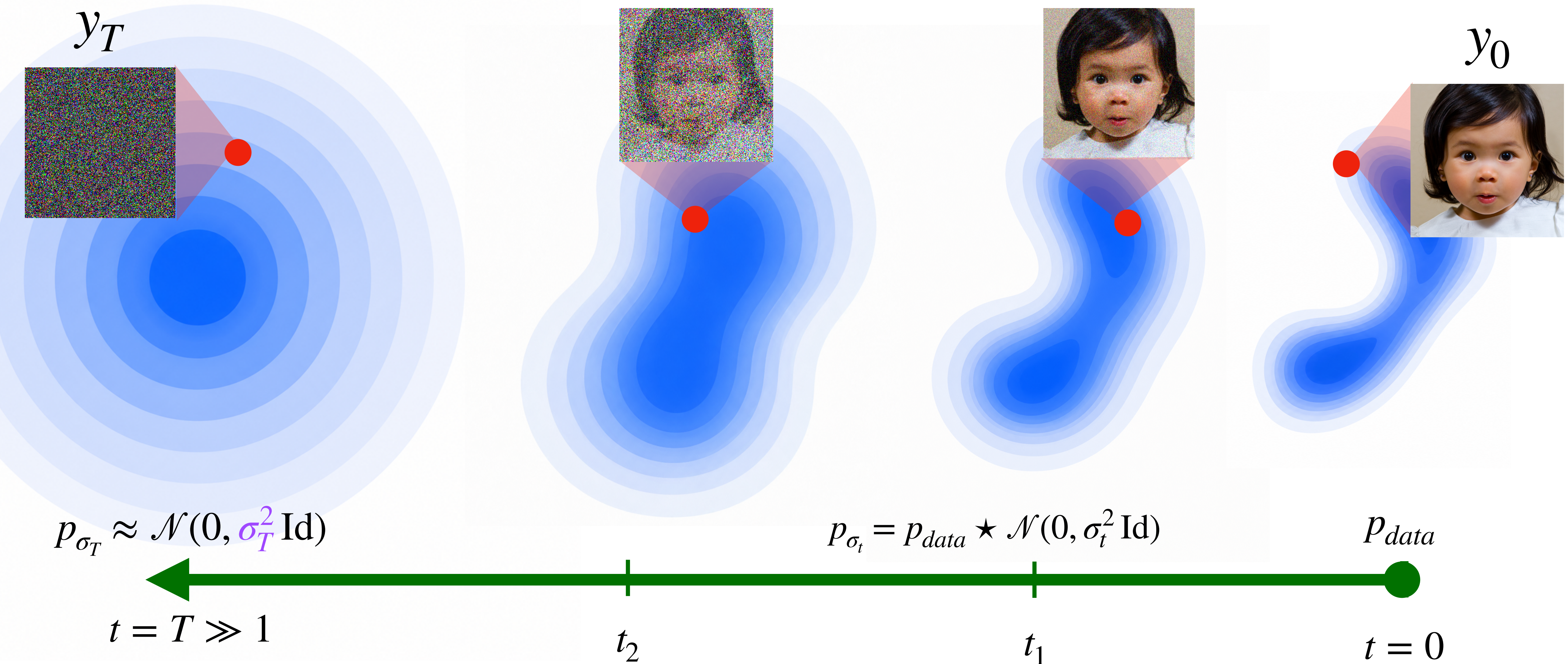


$$p_{\sigma_4} = p_{\text{data}} \star \mathcal{N}(0, \sigma_4^2 \text{Id})$$

Forward noising process: from data to noise

$$y_t = y_0 + \sigma_t w$$

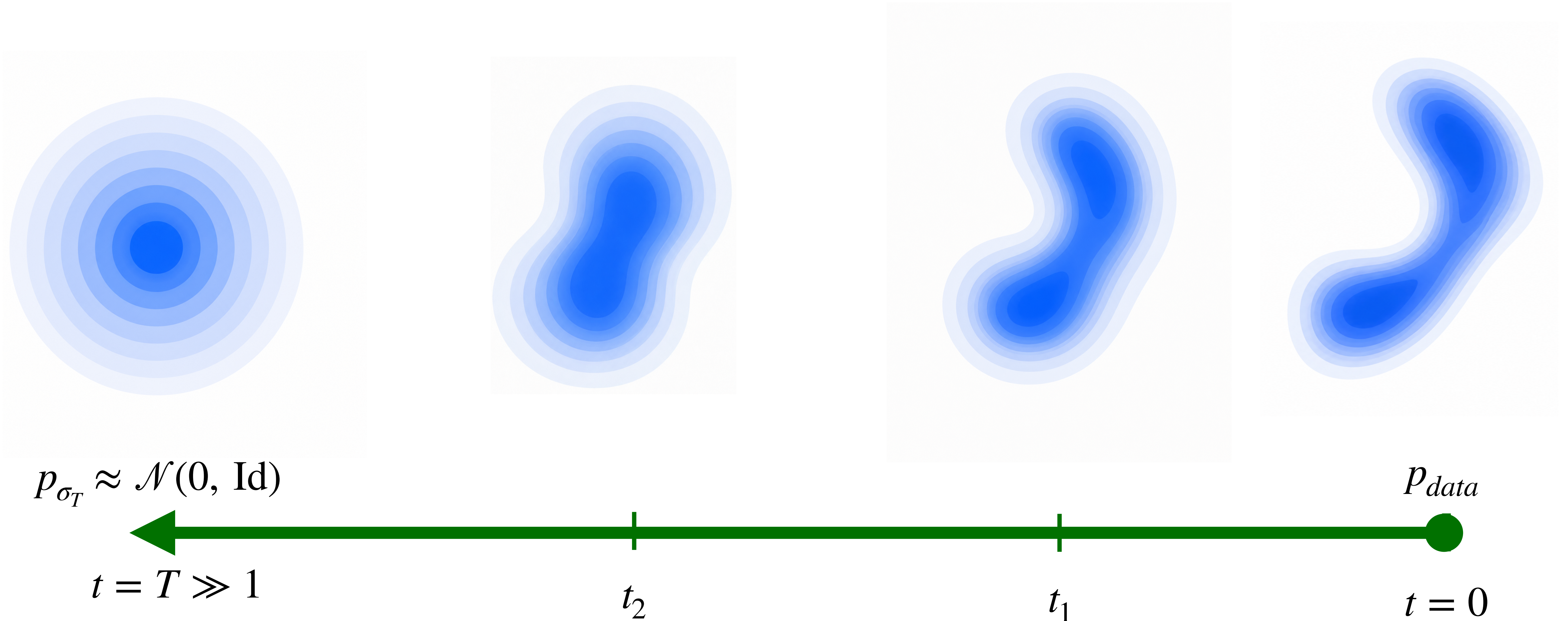
With $\sigma_0 = 0$ and increasing noise level σ_t e.g. $\sigma_t = t^\beta$



Forward noising process: from data to noise

$$y_t = s_t(y_0 + \sigma_t w) \quad \text{With } \sigma_0 = 0 \text{ and increasing noise level } \sigma_t \text{ e.g. } \sigma_t = t^\beta$$

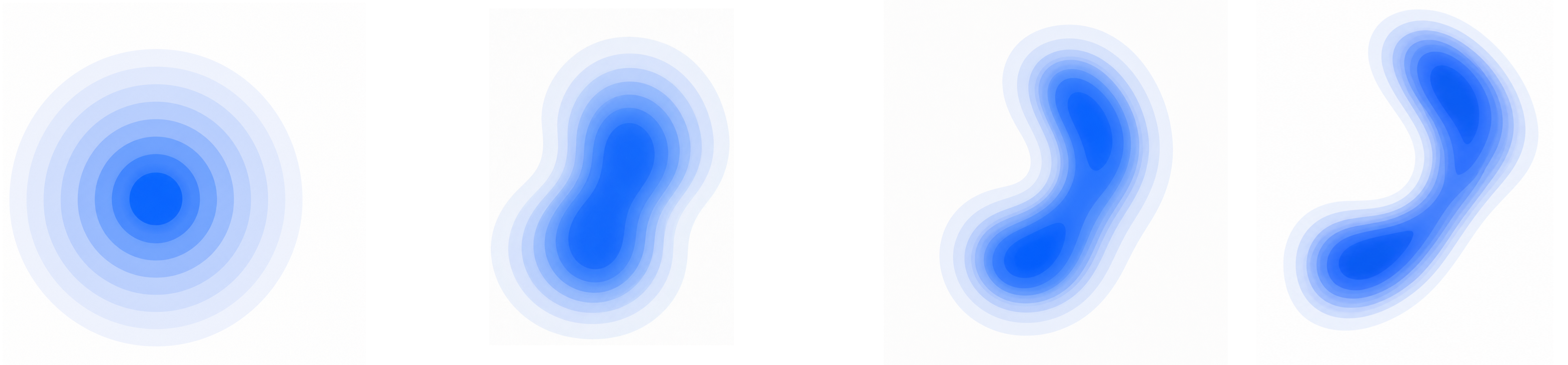
Exercise: propose a choice of s_t such that $\text{Cov}(y_t) \rightarrow \text{Id}$



Forward noising process: from data to noise

$$y_t = s_t(y_0 + \sigma_t w) \quad \text{With } \sigma_0 = 0 \text{ and increasing noise level } \sigma_t \text{ e.g. } \sigma_t = t^\beta$$

Variance Preserving: $s_t = \sqrt{\frac{1}{1 + \sigma_t^2}}$



$$p_{\sigma_T} \approx \mathcal{N}(0, \text{Id})$$

$$t = T \gg 1$$

$$t_2$$

$$t_1$$

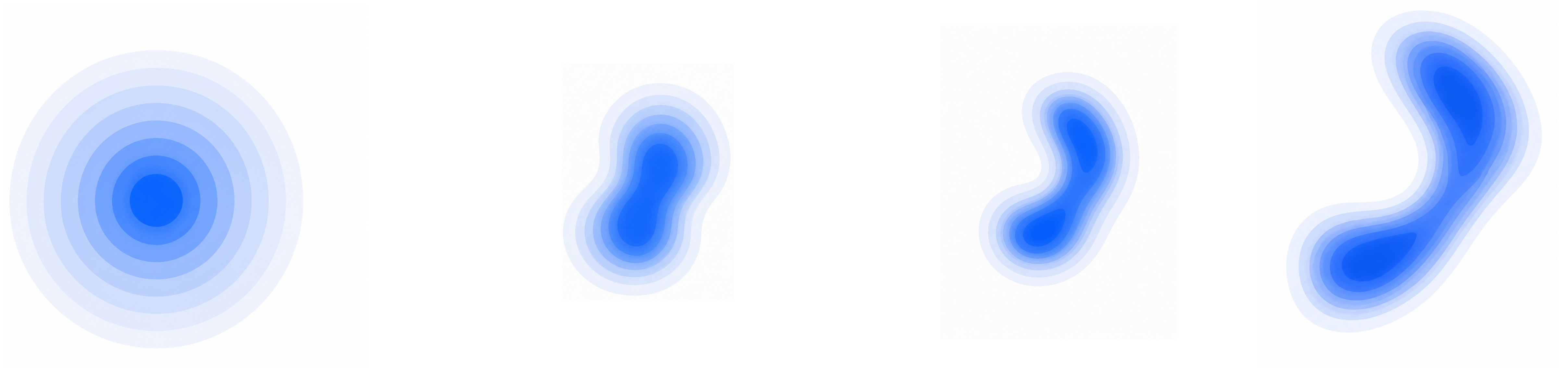
$$p_{data}$$

$$t = 0$$

Forward noising process: from data to noise

$$y_t = s_t(y_0 + \sigma_t w) \quad \text{With } \sigma_0 = 0 \text{ and increasing noise level } \sigma_t \text{ e.g. } \sigma_t = t^\beta$$

Flow-Matching: $s_t = \frac{1}{1 + \sigma_t}$



$$p_{\sigma_T} \approx \mathcal{N}(0, \text{Id})$$

$$t = T \gg 1$$

$$t_2$$

$$t_1$$

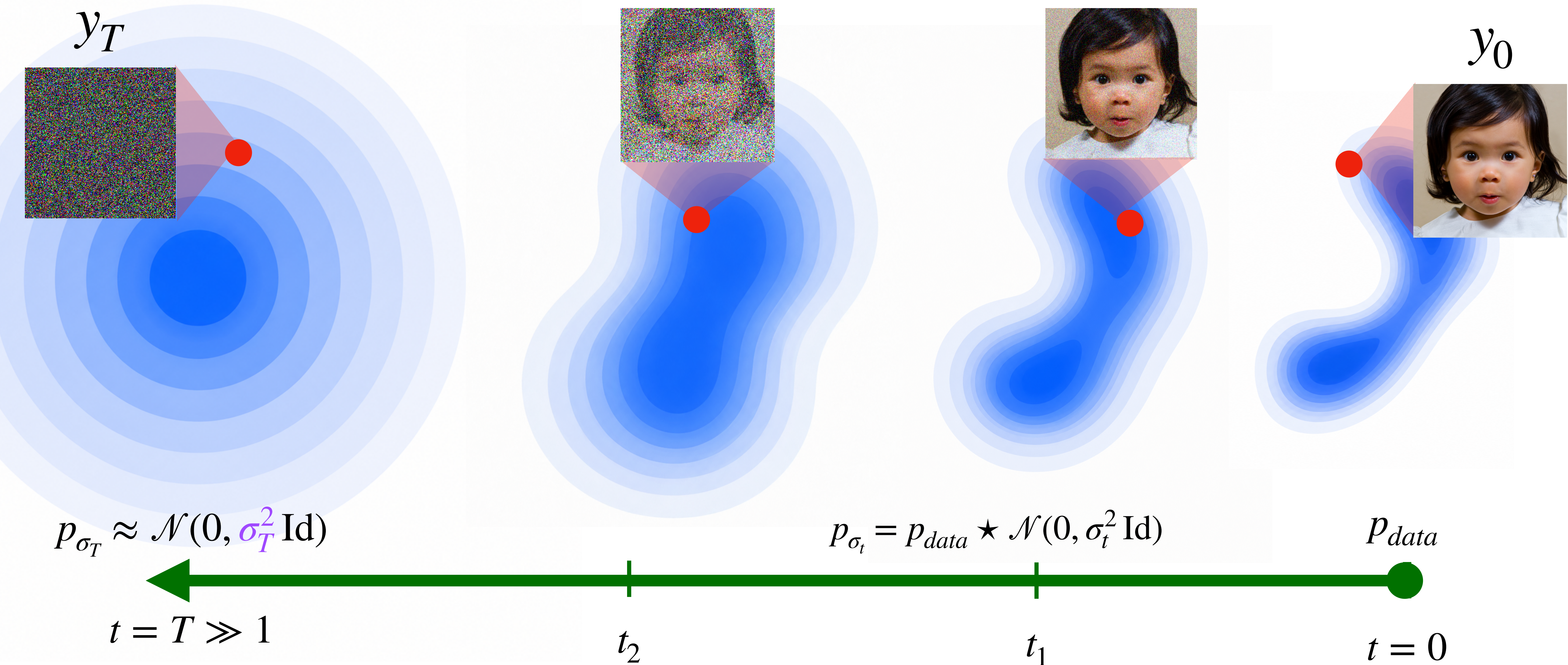
$$p_{data}$$

$$t = 0$$

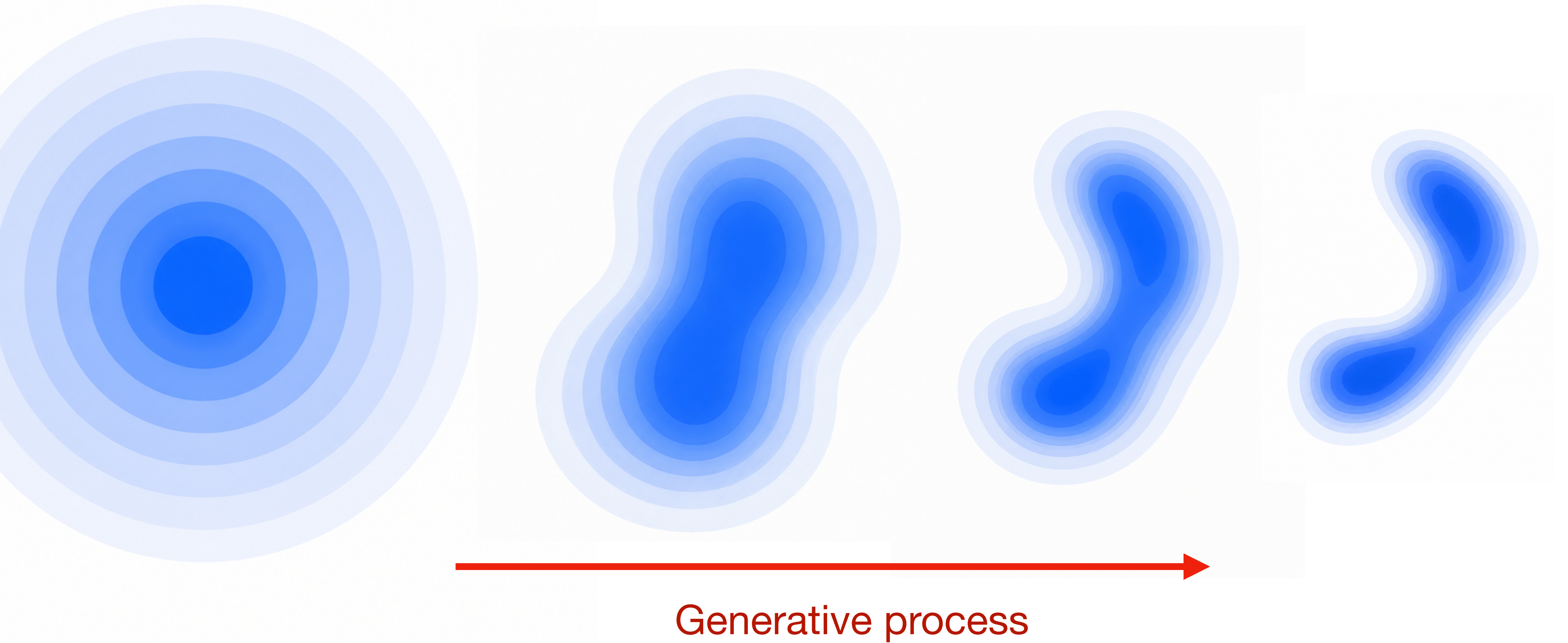
Forward noising process: from data to noise

$$y_t = y_0 + \sigma_t w$$

With $\sigma_0 = 0$ and increasing noise level σ_t e.g. $\sigma_t = t^\beta$



Diffusion models



Generative process: from noise to data

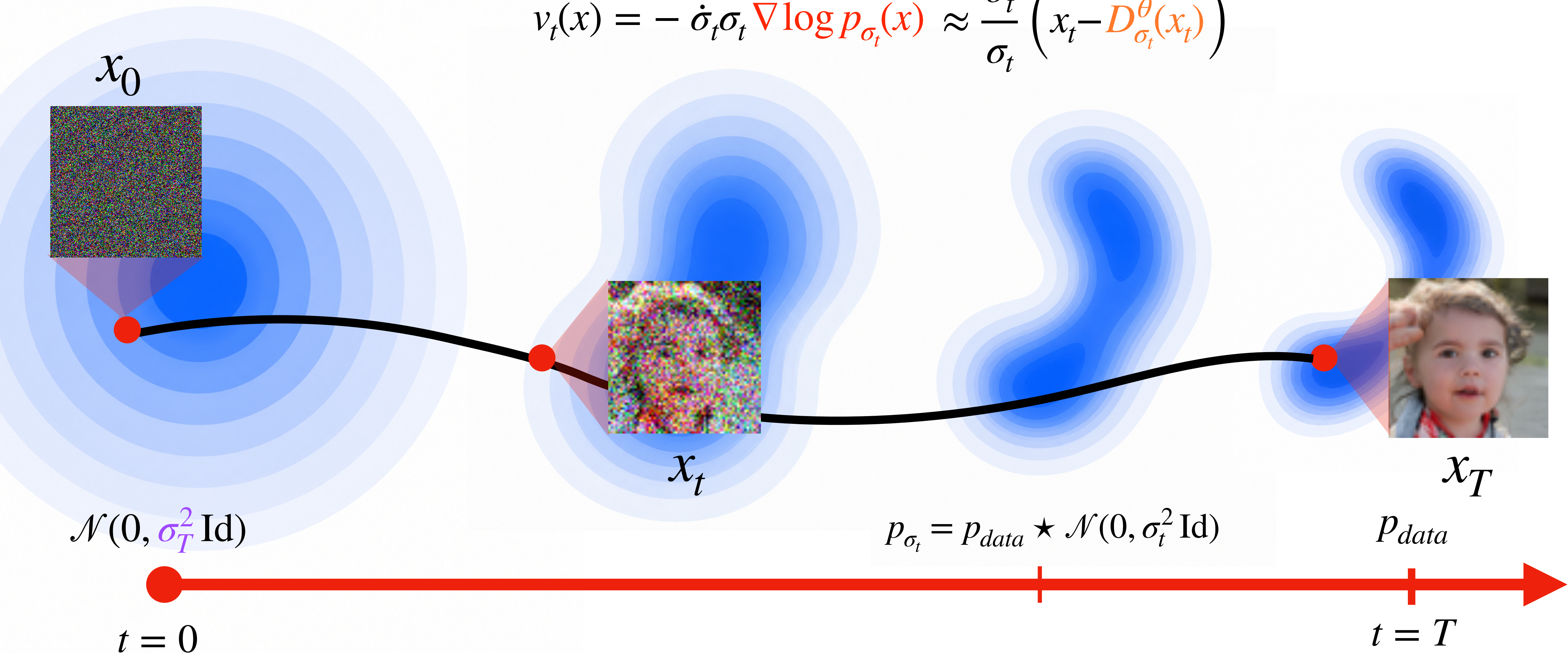


Tweedie formula:

$$D_{\sigma_t}^{\theta}(x_t) \approx x_t + \sigma_t^2 \nabla \log p_{\sigma_t}(x_t)$$

$$x_0 \sim \mathcal{N}(0, \sigma_T^2 \text{Id}), \quad dx_t = v_t(x_t) dt$$

$$v_t(x) = -\dot{\sigma}_t \sigma_t \nabla \log p_{\sigma_t}(x) \approx \frac{\dot{\sigma}_t}{\sigma_t} \left(x_t - D_{\sigma_t}^{\theta}(x_t) \right)$$



Generative process: from noise to data

$$x_0 \sim \mathcal{N}(0, \sigma_T^2 \text{Id}), \quad dx_t = v_t(x_t) dt$$

$$v_t(x) = -\dot{\sigma}_t \sigma_t \nabla \log p_{\sigma_t}(x) \approx \frac{\dot{\sigma}_t}{\sigma_t} \left(x_t - D_{\sigma_t}^{\theta}(x_t) \right)$$

In practice, the differential equation is **discretized** in time:

Euler discretization: $x_{k+1} - x_k = v_{t_k}(x_{t_k})(t_{k+1} - t_k)$

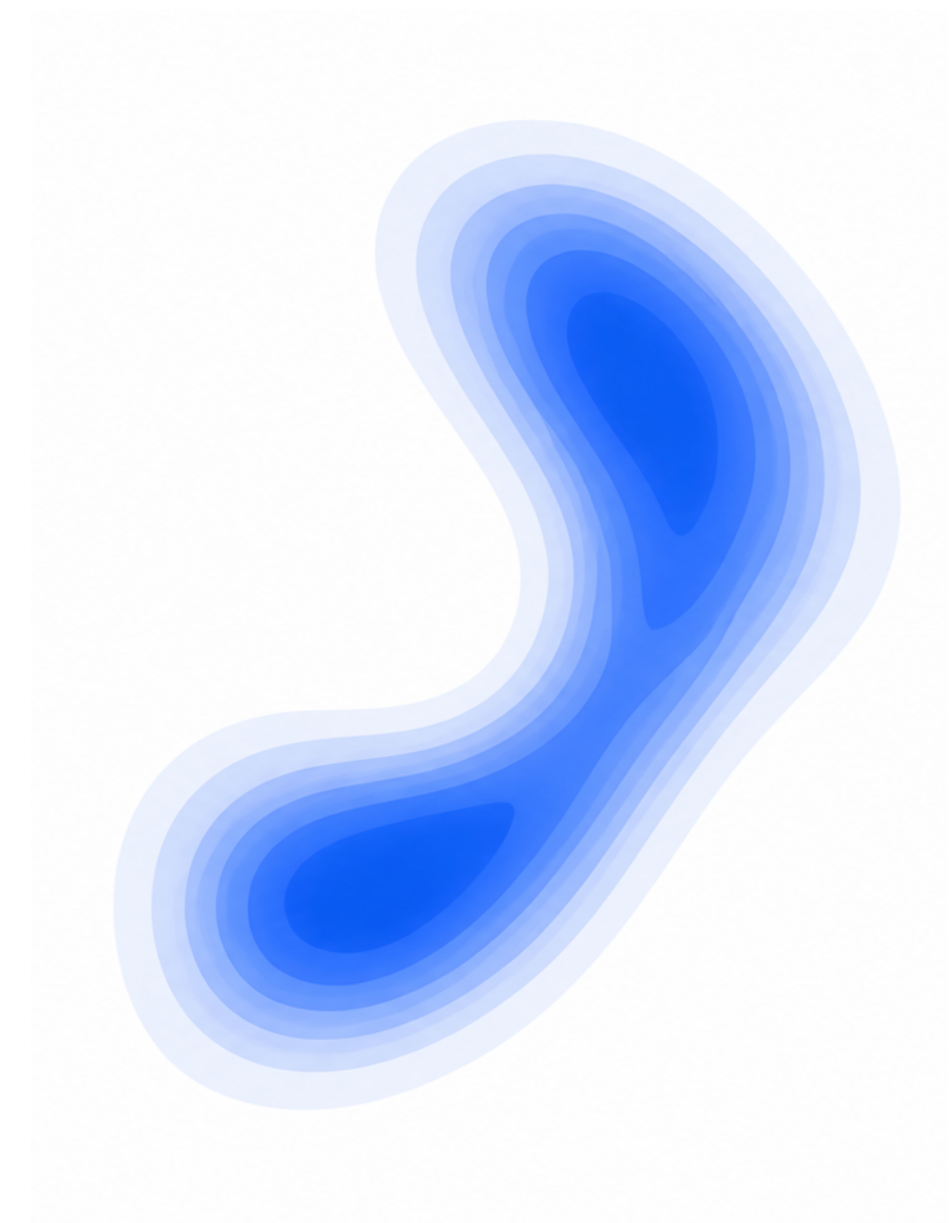
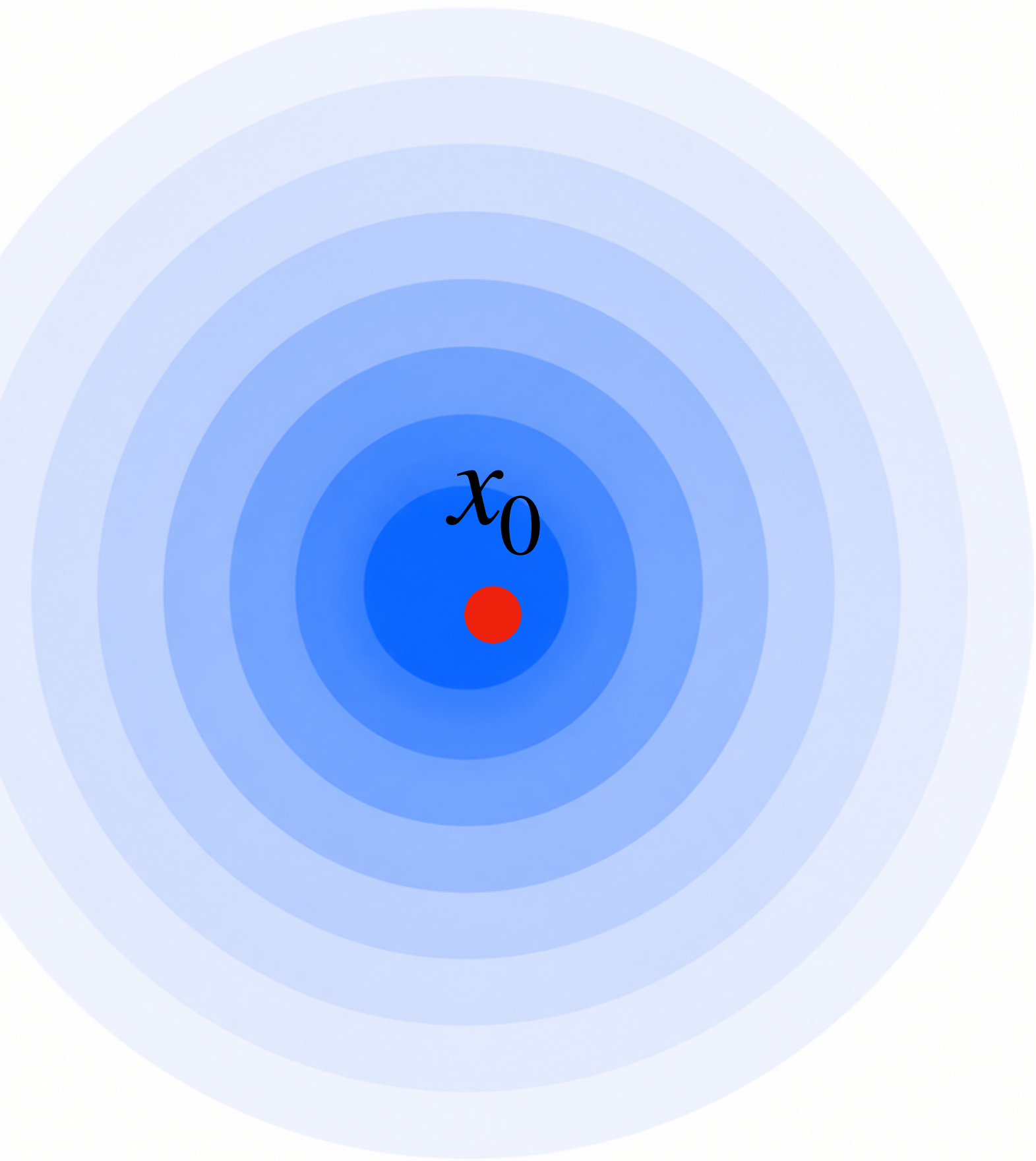
$$\Leftrightarrow x_{k+1} = (1 - \beta_k)x_k + \beta_k D_{\sigma_{t_k}}^{\theta}(x_k)$$

$$\beta_k := -\delta t_k \frac{\dot{\sigma}_{t_k}}{\sigma_{t_k}}$$

Diffusion sampling: progressive denoising

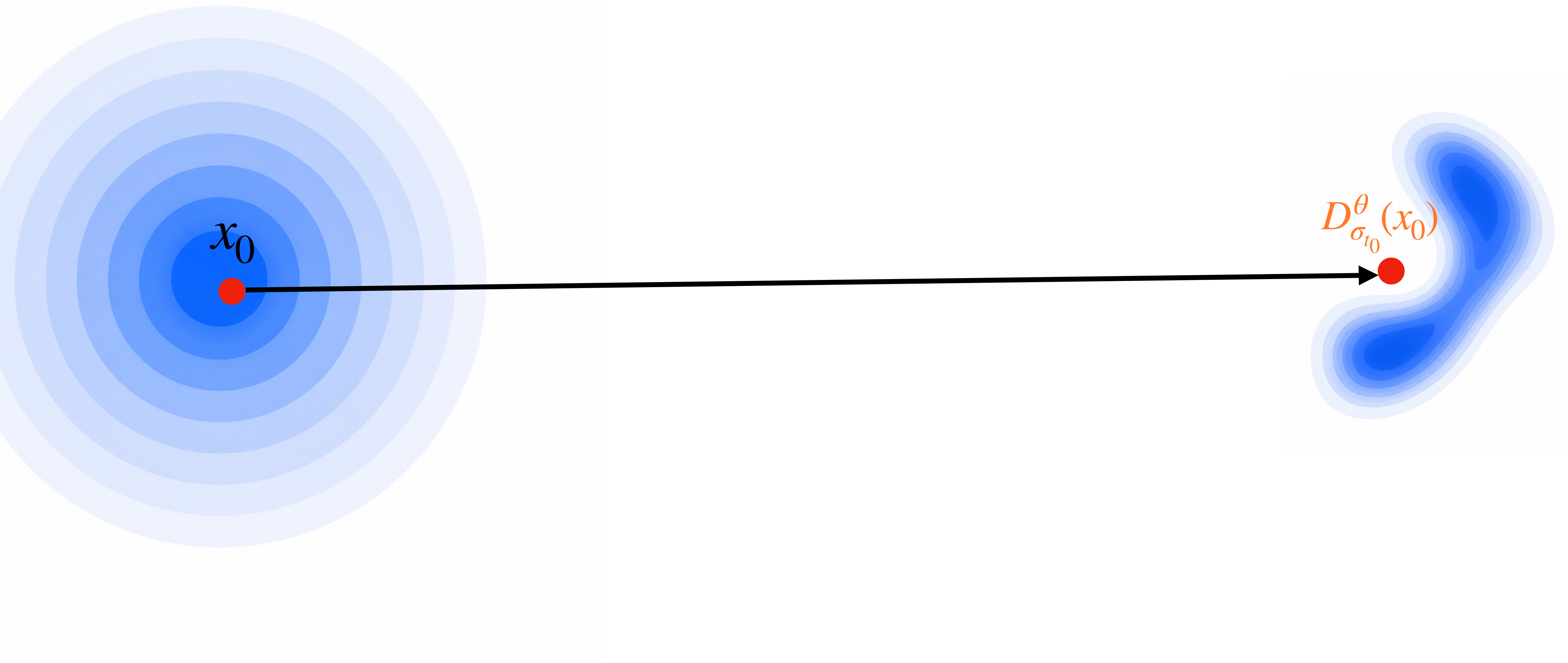
$$x_0 \sim \mathcal{N}(0, \sigma_T^2 \text{Id})$$

$$x_{k+1} = (1 - \beta_k)x_k + \beta_k D_{\sigma_{t_k}}^{\theta}(x_k)$$



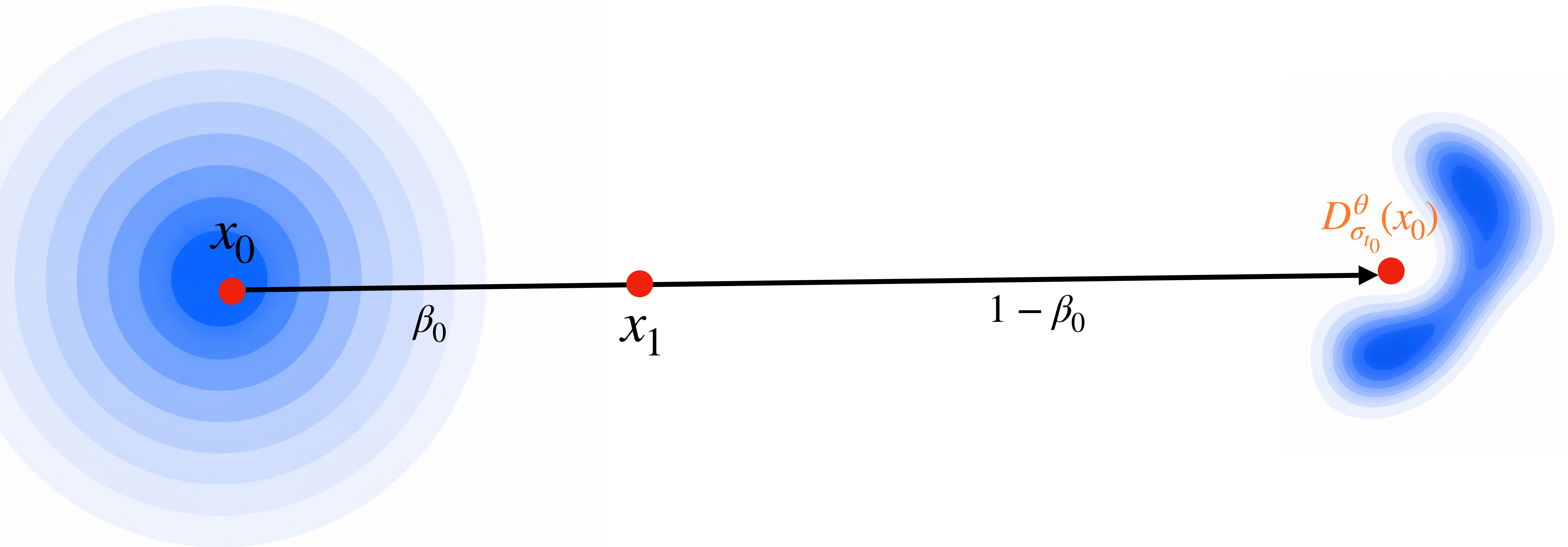
Diffusion sampling: progressive denoising

$$x_{k+1} = (1 - \beta_k)x_k + \beta_k D_{\sigma_{t_k}}^{\theta}(x_k)$$



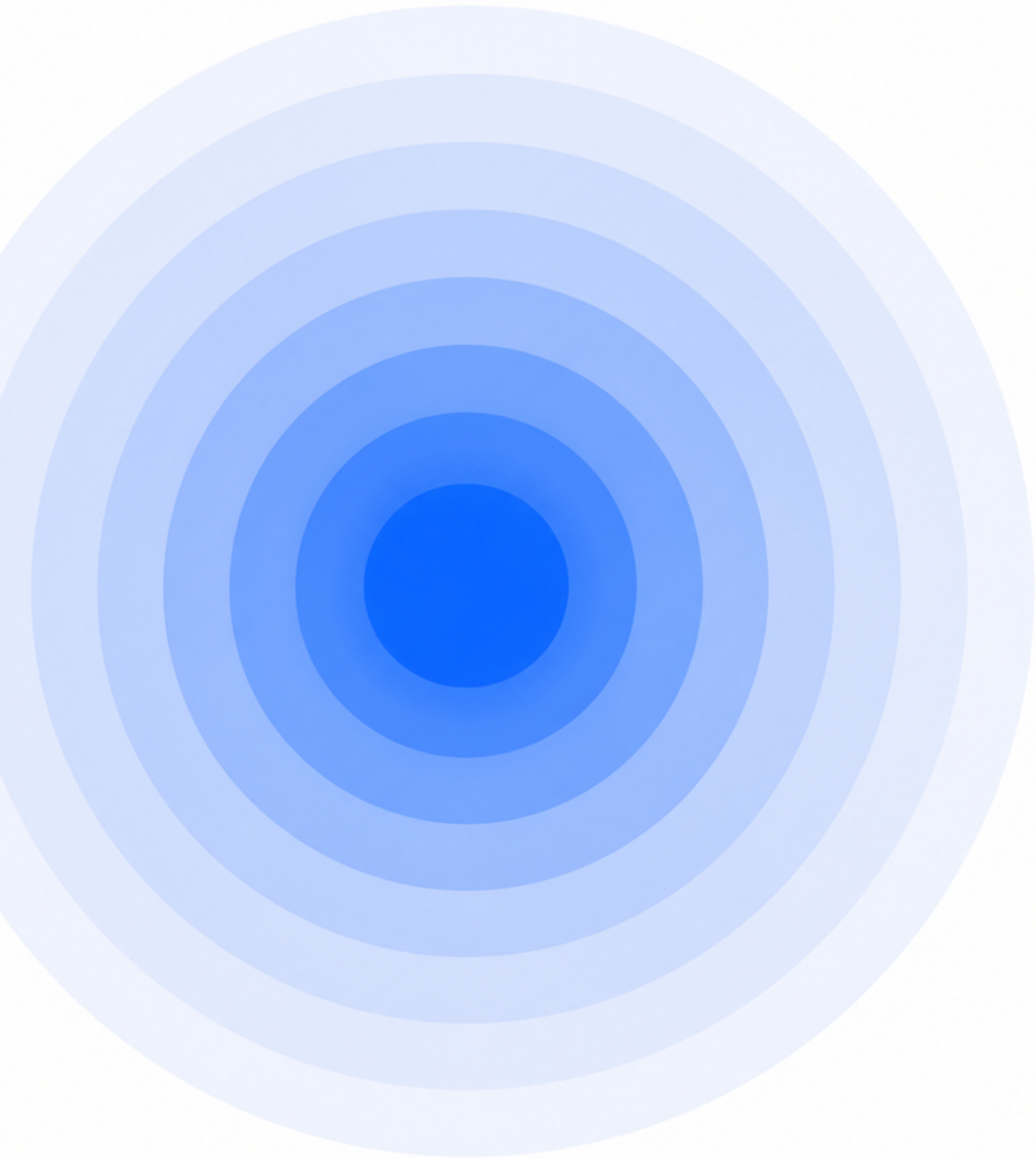
Diffusion sampling: progressive denoising

$$x_{k+1} = (1 - \beta_k)x_k + \beta_k D_{\sigma_{t_k}}^{\theta}(x_k)$$

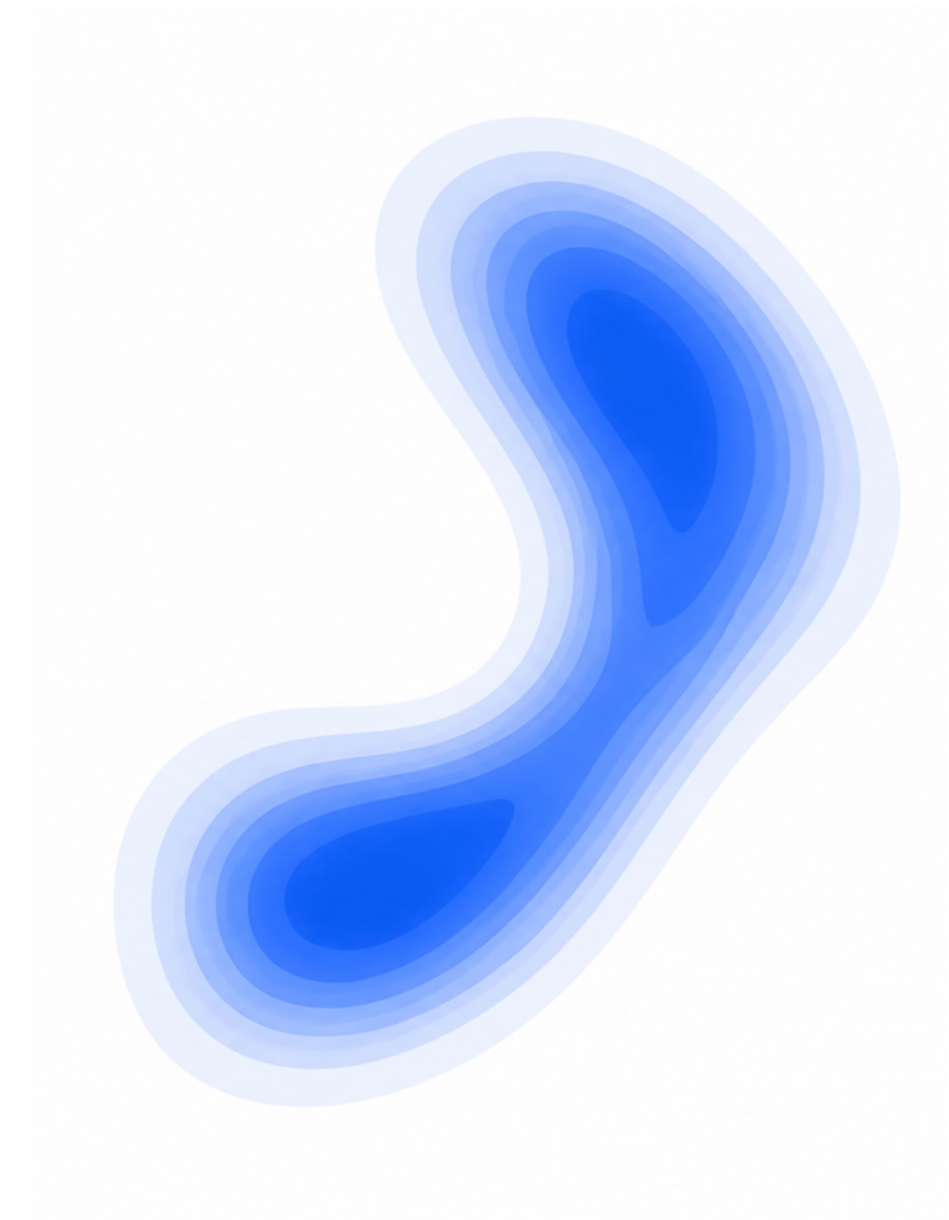


Diffusion sampling: progressive denoising

$$x_{k+1} = (1 - \beta_k)x_k + \beta_k D_{\sigma_{t_k}}^{\theta}(x_k)$$

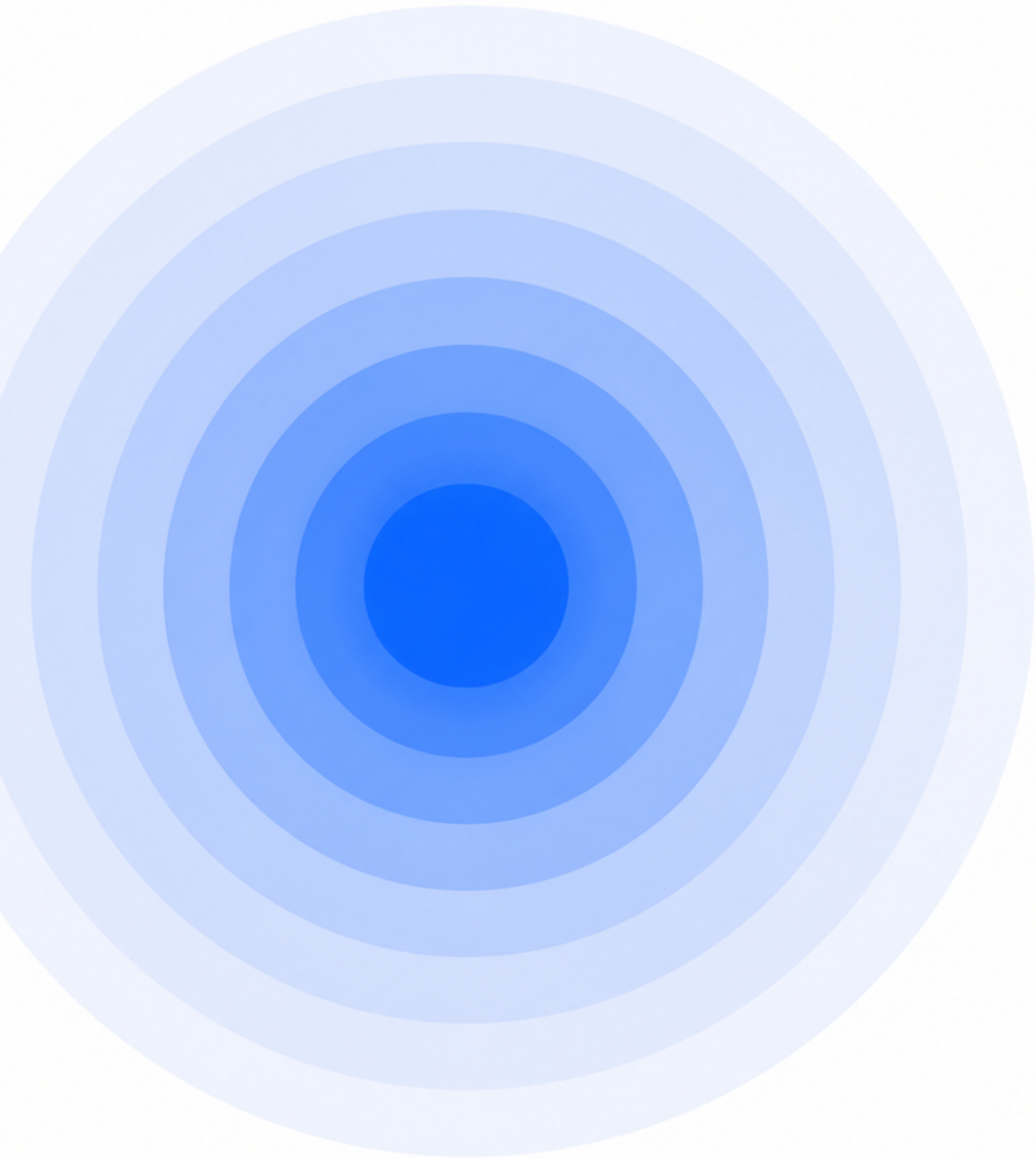



 x_1

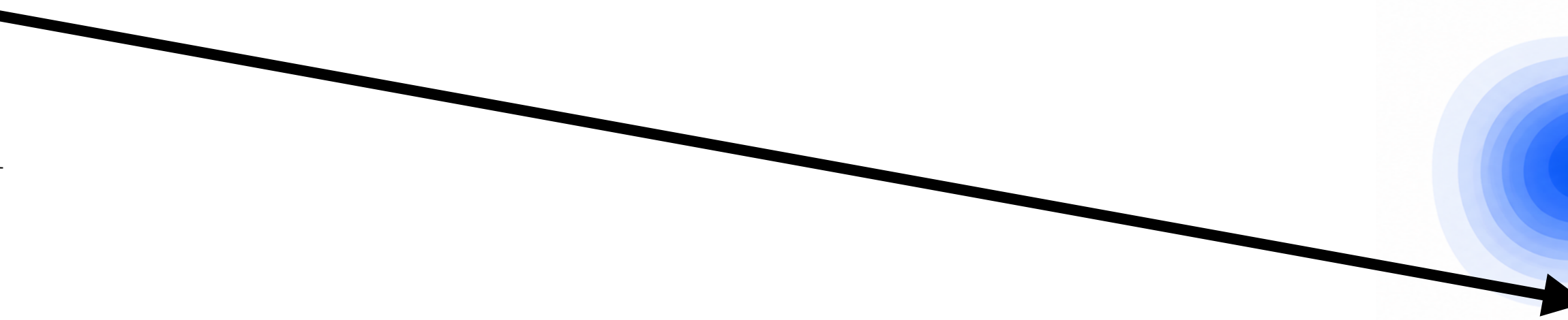


Diffusion sampling: progressive denoising

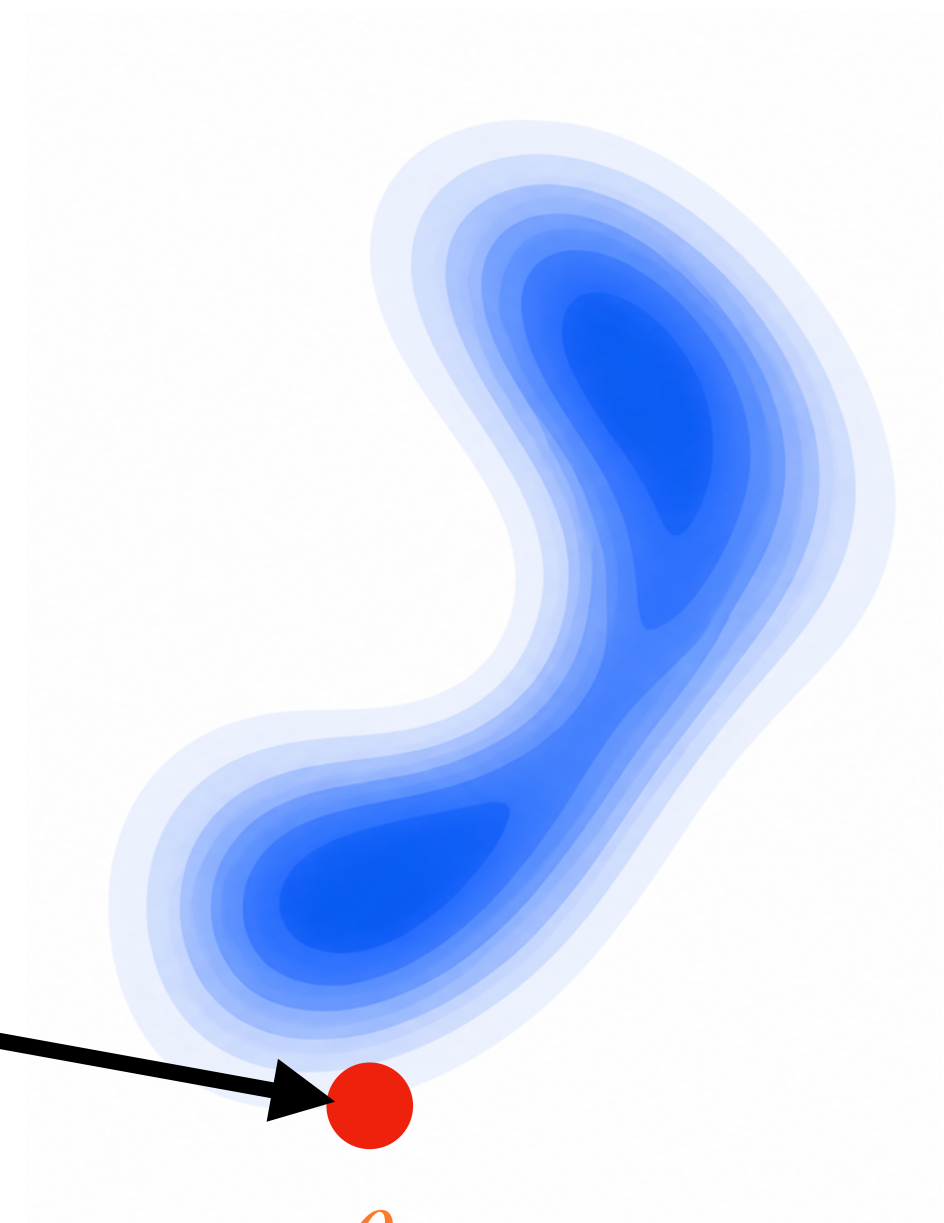
$$x_{k+1} = (1 - \beta_k)x_k + \beta_k D_{\sigma_{t_k}}^{\theta}(x_k)$$



x_1

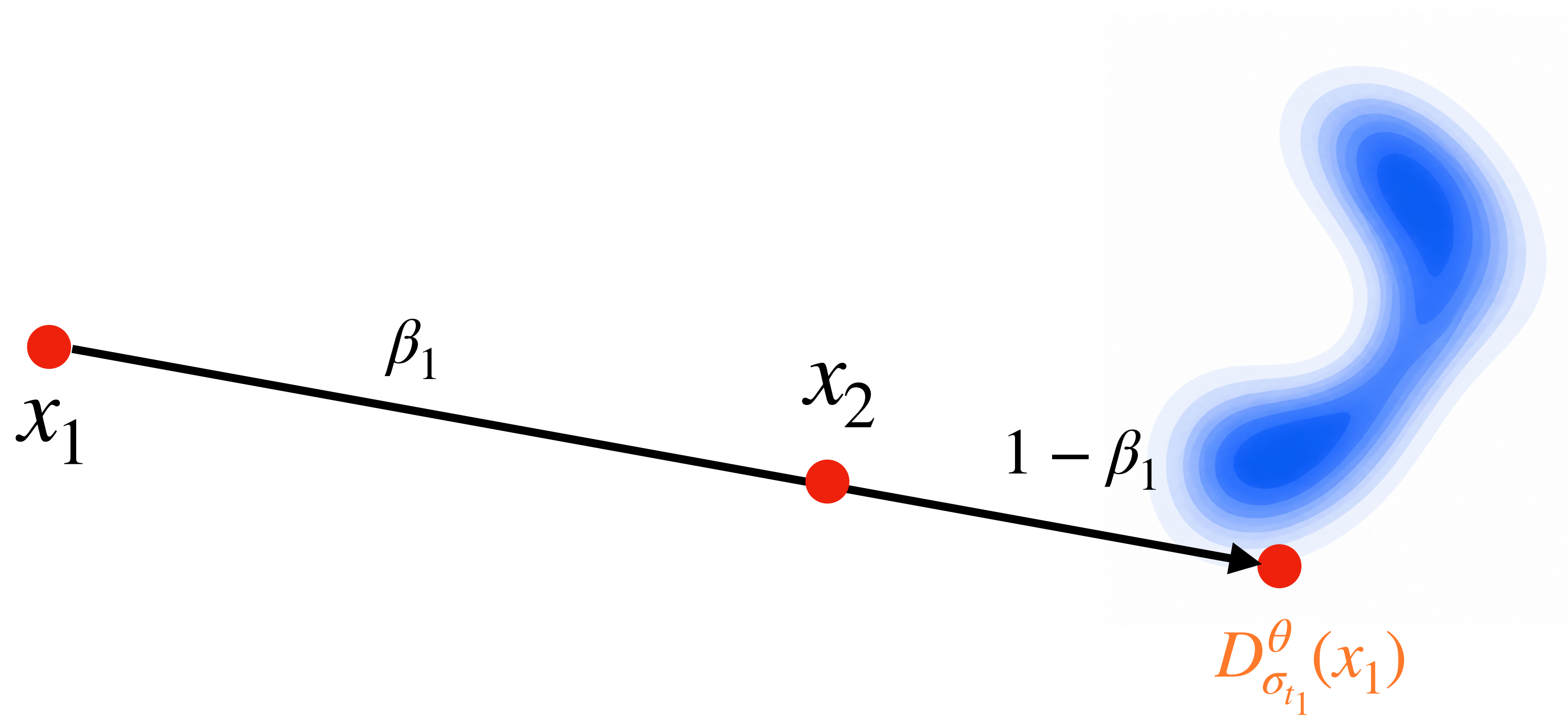
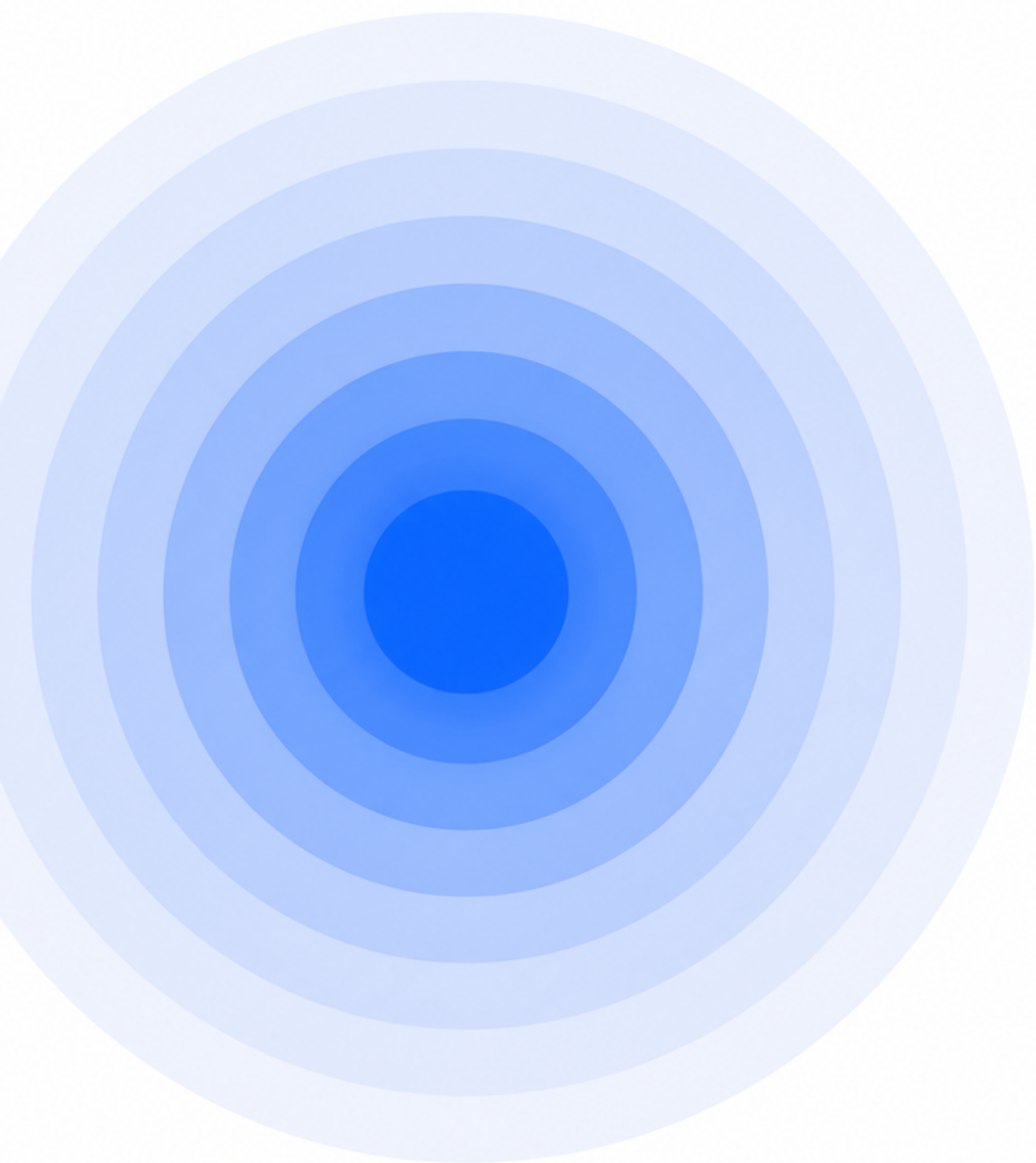


$D_{\sigma_{t_1}}^{\theta}(x_1)$



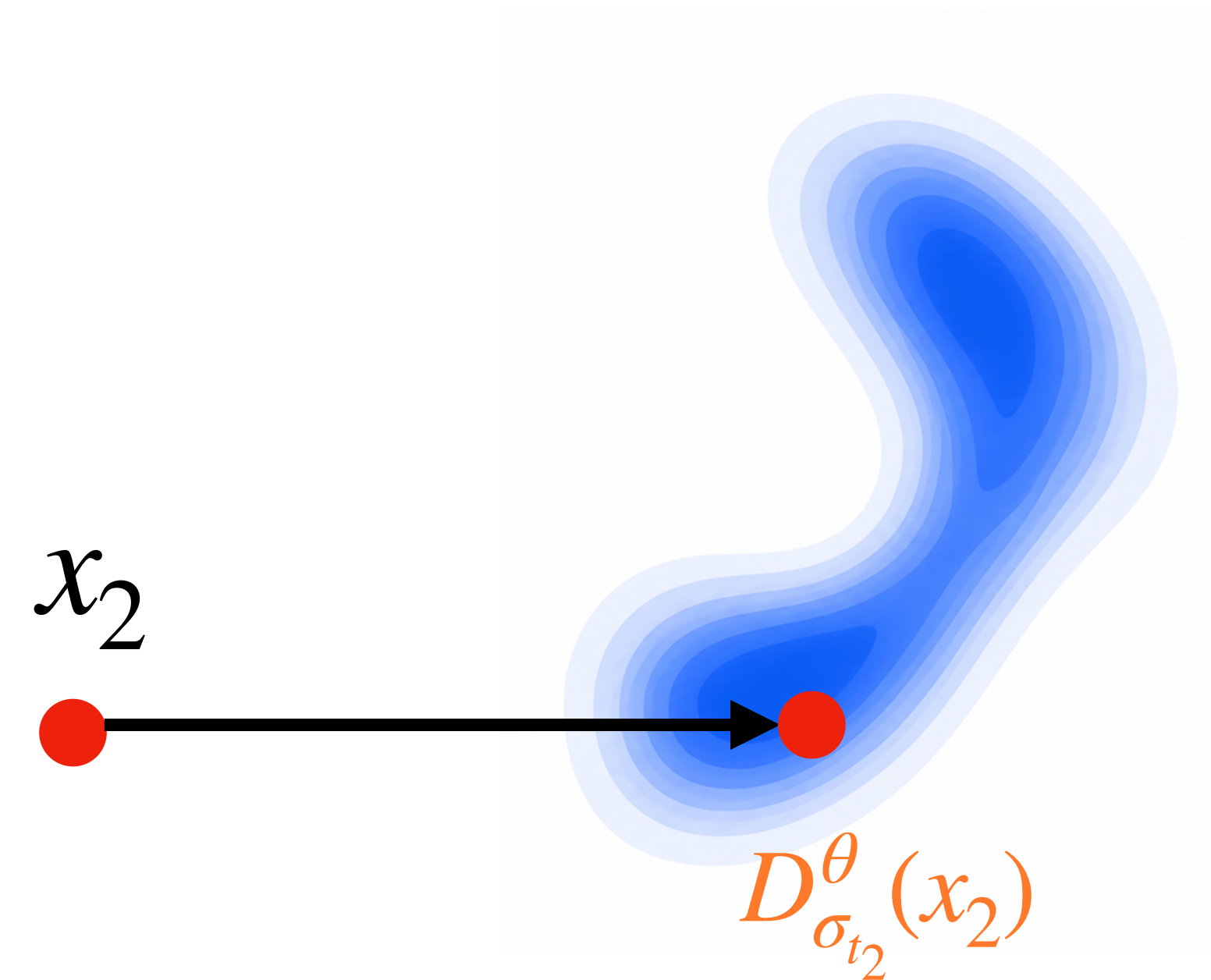
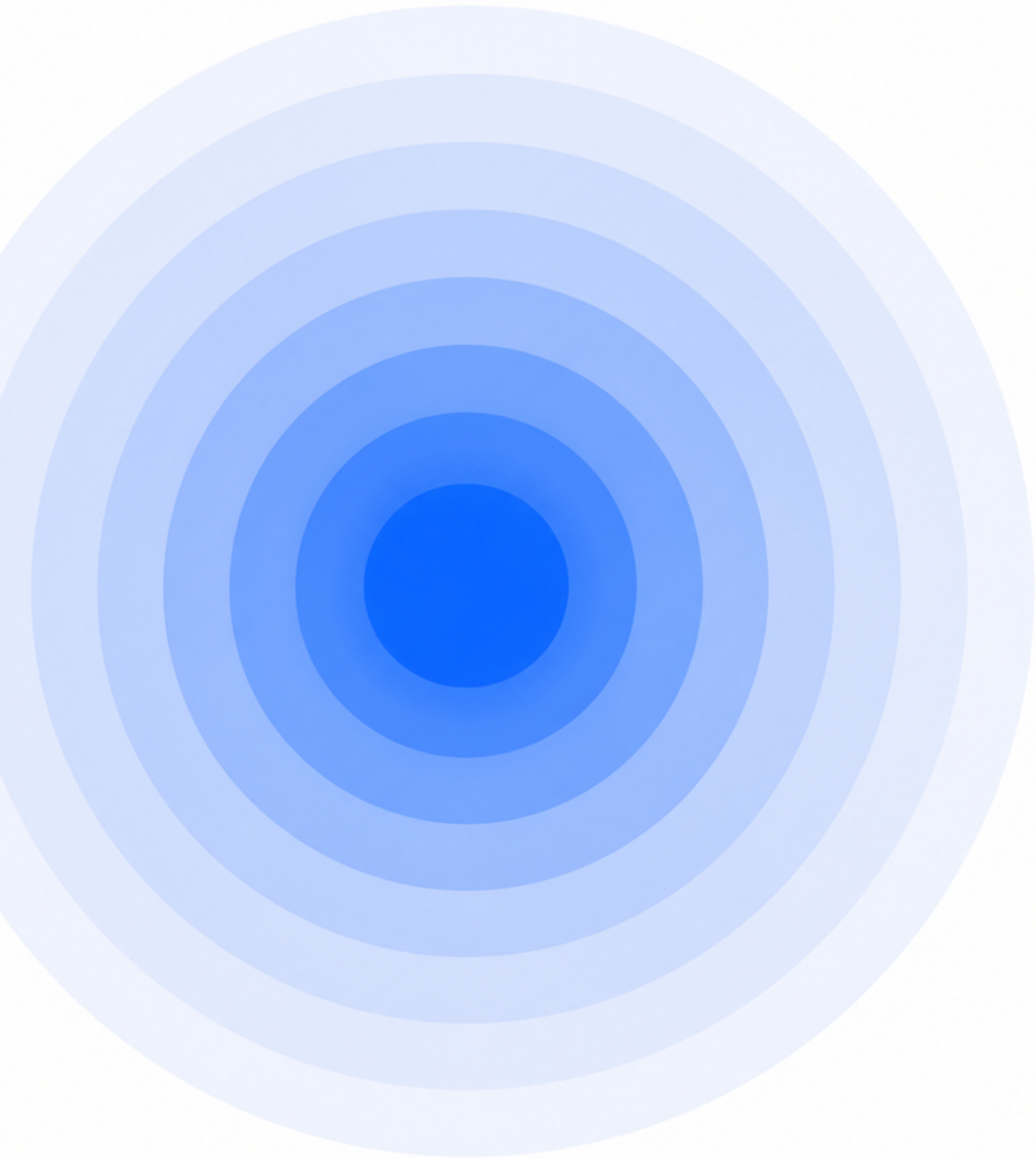
Diffusion sampling: progressive denoising

$$x_{k+1} = (1 - \beta_k)x_k + \beta_k D_{\sigma_{t_k}}^{\theta}(x_k)$$



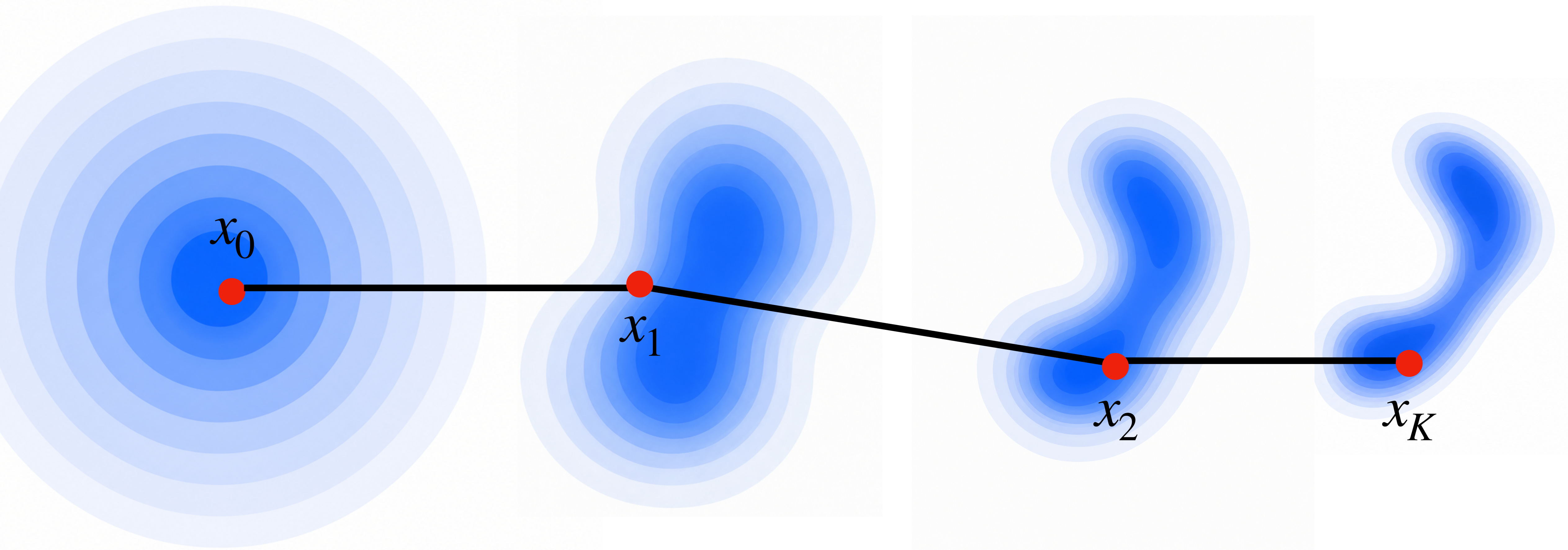
Diffusion sampling: progressive denoising

$$x_{k+1} = (1 - \beta_k)x_k + \beta_k D_{\sigma_{t_k}}^{\theta}(x_k)$$

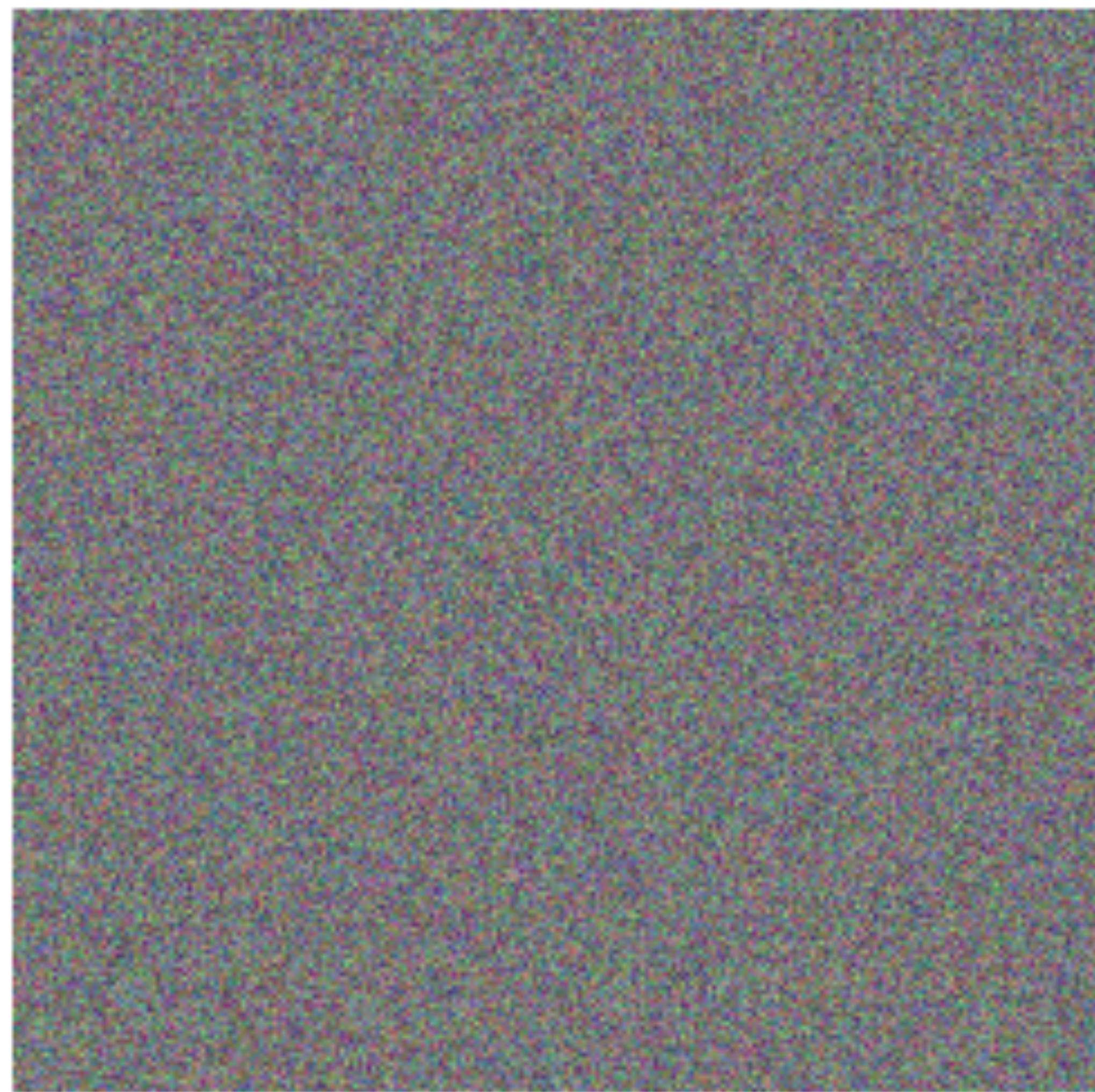


Diffusion sampling: progressive denoising

$$x_{k+1} = (1 - \beta_k)x_k + \beta_k D_{\sigma_{t_k}}^{\theta}(x_k)$$



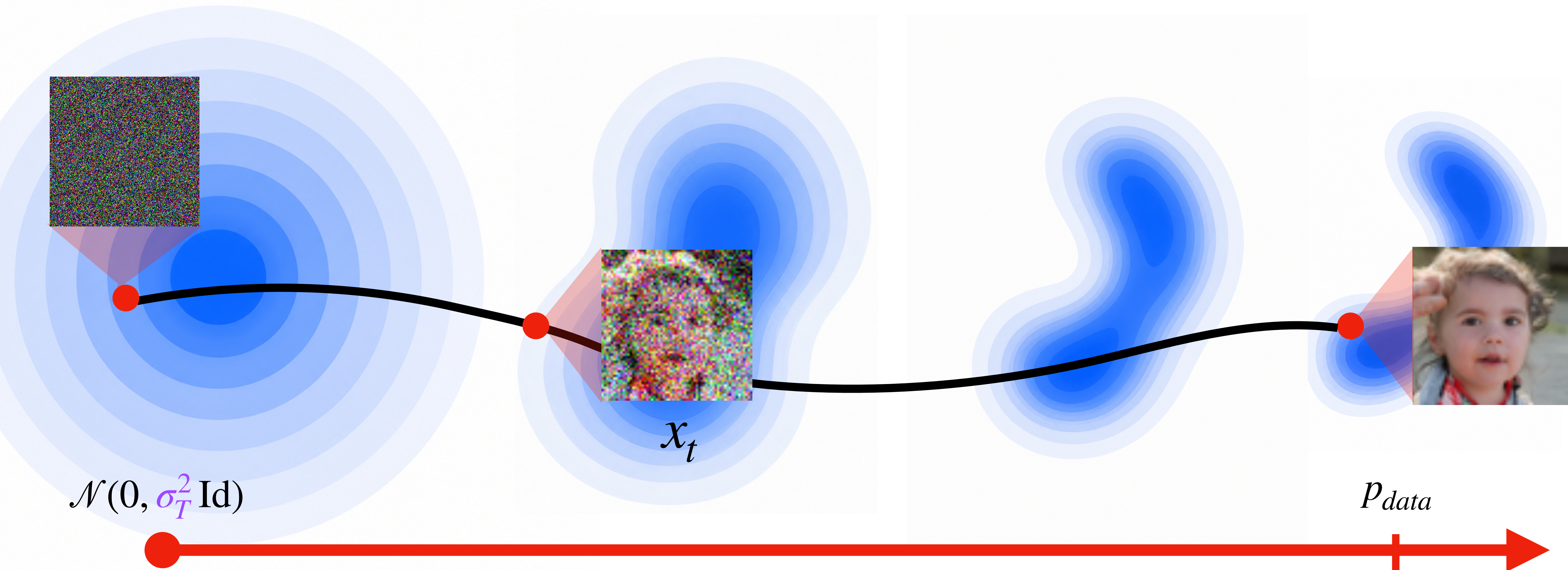




Various diffusion sampling algorithms

$$x_0 \sim \mathcal{N}(0, \sigma_T^2 \text{Id}), \quad dx_t = v_t(x_t) dt$$

$$v_t = \frac{\dot{\sigma}_t}{\sigma_t} \left(x_t - D_{\sigma_t}^*(x_t) \right)$$



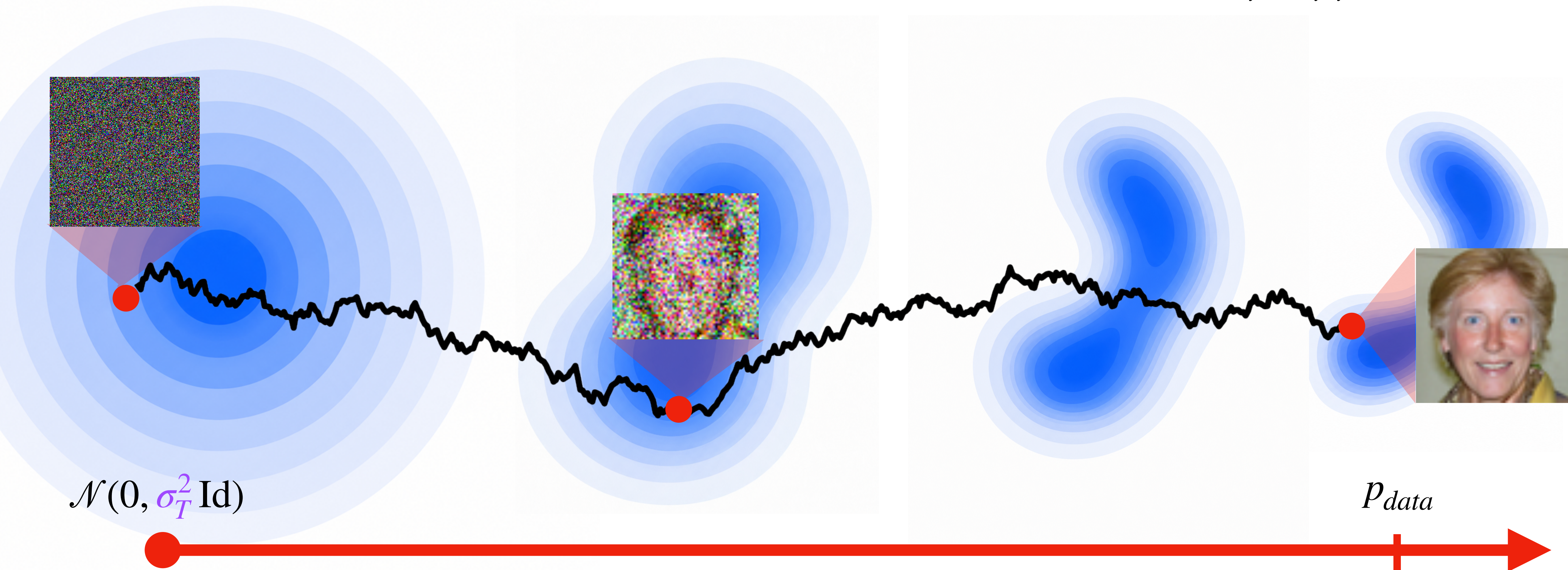
Various diffusion sampling algorithms: **stochasticity**

For any $\alpha \geq 0$

$$x_0 \sim \mathcal{N}(0, \sigma_T^2 \text{Id}), \quad dx_t = v_t(x_t) dt + \sqrt{2\alpha a_t} dW_t$$

$$v_t = \frac{\dot{\sigma}_t}{\sigma_t} \left(x_t - D_{\sigma_t}^*(x_t) \right)$$

$$a_t = \dot{\sigma}_t \sigma_t$$

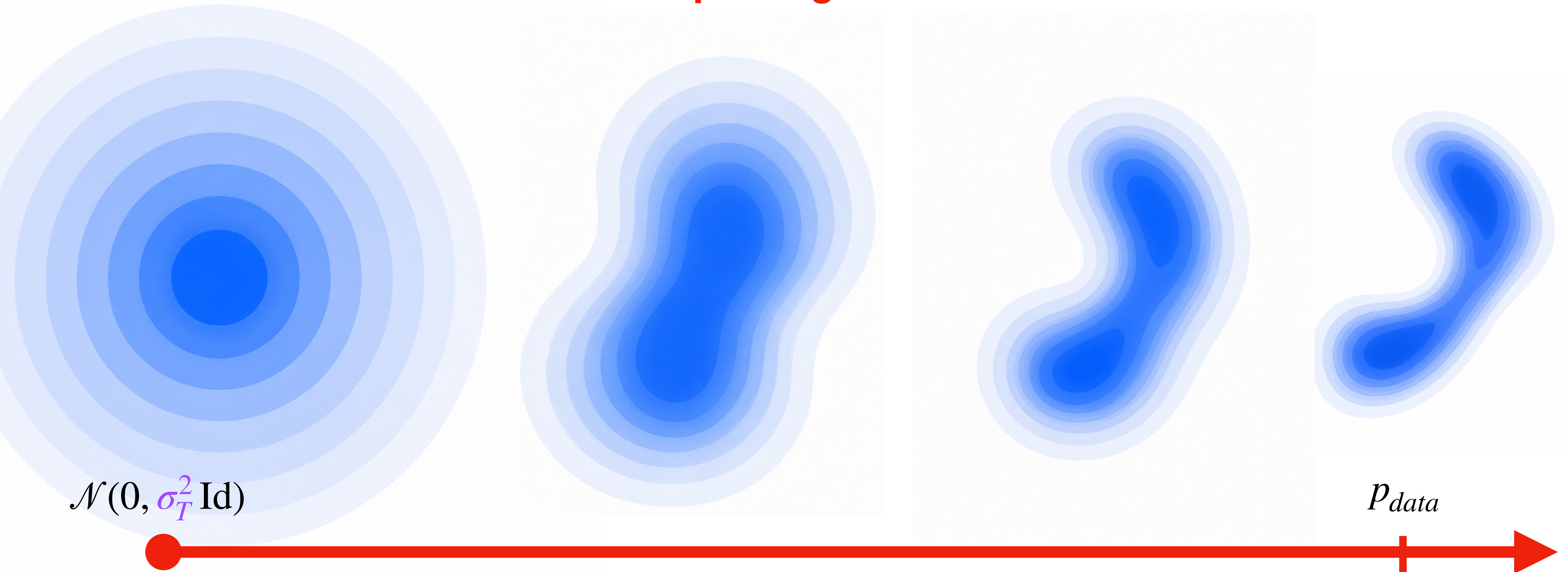


Various diffusion sampling algorithms

$$x_0 \sim \mathcal{N}(0, \sigma_T^2 \text{Id}), \quad dx_t = v_t(x_t) dt$$

$$v_t = \frac{\dot{\sigma}_t}{\sigma_t} \left(x_t - D_{\sigma_t}^*(x_t) \right)$$

« Variance Exploding »



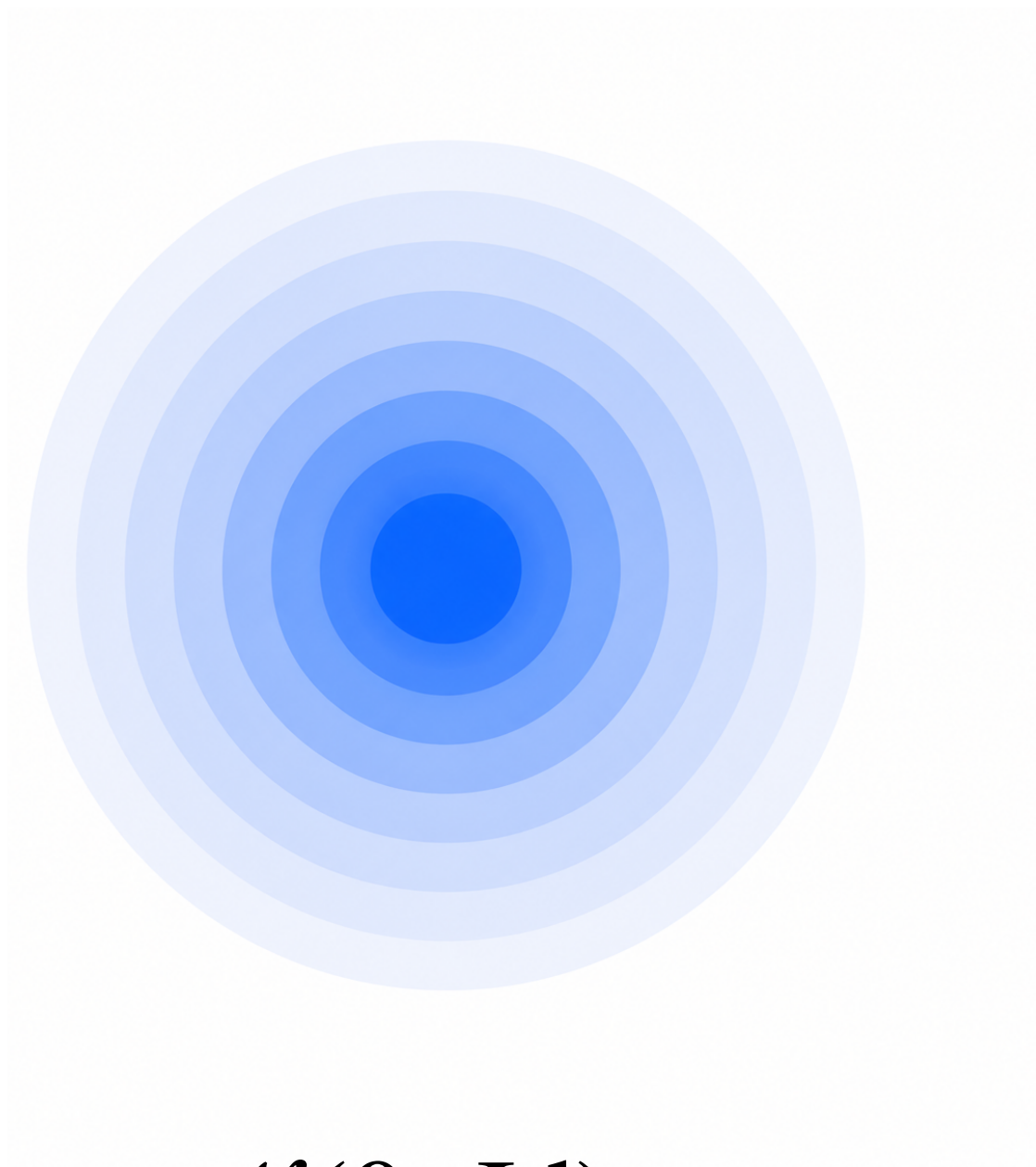
Various diffusion sampling algorithms

$$x_0 \sim \mathcal{N}(0, \sigma_T^2 \text{Id}), \quad dx_t = v_t(x_t) dt$$

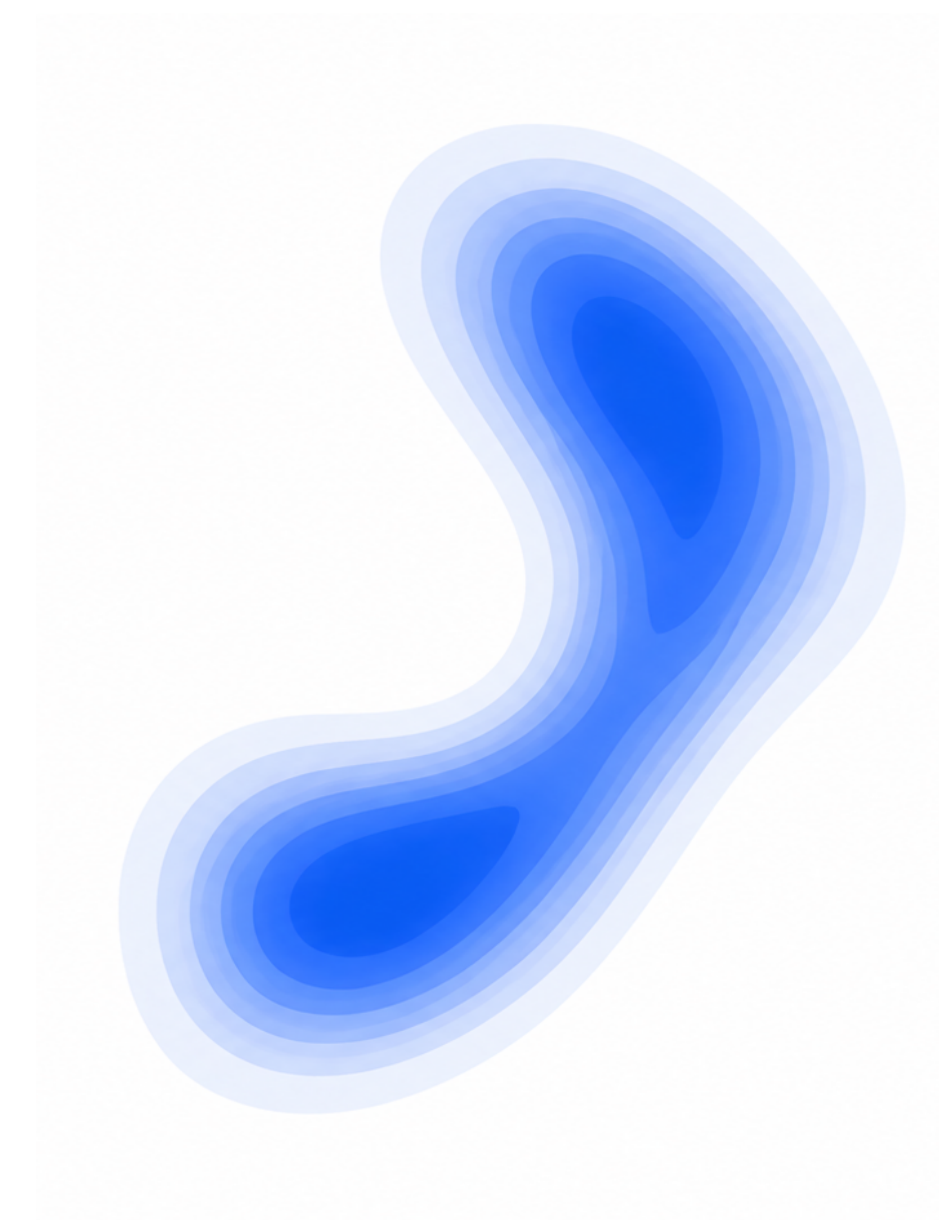
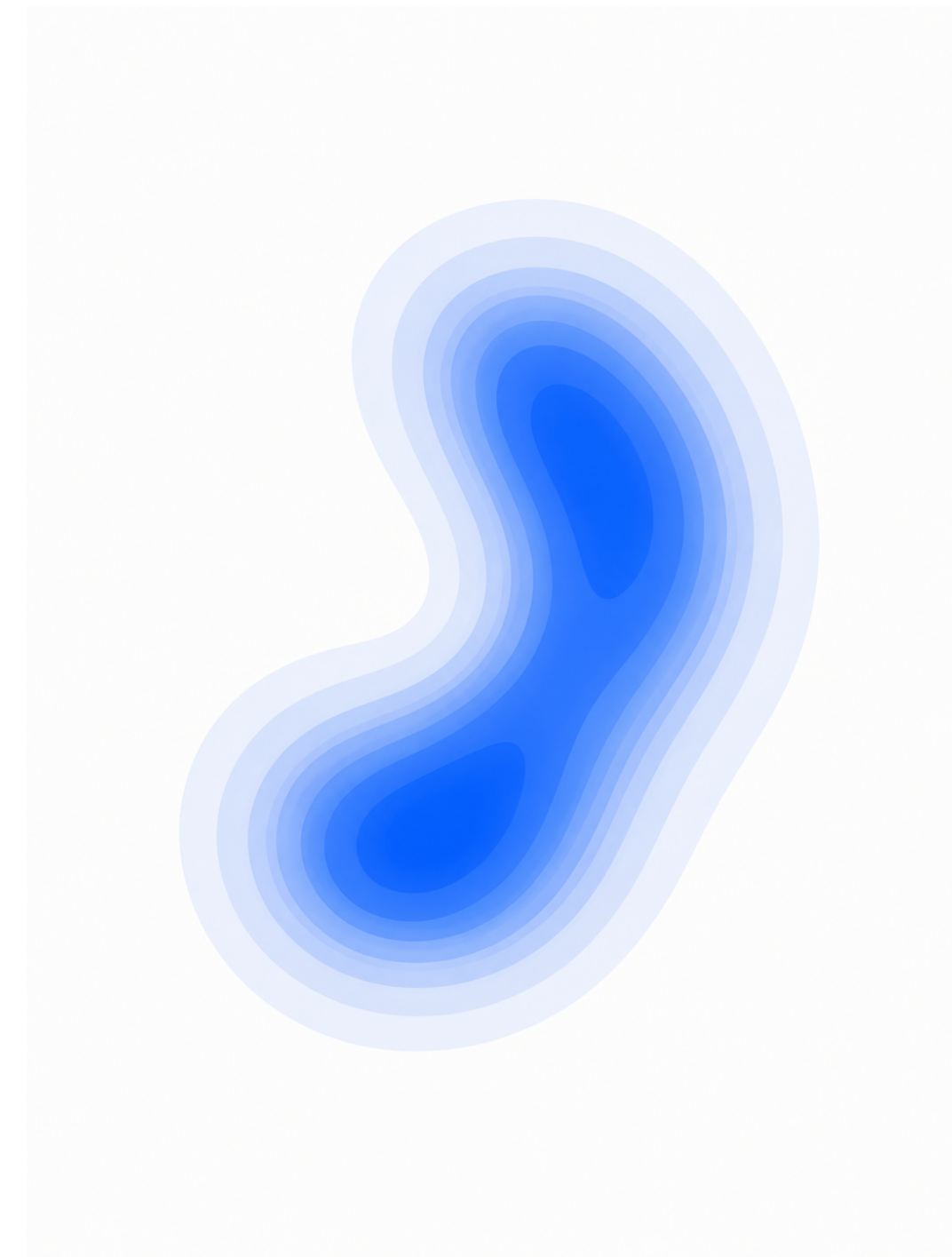
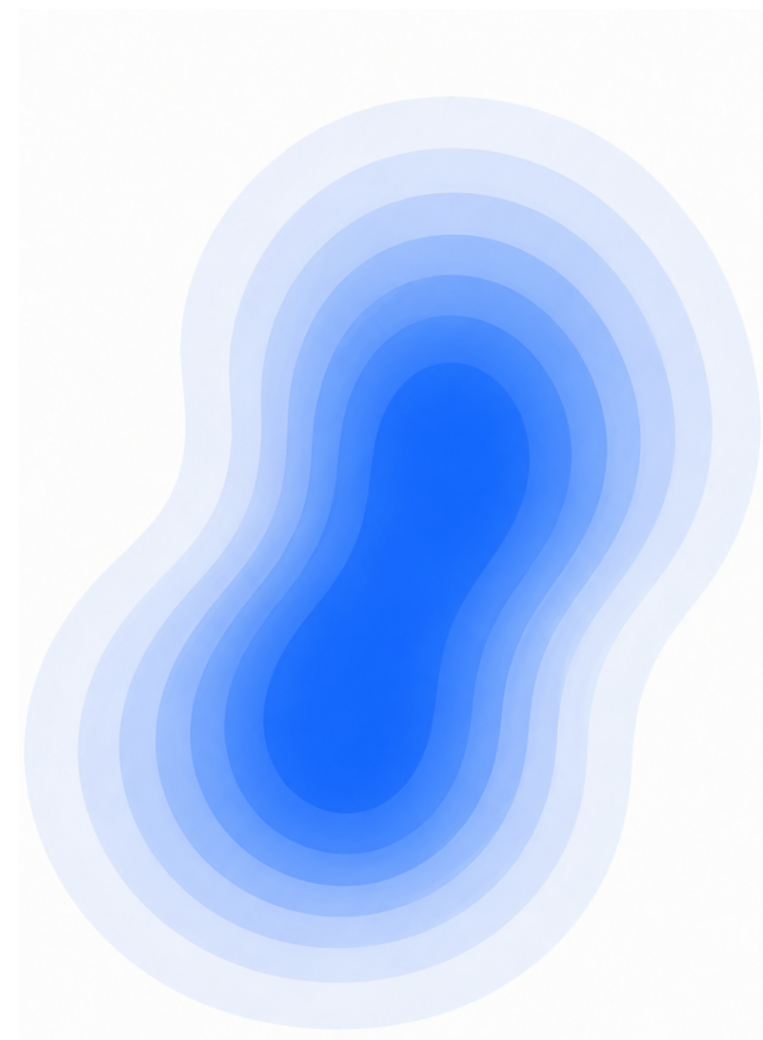
« Variance Preserving »

$$v_t(x_t) = \frac{\dot{s}_t}{s_t} x_t + \frac{\dot{\sigma}_t}{\sigma_t} \left[x_t - s_t D_{\sigma_t}^* \left(\frac{x_t}{s_t} \right) \right]$$

With a rescaling schedule s_t



$\mathcal{N}(0, \text{Id})$



p_{data}



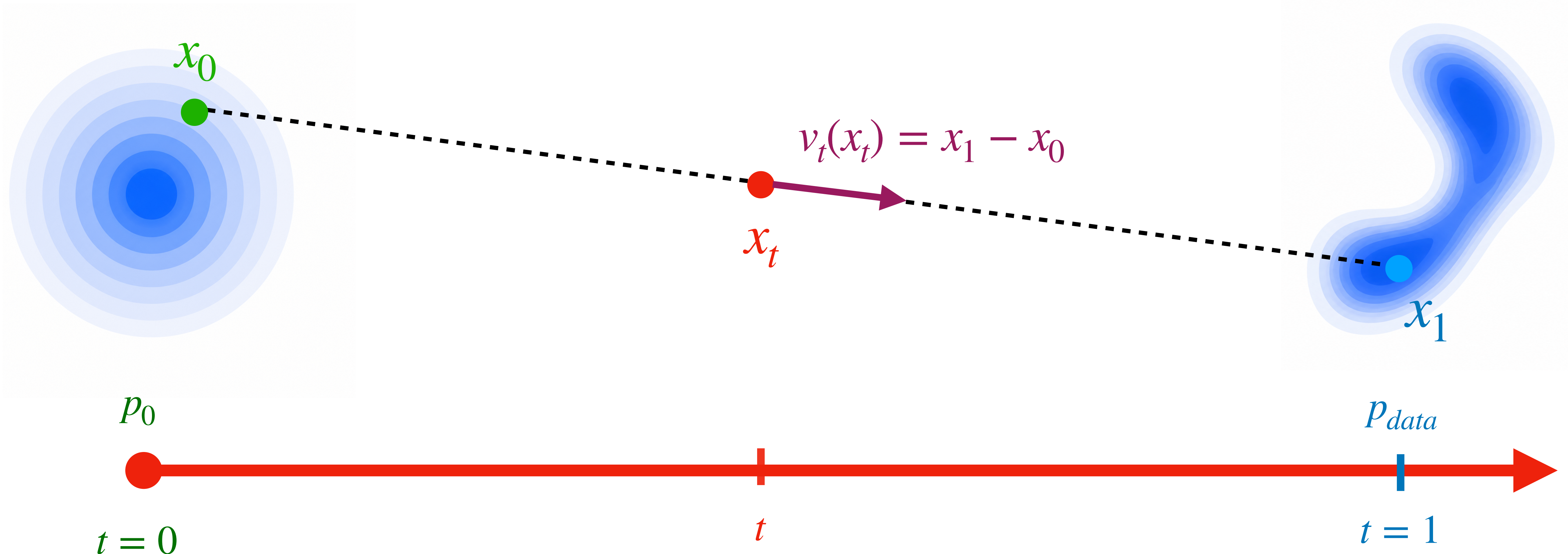
Flow Matching

Objective: Find a velocity field v_t such that...

$$x_0 \sim p_0, \quad dx_t = v_t(x_t) dt$$

interpolates between p_0 and $p_1 = p_{data}$

- Take $x_0 \sim p_0$ and $x_1 \sim p_{data}$ (not necessarily independent)
- Define $x_t = (1 - t)x_0 + tx_1$



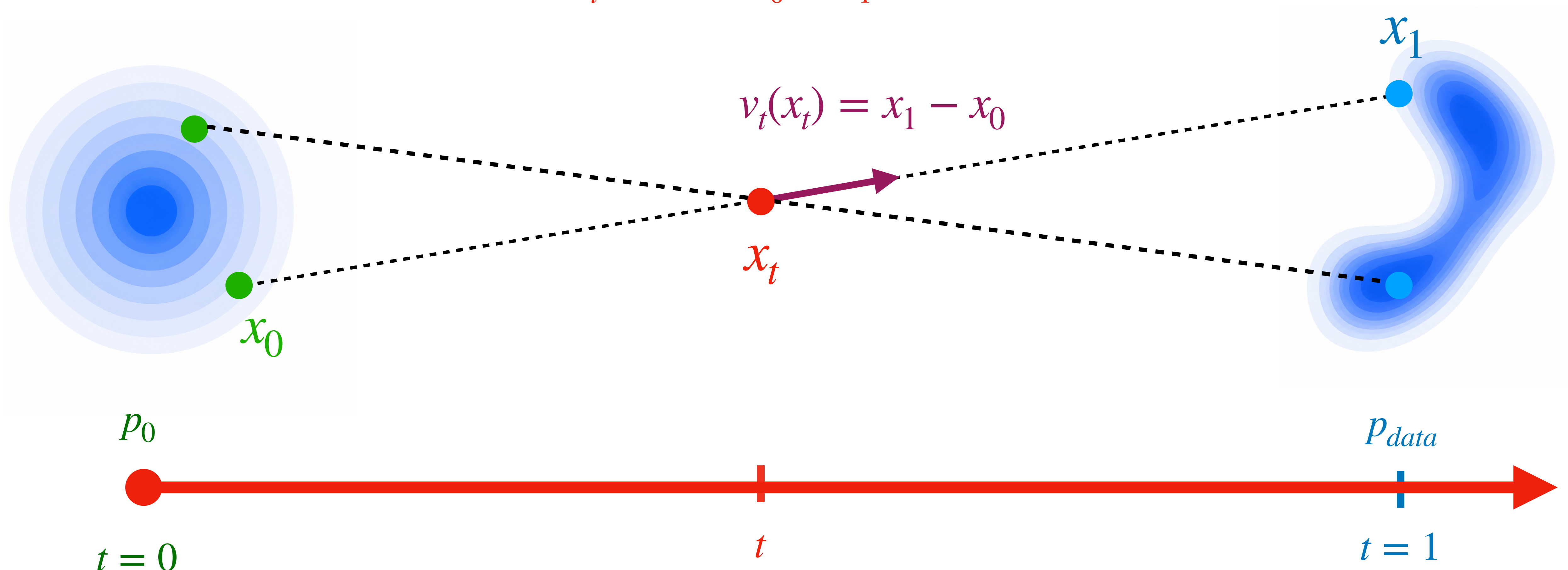
Flow Matching

Objective: Find a velocity field v_t such that...

$$x_0 \sim p_0, \quad dx_t = v_t(x_t) dt$$

interpolates between p_0 and $p_1 = p_{data}$

- Take $x_0 \sim p_0$ and $x_1 \sim p_{data}$ (not necessarily independent)
- Define $x_t = (1 - t)x_0 + tx_1$



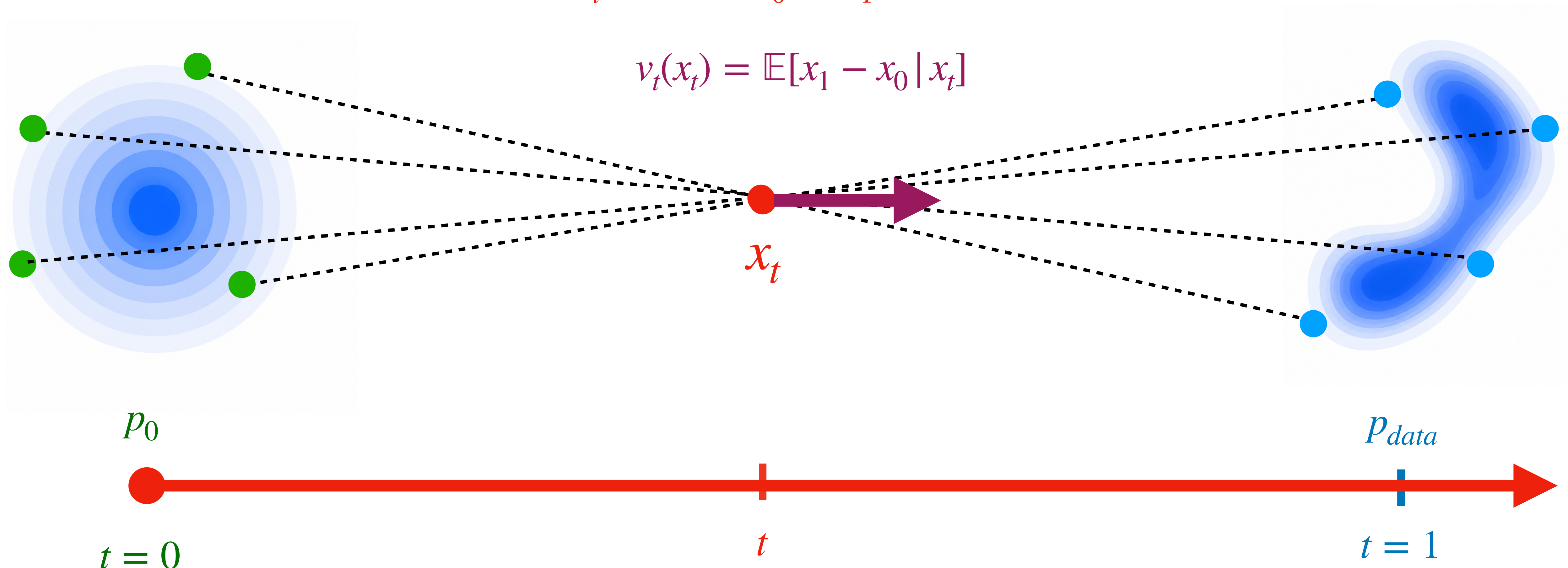
Flow Matching

Objective: Find a velocity field v_t such that...

$$x_0 \sim p_0, \quad dx_t = v_t(x_t) dt$$

interpolates between p_0 and $p_1 = p_{data}$

- Take $x_0 \sim p_0$ and $x_1 \sim p_{data}$ (not necessarily independent)
- Define $x_t = (1 - t)x_0 + tx_1$



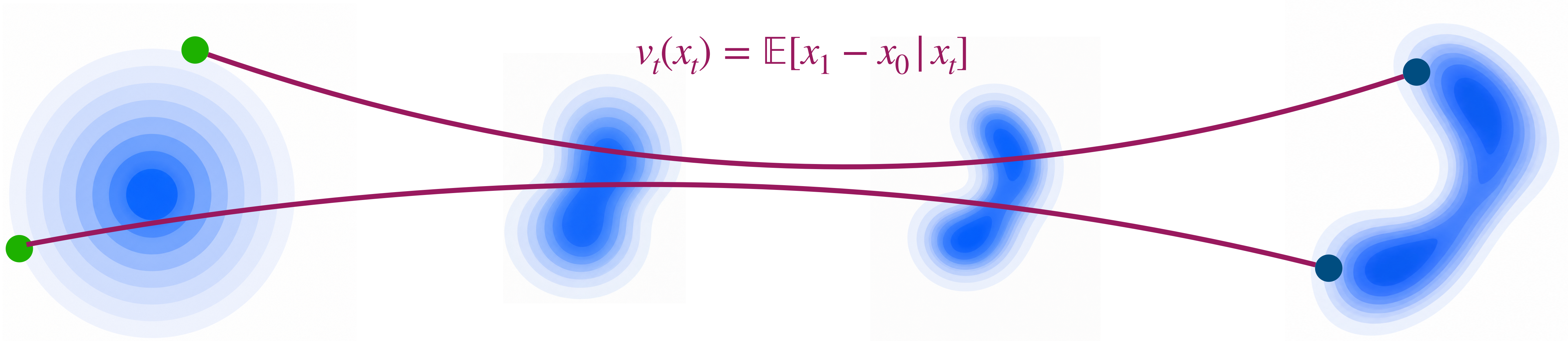
Flow Matching

Objective: Find a velocity field v_t such that...

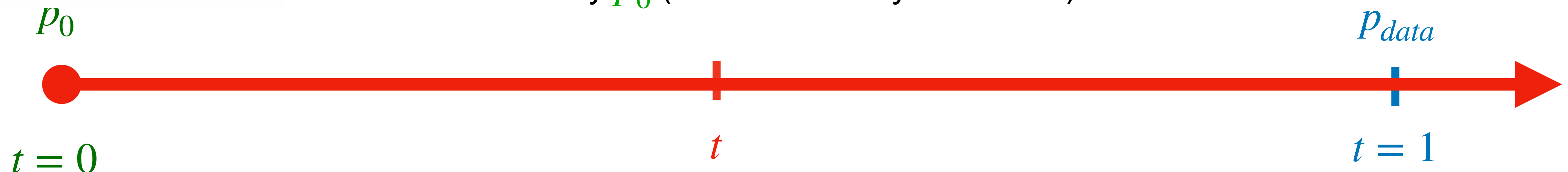
$$x_0 \sim p_0, \quad dx_t = v_t(x_t) dt$$

interpolates between p_0 and $p_1 = p_{data}$

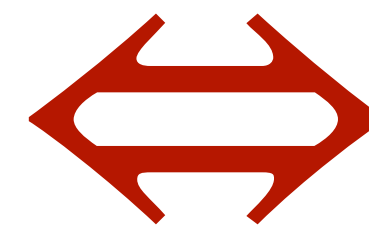
- Take $x_0 \sim p_0$ and $x_1 \sim p_{data}$ (not necessarily independent)
- Define $x_t = (1 - t)x_0 + tx_1$



- Works for any p_0 (not necessarily Gaussian)



Flow Matching



Diffusion models

$$x_0 \sim p_0, \quad dx_t = v_t(x_t) dt$$

$$v_t(x_t) = \mathbb{E}[x_1 - x_0 | x_t = (1 - t)x_0 + tx_1]$$

Exercise:

If $p_0 = \mathcal{N}(0, \text{Id})$ and (x_0, x_1) independent

$$x_0 \sim p_0, \quad dx_t = v_t(x_t) dt$$

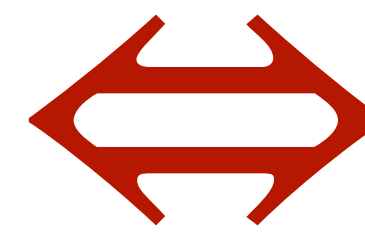
$$v_t(x_t) = \frac{\dot{s}_t}{s_t} x_t + \frac{\dot{\sigma}_t}{\sigma_t} \left[x_t - s_t D_{\sigma_t}^* \left(\frac{x_t}{s_t} \right) \right]$$

$$v_t(x_t) = \dots$$

$$\sigma_t = ?$$

$$s_t = ?$$

Flow Matching



Diffusion models

$$x_0 \sim p_0, \quad dx_t = v_t(x_t) dt$$

$$v_t(x_t) = \mathbb{E}[x_1 - x_0 | x_t = (1-t)x_0 + tx_1]$$

Exercise:

If $p_0 = \mathcal{N}(0, \text{Id})$ and (x_0, x_1) independent

$$\begin{aligned} v_t(x_t) &= \mathbb{E}[x_1 - x_0 | x_t] \\ &= \frac{1}{1-t} \left(\mathbb{E}[x_1 | x_t] - x_t \right) \end{aligned} \quad \left. \begin{array}{l} \\ \end{array} \right\} x_t = (1-t)x_0 + tx_1$$

MMSE denoiser $\mathbb{E}[x_1 | x_t] = \mathbb{E} \left[x_1 | x_t = \overset{s_t}{t} \left(x_1 + \overset{\sigma_t}{\frac{1-t}{t}} x_0 \right) \right] = D_{\sigma_t}^* \left(\frac{x_t}{t} \right)$

$s_t = t$

$$v_t(x_t) = \frac{1}{1-t} \left(D_{\sigma_t}^* \left(\frac{x_t}{t} \right) - x_t \right)$$

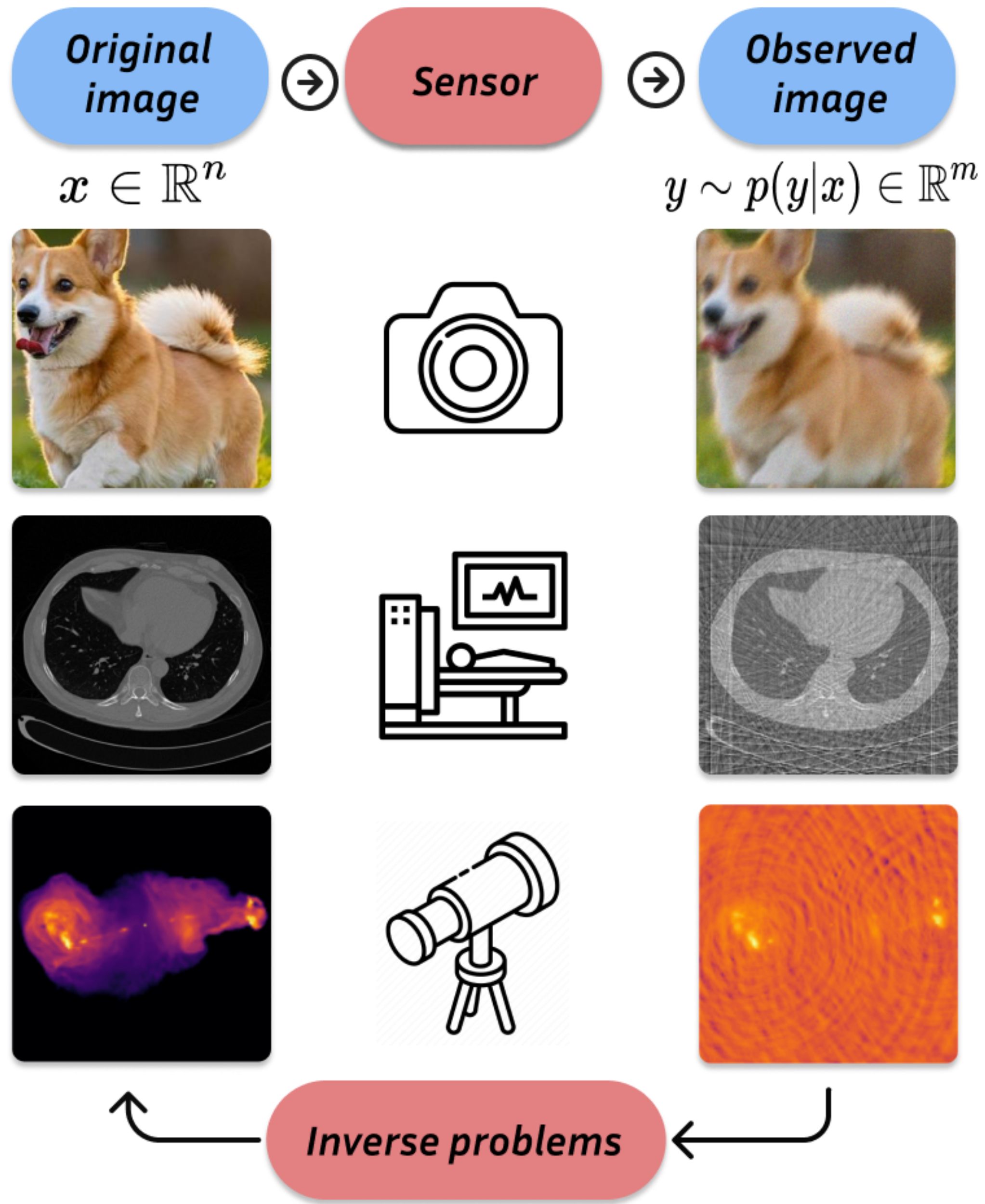
$$\sigma_t = \frac{1-t}{t}$$

$$x_0 \sim p_0, \quad dx_t = v_t(x_t) dt$$

$$v_t(x_t) = \frac{\dot{s}_t}{s_t} x_t + \frac{\dot{\sigma}_t}{\sigma_t} \left[x_t - s_t D_{\sigma_t}^* \left(\frac{x_t}{s_t} \right) \right]$$

Application to Inverse Problems

Image Inverse Problems: plug-and-play methods for MAP



Observation model (physics):

$$w \sim \mathcal{N}(0, \nu^2 \text{Id}), \quad y = Ax + w$$

$$p(y|x) = \mathcal{N}(Ax, \nu^2 \text{Id})$$

Objective: $\max_x p(x|y)$ (Maximum A-Posteriori)

Bayes: $p(x|y) \propto p(y|x) p_{data}(x)$

$$\Leftrightarrow \text{minimize } -\log p(x|y) = -\log p(y|x) - \log p_{data}(x)$$

e.g. with gradient descent

« data-fidelity »

✓ Closed-form

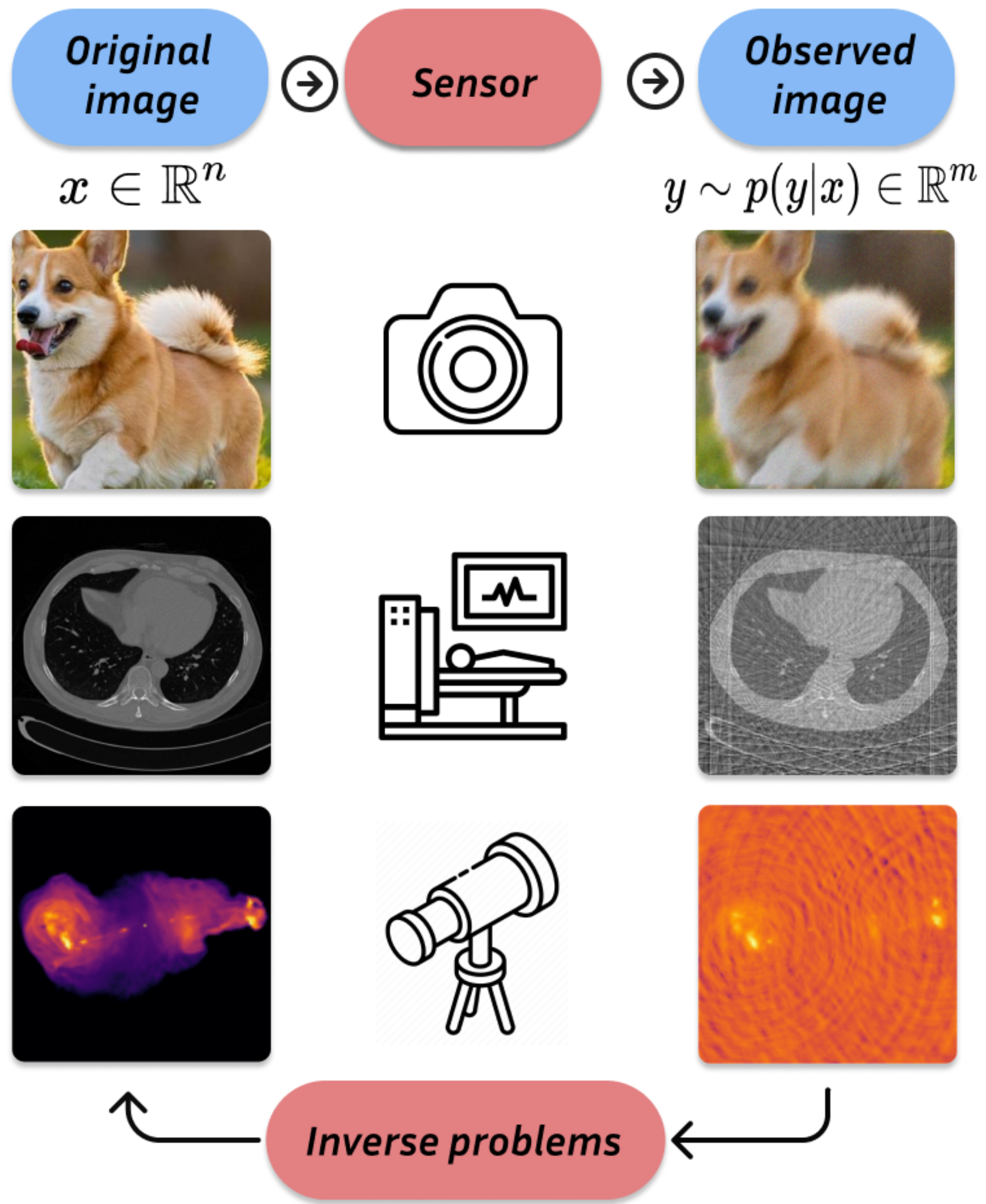
« prior »

???

Tweedie

$$D_\sigma(x) \approx x + \sigma^2 \nabla \log p_\sigma(x)$$

Image Inverse Problems: posterior sampling



Observation model (physics):

$$w \sim \mathcal{N}(0, \nu^2 \text{Id}), \quad y = Ax + w$$

Objective: Generate samples $x_i \sim p(x|y)$

→ Reconstruction: $\mathbb{E}[x|y] \approx \frac{1}{N} \sum_{i=1}^N x_i$

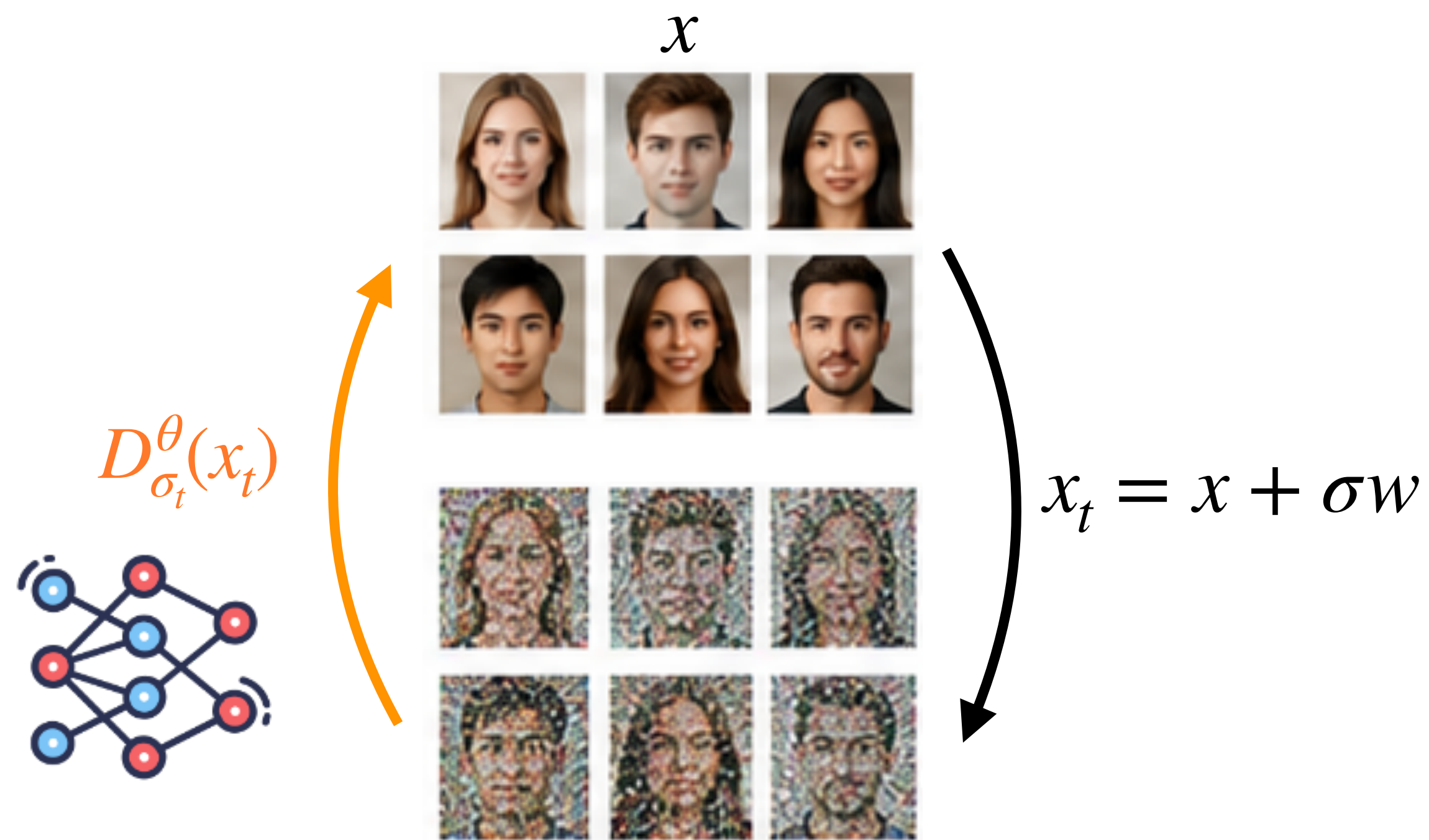
→ Uncertainty:

$$\mathbb{V}[x|y] \approx \frac{1}{N} \sum_{i=1}^N (x_i - \mathbb{E}[x|y])(x_i - \mathbb{E}[x|y])^\top$$

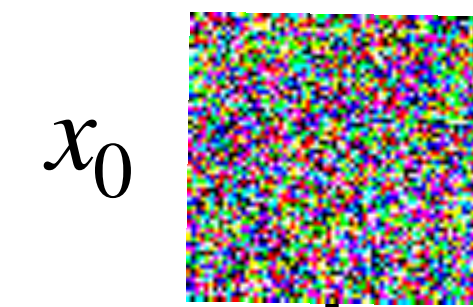
Reminder: **Unconditional** sampling of p_{data}

Unconditional Denoiser Training:

$$\operatorname{argmin}_{\theta} \mathbb{E}_{x \sim p_{data}, w \sim \mathcal{N}(0, I)} \|D_{\sigma_t}^{\theta}(x + \sigma_t w) - x\|^2$$



Unconditional Diffusion Sampling:

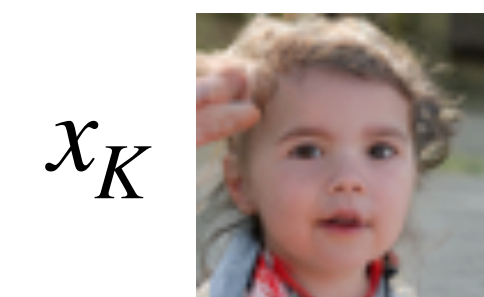


For $k = 1 \dots K$:

$$\hat{x} = D_{\sigma_{t_k}}^{\theta}(x_k)$$

$$x_{k+1} = (1 - \beta_k)x_k + \beta_k \hat{x}$$

Optionally add noise



Conditional sampling of $p(x|y)$

SR3 [Saharia et. al, '21]

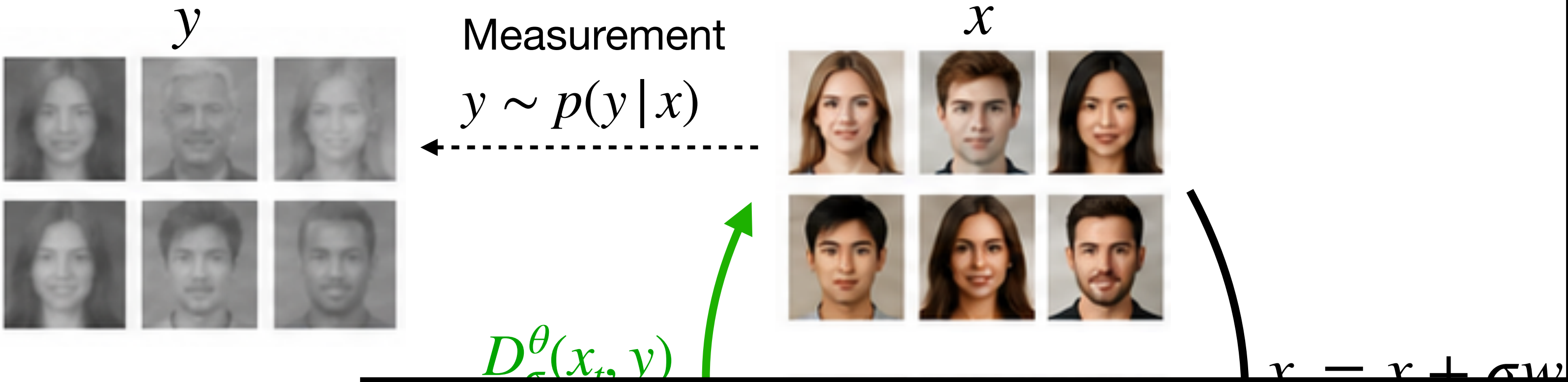
Palette [Saharia et. al, '22]

EPS [Mammadov et. al, '26]

First idea: same recipe with $p_{data} \rightarrow p(x|y)$

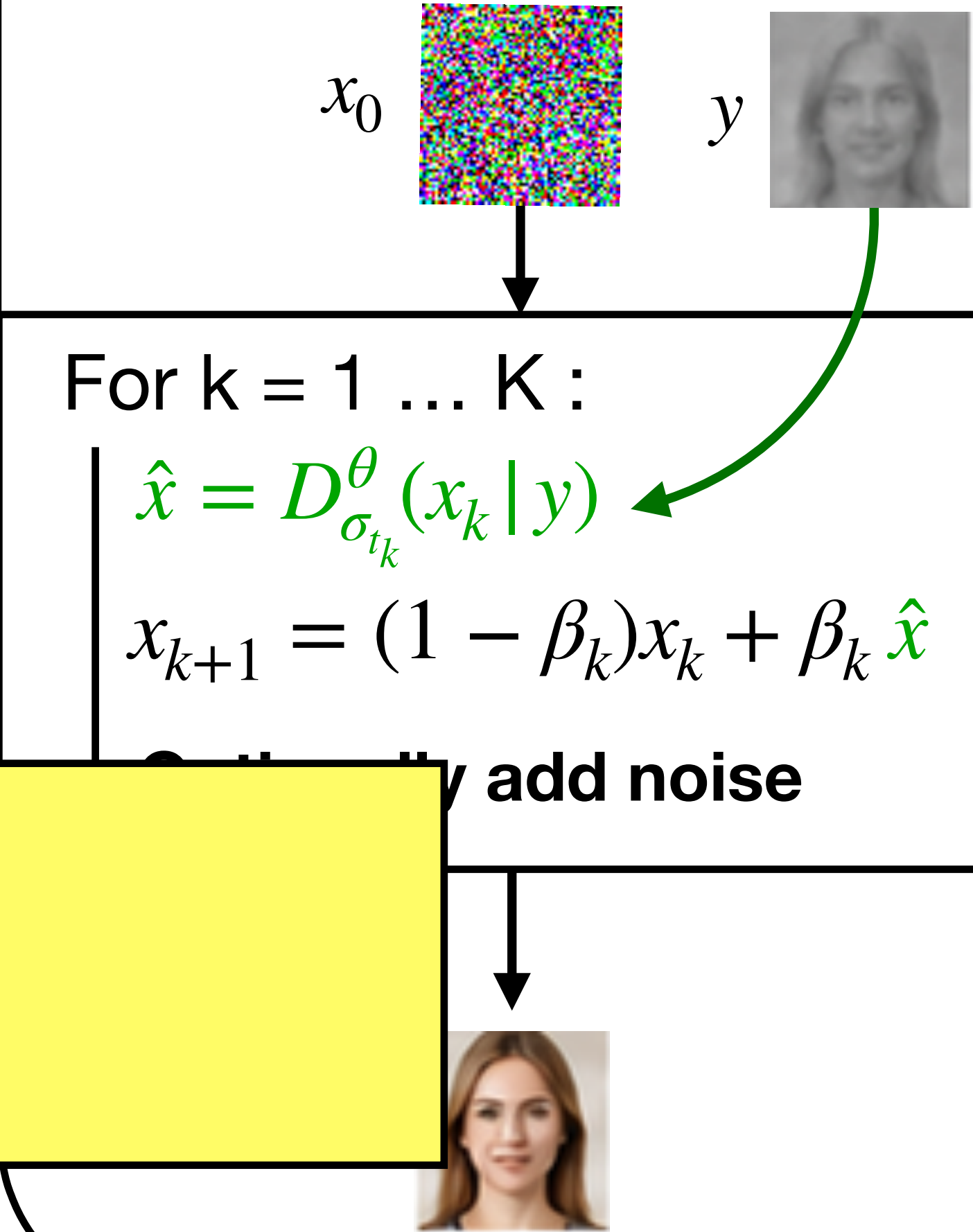
Conditional Denoiser Training:

$$\operatorname{argmin}_{\theta} \mathbb{E}_{x \sim p_{data}, y \sim p(y|x), w \sim \mathcal{N}(0, I)} \|D_{\sigma_t}^{\theta}(x + \sigma_t w, y) - x\|^2$$



- ✗ Requires a lot of data (x_i, y_i)
- ✗ Need to **retrain** for each degradation

Conditional Diffusion Sampling:

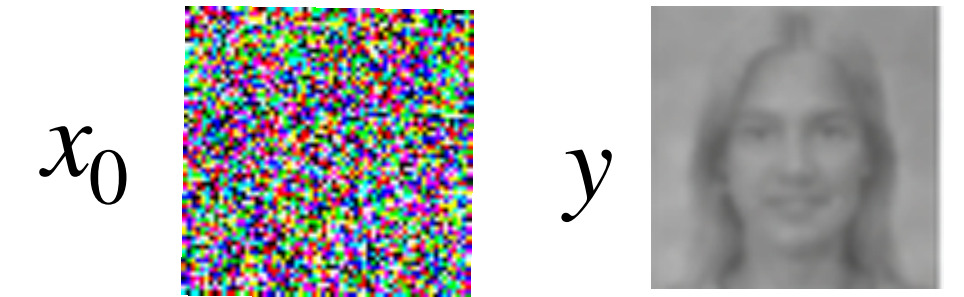


Conditional sampling of $p(x | y)$

Second idea: only use the **unconditional** denoiser

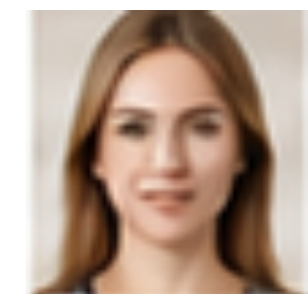
Exercise: any idea how ?

**Conditional Diffusion
Sampling:**
(continuous version)



$$dx_t = v_t(x_t | y) dt$$

$$v_t(x_t | y) = \dot{\sigma}_t \sigma_t \nabla \log p_{\sigma_t}(x_t | y)$$



Conditional sampling of $p(x | y)$

Second idea: only use the **unconditional** denoiser

Bayes: $p_{\sigma_t}(x | y) \propto p_{\sigma_t}(y | x) p_{\sigma_t}(x)$

$$\nabla \log p_{\sigma_t}(x_t | y) = \nabla \log p_{\sigma_t}(y | x_t) + \nabla \log p_{\sigma_t}(x_t)$$

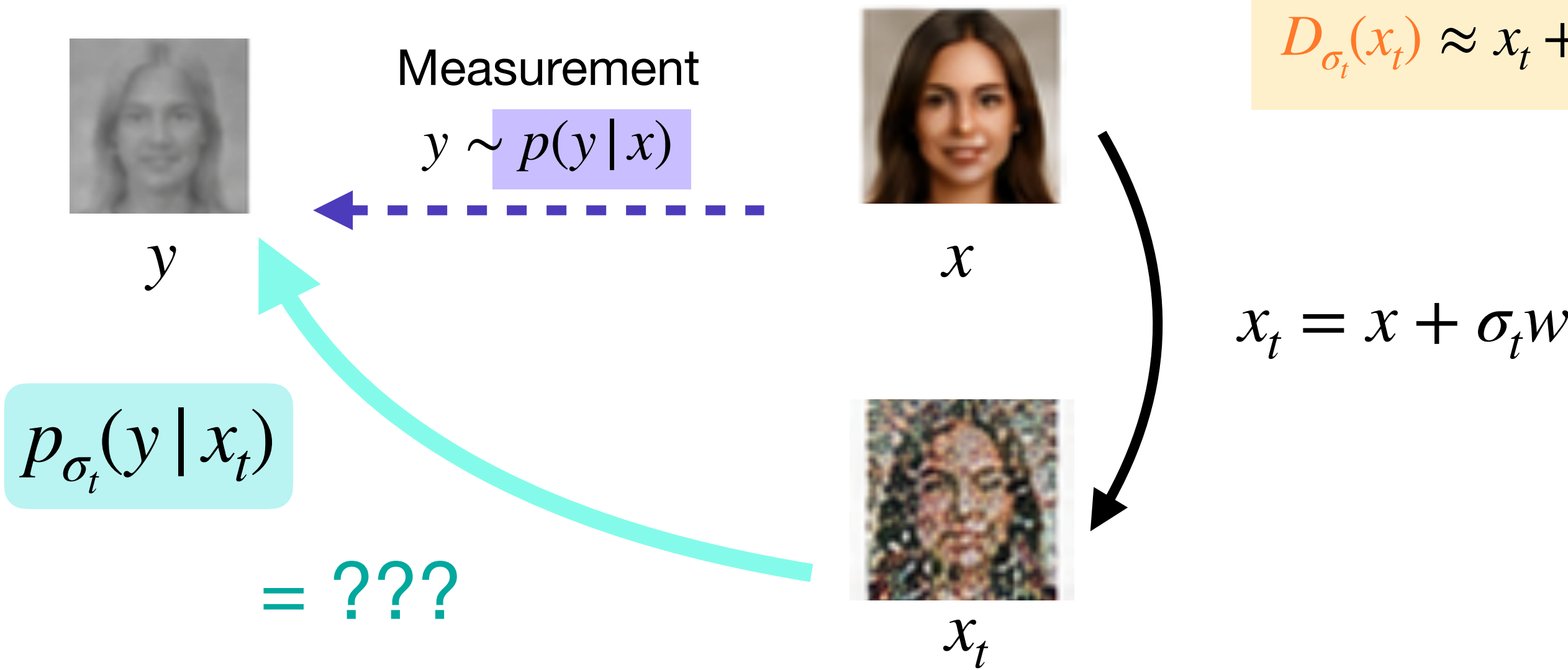
« time-dependent data-fidelity » « time-dependent prior »

✗ Not closed form

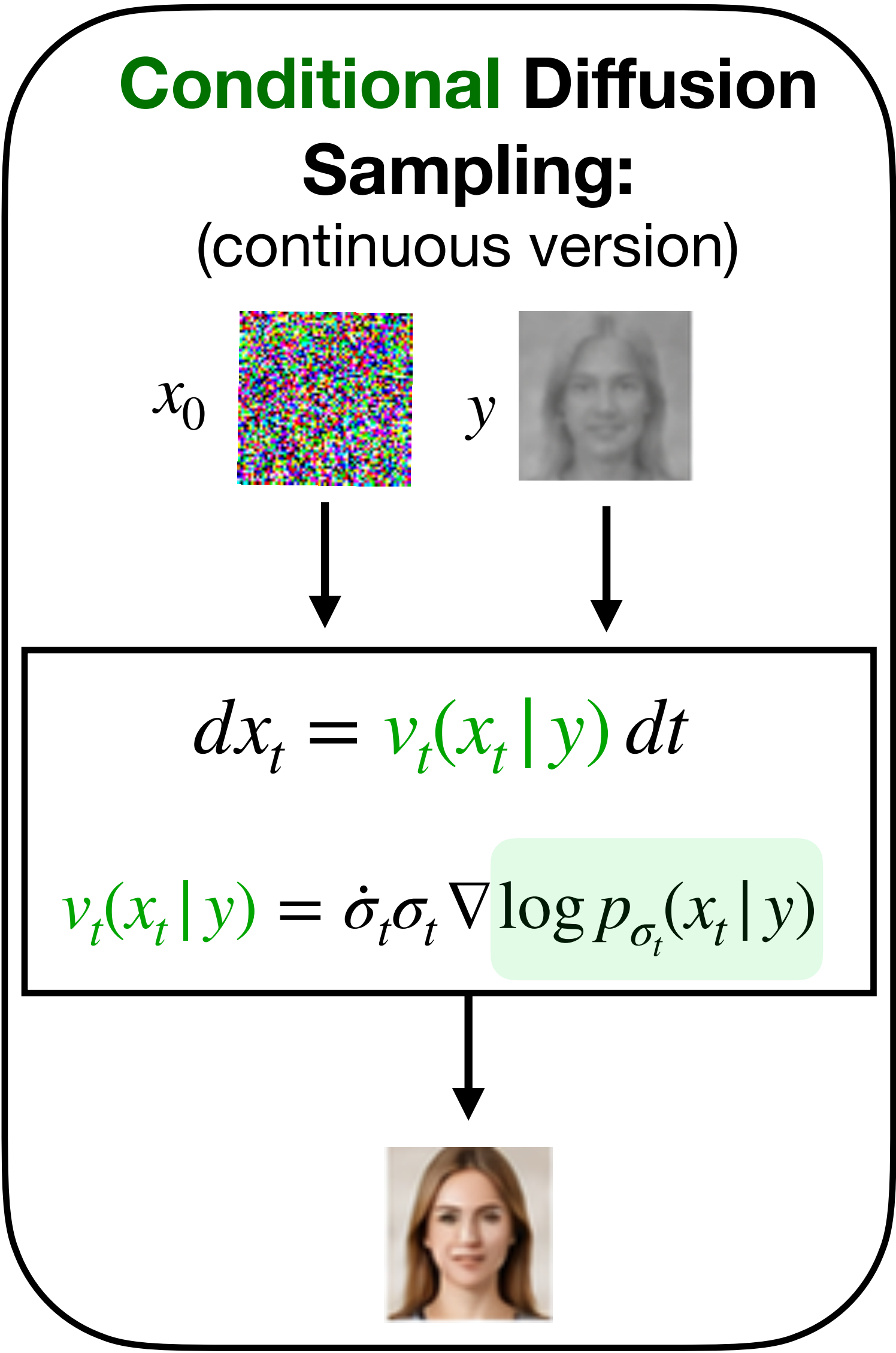
✓ Unconditional Denoiser

$$D_{\sigma_t}(x_t) \approx x_t + \sigma_t^2 \nabla \log p_{\sigma_t}(x_t)$$

Tweedie



Score-ALD [Jalal et. al, '21]: $p_{\sigma_t}(y | x_t) \approx p(y | x_t) = \mathcal{N}(Ax, \nu^2 \text{Id})$ 🤔



Conditional sampling of $p(x | y)$

Second idea: only use the **unconditional** denoiser

Bayes: $p_{\sigma_t}(x | y) \propto p_{\sigma_t}(y | x) p_{\sigma_t}(x)$

$$\nabla \log p_{\sigma_t}(x_t | y) = \nabla \log p_{\sigma_t}(y | x_t) + \nabla \log p_{\sigma_t}(x_t)$$

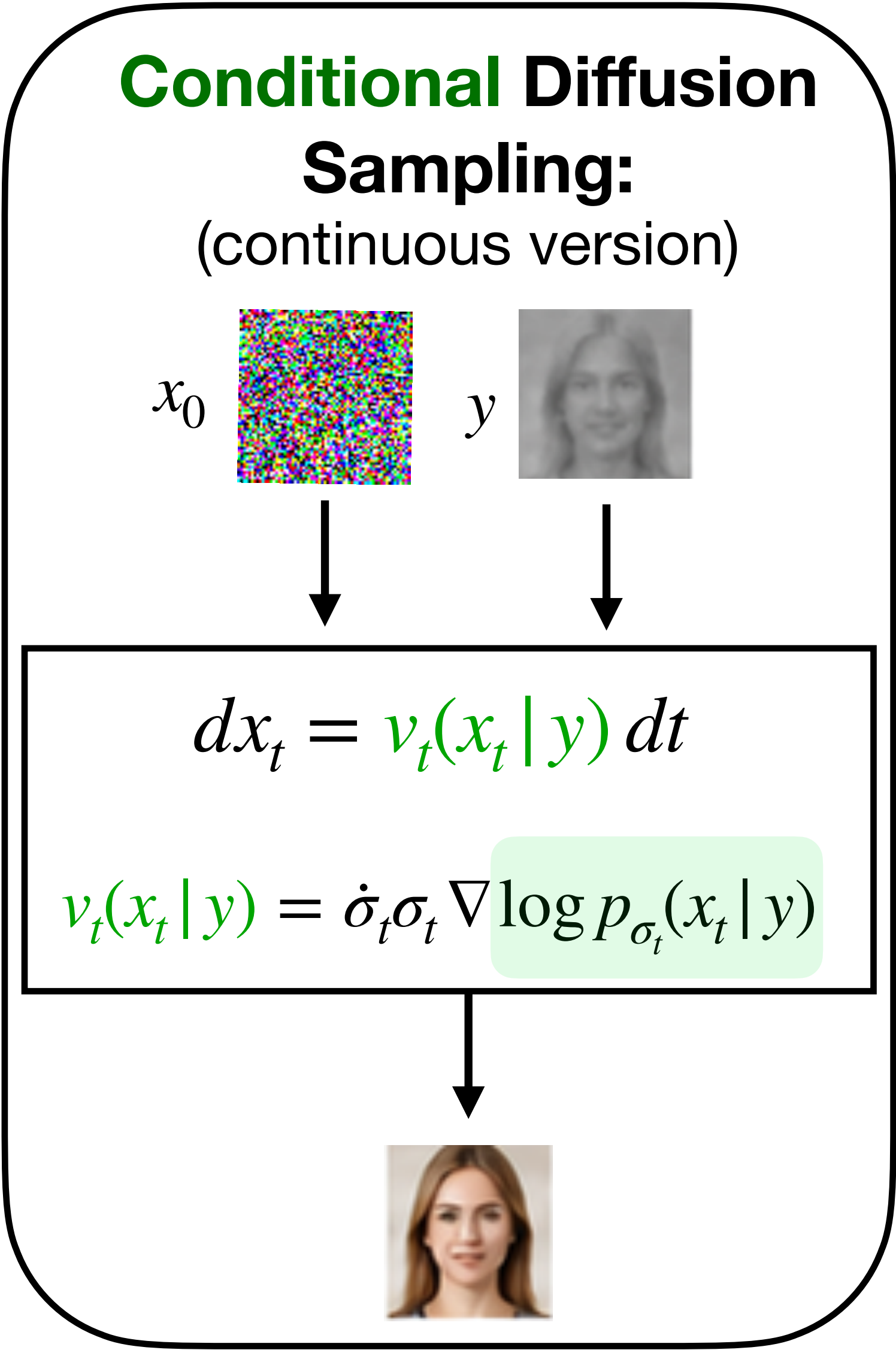
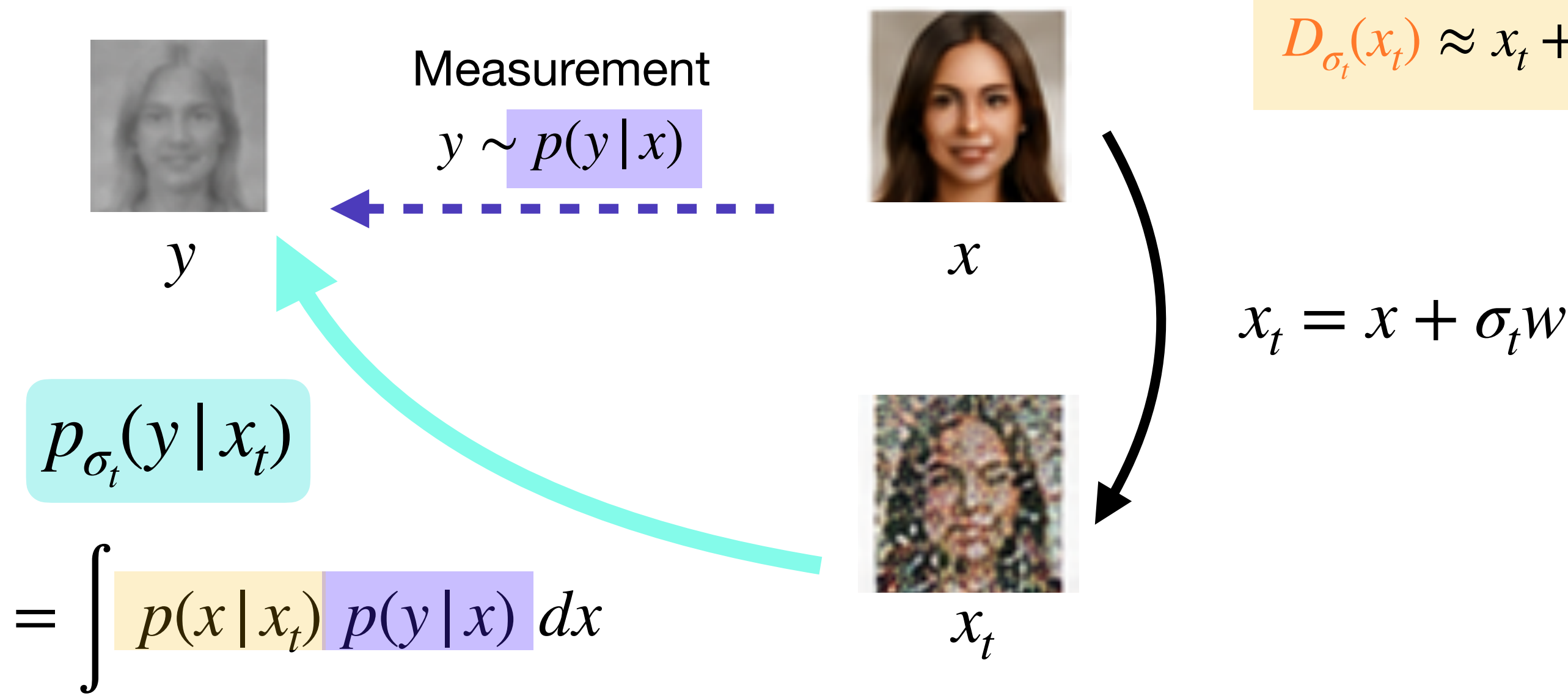
« time-dependent data-fidelity » « time-dependent prior »

✗ Not closed form

✓ Unconditional Denoiser

$$D_{\sigma_t}(x_t) \approx x_t + \sigma_t^2 \nabla \log p_{\sigma_t}(x_t)$$

Tweedie



Conditional sampling of $p(x | y)$

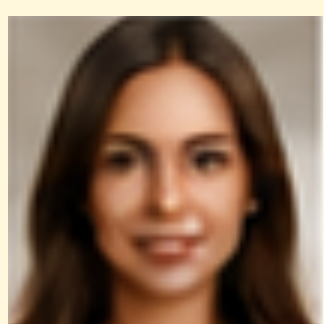
Second idea: only use the **unconditional** denoiser

Bayes: $p_{\sigma_t}(x | y) \propto p_{\sigma_t}(y | x) p_{\sigma_t}(x)$

$$\nabla \log p_{\sigma_t}(x_t | y) = \nabla \log p_{\sigma_t}(y | x_t) + \nabla \log p_{\sigma_t}(x_t)$$

$$p_{\sigma_t}(y | x_t) = \int p(x | x_t) p(y | x) dx$$

Denoising posterior $p(x | x_t)$



x

$$D_{\sigma_t}^{\theta}(x_t) \approx \mathbb{E}[x | x_t]$$



x_t

Observation model (physics): $y = Ax + w$

$$p(y | x) = \mathcal{N}(Ax, \nu^2 \text{Id})$$

✗ But the integral is untractable ...

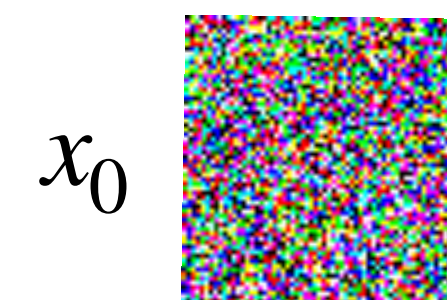


Let's try to approximate it !

Conditional Diffusion

Sampling:

(continuous version)



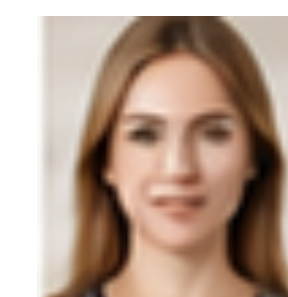
x_0



y

$$dx_t = v_t(x_t | y) dt$$

$$v_t(x_t | y) = \dot{\sigma}_t \sigma_t \nabla \log p_{\sigma_t}(x_t | y)$$



Conditional sampling of $p(x | y)$

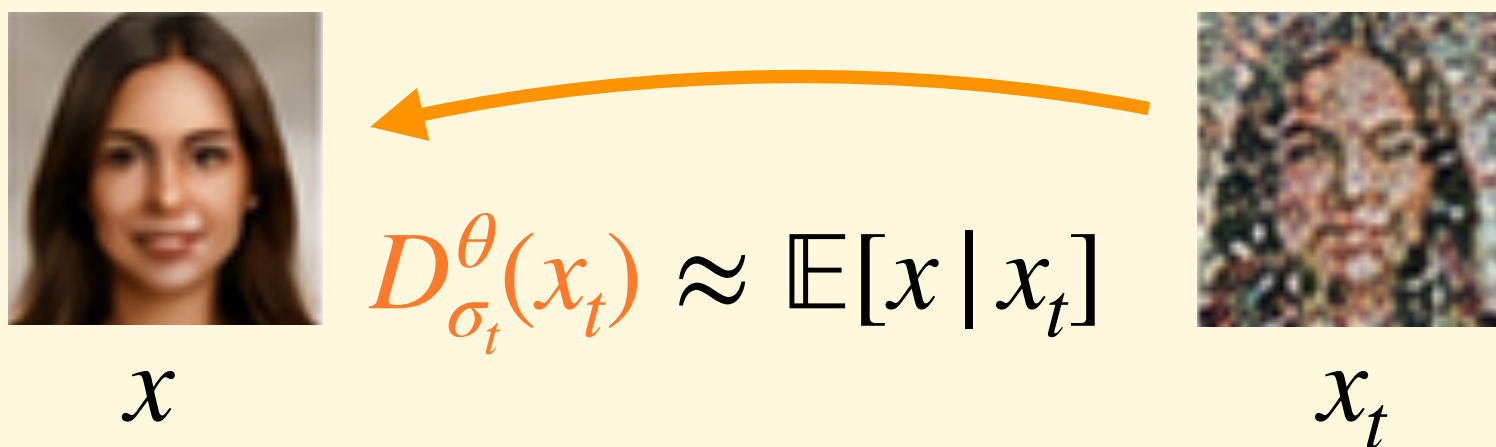
Second idea: only use the **unconditional** denoiser

Bayes: $p_{\sigma_t}(x | y) \propto p_{\sigma_t}(y | x) p_{\sigma_t}(x)$

$$\nabla \log p_{\sigma_t}(x_t | y) = \nabla \log p_{\sigma_t}(y | x_t) + \nabla \log p_{\sigma_t}(x_t)$$

$$p_{\sigma_t}(y | x_t) = \int p(x | x_t) p(y | x) dx$$

Denoising posterior $p(x | x_t)$



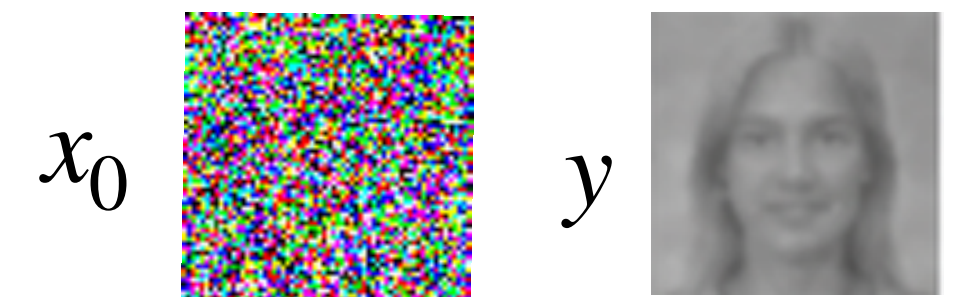
Observation model (physics): $y = Ax + w$
 $p(y | x) = \mathcal{N}(Ax, \nu^2 \text{Id})$

DPS [Chung et. al, '23]:

$$p(x | x_t) \approx \mathbf{1}_{x=D_{\sigma_t}^{\theta}(x_t)}$$

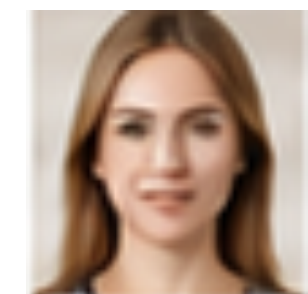
$$\Rightarrow p_{\sigma_t}(y | x_t) \approx ? \text{ Exercise}$$

Conditional Diffusion Sampling:
(continuous version)



$$dx_t = v_t(x_t | y) dt$$

$$v_t(x_t | y) = \dot{\sigma}_t \sigma_t \nabla \log p_{\sigma_t}(x_t | y)$$



Conditional sampling of $p(x | y)$

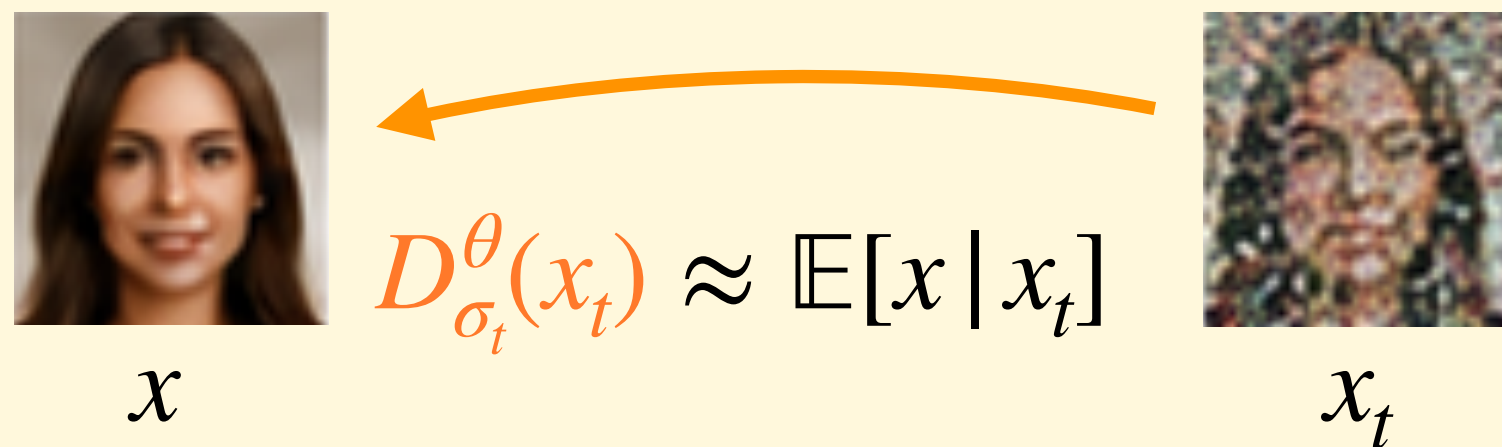
Second idea: only use the **unconditional** denoiser

Bayes: $p_{\sigma_t}(x | y) \propto p_{\sigma_t}(y | x) p_{\sigma_t}(x)$

$$\nabla \log p_{\sigma_t}(x_t | y) = \nabla \log p_{\sigma_t}(y | x_t) + \nabla \log p_{\sigma_t}(x_t)$$

$$p_{\sigma_t}(y | x_t) = \int p(x | x_t) p(y | x) dx$$

Denoising posterior $p(x | x_t)$



Observation model (physics): $y = Ax + w$
 $p(y | x) = \mathcal{N}(Ax, \nu^2 \text{Id})$

DPS [Chung et. al, '23]:

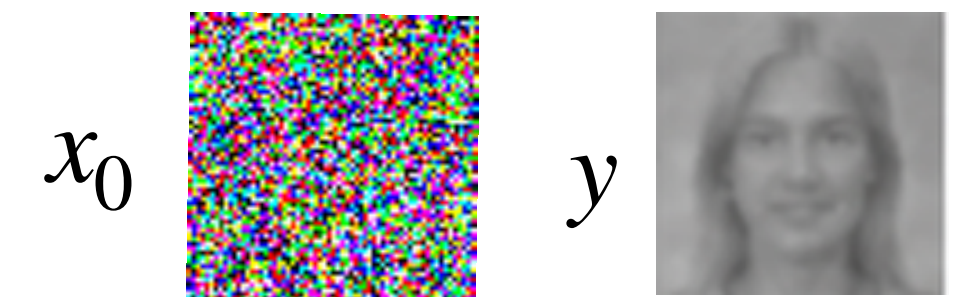
$$p(x | x_t) \approx \mathbf{1}_{x=D_{\sigma_t}^{\theta}(x_t)}$$

$$\Rightarrow p_{\sigma_t}(y | x_t) \approx \mathcal{N}(AD_{\sigma_t}^{\theta}(x_t), \nu^2 \text{Id})$$

Conditional Diffusion

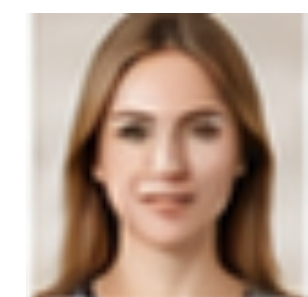
Sampling:

(continuous version)



$$dx_t = v_t(x_t | y) dt$$

$$v_t(x_t | y) = \dot{\sigma}_t \sigma_t \nabla \log p_{\sigma_t}(x_t | y)$$



Conditional sampling of $p(x | y)$

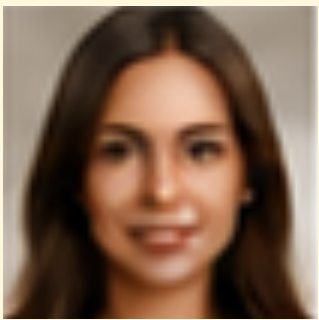
Second idea: only use the **unconditional** denoiser

Bayes: $p_{\sigma_t}(x | y) \propto p_{\sigma_t}(y | x) p_{\sigma_t}(x)$

$$\nabla \log p_{\sigma_t}(x_t | y) = \nabla \log p_{\sigma_t}(y | x_t) + \nabla \log p_{\sigma_t}(x_t)$$

$$p_{\sigma_t}(y | x_t) = \int p(x | x_t) p(y | x) dx$$

Denoising posterior $p(x | x_t)$



x

$D_{\sigma_t}^{\theta}(x_t) \approx \mathbb{E}[x | x_t]$



x_t

Observation model (physics): $y = Ax + w$

$$p(y | x) = \mathcal{N}(Ax, \nu^2 \text{Id})$$

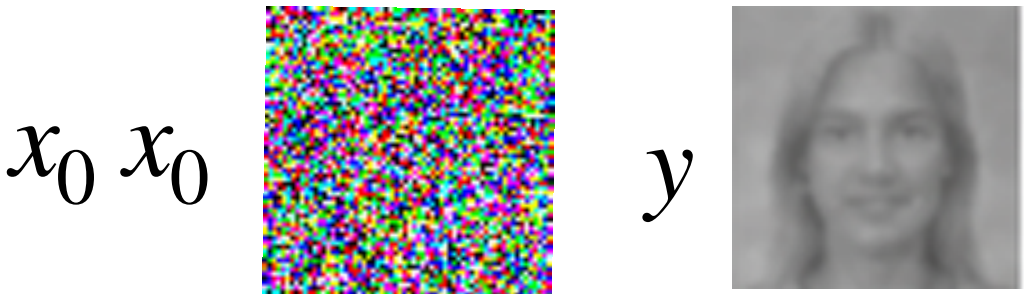
PiGDM [Song et. al, '23]:

$$p(x | x_t) \approx \mathcal{N}(D_{\sigma_t}^{\theta}(x_t), r_t^2 \text{Id})$$

$p_{\sigma_t}(y | x_t) \approx ?$ Exercise

Conditional Diffusion Sampling:

(continuous version)



$$dx_t = v_t(x_t | y) dt$$

$$v_t(x_t | y) = \dot{\sigma}_t \sigma_t \nabla \log p_{\sigma_t}(x_t | y)$$



Conditional sampling of $p(x | y)$

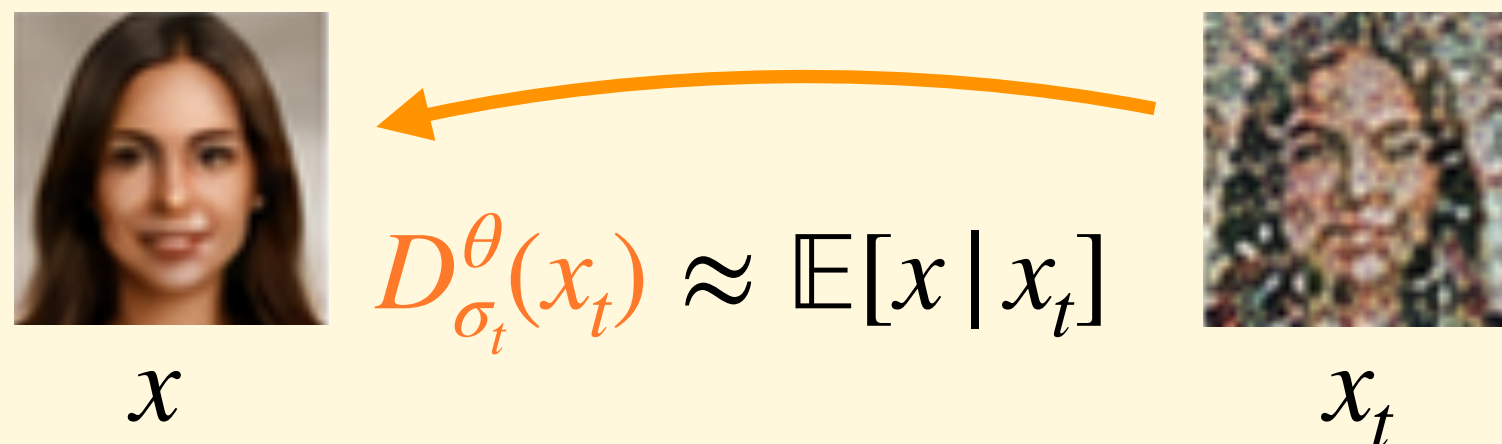
Second idea: only use the **unconditional** denoiser

Bayes: $p_{\sigma_t}(x | y) \propto p_{\sigma_t}(y | x) p_{\sigma_t}(x)$

$$\nabla \log p_{\sigma_t}(x_t | y) = \nabla \log p_{\sigma_t}(y | x_t) + \nabla \log p_{\sigma_t}(x_t)$$

$$p_{\sigma_t}(y | x_t) = \int p(x | x_t) p(y | x) dx$$

Denoising posterior $p(x | x_t)$



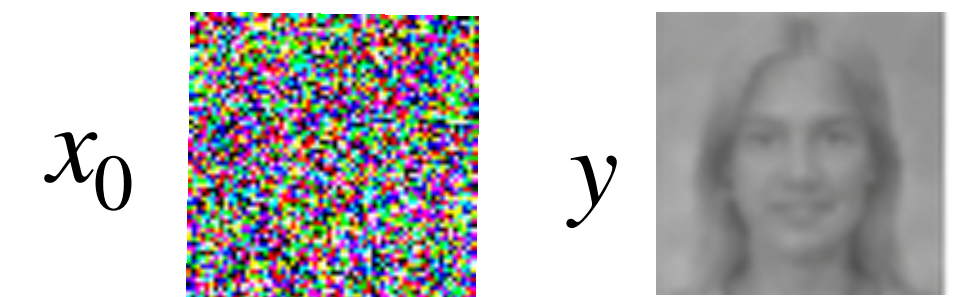
Observation model (physics): $y = Ax + w$
 $p(y | x) = \mathcal{N}(Ax, \nu^2 \text{Id})$

PiGDM [Song et. al, '23]:

$$p(x | x_t) \approx \mathcal{N}(D_{\sigma_t}^{\theta}(x_t), r_t^2 \text{Id})$$

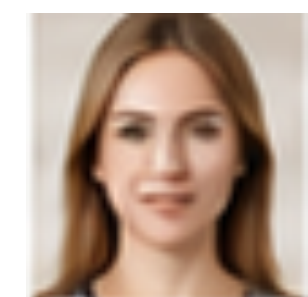
$$\Rightarrow p_{\sigma_t}(y | x_t) \approx \mathcal{N}(AD_{\sigma_t}^{\theta}(x_t), r_t^2 AA^{\top} + \nu^2 \text{Id})$$

Conditional Diffusion Sampling:
(continuous version)



$$dx_t = v_t(x_t | y) dt$$

$$v_t(x_t | y) = \dot{\sigma}_t \sigma_t \nabla \log p_{\sigma_t}(x_t | y)$$



Conditional sampling of $p(x | y)$

Second idea: only use the **unconditional** denoiser

Bayes: $p_{\sigma_t}(x | y) \propto p_{\sigma_t}(y | x) p_{\sigma_t}(x)$

$$\nabla \log p_{\sigma_t}(x_t | y) = \nabla \log p_{\sigma_t}(y | x_t) + \nabla \log p_{\sigma_t}(x_t)$$

$$p_{\sigma_t}(y | x_t) = \int p(x | x_t) p(y | x) dx$$

Denoising posterior $p(x | x_t)$



$$\sigma_t^2 \nabla_{x_t} (D_{\sigma_t}^{\theta}(x_t)) \approx \text{Cov}[x | x_t]$$

Observation model (physics): $y = Ax + w$

$$p(y | x) = \mathcal{N}(Ax, \nu^2 \text{Id})$$

Moment Matching [Rozet et. al, '24]:

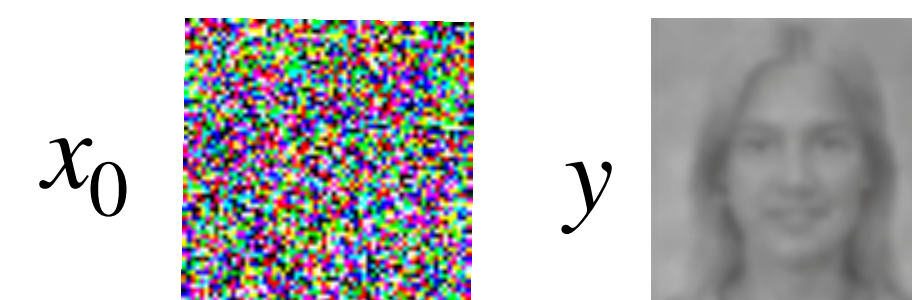
$$p(x | x_t) \approx \mathcal{N}(D_{\sigma_t}^{\theta}(x_t), \sigma^2 \nabla D_{\sigma_t}^{\theta}(x_t))$$

$$\Rightarrow p_{\sigma_t}(y | x_t) \approx ? \quad \textit{Exercise:}$$

Conditional Diffusion

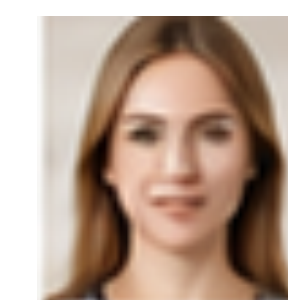
Sampling:

(continuous version)



$$dx_t = v_t(x_t | y) dt$$

$$v_t(x_t | y) = \dot{\sigma}_t \sigma_t \nabla \log p_{\sigma_t}(x_t | y)$$



Conditional sampling of $p(x | y)$

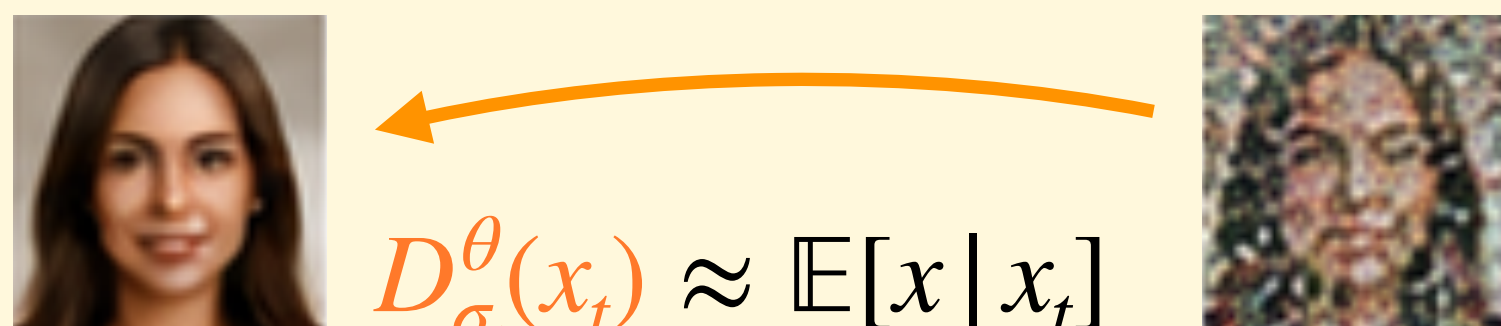
Second idea: only use the **unconditional** denoiser

Bayes: $p_{\sigma_t}(x | y) \propto p_{\sigma_t}(y | x) p_{\sigma_t}(x)$

$$\nabla \log p_{\sigma_t}(x_t | y) = \nabla \log p_{\sigma_t}(y | x_t) + \nabla \log p_{\sigma_t}(x_t)$$

$$p_{\sigma_t}(y | x_t) = \int p(x | x_t) p(y | x) dx$$

Denoising posterior $p(x_0 | x_t)$



$$D_{\sigma_t}^{\theta}(x_t) \approx \mathbb{E}[x | x_t]$$

$$\sigma_t^2 \nabla_{x_t} (D_{\sigma_t}^{\theta}(x_t)) \approx \text{Cov}[x | x_t]$$

Observation model (physics): $y = Ax + w$

$$p(y | x) = \mathcal{N}(Ax, \nu^2 \text{Id})$$

Moment Matching [Rozet et. al, '24]:

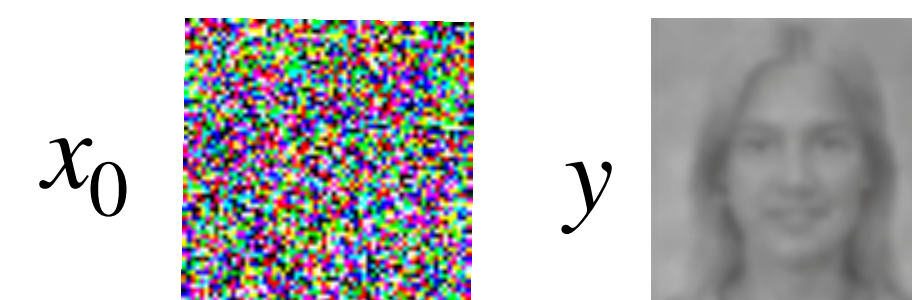
$$p(x_0 | x_t) \approx \mathcal{N}(D_{\sigma_t}^{\theta}(x_t), \sigma^2 \nabla D_{\sigma_t}^{\theta}(x_t))$$

$$\Rightarrow p_{\sigma_t}(y | x_t) \approx \mathcal{N}(AD_{\sigma_t}^{\theta}(x_t), \sigma_t^2 A \nabla D_{\sigma_t}^{\theta}(x_t) A^{\top} + \nu^2 \text{Id})$$

Conditional Diffusion

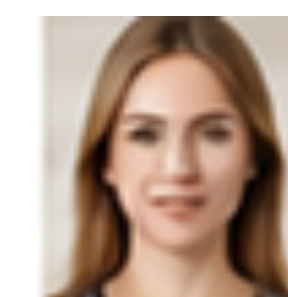
Sampling:

(continuous version)



$$dx_t = v_t(x_t | y) dt$$

$$v_t(x_t | y) = \dot{\sigma}_t \sigma_t \nabla \log p_{\sigma_t}(x_t | y)$$



Conditional sampling of $p(x | y)$

Third idea: approximation of the conditional score

$$\nabla \log p_{\sigma_t}(x_t | y) = \nabla \log p_{\sigma_t}(y | x_t) + \nabla \log p_{\sigma_t}(x_t)$$

Score-ALD

[Jalal et. al, '21]:

$$p_{\sigma_t}(y | x_t) \approx p(y | x_t)$$

$$\nabla p_{\sigma_t}(y | x_t) \propto -A^\top (y - Ax_t)$$

DPS [Chung et. al, '23]:

$$p_{\sigma_t}(y | x_t) \approx \mathcal{N}(AD_{\sigma_t}^\theta(x_t), \nu^2 \text{Id})$$

$$\nabla p_{\sigma_t}(y | x_t) \propto -\nabla D_{\sigma_t}^\theta(x_t)^\top A^\top (y - AD_{\sigma_t}^\theta(x_t))$$

PiGDM [Song et. al, '23]:

$$p_{\sigma_t}(y | x_t) \approx \mathcal{N}(AD_{\sigma_t}^\theta(x_t), r_t^2 AA^\top + \nu^2 \text{Id})$$

$$\nabla p_{\sigma_t}(y | x_t) \propto -\nabla D_{\sigma_t}^\theta(x_t)^\top A^\top (r_t^2 AA^\top + \nu^2 \text{Id})^{-1} (y - AD_{\sigma_t}^\theta(x_t))$$

Moment Matching

[Rozet et. al, '24]:

$$p_{\sigma_t}(y | x_t) \approx \mathcal{N}(AD_{\sigma_t}^\theta(x_t), A \nabla D_{\sigma_t}^\theta(x_t)^\top A^\top + \nu^2 \text{Id})$$

$$\nabla p_{\sigma_t}(y | x_t) \propto -\nabla D_{\sigma_t}^\theta(x_t)^\top A^\top (\sigma_t^2 A \nabla D_{\sigma_t}^\theta(x_t)^\top A^\top + \nu^2 \text{Id})^{-1} (y - AD_{\sigma_t}^\theta(x_t))$$

Error vector in y-space

Preconditioner

re-scaling

+ more in [Daras et al, 24]

Conclusion

- Diffusion / Flow Matching models **state-of-the-art** for sampling images.
- Based on **image denoising**: well understood and « easy » to train.
- For application to posterior sampling **without retraining**:
requires an **approximation** of the posterior ...

Conclusion

- Diffusion / Flow Matching models **state-of-the-art** for sampling images.
- Based on **image denoising**: well understood and « easy » to train.
- For application to posterior sampling **without retraining**:
requires an **approximation** of the posterior ...

Some recent research directions

- How to select parameters *with respect to the target distribution*.

Diffusion models parameters

$$x_0 \sim \mathcal{N}(0, \sigma_T^2 \text{Id}), \quad dx_t = v_t(x_t) dt + \sqrt{2\alpha a_t} dW_t \quad v_t(x_t) = f(s_t, \sigma_t, \alpha, D_{\sigma_t}^*(x_t))$$

Parameters of the algorithm: diffusion coefficient α , time schedulers s_t, σ_t

[Hurault et al., '25 '26] Optimization of parameters to minimize sampling errors

From Score Matching to Diffusion:
A Fine-Grained Error Analysis in the Gaussian Setting

Samuel Hurault
ENS Paris, PSL, CNRS
samuel.hurault@ens.fr

Matthieu Terris
Univ. Paris-Saclay, Inria, CEA
matthieu.terris@inria.fr

Thomas Moreau
Univ. Paris-Saclay, Inria, CEA
thomas.moreau@inria.fr

Gabriel Peyré
ENS Paris, PSL, CNRS
gabriel.peyre@ens.fr

Geometry-Aware Discretization Error of
Diffusion Models

Samuel Hurault
Univ. Gustave Eiffel, CNRS, LIGM
samuel.hurault@univ-eiffel.fr

Thomas Moreau
Univ. Paris-Saclay, Inria, CEA
thomas.moreau@inria.fr

Gabriel Peyré
CNRS, ENS, PSL Université
gabriel.peyre@ens.fr

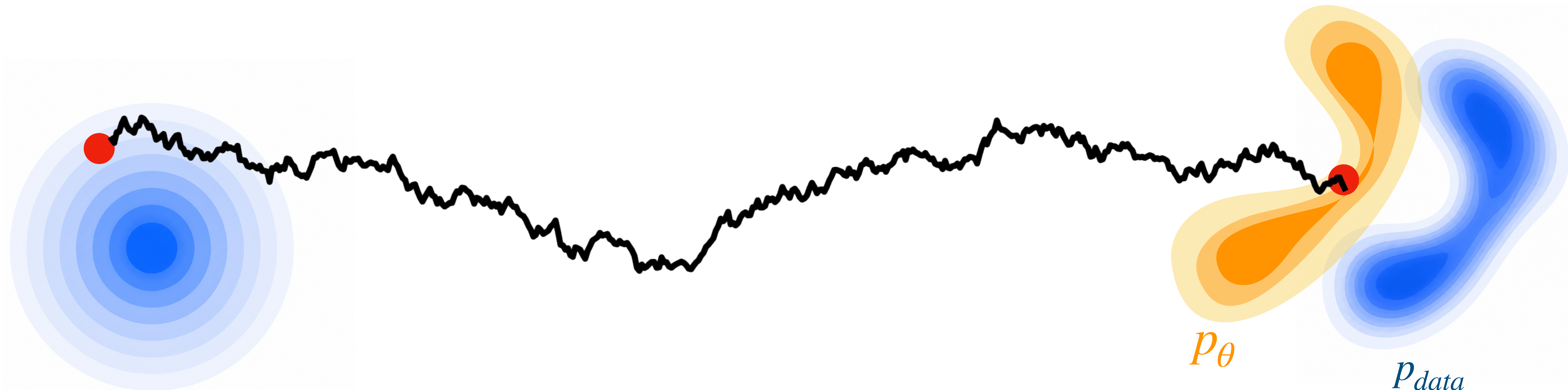
Diffusion models errors: **training** error

$$x_0 \sim \mathcal{N}(0, \sigma_T^2 \text{Id}), \quad dx_t = v_t(x_t) dt + \sqrt{2\alpha a_t} dW_t \quad v_t(x_t) = f(s_t, \sigma_t, \alpha, D_{\sigma_t}^*(x_t))$$

Train a neural network D_{σ}^{θ} to minimize
 $\text{argmin}_{\theta} \mathbb{E}_{x \sim p_{data}, w \sim \mathcal{N}(0, I)} \|D_{\sigma}^{\theta}(x + \sigma w) - x\|^2$

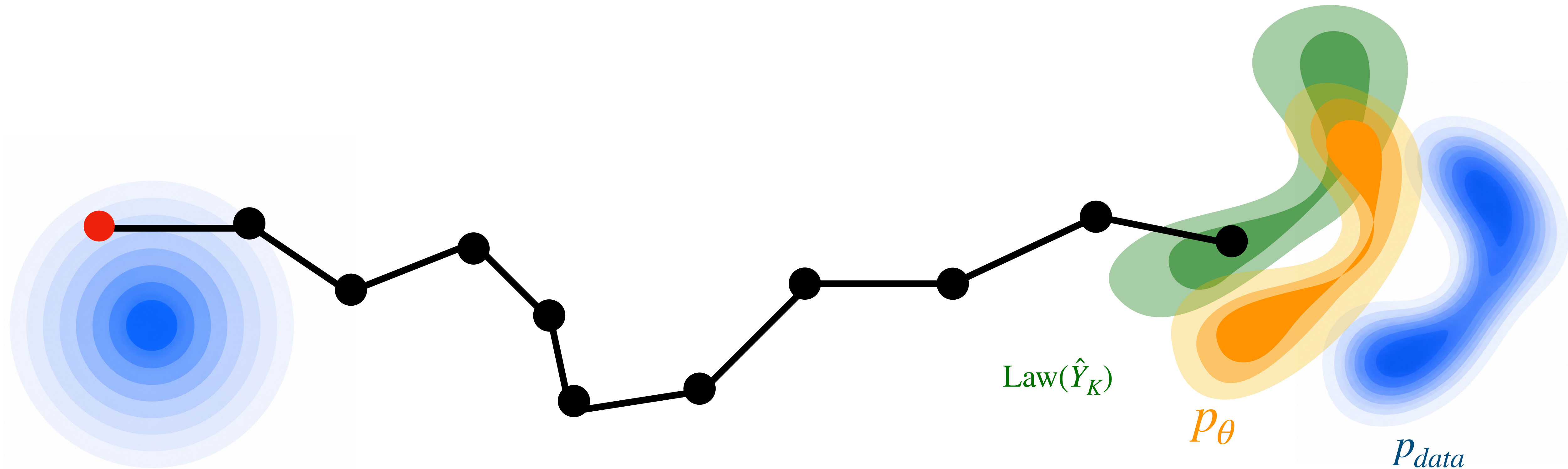
$$D_{\sigma_t}^{\theta}(x) \approx D_{\sigma_t}^*(x)$$

↑
Denoiser training error



Diffusion models errors: discretization error

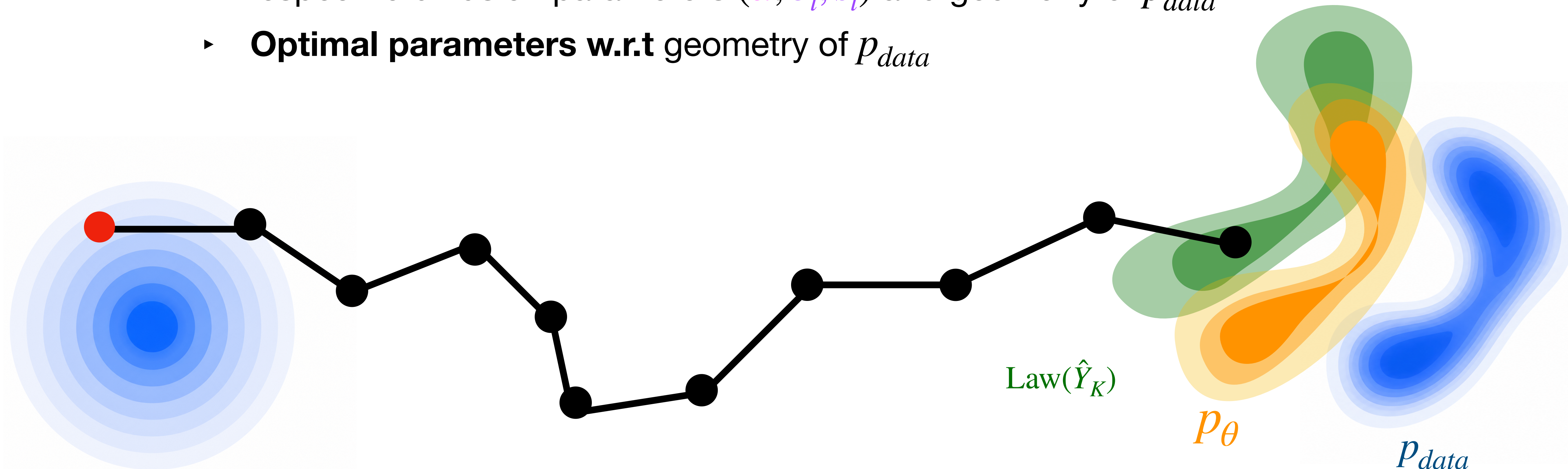
$$x_0 \sim \mathcal{N}(0, \sigma_T^2 \text{Id}), \quad dx_t = v_t(x_t) dt + \sqrt{2\alpha a_t} dW_t \quad v_t(x_t) = f(s_t, \sigma_t, \alpha, D_{\sigma_t}^*(x_t))$$



Diffusion models errors

$$x_0 \sim \mathcal{N}(0, \sigma_T^2 \text{Id}), \quad dx_t = v_t(x_t) dt + \sqrt{2\alpha a_t} dW_t \quad v_t(x_t) = f(s_t, \sigma_t, \alpha, D_{\sigma_t}^*(x_t))$$

- We calculate explicitly sampling error due to **training** and **discretization**
- With respect to diffusion parameters (α, σ_t, s_t) and geometry of p_{data}
 - **Optimal parameters w.r.t** geometry of p_{data}



Diffusion models errors

$$x_0 \sim \mathcal{N}(0, \sigma_T^2 \text{Id}), \quad dx_t = v_t(x_t) dt + \sqrt{2\alpha a_t} dW_t \quad v_t(x_t) = f(s_t, \sigma_t, \alpha, D_{\sigma_t}^*(x_t))$$

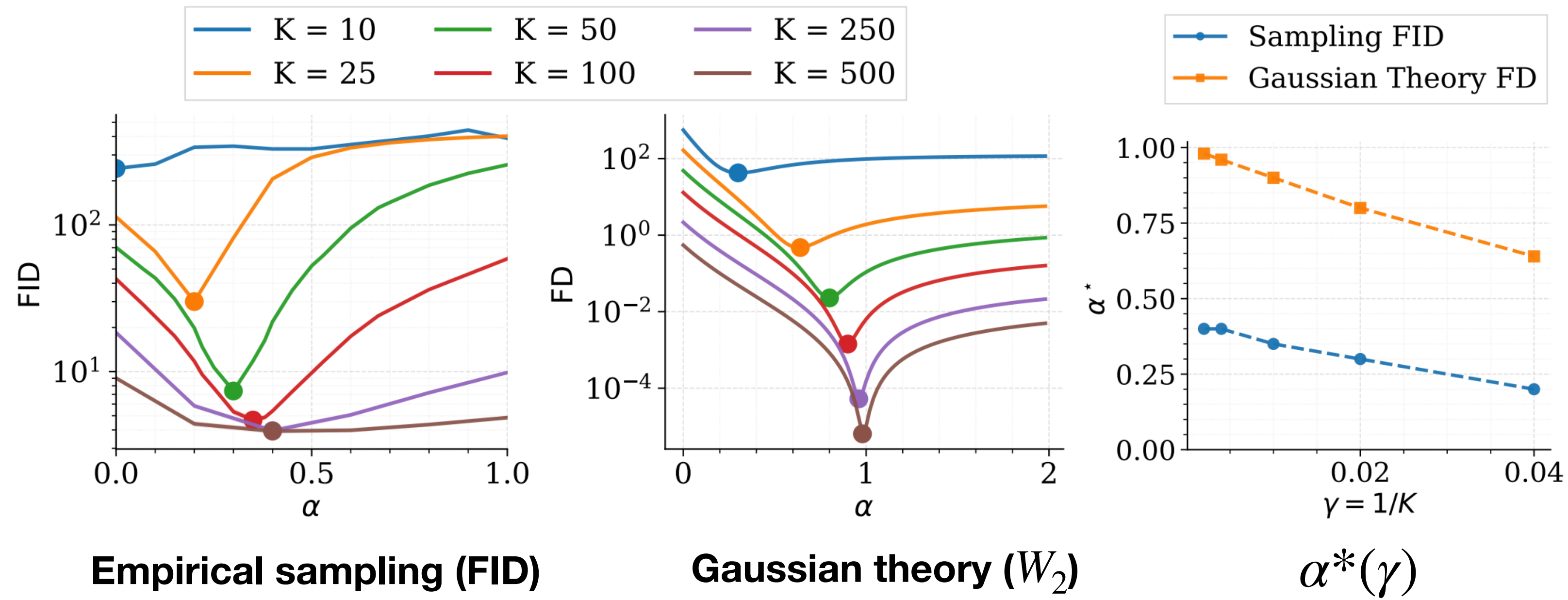
- We calculate explicitly sampling error due to training and discretization
- With respect to diffusion parameters (α, σ_t, s_t) and geometry of p_{data}
 - Optimal parameters w.r.t geometry of p_{data}

Assuming Gaussian data

$$p_{data} \sim \mathcal{N}(\mu_{data}, \Sigma_{data} = U \text{Diag}(\lambda_i) U^\top)$$

Optimization of the stochasticity parameter α

Image dataset sampling (FFHQ) Sampling error w.r.t number of discretization steps K (stepsize $\gamma = 1/K$) :



α^* decreases linearly with the stepsize γ

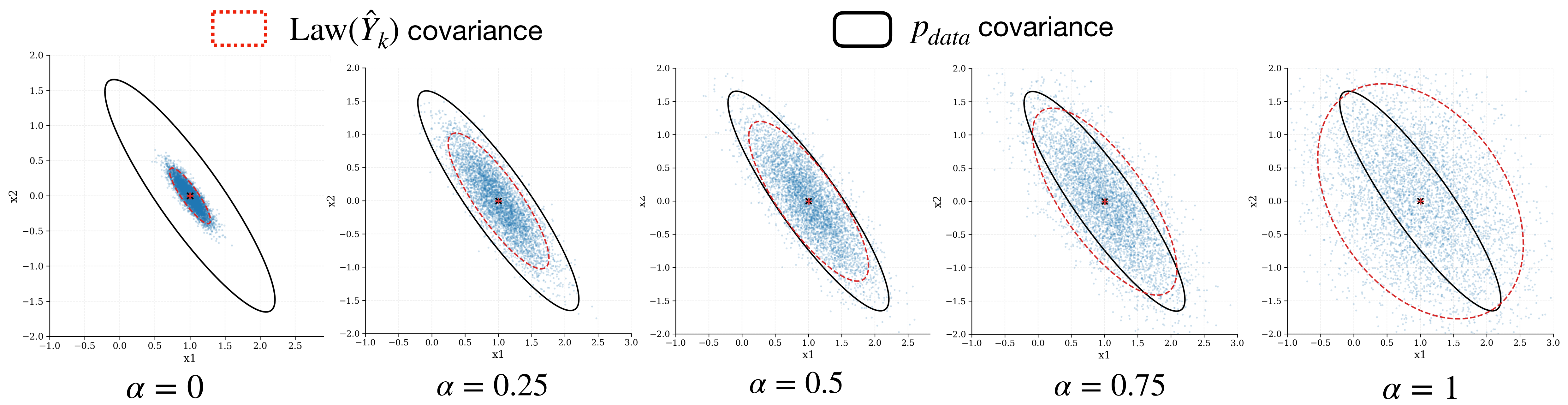
For *Gaussian data*,

$$\alpha^*(\gamma) = \operatorname{argmin}_{\alpha} W_2(p_{\text{data}}, \text{Law}(\hat{Y}_K)) = 1 - \gamma \sum_{i=1}^d w_i \left(\frac{1}{\lambda_i} \right)^{\frac{1}{2\beta}} + o(\gamma)$$

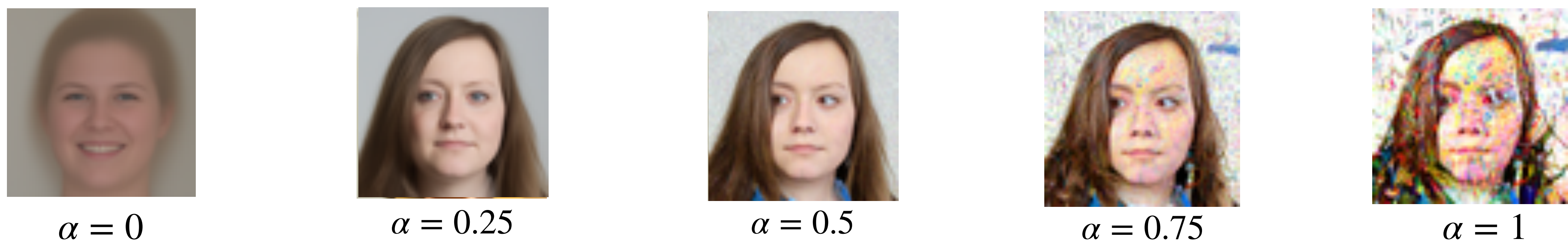
The decrease slope depends on $(\lambda_i)_{i \leq d}$:
smaller variance $\lambda_i \rightarrow$ smaller α^*

Optimization of the stochasticity parameter α

Gaussian sampling in 10 steps



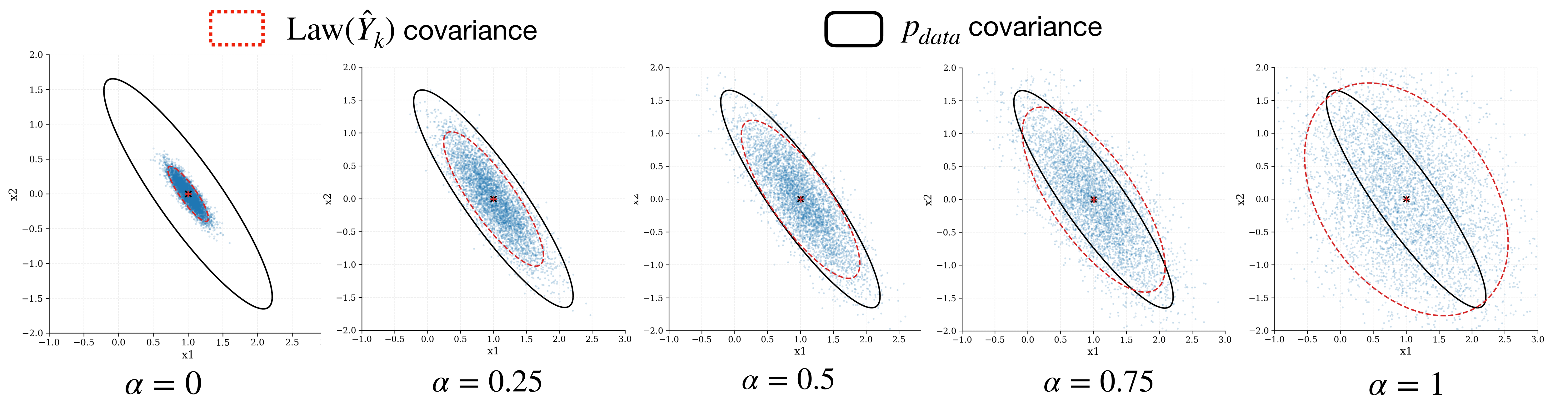
FFHQ sampling in 50 steps:



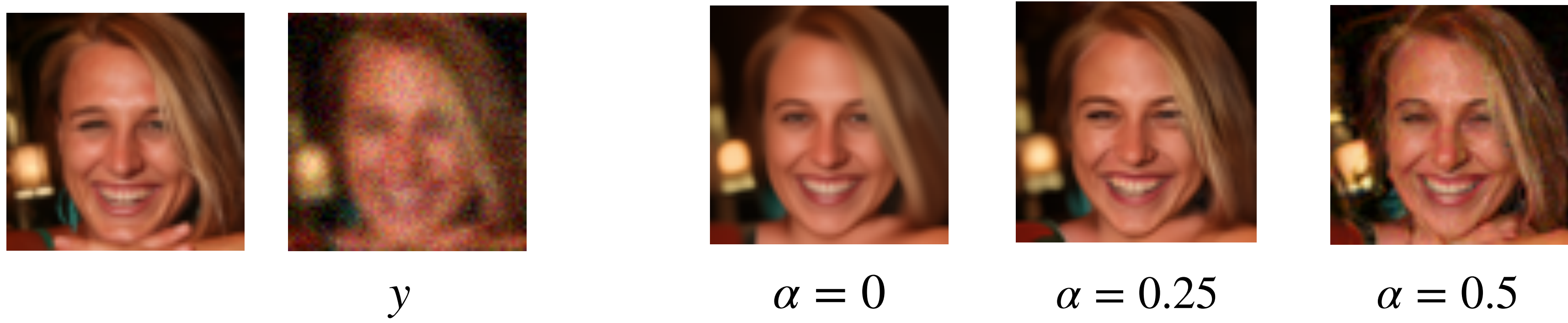
▸ For small number of steps, ODE sampling ($\alpha = 0$) samples close to the data mean

Optimization of the stochasticity parameter α

Gaussian sampling in 10 steps



Posterior sampling: sample the posterior $p(x|y)$



Discretization error allows **MMSE estimation** using diffusion sampling

▸ For small number of steps, choosing $\alpha = 0$ samples close to the **MMSE**

Conclusion

- Diffusion / Flow Matching models **state-of-the-art** for sampling images.
- Based on **image denoising**: well understood and « easy » to train.
- For application to posterior sampling **without retraining**:
requires an **approximation** of the posterior ...

Some recent research directions

- How to select parameters *with respect to the target distribution*.
- Why these models generalize ? « *they work because they fail* » [Bertrand et al, `26]
- Relation with (semi-discrete) optimal transport [Hertrich et al, `25, Pierret et al, `26]
- « **Long** » generative process: need to solve a ODE/SDE.
 - Straighter trajectories [Liu et al `23, Boîte et al, `26]
 - One-step generative models *Consistency models* [Song et al, `23] [Boffi et al, `24]
Drifting models [Deng et. al, `26]

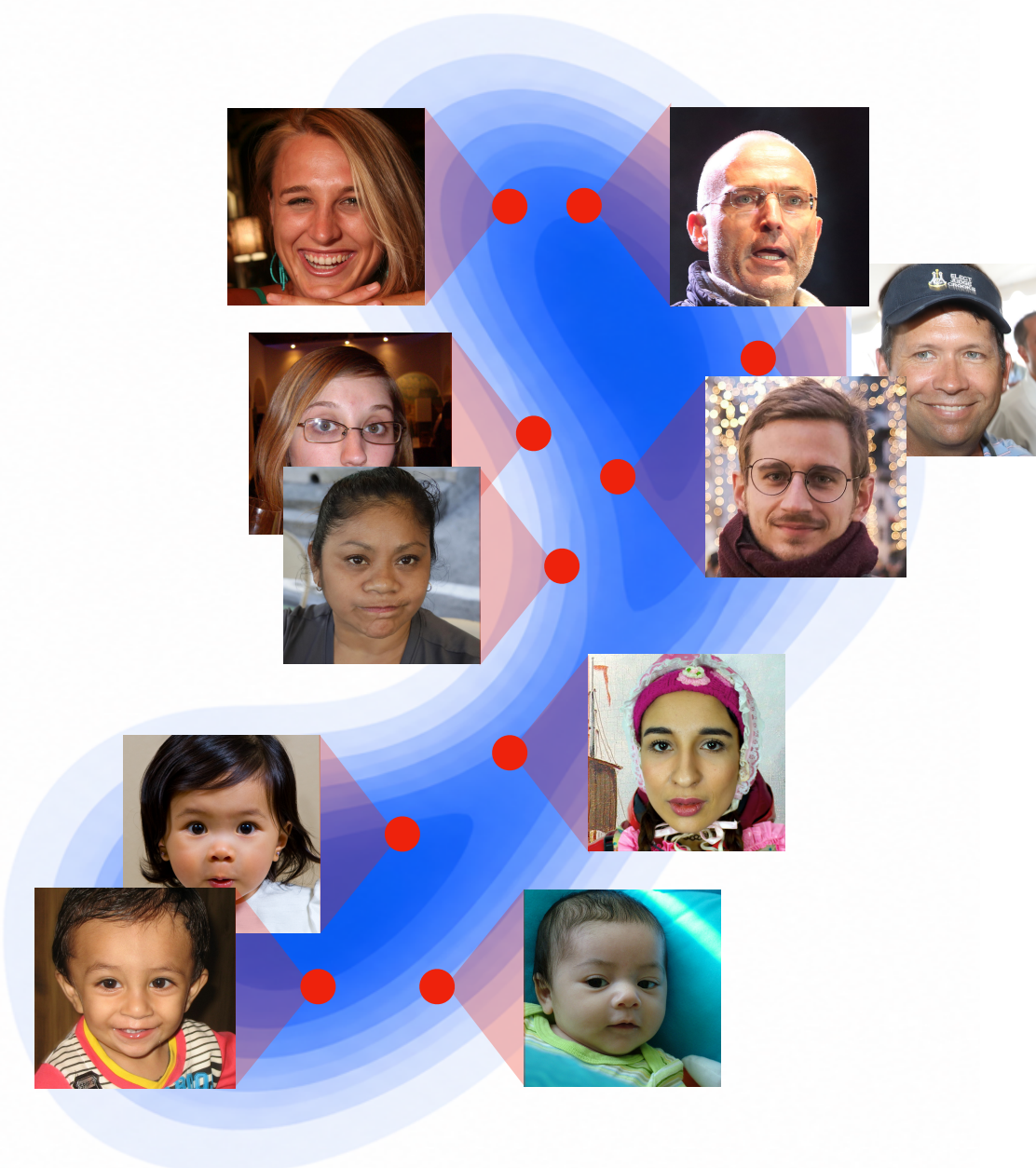
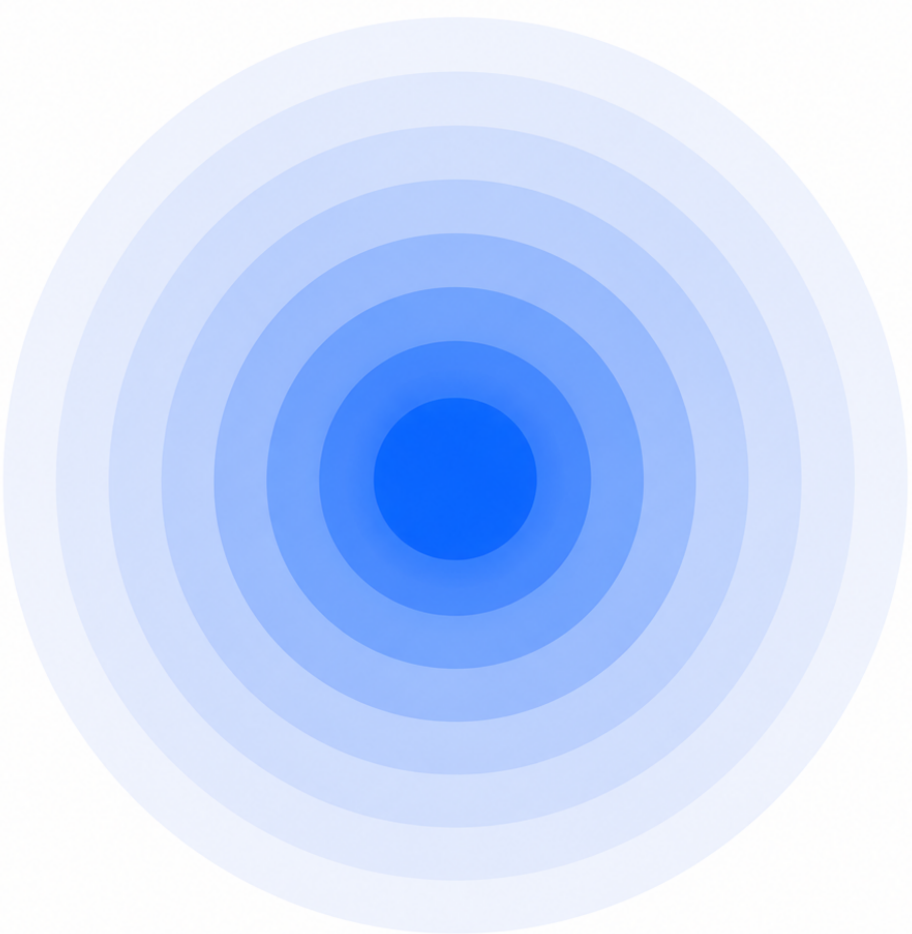
Generalization of diffusion models

In practice we only have **N samples** of p_{data} : $p_{data} \rightarrow \hat{p}_N = \frac{1}{N} \sum_{i=1}^N \delta_{x_i}$

The diffusion algorithm should re-sample training data [Scarvelis et al., 2023]

$$D_{\sigma}^*(x) = \mathbb{E}[x | y] = \int x \frac{p(y | x)}{p(y)} \hat{p}_N(x) dx = \sum_{i=1}^N x_i \text{softmax} \left(\frac{-\|y - x_i\|^2}{2\sigma^2} \right)$$

$\xrightarrow{\sigma \rightarrow 0}$ NearestNeighbor($y | \{x_i\}$)



Generalization of diffusion models

In practice we only have **N samples** of p_{data} : $p_{data} \rightarrow \hat{p}_N = \frac{1}{N} \sum_{i=1}^N \delta_{x_i}$

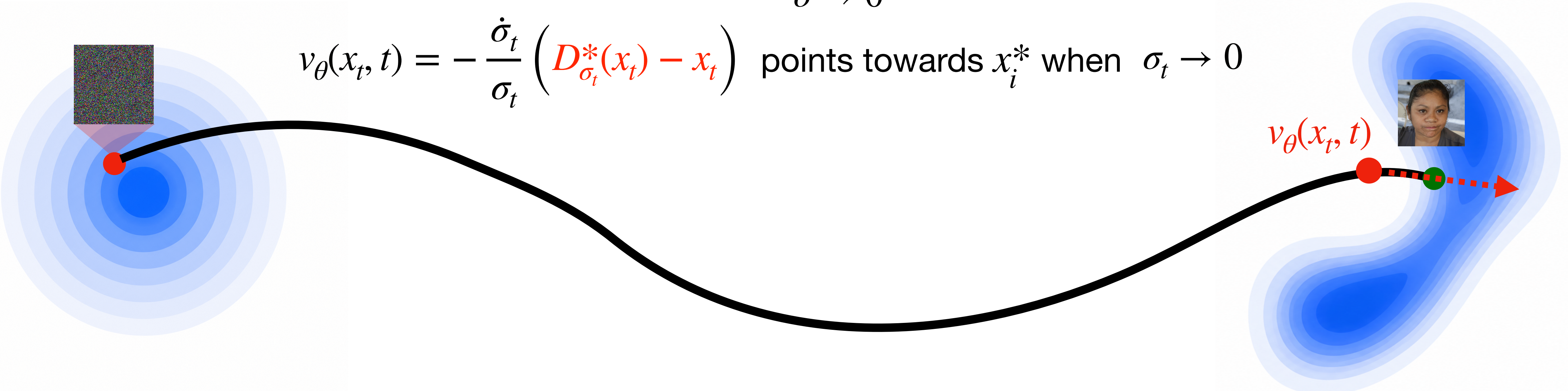
The diffusion algorithm should **sample training points** [Scarvelis et al., 2023]

Exercise: proof

$$D_{\sigma}^*(x) = \mathbb{E}[x | y] = \int x \frac{p(y | x)}{p(y)} \hat{p}_N(x) dx = \sum_{i=1}^N x_i \text{softmax} \left(\frac{-\|y - x_i\|^2}{2\sigma^2} \right)$$

$\xrightarrow{\sigma \rightarrow 0}$ NearestNeighbor($y | \{x_i\}$)

$$v_{\theta}(x_t, t) = -\frac{\dot{\sigma}_t}{\sigma_t} \left(D_{\sigma_t}^*(x_t) - x_t \right) \text{ points towards } x_i^* \text{ when } \sigma_t \rightarrow 0$$



Generalization of diffusion models

In practice we only have **N samples** of p_{data} : $p_{data} \rightarrow \hat{p}_N = \frac{1}{N} \sum_{i=1}^N \delta_{x_i}$

The diffusion algorithm should **sample training points** [Scarvelis et al., 2023]

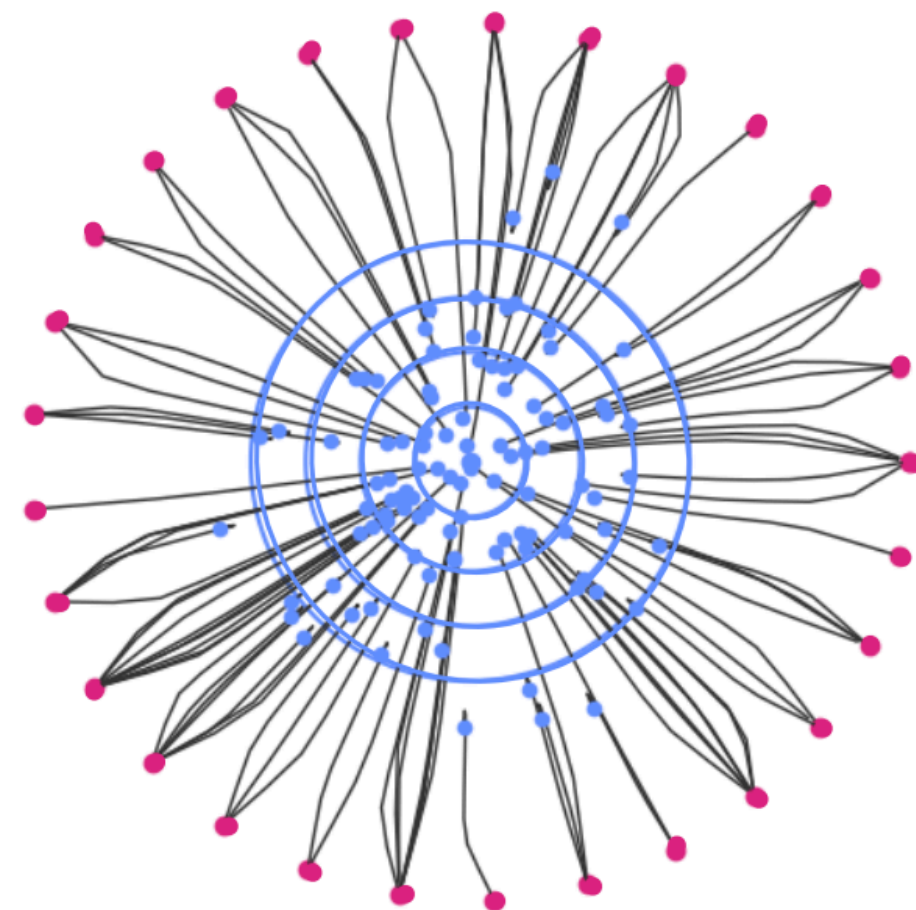
Exercise: proof

$$D_{\sigma}^*(x) = \mathbb{E}[x | y] = \int x \frac{p(y | x)}{p(y)} \hat{p}_N(x) dx = \sum_{i=1}^N x_i \text{softmax} \left(\frac{-\|y - x_i\|^2}{2\sigma^2} \right)$$

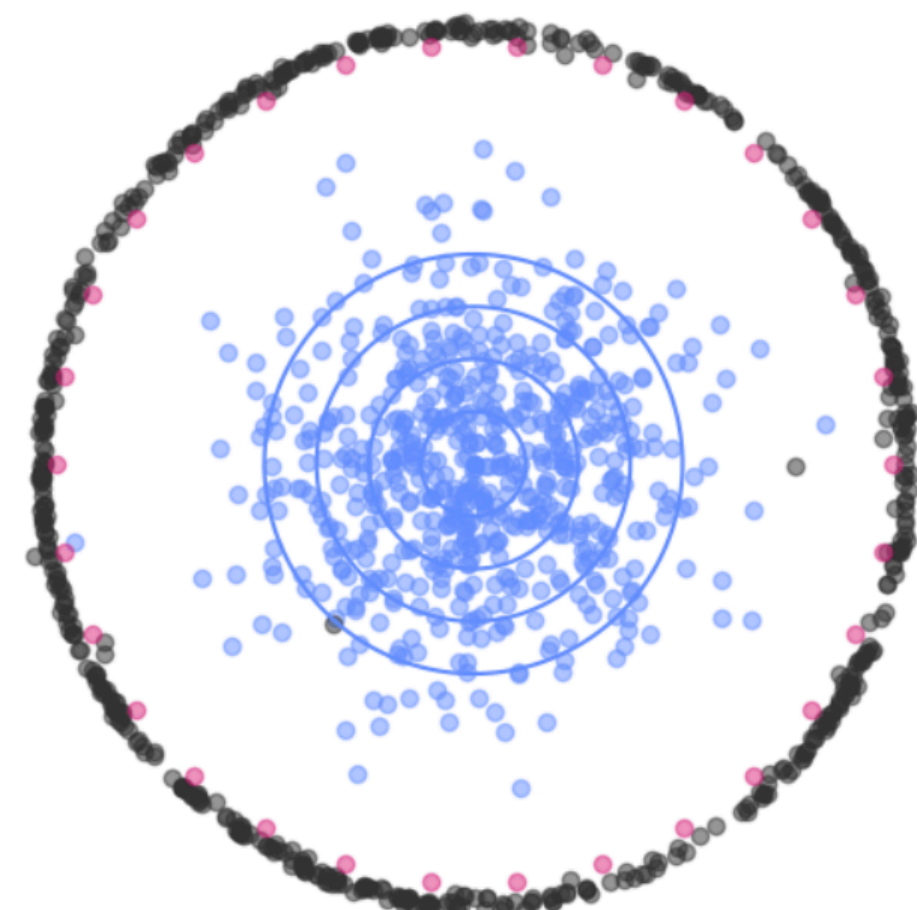
$\xrightarrow{\sigma \rightarrow 0}$ NearestNeighbor($y | \{x_i\}$)

[Bertrand et al., 2025] Generalizes thanks to denoiser **training error**

[Pierret et al., 2026]



with **true** denoiser



with **trained** denoiser

« It works because it fails »

Conclusion

- Diffusion / Flow Matching models **state-of-the-art** for sampling images.
- Based on **image denoising**: well understood and « easy » to train.
- For application to posterior sampling **without retraining**:
requires an **approximation** of the posterior ...

Some recent research directions

- How to select parameters *with respect to the target distribution*.
- Why these models generalize ? « *they work because they fail* » [Bertrand et al, `26]
- Relation with (semi-discrete) optimal transport [Hertrich et al, `25, Pierret et al, `26]
- « **Long** » generative process: need to solve a ODE/SDE.
 - Straighter trajectories [Liu et al `23, Boîte et al, `26]
 - One-step generative models *Consistency models* [Song et al, `23] [Boffi et al, `24]
Drifting models [Deng et. al, `26]

Relation with (semi-discrete) Optimal Transport

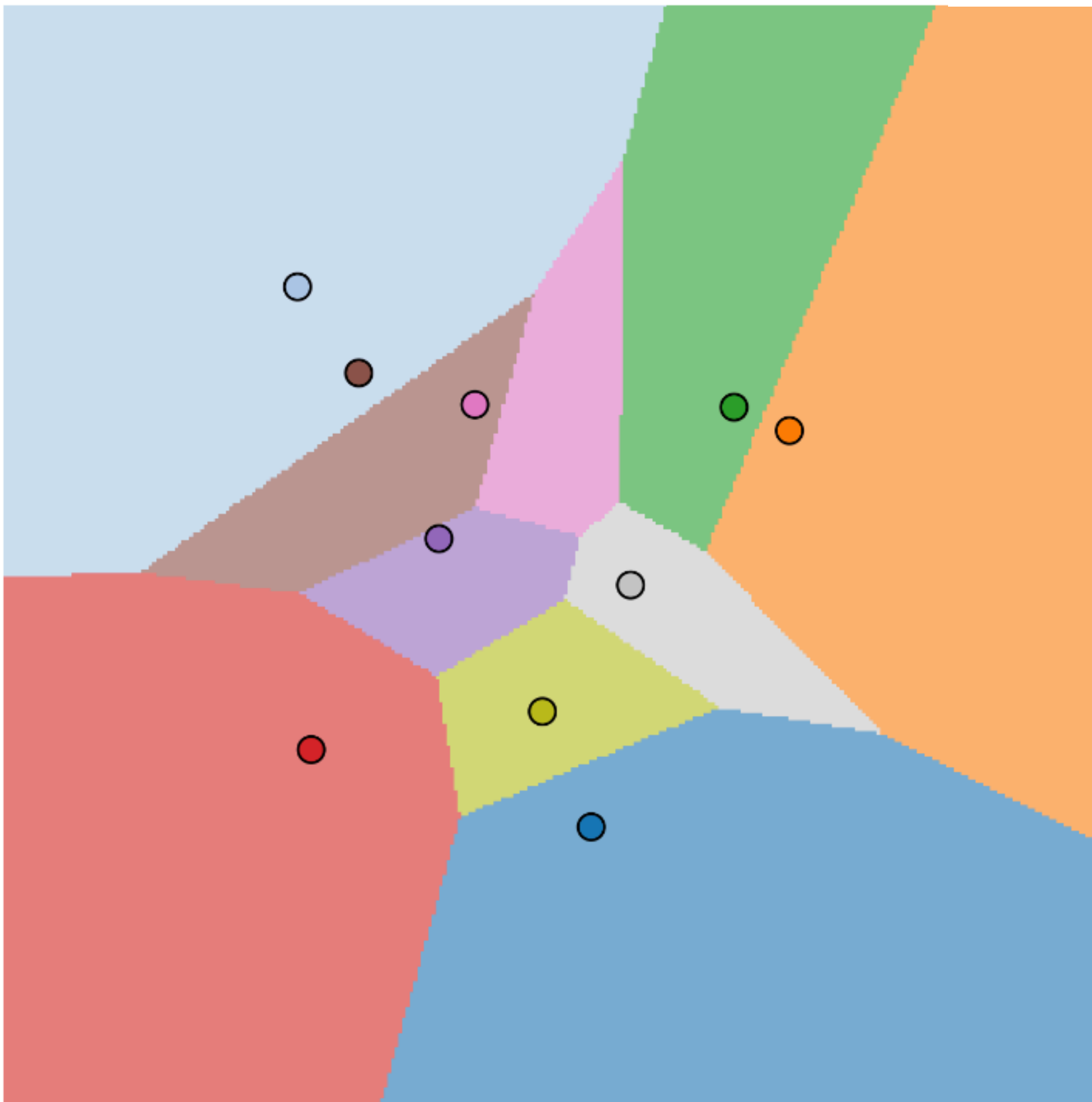
Tessellations of Semi-Discrete Flow Matching

Emile Pierret*
DMA, ENS-PSL, MAP5
`pierret@math.cnrs.fr`

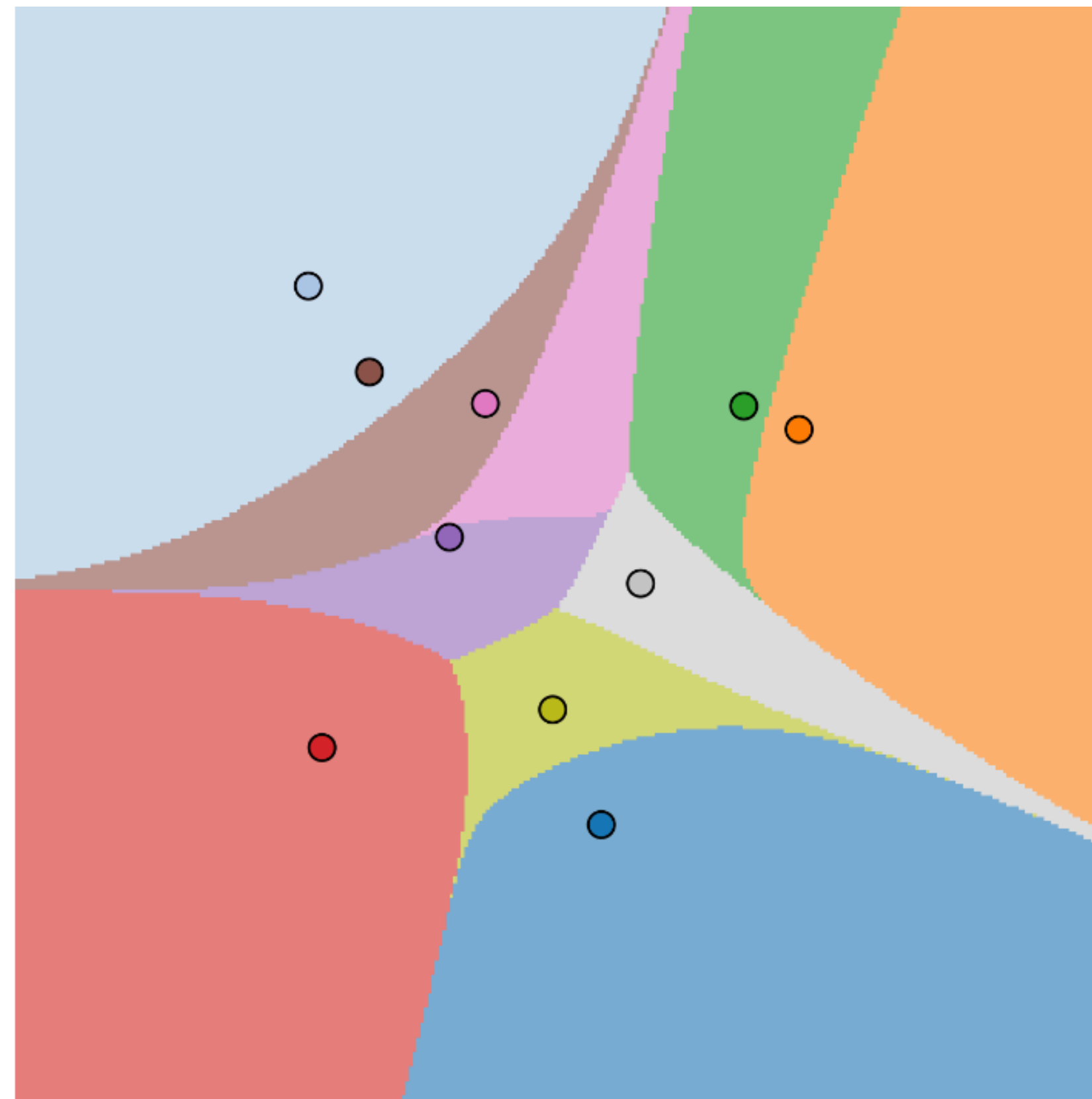
Johannes Hertrich*
DMA, ENS-PSL
`johannes.hertrich@ens.fr`

Samuel Hurault*
LIGM, Univ. Gustave Eiffel, CNRS
`amuel.hurault@univ-eiffel.fr`

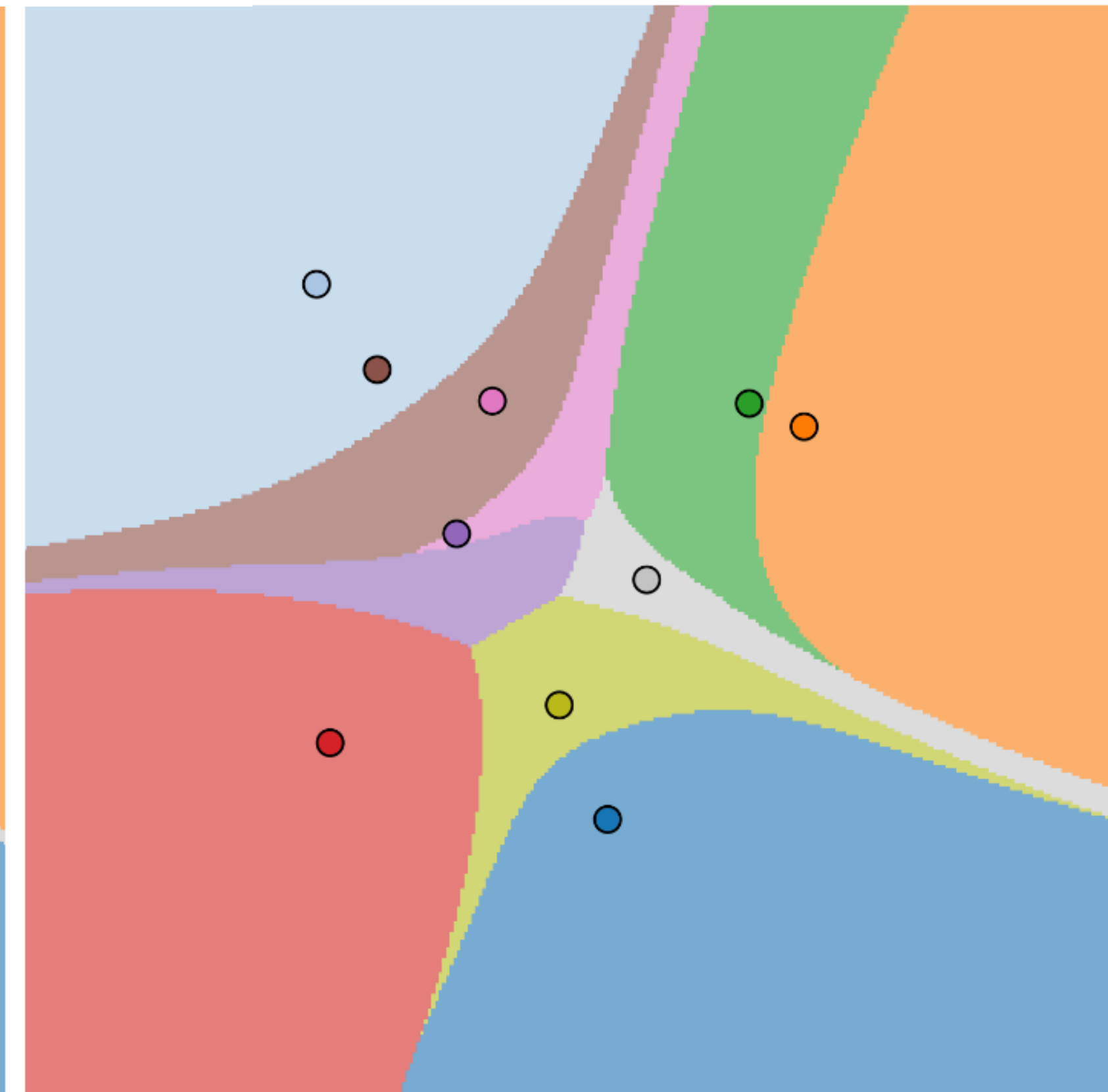
Julie Delon
DMA, ENS-PSL
`julie.delon@ens.fr`



(a) OT



(b) FM with exact velocity



(c) FM with learned velocity

Conclusion

- Diffusion / Flow Matching models **state-of-the-art** for sampling images.
- Based on **image denoising**: well understood and « easy » to train.
- For application to posterior sampling **without retraining**:
requires an **approximation** of the posterior ...

Some recent research directions

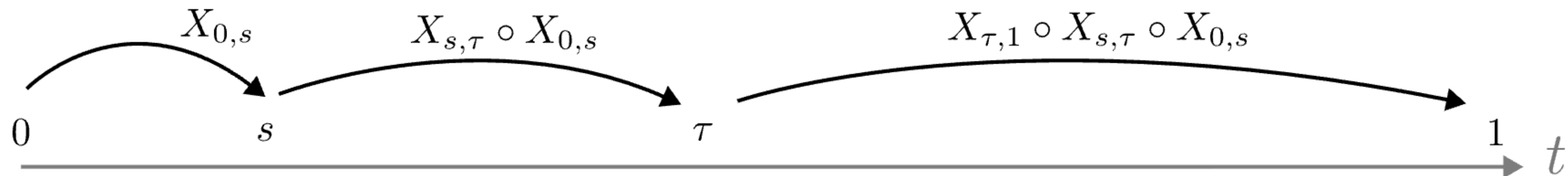
- How to select parameters *with respect to the target distribution*.
- Why these models generalize ? « *they work because they fail* » [Bertrand et al, `26]
- Relation with (semi-discrete) optimal transport [Hertrich et al, `25, Pierret et al, `26]
- « **Long** » generative process: need to solve a ODE/SDE.
 - Straighter trajectories [Liu et al `23, Boîte et al, `26]
 - One-step generative models *Consistency models* [Song et al, `23] [Boffi et al, `24]
Drifting models [Deng et. al, `26]

Consistency models [Song et al, '23] [Boffi et al, '24]

Flow map $X_{s,t}$ parametrized by a neural network conditioned on two timesteps (s, t)

$$\mathcal{L}_X = \int_0^1 \int_0^t \mathbb{E}_{x_0, x_1} \left[\left\| \partial_t X_{s,t}(x_s) - v_t(X_{s,t}(x_s)) \right\|_2^2 \right] ds dt.$$

Pretrained velocity



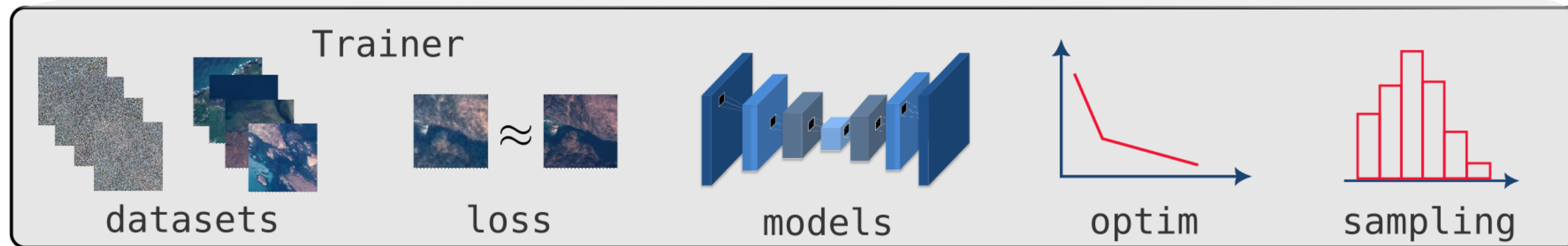
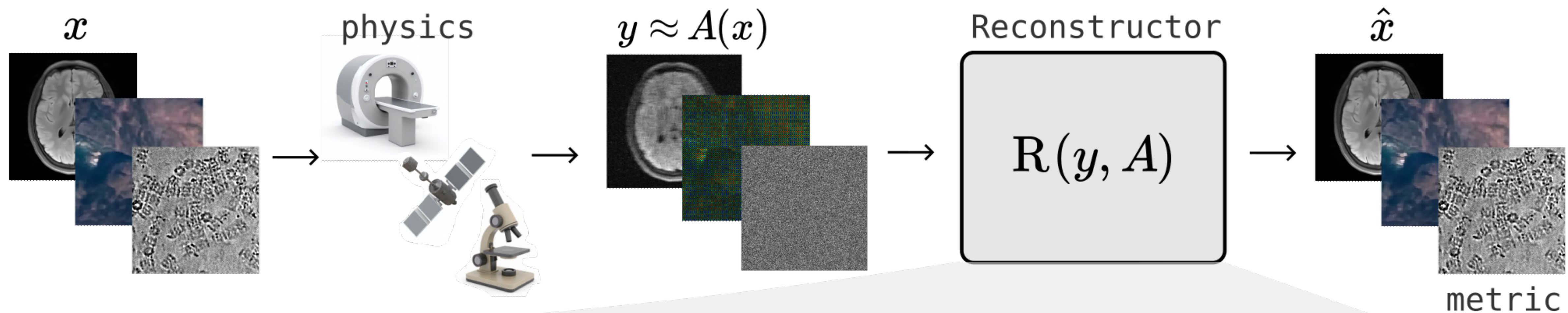
$X_{\tau,s} \circ X_{s,\tau} \circ X_{0,s}$

$X_{0,1}$

DeepInverse: a PyTorch library for imaging with deep learning



DeepInverse in a nutshell



Why using DeepInverse ?

Developing **new reconstruction methods**

- Build your algo without reinventing the wheel
- Try it out on multiple inverse problems
- Make it reproducible

Solving your **specific inverse problem**

- Quickly try SOTA methods
- Easily train or fine-tune a model

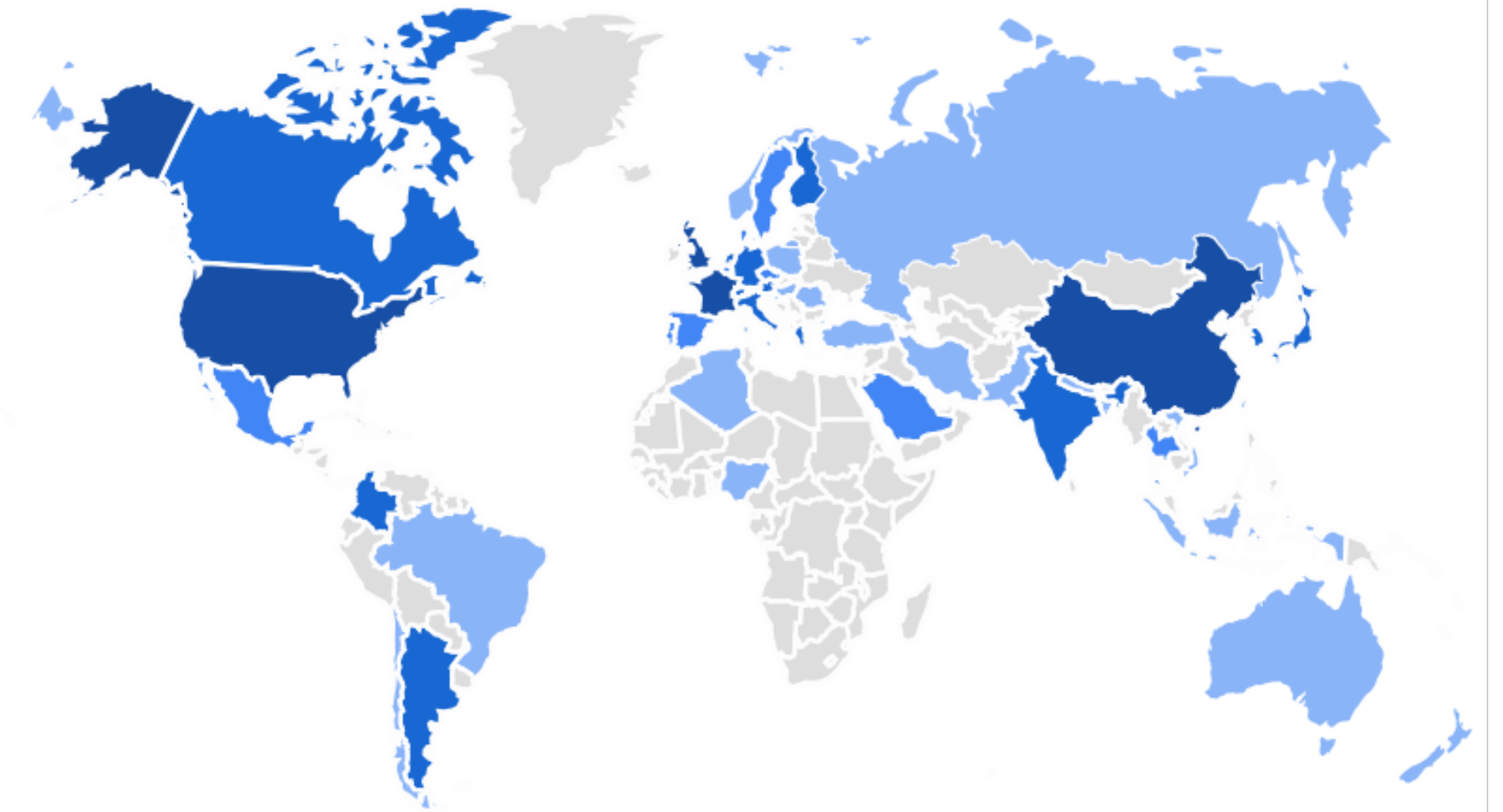
Contributing

Join us!

- Library is continuously evolving
- Community-driven effort
- All contributions (small or big) will be appropriately acknowledged

How to start?

- Chat with us
- Raise an issue in the GitHub repo
- Join the conversation in our discord channel



Community & Hackathons

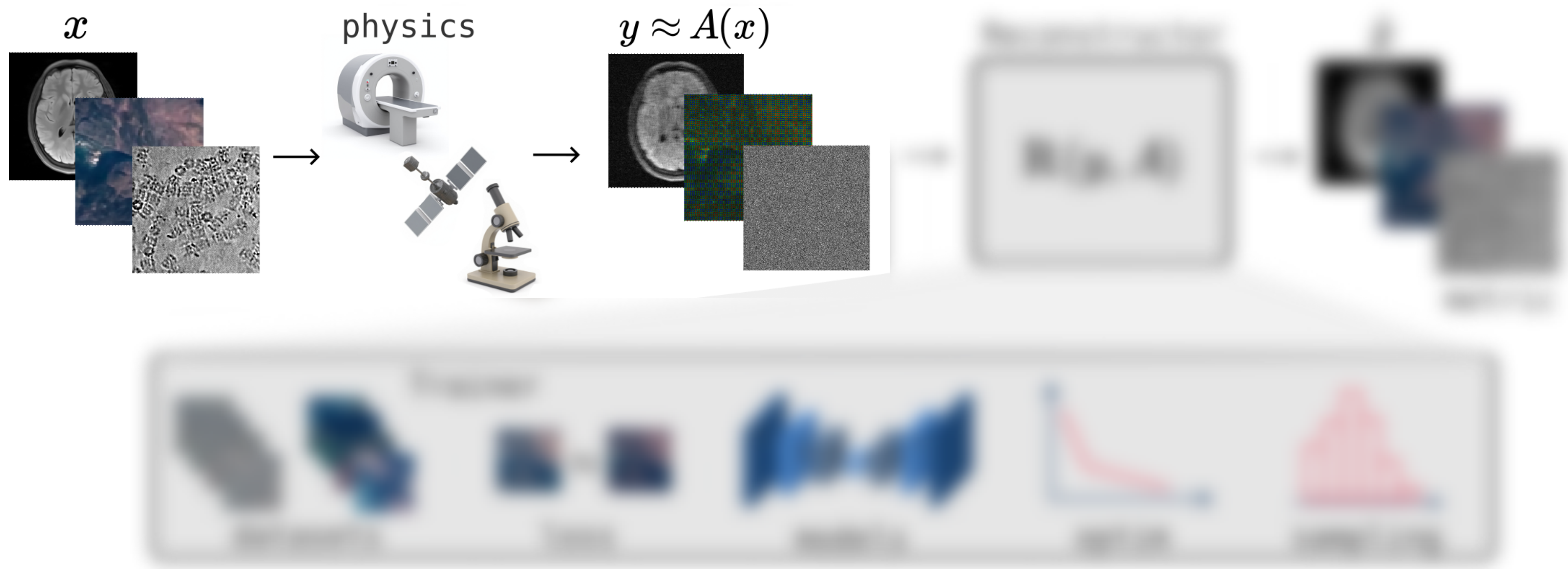
30th August - 2nd September 2026

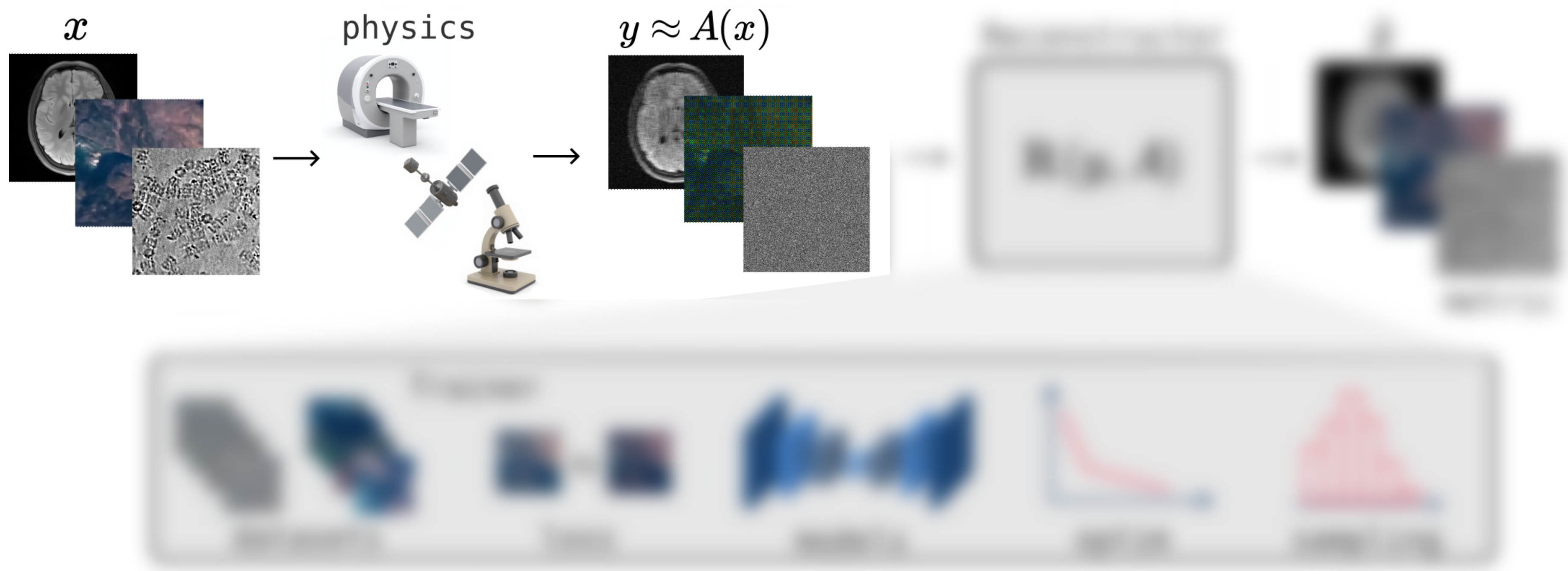


DeepInverse 2nd hackathon, Sept. 2025, Marseille



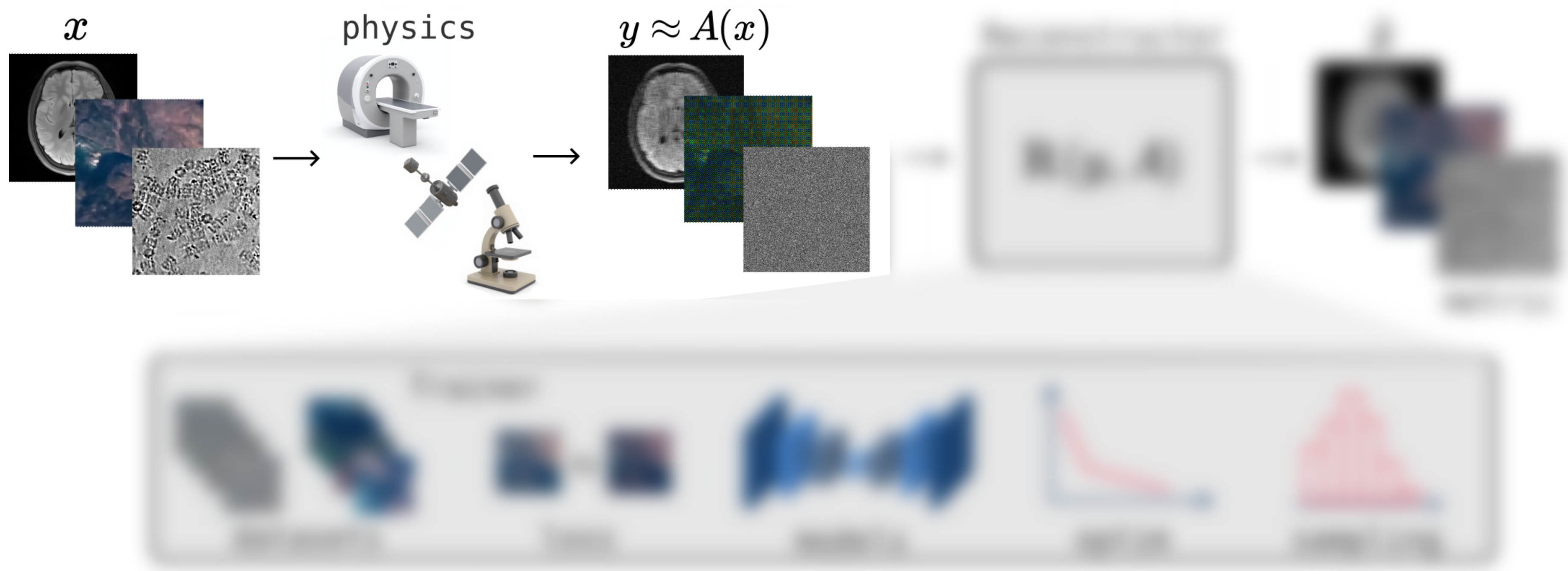
deepinv.org





Gaussian Noise $w \sim \mathcal{N}(0, \nu^2 I)$ $y = x + w$

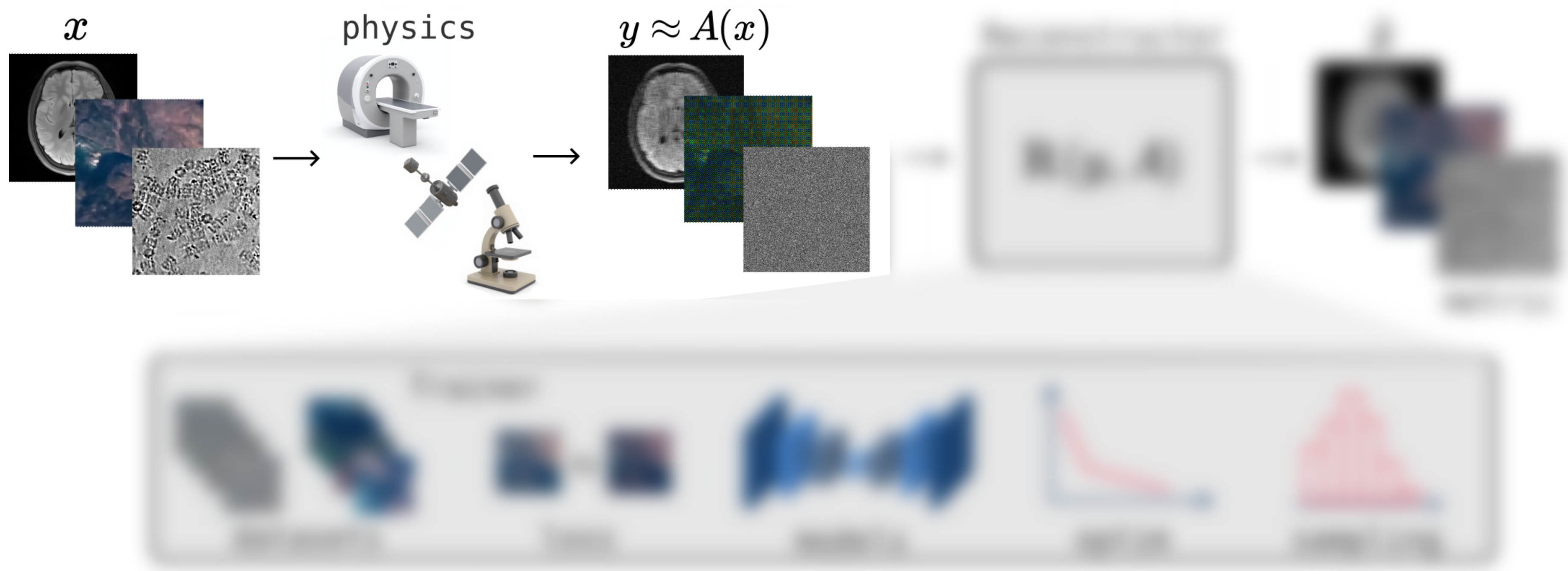
```
physics = dinv.physics.GaussianNoise(sigma=.)  
y = physics(x)
```

Blur

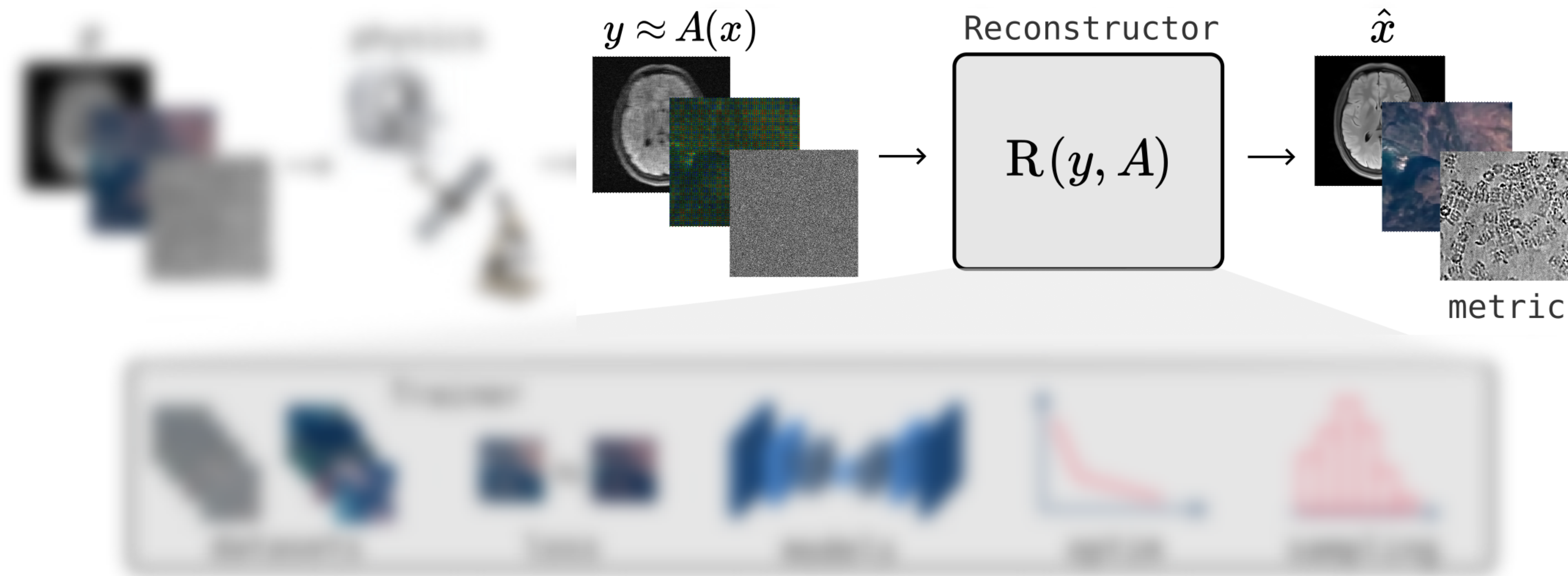
$$w \sim \mathcal{N}(0, \nu^2 I) \quad y = k \star x + w$$

```
physics = dinv.physics.Blur(
    filter=filter,
    noise_model=dinv.physics.GaussianNoise(sigma=sigma)
)
y = physics(x)
```

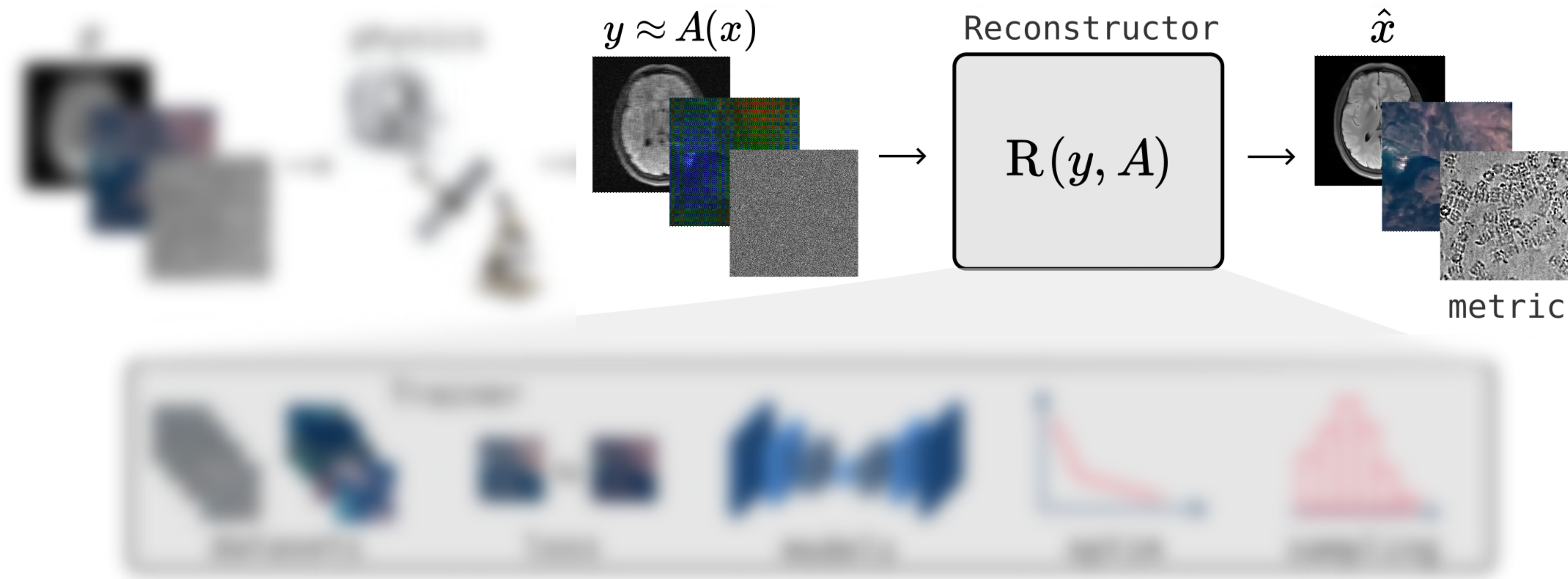



Gives access to lots of important operations :

```
physics.A_adjoint(y) # compute the adjoint operator  
physics.A_dagger(y) # compute the pseudo-inverse operator  
physics.prox_l2(x, y, gamma) # Proximal operator
```

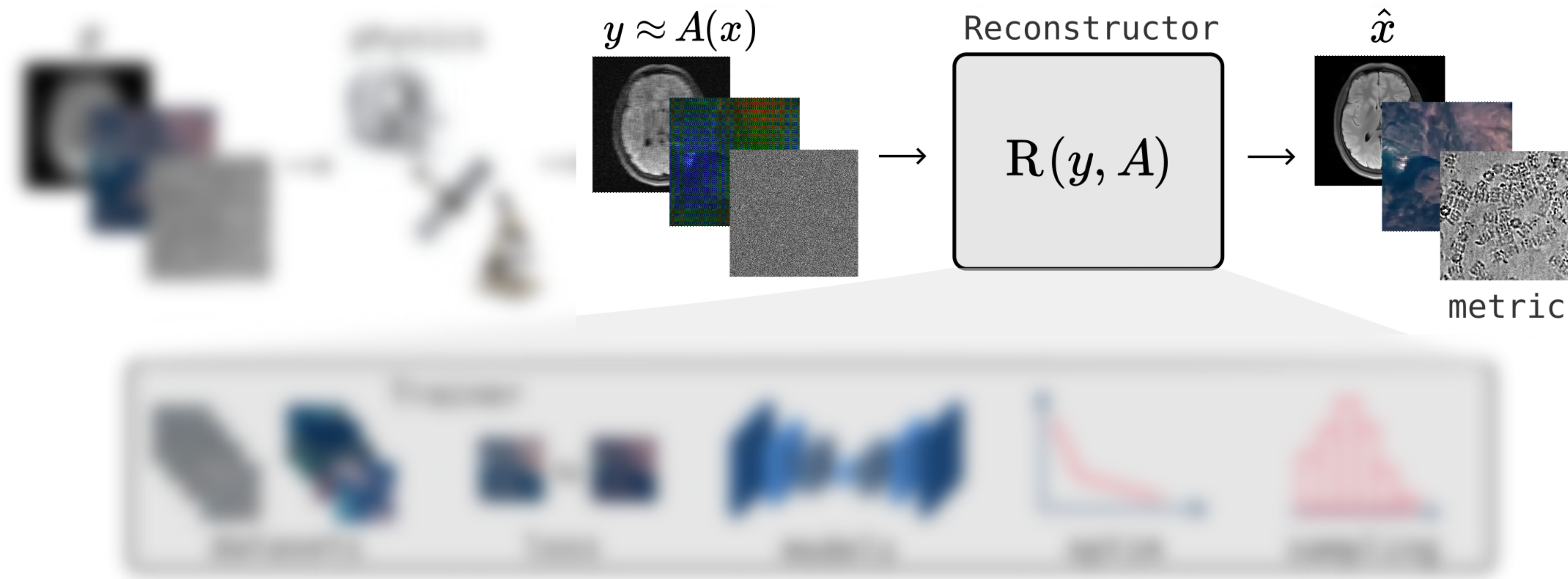



```
model = ...  
x_hat = model(y, physics)
```

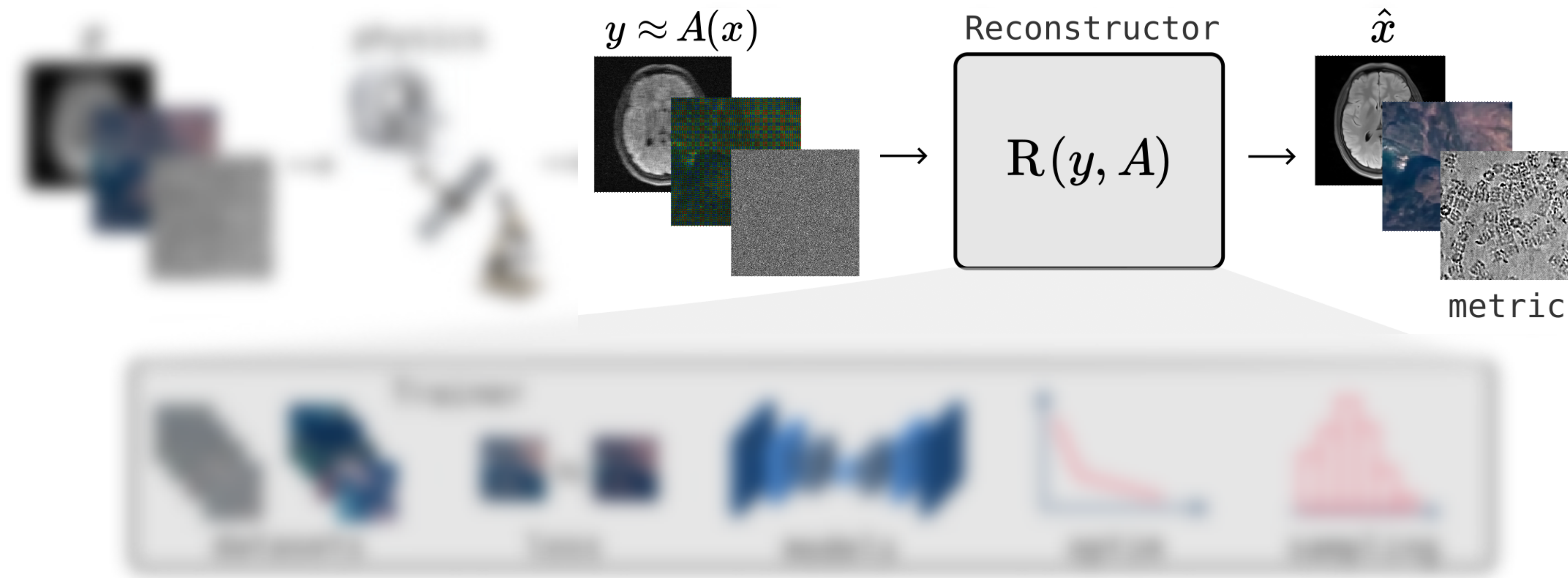
Denoisers e.g. DRUNET [Zhang et al. '21]

```
model = dinv.models.DRUNet()  
x_hat = model(y, physics)
```

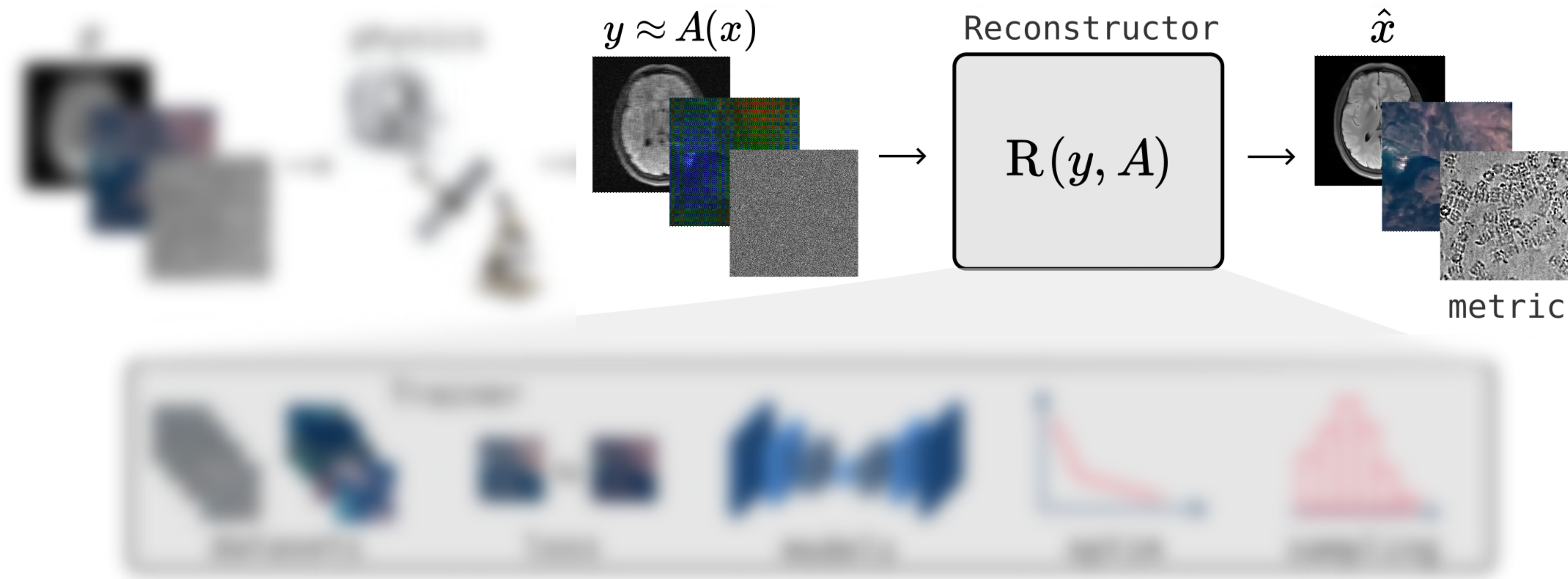
Denoisers e.g. NCSNpp [Song et al. '23]

```
model = dinv.models.NCSNpp()  
x_hat = model(y, physics)
```

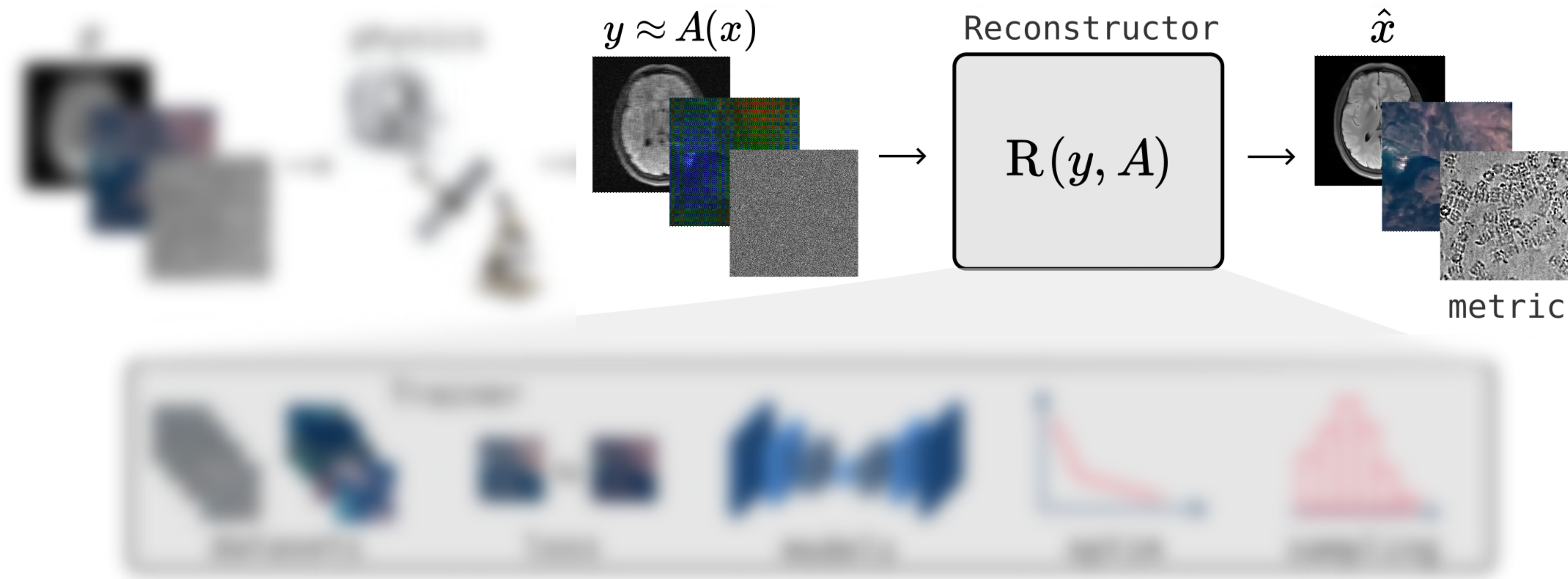
Reconstruct Anything Model (RAM) [Terris et al. '25]

```
model = dinv.models.RAM()  
x_hat = model(y, physics)
```

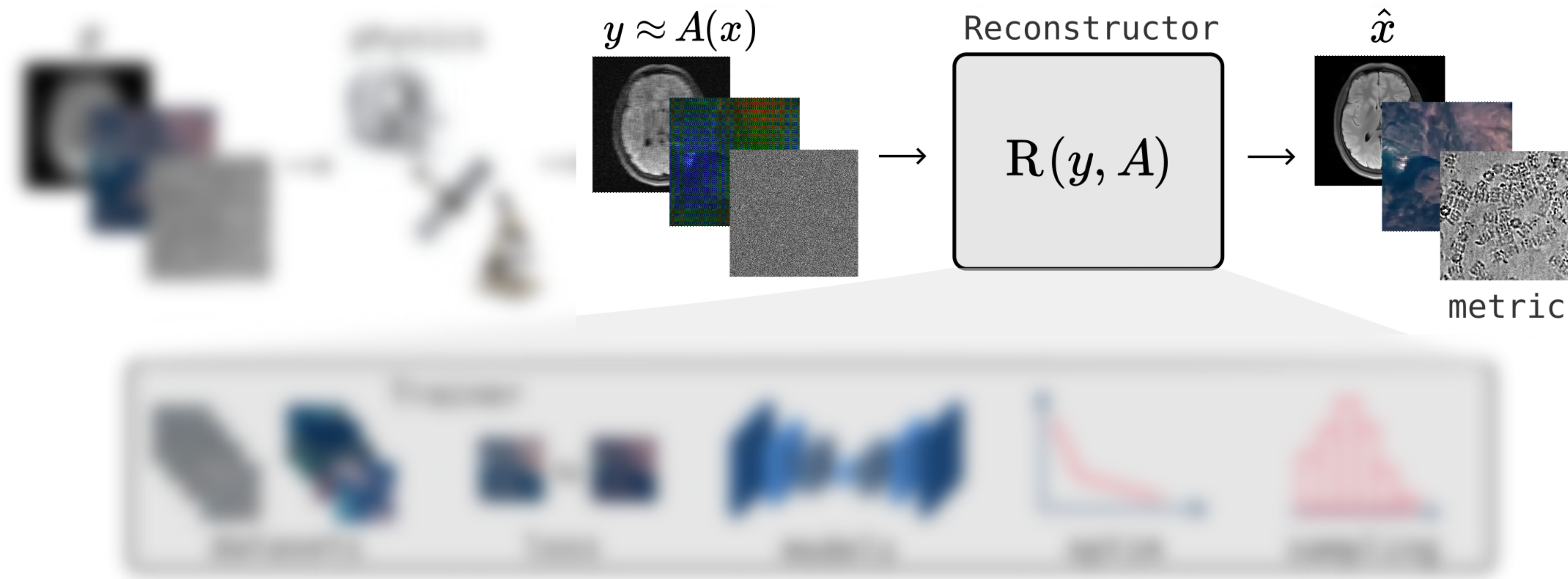
Plug-and-play reconstruction: $\operatorname{argmin}_x \text{DataFidelity}(x, y) + \text{Prior}(x)$

```
data_fid = dinv.optim.L2()  
prior = dinv.prior.PnP(denoiser = denoiser)  
model = dinv.optim.GD(  
    prior = prior,  
    data_fidelity = data_fid,  
    stepsize=1.0)
```

Diffusion-based Posterior Sampling: define **data-fidelity** / **denoiser** / **SDE** / solver

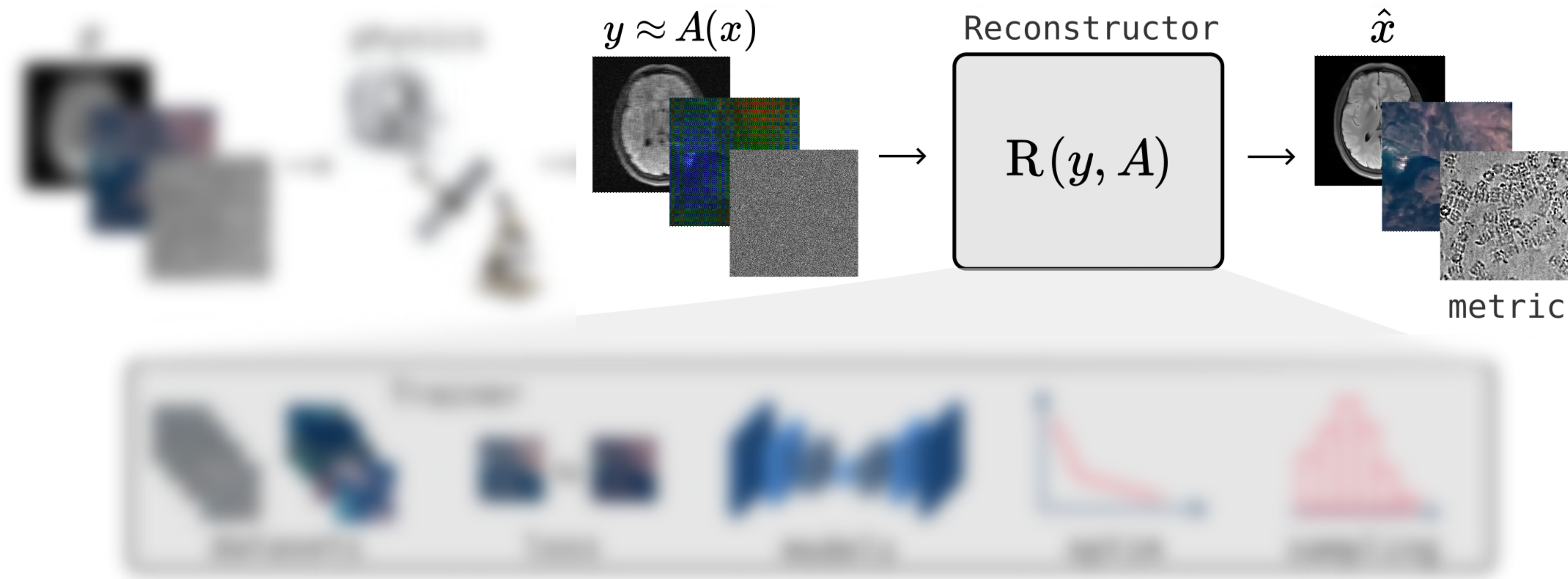
```
model = PosteriorDiffusion(data_fid, denoiser, sde, solver)
```

Diffusion-based Posterior Sampling: define **data-fidelity** / **denoiser** / **SDE** / solver

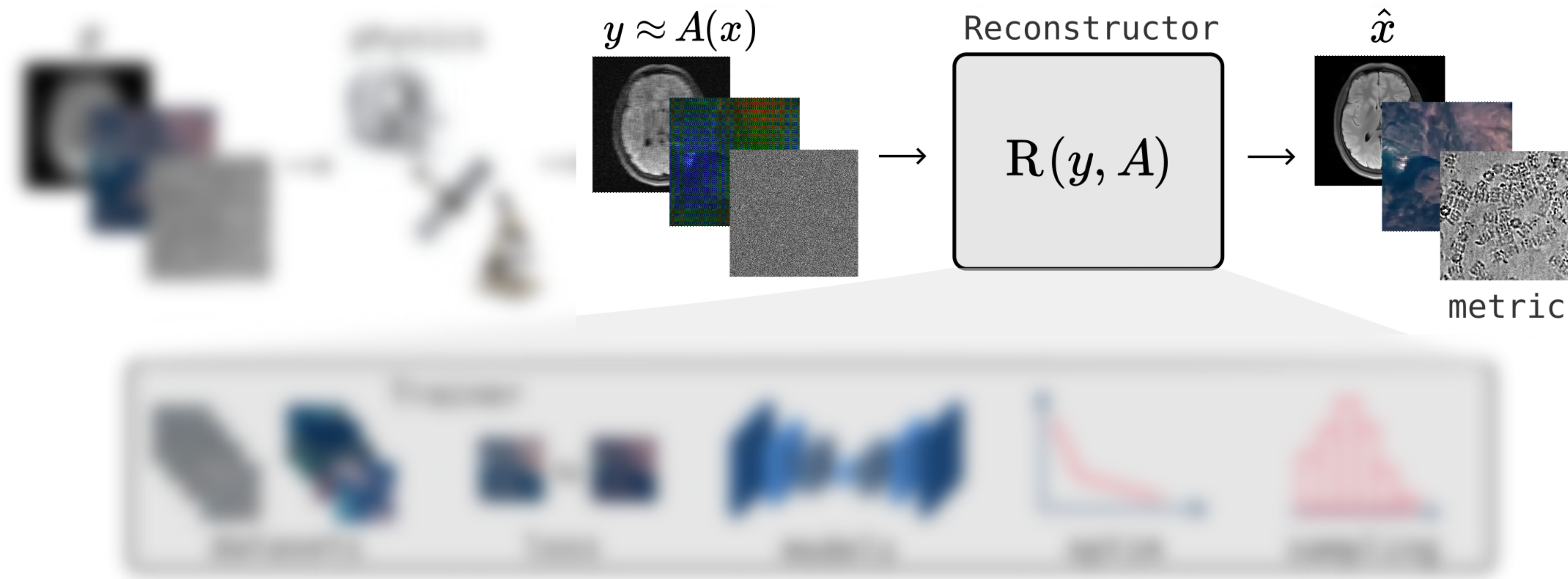
```
data_fid = dinv.sampling.DPSDataFidelity()
```

```
model = PosteriorDiffusion(data_fid, denoiser, sde, solver)
```

Diffusion-based Posterior Sampling: define **data-fidelity** / **denoiser** / **SDE** / solver

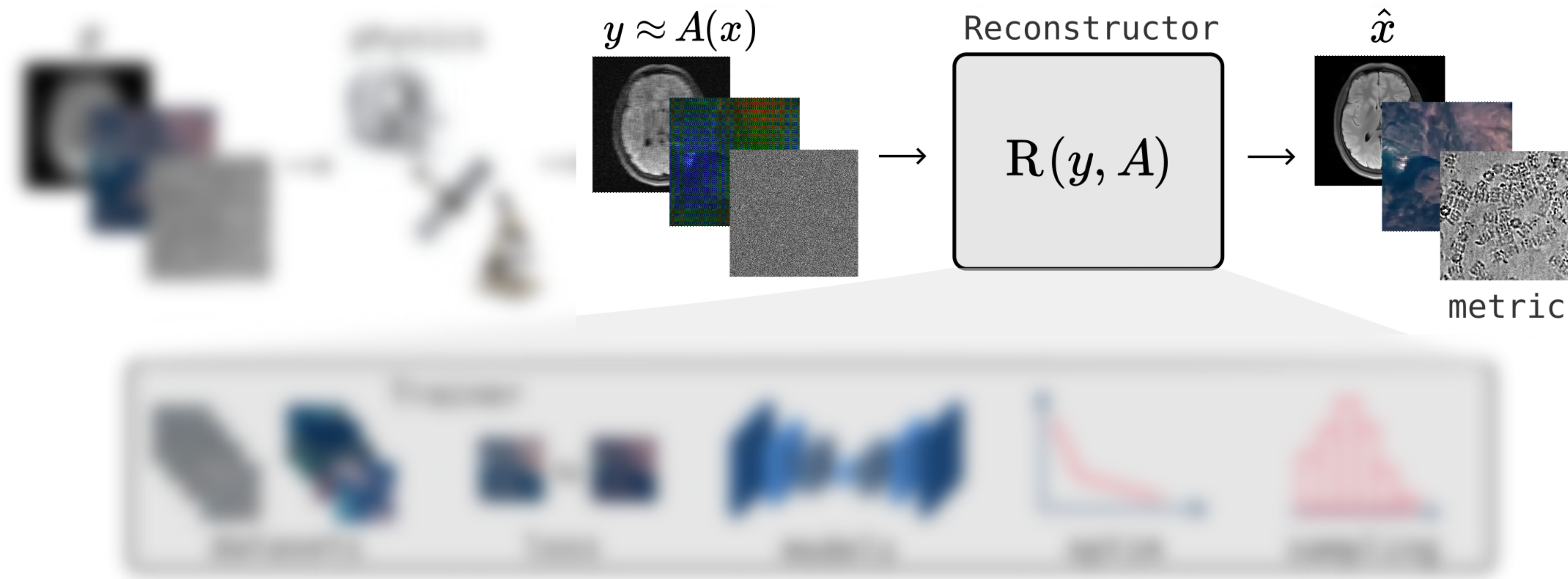
```
data_fid = dinv.sampling.DPSDataFidelity()  
sde = dinv.sampling.VarianceExplodingDiffusion(alpha=...)  
  
model = PosteriorDiffusion(data_fid, denoiser, sde, solver)
```

Diffusion-based Posterior Sampling: define **data-fidelity** / **denoiser** / **SDE** / solver

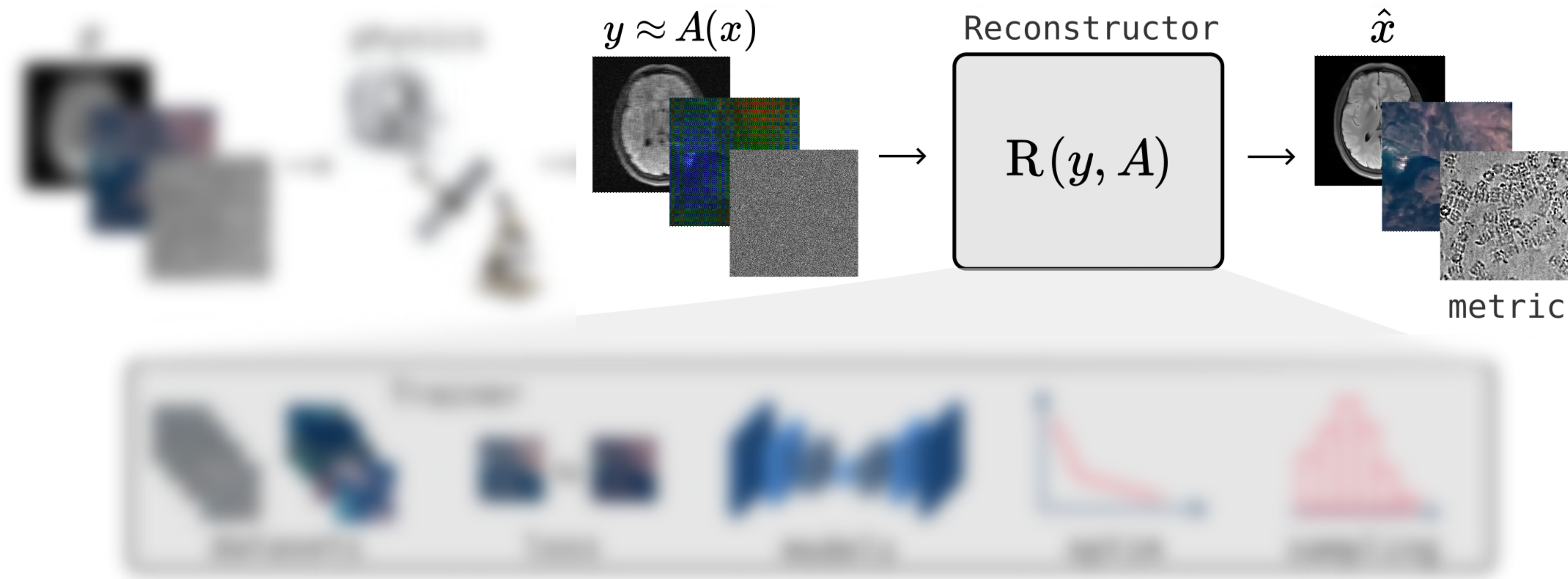
```
data_fid = dinv.sampling.DPSDataFidelity()
sde = dinv.sampling.VarianceExplodingDiffusion
      (sigma_t = lambda t : ..., sigma_prime_t = lambda t : ...)

model = PosteriorDiffusion(data_fid, denoiser, sde, solver)
```

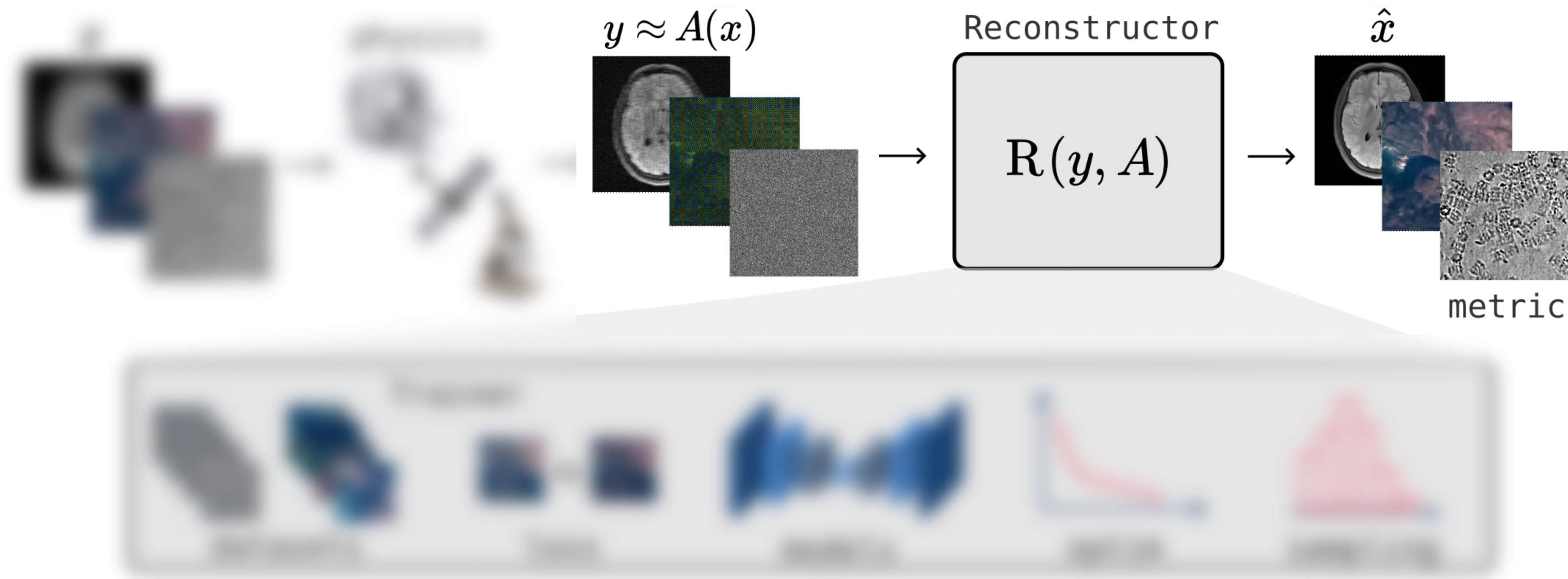
Diffusion-based Posterior Sampling: define **data-fidelity** / **denoiser** / **SDE** / solver

```
data_fid = dinv.sampling.DPSDataFidelity()  
sde = dinv.sampling.VariancePreservingDiffusion()  
  
model = PosteriorDiffusion(data_fid, denoiser, sde, solver)
```

Diffusion-based Posterior Sampling: define **data-fidelity** / **denoiser** / **SDE** / solver

```
data_fid = dinv.sampling.DPSDataFidelity()  
sde = dinv.sampling.FlowMatching()  
  
model = PosteriorDiffusion(data_fid, denoiser, sde, solver)
```

Diffusion-based Posterior Sampling: define **data-fidelity** / **denoiser** / **SDE** / solver

```
data_fid = dinv.sampling.DPSDataFidelity()  
sde = dinv.sampling.FlowMatching()  
timesteps = torch.linspace(1.0, 0.001, num_steps)  
solver = dinv.sampling.EulerSolver(timesteps)  
model = PosteriorDiffusion(data_fid, denoiser, sde, solver)
```


Let's get coding !

Miniproject: building new noisy data-fidelity terms for posterior sampling with diffusion models

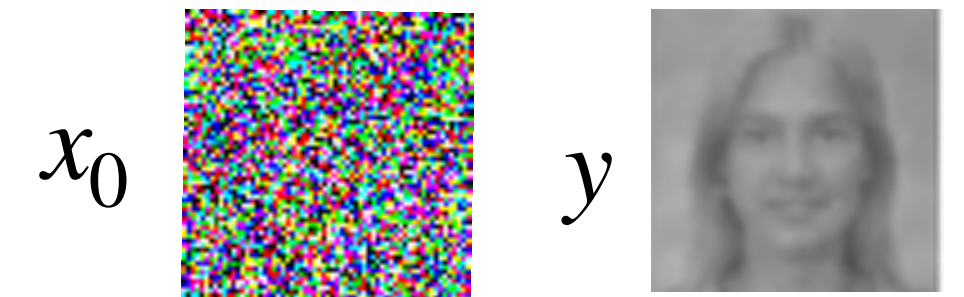
Among the diffusion-based posterior sampling methods presented in class, the library only has « DPS »

Objective: implement a new « noisy data-fidelity » instance for posterior sampling with diffusion models

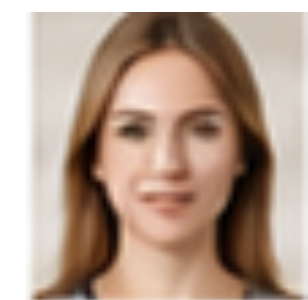
i.e. Score-ALD, DPS, PiGDM, Moment Matching or another from [\[Daras et al, 24\]](#)

🎯 **your own PR in deepinv**

Conditional Diffusion Sampling:



$$dx_t = v_t(x_t | y) dt$$



Conditional sampling of $p(x | y)$

Third idea: approximation of the conditional score

$$\nabla \log p_{\sigma_t}(x_t | y) = \nabla \log p_{\sigma_t}(y | x_t) + \nabla \log p_{\sigma_t}(x_t)$$

Score-ALD

[Jalal et. al, '21]:

$$p_{\sigma_t}(y | x_t) \approx p(y | x_t)$$

$$\nabla p_{\sigma_t}(y | x_t) \propto -A^\top (y - Ax_t)$$

DPS [Chung et. al, '23]:

$$p_{\sigma_t}(y | x_t) \approx \mathcal{N}(AD_{\sigma_t}^\theta(x_t), \nu^2 \text{Id})$$

$$\nabla p_{\sigma_t}(y | x_t) \propto -\nabla D_{\sigma_t}^\theta(x_t)^\top A^\top (y - AD_{\sigma_t}^\theta(x_t))$$

PiGDM [Song et. al, '23]:

$$p_{\sigma_t}(y | x_t) \approx \mathcal{N}(AD_{\sigma_t}^\theta(x_t), r_t^2 AA^\top + \nu^2 \text{Id})$$

$$\nabla p_{\sigma_t}(y | x_t) \propto -\nabla D_{\sigma_t}^\theta(x_t)^\top A^\top (r_t^2 AA^\top + \nu^2 \text{Id})^{-1} (y - AD_{\sigma_t}^\theta(x_t))$$

Moment Matching

[Rozet et. al, '24]:

$$p_{\sigma_t}(y | x_t) \approx \mathcal{N}(AD_{\sigma_t}^\theta(x_t), A \nabla D_{\sigma_t}^\theta(x_t)^\top A^\top + \nu^2 \text{Id})$$

$$\nabla p_{\sigma_t}(y | x_t) \propto -\nabla D_{\sigma_t}^\theta(x_t)^\top A^\top (\sigma_t^2 A \nabla D_{\sigma_t}^\theta(x_t)^\top A^\top + \nu^2 \text{Id})^{-1} (y - AD_{\sigma_t}^\theta(x_t))$$

Error vector in y-space

Preconditioner

re-scaling

+ more in [Daras et al, 24]

Back up slides

Generative models

given samples of p_{data} , generate new samples of p_{data}

Flow-based methods:

→ Choose p_0 easy to sample

→ Find a map $T_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^d$ such that

$x_0 \sim p_0, x = T_\theta(x_0)$ samples p_{data}

$$p_\theta := T_\theta \# p_0 \approx p_{data}$$

Train a neural network T_θ to minimize

$$\min_{\theta} D(p_{data}, p_\theta)$$

GANs [Goodfellow, '14)]

$$\min_{\theta} JS(p_{data}, p_\theta)$$

Variational form:

$$JS(P, Q) = \frac{1}{2} \sup_D \left[\mathbb{E}_{x \sim P} \log D(x) + \mathbb{E}_{x \sim Q} \log(1 - D(x)) \right] + cst$$

Parametrize D by a second neural network (discriminator) D_ϕ and training of parameters (θ, ϕ)

$$\min_{\theta} \max_{\phi} \left[\mathbb{E}_{x \sim p_{data}} \log D_\phi(x) + \mathbb{E}_{x \sim p_0} \log(1 - D_\phi(T_\theta(x))) \right]$$

Extended to all D with variational forms: f -GAN, WGAN, MMD-GAN

✓ First « working » generative models for images

✗ Unstable training (oscillations, mode collapse)

Generative models given samples of p_{data} , generate new samples of p_{data}

Flow-based methods:

→ Choose p_0 easy to sample

→ Find a map $T_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^d$ such that

$x_0 \sim p_0, x = T_\theta(x_0)$ samples p_{data}

$$p_\theta := T_\theta \# p_0 \approx p_{data}$$

First idea: train a neural network T_θ to minimize

$$\min_{\theta} D(p_{data}, p_\theta)$$

Normalizing Flows

[Rezende, & Mohamed, '15)]

$$\min_{\theta} KL(p_{data}, p_\theta) = \min_{\theta} \mathbb{E}_{x \sim p_{data}} [-\log p_\theta(x)]$$

Change of variable formula:

$$\log p_\theta(x) = \log p_0(T_\theta^{-1}(x)) + \log \left| \det \frac{dT_\theta^{-1}(x)}{dx} \right|$$

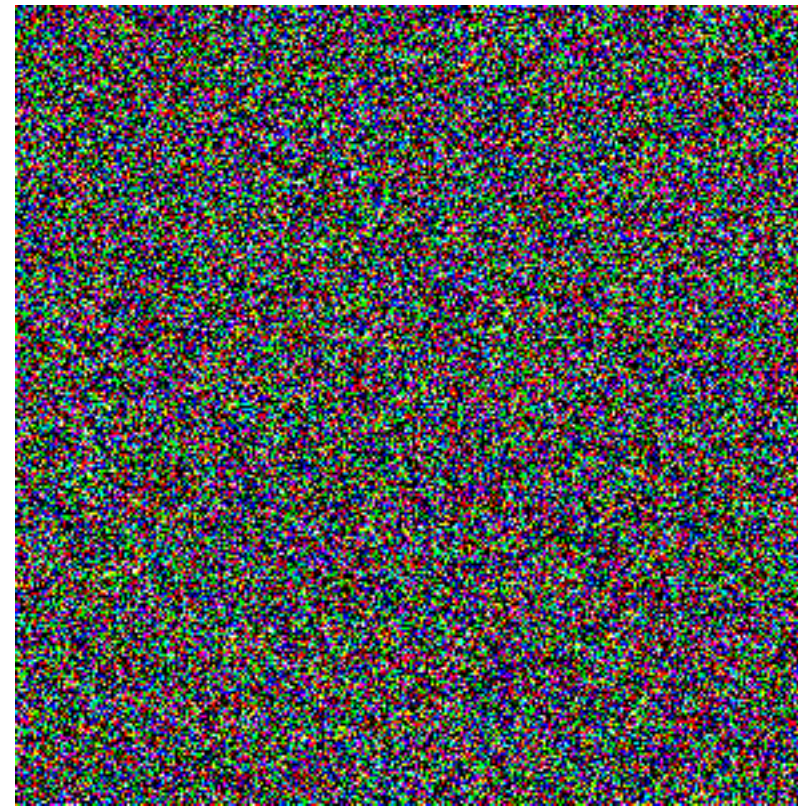
✗ Requires to parametrize T_θ to be invertible

Denoising: to go further (MMSE = proximal operator)



$$x \sim p_{\text{data}}$$

$+ \sigma \cdot$



$$w \sim \mathcal{N}(0, \text{Id})$$

$=$



$$y = x + \sigma w$$

Denoising

Tweedie formula: [Efron, '11]

$$D_{\sigma}^*(y) = \mathbb{E}[x | y] = y + \sigma^2 \nabla \log p_{\sigma}(y)$$

$$= \nabla_y \left(\frac{1}{2} \|y\|^2 + \sigma^2 \log p_{\sigma}(y) \right)$$

$$= \nabla_y \left(\text{convex function} \right)$$

[Gribonval, '11] There is $\psi_{\sigma} : \mathbb{R}^d \rightarrow \mathbb{R}^d$ s.t.

$$D_{\sigma}^*(y) = \text{Prox}_{\psi_{\sigma}}(y)$$

Brenier theorem:

D_{σ}^* is an **optimal transport** map

Conditional sampling of $p(x|y)$

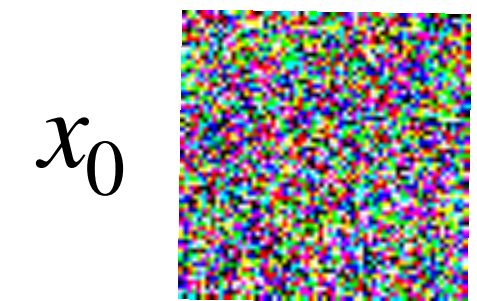
Second idea: add a « measurement consistency step »

$$p(x|y) \propto p(y|x) p_{data}(x)$$

- Start from the diffusion with **unconditional** denoiser

Denoising step $\approx$ gradient descent step on p_{data}

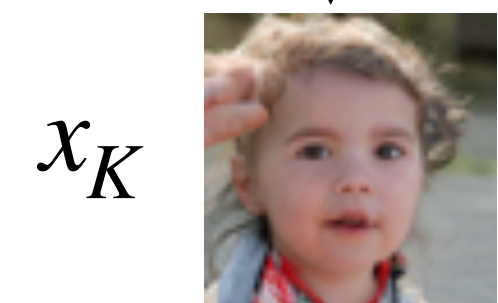
Unconditional Diffusion Sampling:



For $k = 1 \dots K$:

$$\hat{x} = D_{\sigma_{t_k}}^{\theta}(x_k)$$

$$x_{k+1} = (1 - \beta_k)x_k + \beta_k \hat{z}$$



Conditional sampling of $p(x|y)$

Second idea: add a « measurement consistency step »

$$p(x|y) \propto p(y|x) p_{data}(x)$$

- Start from the diffusion with **unconditional** denoiser

Denoising step $\approx$ gradient descent step on p_{data}

- At each iteration, ensure consistency to the observation y

DiffPIR [Zhu et. al, '23] : descent step on $p(y|x)$

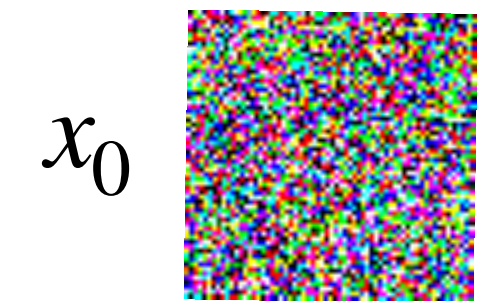
✓ Does not require to train a specific denoiser

✗ **Empirical:** no guarantee to sample $p(x|y)$

DiffPIR [Zhu et. al, '23]

ReSample [Song et. al, '24]

Conditional Diffusion Sampling:

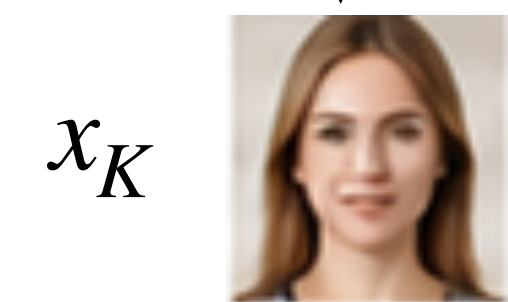


For $k = 1 \dots K$:

$$\hat{x} = D_{\sigma_{t_k}}^{\theta}(x_k)$$

$$\hat{z} \leftarrow \text{data-consistency}(\hat{x}, y)$$

$$x_{k+1} = (1 - \beta_k)x_k + \beta_k \hat{z}$$



y

Each image i is a point in $x_i \in \mathbb{R}^d$

Assumption:

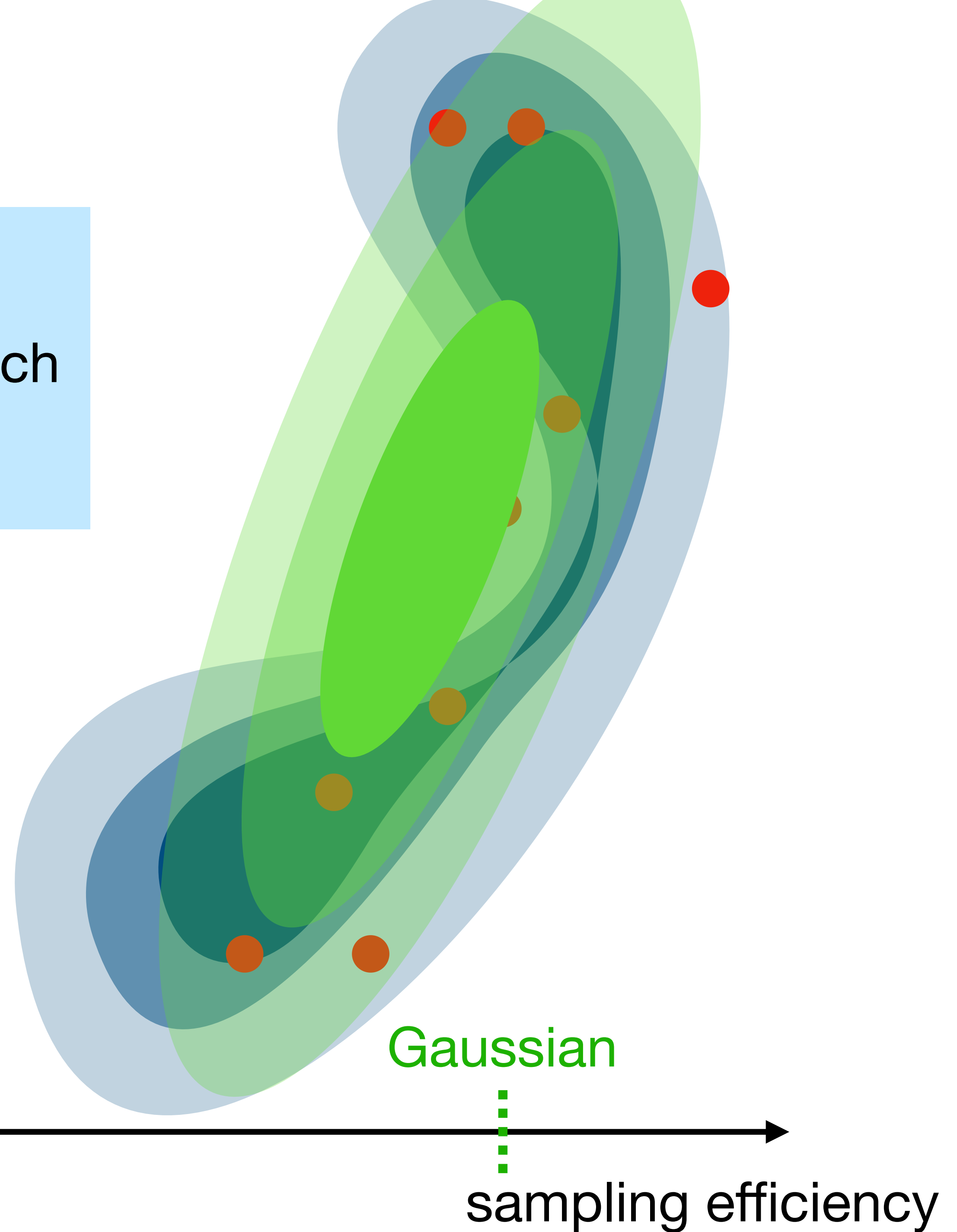
There exists a probability distribution p_{data} such that each image $x_i \sim p_{\text{data}}$

Objective:

Approximate p_{data} with a parametric distribution p_{θ} « easy to sample »

Trade-off:

← approximation quality $p_{\theta} \approx p_{\text{data}}$ sampling efficiency →



Each image i is a point in $x_i \in \mathbb{R}^d$

Assumption:

There exists a probability distribution p_{data} such that each image $x_i \sim p_{\text{data}}$

Objective:

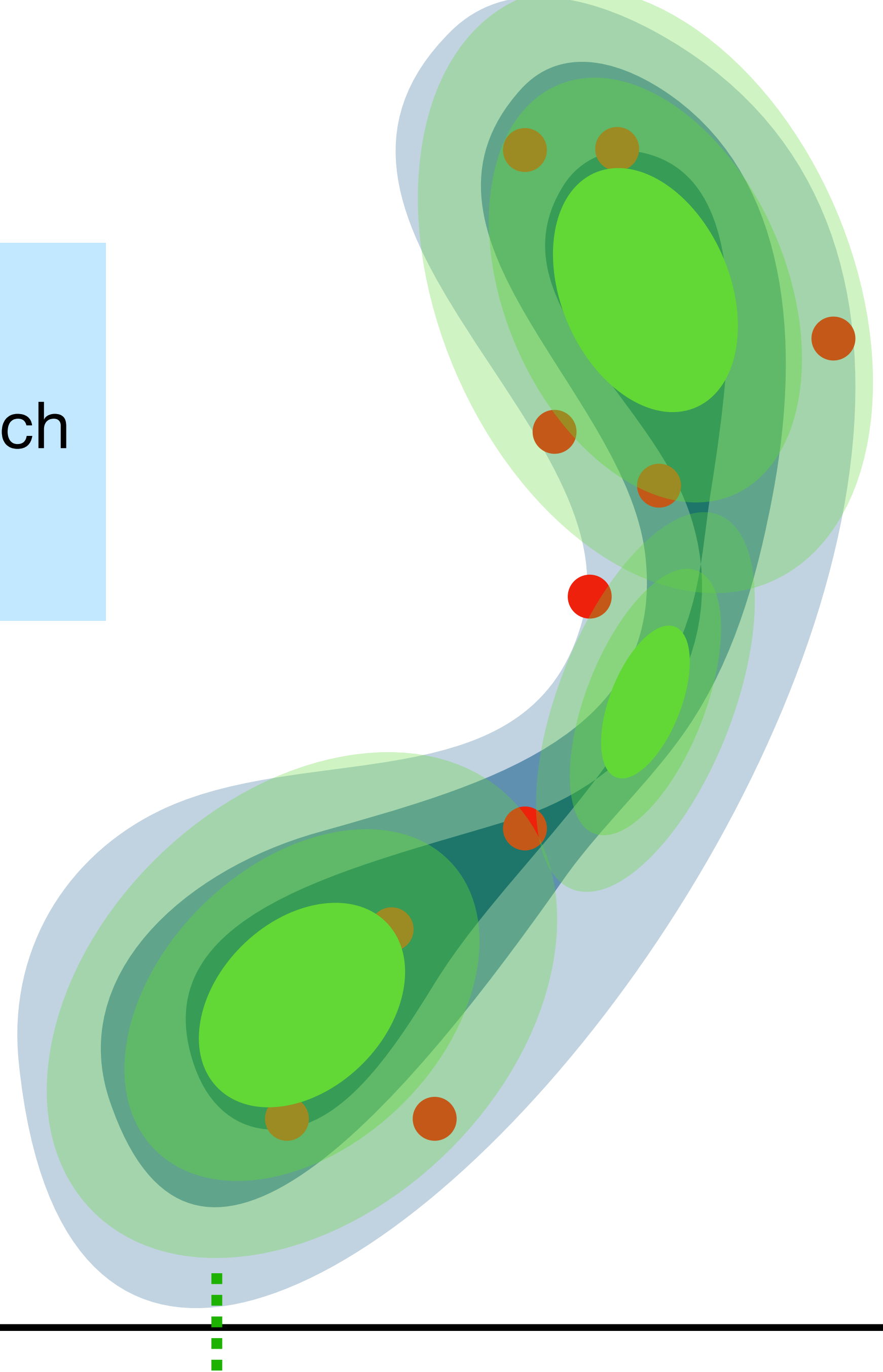
Approximate p_{data} with a parametric distribution p_{θ} « easy to sample »

Trade-off:

← approximation quality $p_{\theta} \approx p_{\text{data}}$

Gaussian Mixture

→ sampling efficiency



Each image i is a point in $x_i \in \mathbb{R}^d$

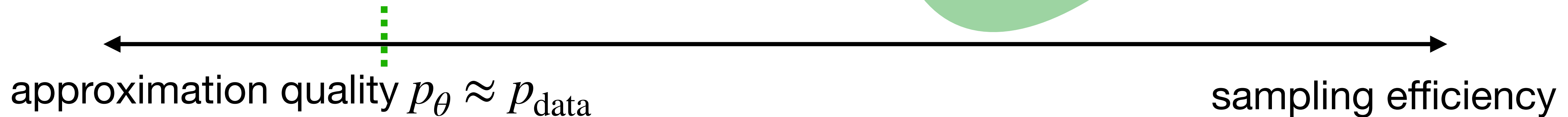
Assumption:

There exists a probability distribution p_{data} such that each image $x_i \sim p_{\text{data}}$

Objective:

Approximate p_{data} with a parametric distribution p_{θ} « easy to sample »

Trade-off: Large neural network

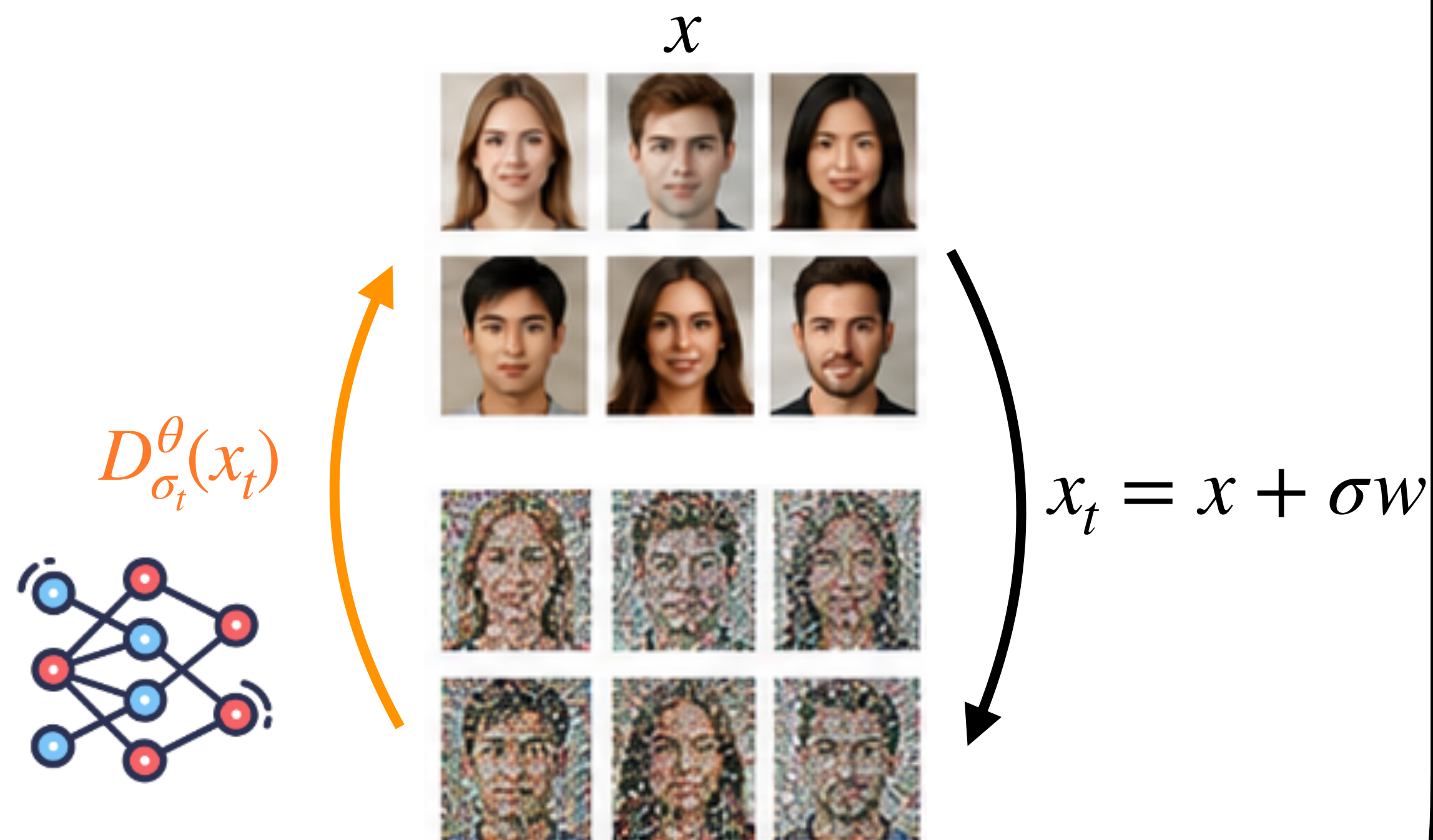


Conditional sampling of $p(x|y)$

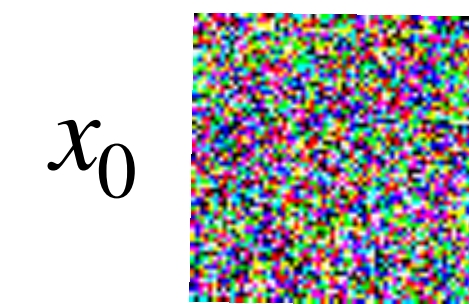
Reminder: **Unconditional** sampling of p_{data}

Denoiser Training:

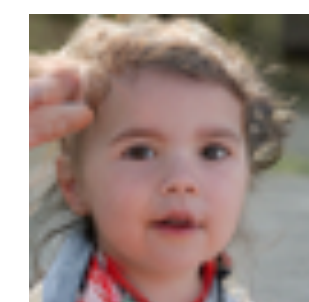
$$\operatorname{argmin}_{\theta} \mathbb{E}_{x \sim p_{data}, w \sim \mathcal{N}(0, I)} \|D_{\sigma_t}^{\theta}(x + \sigma_t w) - x\|^2$$



Diffusion Sampling:



$$dx_t = v_t(x_t) dt$$
$$v_t(x) = \dot{\sigma}_t \sigma_t \nabla \log p_{\sigma_t}(x) \text{ ???}$$

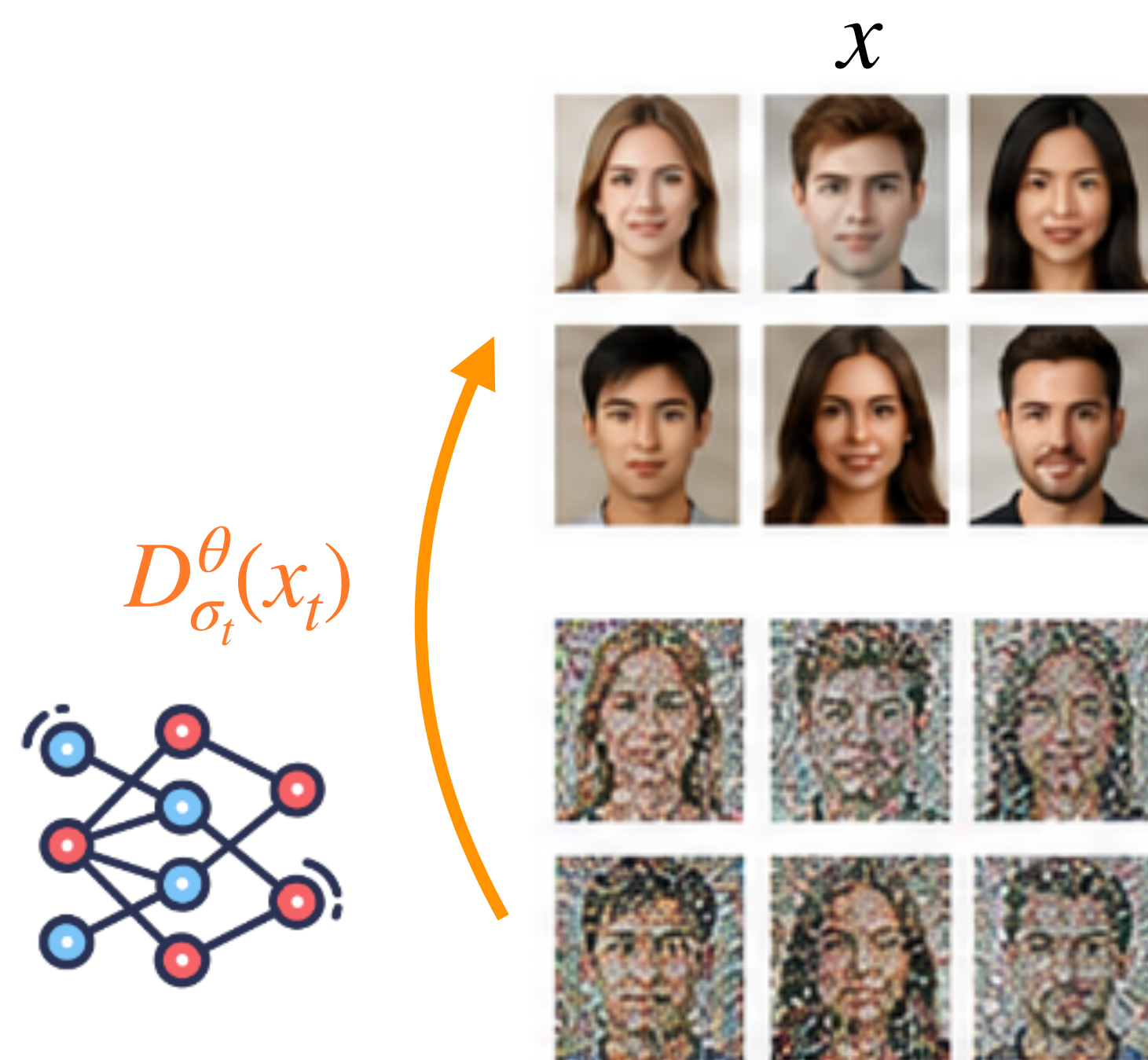


Conditional sampling of $p(x|y)$

Reminder: **Unconditional** sampling of p_{data}

Denoiser Training:

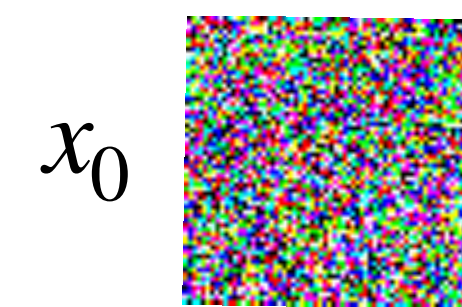
$$\operatorname{argmin}_{\theta} \mathbb{E}_{x \sim p_{data}, w \sim \mathcal{N}(0, I)} \|D_{\sigma_t}^{\theta}(x + \sigma_t w) - x\|^2$$



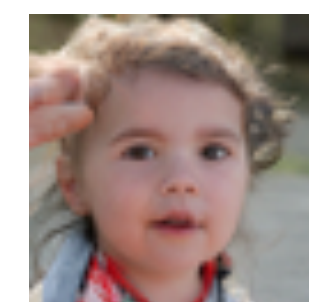
$$D_{\sigma_t}(x_t) \approx x_t + \sigma_t^2 \nabla \log p_{\sigma_t}(x_t)$$

$$x_t = x + \sigma w$$

Diffusion Sampling:



$$dx_t = v_t(x_t) dt$$
$$v_t(x_t) = \frac{\dot{\sigma}_t}{\sigma_t} \left(D_{\sigma_t}^{\theta}(x_t) - x_t \right)$$



Recap: Diffusion sampling **in theory**

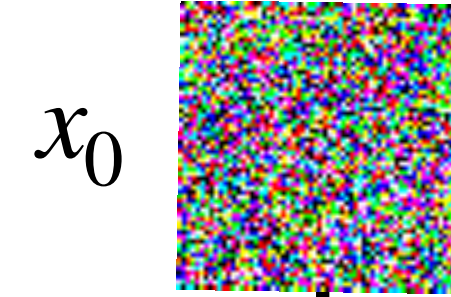
Exact MMSE denoiser

$$v_t(x_t) = \frac{\dot{s}_t}{s_t} x_t + \frac{\dot{\sigma}_t}{\sigma_t} \left[x_t - s_t D_{\sigma_t}^* \left(\frac{x_t}{s_t} \right) \right]$$

Parameters of the algorithm:

diffusion coefficient α , time schedulers s_t, σ_t

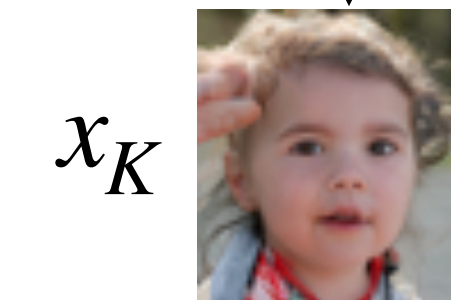
Diffusion Sampling:



Solve

$$x_0 \sim \mathcal{N}(0, \sigma_T^2 \text{Id})$$

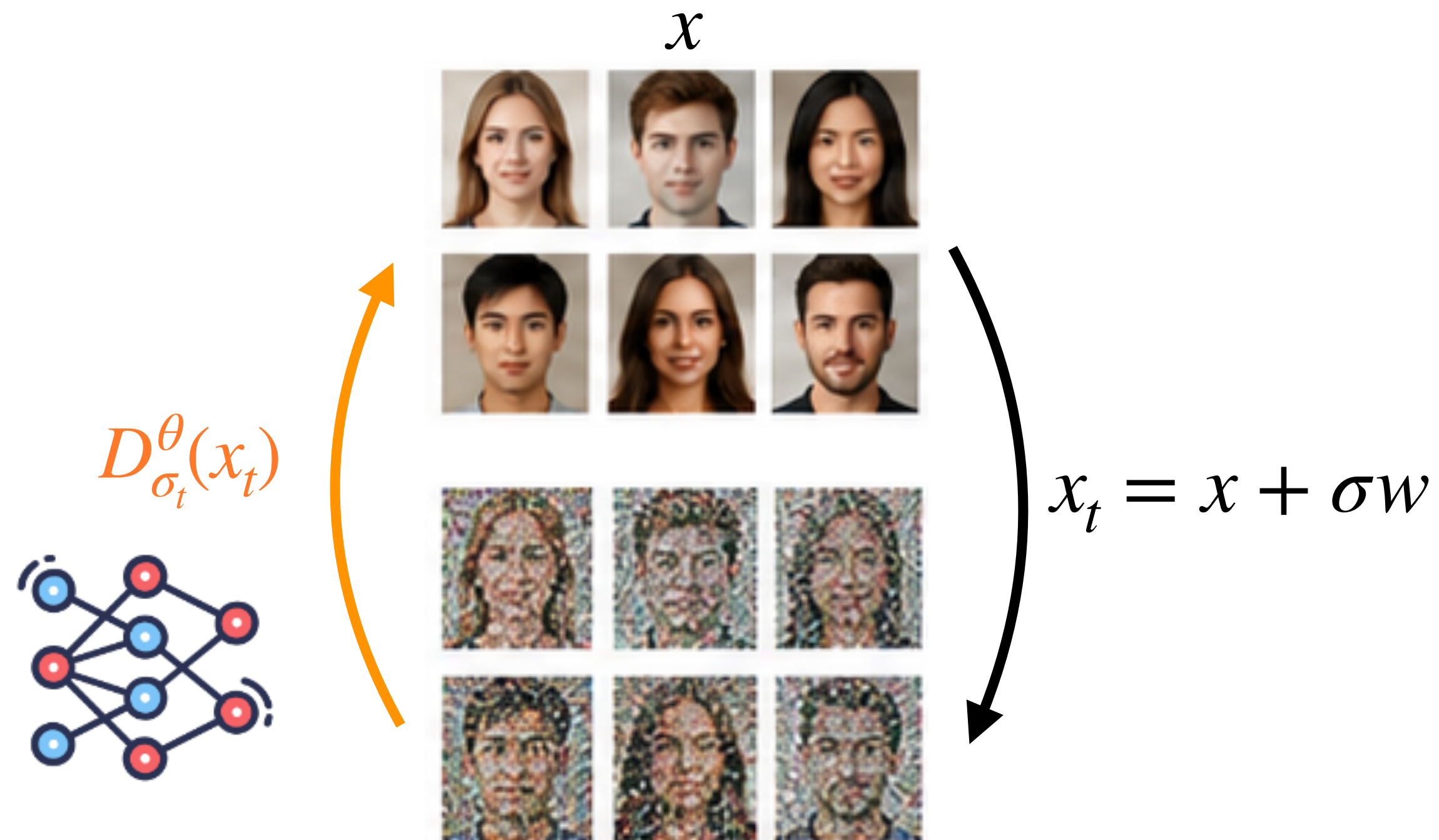
$$dx_t = v_t(x_t)dt + \sqrt{2\alpha a_t}dW_t$$



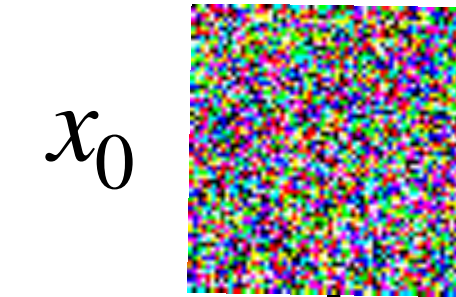
Recap: Diffusion sampling **in practice**

Denoiser Training:

$$\operatorname{argmin}_{\theta} \mathbb{E}_{x \sim p_{data}, w \sim \mathcal{N}(0, I)} \| D_{\sigma_t}^{\theta}(x + \sigma_t w) - x \|^2$$



Diffusion Sampling:



For $k = 1 \dots K$:

$$\hat{x} = D_{\sigma_{t_k}}^{\theta}(x_k)$$

$$x_{k+1} = (1 - \beta_k)x_k + \beta_k \hat{x} \quad \text{Denoise Average}$$

$$w \sim \mathcal{N}(0, I)$$

$$x_{k+1} = x_{k+1} + \alpha a_k w \quad \text{Add noise (Optional)}$$

