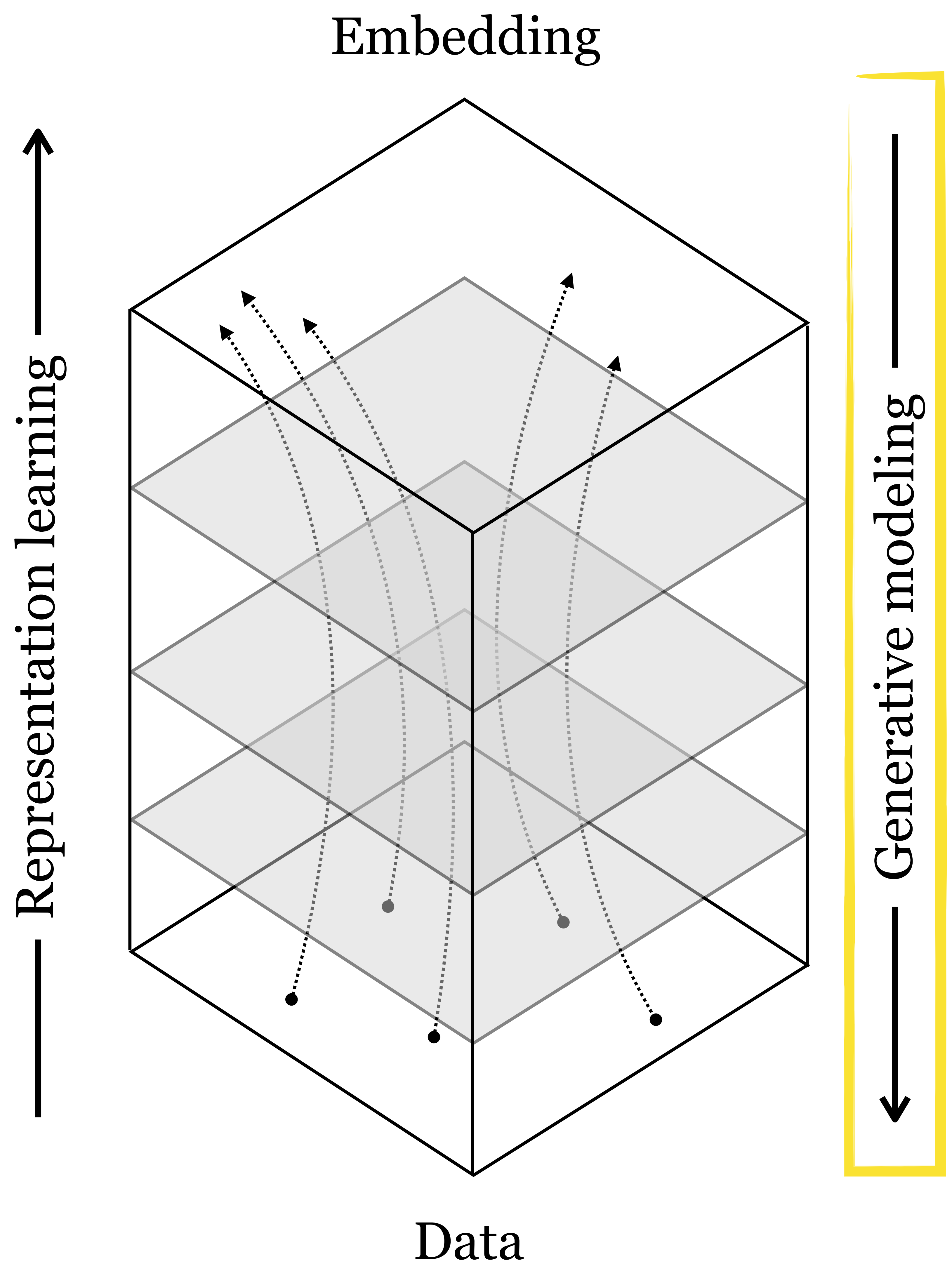
Lecture 14: Deep Generative Models I

Speaker: Phillip Isola

Deep generative models

- Lecture 14: fundamentals, a tour of popular models
- Lecture 15: generative modeling meets representation learning
- Lecture 16: conditional models, data prediction



Deep generative models I

- Math background
- Fundamentals of generative modeling
- Density functions, energy functions, and samplers
- Autoregressive models
- Diffusion models
- Generative adversarial networks

What is a generative model?

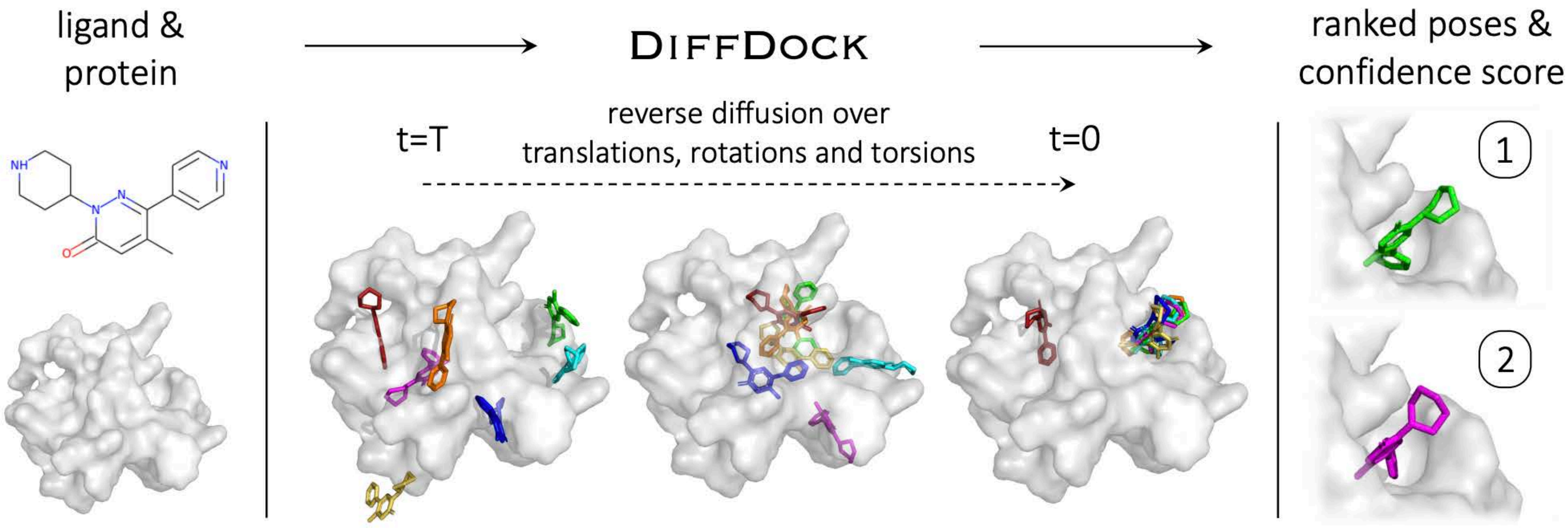
1. An algorithm that generates data

2. A statistical model of the joint distribution of some data, $p(x, y, \ldots)$

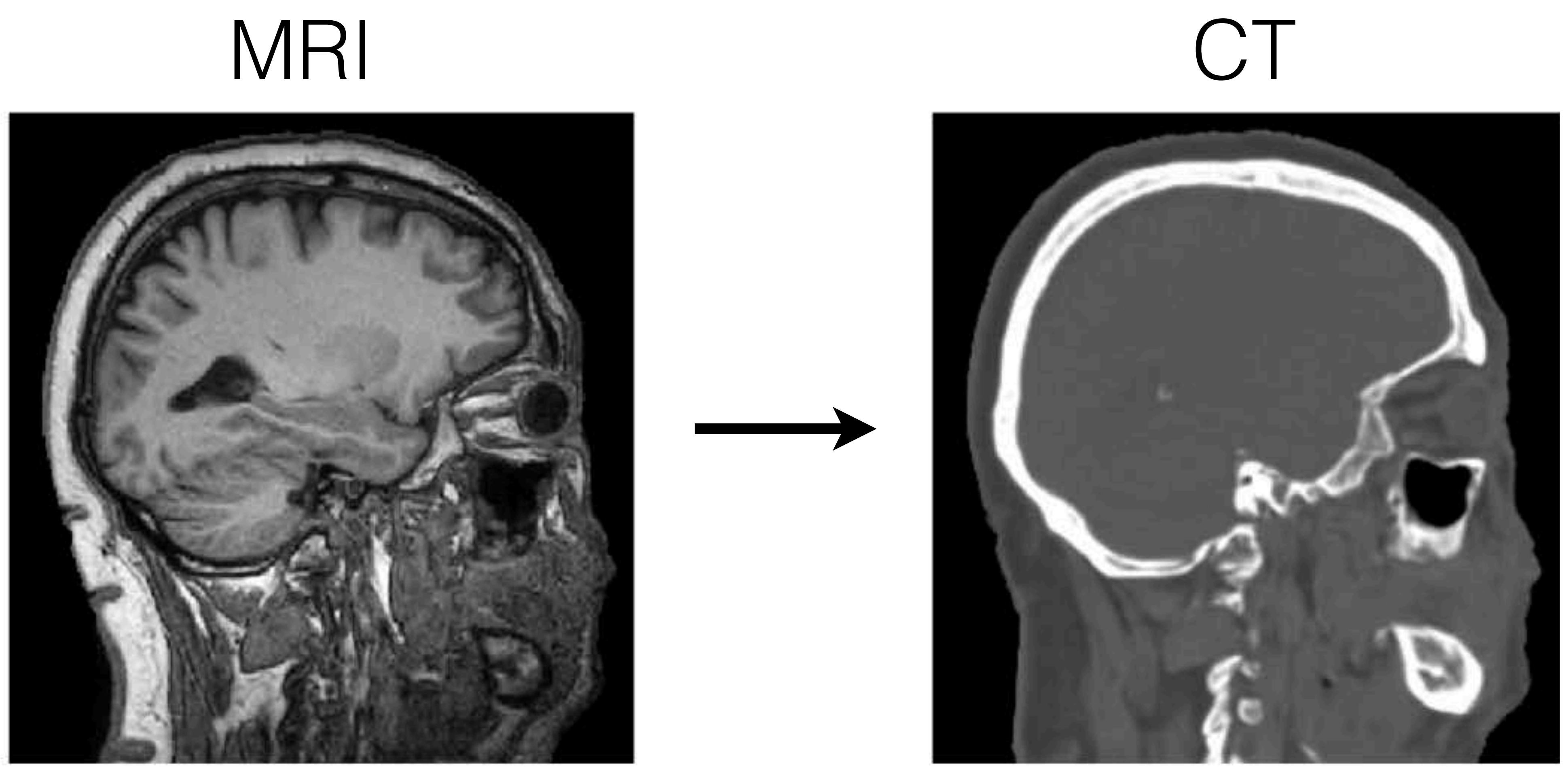
A photo of a group of robots building the Stata Center

https://openai.com/dall-e-2/

Created with DallE.



[Corso*, Stark*, Jing*, Barzilay, Jaakkola, 2022]



[Wolterink, Dinkla, Savenije et al., 2017]

Math background

- So far we have mostly thought of neural nets as mappings $f_\theta : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y}$ is some space of possible outputs.
  - e.g., image classifier into d classes: $f_\theta : \mathbb{R}^{N \times M \times C} \to \{1,\ldots,d\}$
- Now, we will instead consider neural nets as mappings $f_\theta : \mathcal{X} \to \mathcal{P}(\mathcal{Y})$, where $\mathcal{P}(\mathcal{Y})$ is the space of probability distributions over $\mathcal{Y}$.
  - e.g., softmax regression to model $P(\text{class} \mid X = \mathbf{x})$: $f_\theta : \mathbb{R}^{N \times M \times C} \to \Delta^{d-1}$
  - *Main perspective: the outputs of our neural nets are, implicitly or explicitly, distributions*

Random variable: X, $\quad p(X) \in \mathscr{P}(\mathscr{X})$ is the probability density/mass function

Realization: $x \sim p(X)$, $\quad p(x) \in \mathscr{X}$ is the probability mass/density of x

Probability mass function: $p : \mathscr{X} \rightarrow \mathscr{R}, \quad 0 \leq p(x) \leq 1, \quad \sum_{x \in \mathscr{X}} p(x) = 1$

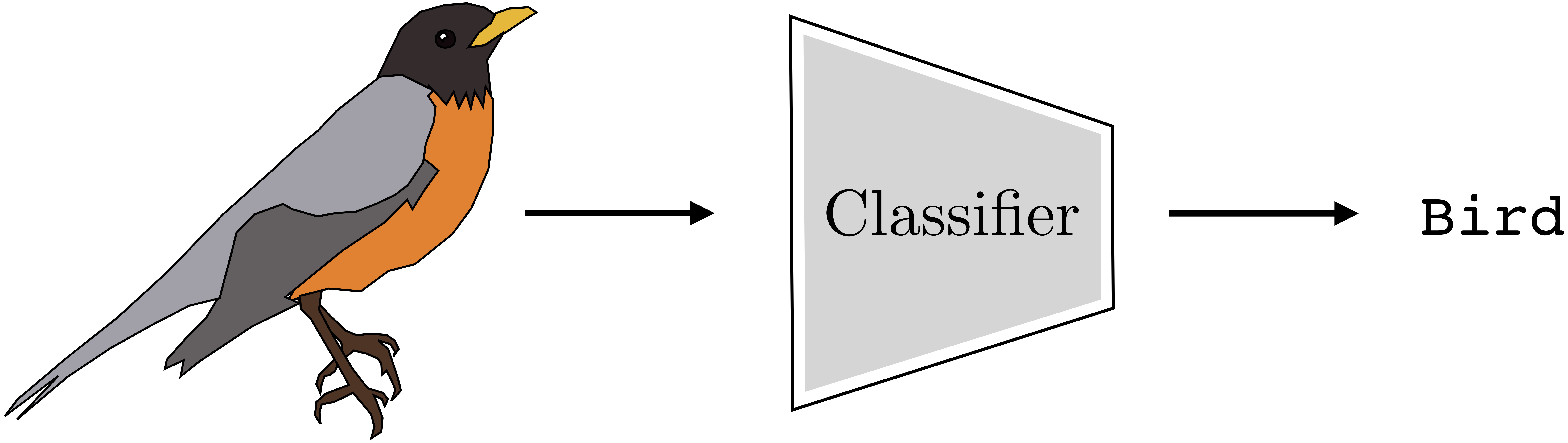
Probability density function: $p : \mathscr{X} \rightarrow \mathscr{R}, \quad p(x) \geq 0, \quad \int_{x \in \mathscr{X}} p(x) dx = 1$

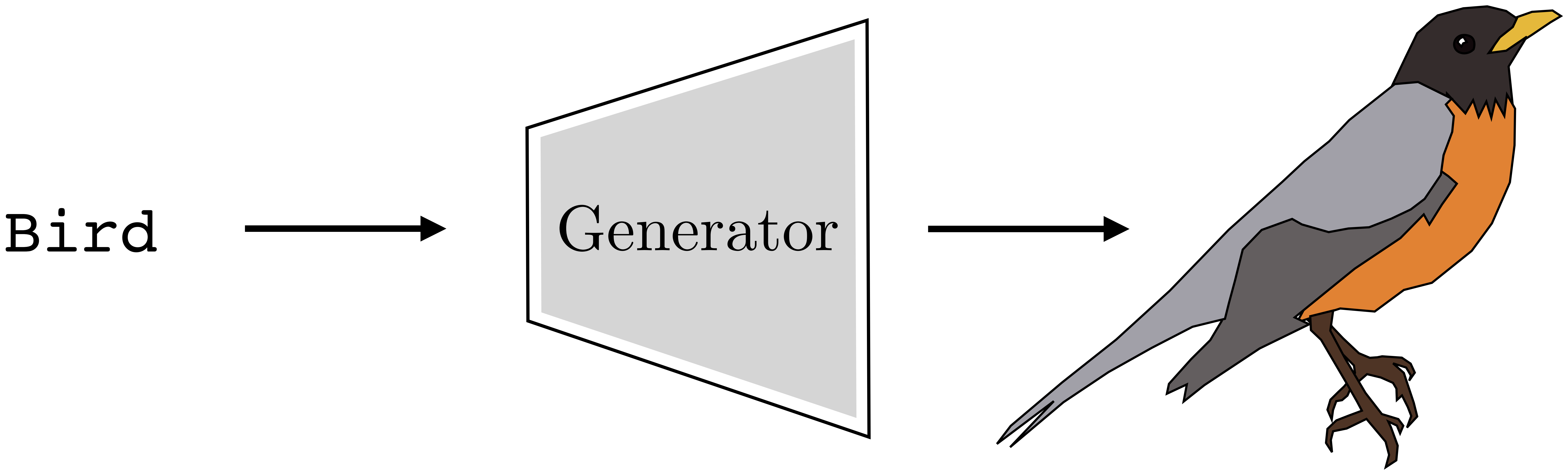
Probabilities
We will typically not distinguish between random variables and realizations of those variables; which we mean should be clear from context. When it is important to make a distinction, we will use non-bold capital letters to refer to random variables and lowercase to refer to realizations.

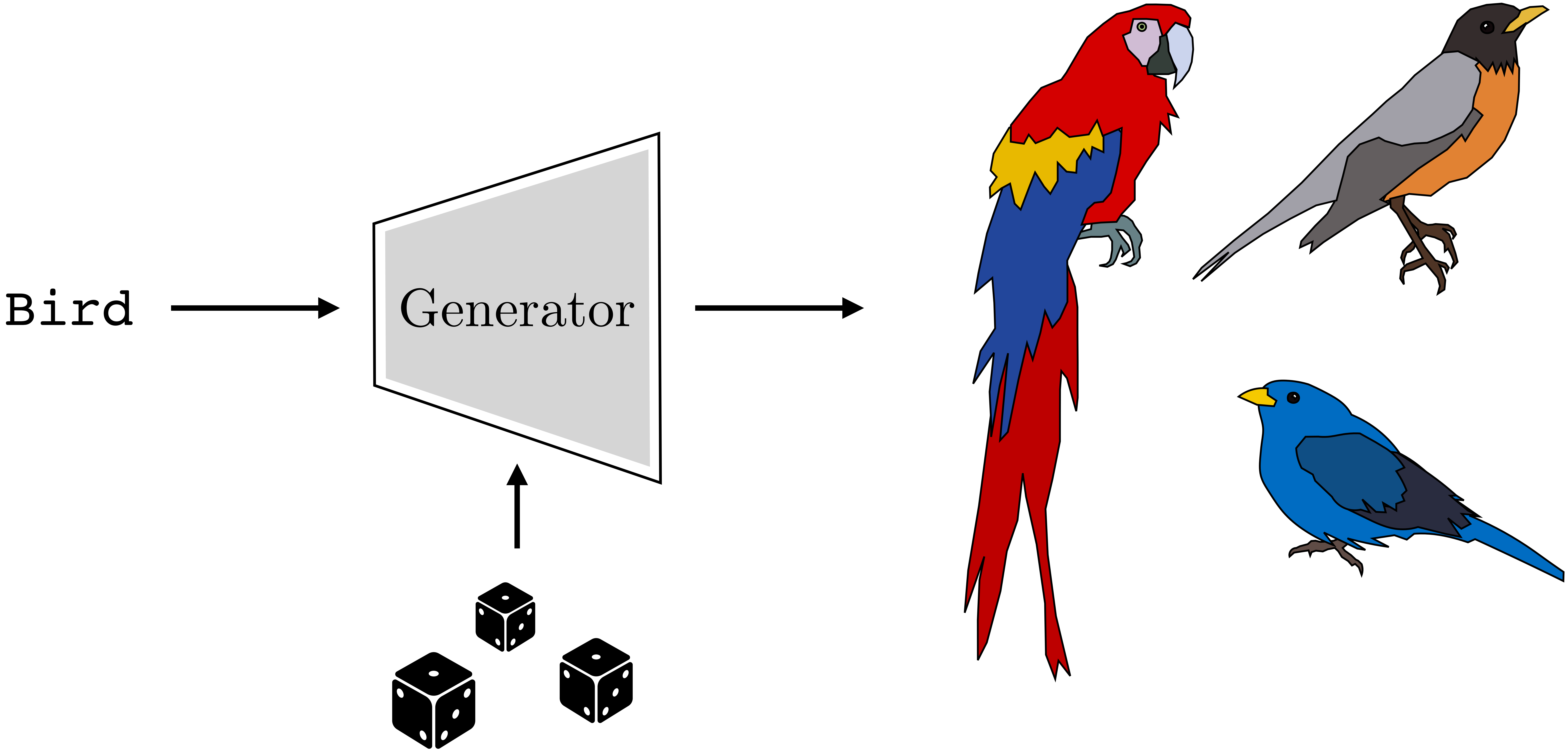
Suppose X, Y are discrete random variables and $\mathbf{x}, \mathbf{y}$ are realizations of those variables. X and Y may take on values in the sets $\mathcal{X}$ and $\mathcal{Y}$ respectively.

- $a = p(X = \mathbf{x}|\ldots)$ is the probability of the realization $X = \mathbf{x}$, possibly conditioned on some observations (a is a scalar).
- $f = p(X|\ldots)$ is the probability distribution over X, possibly conditioned on some observations (f is a function: $f : \mathcal{X} \to \mathbb{R}$). If $\mathcal{X}$ is discrete, f is the *probability mass function*. If $\mathcal{X}$ is continuous, f is the *probability density function*.
- $p(\mathbf{x}|\ldots)$ is shorthand for $p(X = \mathbf{x}|\ldots)$.
- and so forth, following these patterns.
- Suppose we have defined a named distribution, e.g., p_θ; then referring to p_θ on its own is shorthand for $p_\theta(X)$

https://phillipi.github.io/6.7960/materials/notation.pdf


Classifier
Bird


Bird
Generator


Bird
Generator

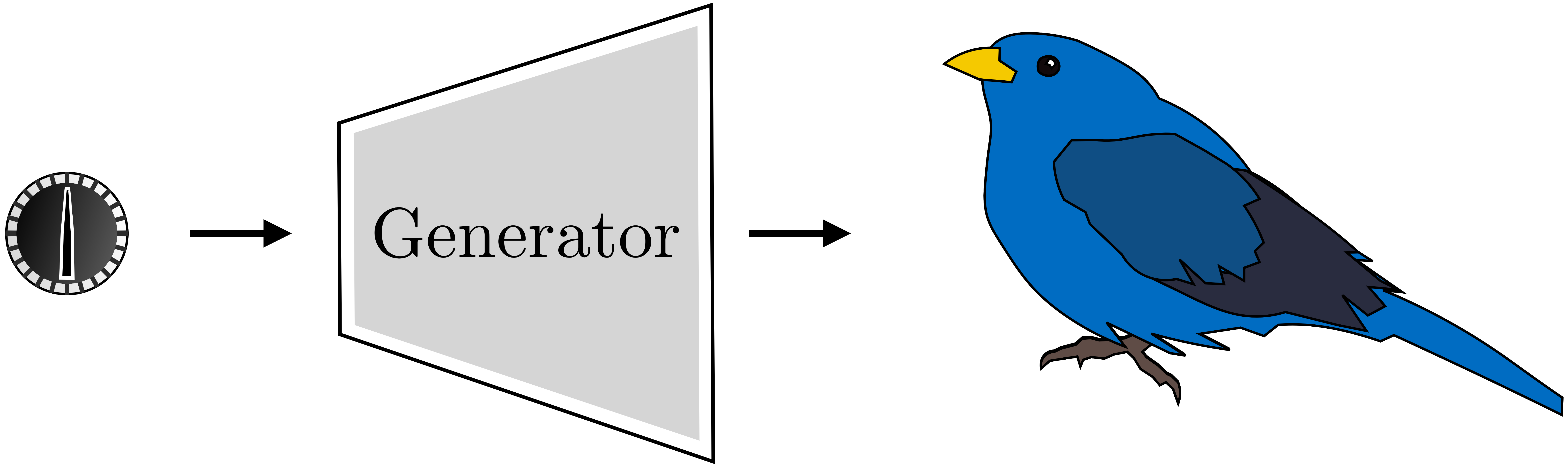
which color?

what angle?

what size?

⋮

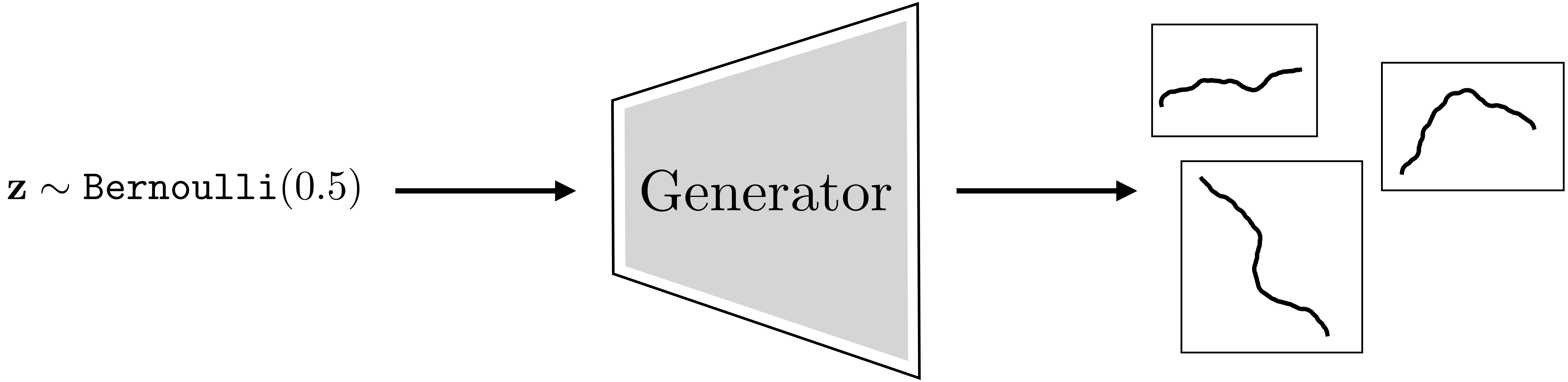
which color?



$\mathbf{z} \sim \texttt{Bernoulli}(0.5)$

for $i = 1, \ldots, N$ **do**

```
extend line 1 unit in current heading direction
if z_i == 1 then
    rotate heading 10° to the right
else
    rotate heading 10° to the left
```



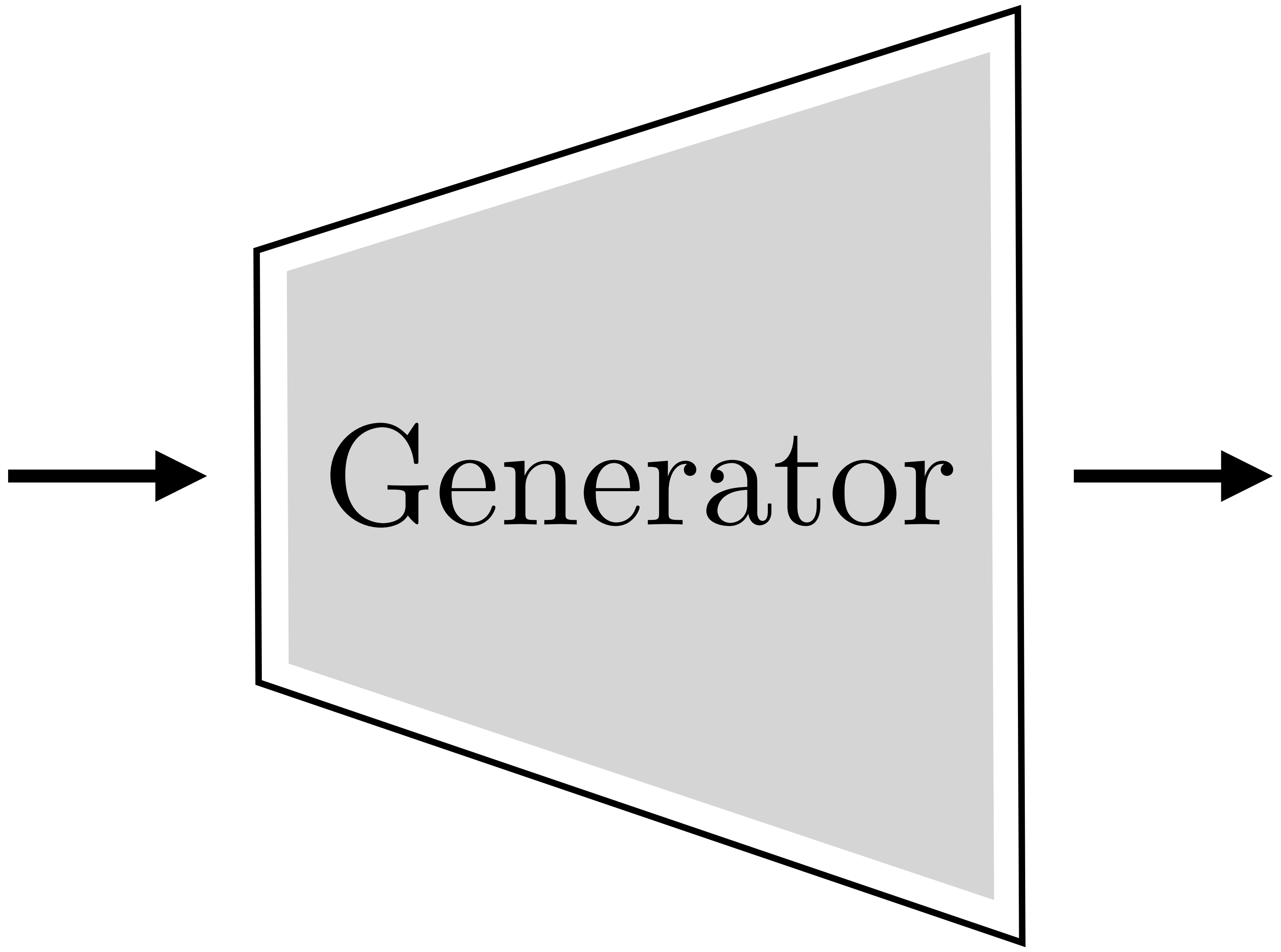
“noise” “latent variables”

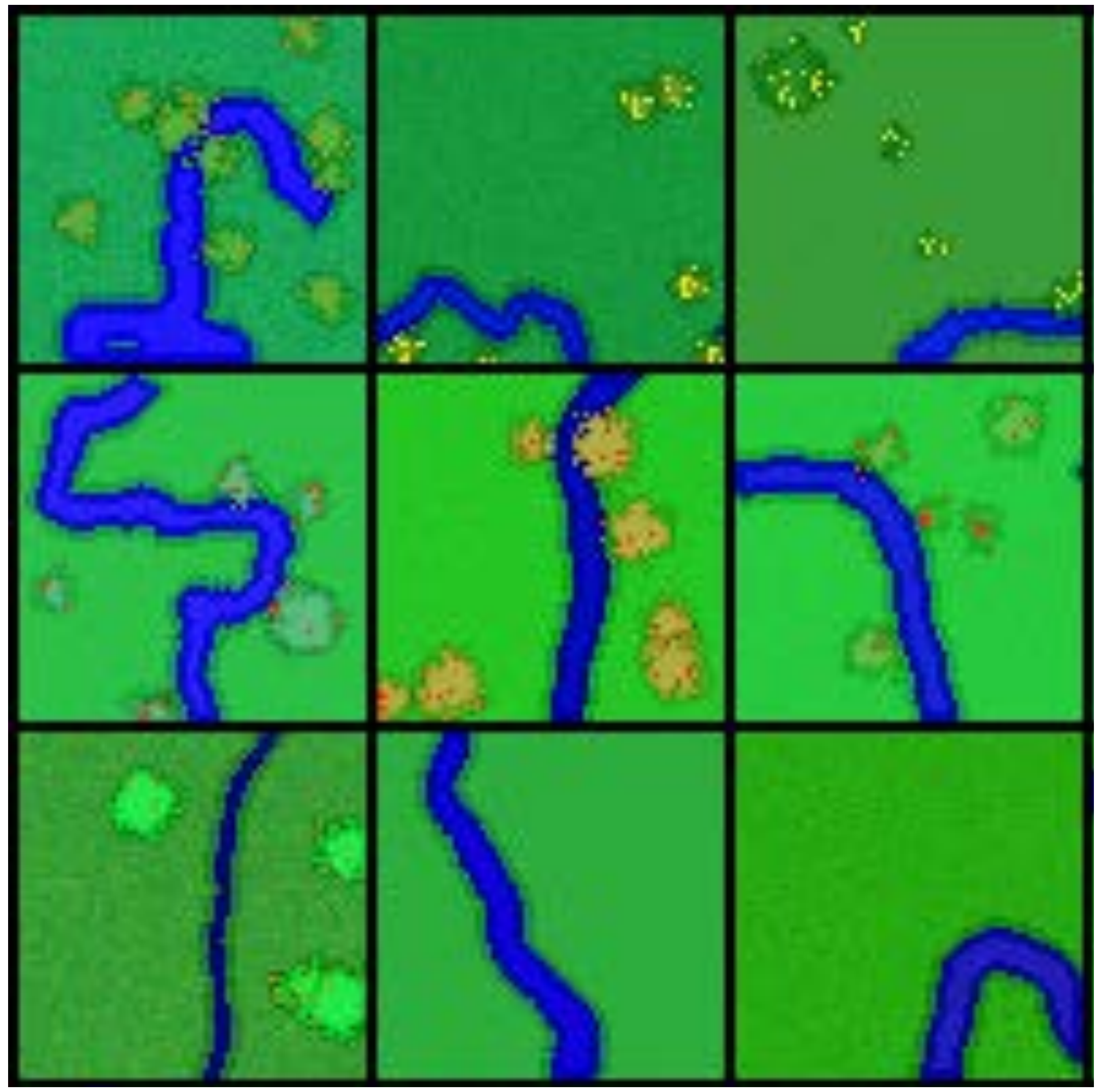
$\mathbf{z}_1 \sim \texttt{Bernoulli}(0.5)$ (River turns)

$\mathbf{z}_2 \sim \texttt{Normal}(\mu_1, \Sigma_1)$ (Grass color)

$\mathbf{z}_3 \sim \texttt{Unif}(0, 10)$ (Number trees)

$\vdots$





Concept #1: noise is latent variables

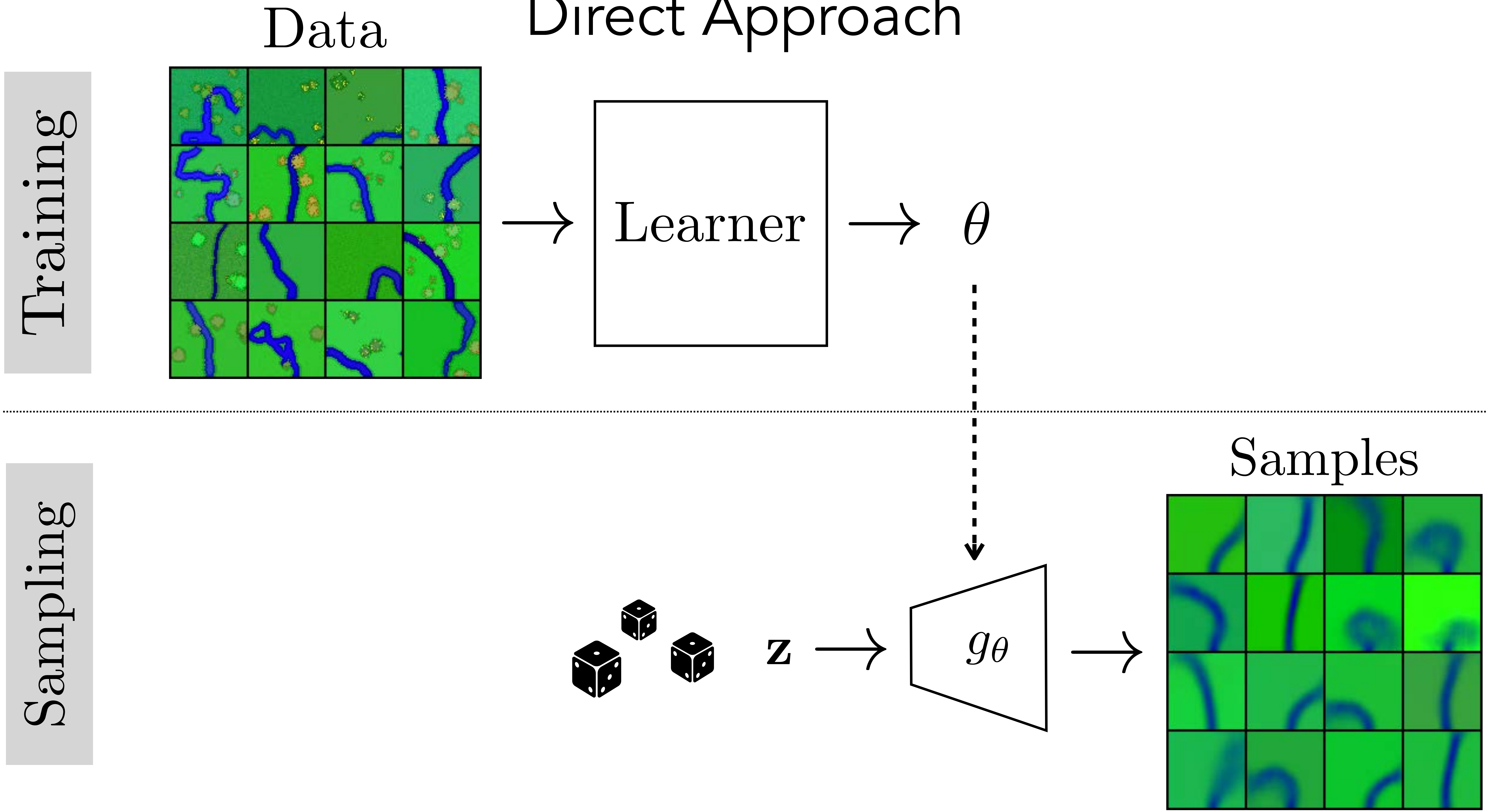
Learning data generators

Two approaches: confusingly, sometimes called an "implicit generative model"

1. **Direct approach**: learn a function that generates data directly $G : \mathcal{Z} \rightarrow \mathcal{X}$

2. **Indirect approach**: learn a function that scores data; generate data by finding points that score highly under this function $E : \mathcal{X} \rightarrow \mathbb{R}$

Direct Approach



Indirect approach

e.g., likelihood, energy,
“score function”

What's the goal of generative modeling?

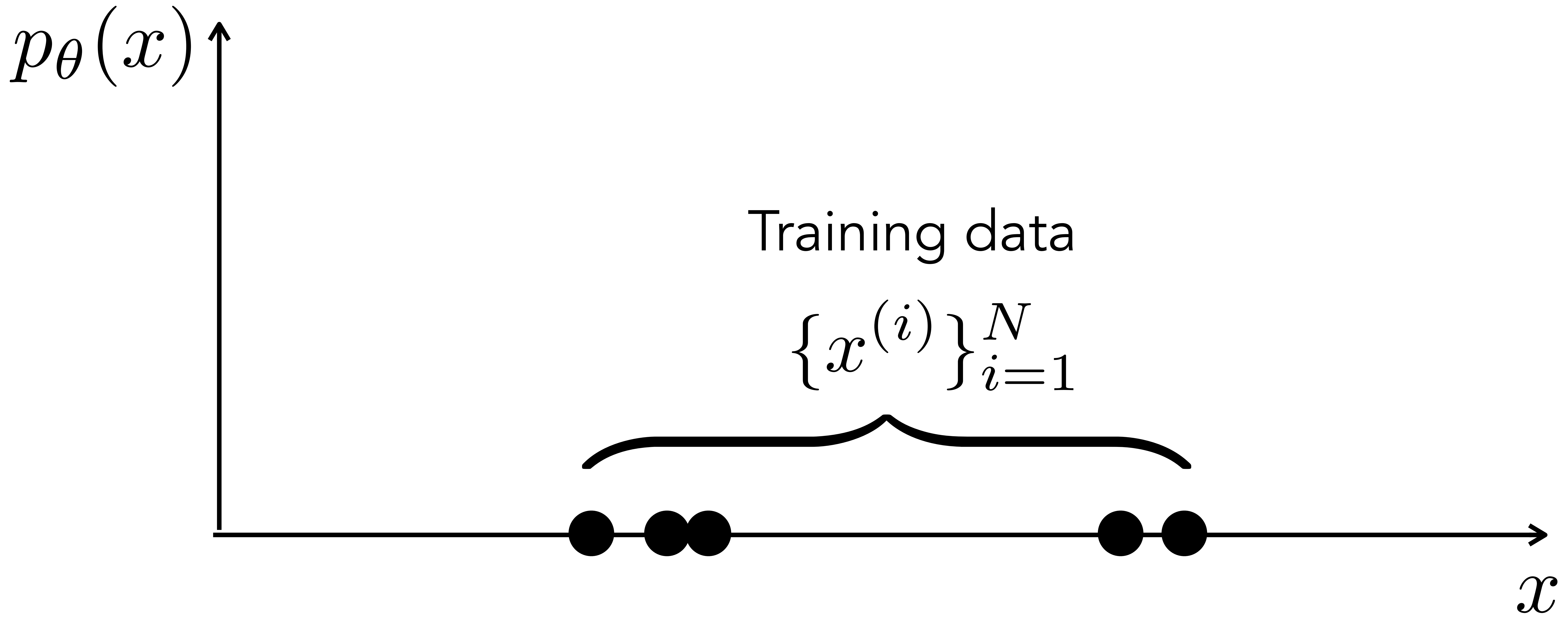
Make synthetic data that "looks like" real data.

How to measure "looks like"?

The main answer in deep generative models is: "has high probability under a density model fit to real data."

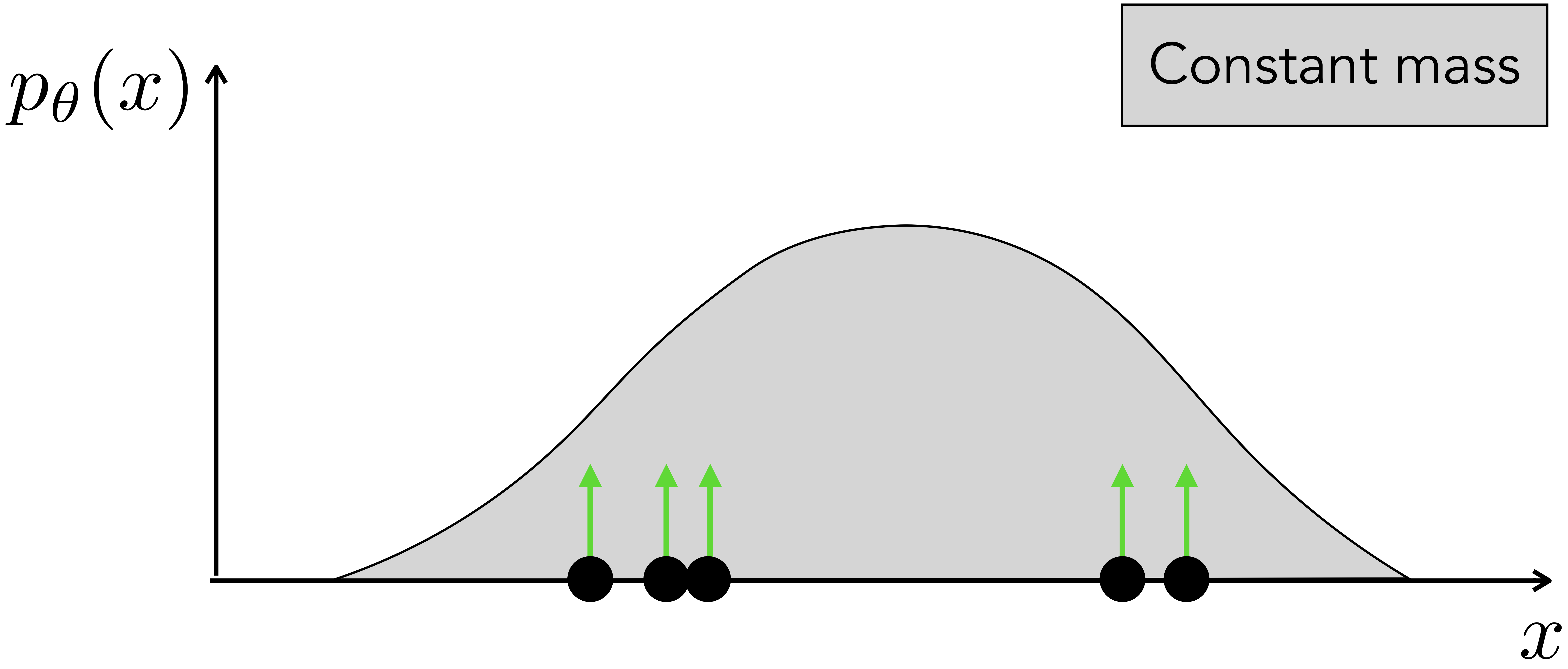
Density models

$$p_\theta : \mathcal{X} \to [0, \infty)$$



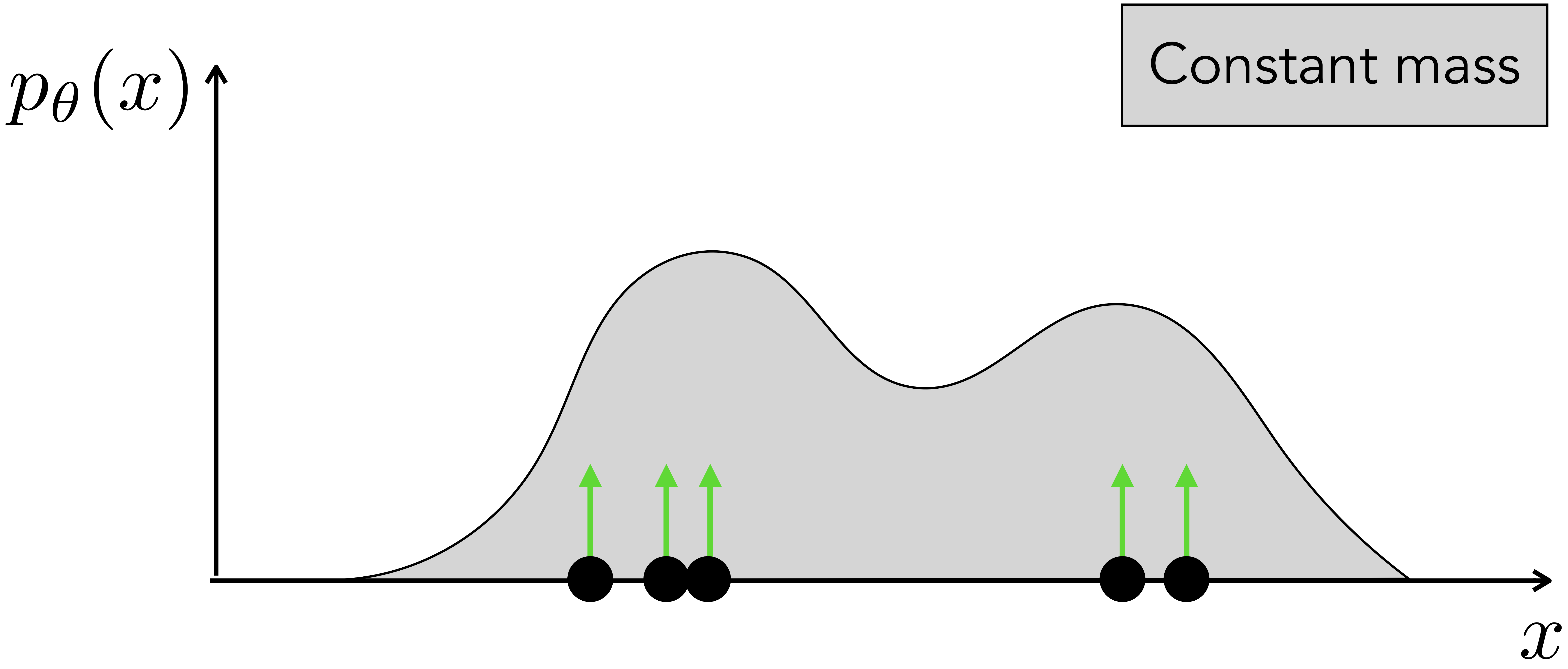
Density models

$$p_\theta : \mathcal{X} \rightarrow [0, \infty)$$

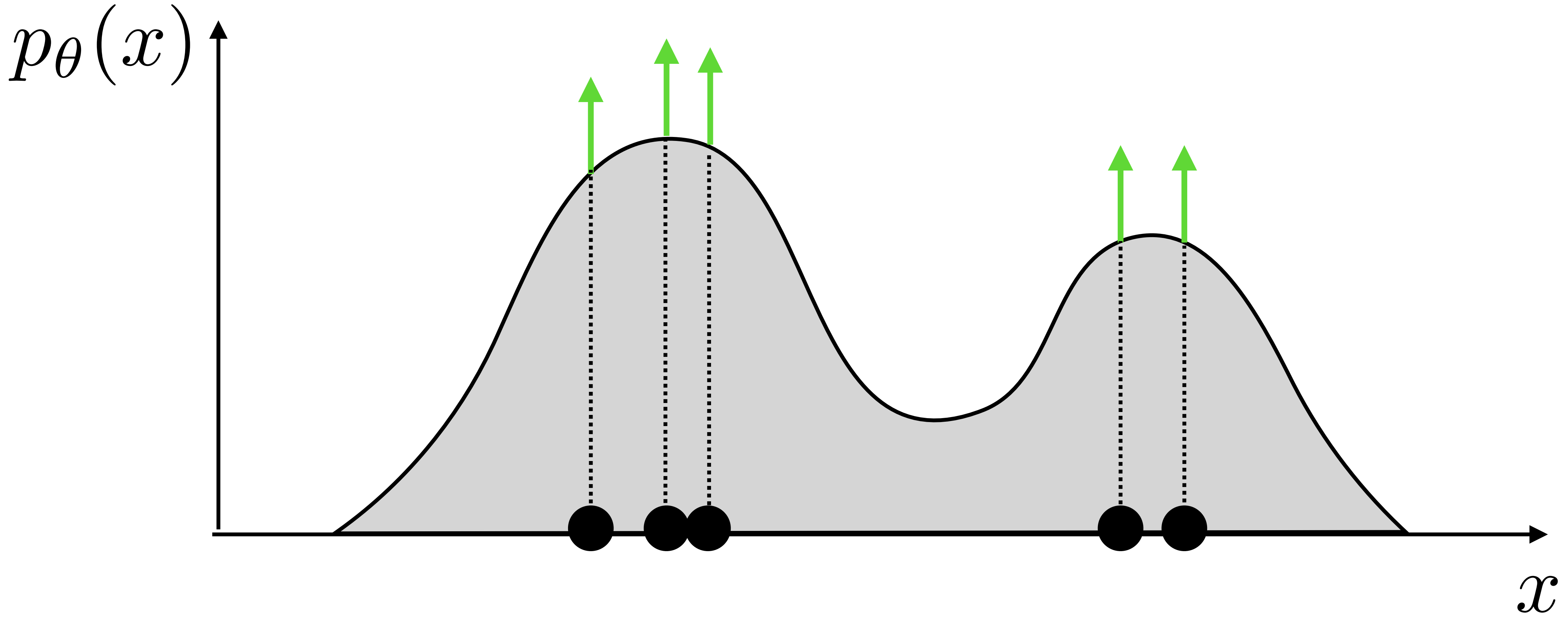


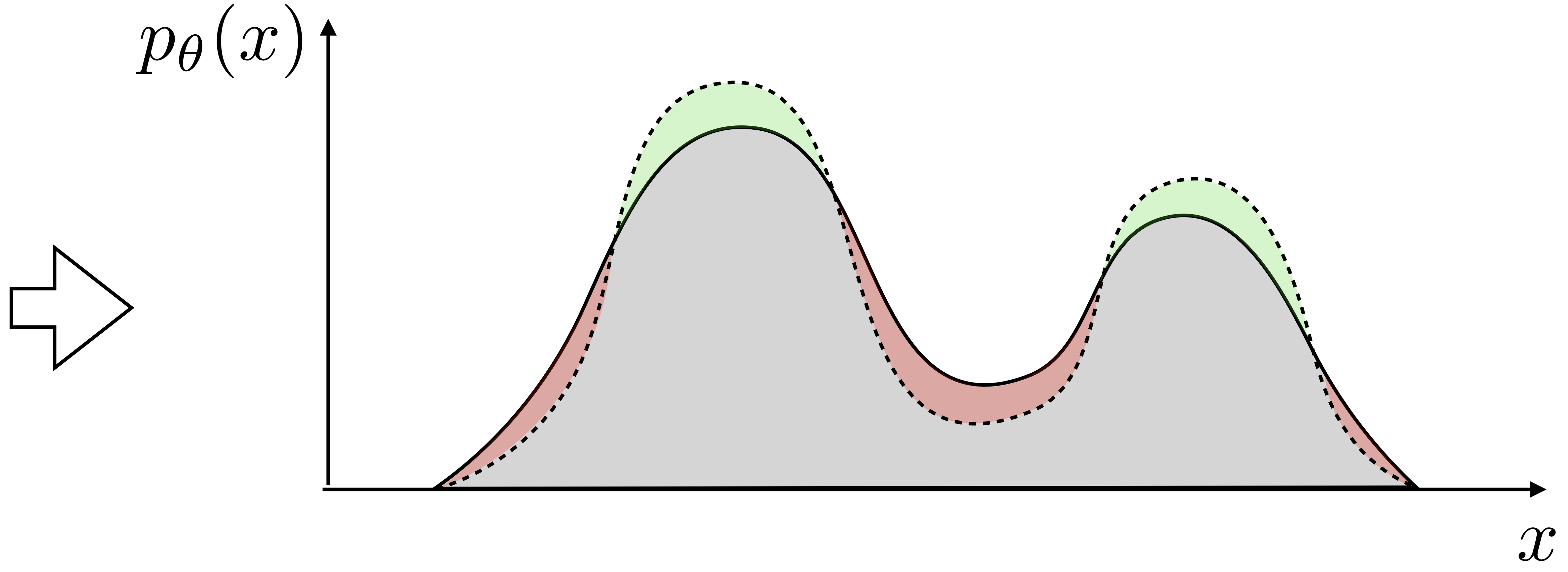
Density models

$$p_\theta : \mathcal{X} \to [0, \infty)$$



Density models





$$p_\theta^* = \arg\min_{p_\theta} \mathtt{KL}(p_{\mathtt{data}}, p_\theta)$$

$$= \arg\min_{p_\theta} \mathbb{E}_{\mathbf{x}\sim p_{\mathtt{data}}}\left[-\log \frac{p_\theta(\mathbf{x})}{p_{\mathtt{data}}(\mathbf{x})}\right]$$

$$= \arg\max_{p_\theta} \mathbb{E}_{\mathbf{x}\sim p_{\mathtt{data}}}\left[\log p_\theta(\mathbf{x})\right] - \mathbb{E}_{\mathbf{x}\sim p_{\mathtt{data}}}\left[\log p_{\mathtt{data}}(\mathbf{x})\right]$$

$$= \arg\max_{p_\theta} \mathbb{E}_{\mathbf{x}\sim p_{\mathtt{data}}}\left[\log p_\theta(\mathbf{x})\right] \quad \triangleleft \quad \text{dropped second term since no dependence on } p_\theta$$

$$\approx \arg\max_{p_\theta} \frac{1}{N}\sum_{i=1}^{N} \log p_\theta(\mathbf{x}^{(i)})$$

max likelihood

Is the filing cabinet a good generative model?

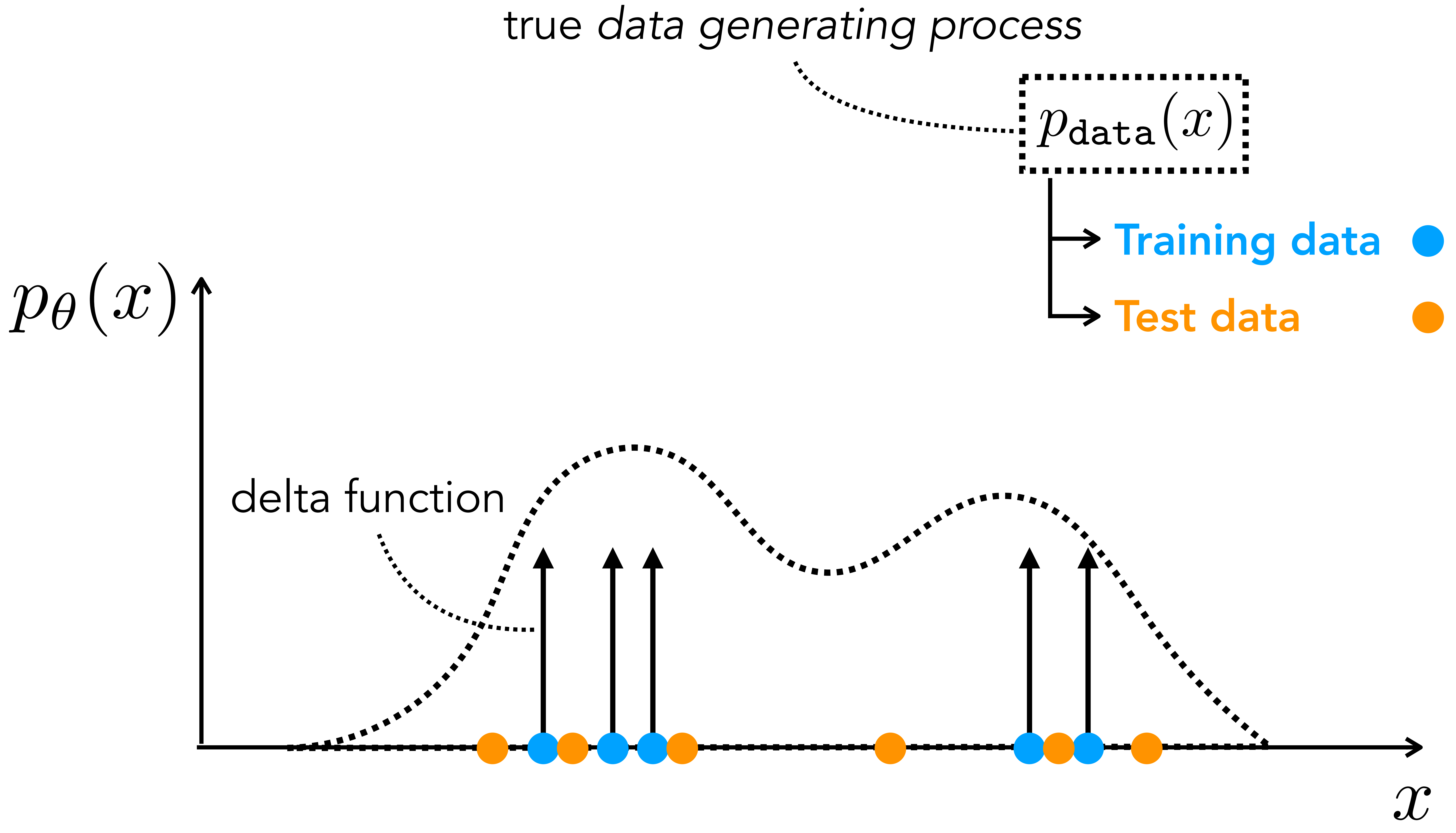
Every time we see a new training point (x), we put it in the cabinet.

```
def train(X):
  for x in X:
    cabinet.append(x)

def generate():
  return cabinet[np.random.randint(len(cabinet))]
```

Sample by picking a drawer at random.

What is the pdf the filing cabinet is sampling from?



What's the goal of generative modeling?

The goal is not to replicate the training data but to make ***new*** data that is ***realistic*** (captures the essential properties of real data)

One way to quantify this is: likelihood of the ***test data*** under the model. (A model that memorizes the training data is overfit in exactly the same sense as a classifier can be overfit.)

$$\{x^{(i)}_{\texttt{test}}\}_{i=1}^{N}, \quad x^{(i)}_{\texttt{test}} \sim p_{\texttt{data}}$$

$$\text{generalization error} = \sum_i \log p_\theta(x^{(i)}_{\texttt{test}})$$

Energy-based models

i.e. unnormalized probability models

$$\int_{\mathbf{x}} p_\theta(\mathbf{x})d\mathbf{x} = 1$$

$$p_\theta = \frac{e^{-E_\theta}}{Z(\theta)} \qquad Z(\theta) = \int_{\mathbf{x}} e^{-E_\theta(\mathbf{x})}d\mathbf{x}$$

$$\frac{p_\theta(\mathbf{x}_1)}{p_\theta(\mathbf{x}_2)} = \frac{e^{-E_\theta(\mathbf{x}_1)}/Z(\theta)}{e^{-E_\theta(\mathbf{x}_2)}/Z(\theta)} = \frac{e^{-E_\theta(\mathbf{x}_1)}}{e^{-E_\theta(\mathbf{x}_2)}}$$

<— Relative probabilities are often all you need (e.g., for sampling)

Energy-based models

At convergence, green (data) and red (model) samples are identical and model update (green-red) cancels out

Contrastive divergence

Energy-based models — learning model parameters

$$\nabla_\theta \mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}[\log p_\theta(\mathbf{x})] = \nabla_\theta \mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}[\log \frac{e^{-E_\theta(\mathbf{x})}}{Z(\theta)}]$$

$$= -\mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}[\nabla_\theta E_\theta(\mathbf{x})] - \underbrace{\nabla_\theta \log Z(\theta)}$$

How to measure this?

Energy-based models — learning model parameters

$$\begin{aligned}
\boxed{-\nabla_\theta \log Z(\theta)} &= \frac{1}{Z(\theta)} \nabla_\theta Z(\theta) && \triangleleft \quad \nabla_x \log f(x) = \tfrac{1}{f(x)} \nabla_x f(x) \\
&= \frac{1}{Z(\theta)} \nabla_\theta \int_x e^{-E_\theta(\mathbf{x})} d\mathbf{x} && \triangleleft \quad \text{definition of } Z \\
&= \frac{1}{Z(\theta)} \int_x \nabla_\theta e^{-E_\theta(\mathbf{x})} d\mathbf{x} && \triangleleft \quad \text{exchange sum and grad} \\
&= \frac{1}{Z(\theta)} - \int_x e^{-E_\theta(\mathbf{x})} \nabla_\theta E_\theta(\mathbf{x}) d\mathbf{x} \\
&= - \int_x \frac{e^{-E_\theta(\mathbf{x})}}{Z(\theta)} \nabla_\theta E_\theta(\mathbf{x}) d\mathbf{x} \\
&= - \int_x p_\theta(\mathbf{x}) \nabla_\theta E_\theta(\mathbf{x}) d\mathbf{x} && \triangleleft \quad \text{definition of } p_\theta \\
&= -\mathbb{E}_{\mathbf{x} \sim p_\theta} [\nabla_\theta E_\theta(\mathbf{x})] && \triangleleft \quad \text{definition of expectation}
\end{aligned}$$

Energy-based models — learning model parameters

$$\nabla_\theta \mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}[\log p_\theta(\mathbf{x})] = \nabla_\theta \mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}[\log \frac{e^{-E_\theta(\mathbf{x})}}{Z(\theta)}]$$

$$= -\mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}[\nabla_\theta E_\theta(\mathbf{x})] - \nabla_\theta \log Z(\theta)$$

$$= -\mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}[\nabla_\theta E_{\theta(\mathbf{x})}] + \mathbb{E}_{\mathbf{x}\sim p_\theta}[\nabla_\theta E_\theta(\mathbf{x})]$$

$$\approx -\frac{1}{N}\sum_{i=1}^{N} \nabla_\theta E_\theta(\mathbf{x}^{(i)}) + \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta E_\theta(\hat{\mathbf{x}}^{(i)})$$

$$\mathbf{x}^{(i)} \sim p_{\texttt{data}} \qquad \hat{\mathbf{x}}^{(i)} \sim p_\theta$$

Energy-based models

At convergence, green (data) and red (model) samples are identical and model update (green-red) cancels out

Contrastive divergence

Data

$\{x^{(i)}\}_{i=1}^{N} \rightarrow$ **Generative modeling** $\rightarrow$

Density function

$p_\theta : \mathcal{X} \rightarrow [0, \infty)$

Energy function

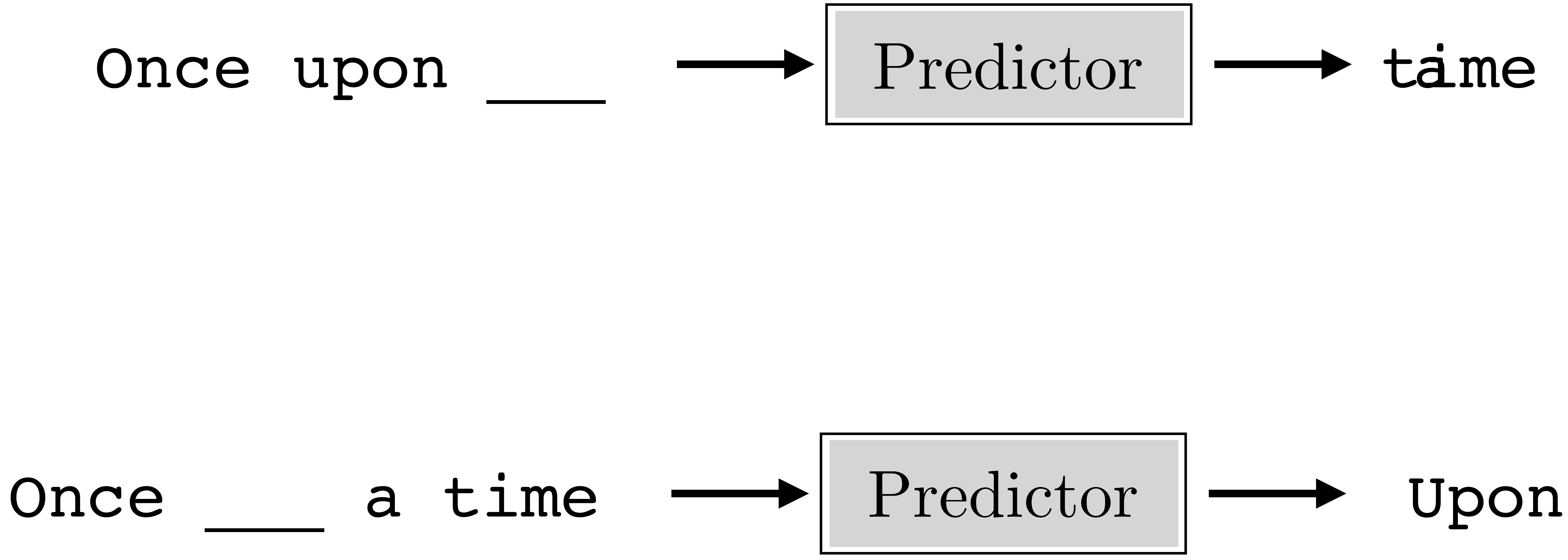
$E_\theta : \mathcal{X} \rightarrow \mathbb{R}$

Generator

$G_\theta : \mathcal{Z} \rightarrow \mathcal{X}$

Concept #2: **you can represent the data generating process directly or indirectly**

Autoregressive models




Training
$\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}$
$\mathbf{x}_n$
Once upon a , time
There and back , again
The slow brown , fox
To be or not to, be
Learner
Predictor
Sampling
$\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}$
Colorless green ideas sleep
Predictor
$\hat{\mathbf{x}}_n$
furiously

Autoregressive probability model

$$p(\mathbf{X}) = p(\mathbf{x_n}|\mathbf{x_1}, \ldots, \mathbf{x}_{n-1})p(\mathbf{x}_{n-1}|\mathbf{x}_1, \ldots, \mathbf{x}_{n-2}) \quad \ldots \quad p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_1)$$

$$p(\mathbf{X}) = \prod_{i=1}^{n} p(\mathbf{x}_i|\mathbf{x}_1, \ldots, \mathbf{x}_{i-1})$$

$$p(\texttt{time}|\texttt{Once}, \texttt{upon}, \texttt{a})$$

$$p(\texttt{a}|\texttt{Once}, \texttt{upon})$$

$$p(\texttt{Once upon a time})$$

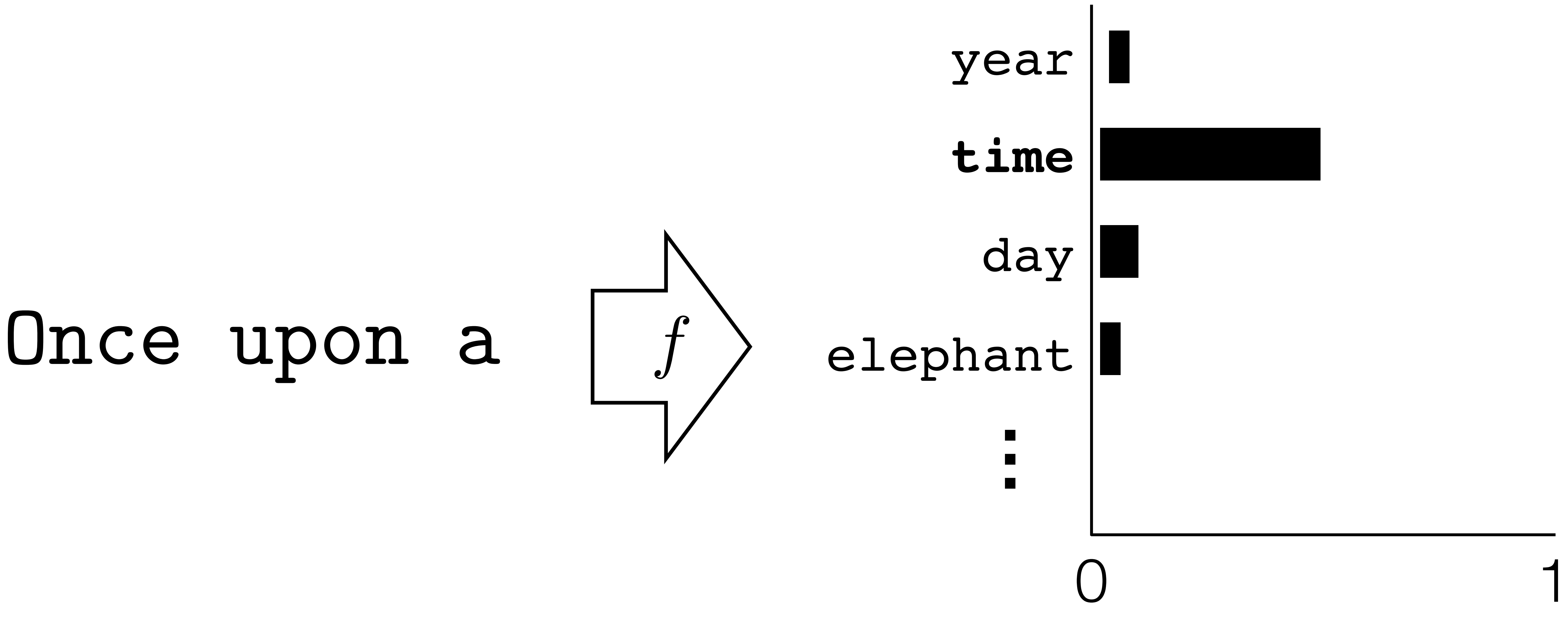
$$p(\texttt{Once})$$

$$p(\texttt{upon}|\texttt{Once})$$

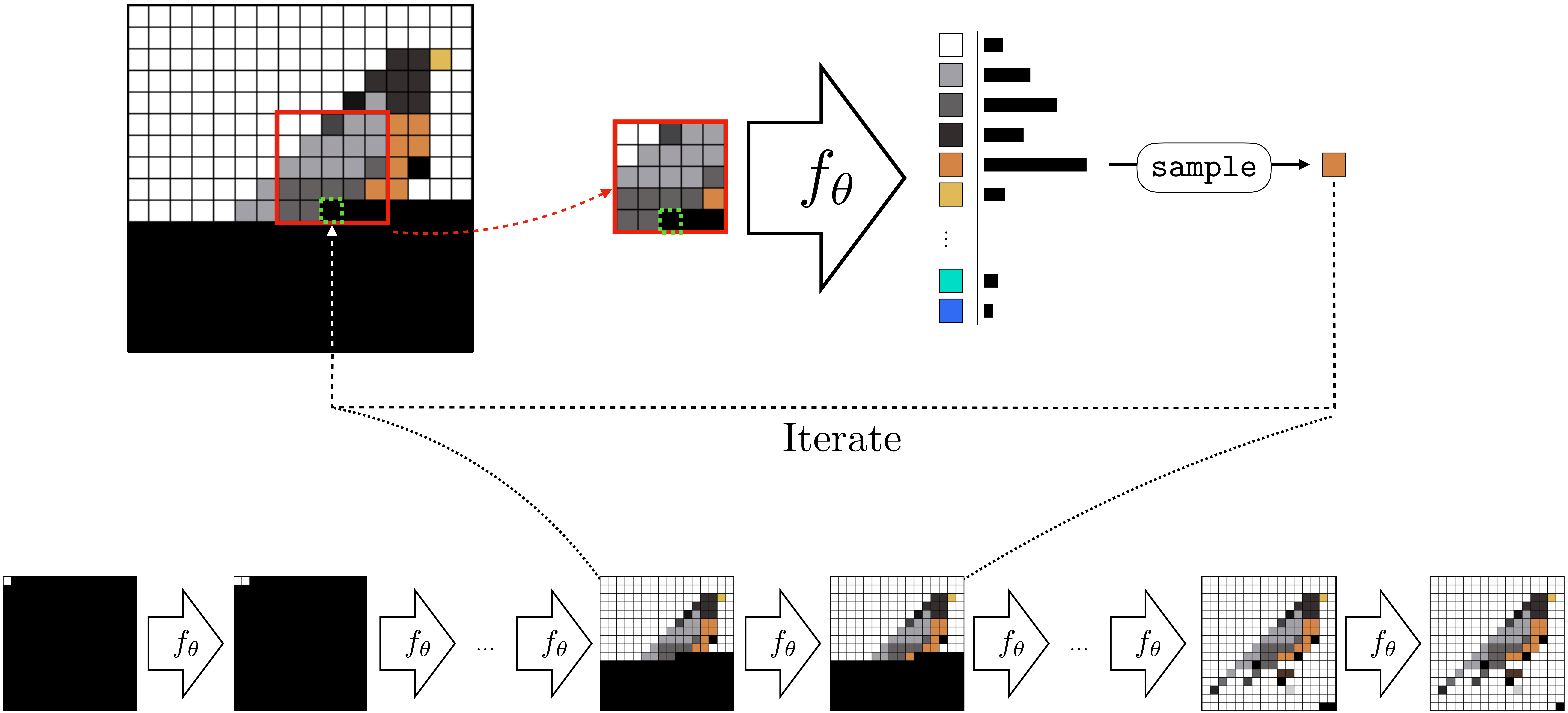
Modeling a sequence of words

How to model $p(\texttt{time}|\texttt{Once}, \texttt{upon}, \texttt{a})$?

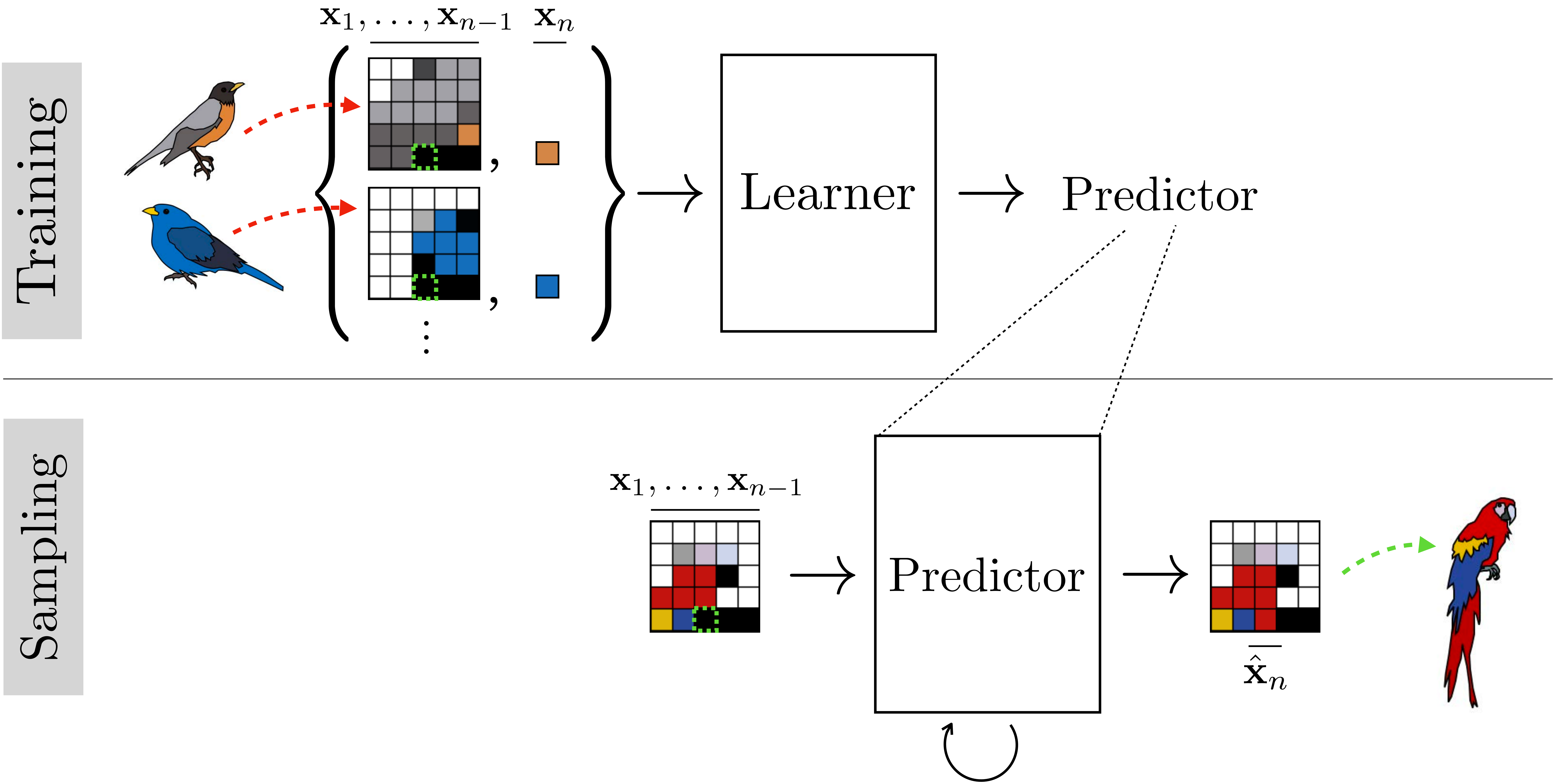
Just treat it as a next word classifier!



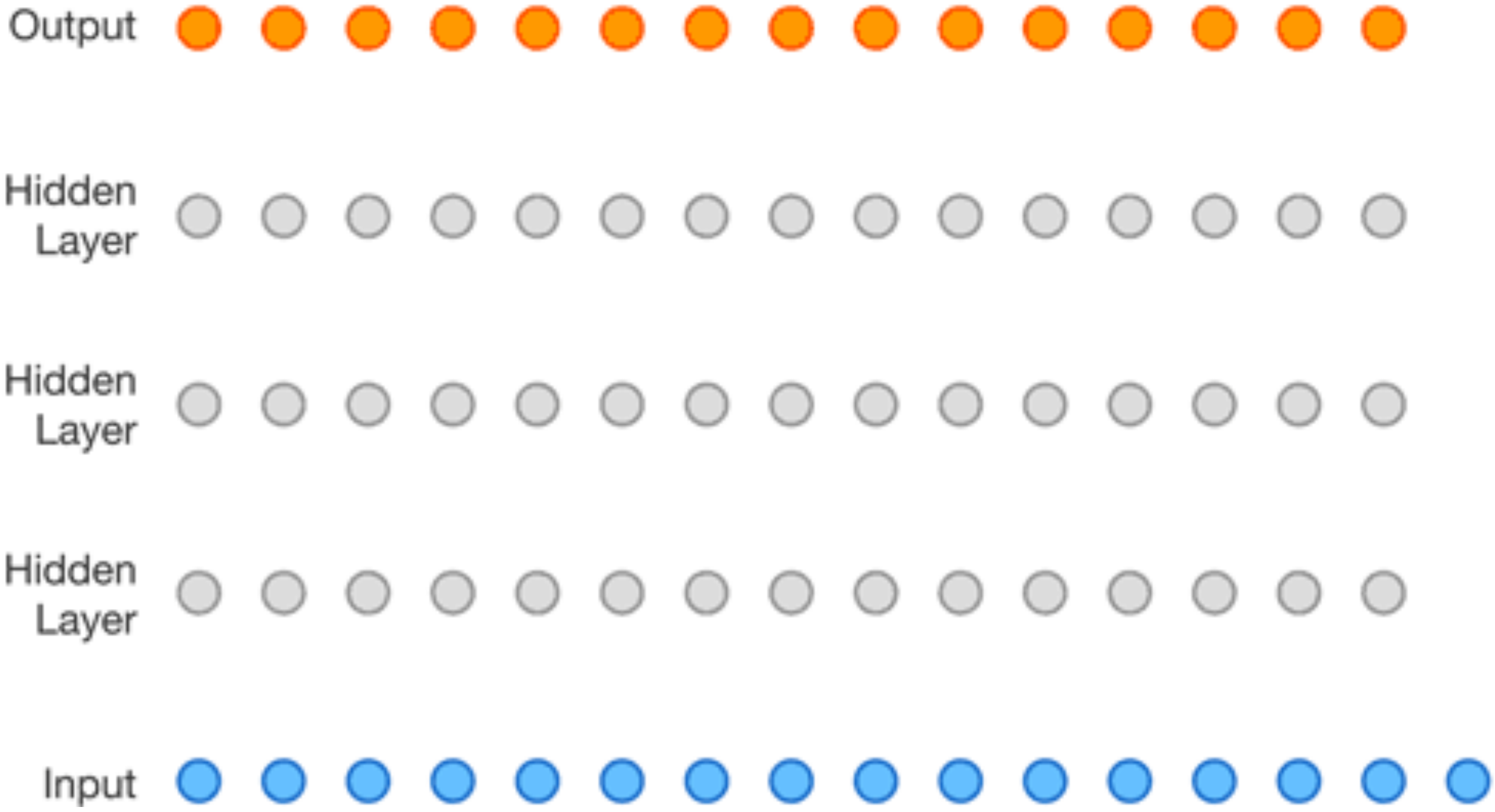
Autoregressive model of pixels




$\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}$
f_θ
Prediction
$\hat{\mathbf{x}}_n$
- ∞
log prob
0
$\odot$
Ground truth label
$\mathbf{x}_n$
0
Prob
1
Elementwise scores
$\mathbf{x}_n \odot \log \hat{\mathbf{x}}_n$
- ∞
0

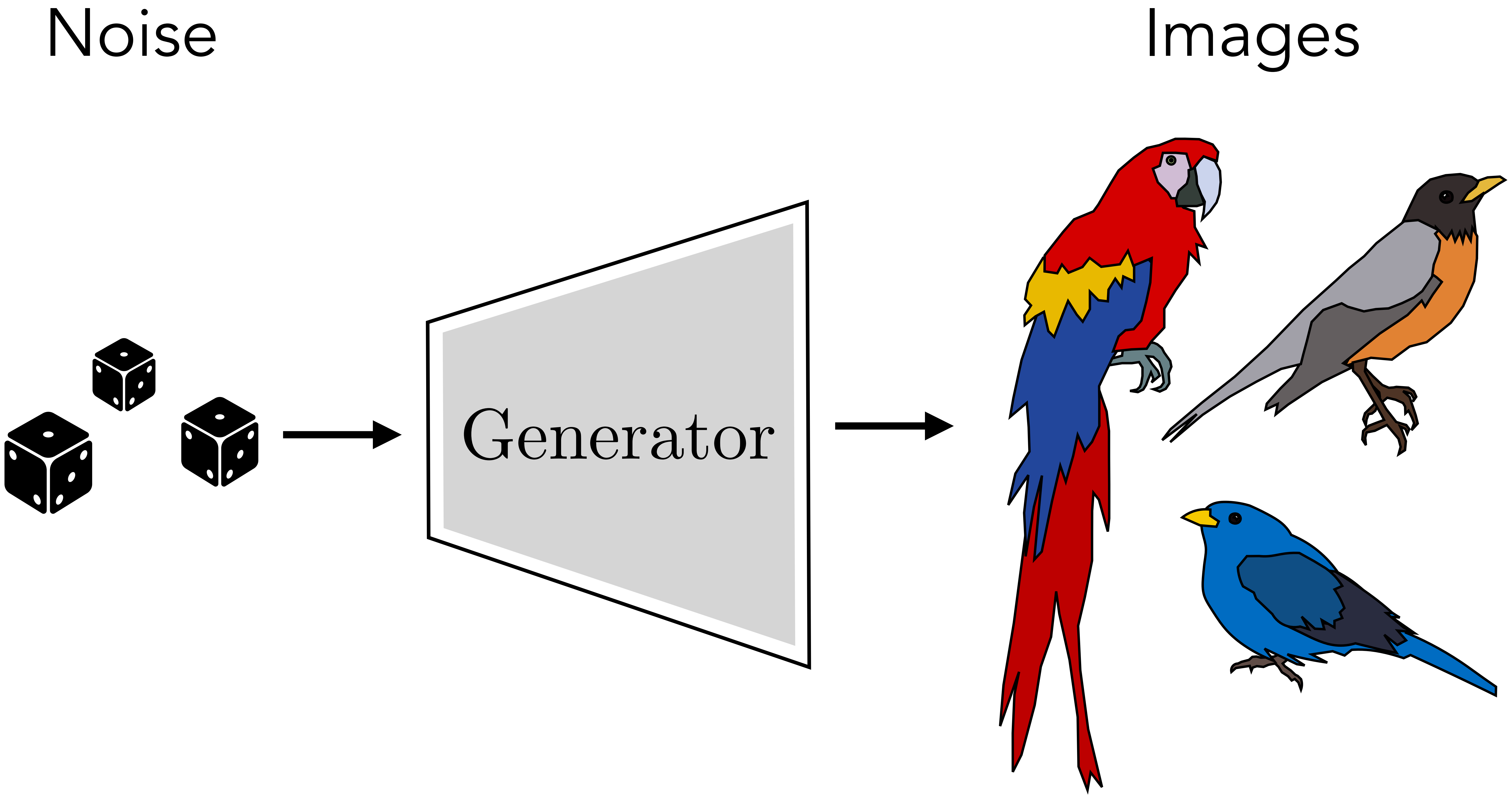


Training
$\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}$
$\mathbf{x}_n$
Learner
Predictor
Sampling
$\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}$
Predictor
$\hat{\mathbf{x}}_n$

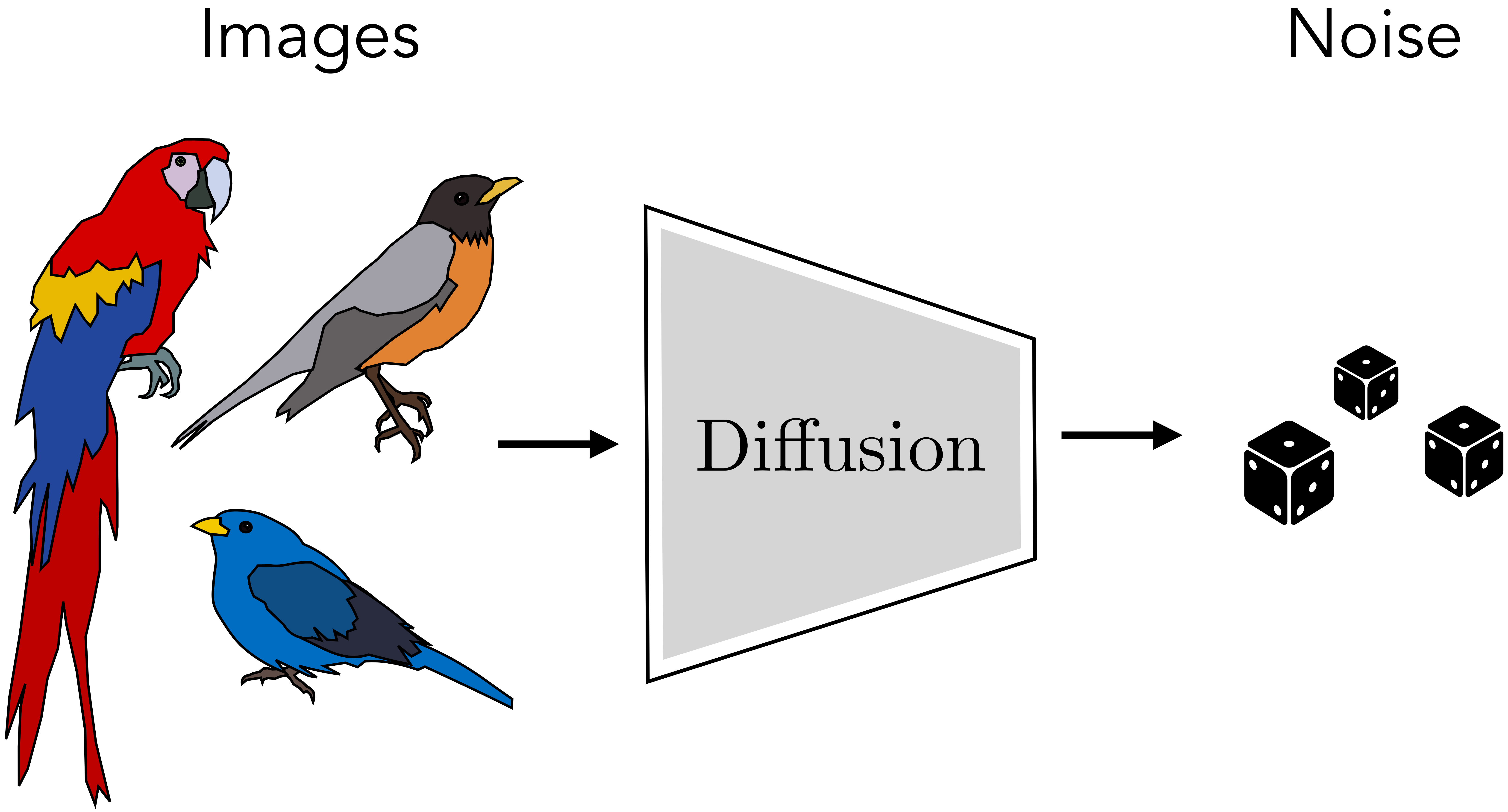


[**Wavenet**, https://deepmind.com/blog/wavenet-generative-model-raw-audio/]

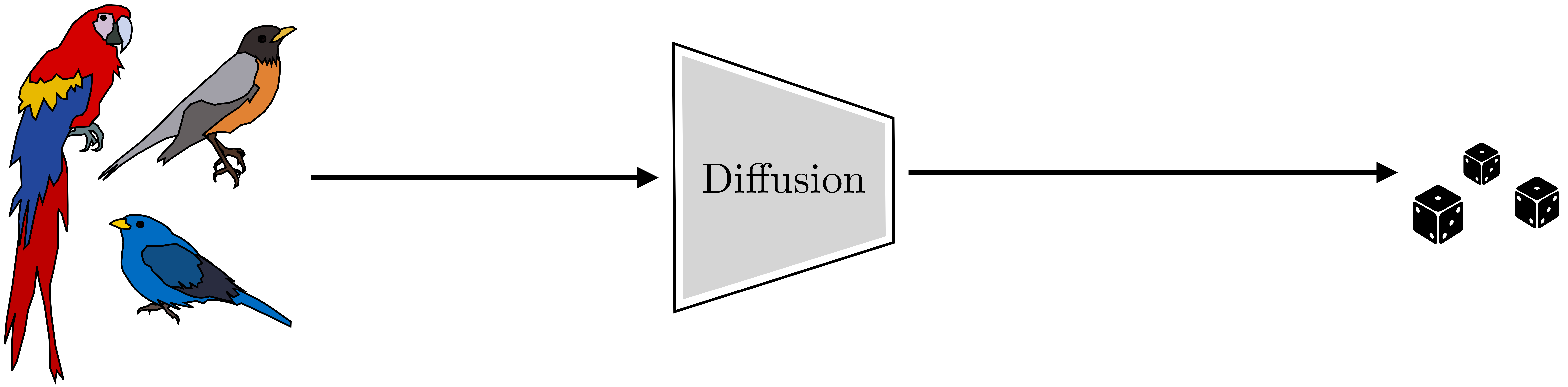
Diffusion models

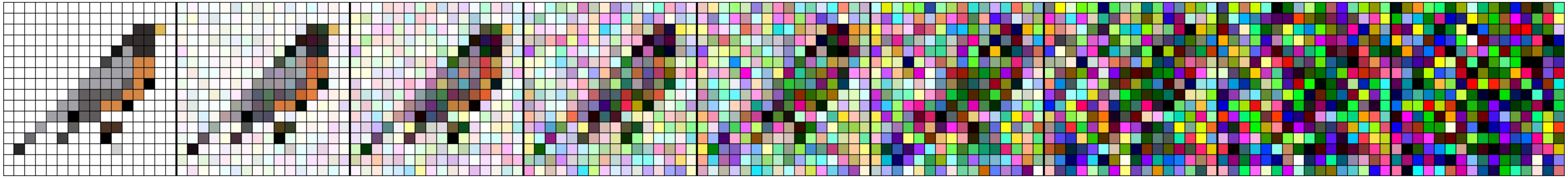


Diffusion models



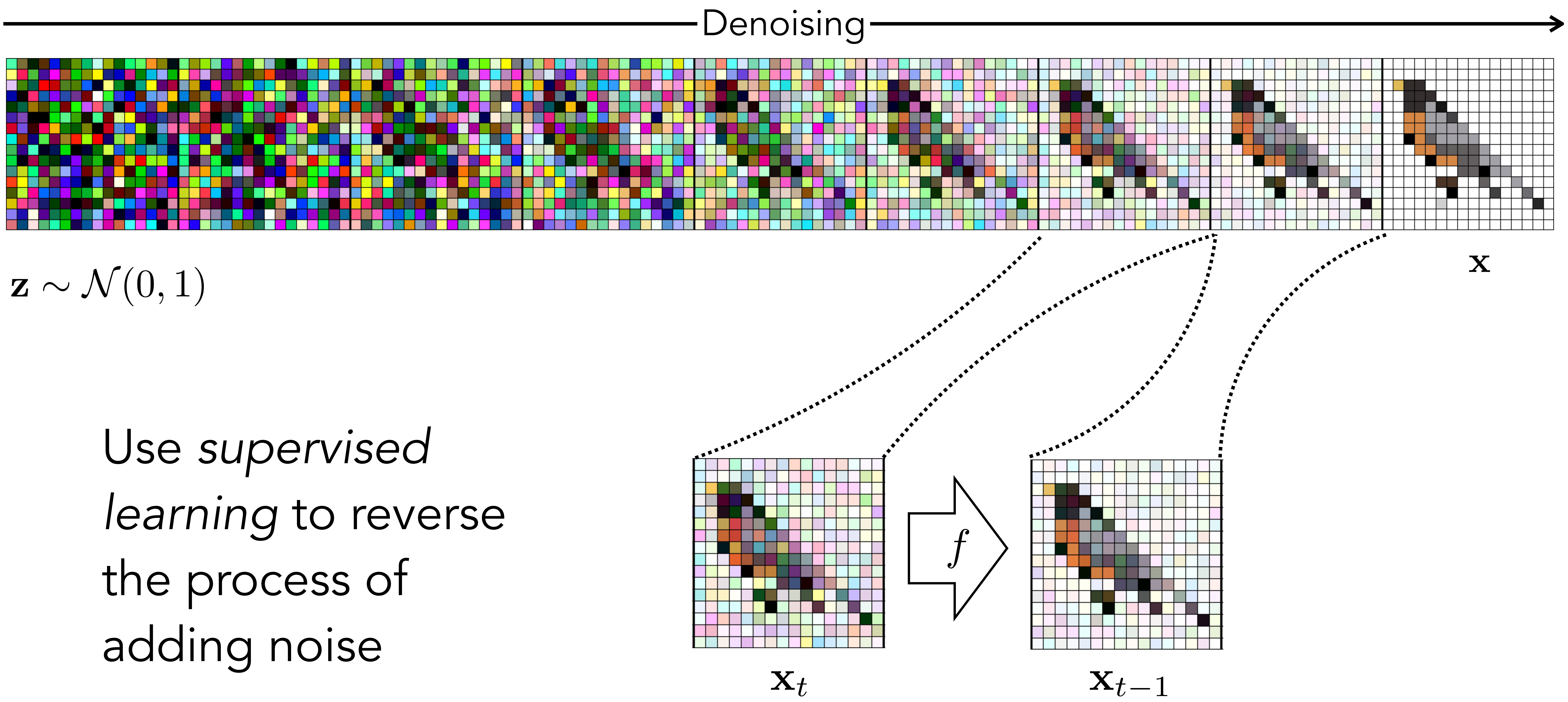
Diffusion models





Diffusion: Just add noise

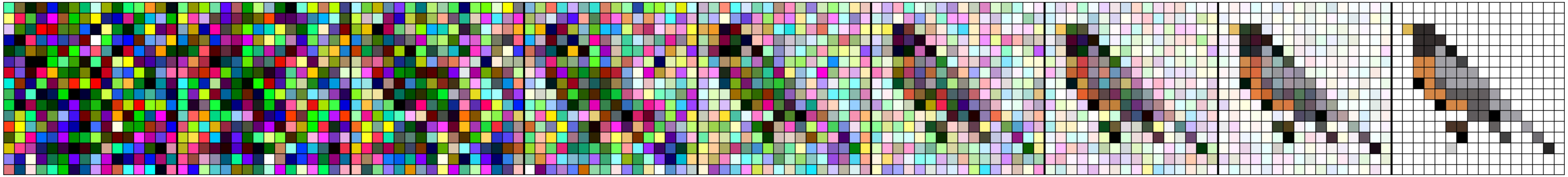
Diffusion models



Use *supervised learning* to reverse the process of adding noise

Diffusion models

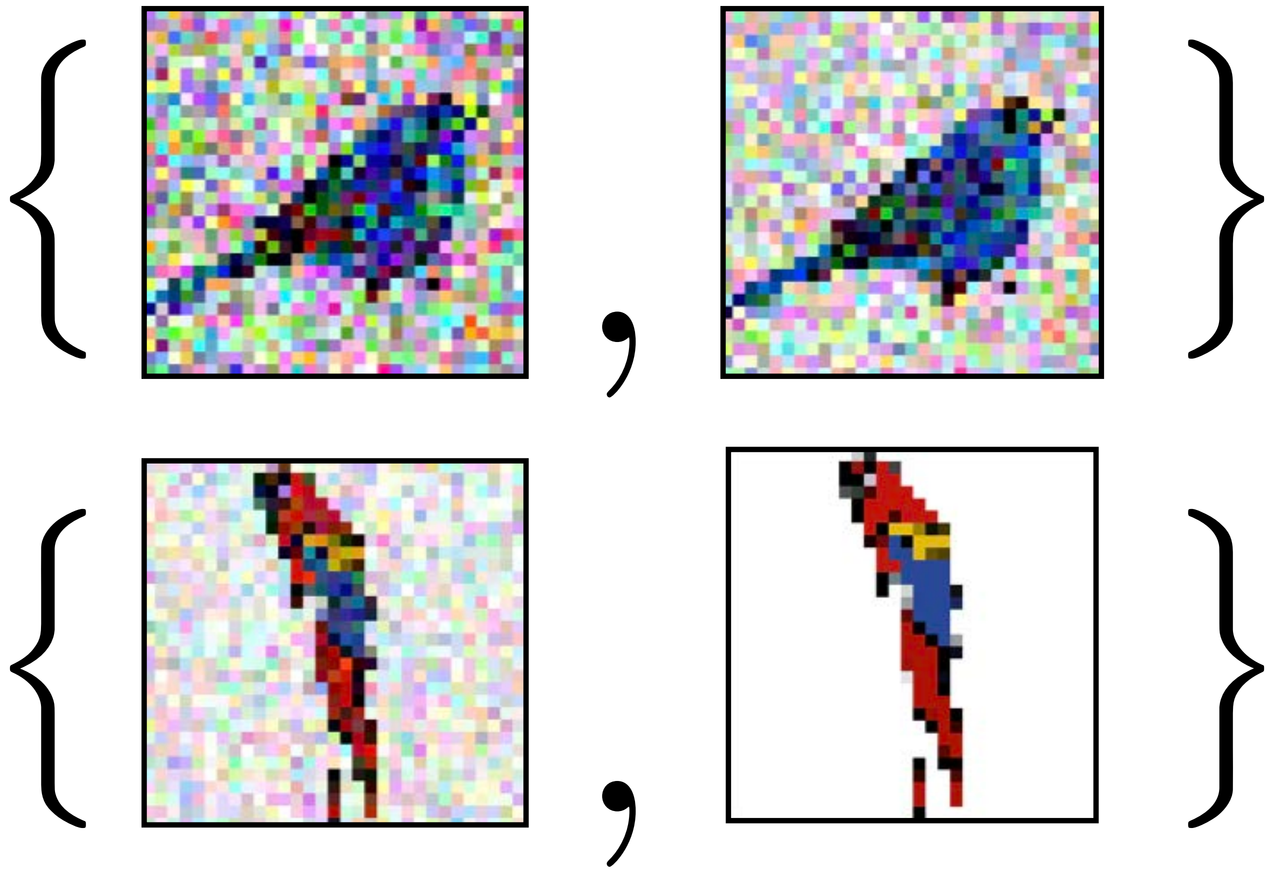
$\mathbf{z} \sim \mathcal{N}(0,1)$ $\mathbf{x}$



Denoising →

Training data

$\mathbf{x}_t$ $\mathbf{x}_{t-1}$



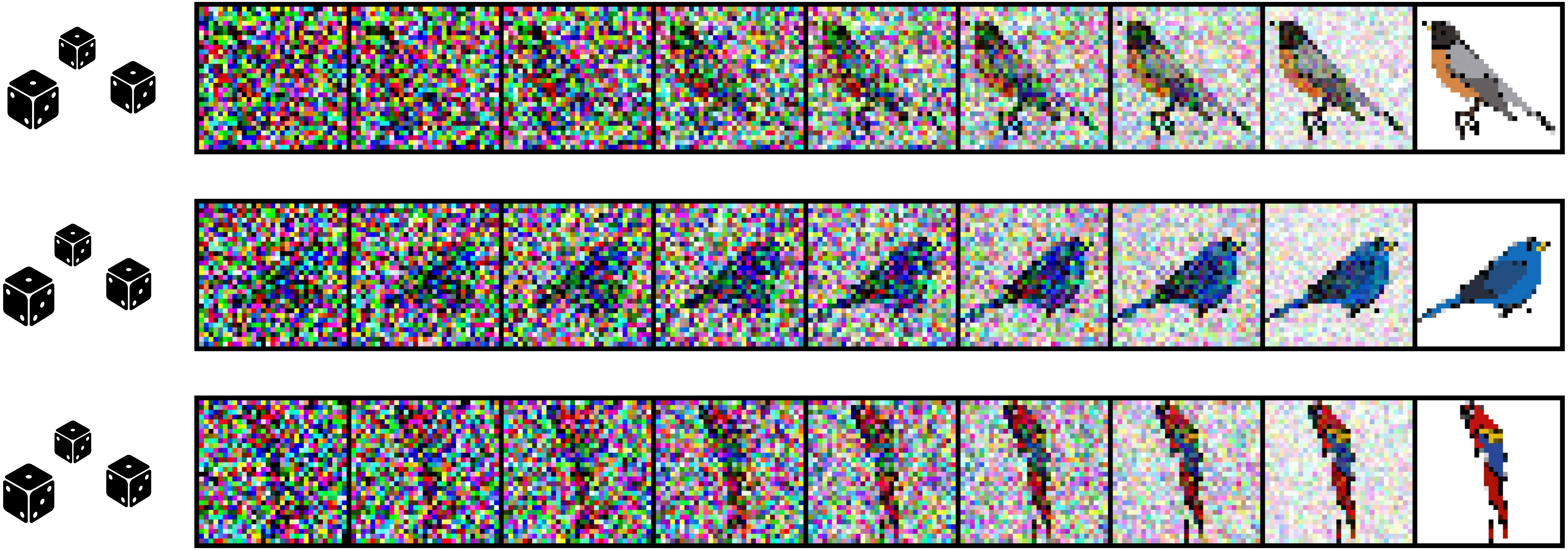
$\vdots$

$$\arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} \mathcal{L}(f(\mathbf{x}_t), \mathbf{x}_{t-1})$$

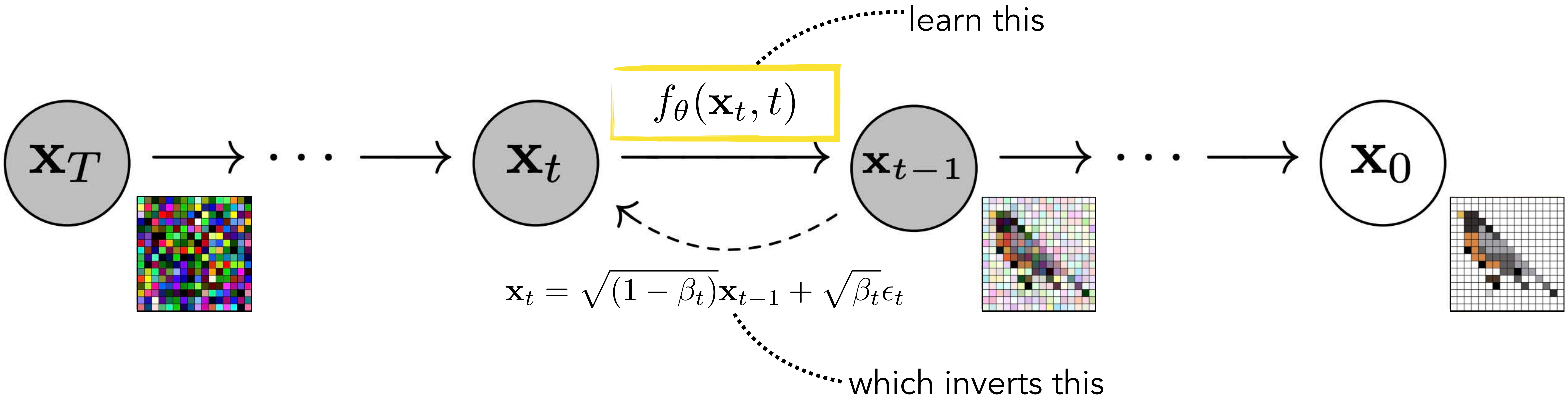
Converts generative modeling into a bunch of supervised prediction problems

Diffusion models

Different noise samples (dice rolls) result in different images



Gaussian diffusion models



Forward process:

$$\epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \qquad \mathbf{x}_t = \sqrt{(1-\beta_t)}\mathbf{x}_{t-1} + \sqrt{\beta_t}\epsilon_t$$

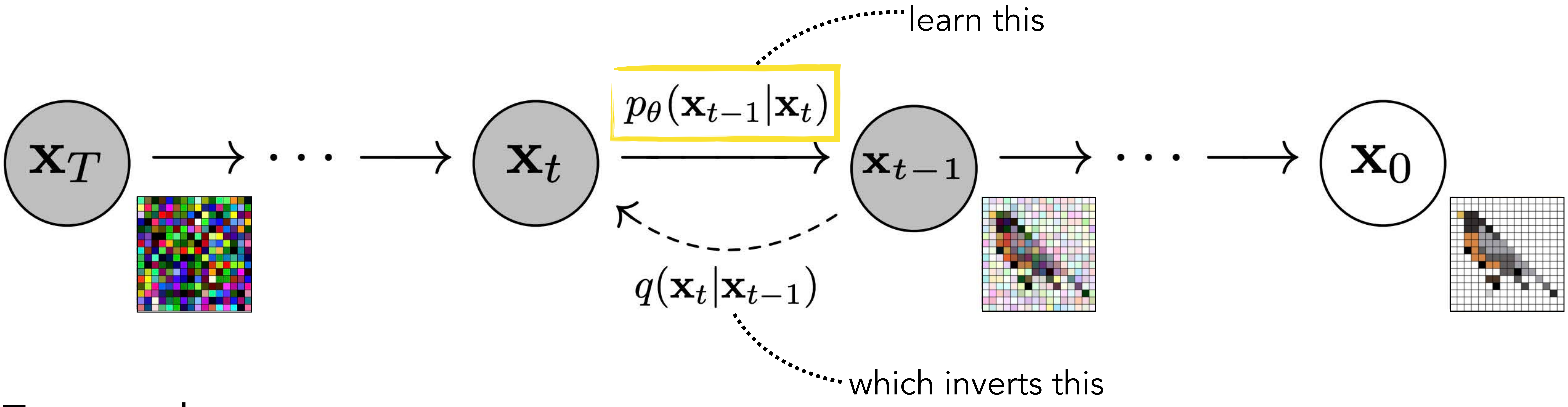
Reverse process:

$$\mu = f_\theta(\mathbf{x}_t, t) \qquad \mathbf{x}_{t-1} \sim \mathcal{N}(\mu, \sigma^2)$$

The variances, beta and sigma, are modeling choices. See Ho, Jain, and Abbeel for details.

[Fig adapted from Ho, Jain, Abbeel, 2020]

Gaussian diffusion models



Forward process:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t)$$

Reverse process:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(f_\theta(\mathbf{x}_t, t), \sigma^2)$$

The variances, beta and sigma, are modeling choices. See Ho, Jain, and Abbeel for details.

[Fig adapted from Ho, Jain, Abbeel, 2020]

Stripped down training algorithm

Algorithm 1.2: Training a diffusion model.

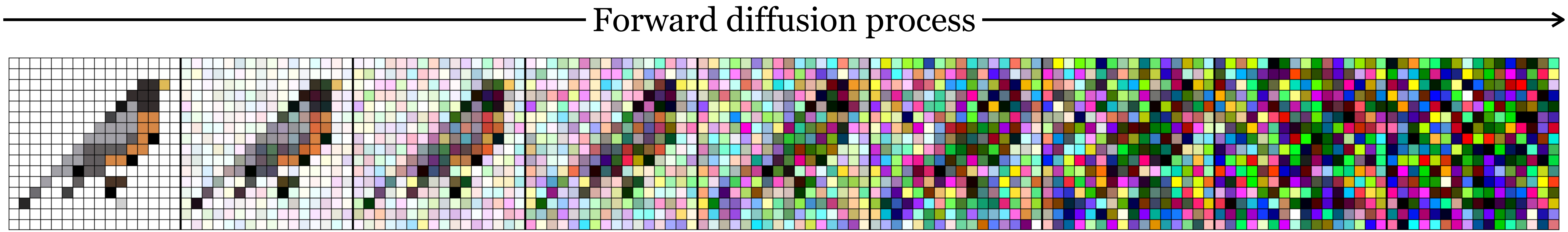
Input: training data $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$
Output: trained model f_θ
Generate training sequences via diffusion:
for $i = 1, \ldots, N$ **do**
  for $t = 1, \ldots, T$ **do**
    $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\mathbf{x}_t^{(i)} \leftarrow \sqrt{(1-\beta_t)}\mathbf{x}_{t-1}^{(i)} + \sqrt{\beta_t}\epsilon_t$

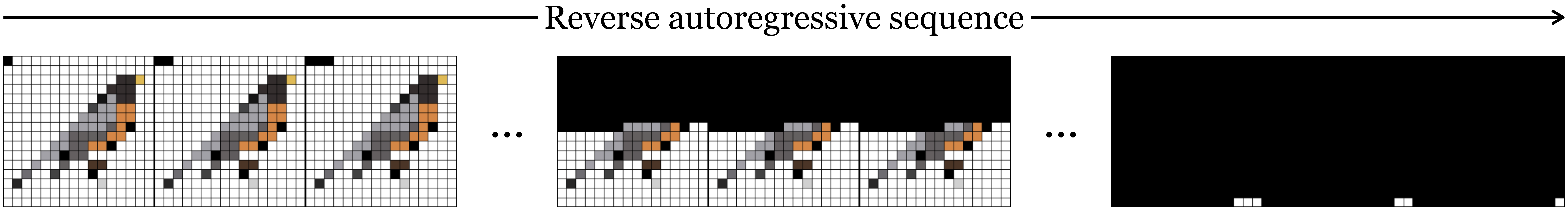
Train denoiser f_θ to reverse these sequences:
$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \sum_{t=1}^{T} \mathcal{L}(f_\theta(\mathbf{x}_t^{(i)}, t), \mathbf{x}_{t-1}^{(i)}))$
Return: f_{θ^*}

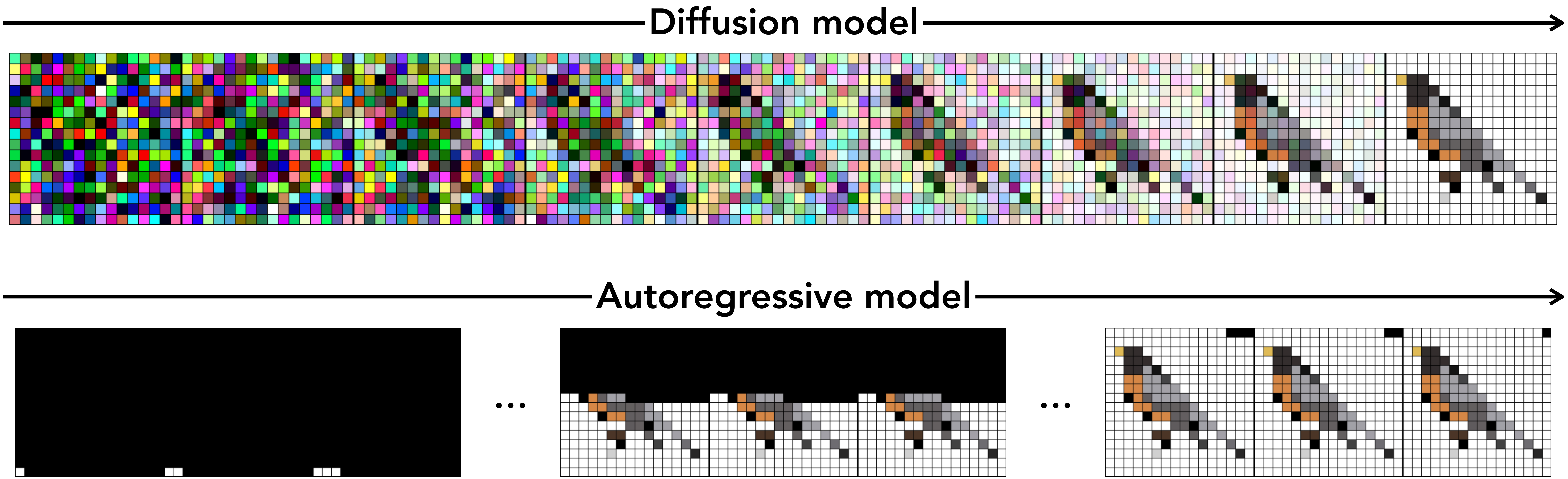
Colab:

https://colab.research.google.com/drive/1YUFwGs0z0lEaBUpSdJIEZtCATe44TUjw?usp=sharing

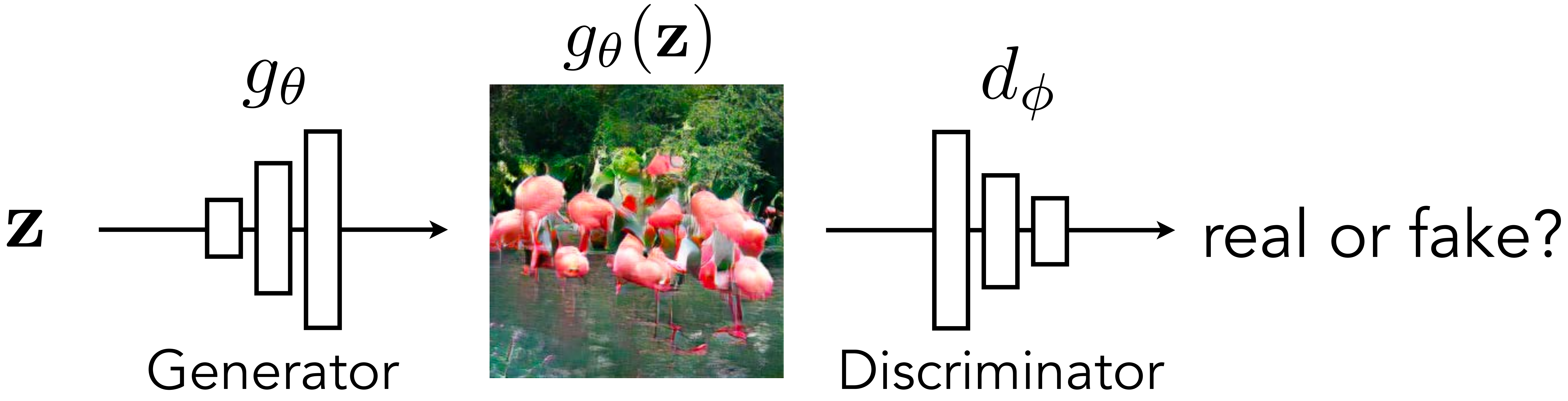
Autoregressive models vs diffusion models







Concept #3: **A common strategy is to turn generative modeling into a sequence of supervised learning problems**

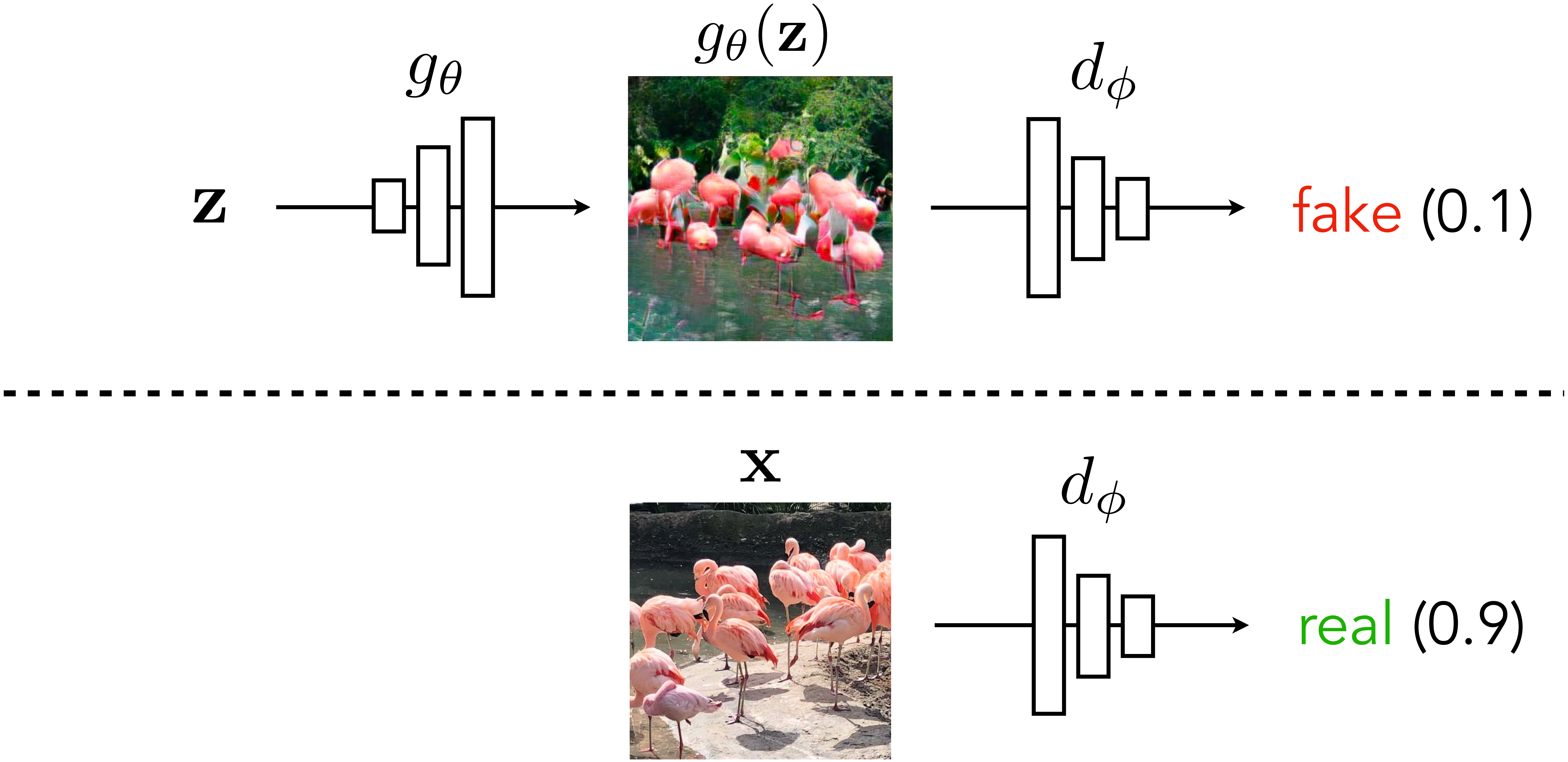


Generative Adversarial Networks (GANs)

g tries to synthesize fake images that fool d

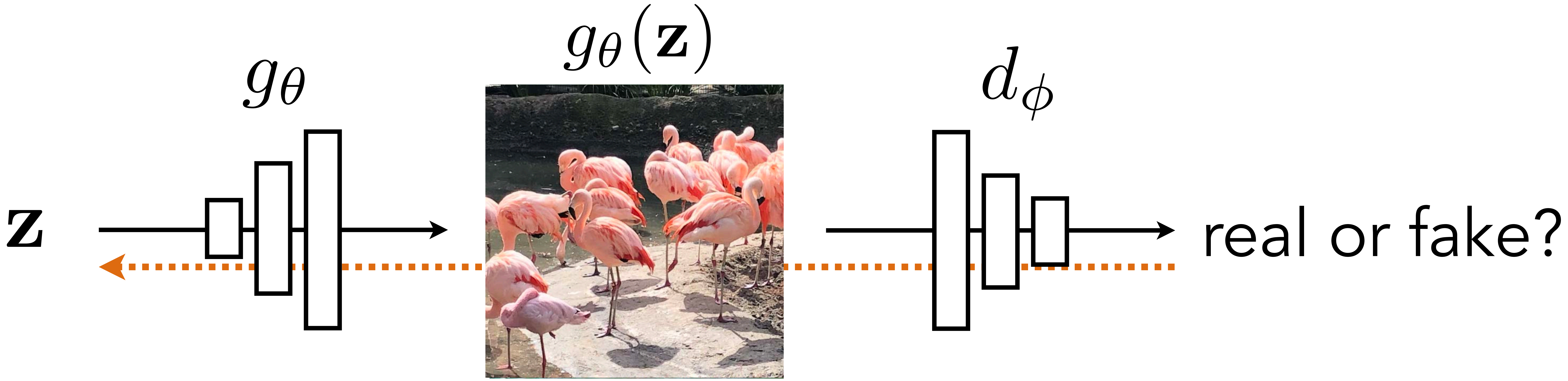
g tries to identify the fakes

[Goodfellow et al., 2014]



$$d^*_\phi = \arg\max_\phi \mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}[\log d_\phi(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\log(1-d_\phi(g_\theta(\mathbf{z})))]$$

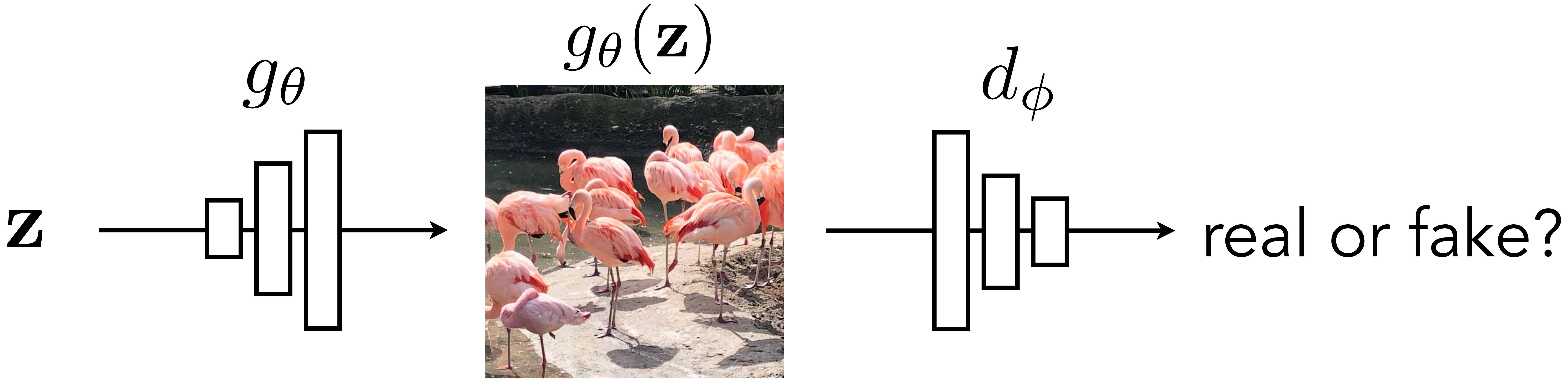
[Goodfellow et al., 2014]



g tries to synthesize fake images that *fool* d:

$$\arg\min_\theta \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\log(1-d^*_\phi(g_\theta(\mathbf{z})))]$$

[Goodfellow et al., 2014]

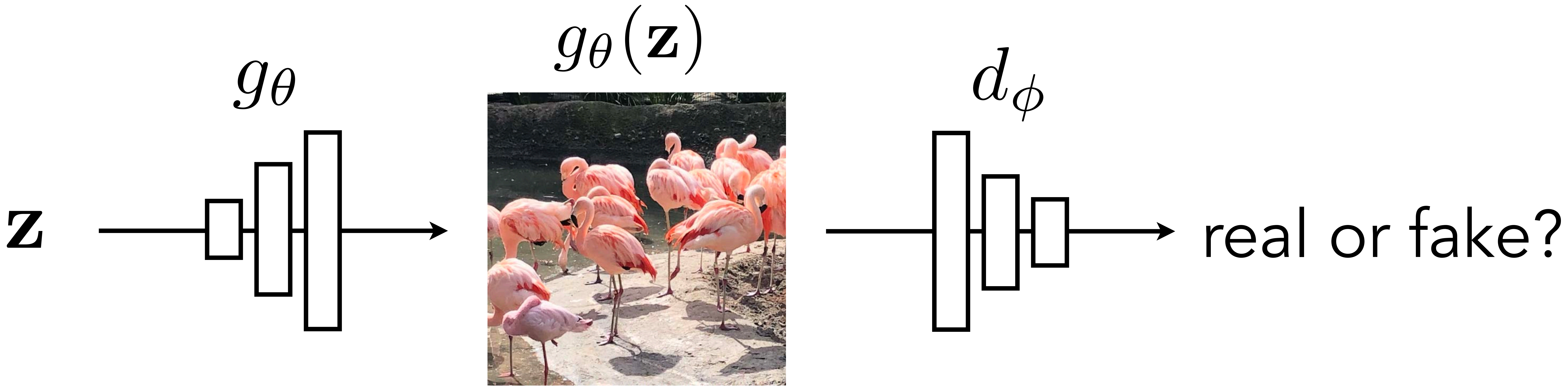


g tries to synthesize fake images that *fool* the *best* d:

$$\arg\min_{\theta}\max_{\phi}\mathbb{E}_{\mathbf{x}\sim p_{\mathtt{data}}}[\log d_\phi(\mathbf{x})]+\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\log(1-d_\phi(g_\theta(\mathbf{z})))]$$

[Goodfellow et al., 2014]

GANs — Training

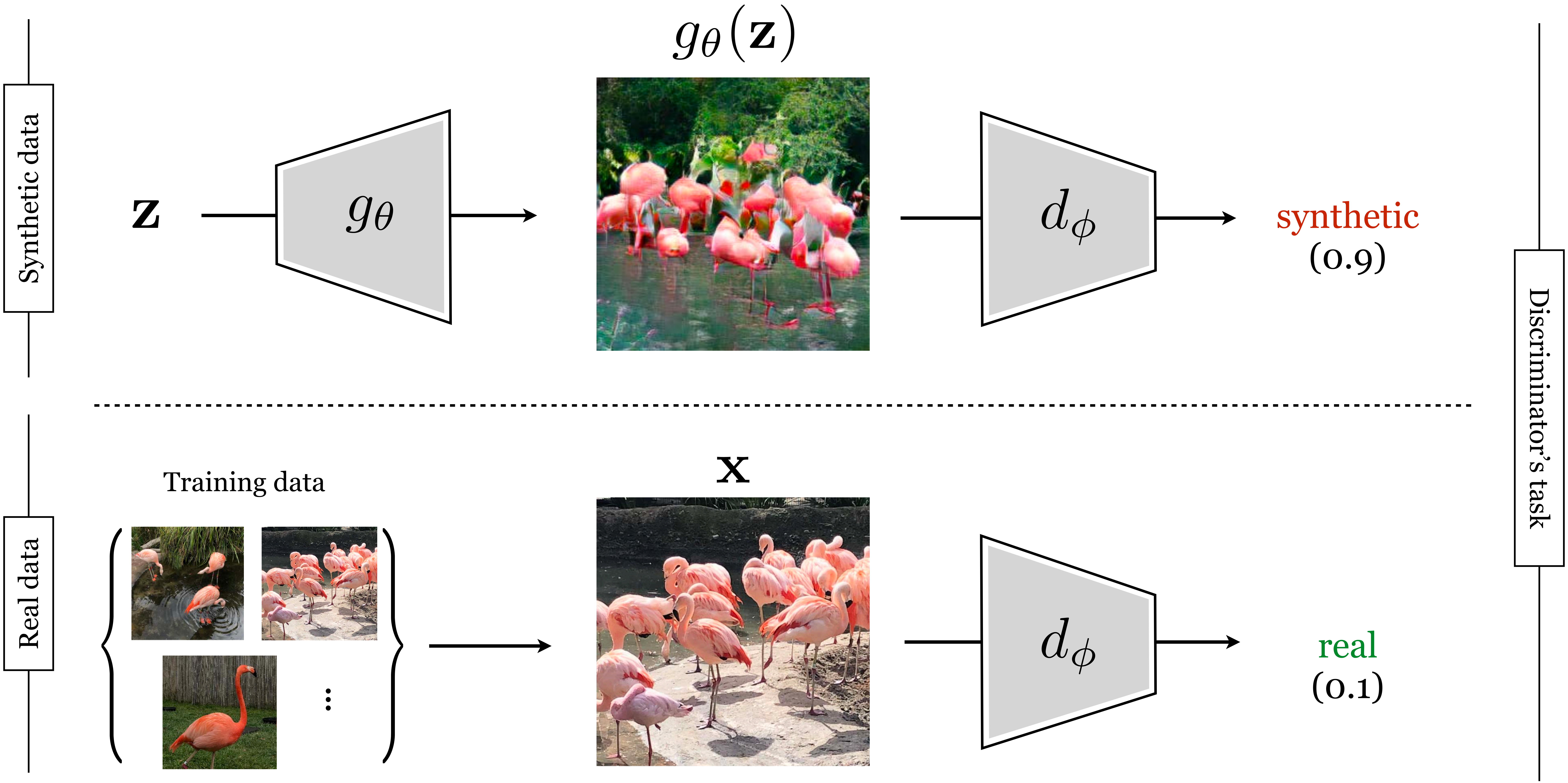


g tries to synthesize fake images that fool d

d tries to identify the fakes

- Training: iterate between training d and g with backprop.
- Global optimum when g reproduces data distribution.

[Goodfellow et al., 2014]


Synthetic data
$g_\theta(\mathbf{z})$
$\mathbf{z}$
g_θ
d_ϕ
synthetic
(0.9)
Discriminator's task
Real data
Training data
$\mathbf{x}$
d_ϕ
real
(0.1)