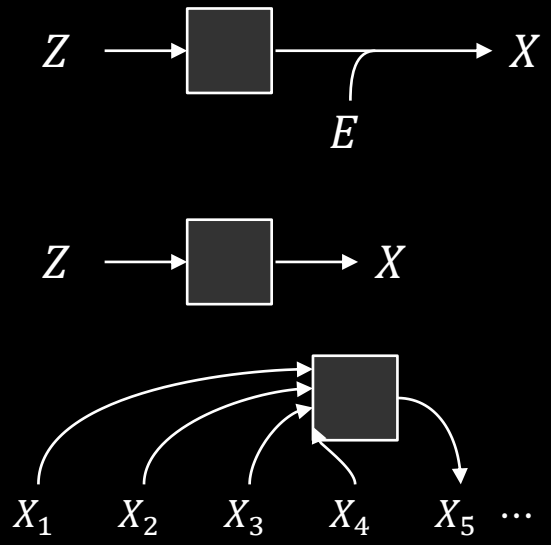
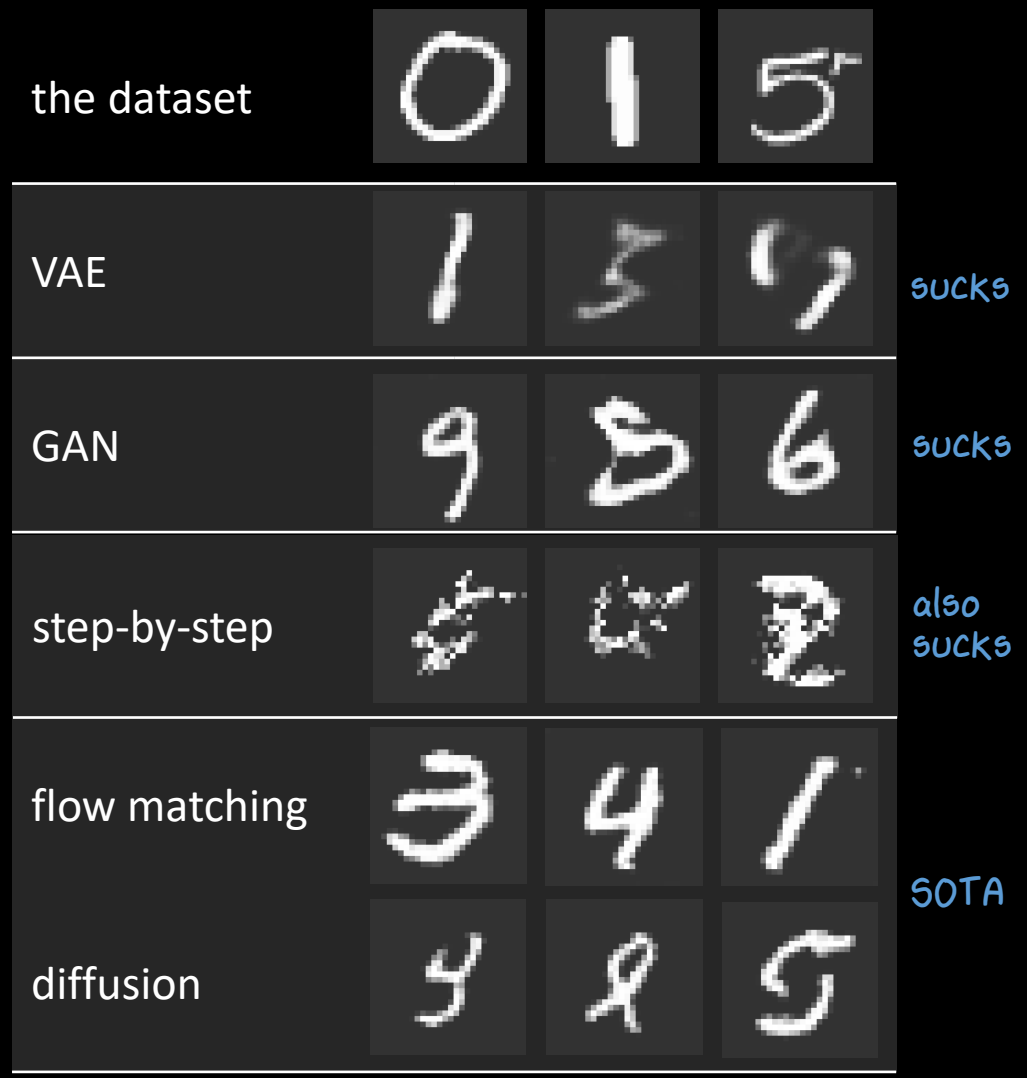
It’s easier to learn to generate step-by-step than one-shot



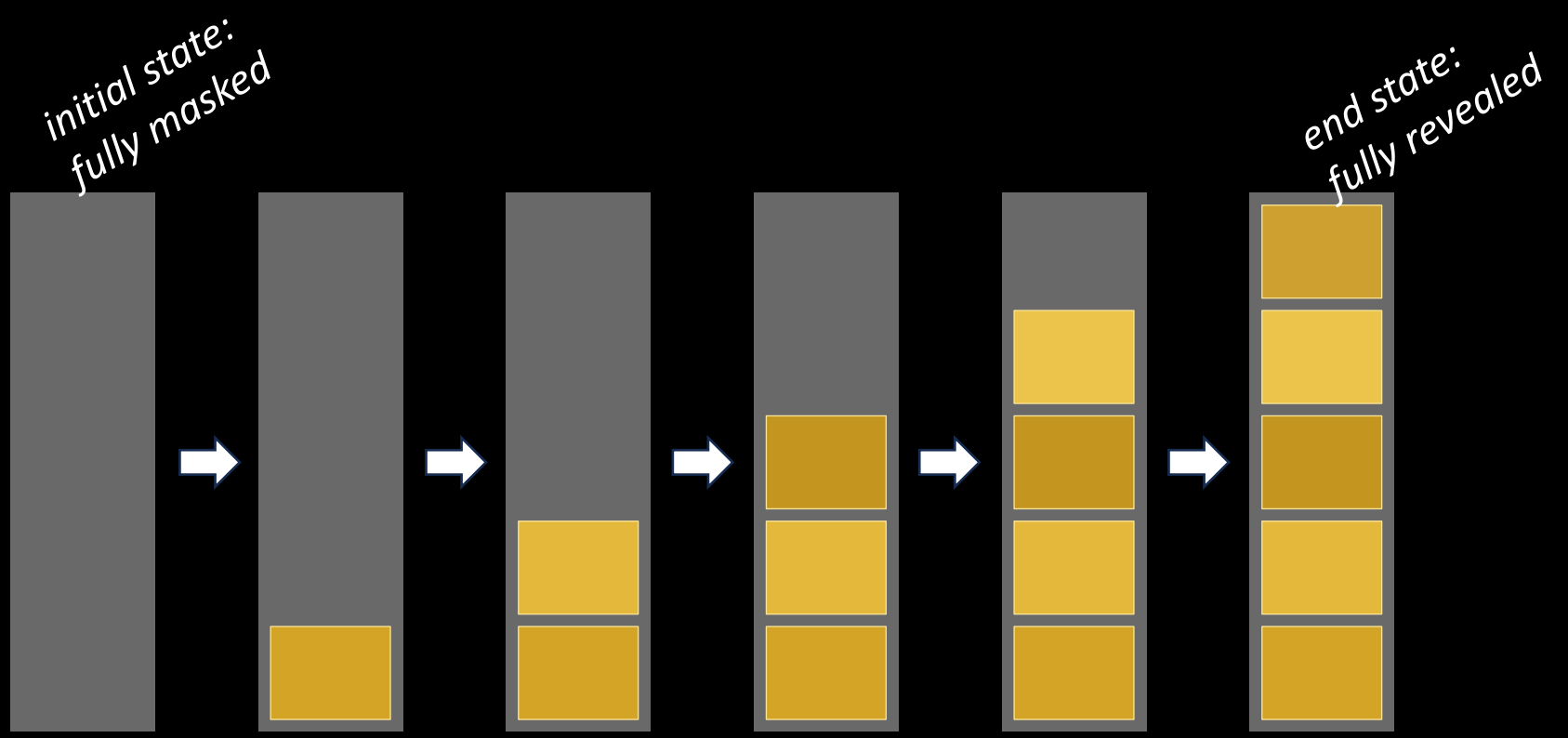


What are the pros and cons of learning step-by-step generation *versus* one-shot generation?

Brainwave: generate step by step (autoregressively), as with a transformer.

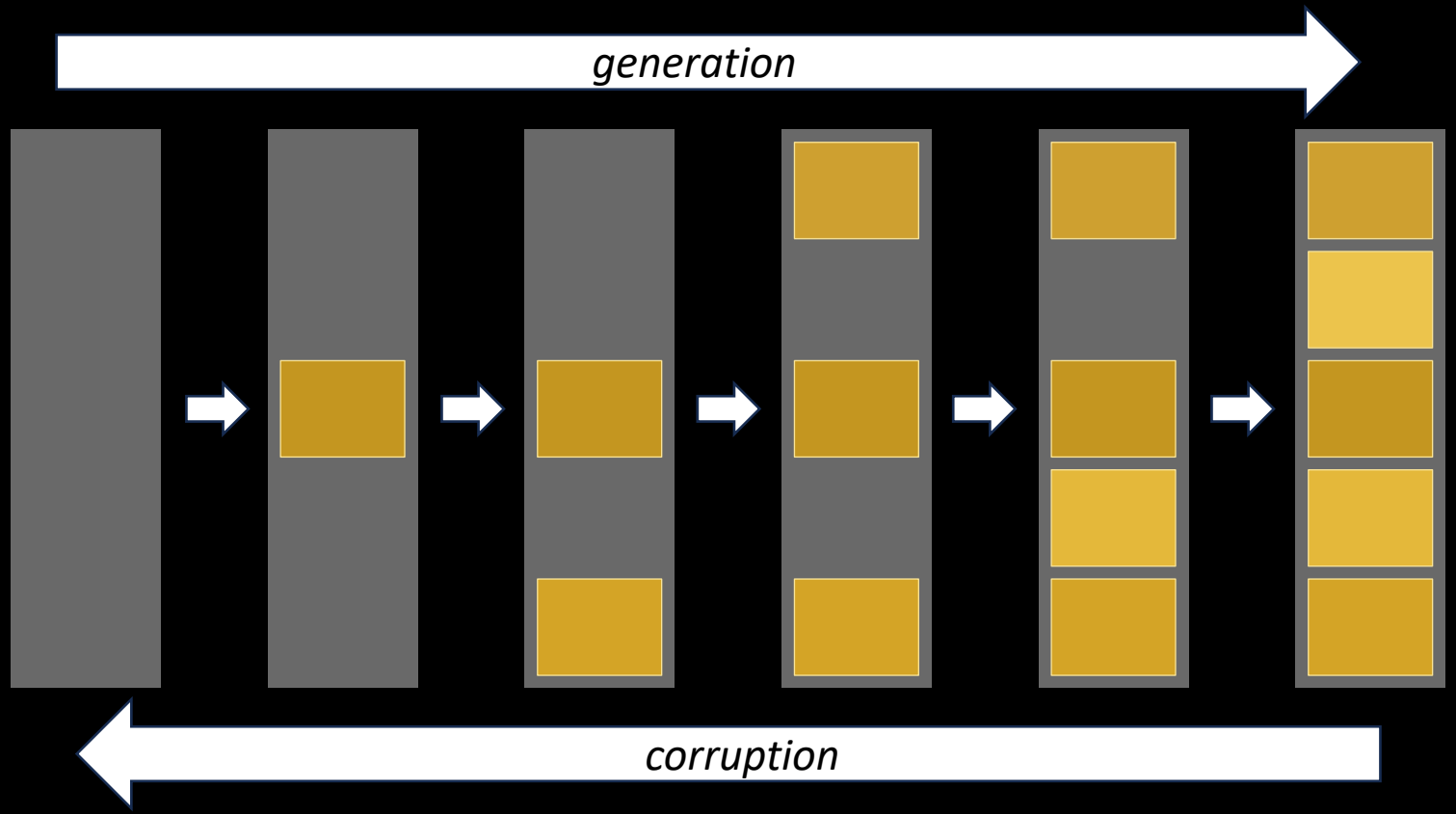
There is a more general way to think about step-by-step generation ...

PixelCNN, pixel by pixel image generation in raster order: van den Oord, Kalchbrenner and Kavukcuoglu, 2016.



We can see token-by-token generation as state evolution

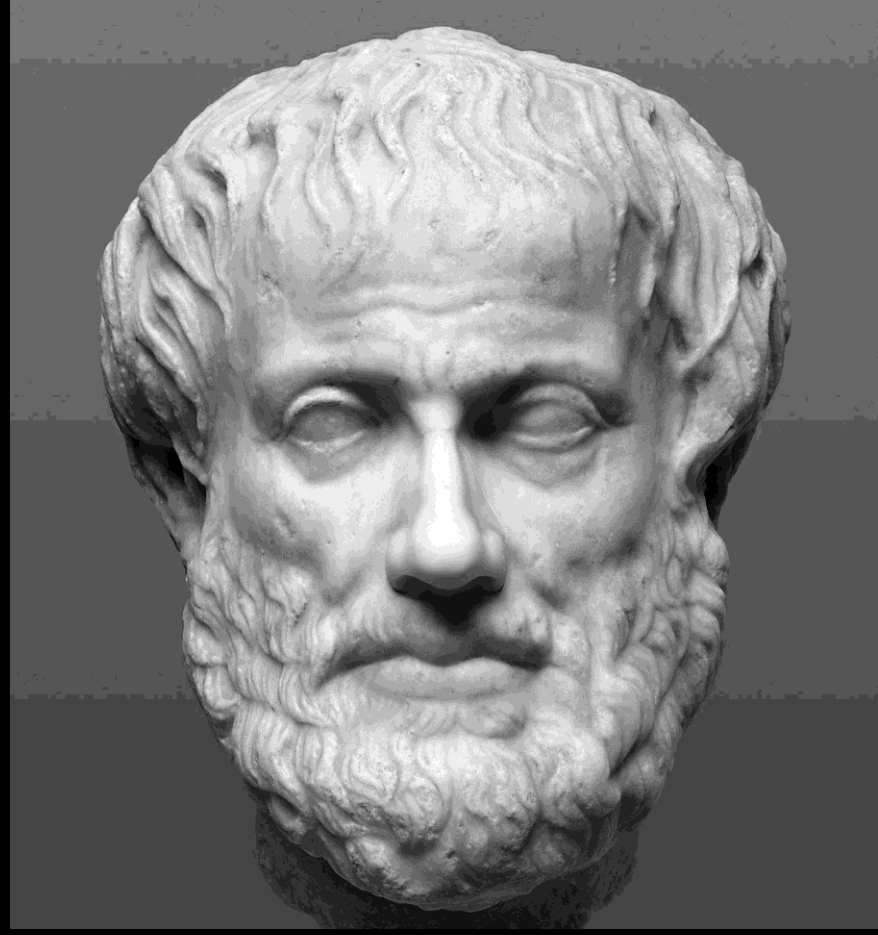
Generation and corruption



We can see token-by-token generation as state evolution

We can generate the tokens in any order

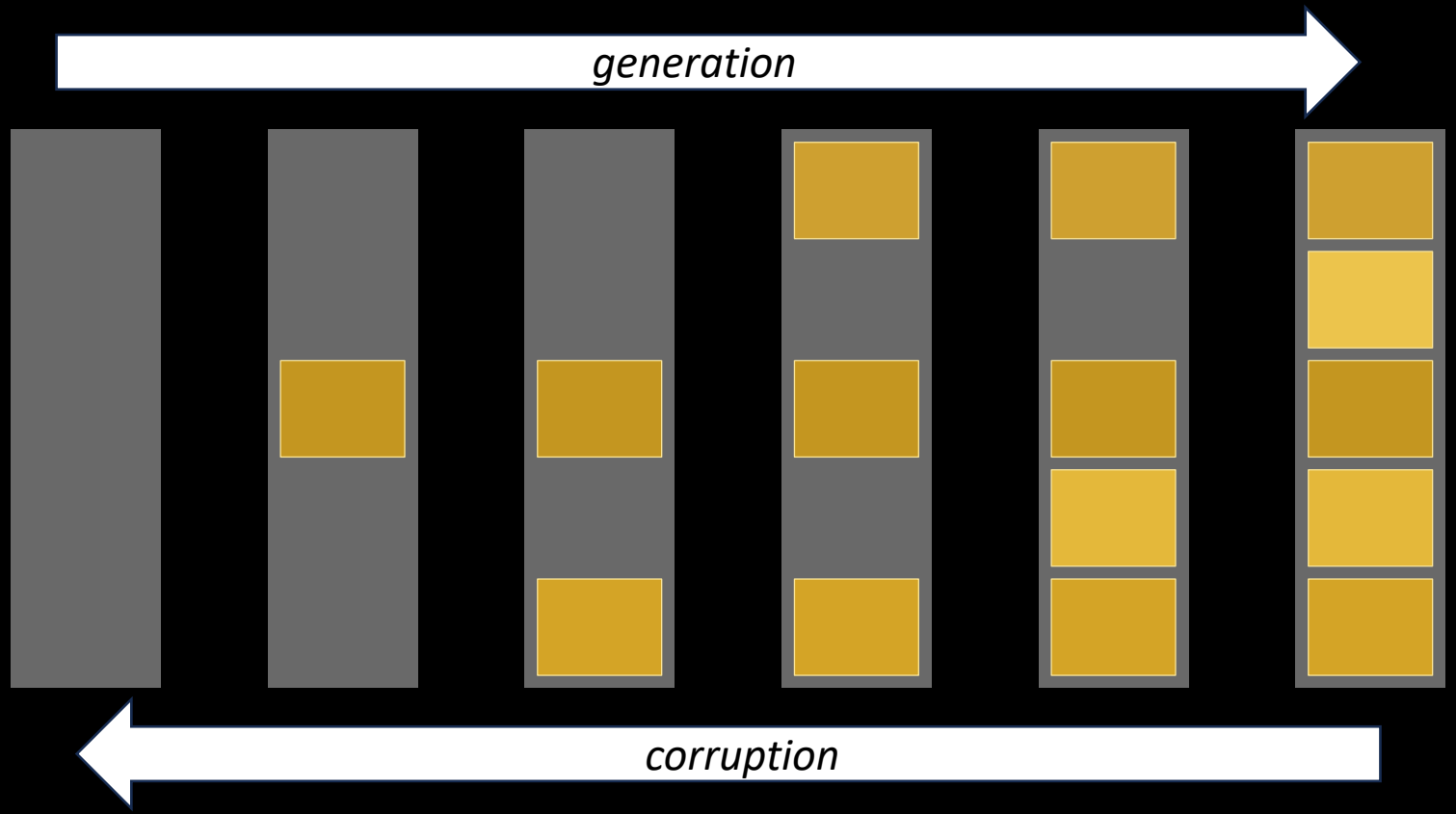
The evolution reads both as generation and as corruption



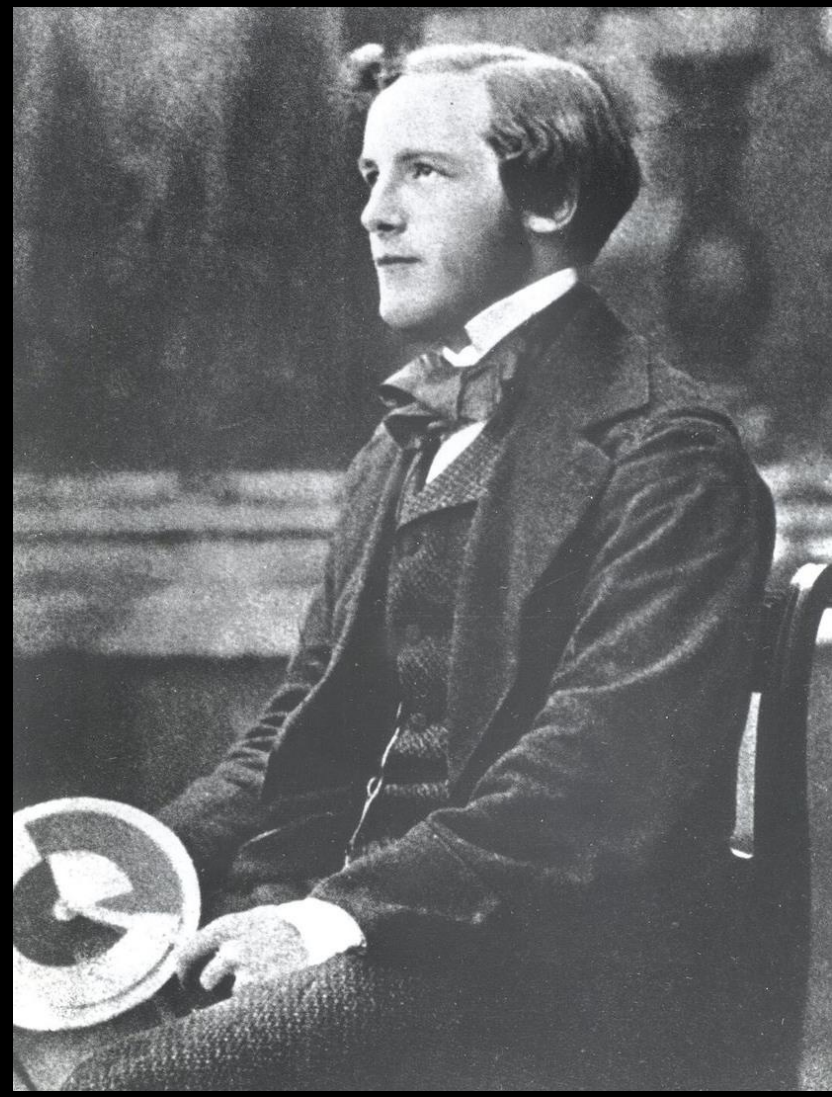
"Our next task is to study coming-to-be and passing-away. […] Further, we are to study growth and 'alteration'. We must inquire what each of them is; and whether 'alteration' is to be identified with coming-to-be, or whether to these different names there correspond two separate processes with distinct natures.

Aristotle, *On Generation and Corruption*

Generation and corruption

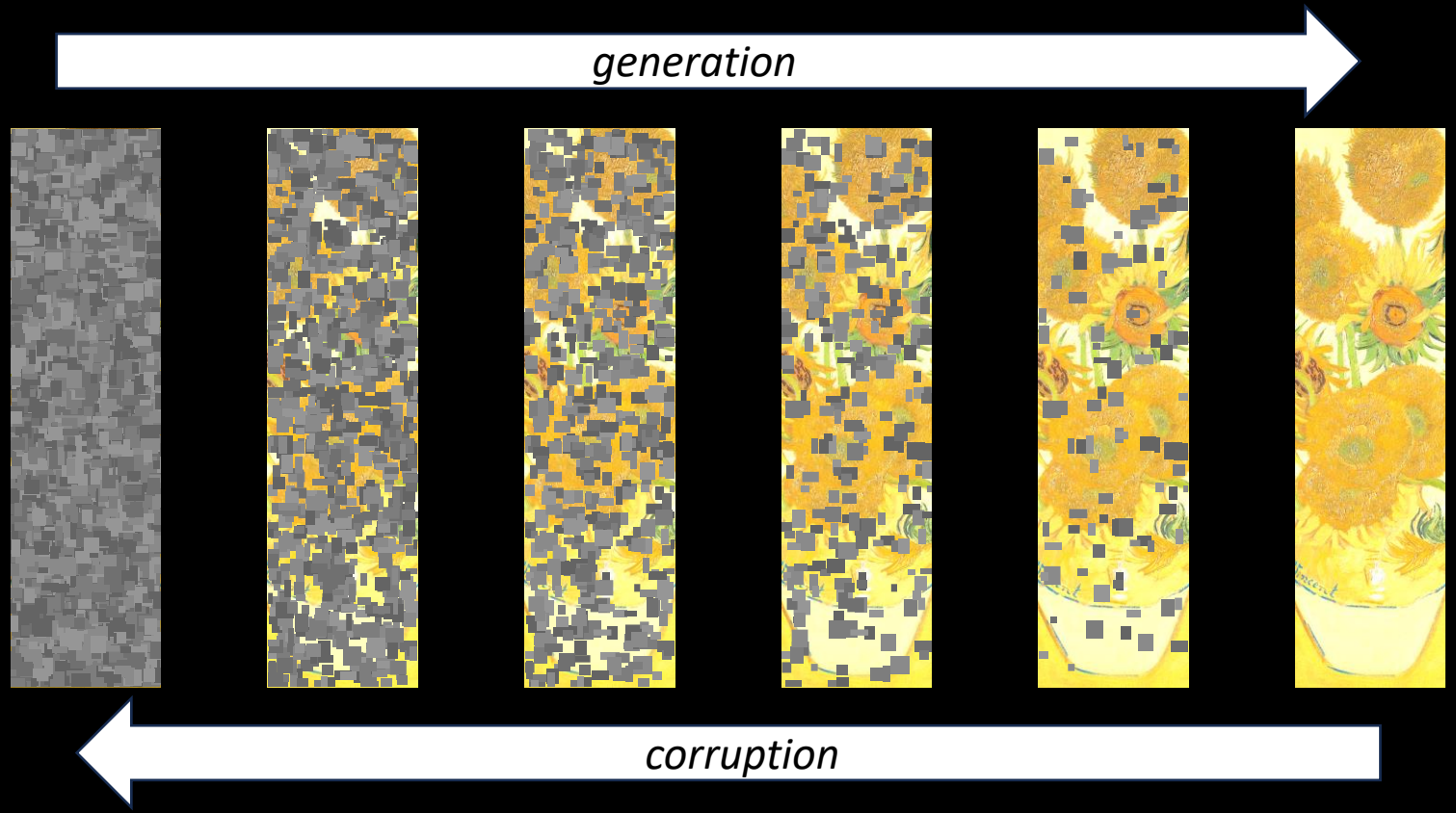


We can see token-by-token generation as state evolution
We can generate the tokens in any order
The evolution reads both as generation and as corruption



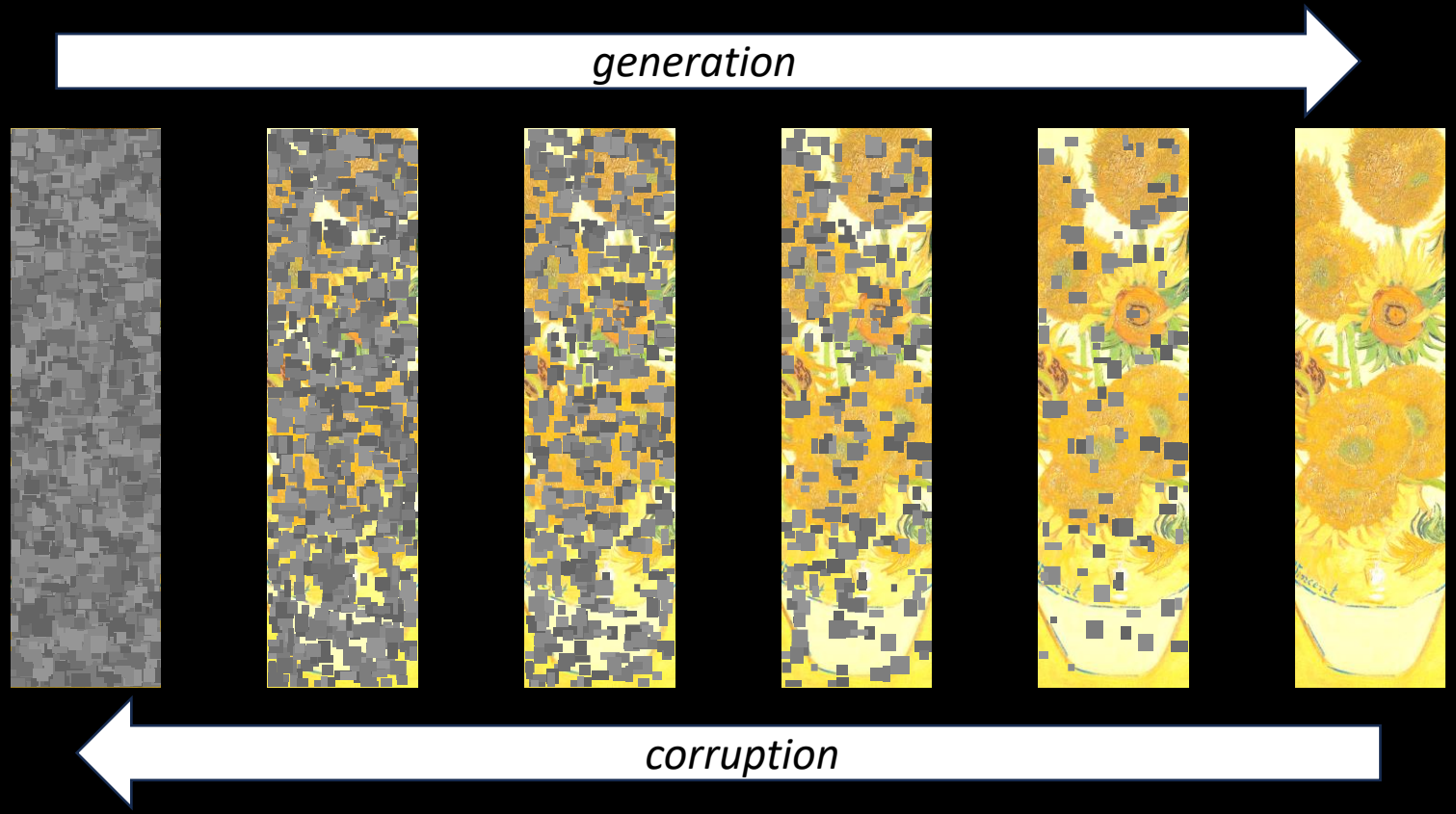
James Clerk Maxwell, first professor of the Cavendish Laboratory
[Photograph: National Museums Scotland]

Generation and corruption

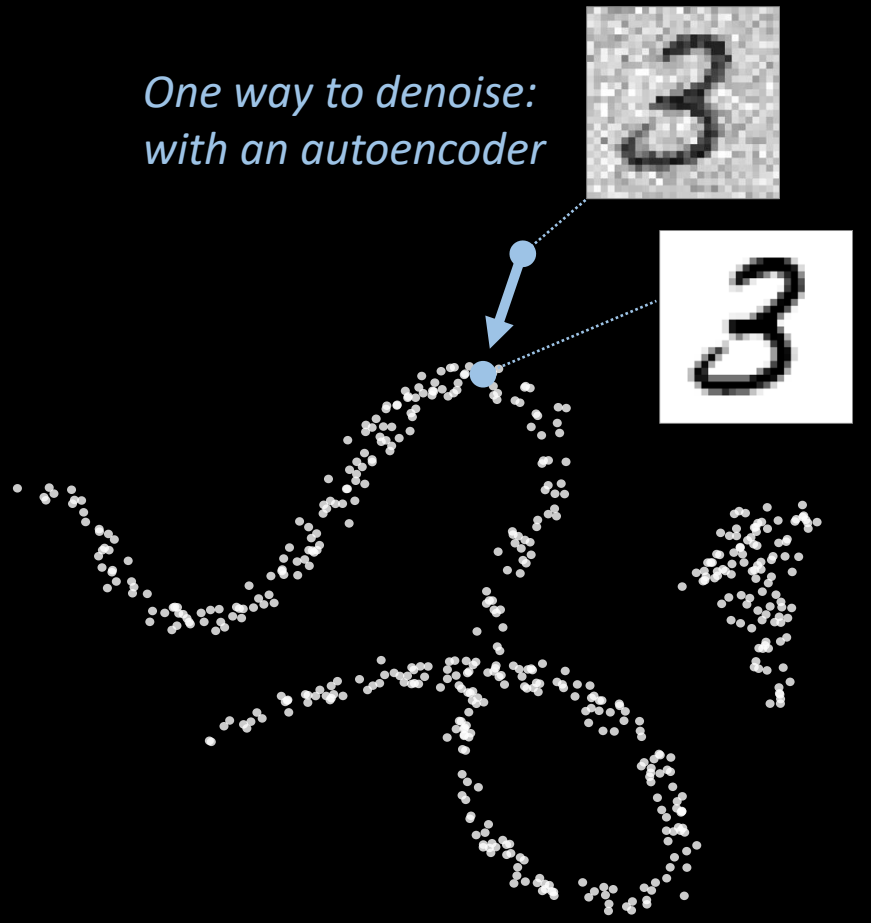


We can see token-by-token generation as state evolution
We can generate the tokens in any order
The evolution reads both as generation and as corruption
Another example: images

Generation and corruption



One way to denoise:
with an autoencoder



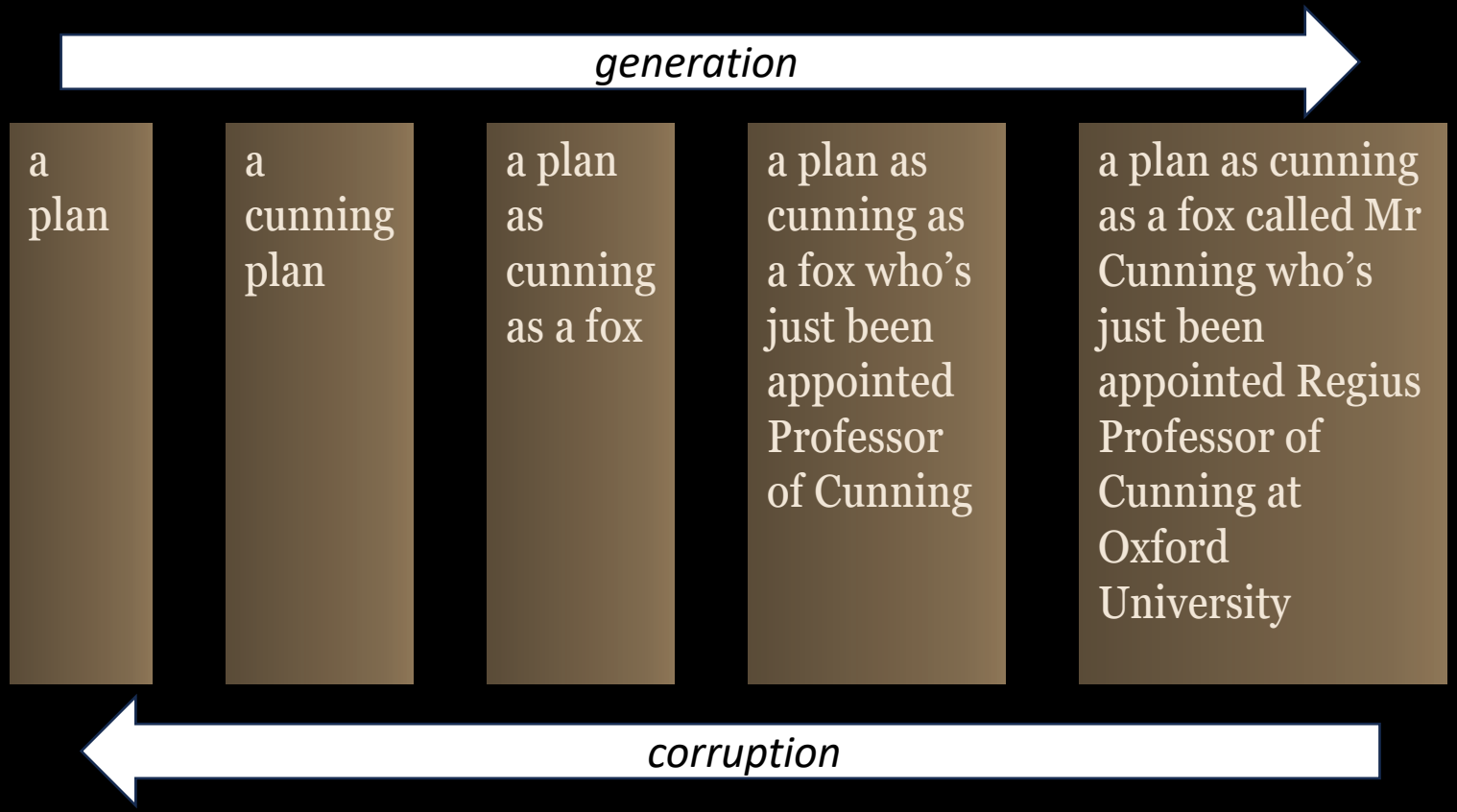
We can see token-by-token generation as state evolution *corrupt = mask a token // generate = predict token*

We can generate the tokens in any order

The evolution reads both as generation and as corruption

Another example: images *corrupt = add noise // generate = denoise*

Generation and corruption



We can see token-by-token generation as state evolution *corrupt = mask a token // generate = predict token*

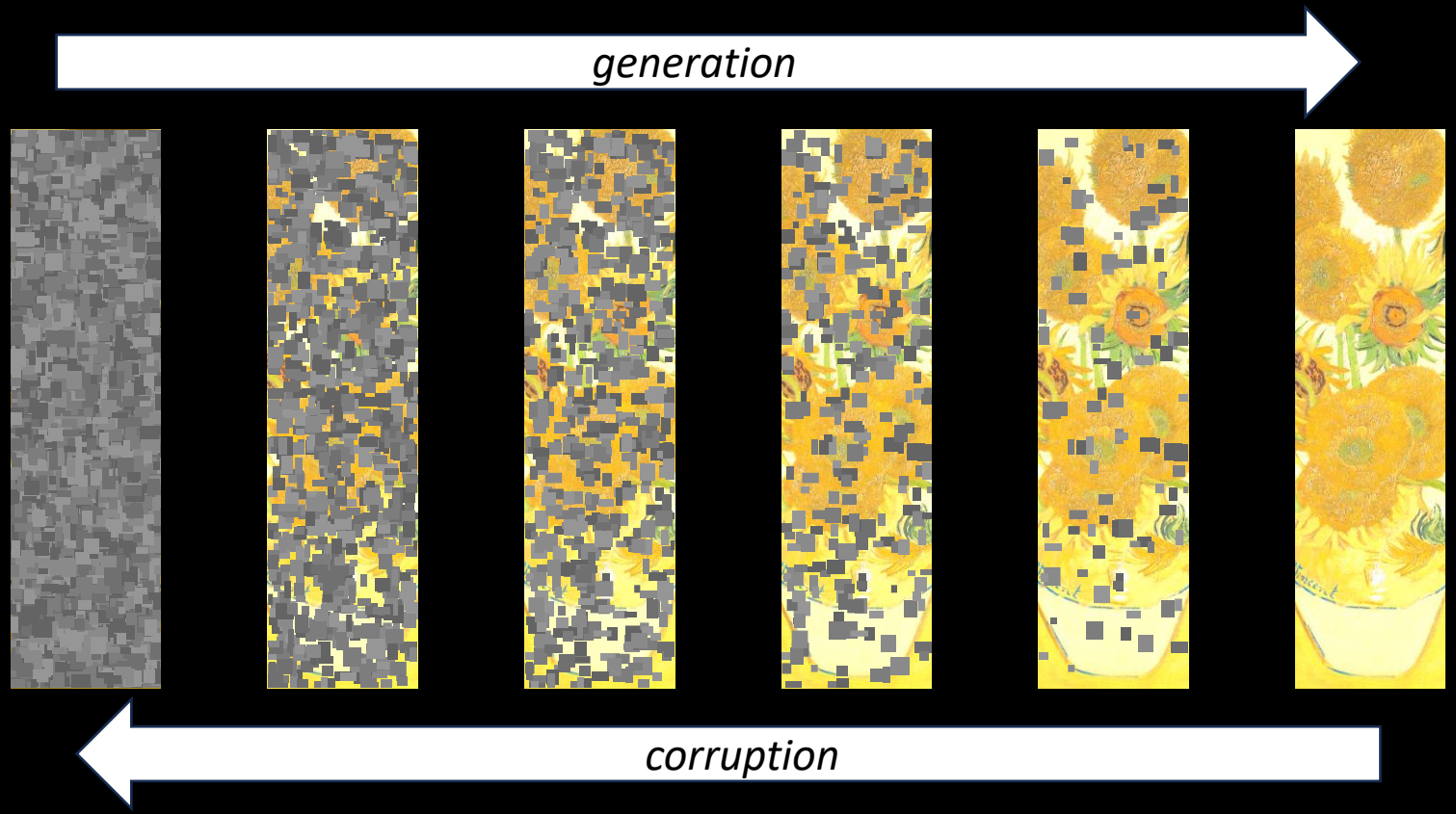
We can generate the tokens in any order

The evolution reads both as generation and as corruption

Another example: images *corrupt = add noise // generate = denoise*

Another example: thought *corrupt = semantic simplification // generate = ???*

Generation and corruption



Cunning idea. Make training data with whatever corruption process you like;
then learn to undo the corruption — that's what generation is.

The task. Choose a corruption process; train a generation process to undo it. We want

- matched generation / corruption: the generator has to learn to undo the corruption steps
- an easy-to-run generator, so we can replay it at inference time – let's use Markovian generation

To learn a good model for X, learn to undo the corruption process

Notation. Let $X_{0:1} = (X_0, X_\Delta, X_{2\Delta}, \ldots, X_{m\Delta} = X_1)$ be a process consisting of $m + 1$ snapshots, where $\Delta = 1/m$.
Call X_0 the "noise", and $X := X_1$ the "data".



noise
X_0



data
X_1

We are interested in two distributions for this process:

Joint distribution q: $(X_0, \quad X_\Delta, \quad \cdots, \quad X_{1-\Delta}) \longleftarrow X_1 =: X$

the corruption process q, which we design: starts with data $X_1 \sim q_X$, then generates $X_{<1}$ however we specify

Joint distribution p^θ: $X_0 \longrightarrow X_\Delta \longrightarrow \cdots \longrightarrow X_{1-\Delta} \longrightarrow X_1 =: X$

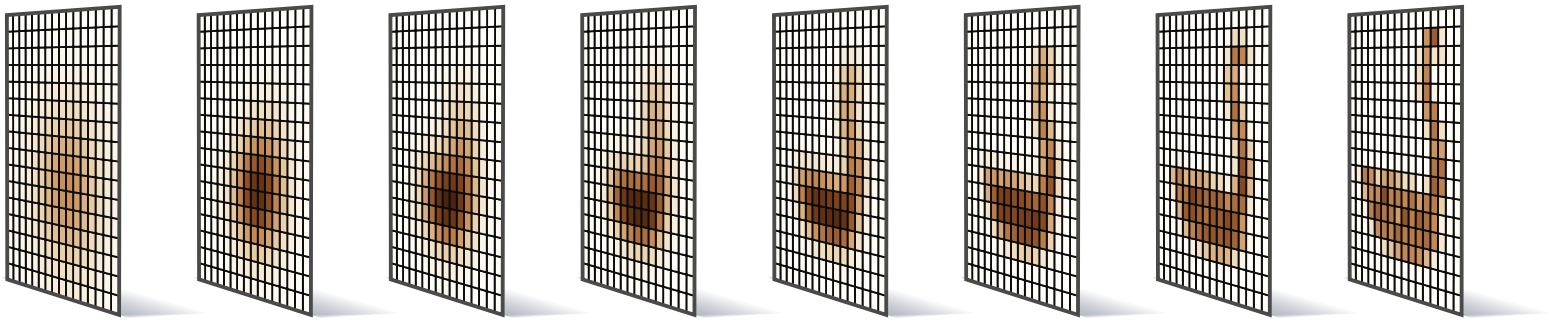
the generation process p^θ, which we train: starts with noise and generates data using a Markov chain

The task. Fit a generation process p^θ, using samples of the corruption process.

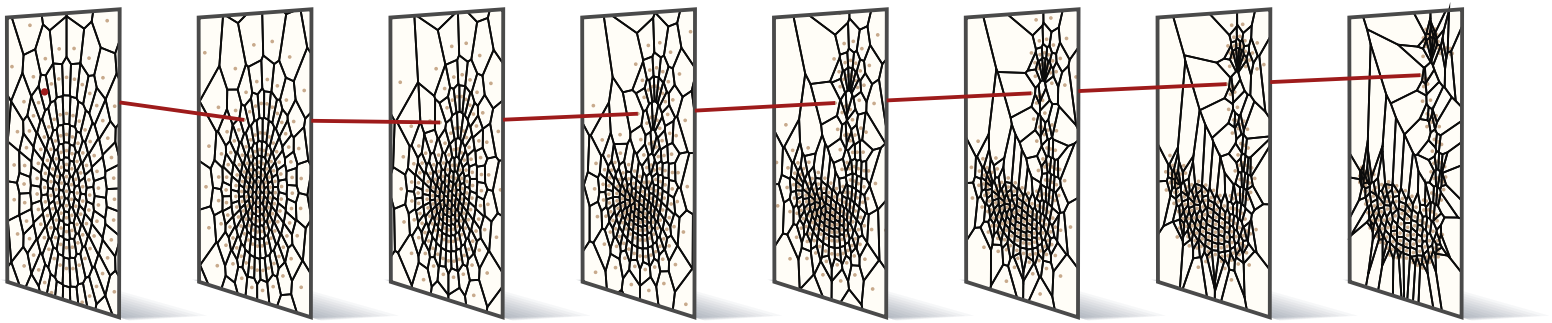
- We design the corruption, including q_{X_0}, so we'll choose $p_{X_0} = q_{X_0}$ as the distribution of the initial noise state
- If p^θ learns to match the full process, we'll get $p^\theta_X \approx q_X$.

Both directions can be described as fluid flows

Look at the marginal distributions $q_x(t) := q_{X_t}(x)$, $t \in \{0, \Delta, 2\Delta, \dots\}$.
All that matters is what these marginals are, and that p^θ have the same marginals.



Cunning idea: fluid mechanics has better language for this. Think of a particle density, for particles following a time-varying velocity field $v_t(x)$:



Pick the initial density of particles to be $p(0) \equiv q(0)$

The velocity field says how particles move:
$x_{t+\Delta} = x_t + \Delta\, v_t(x_t) + O(\Delta^2)$

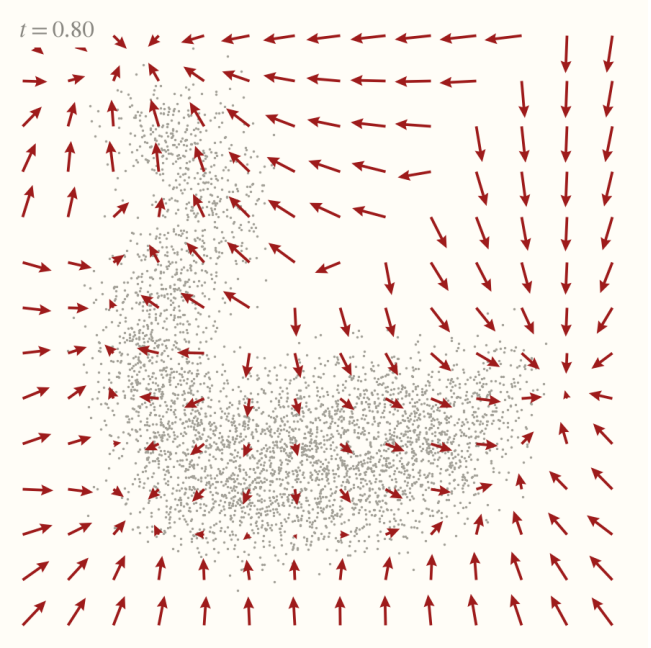
A particle's location at time t is thus
$x_t = x_0 + \int_0^t v_s(x_s)\, ds =: \mathsf{F}_{x_0 @ 0}^{t}\, v_s(\cdot)\, ds$

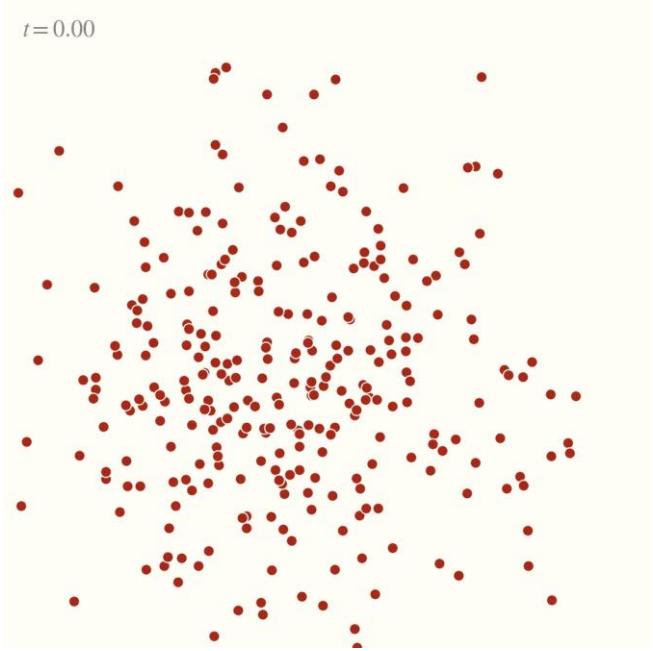
The flow induces a density of particles at time t:
$p^\theta(t)$ is the distribution of $X_t := \mathsf{F}_{X_0 @ 0}^{t}\, v_s\, ds$

If the velocity field pushes particles close together at some location x, the density $p_x^\theta(t)$ will be large

What we need to learn: a velocity field

Task. Learn the velocity field $v_s(\cdot)$ at each timepoint s.
Then, to generate X, we just start with noise X_0 and replay the velocity field.





Lemma 1. Let $v_t(x)$ be continuous in (t, x) and uniformly one-sided Lipschitz in x, and let X_0 have a given distribution. Then

- from every x_0 there is exactly one path that satisfies $\dot{x}_t = v_t(x_t)$ on $[0,1]$, thus $\phi_t(x_0) \coloneqq \int_{x_0 @ 0}^{t} v_s(\cdot)\, ds$ is well defined;
- the distribution of $X_t \coloneqq \phi_t(X_0)$, for $t > 0$, is the unique solution to the continuity equation

$$\frac{d}{dt}\,\mathbb{E}\,h(X_t) = \mathbb{E}[\nabla h(X_t) \cdot v_t(X_t)]$$

for every smooth h

Lipschitz means 'not too steep'.
One-sided Lipschitz in x: $\exists L\ \forall t, x, y.$
$(v_t(x) - v_t(y)) \cdot (x - y) \leq L\,\|x - y\|^2$

The idea of learning a velocity field, and the formula on a later slide for $\log p_X^\theta(x)$*, are due to Chen, Rubanova, Bettencourt and Duvenaud 2018*

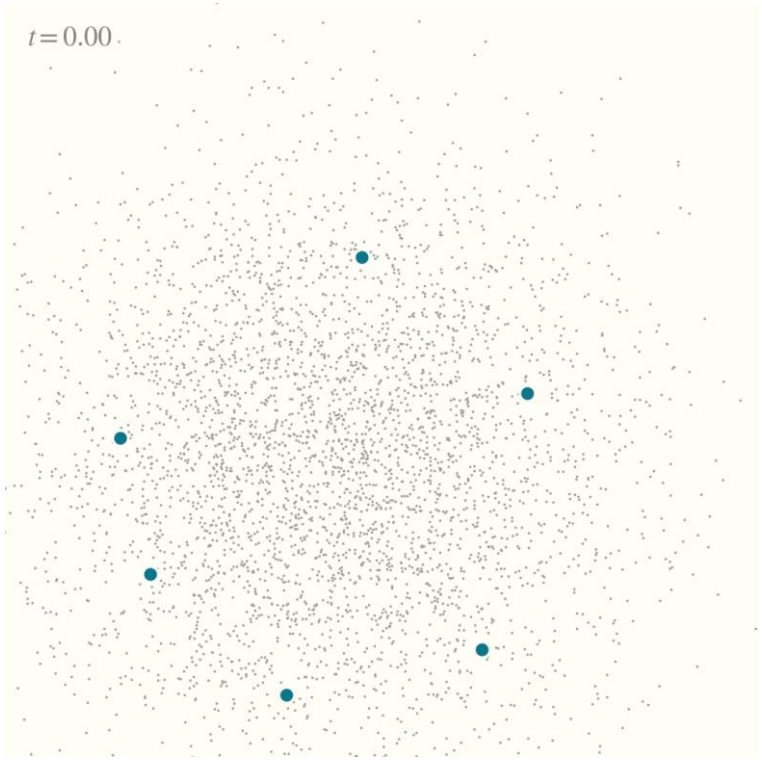
Let's figure out the simplest velocity field that works

Design the corruption.

1. Pick a noise distribution $\Pr_Z$.
2. Let $X \sim q_X$ and $Z \sim \Pr_Z$ be independent, and define $X_t := tX + (1-t)Z$.

(Thus $\Pr_{X_0} = \Pr_Z$ by construction, so it's easy to start the generation process.)

Any path from data to noise would do, but why complicate things?



What's its velocity field? For any smooth h,

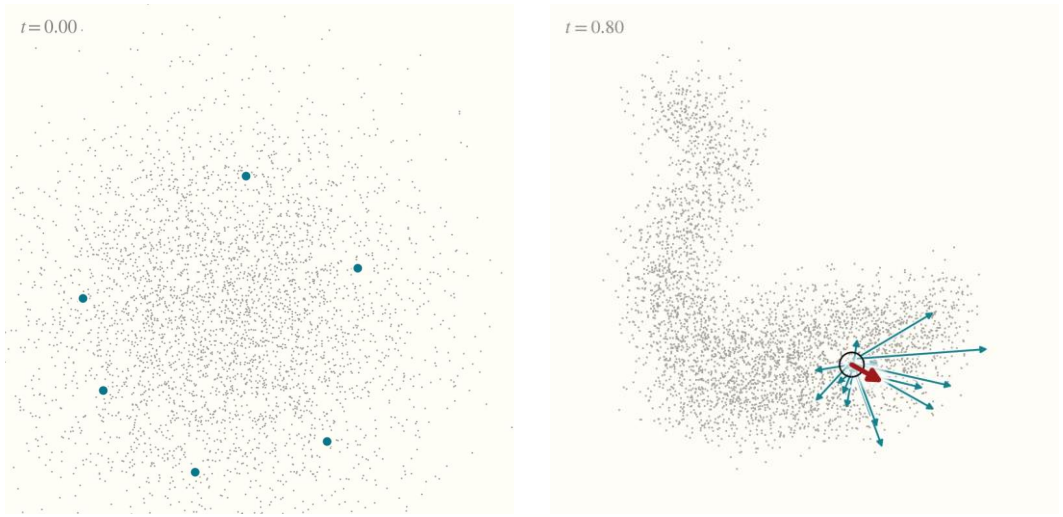
$$\begin{aligned}\frac{d}{dt}\,\mathbb{E}\,h(X_t) &= \mathbb{E}_{X,Z}\,\frac{d}{dt}h(X_t)\\ &= \mathbb{E}_{X,Z}\left[\nabla h(X_t)\cdot \dot{X}_t\right]\\ &= \mathbb{E}_{X_t}\mathbb{E}_{X,Z|X_t}\left[\nabla h(X_t)\cdot \dot{X}_t\right]\\ &= \mathbb{E}_{X_t}\left[\nabla h(X_t)\cdot \mathbb{E}_{X,Z|X_t}\dot{X}_t\right]\end{aligned}$$

In other words, the velocity field $v_t(x) := \mathbb{E}_{X,Z}(\dot{X}_t \mid X_t = x)$ solves the continuity equation for our corruption process q_{X_t}, $t > 0$. Therefore (by Lemma 1) it reproduces q_{X_t}.

We can learn this velocity field by simple regression

Our simple corruption process has velocity field $v_t(x) := \mathbb{E}_{X,Z}(\dot{X}_t \mid X_t = x)$

This is the average velocity of the particles passing through x at time t.



Fit it by regression. We can train a neural network $\tilde{v}^\theta(t,x)$ to approximate $v_t(x)$, by solving

$$\min_\theta \mathbb{E}_{X\sim q_X,\, Z\sim \Pr_Z,\, T\sim U[0,1]} \left\| \tilde{v}^\theta(T, X_T) - \dot{X}_t \right\|^2$$

A perfect fit would, for any t, x, solve $\tilde{v}^\theta(t,x) = \mathbb{E}(\dot{X}_t \mid X_t = x)$ as required.
For a finite parametric network $\tilde{v}^\theta$ we'll get some approximation.
Computation: sample X, Z, T, compute the loss; take a gradient step.

Lipman, Chen, Ben-Hamu, Nickel and Le 2023: the idea of fitting a flow by a regression on per-datapoint flows

The flow is invertible, and this allows us to evaluate log likelihood.

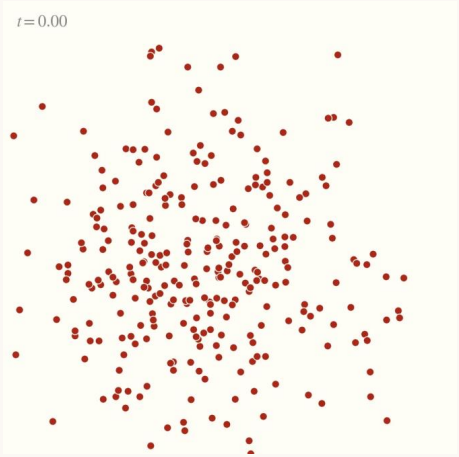
Our generation process: pick $X_0 \sim$ noise and then replay the flow,

$$X \coloneqq X_1 = \int_{X_0 @ 0}^{1} \tilde{v}^\theta(s,\cdot)\, ds$$

This map $X_0 \mapsto X_1$ is invertible.

In fact, we recover X_0 by starting at X_1 and following the velocity field *backwards*.

This lets us compute $\log p^\theta_{X_1}(x)$ — which lets us do holdout-set evaluation.



Could we train directly by maximizing log likelihood?

- Yes, in principle. But the computation needs us to compute the path back from x, which is an integral.
- Learning the velocity field is easier: all that needs is samples.

Limitations.

- The map is invertible as long as the velocity field is well-behaved
- A velocity field $\tilde{v}^\theta(s,x)$ computed by a neural network is well behaved
- The true corruption process might not have an invertible flow, for example if it collapses the noise-space onto a lower-dimensional set
- Thus, a neural-network velocity field can't completely reproduce such a corruption process

Lemma 2. Let the velocity field $v_t(x)$ be continuous in (t,x) and uniformly Lipschitz in x, and let X_0 have a density.

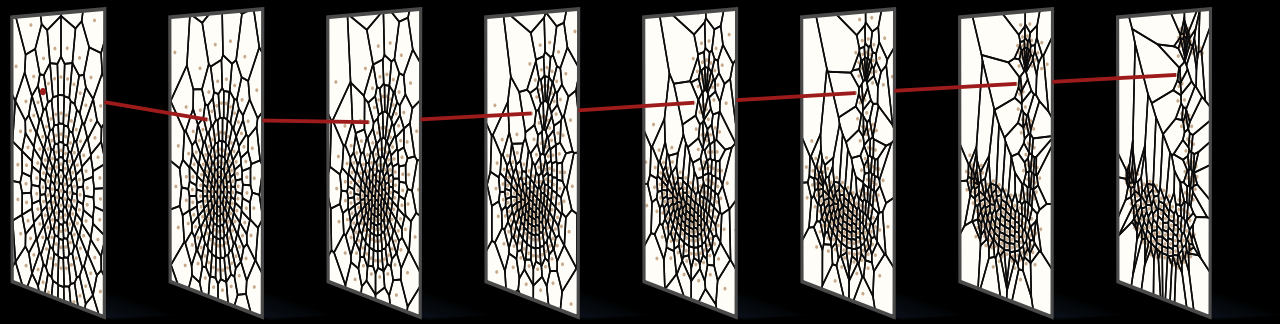
- This meets the conditions of Lemma 1, thus the flow map $\phi_t : x_0 \mapsto x_t$ is well-defined.
- Furthermore ϕ_t is a bijection, with inverse the same field followed backwards,
  $\phi_t^{-1}(x) = \int_{x @ t}^{0} v_s(\cdot)\, ds$
- For all $t > 0$, $X_t \coloneqq \phi_t(X_0)$ has a density

Lipschitz means 'not too steep'.

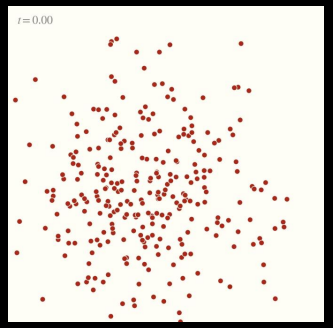
Lipschitz in x: $\exists L\ \forall t, x, y.$

$$|v_t(x) - v_t(y)| \le L\,|x - y|$$

The limitations of flow matching



The whole ‘flow’ notion requires that there *be* a velocity field in the first place.
Velocity is an equation for $\dot{x}_t$ — but this only makes sense for continuous x_t,
not for discrete data such as text.



A neural-network velocity field must be Lipschitz, so it can only learn invertible flows.
Velocity-based generation can only learn dimension-preserving maps,
it can’t learn to ‘collapse’ noise onto a lower-dimensional subset.

After the break: diffusion models can get around these limitations.

When life gives you lemons, make labels

unlabelled but structured data

Today:
corrupt the data in steps, and train a generator to undo them

Contrastive learning (session 2):
destroy the structure, and train a critic to spot it