

Introduction to
Artificial Intelligence
with Python

Learning

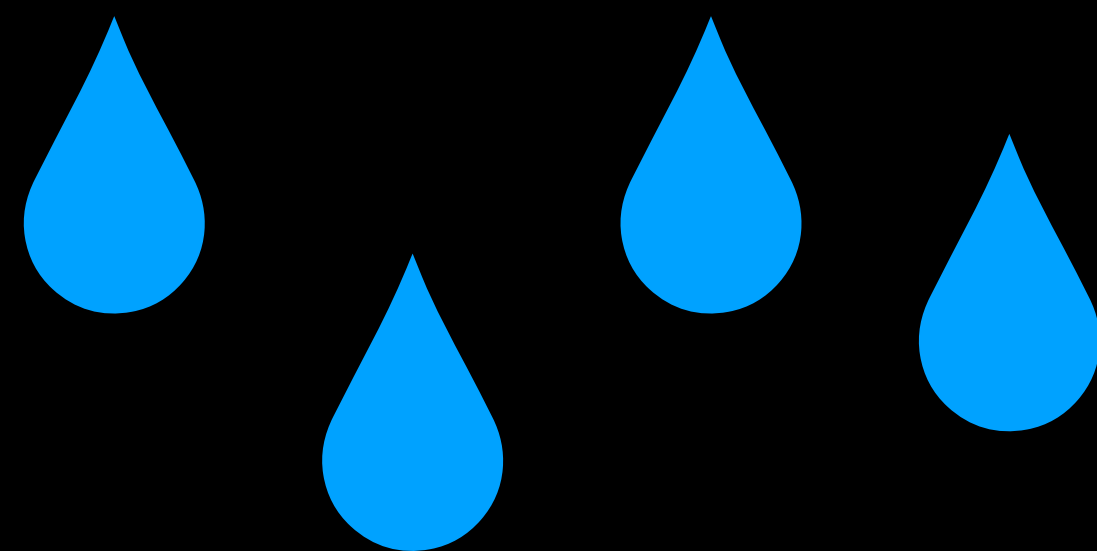
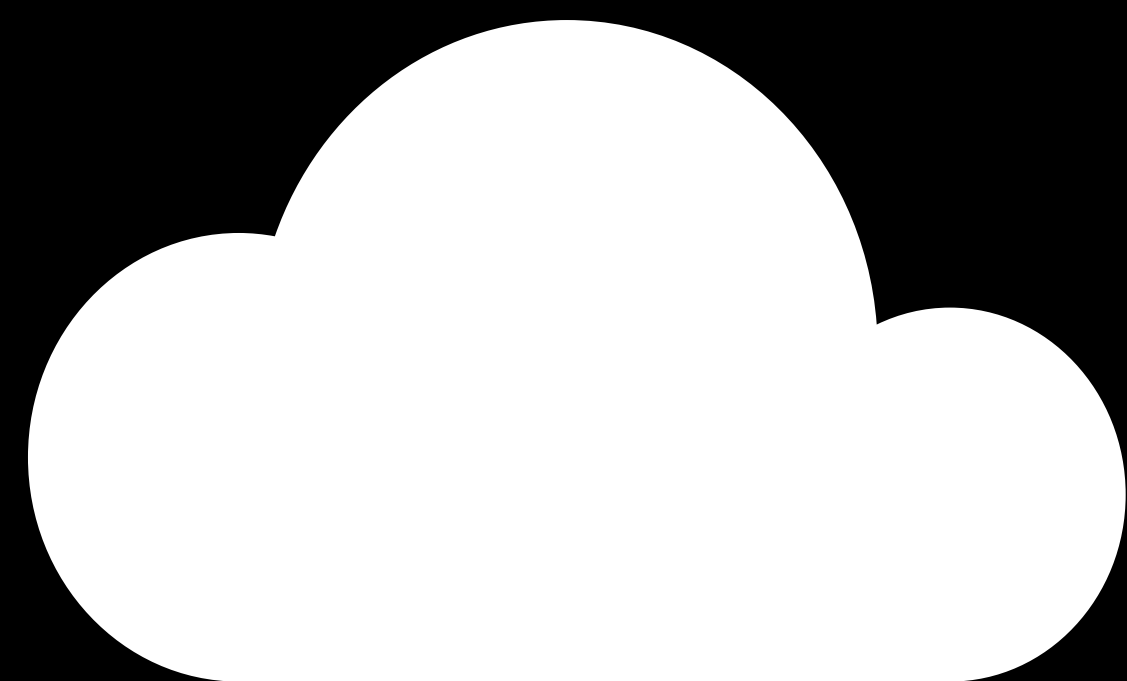
Supervised Learning

supervised learning

given a data set of input-output pairs, learn
a function to map inputs to outputs

classification

supervised learning task of learning a function mapping an input point to a discrete category



Date	Humidity (relative humidity)	Pressure (sea level, mb)	Rain

Date	Humidity (relative humidity)	Pressure (sea level, mb)	Rain
January 1	93%	999.7	Rain
January 2	49%	1015.5	No Rain
January 3	79%	1031.1	No Rain
January 4	65%	984.9	Rain
January 5	90%	975.2	Rain

f(humidity, pressure)

f(93, 999.7) = Rain

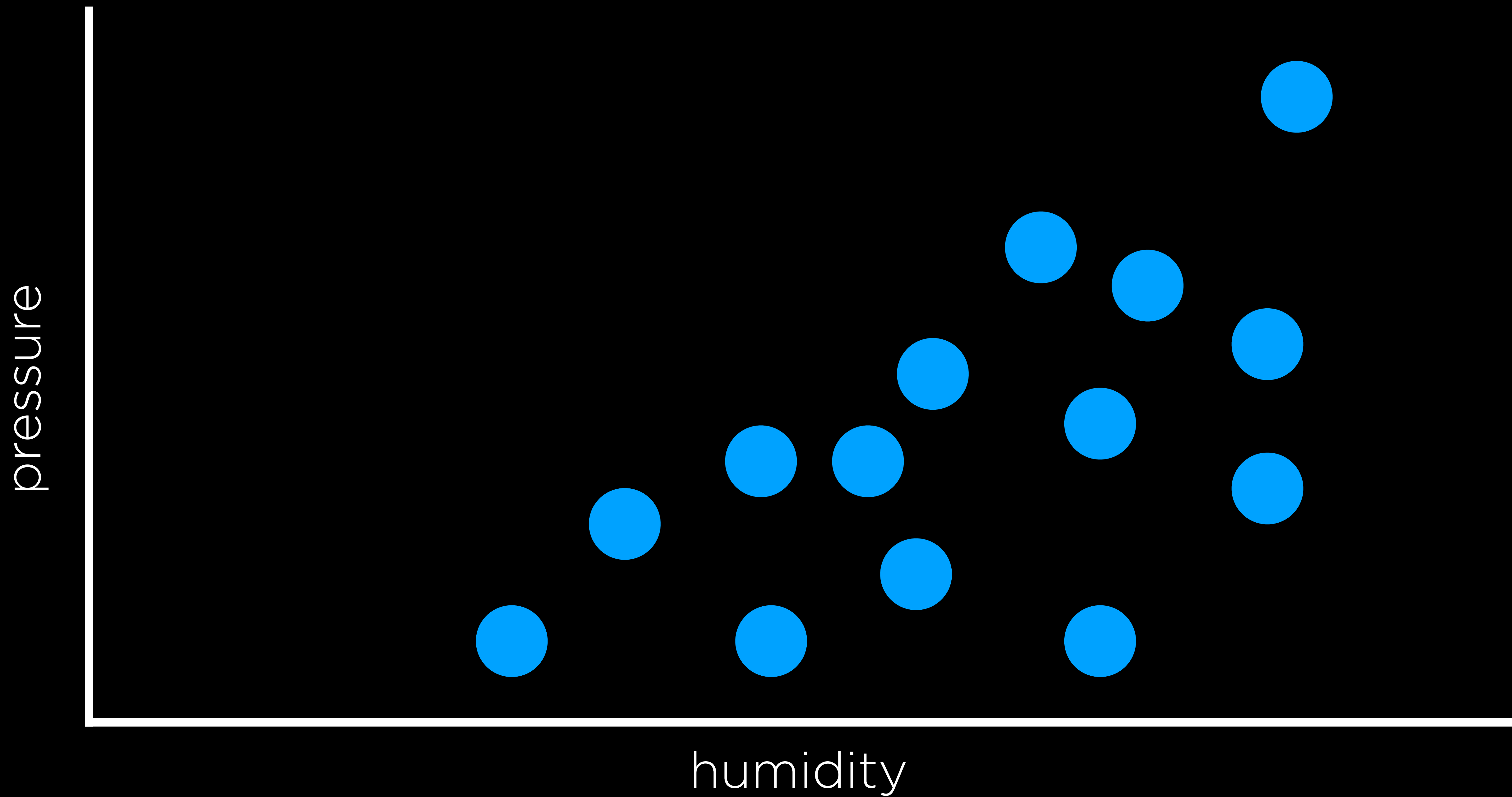
f(49, 1015.5) = No Rain

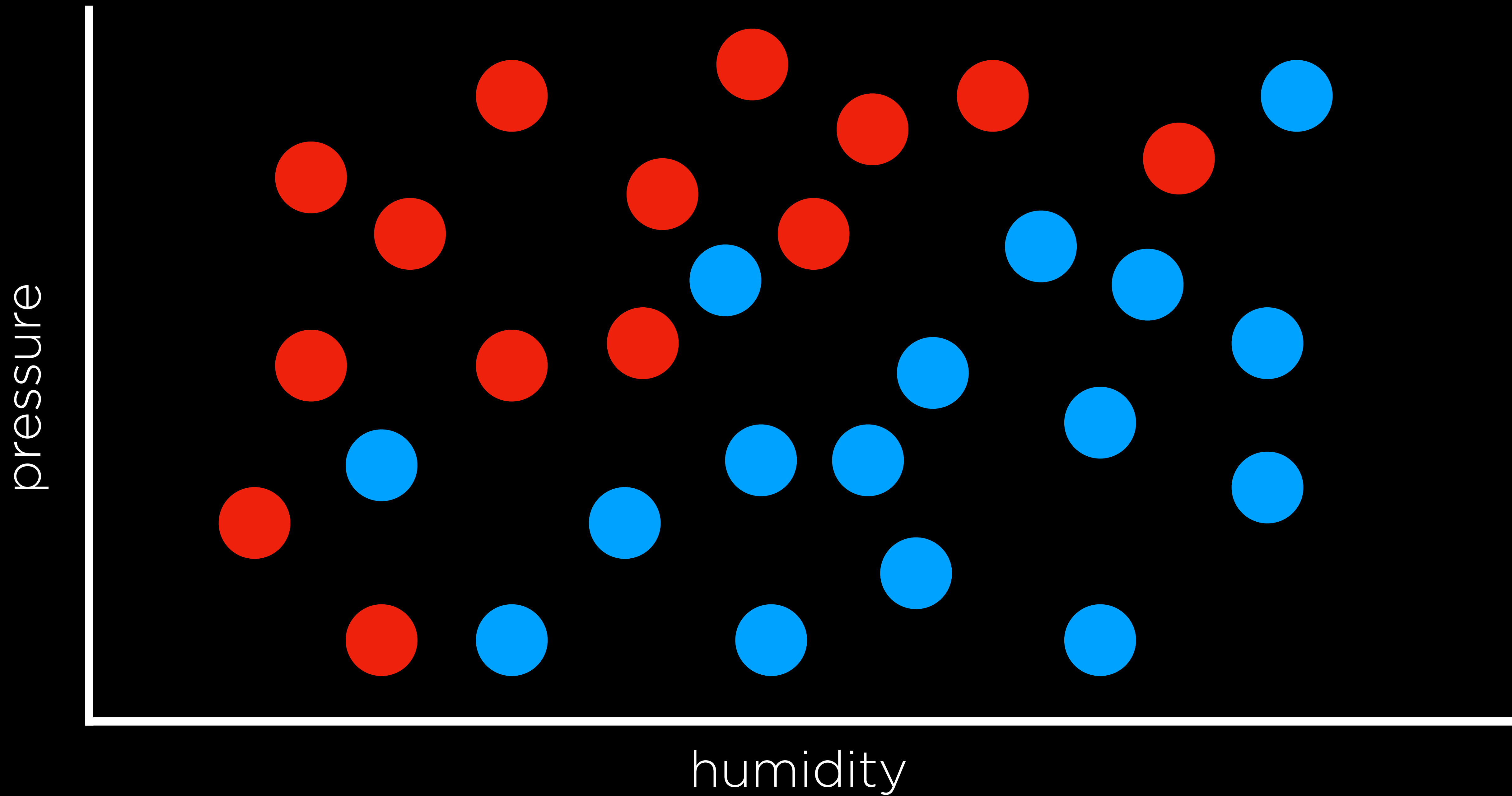
f(79, 1031.1) = No Rain

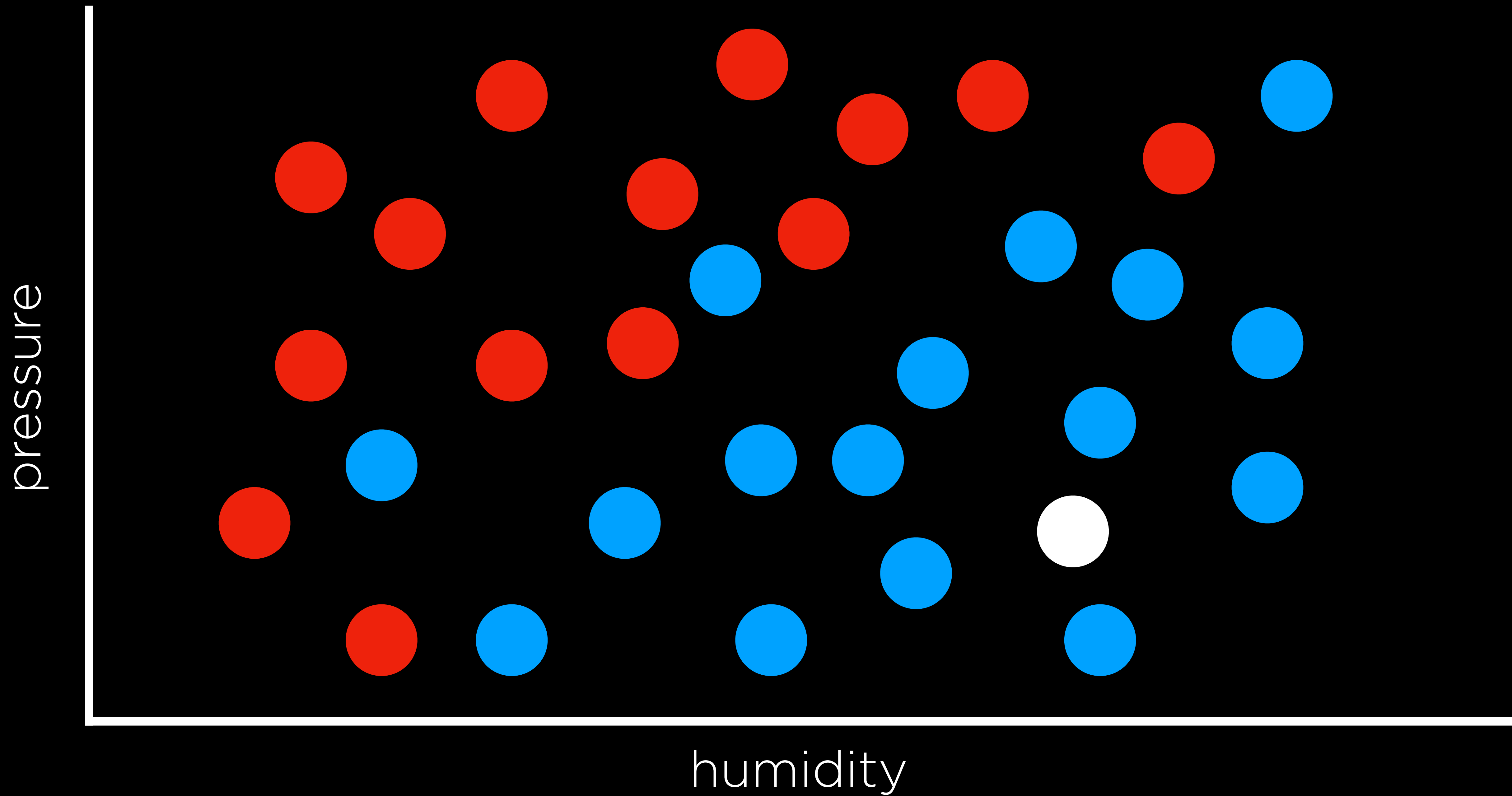
h(humidity, pressure)

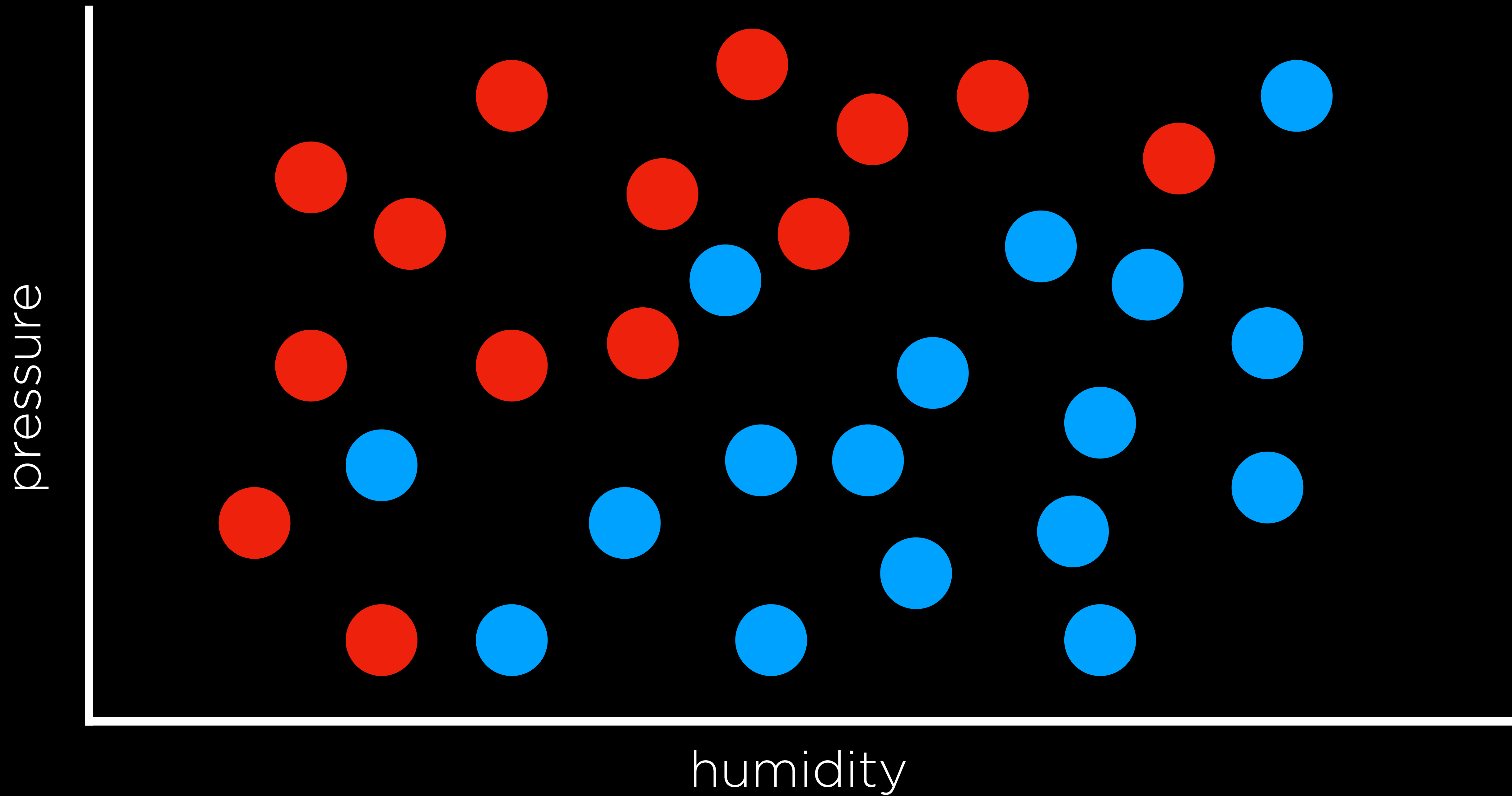
pressure

humidity



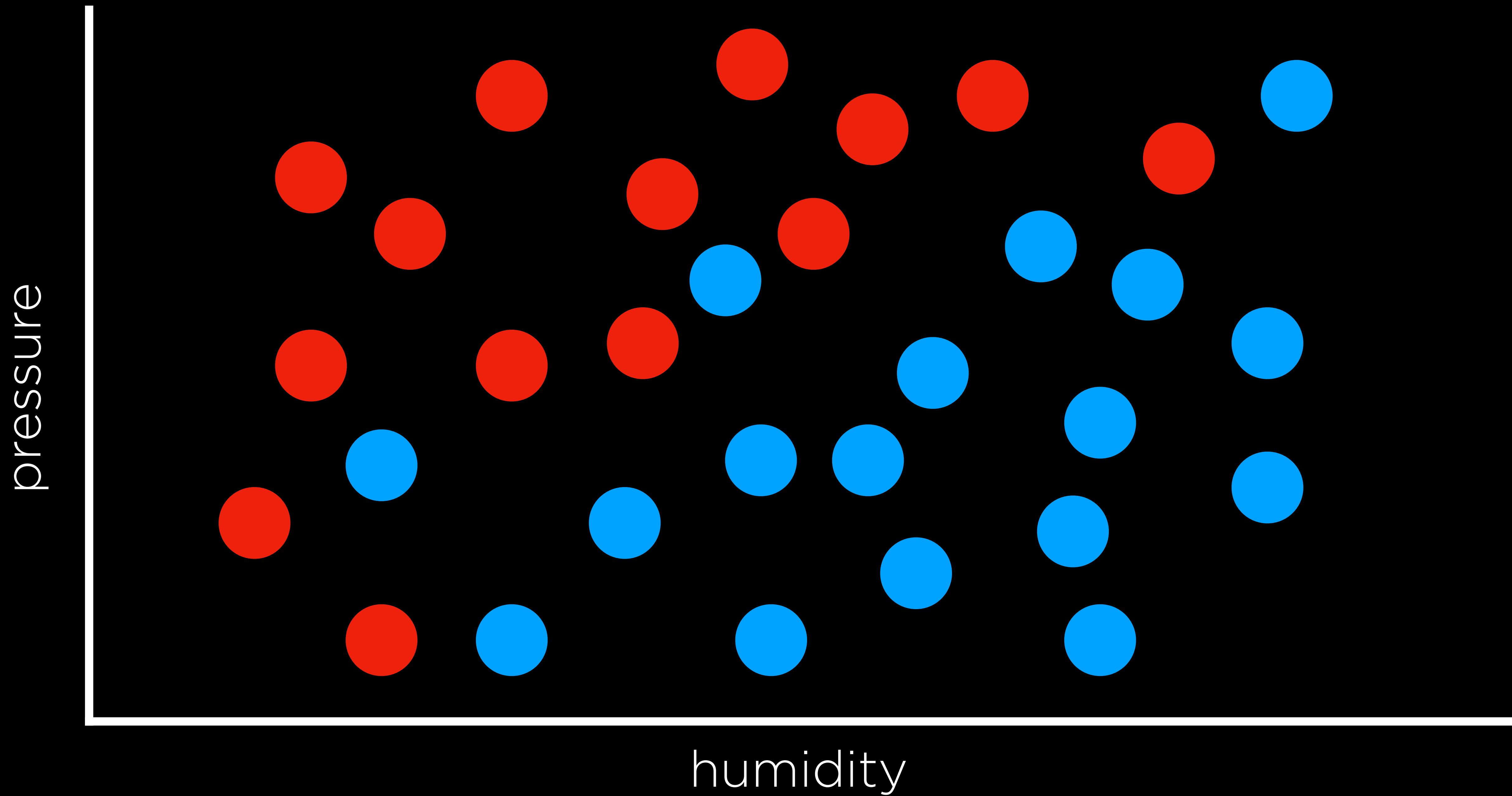


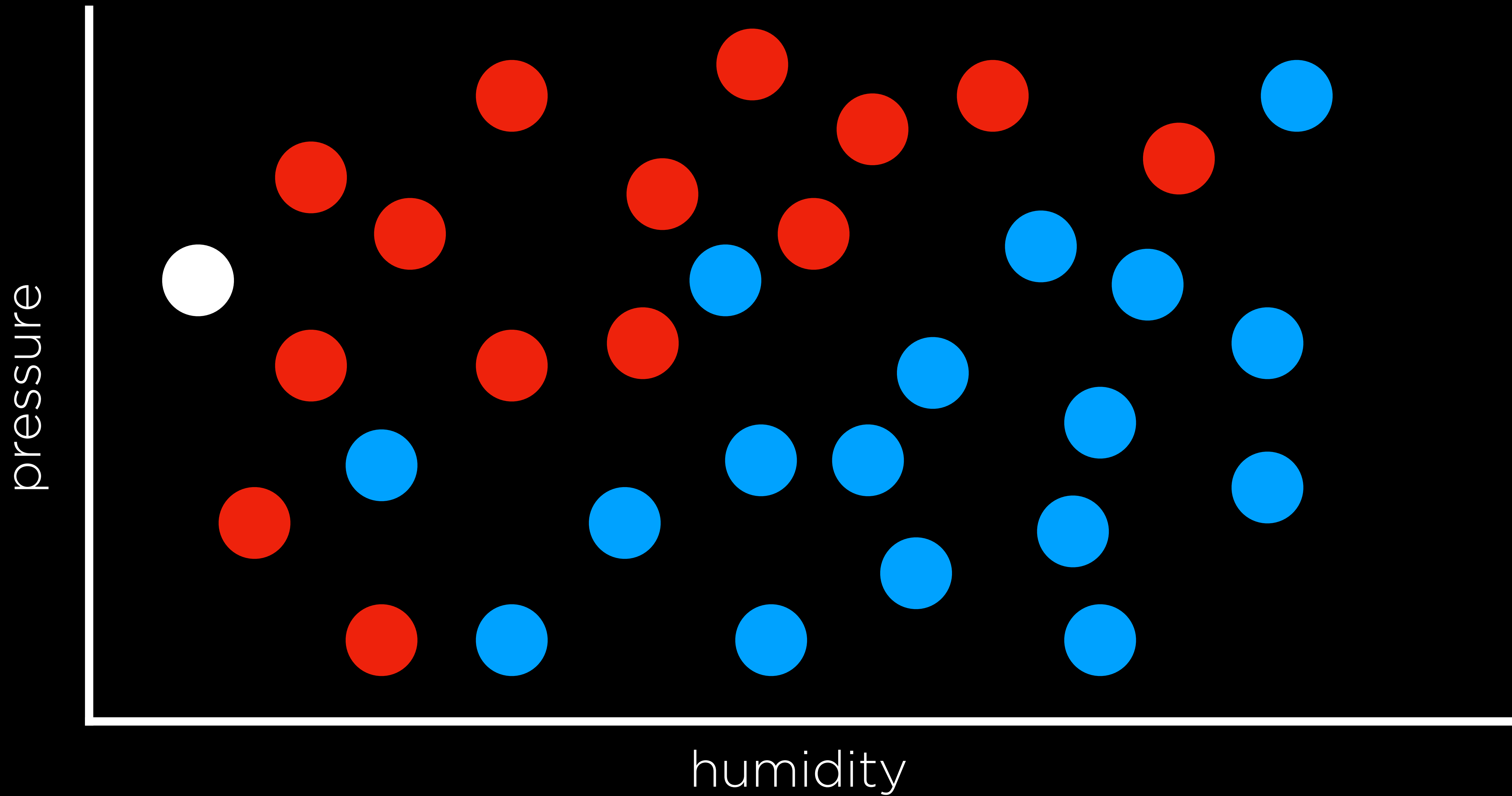


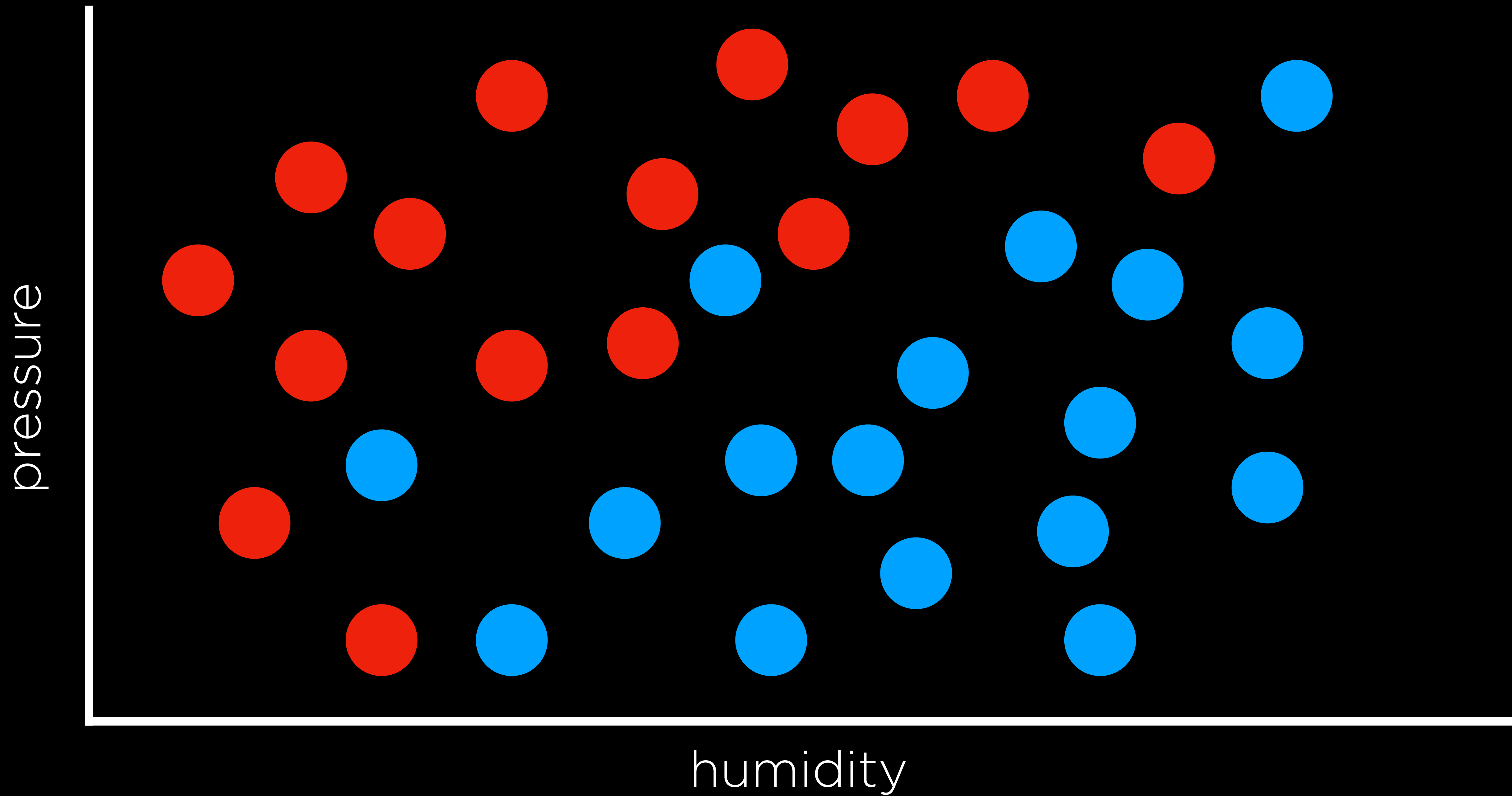


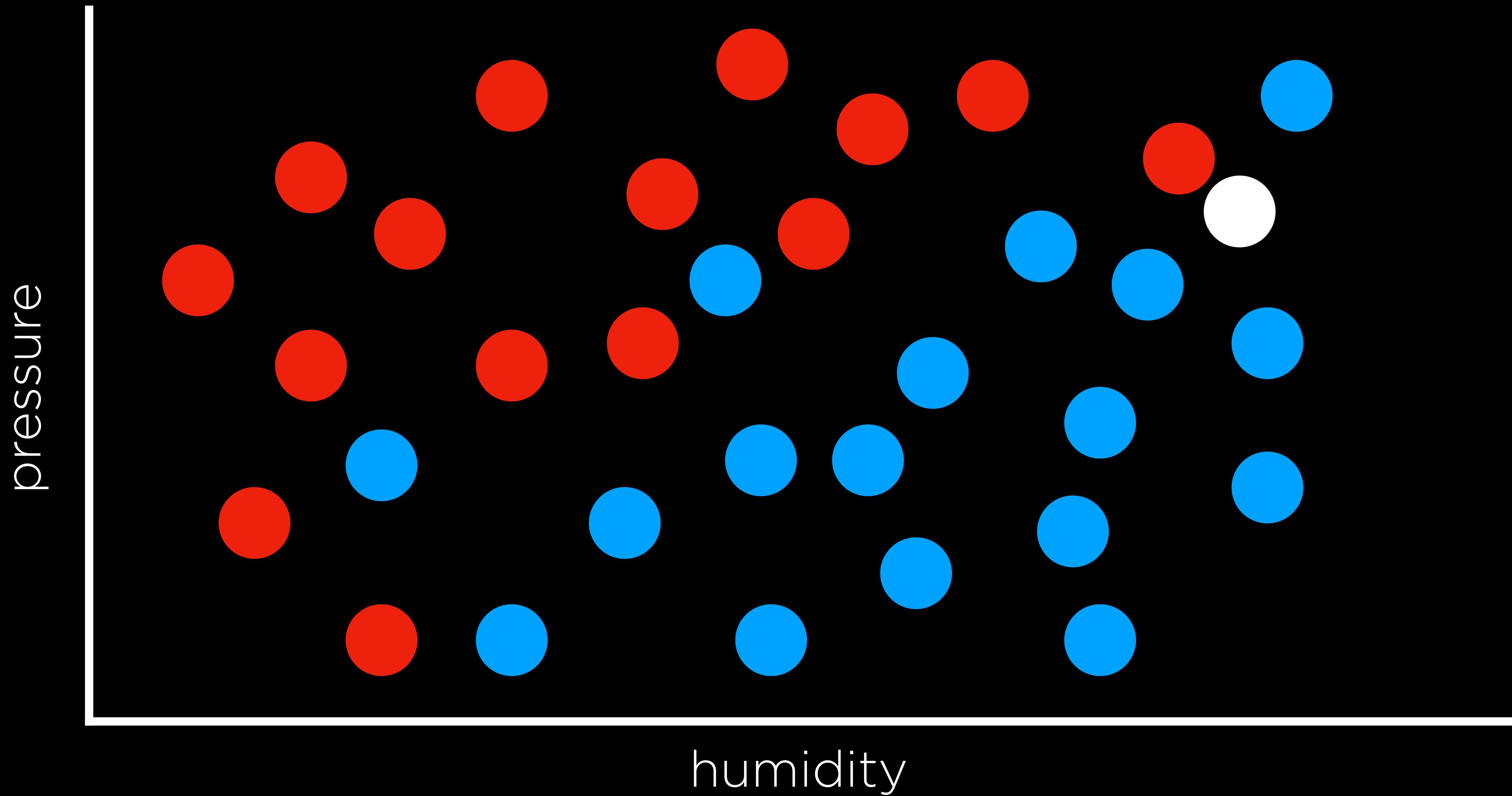
nearest-neighbor classification

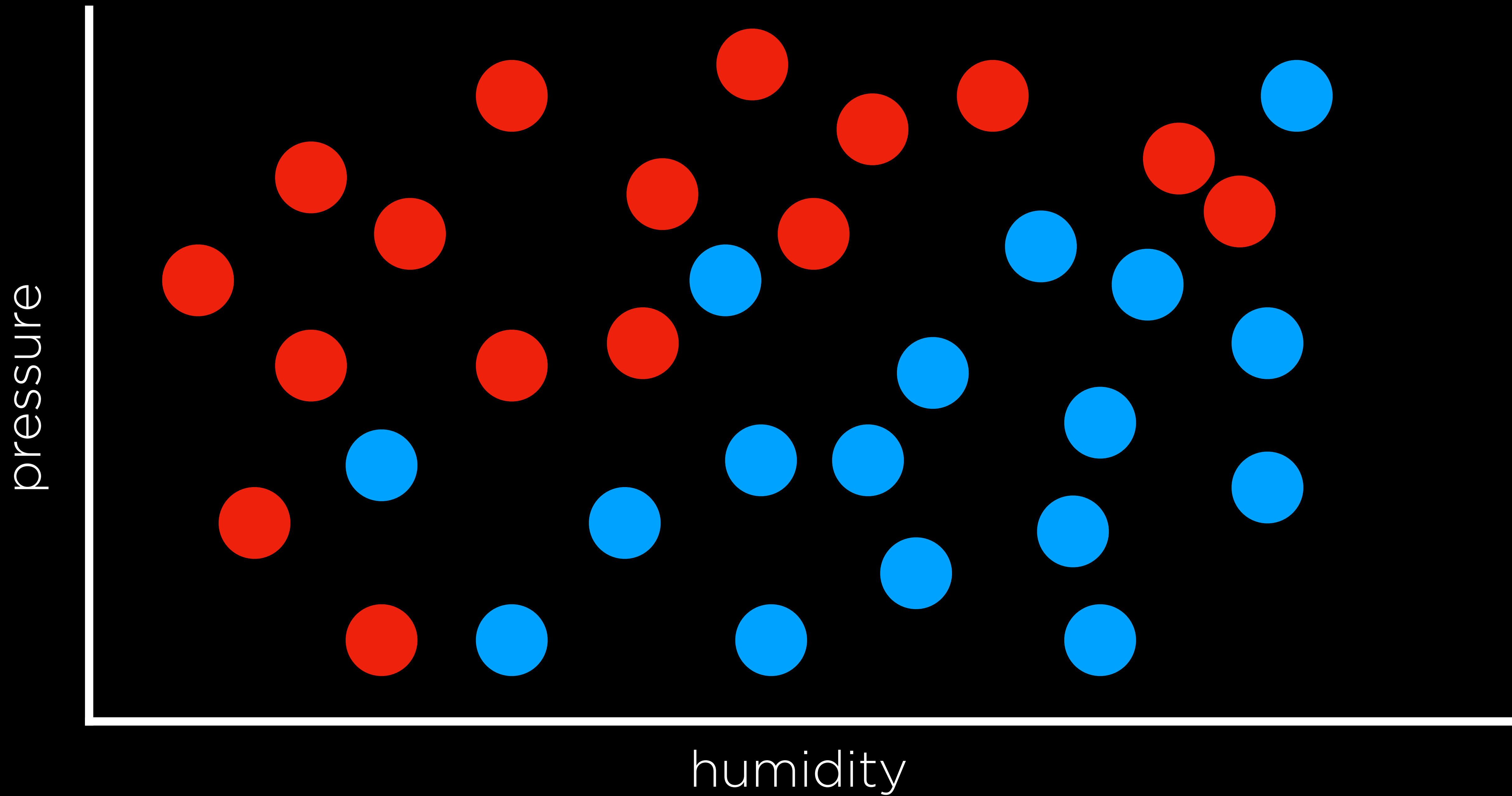
algorithm that, given an input, chooses the class of the nearest data point to that input

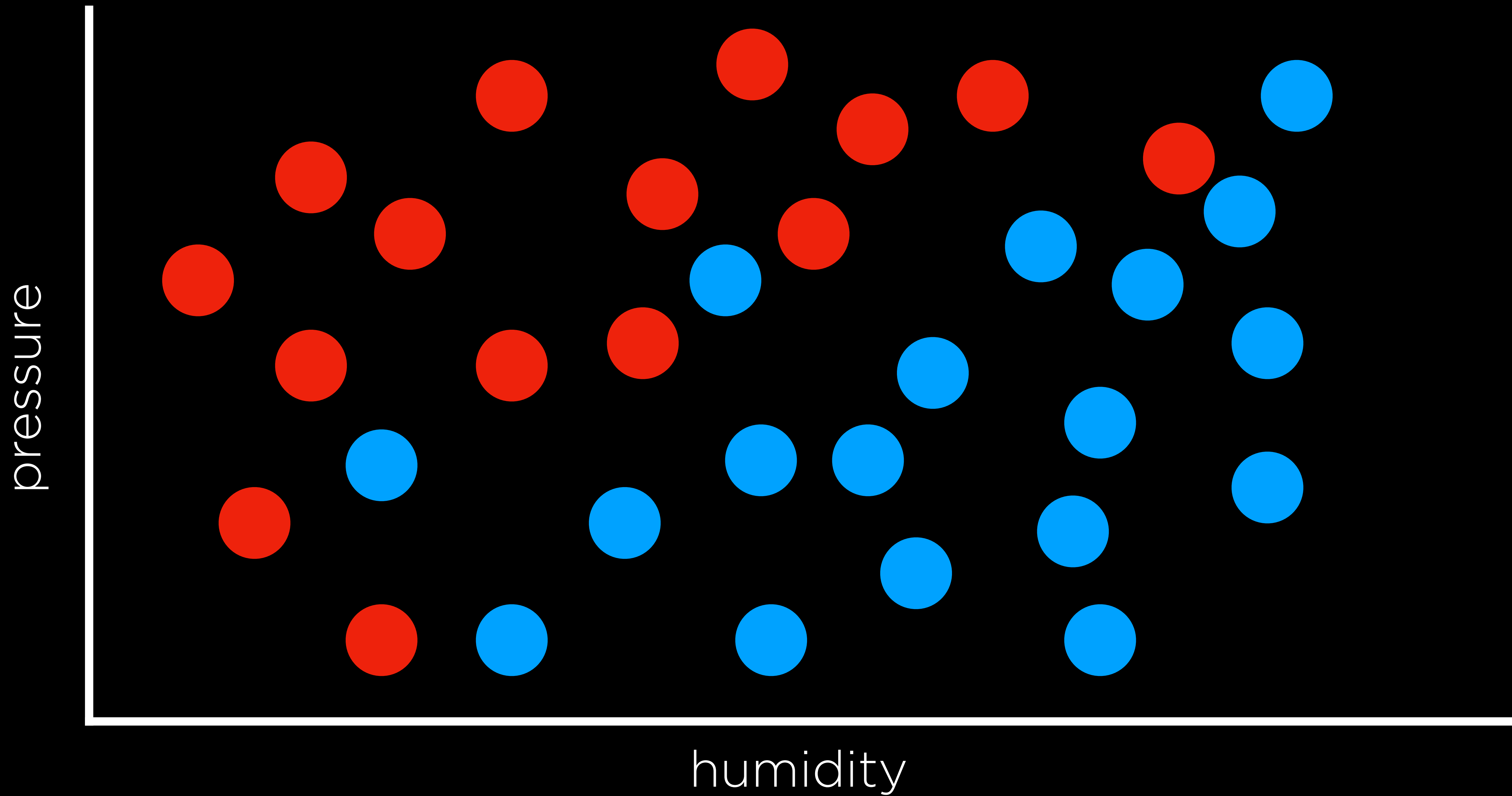






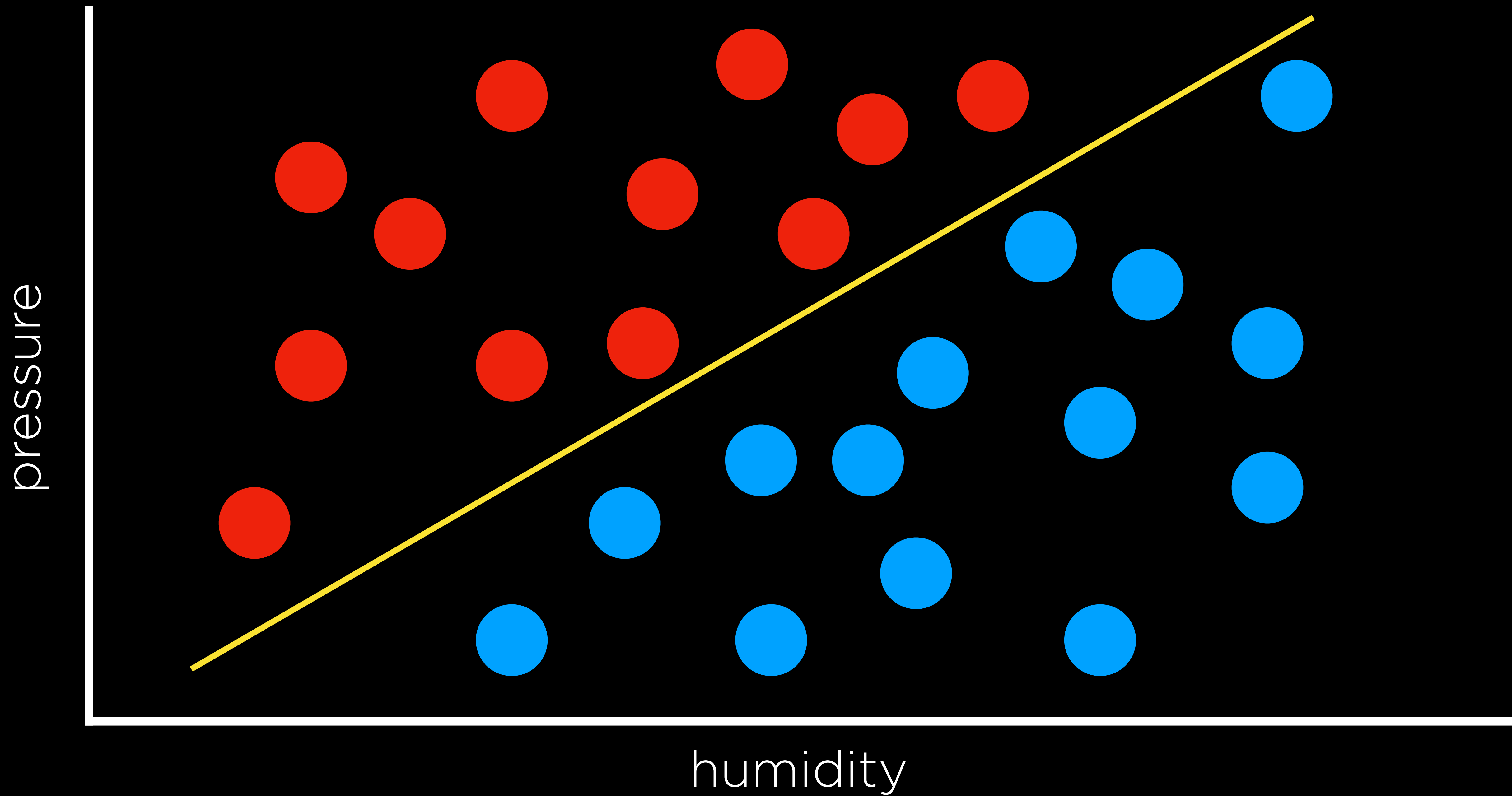


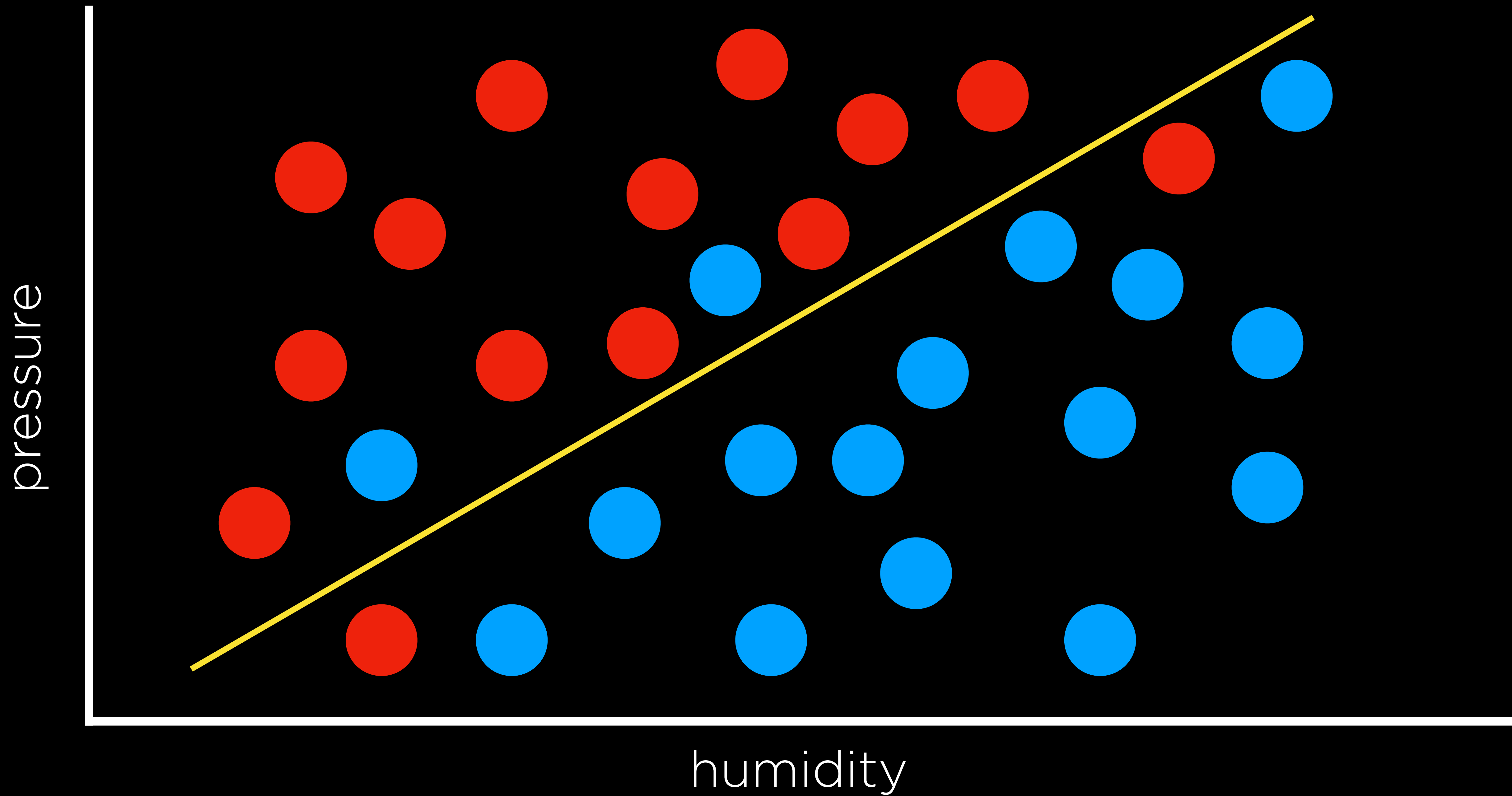




k -nearest-neighbor classification

algorithm that, given an input, chooses the most common class out of the k nearest data points to that input





x_1 = Humidity

x_2 = Pressure

$h(x_1, x_2)$ = Rain if $w_0 + w_1x_1 + w_2x_2 \geq 0$
No Rain otherwise

Weight Vector \mathbf{w} : (w_0, w_1, w_2)

Input Vector \mathbf{x} : $(1, x_1, x_2)$

$$\mathbf{w} \cdot \mathbf{x}: w_0 + w_1x_1 + w_2x_2$$

$$h(x_1, x_2) = \begin{cases} 1 & \text{if } w_0 + w_1x_1 + w_2x_2 \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Weight Vector \mathbf{w} : (w_0, w_1, w_2)

Input Vector \mathbf{x} : $(1, x_1, x_2)$

$$\mathbf{w} \cdot \mathbf{x}: w_0 + w_1x_1 + w_2x_2$$

$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

perceptron learning rule

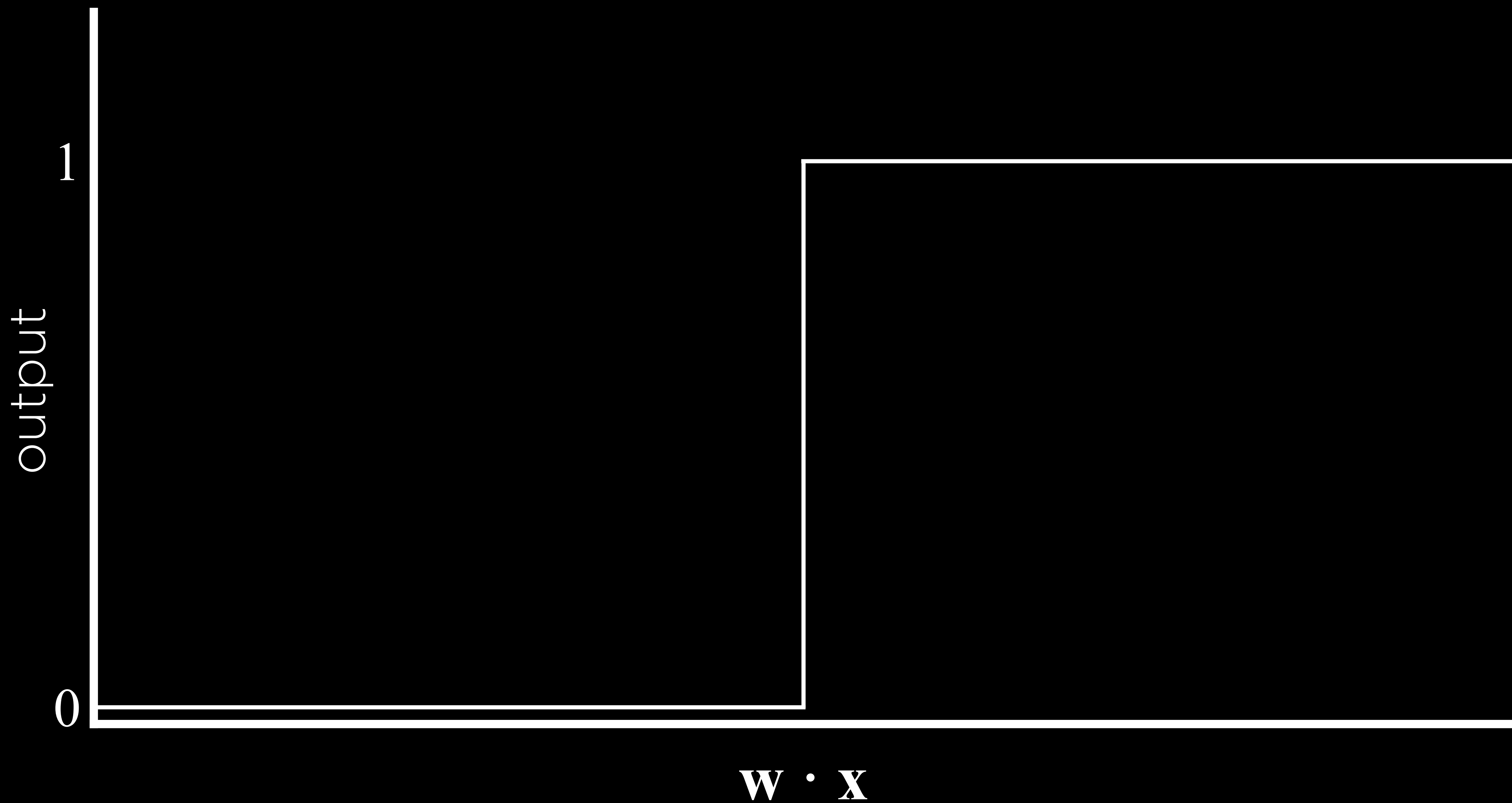
Given data point (\mathbf{x}, y) , update each weight according to:

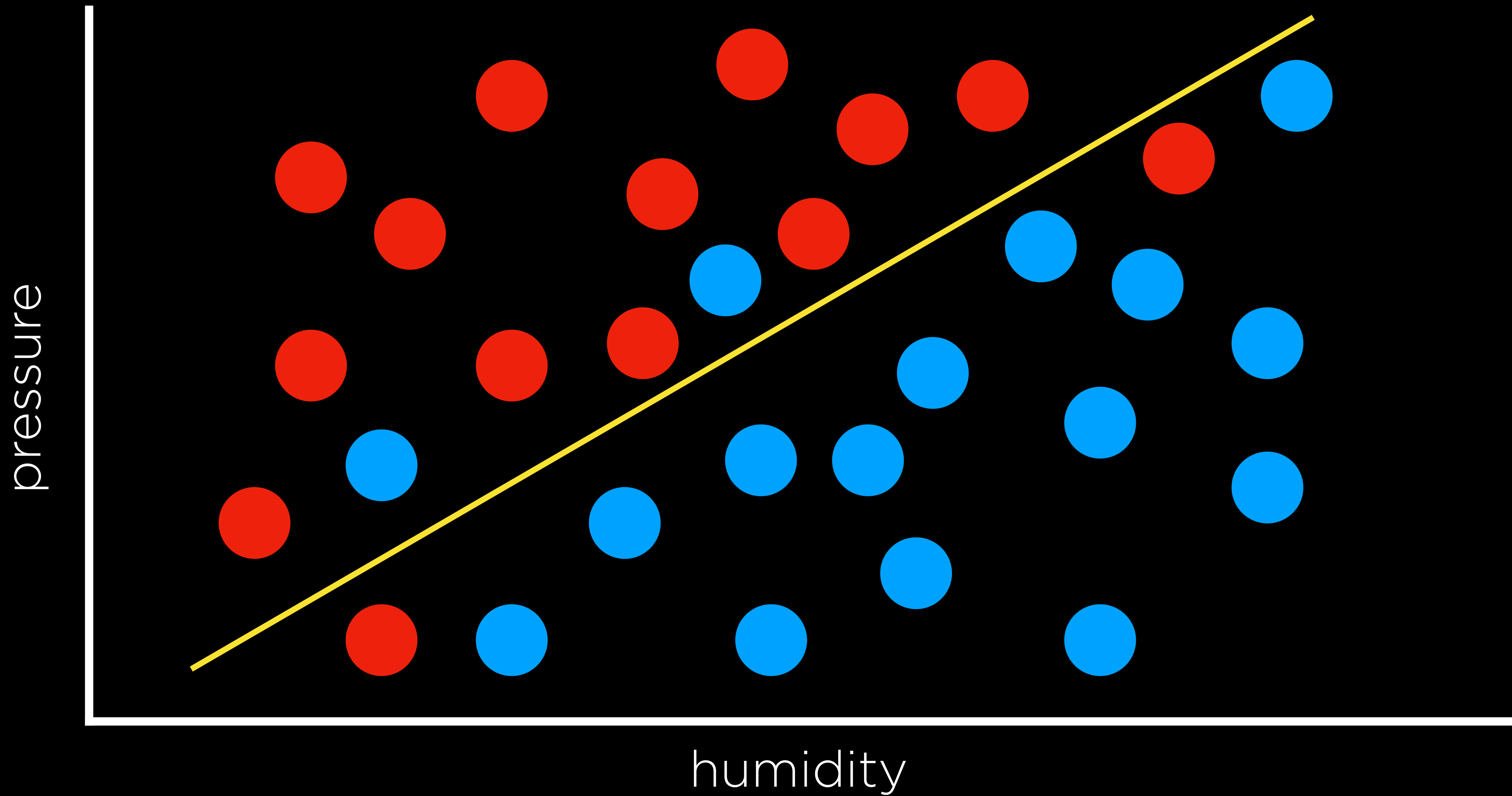
$$w_i = w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x})) \times x_i$$

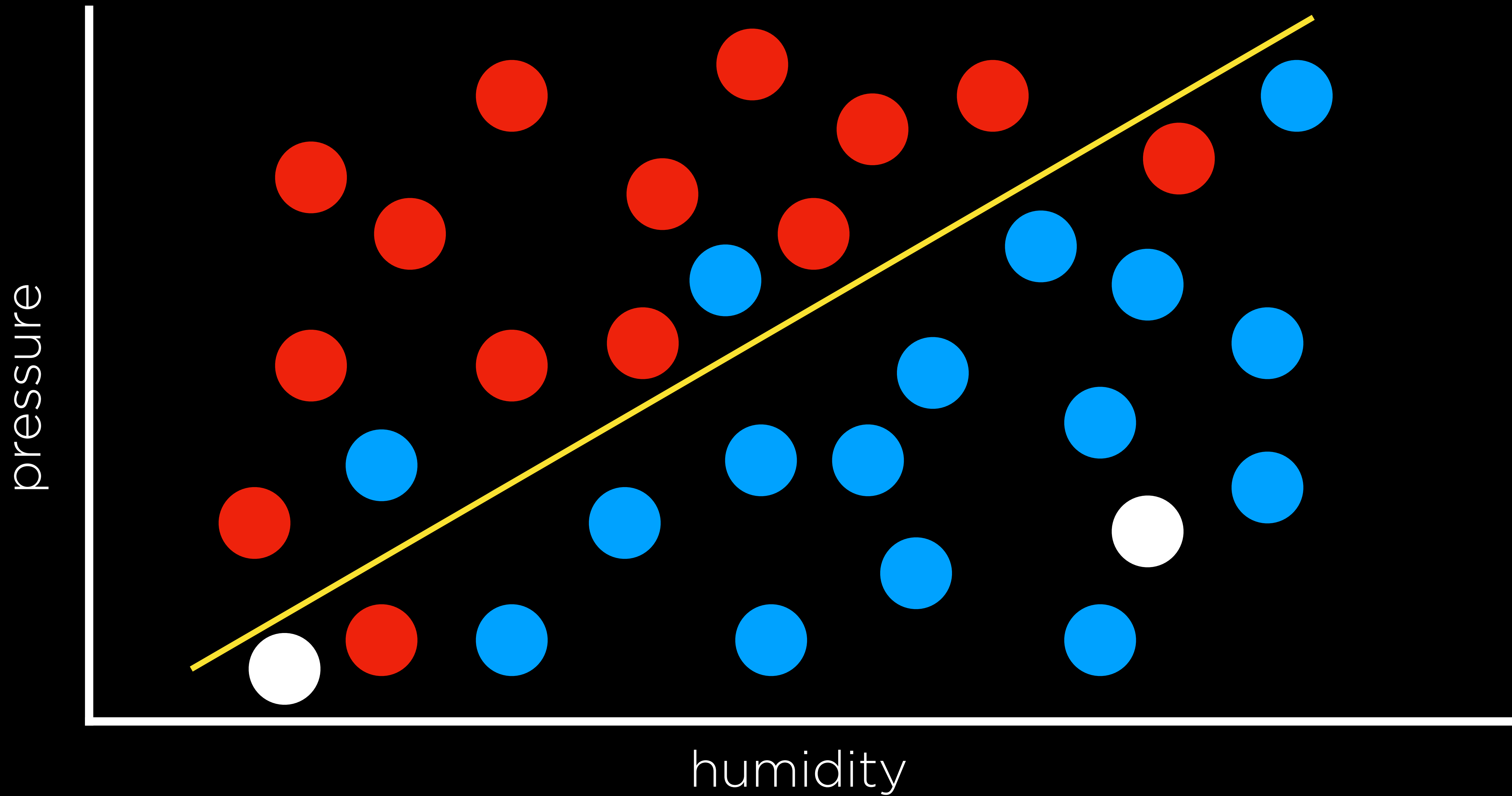
perceptron learning rule

Given data point (\mathbf{x}, y) , update each weight according to:

$$w_i = w_i + \alpha(\text{actual value} - \text{estimate}) \times x_i$$







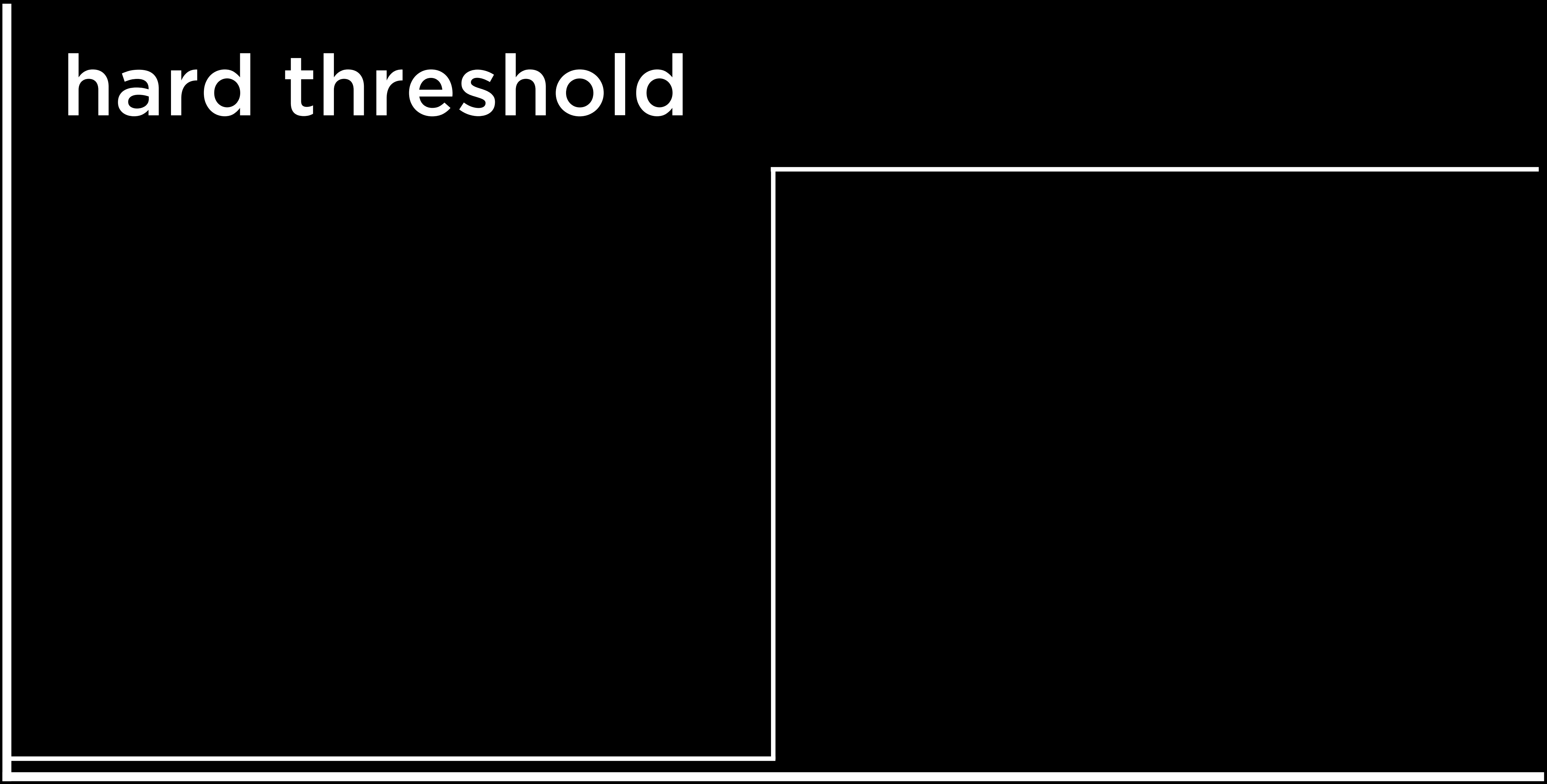
hard threshold

output

1

0

$\mathbf{w} \cdot \mathbf{x}$



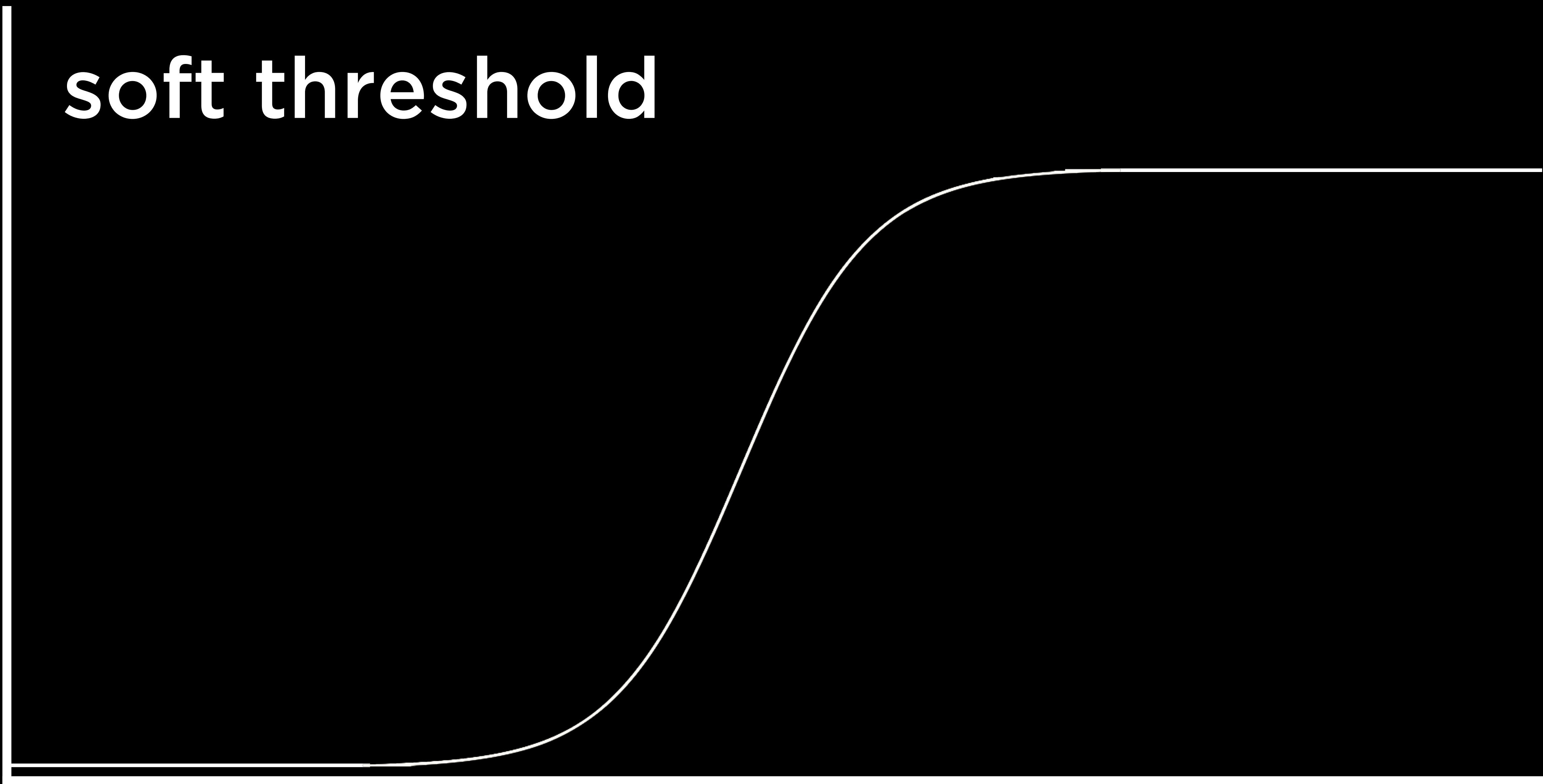
soft threshold

output

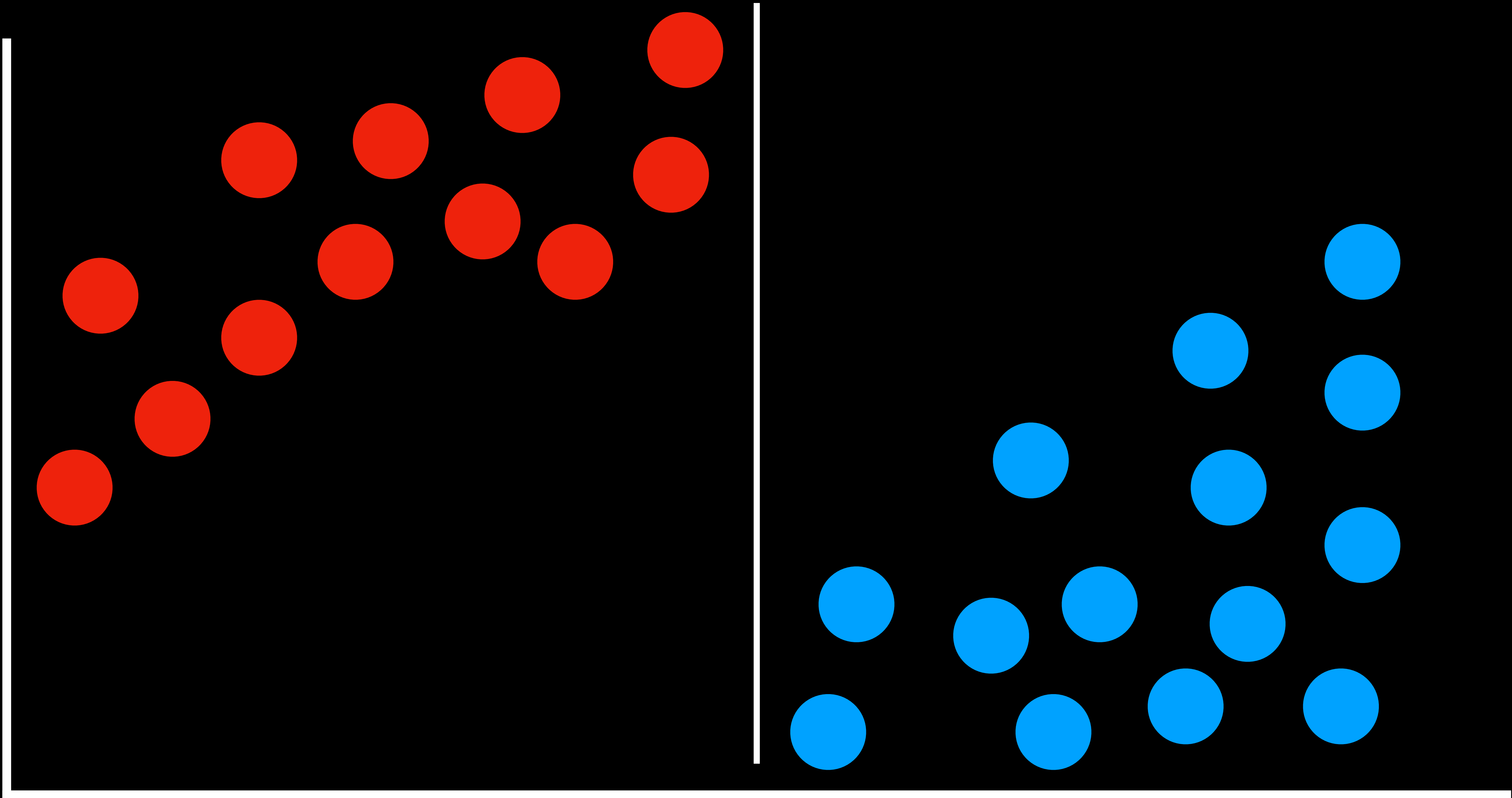
1

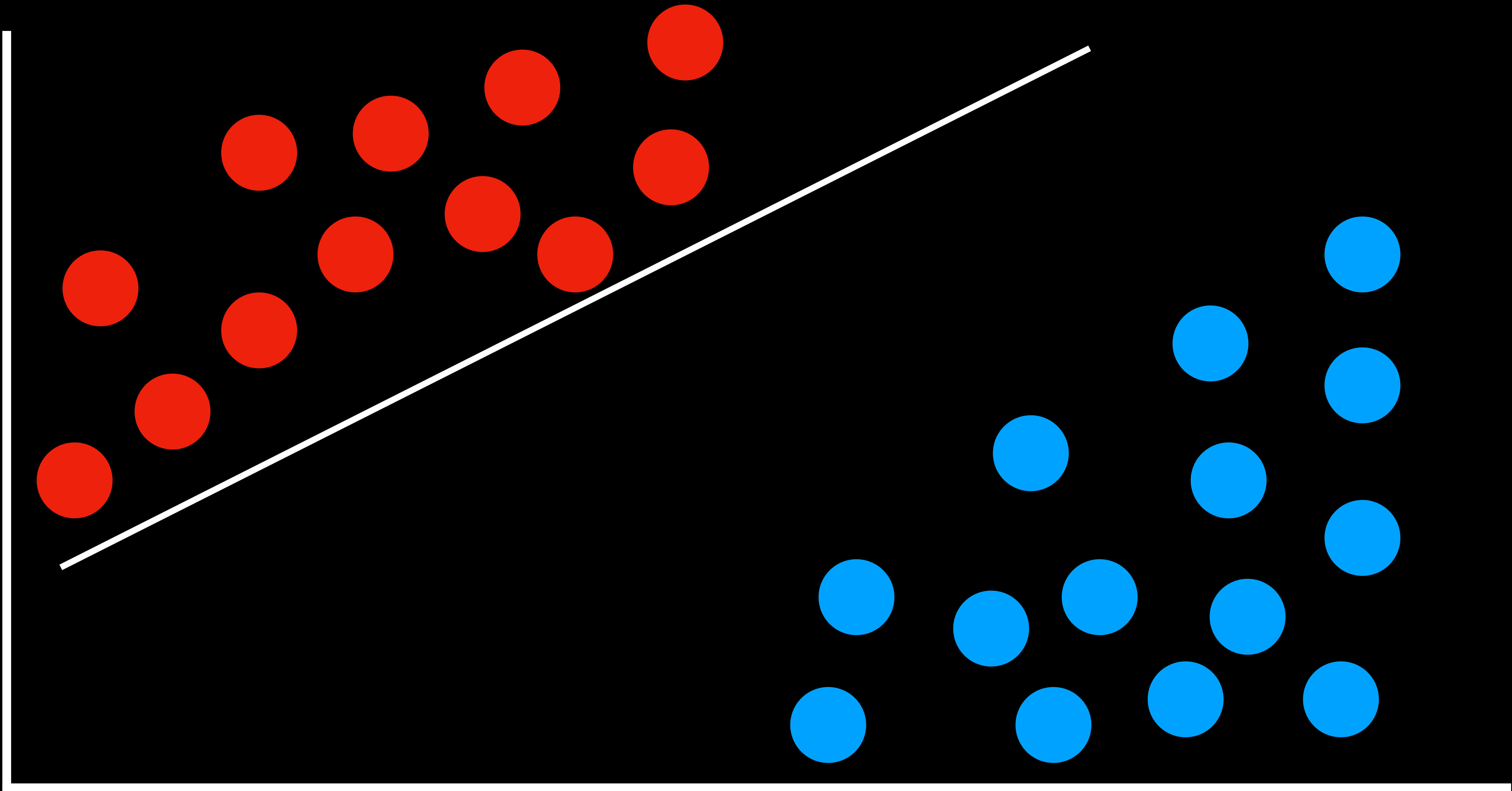
0

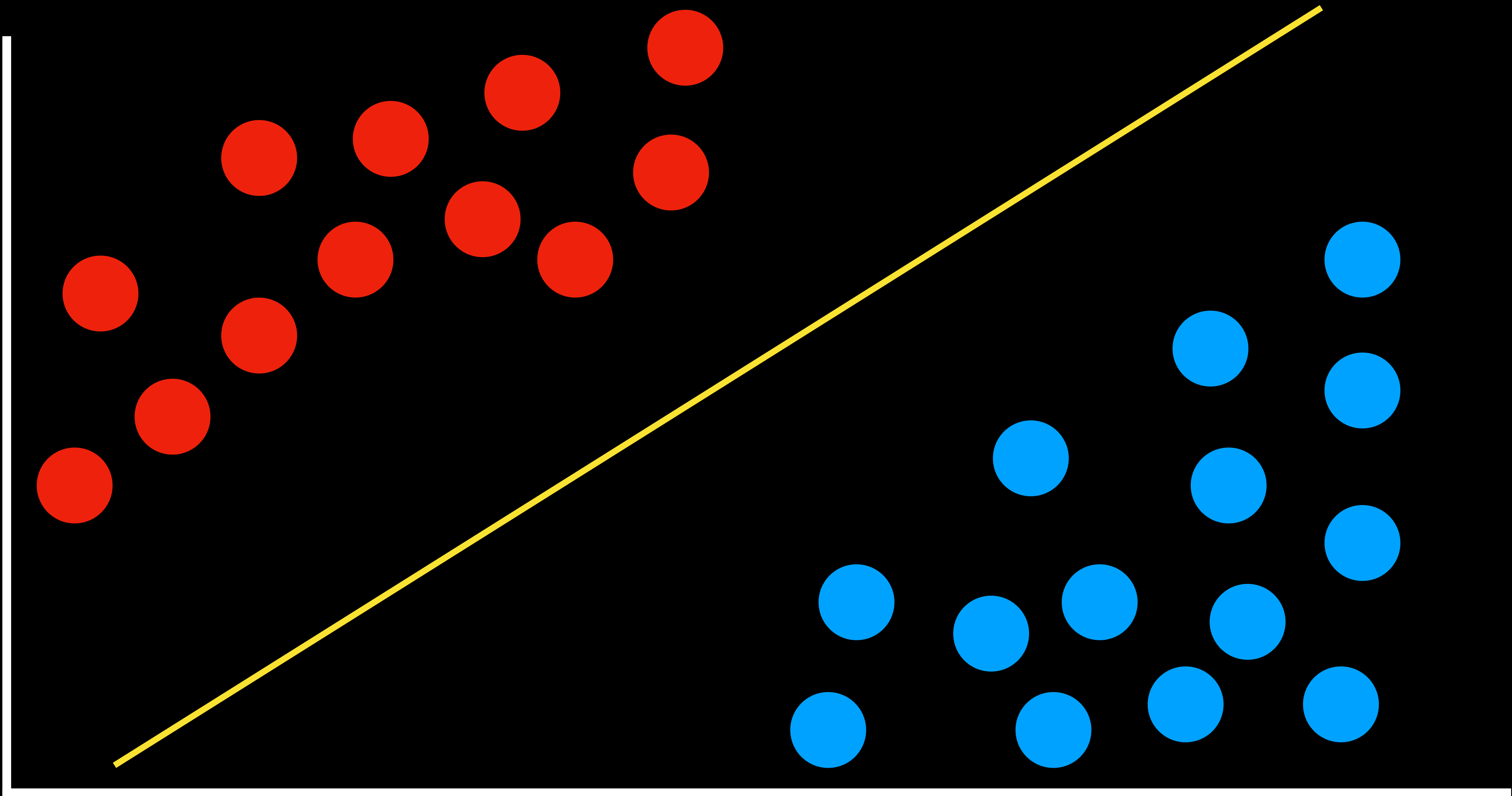
$\mathbf{w} \cdot \mathbf{x}$



Support Vector Machines

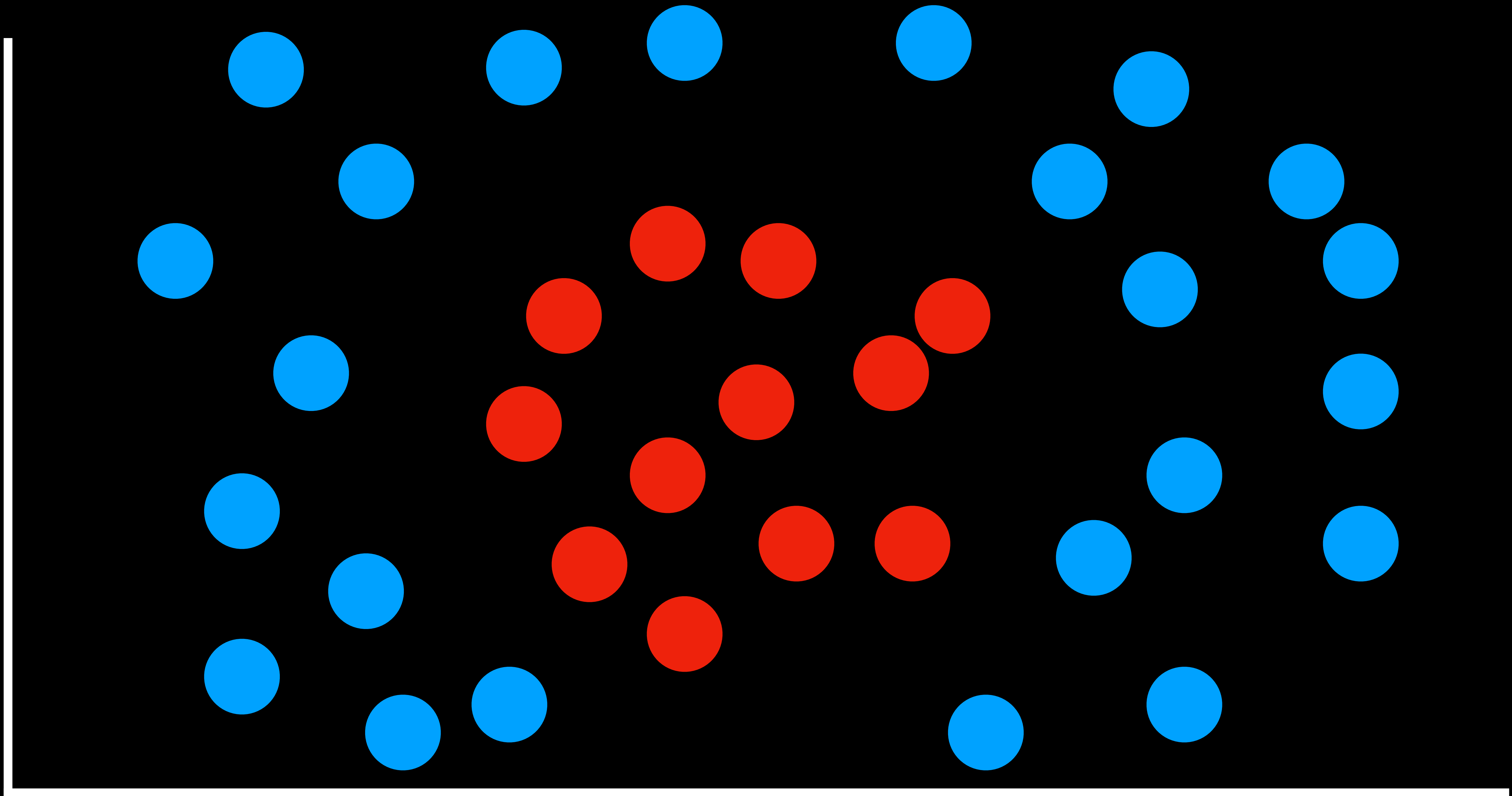


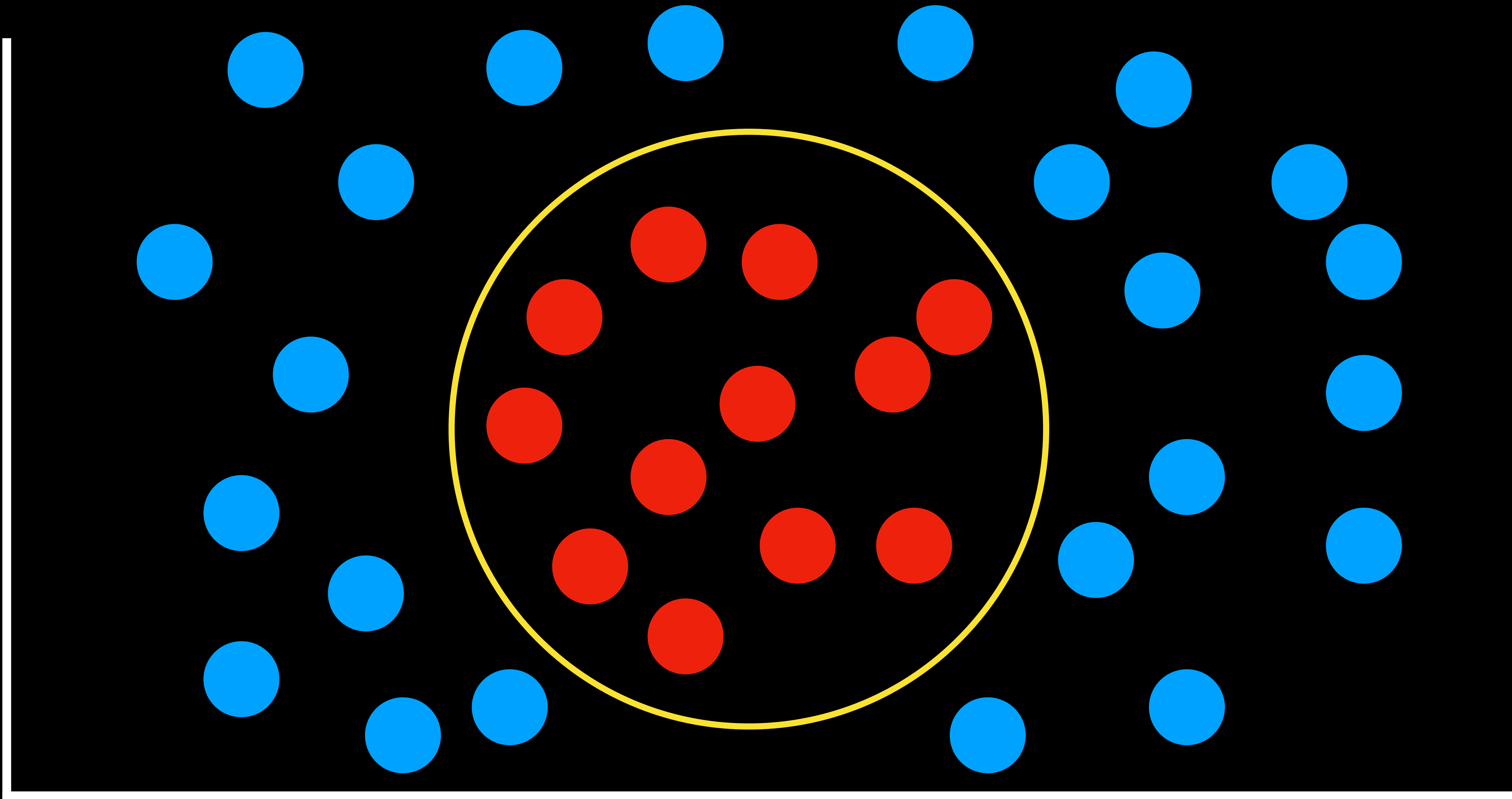




maximum margin separator

boundary that maximizes the distance
between any of the data points





regression

supervised learning task of learning a function mapping an input point to a continuous value

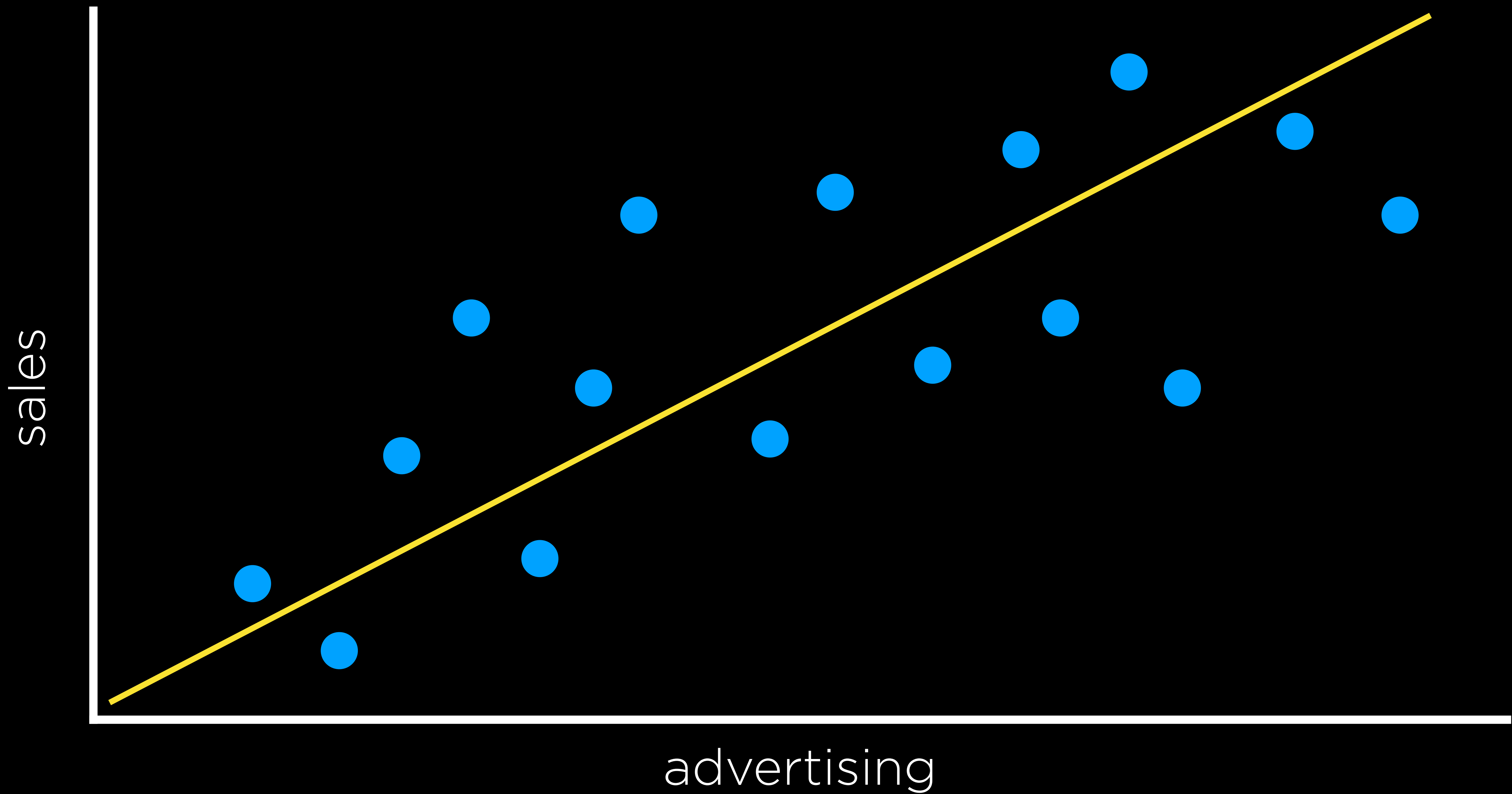
f(advertising)

$$f(1200) = 5800$$

$$f(2800) = 13400$$

$$f(1800) = 8400$$

h(advertising)



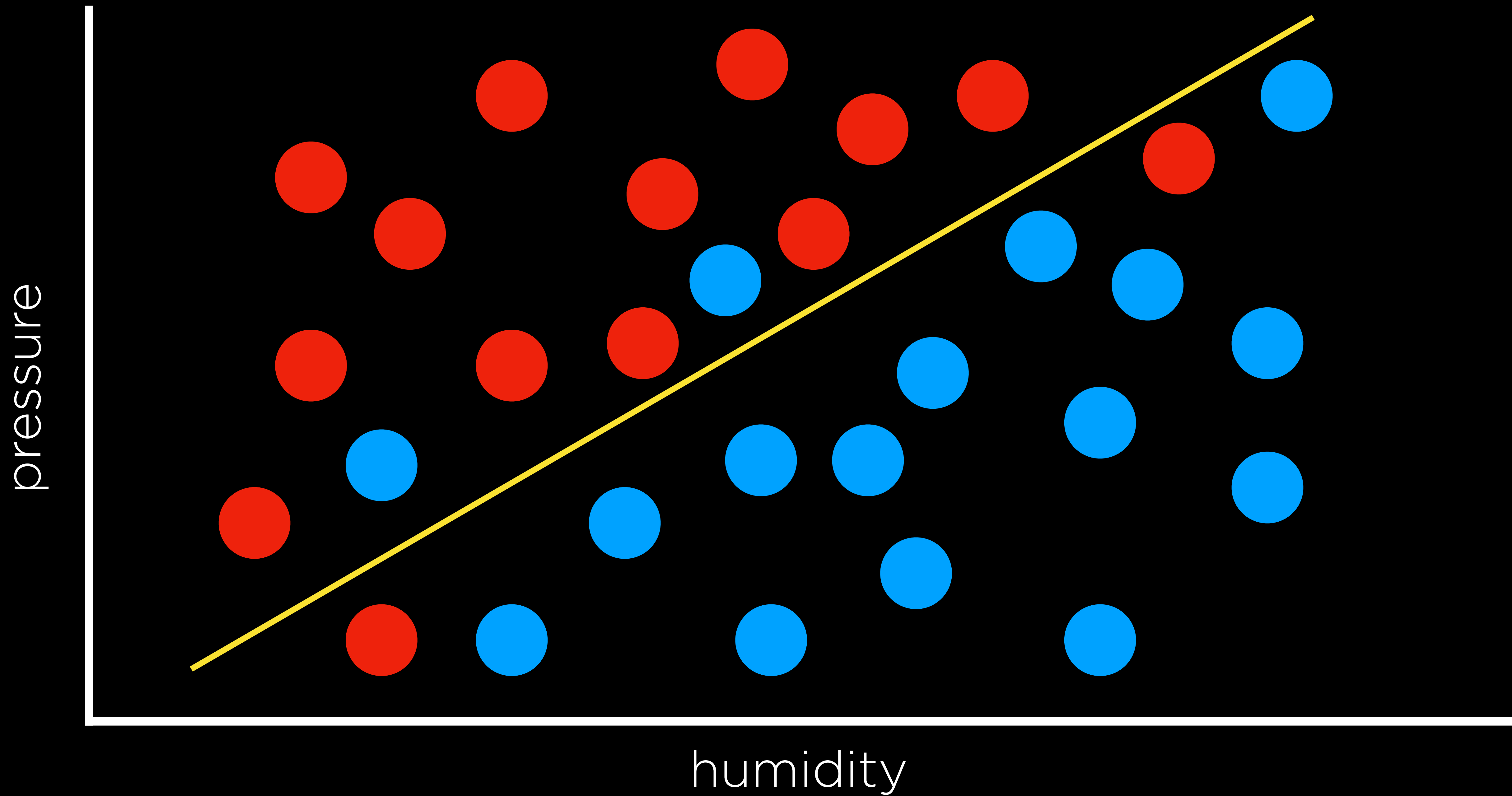
Evaluating Hypotheses

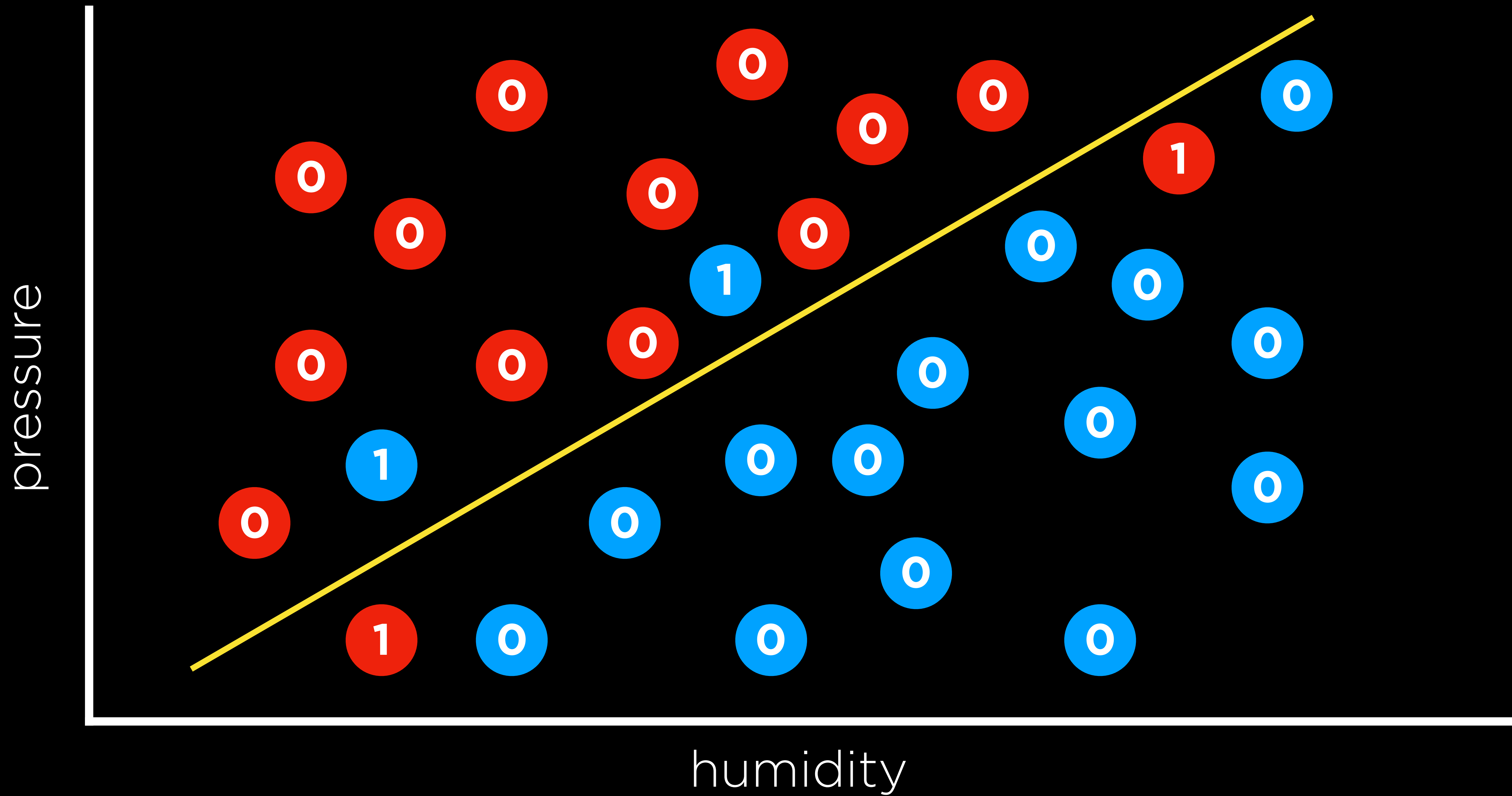
loss function

function that expresses how poorly our hypothesis performs

0-1 loss function

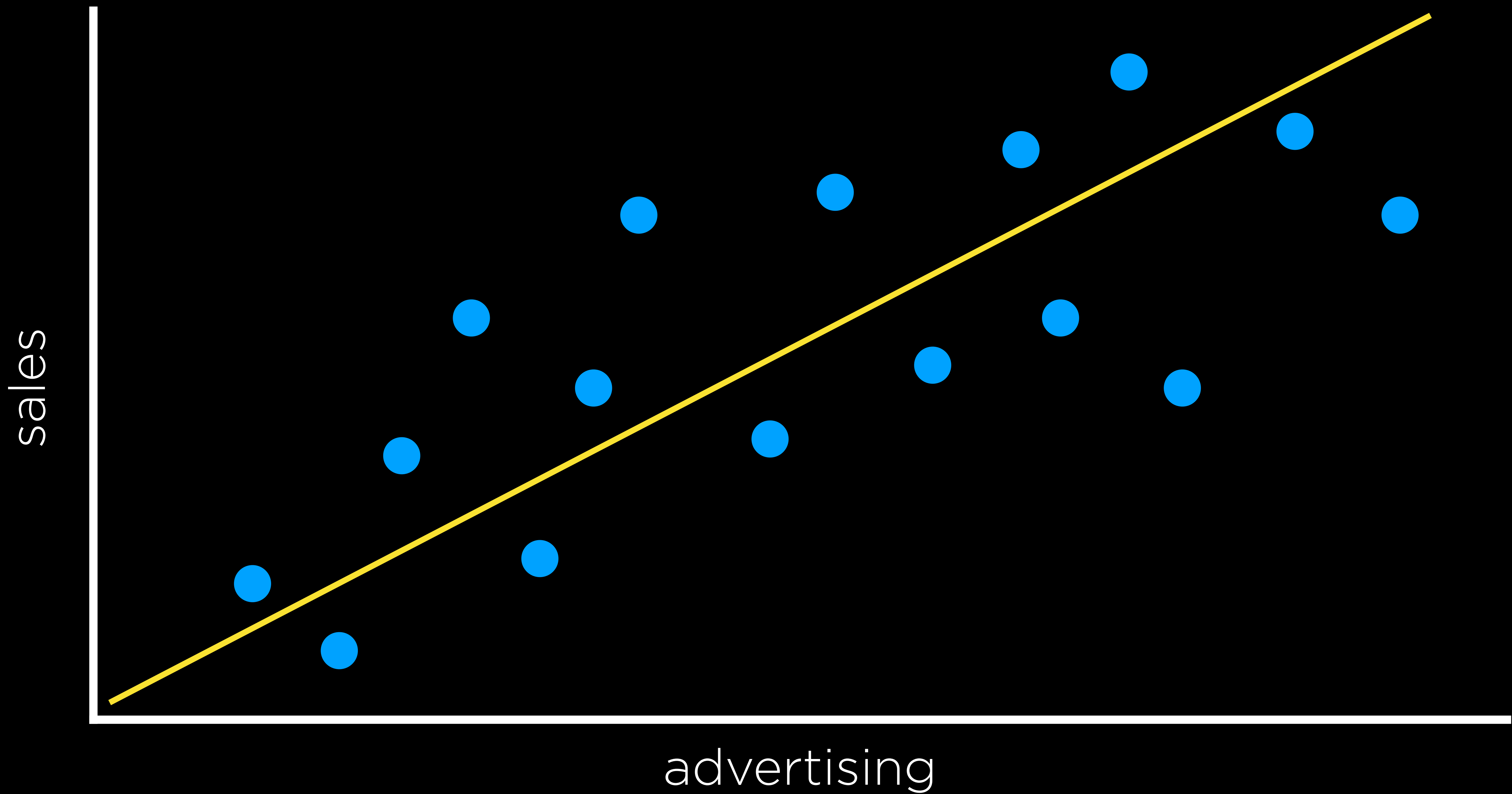
$$L(\text{actual}, \text{predicted}) = \begin{cases} 0 & \text{if actual} = \text{predicted}, \\ 1 & \text{otherwise} \end{cases}$$

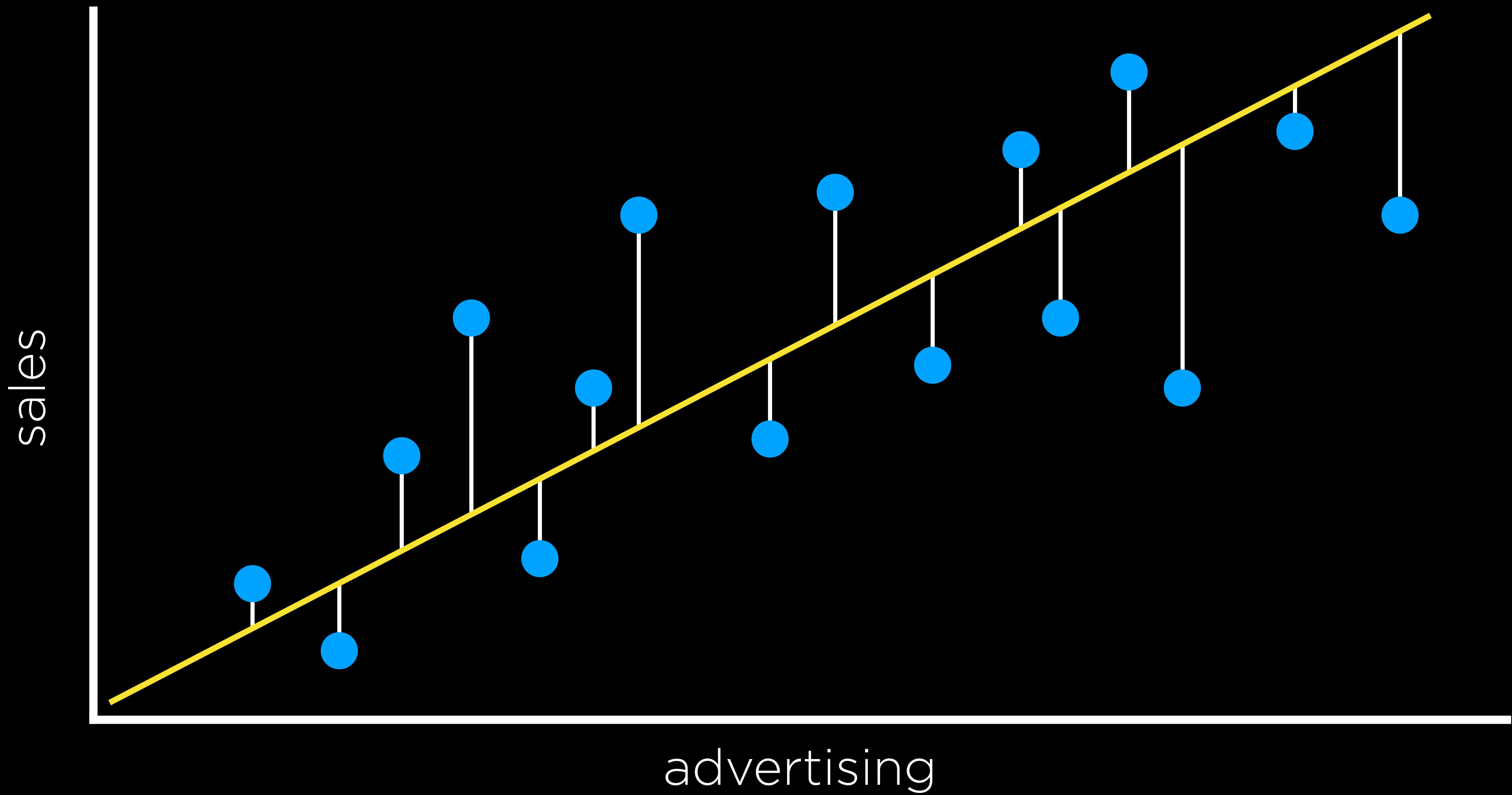




L₁ loss function

$$L(\text{actual}, \text{predicted}) = | \text{actual} - \text{predicted} |$$





L₂ loss function

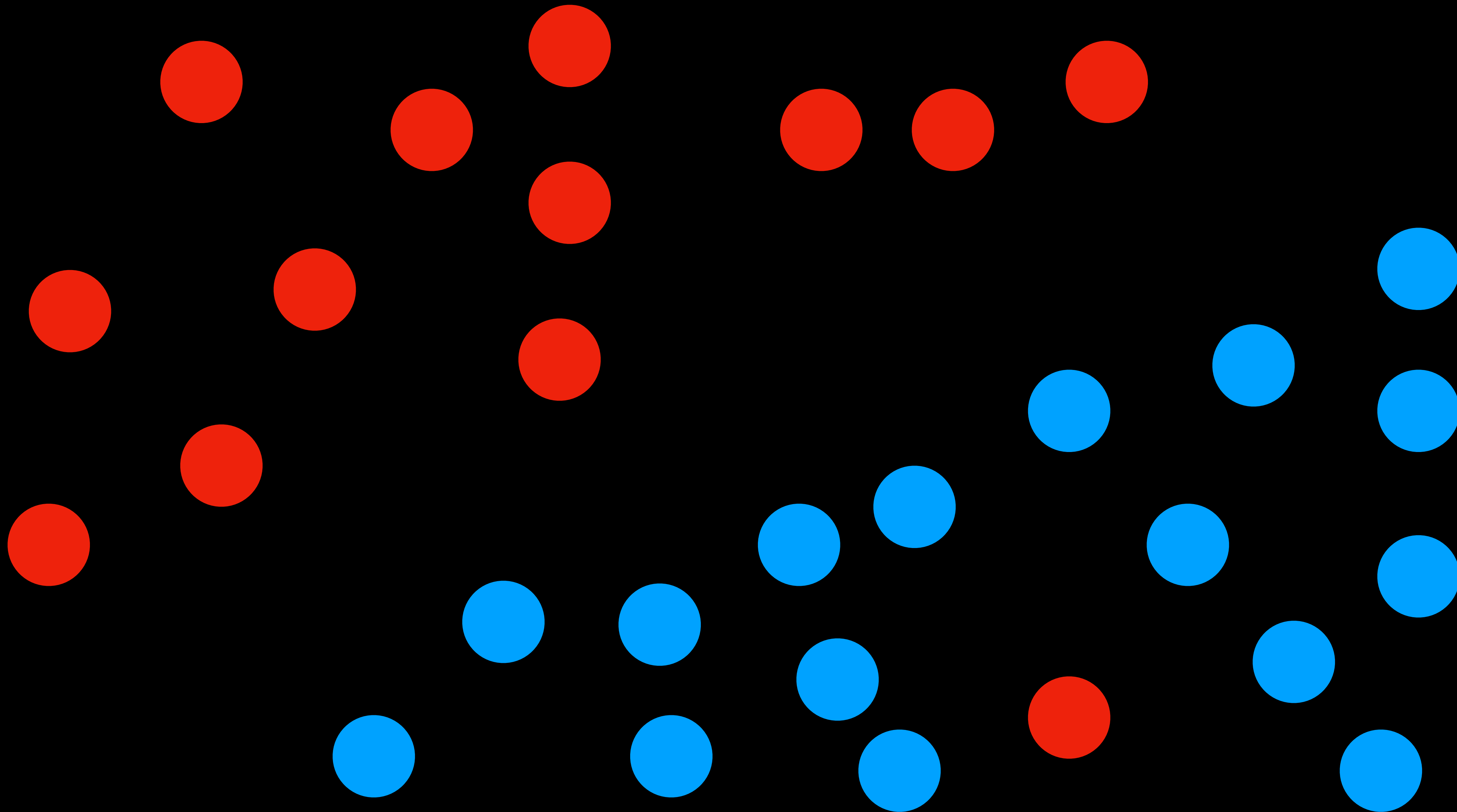
$$L(\text{actual}, \text{predicted}) = (\text{actual} - \text{predicted})^2$$

overfitting

a model that fits too closely to a particular data set and therefore may fail to generalize to future data

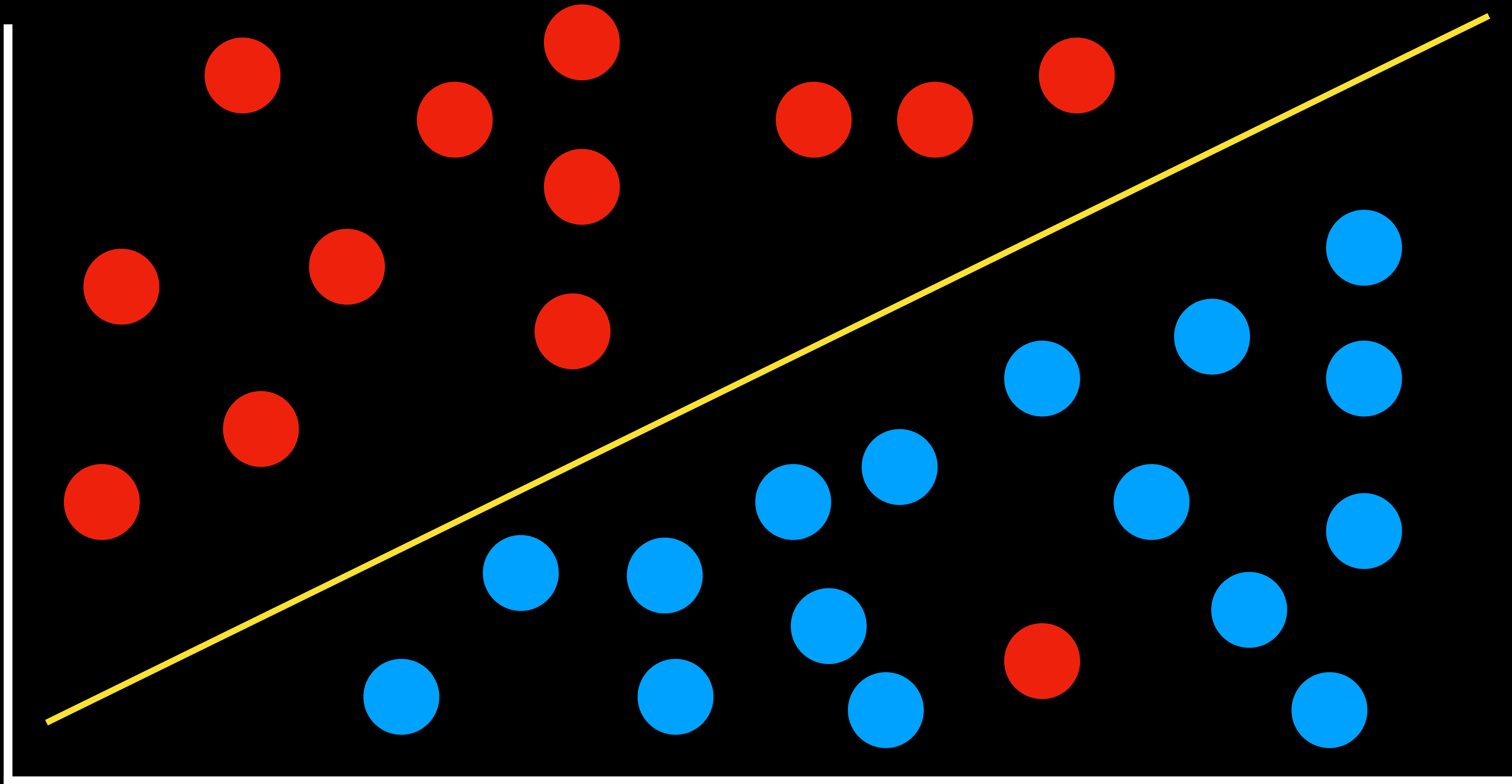
pressure

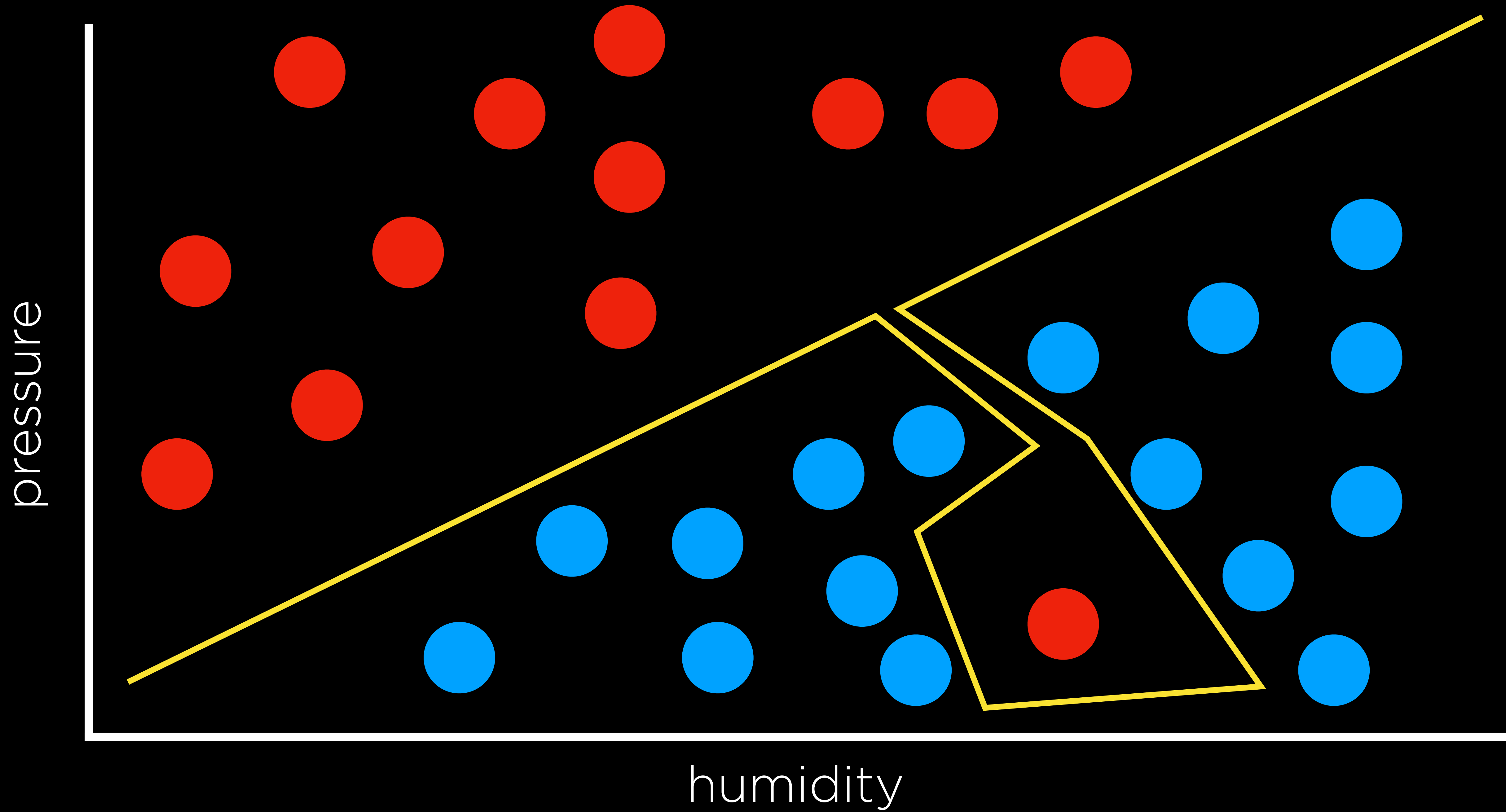
humidity

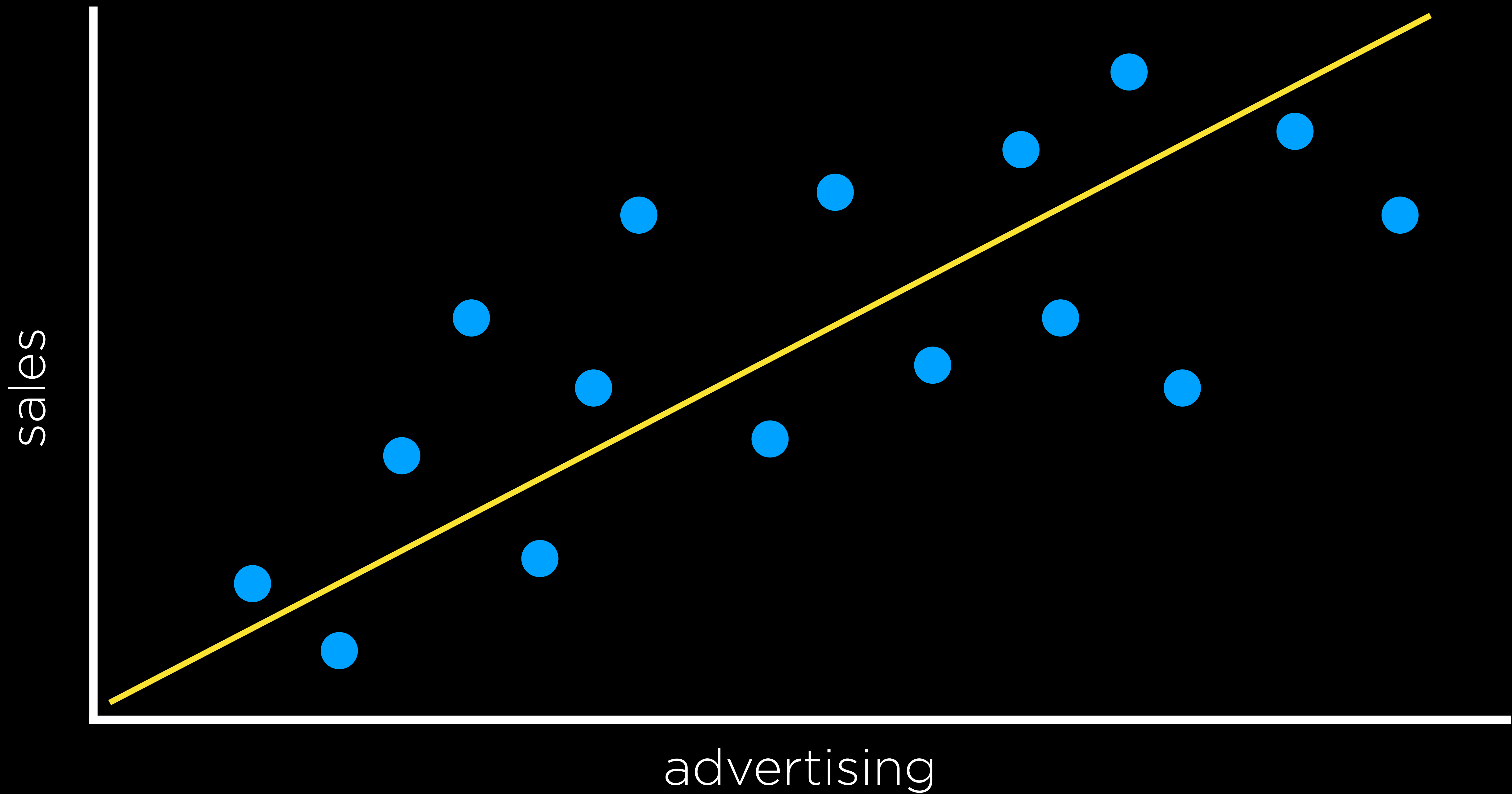


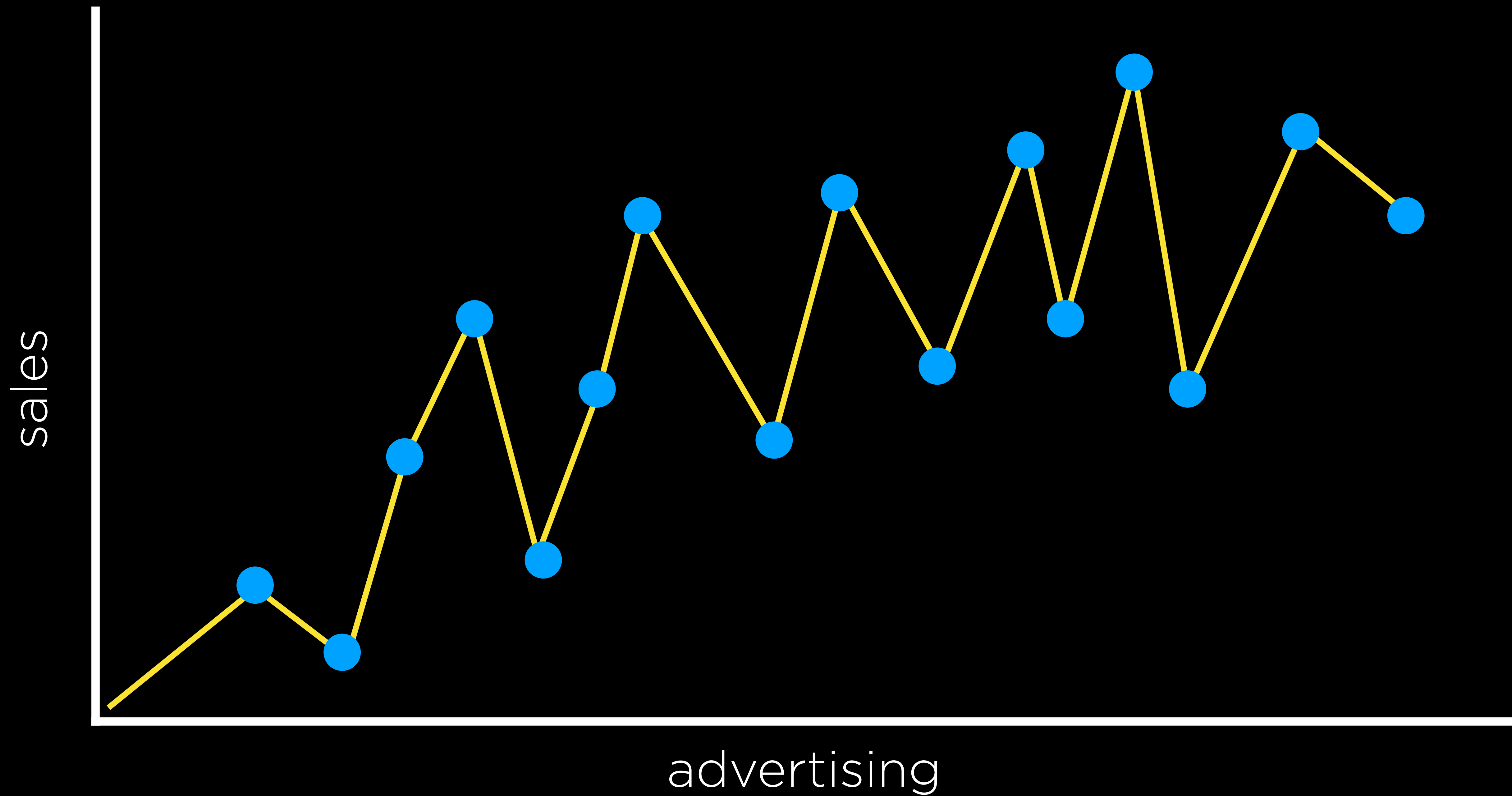
pressure

humidity









$$\textit{cost}(\mathbf{h}) = \textit{loss}(\mathbf{h})$$

$$\textit{cost}(\mathbf{h}) = \textit{loss}(\mathbf{h}) + \textit{complexity}(\mathbf{h})$$

$$\textit{cost}(\mathbf{h}) = \textit{loss}(\mathbf{h}) + \lambda \textit{complexity}(\mathbf{h})$$

regularization

penalizing hypotheses that are more complex
to favor simpler, more general hypotheses

$$cost(h) = loss(h) + \lambda complexity(h)$$

holdout cross-validation

splitting data into a **training set** and a **test set**, such that learning happens on the training set and is evaluated on the test set

k -fold cross-validation

splitting data into k sets, and experimenting k times, using each set as a test set once, and using remaining data as training set

scikit-learn

Reinforcement Learning

reinforcement learning

given a set of rewards or punishments, learn
what actions to take in the future



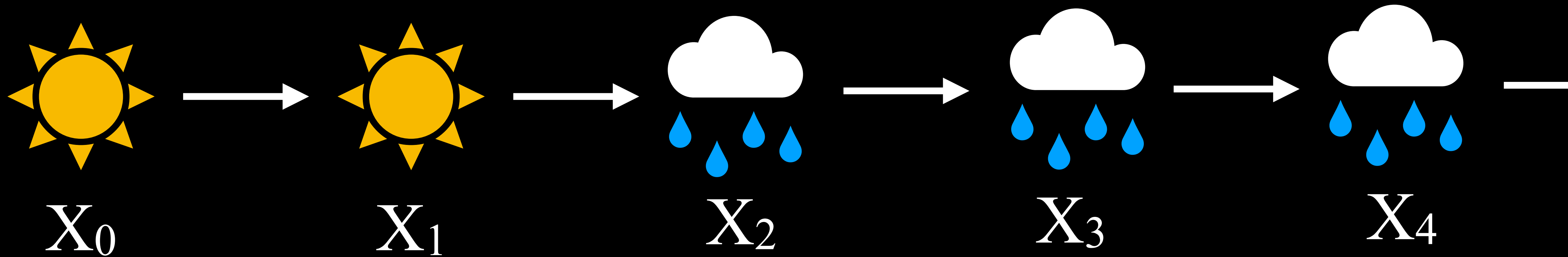
Markov Decision Process

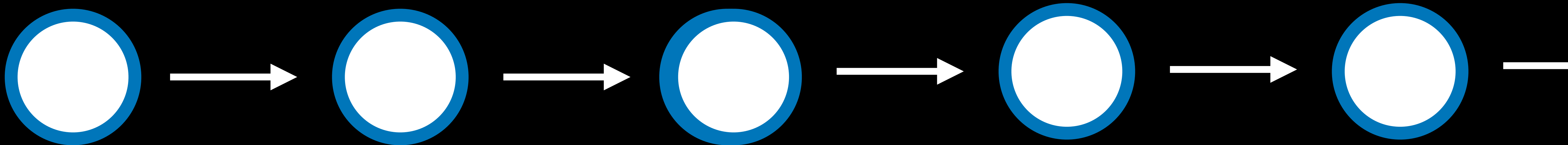
model for decision-making, representing states, actions, and their rewards

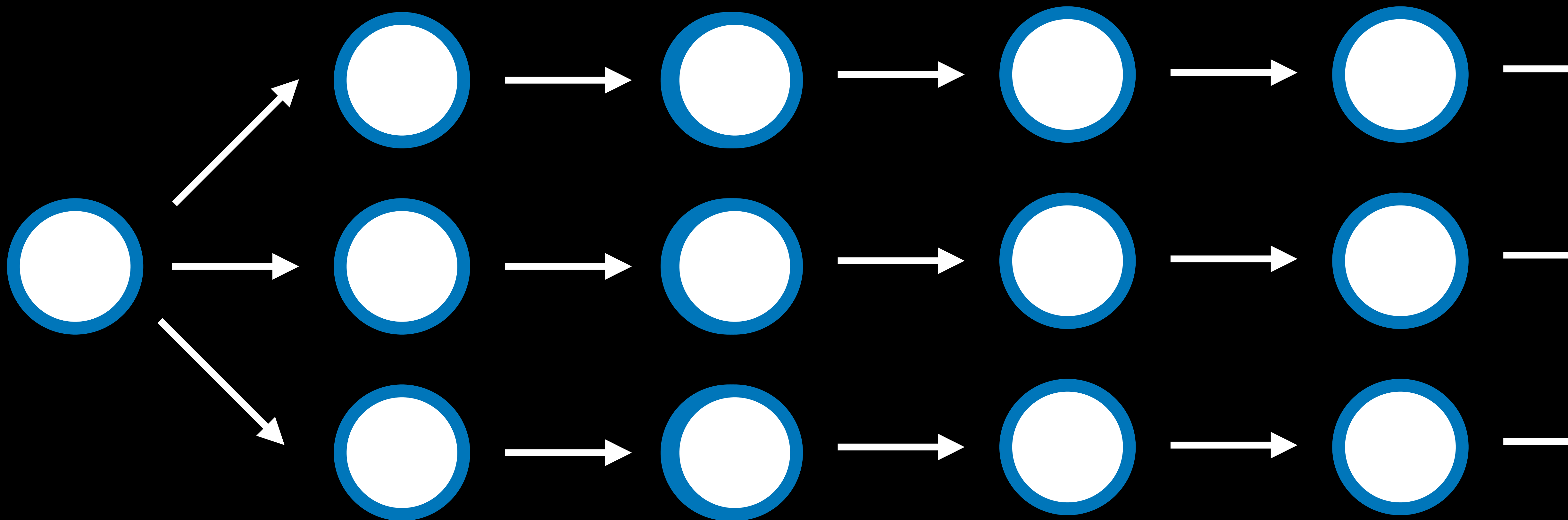
Markov Decision Process

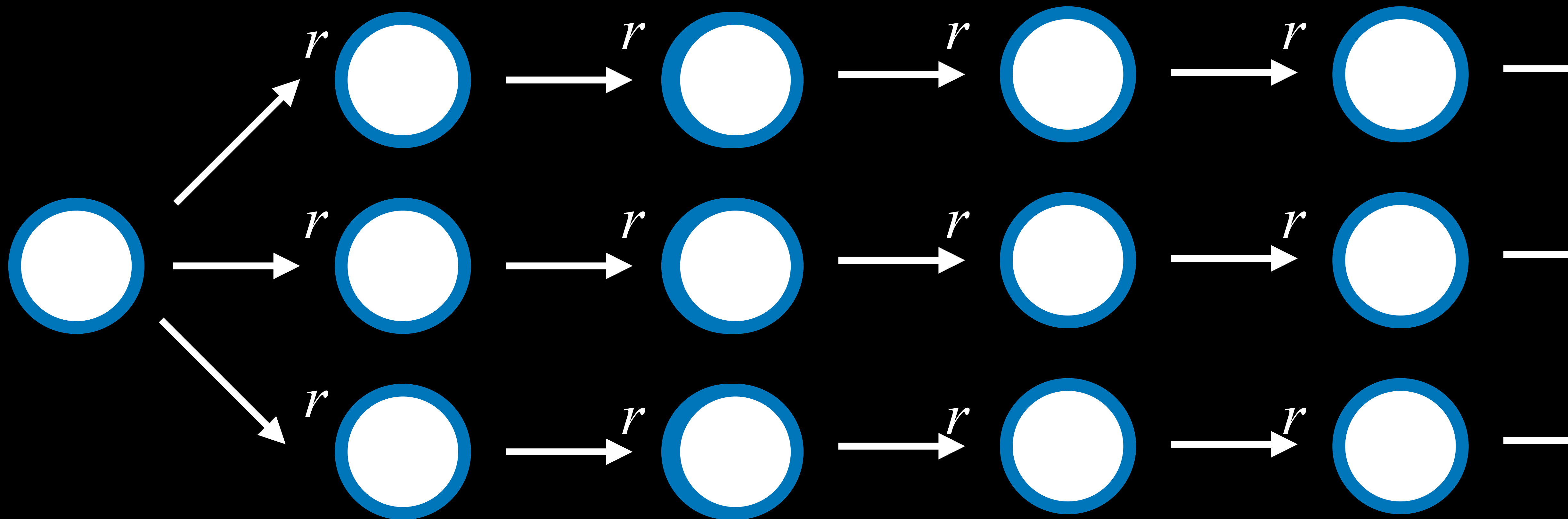
model for decision-making, representing states, actions, and their rewards

Markov Chain



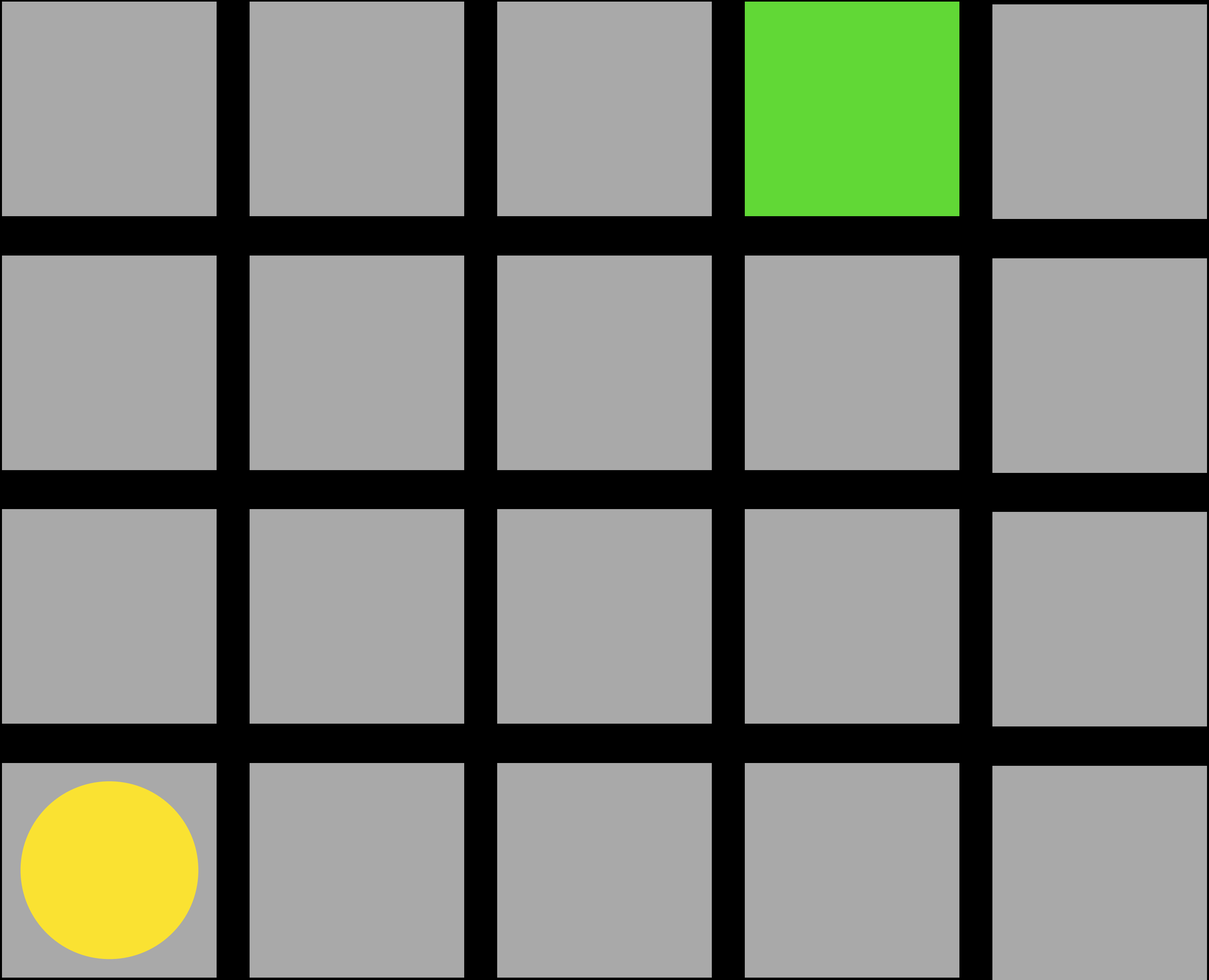


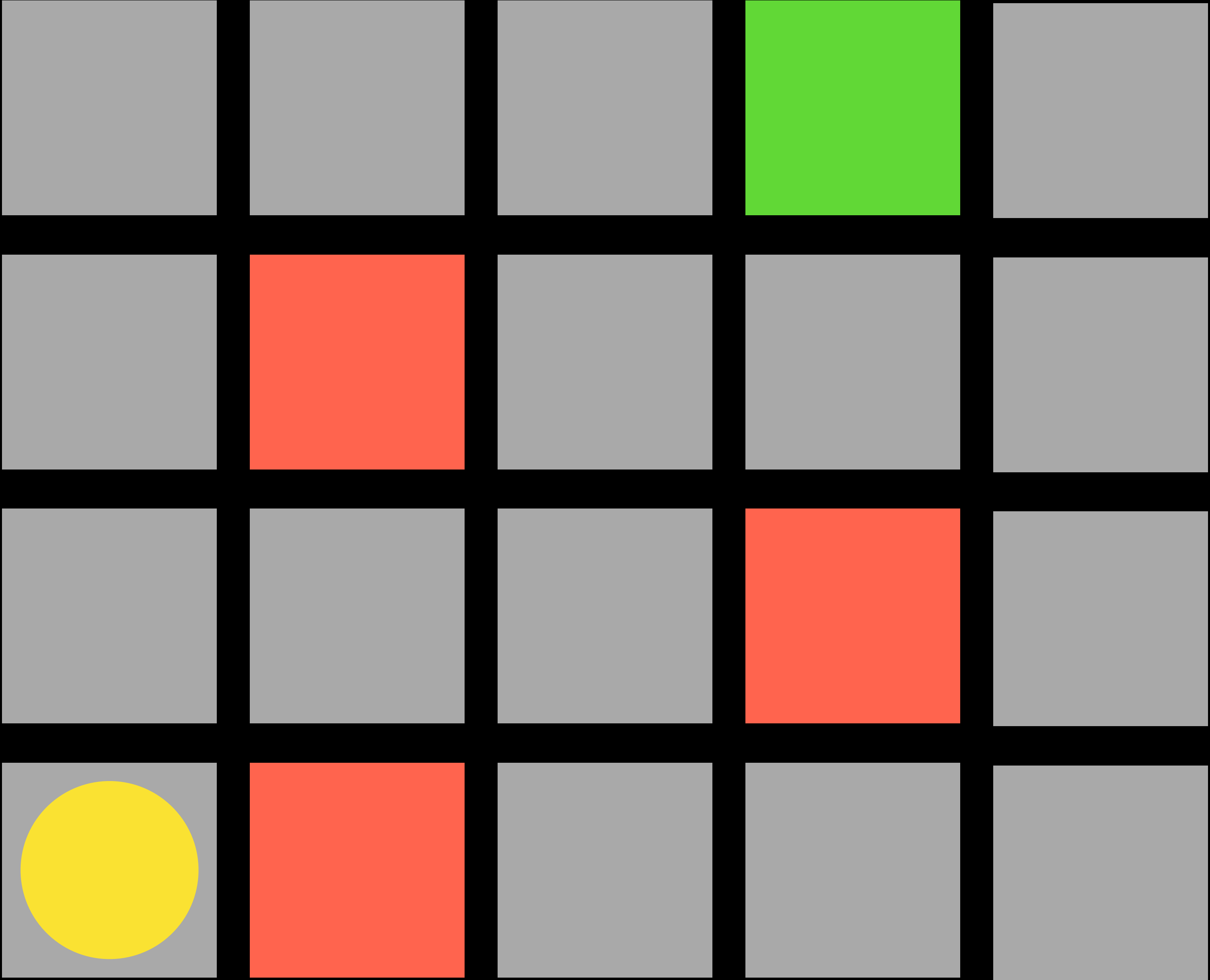


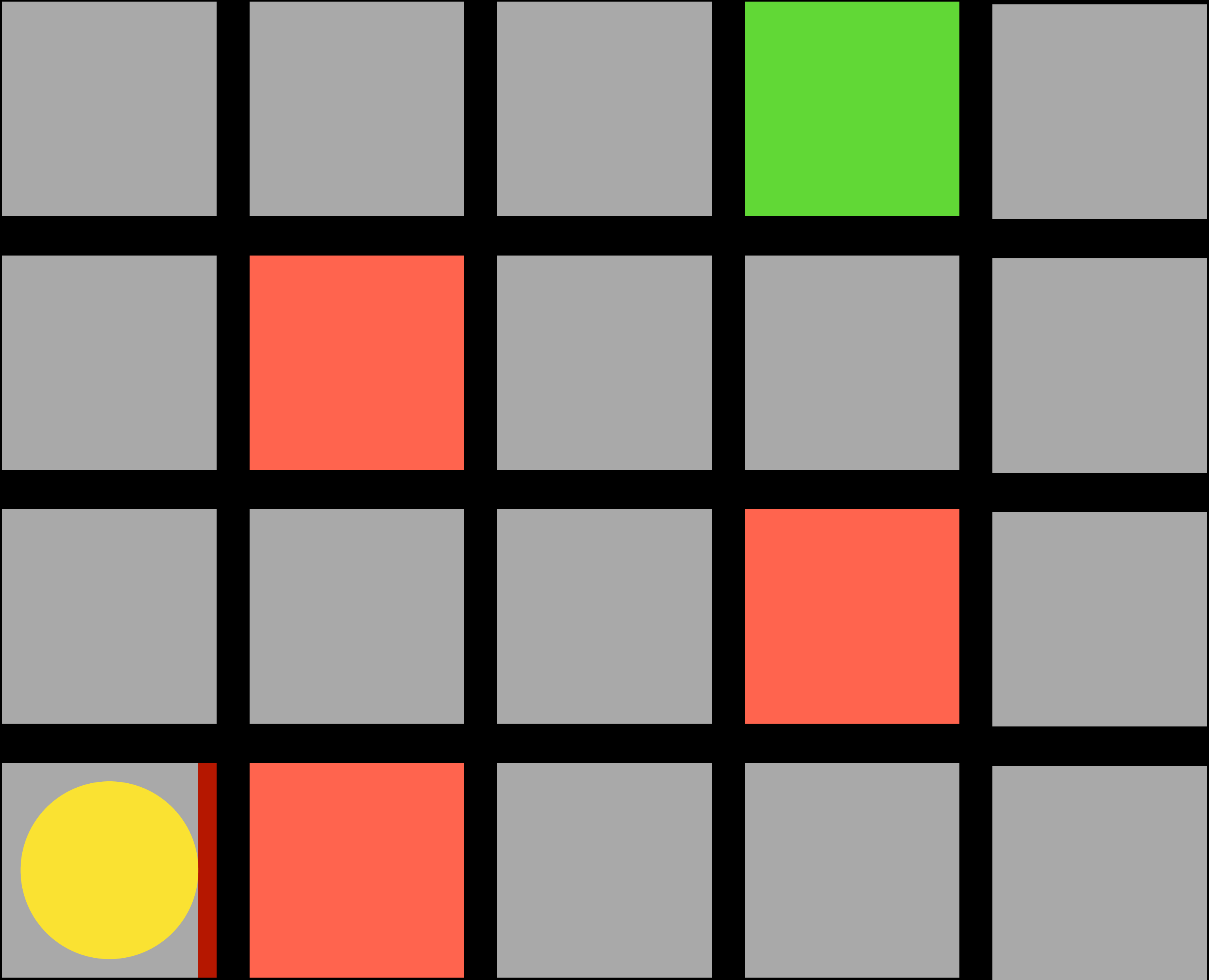


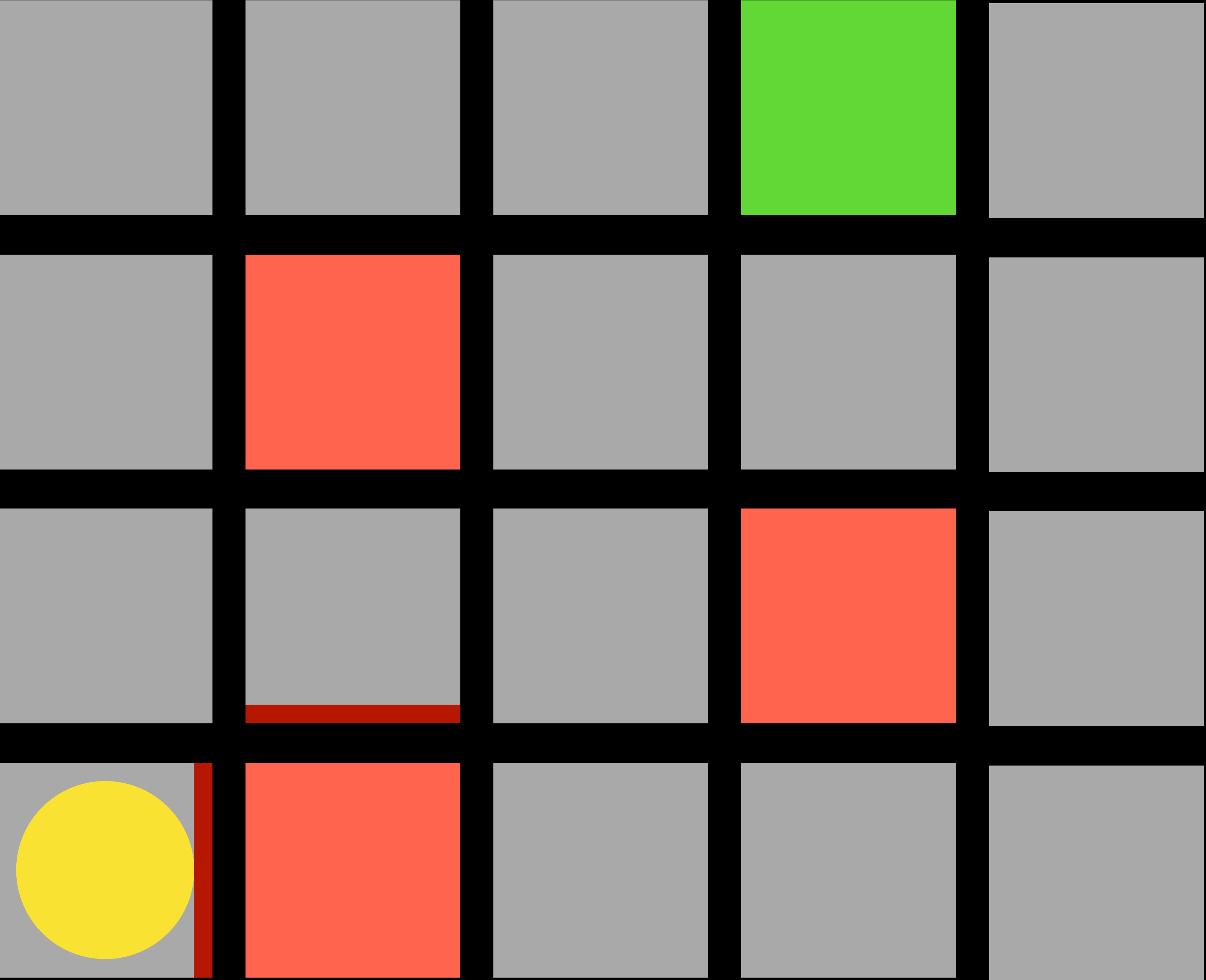
Markov Decision Process

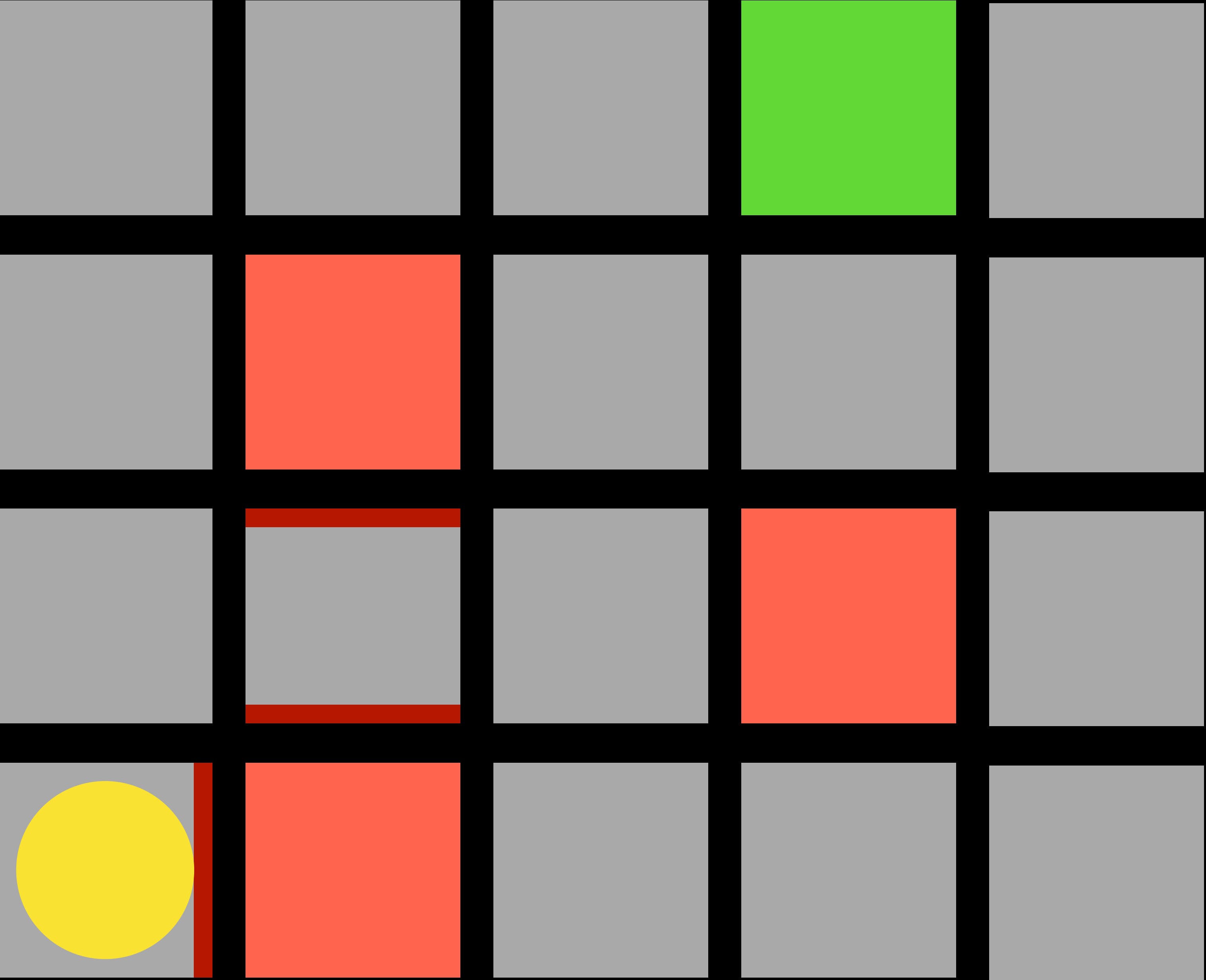
- Set of states \mathcal{S}
- Set of actions $\text{ACTIONS}(s)$
- Transition model $P(s' | s, a)$
- Reward function $R(s, a, s')$

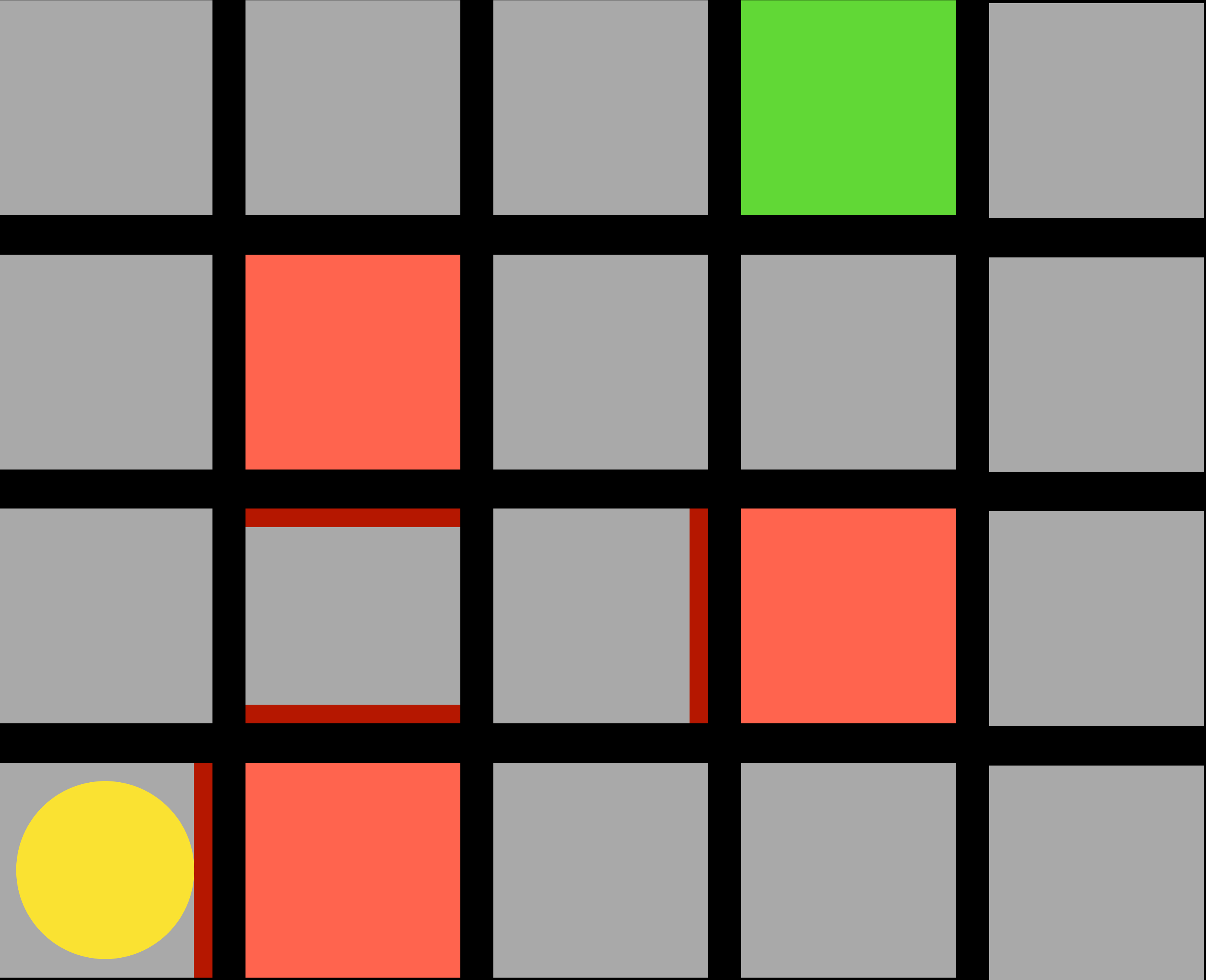


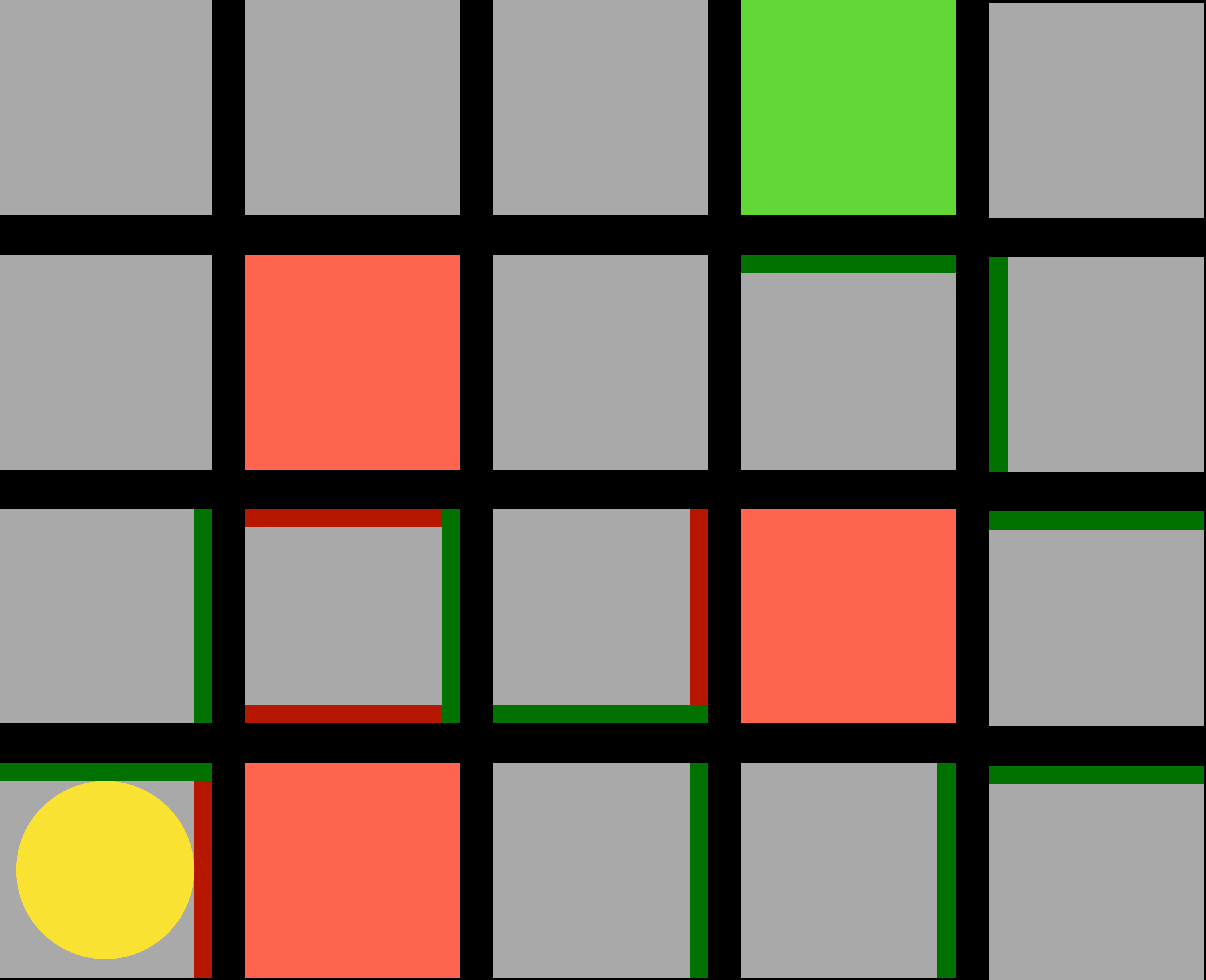


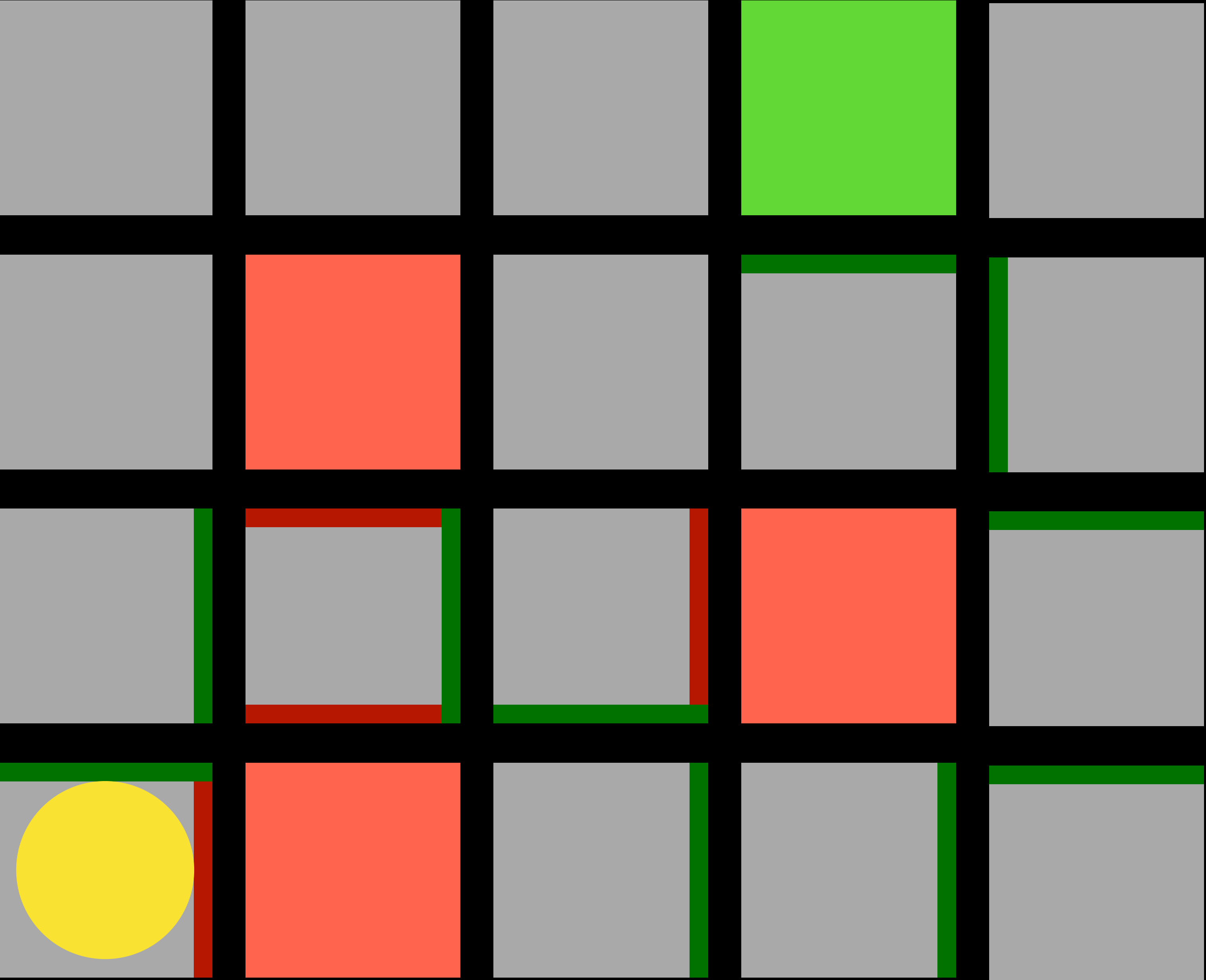












Q-learning

method for learning a function $Q(s, a)$,
estimate of the value of performing action a
in state s

Q-learning Overview

- Start with $Q(s, a) = 0$ for all s, a
- When we taken an action and receive a reward:
 - Estimate the value of $Q(s, a)$ based on current reward and expected future rewards
 - Update $Q(s, a)$ to take into account old estimate as well as our new estimate

Q-learning

- Start with $Q(s, a) = 0$ for all s, a
- Every time we take an action a in state s and observe a reward r , we update:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(\text{new value estimate} - \text{old value estimate})$$

Q-learning

- Start with $Q(s, a) = 0$ for all s, a
- Every time we take an action a in state s and observe a reward r , we update:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(\text{new value estimate} - Q(s, a))$$

when $\alpha=1$, we only consider new value estimate

Q-learning

- Start with $Q(s, a) = 0$ for all s, a
- Every time we take an action a in state s and observe a reward r , we update:

$$Q(s, a) \leftarrow Q(s, a) + \alpha((r + \text{future reward estimate}) - Q(s, a))$$

Q-learning

- Start with $Q(s, a) = 0$ for all s, a
- Every time we take an action a in state s and observe a reward r , we update:

$$Q(s, a) \leftarrow Q(s, a) + \alpha((r + \max_{a'} Q(s', a')) - Q(s, a))$$

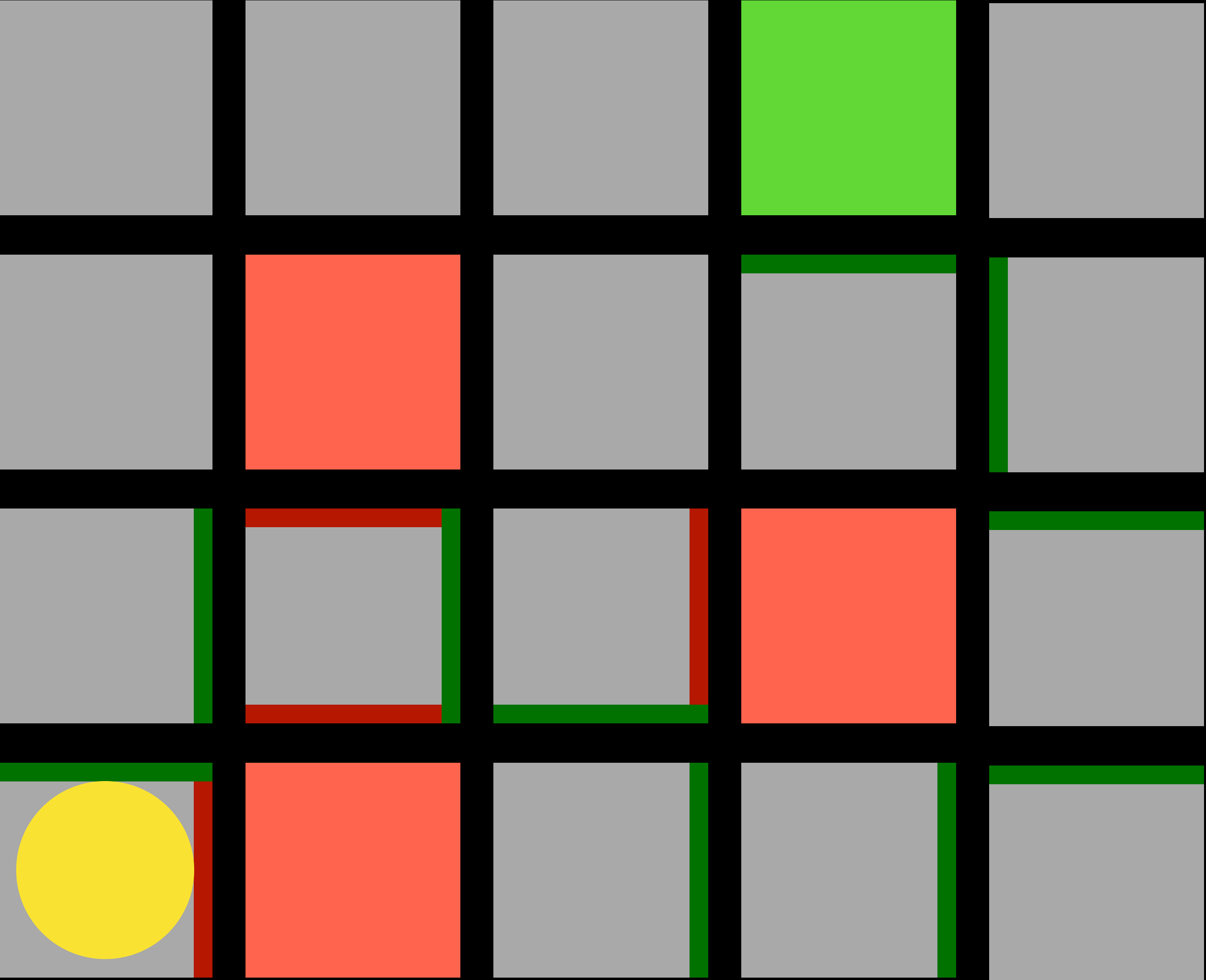
Q-learning

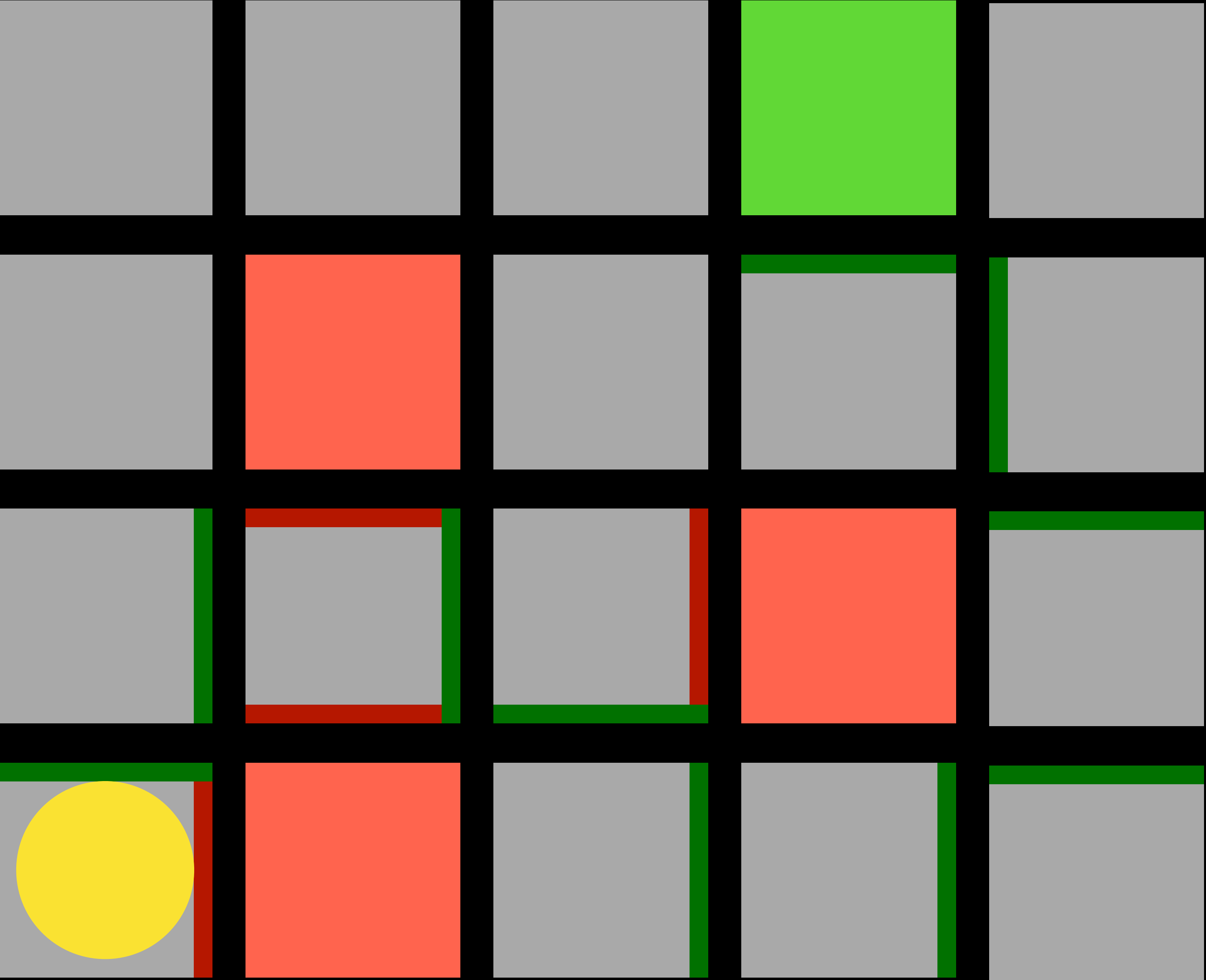
- Start with $Q(s, a) = 0$ for all s, a
- Every time we take an action a in state s and observe a reward r , we update:

$$Q(s, a) \leftarrow Q(s, a) + \alpha((r + \gamma \max_{a'} Q(s', a')) - Q(s, a))$$

Greedy Decision-Making

- When in state s , choose action a with highest $Q(s, a)$





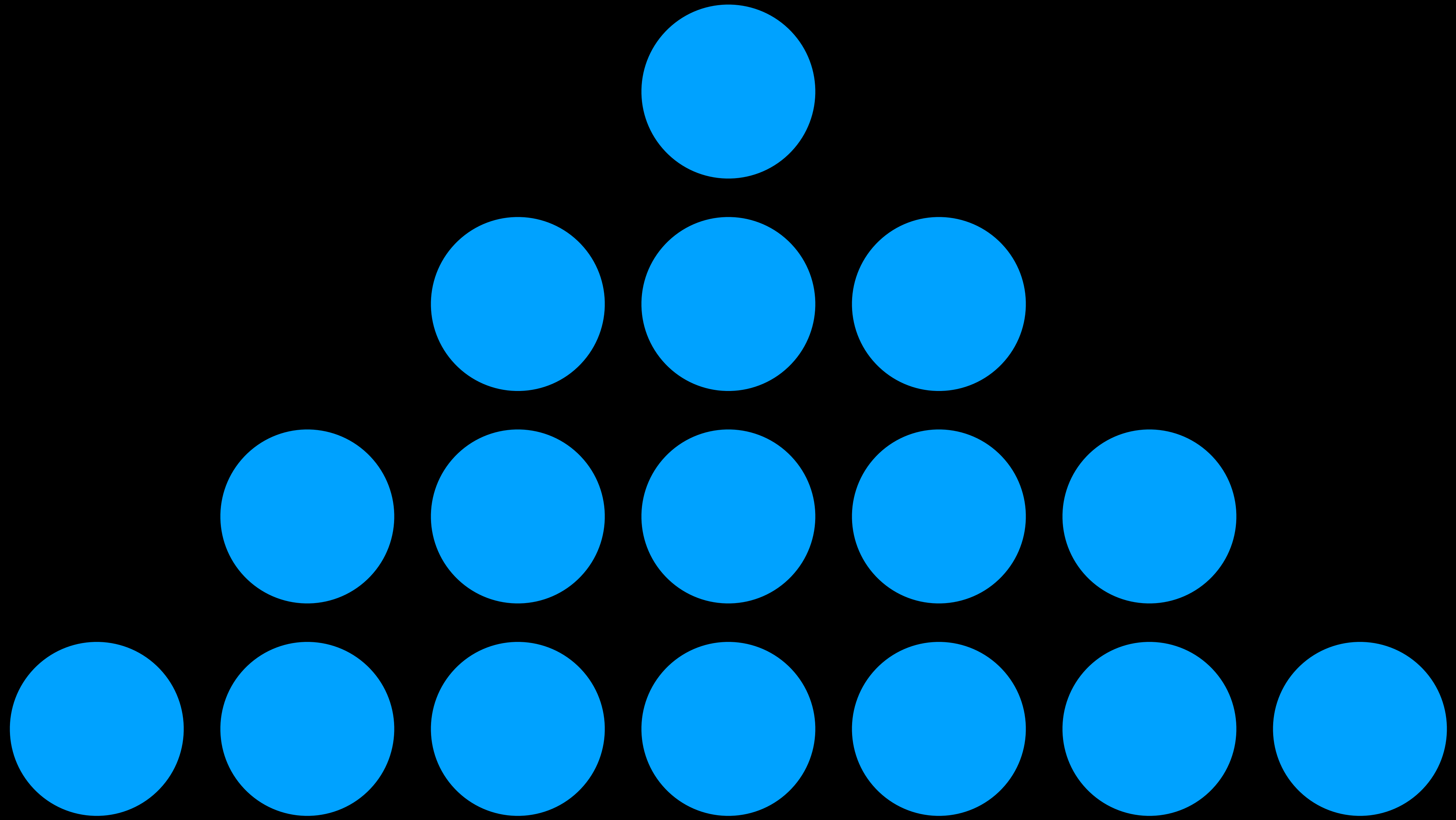
Explore vs. Exploit

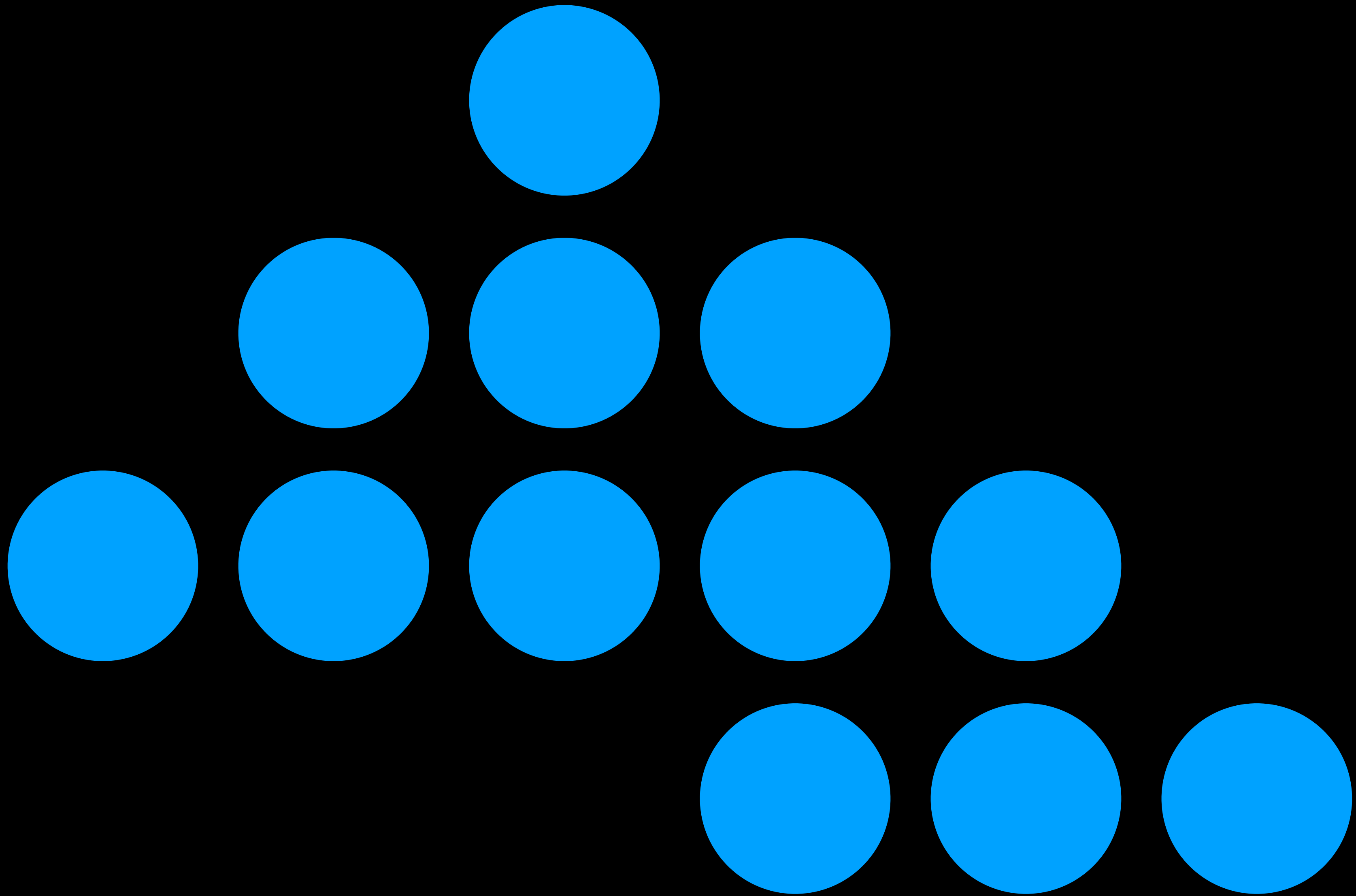
ϵ -greedy

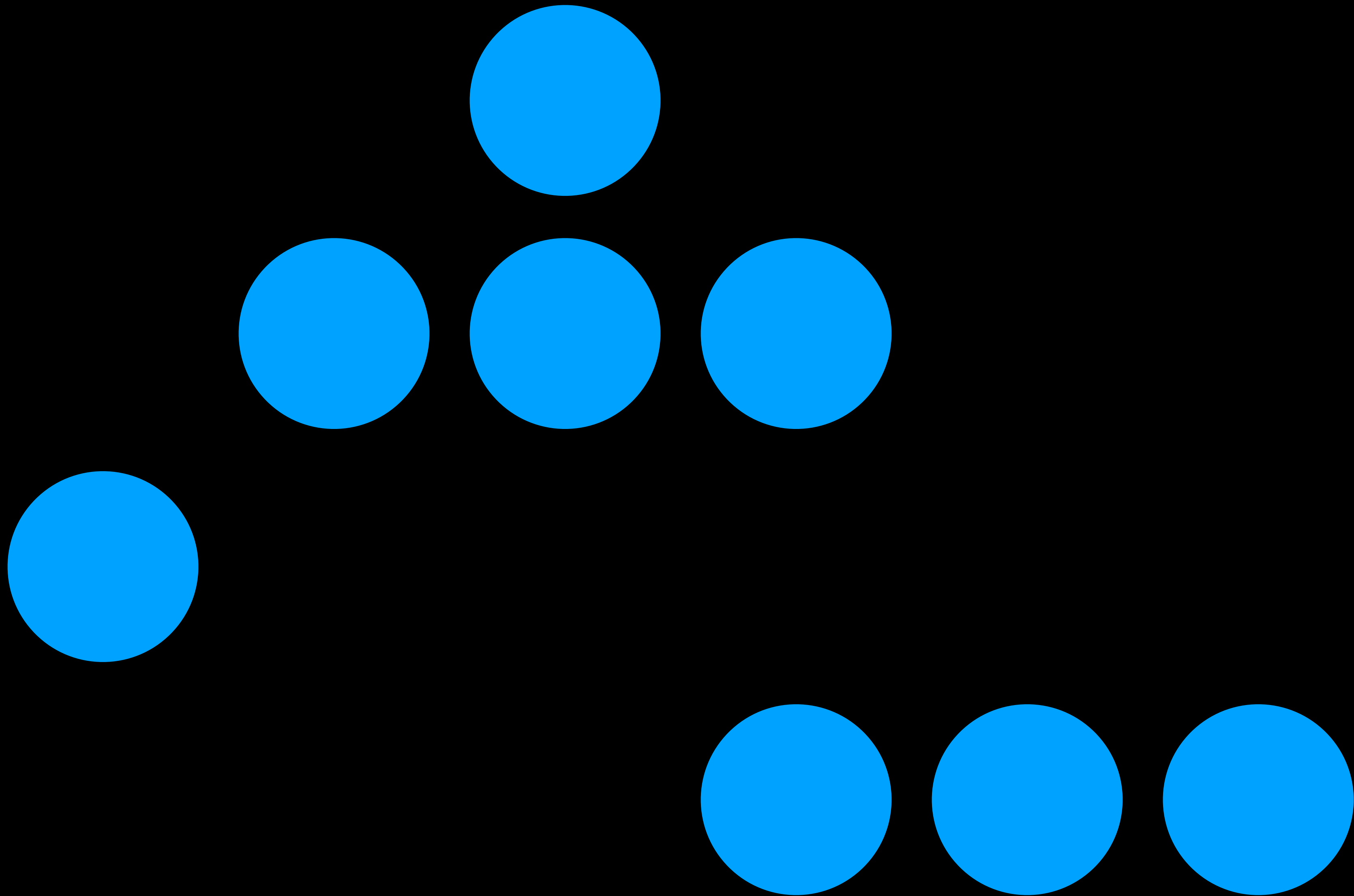
in greedy case, we always choose best move

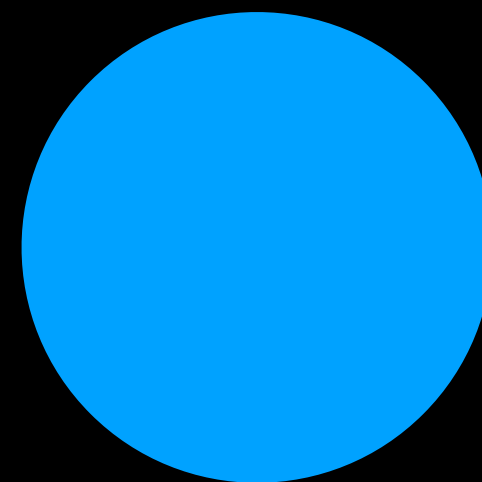
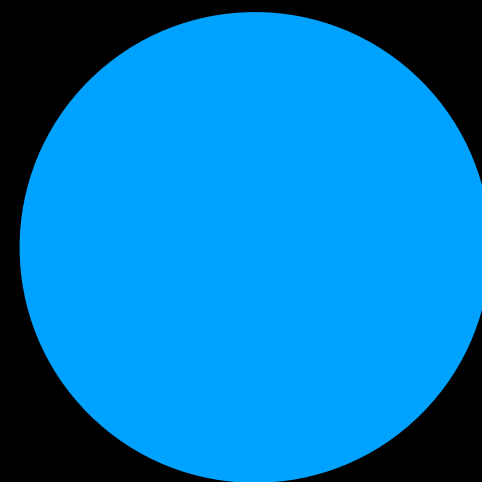
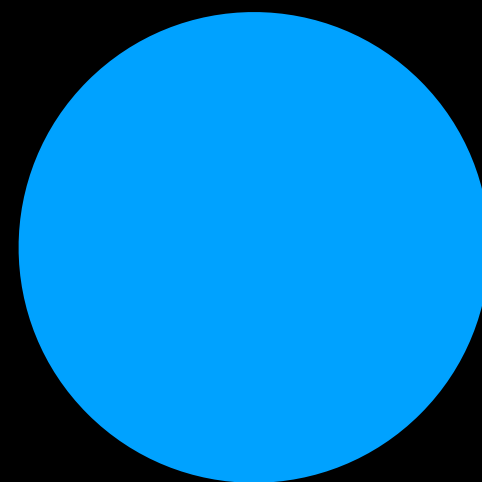
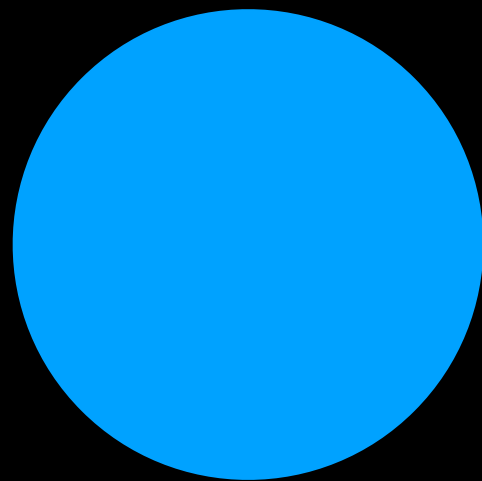
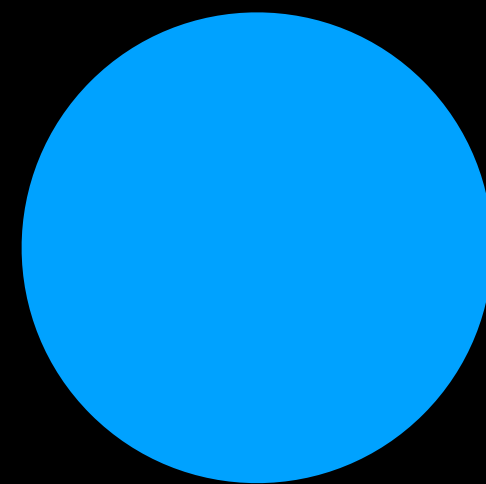
- Set ϵ equal to how often we want to move randomly.
- With probability $1 - \epsilon$, choose estimated best move.
- With probability ϵ , choose a random move.

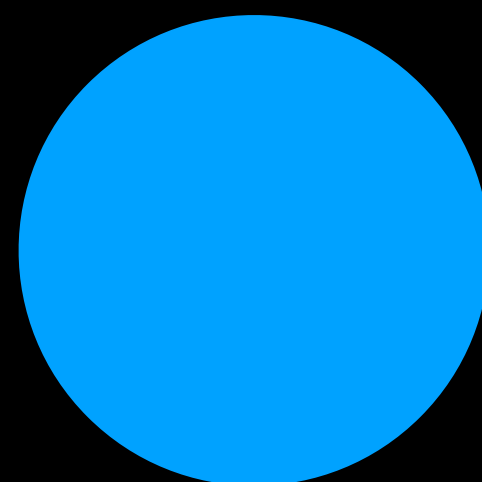
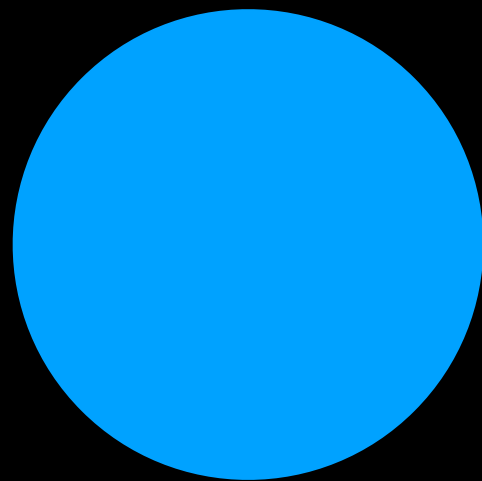
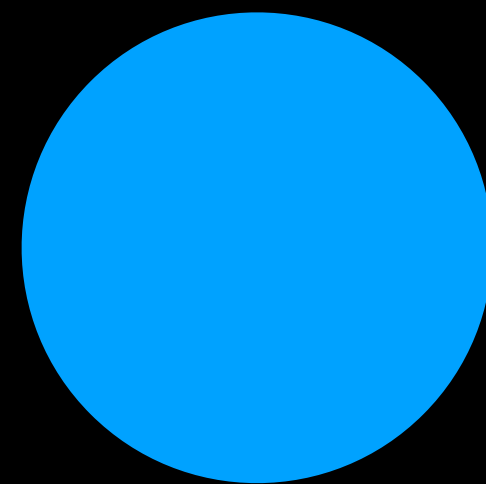
Nim

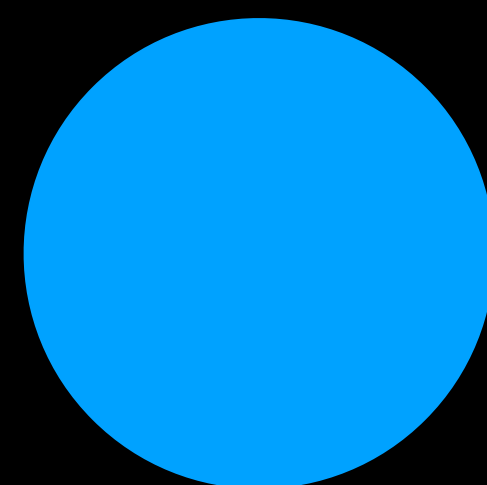
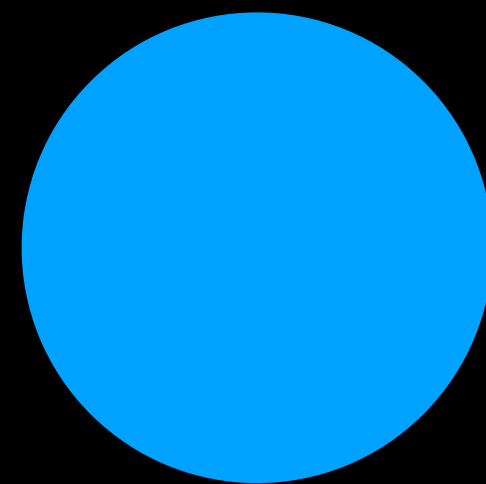


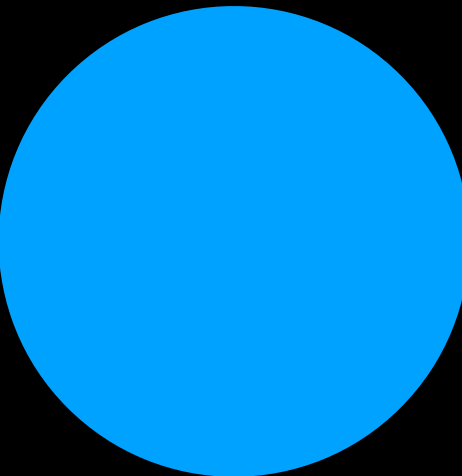












function approximation

approximating $Q(s, a)$, often by a function combining various features, rather than storing one value for every state-action pair

Unsupervised Learning

unsupervised learning

given input data without any additional feedback, learn patterns

Clustering

clustering

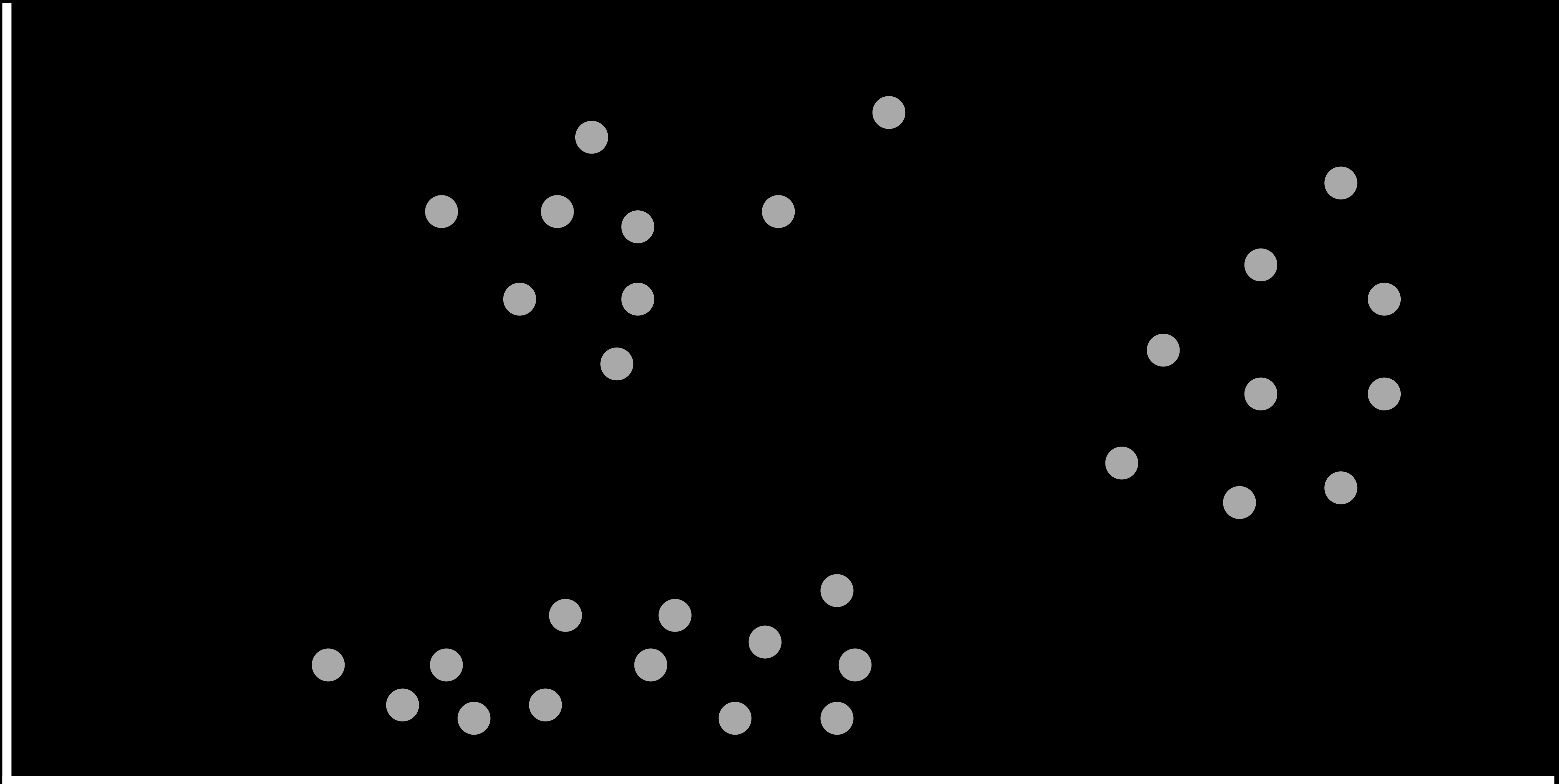
organizing a set of objects into groups in such a way that similar objects tend to be in the same group

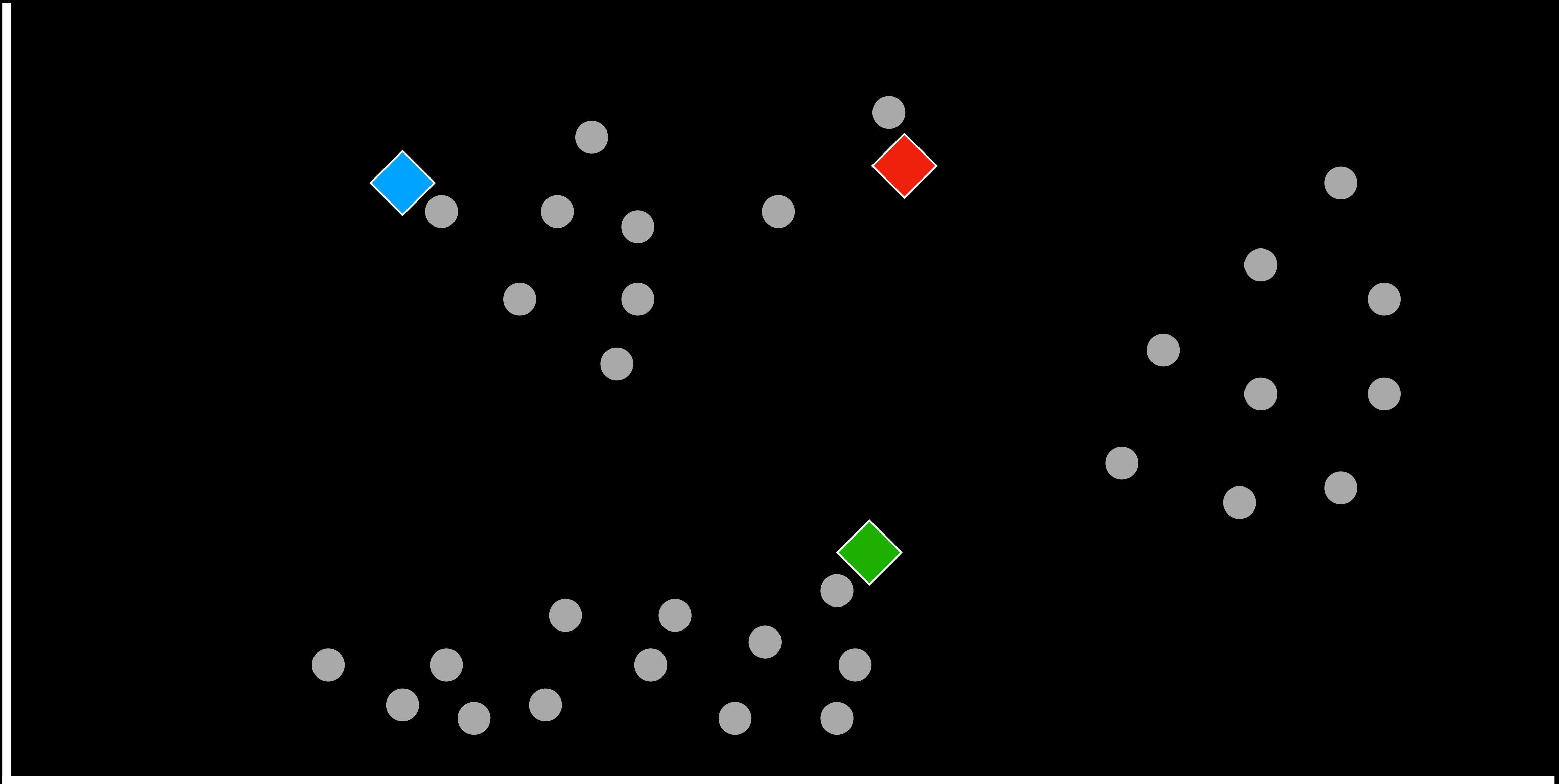
Some Clustering Applications

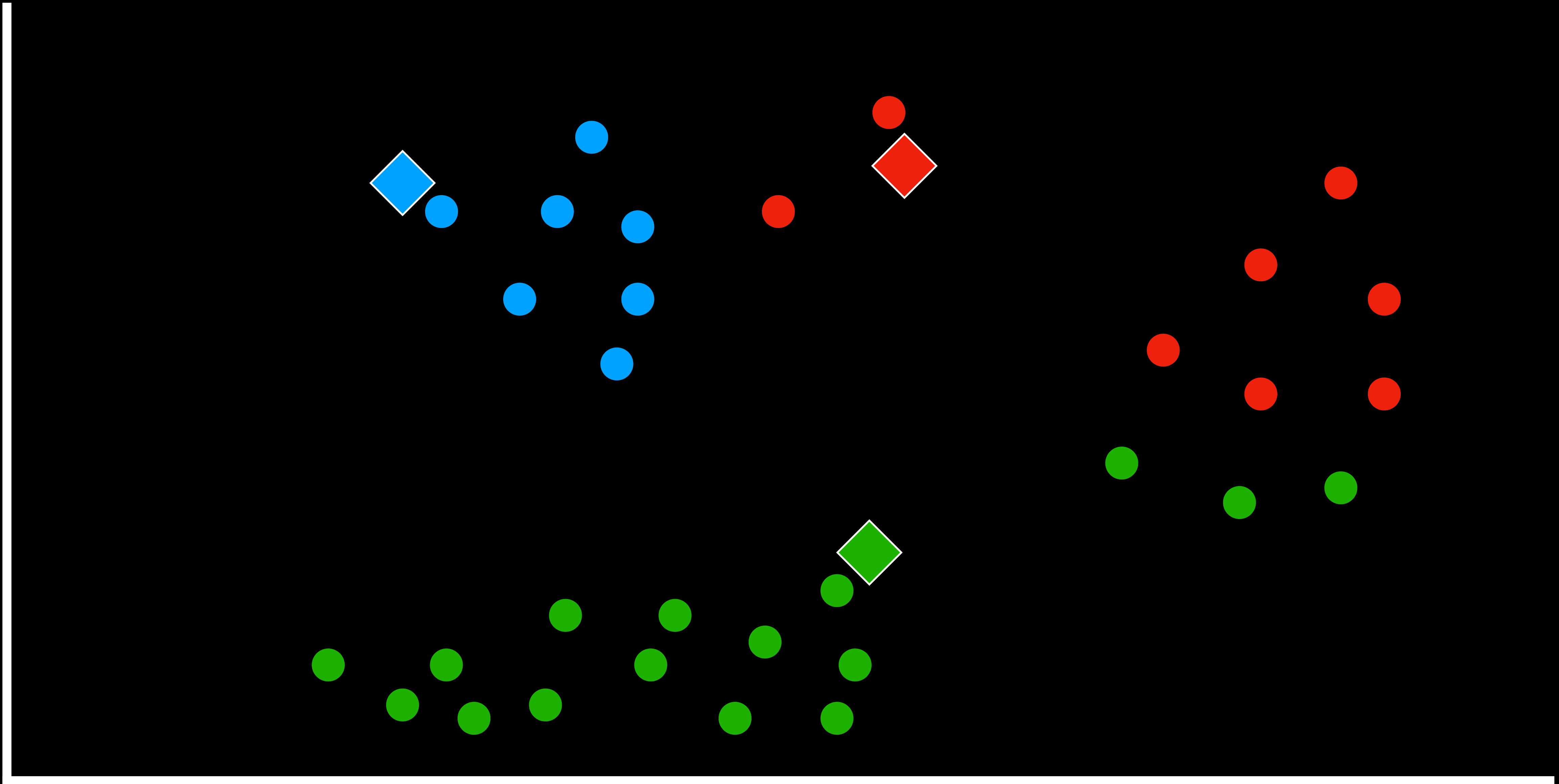
- Genetic research
- Image segmentation
- Market research
- Medical imaging
- Social network analysis.

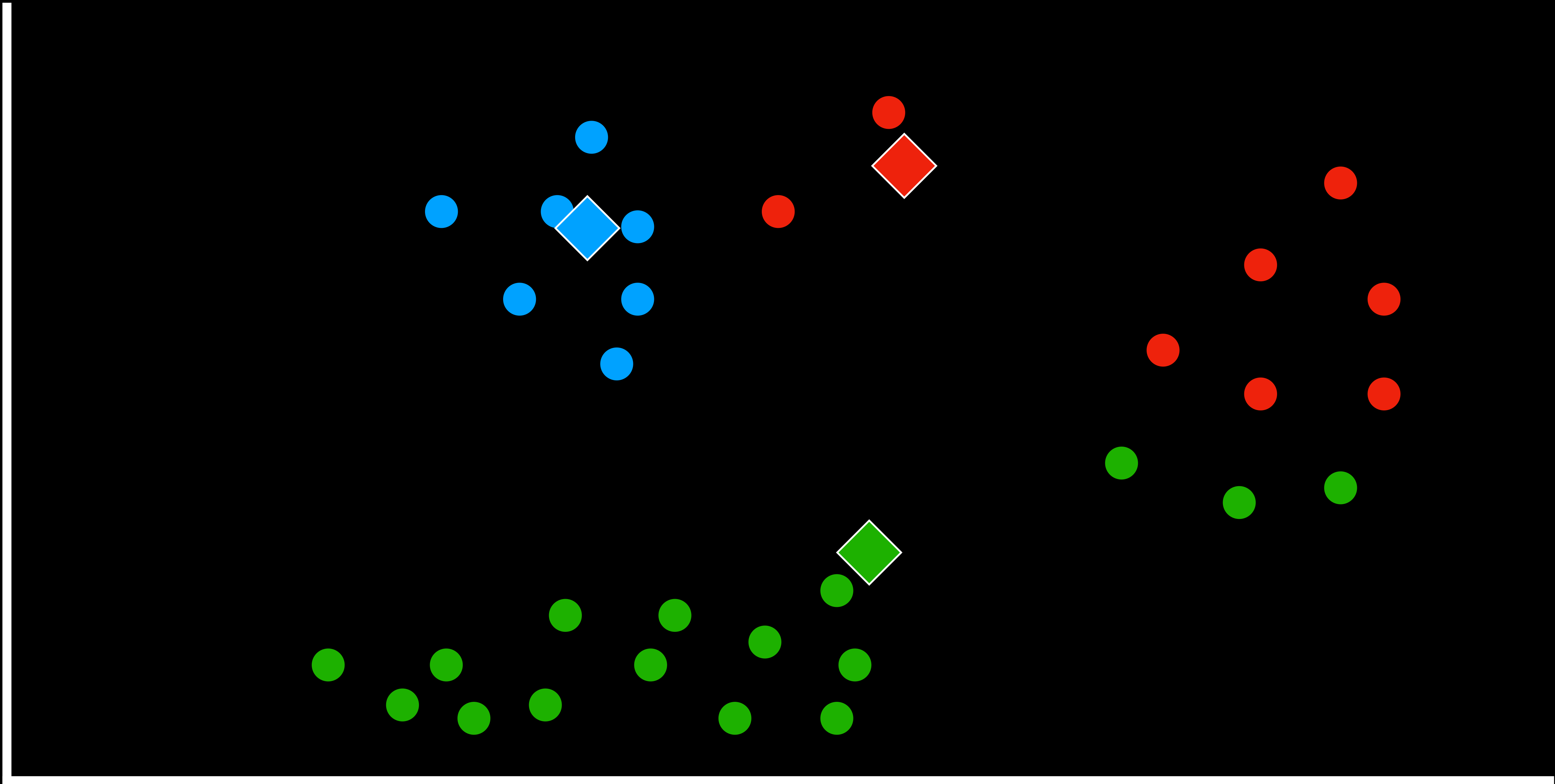
k-means clustering

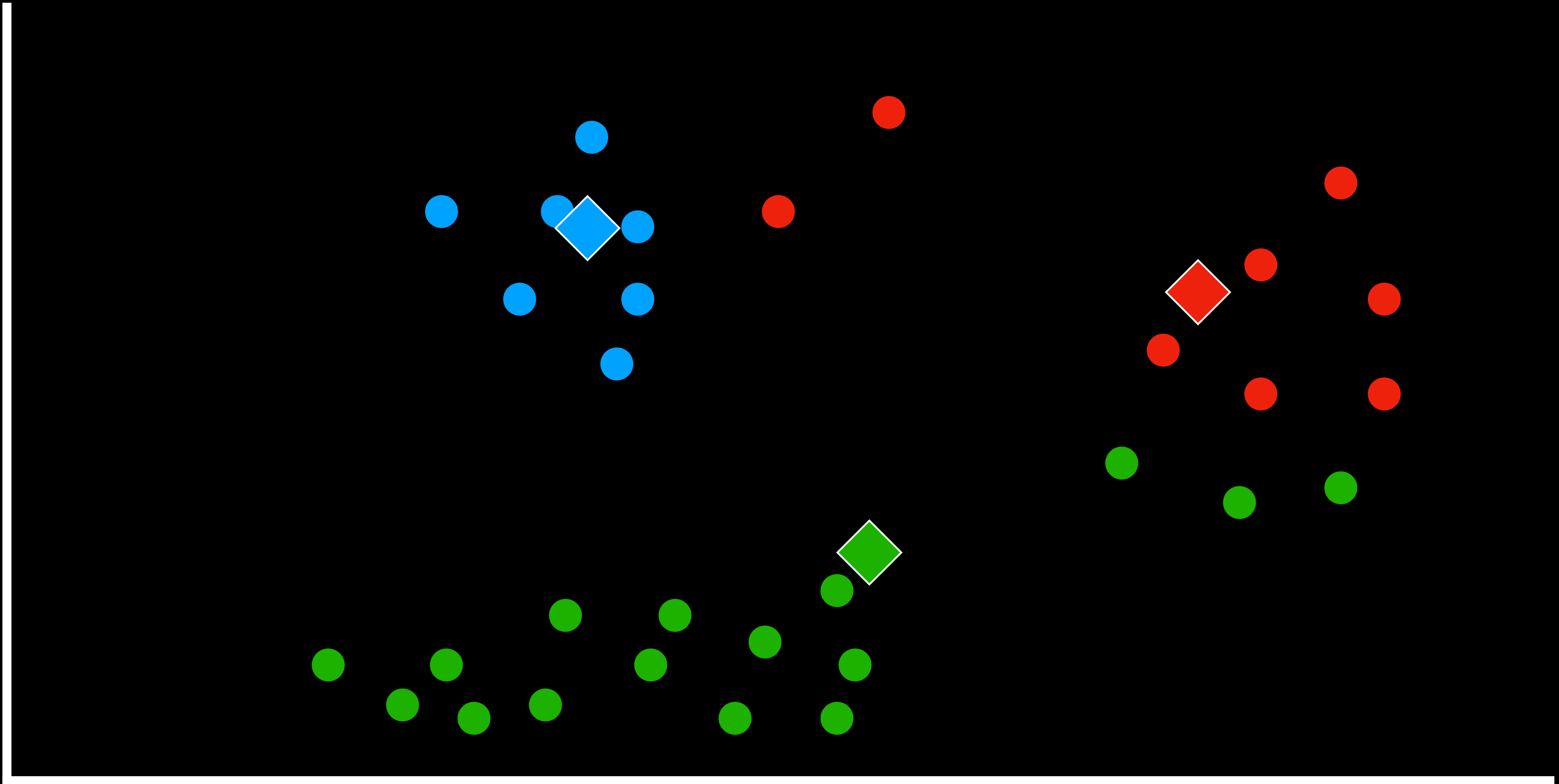
algorithm for clustering data based on repeatedly assigning points to clusters and updating those clusters' centers

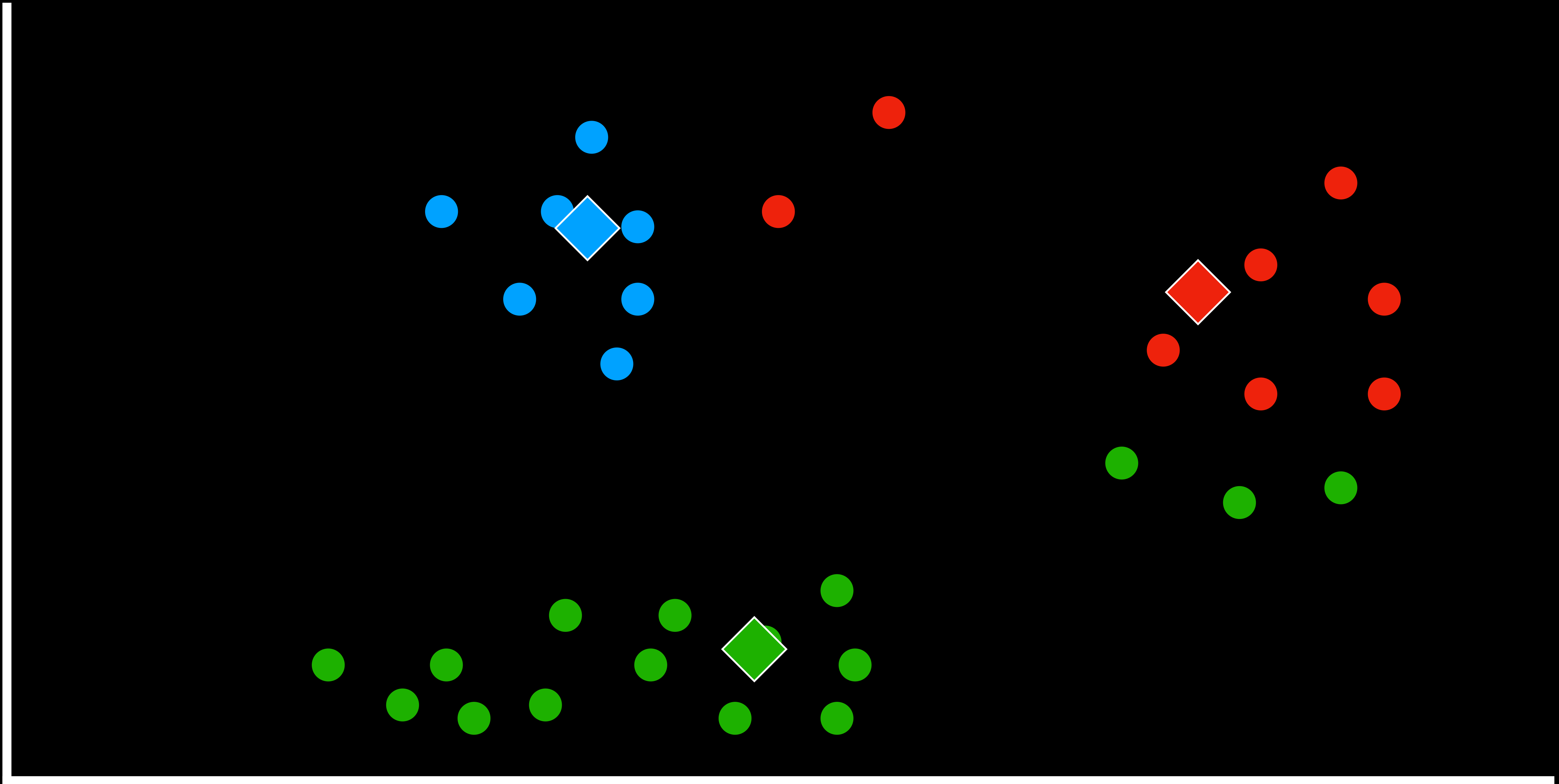


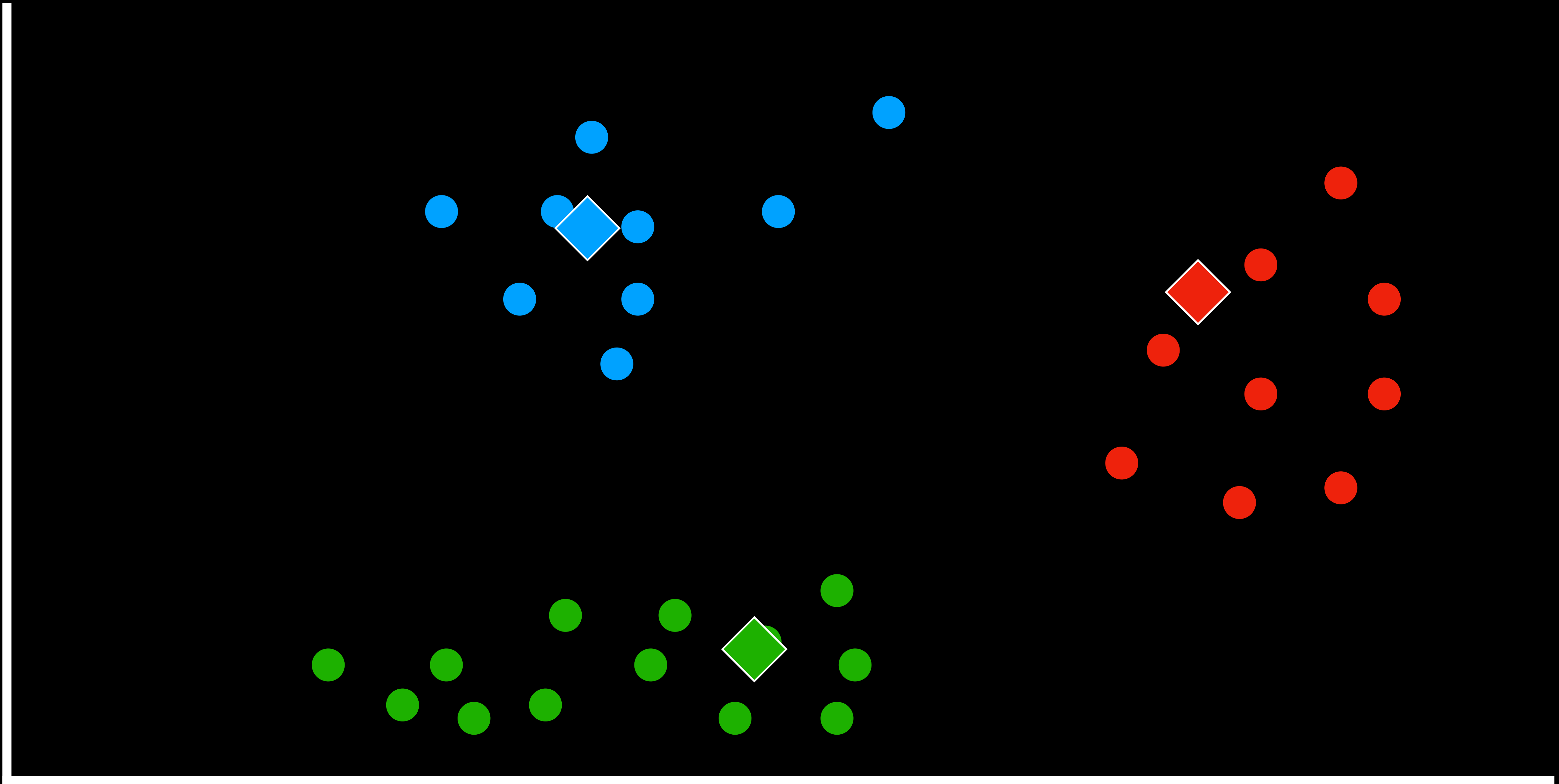


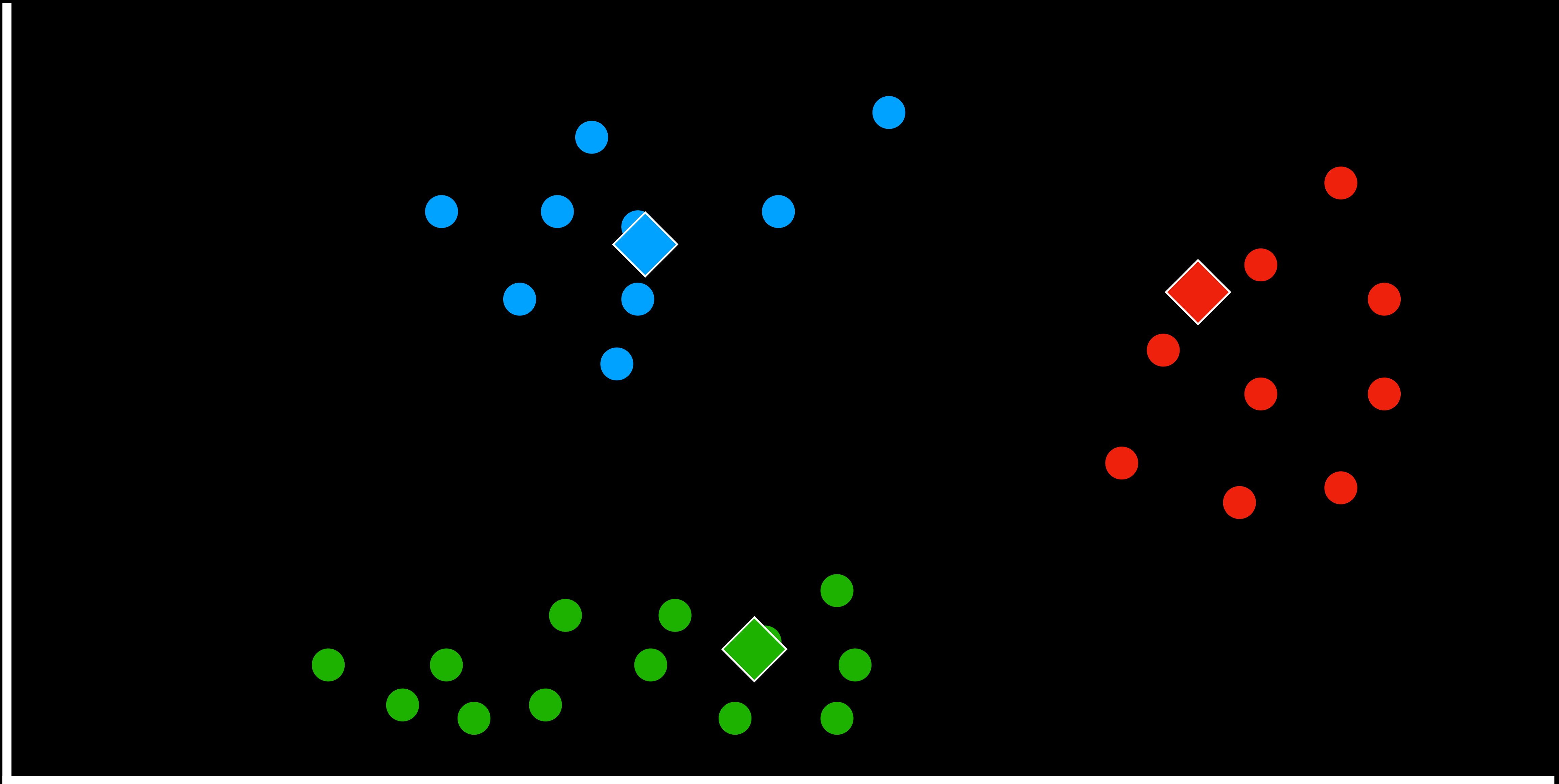


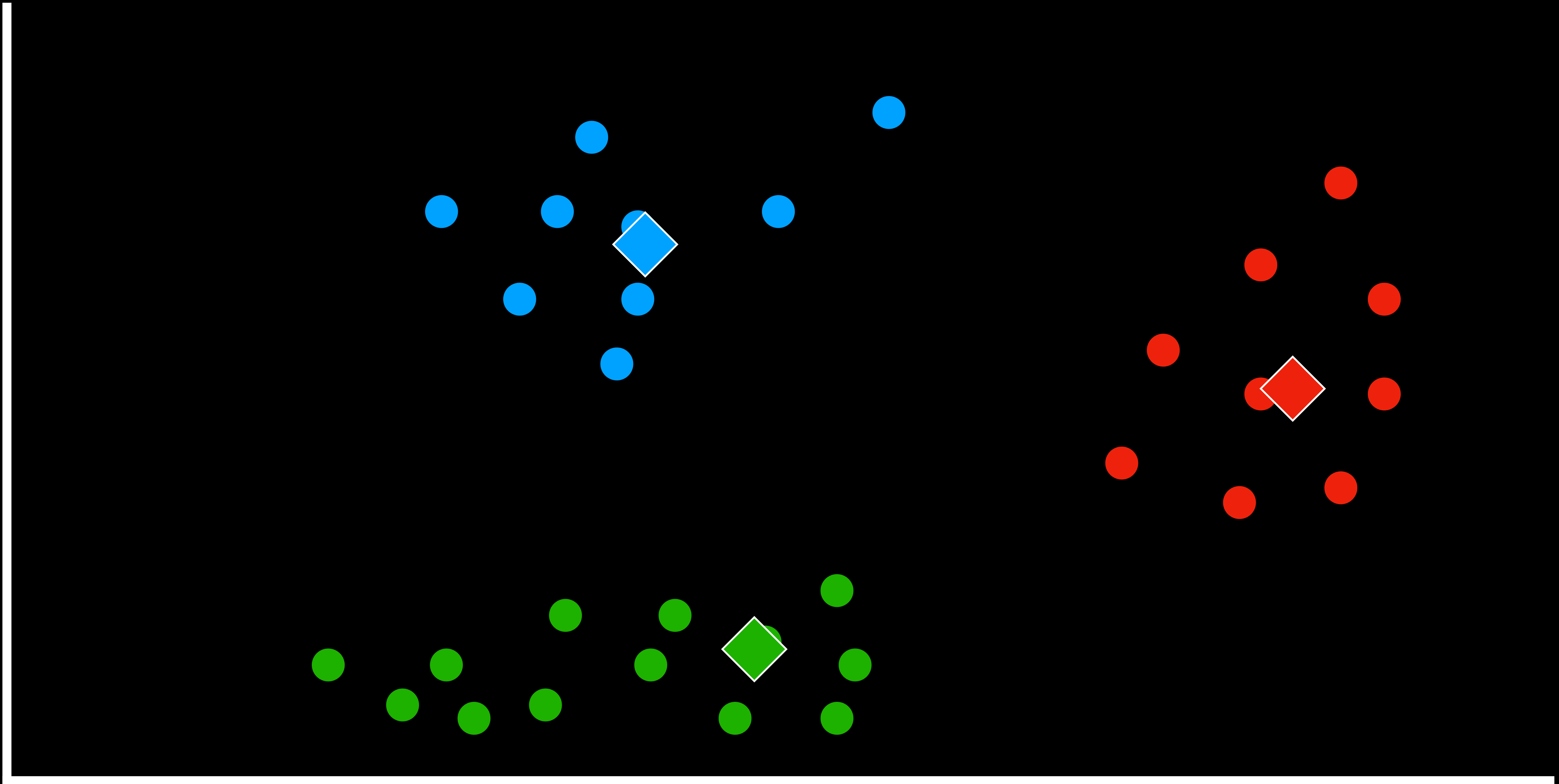


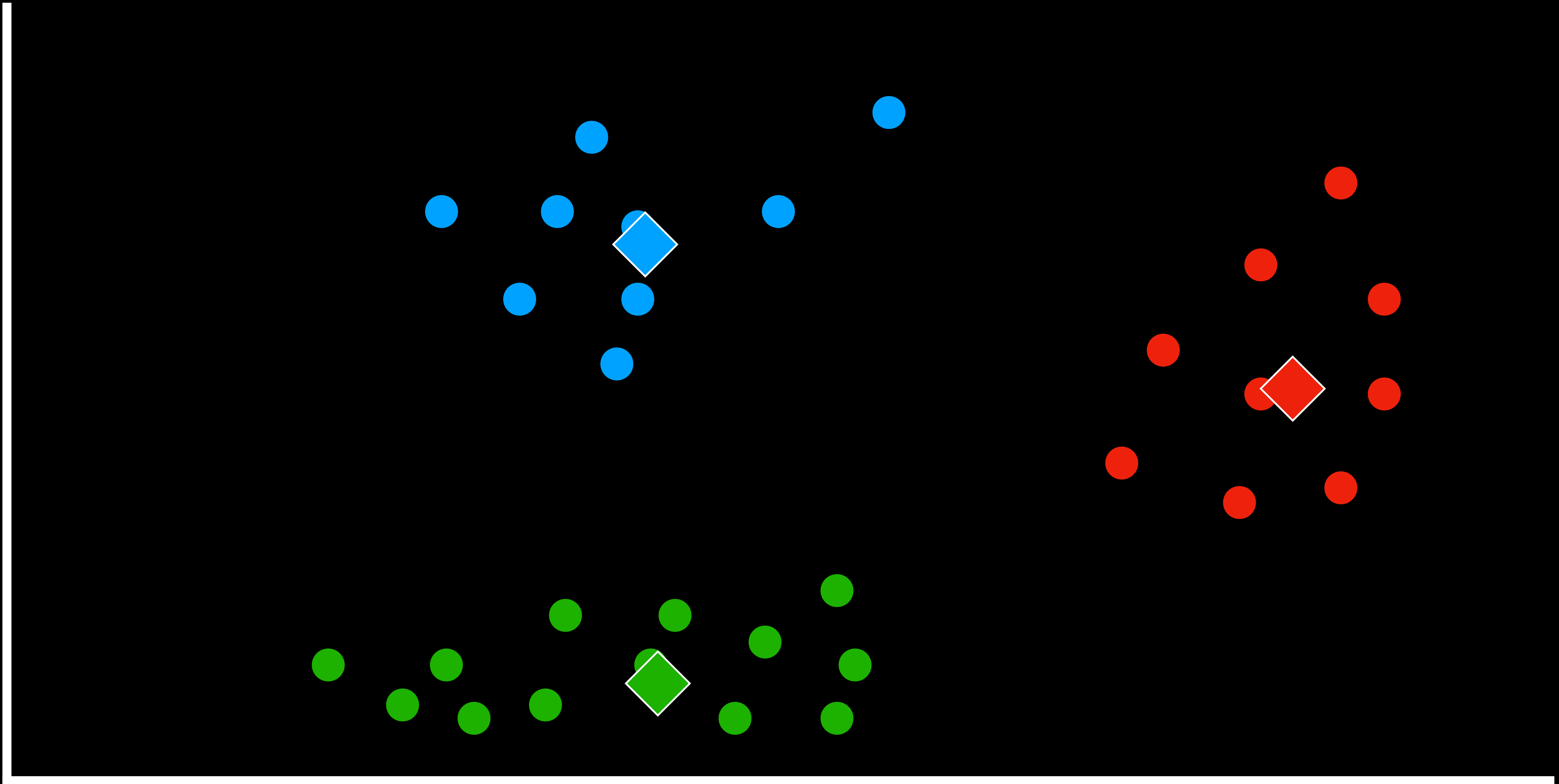


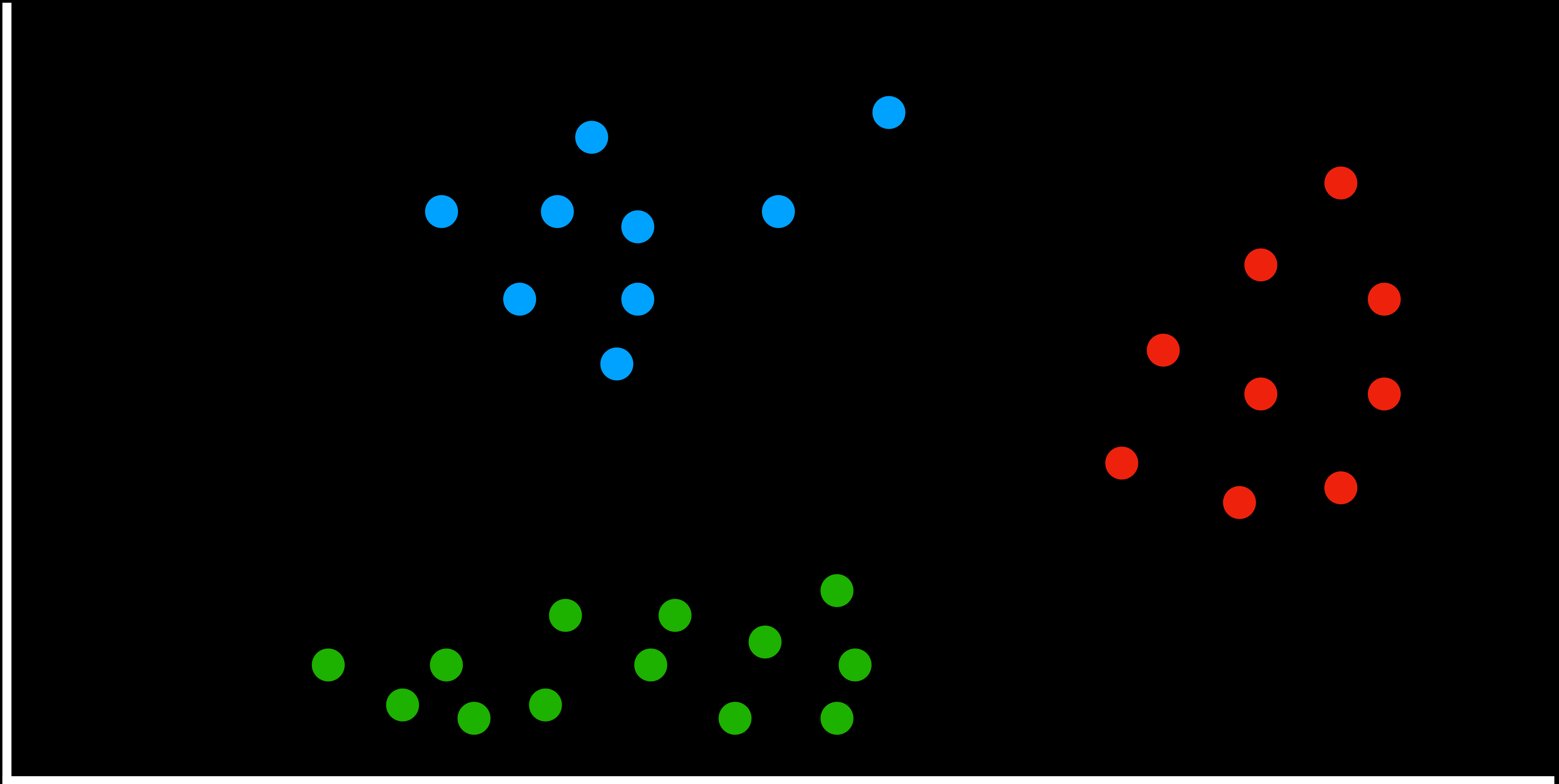












Learning

- Supervised Learning
- Reinforcement Learning
- Unsupervised Learning

Learning

Introduction to
Artificial Intelligence
with Python