

2025 - ISISS MARCO CASAGRANDE

NEURONAL DYNAMICS

Produced By
Saccon, Yasser, Rossetto.
Using Figma Slides.



Contenuti:

L'obiettivo della seguente presentazione è definire i processi alla base dell'attività di un singolo neurone e quelli inter-neuronali atti alla memoria associativa, per tracciare delle connessioni fra essi e le loro controparti al livello matematico-fisico e informatico.

Aspetti grafici ottenuti attraverso il seguente software:

INTRODUZIONE ALLA BIOLOGIA DEL NEURONE

LEAKY INTEGRATE-AND-FIRE MODEL

PROPRIETA NON LINEARI DEL NEURONE BIOLOGICO

MEMORIA BIOLOGICA ASSOCIATIVA

MODELLO HOPFIELD PER I SISTEMI NEURALI

INTRODUZIONE ALLA BIOLOGIA DEL NEURONE



TRIPARTIZIONE DEL NEURONE

DENDRITI:

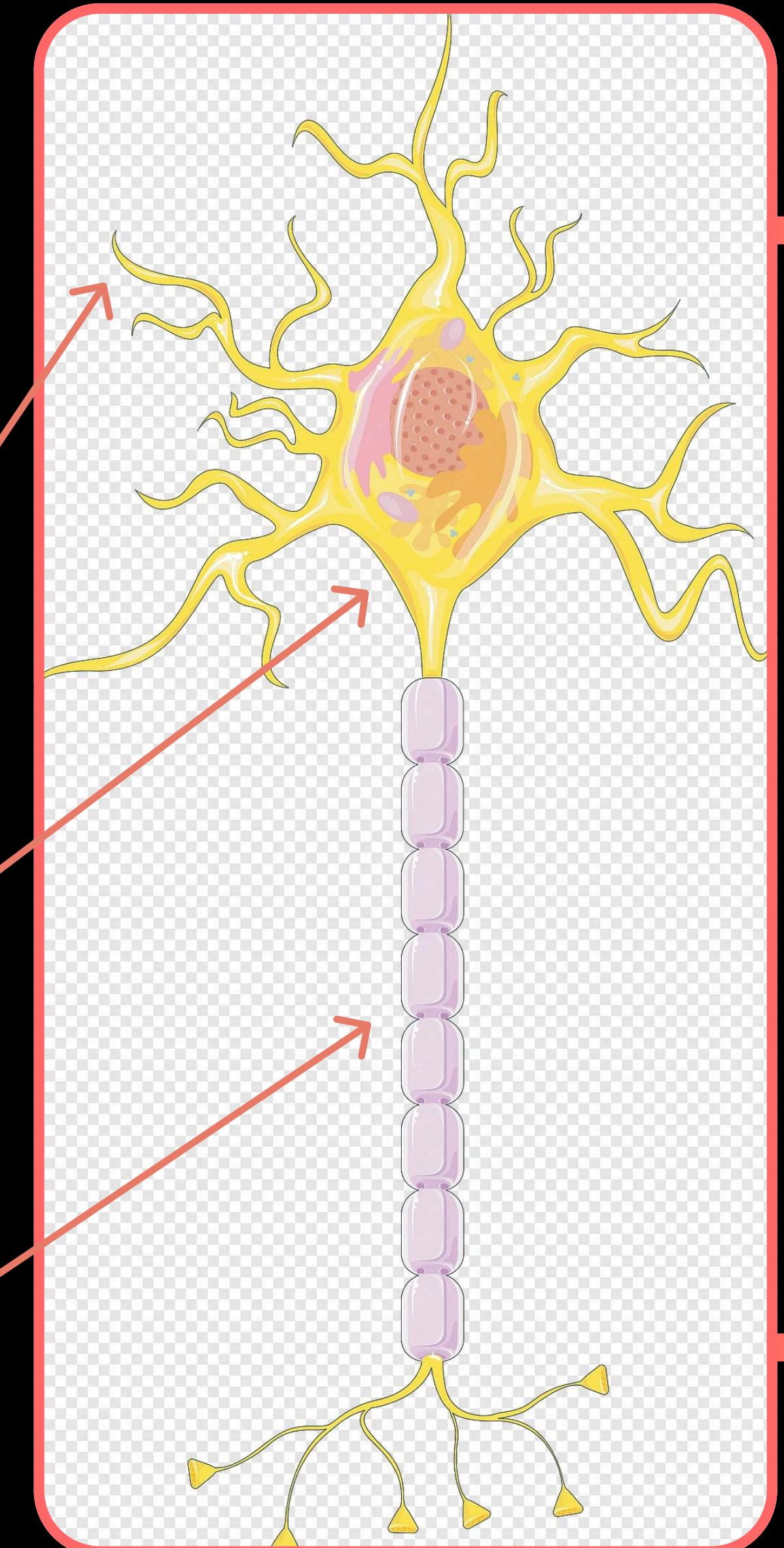
Estensioni morfologicamente variegate, ramificate sottili (0.2–5 µm). Convertono i segnali chimici sinaptici in variazioni locali del potenziale di membrana.

SOMA (Corpo Cellulare):

Regione centrale ove convergono e sono integrate le informazioni provenienti dai dendriti. Contiene un nucleo circondato da organelli specializzati.

ASSONE:

Particolare, lunga estensione del Soma (→1m). L'impulso elaborato dal Soma lo attraversa con maggiore rapidità grazie ai Nodi di Ranvier.



Il sito aveva promesso un'immagine a sfondo trasparente ma ci ha ingannato. Abbiamo deciso di tenerla per ricordo.

TRIPARTIZIONE DELLA SINAPSI

ELEMENTO SINAPTICO:

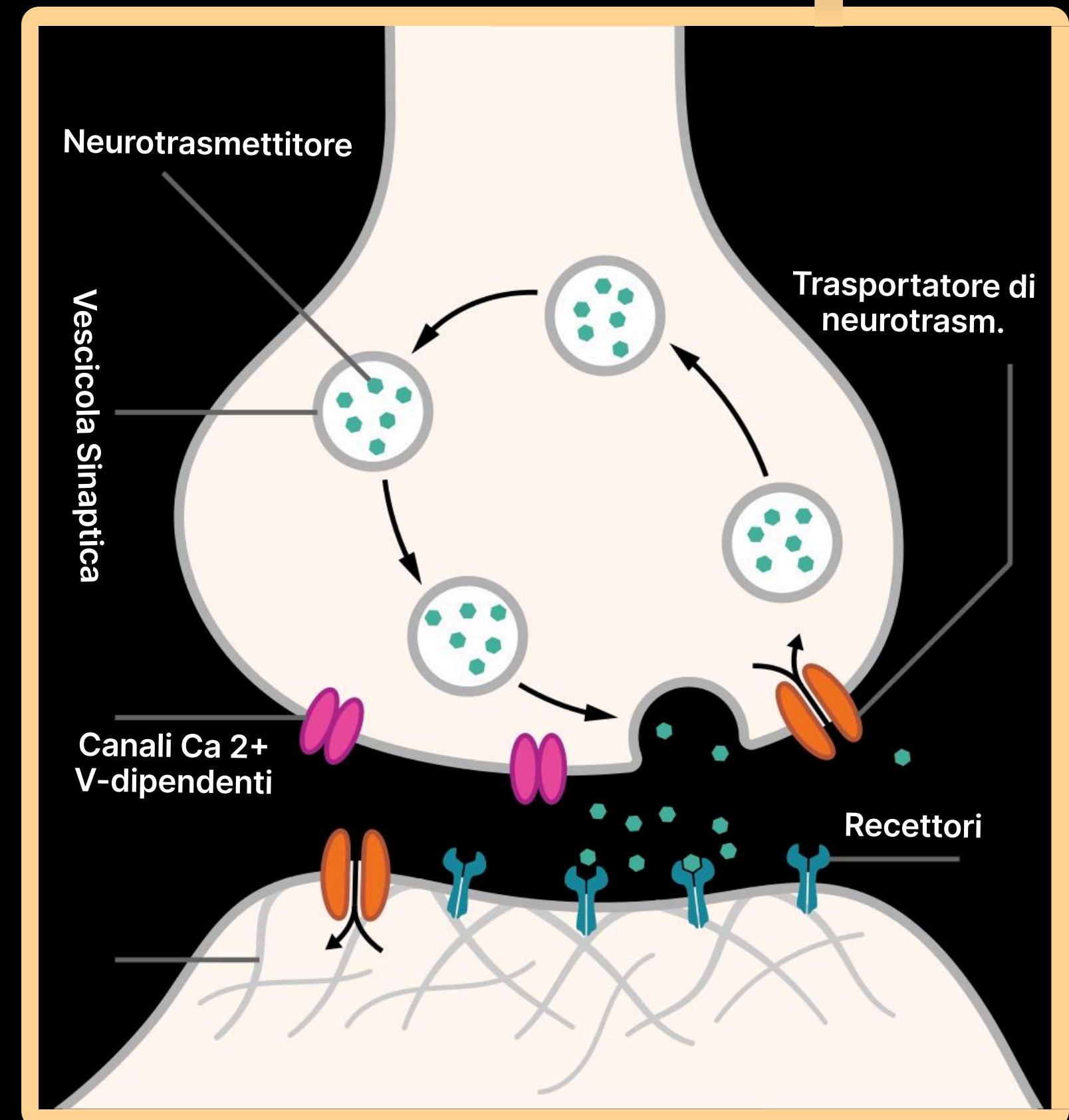
Estremità terminale dell'assone; Presenza di vescicole contenenti Neurotrasmettitori da rilasciate nello spazio sinaptico.

SPAZIO SINAPTICO:

Anche detto "Fessura sinaptica" (30nm). I neurotrasmettori lo attraversano per diffusione.

ELEMENTO POSTSINAPTICO:

Estremità del dendrite ricevenete. Cosparsa di Recettori per i Neurotrasmettitori (canali neurotras.-dipendenti).



NEUROTRASMETTITORI

Neurotrasmettore = molecola che consente la trasmissione di segnali chimici tra le cellule nervose.

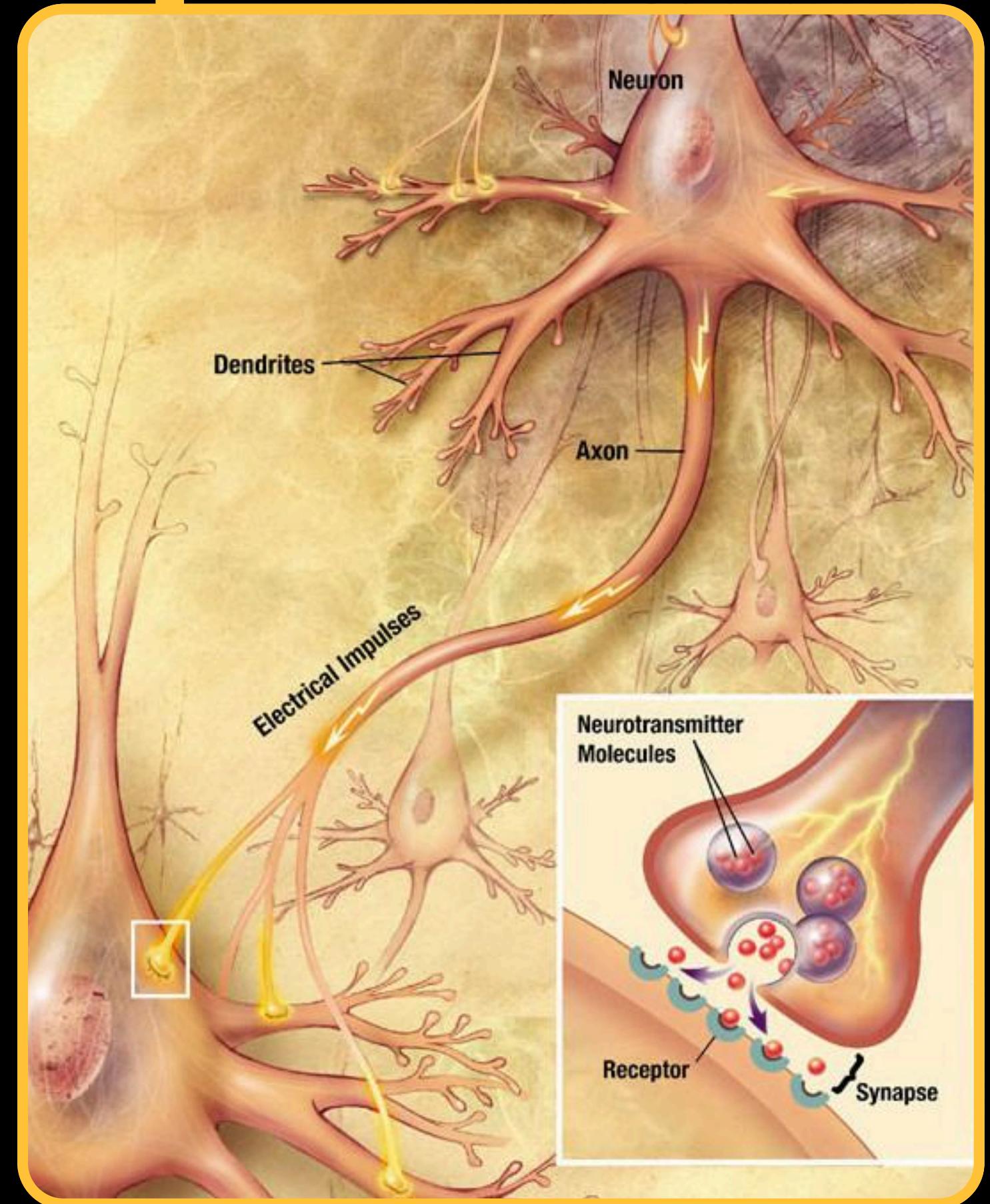
Sintetizzazione: nel citosol del neurone presinaptico.

Immagazzinamento: nelle vescicole dell'elemento presinaptico.

Rilascio: nello spazio sinaptico. Ingresso di ioni Ca^+ → → fusione delle vescicole con la membrana cellulare.

Scopo: stimolazione dei recettori specifici presenti nell'elemento postsinaptico.

Ricaptazione: Riassorbimento dall'elemento presinaptico per essere riutilizzati o smaltiti.



PROPAGAZIONE DEL POTENZIALE

1. FASE A RIPOSO:

V_{rest} di membrana: -70 mV;

Il potenziale è mantenuto costante dal lavoro delle “pompe sodio potassio” (canali attivi K^+ , Na^+), che ad ogni ciclo espellono 3 Na^+ e immettono 2 K^+ nella membrana cellulare.

2. DEPOLARIZZAZIONE:

ΔV raggiunge -55mV canali voltaggio dipendenti Sodio si aprono → ingresso di grandi quantità di Na^+ .

ΔV “aumenta” fino a +30 mV.



PROPAGAZIONE DEL POTENZIALE

3. RIPOLARIZZAZIONE:

Chiusura canali Na⁺ (inutilizzabili per Δt). Apertura canali K⁺ per ripolarizzare la membrana → Drastica ripolarizzazione.

4. IPERPOLARIZZAZIONE:

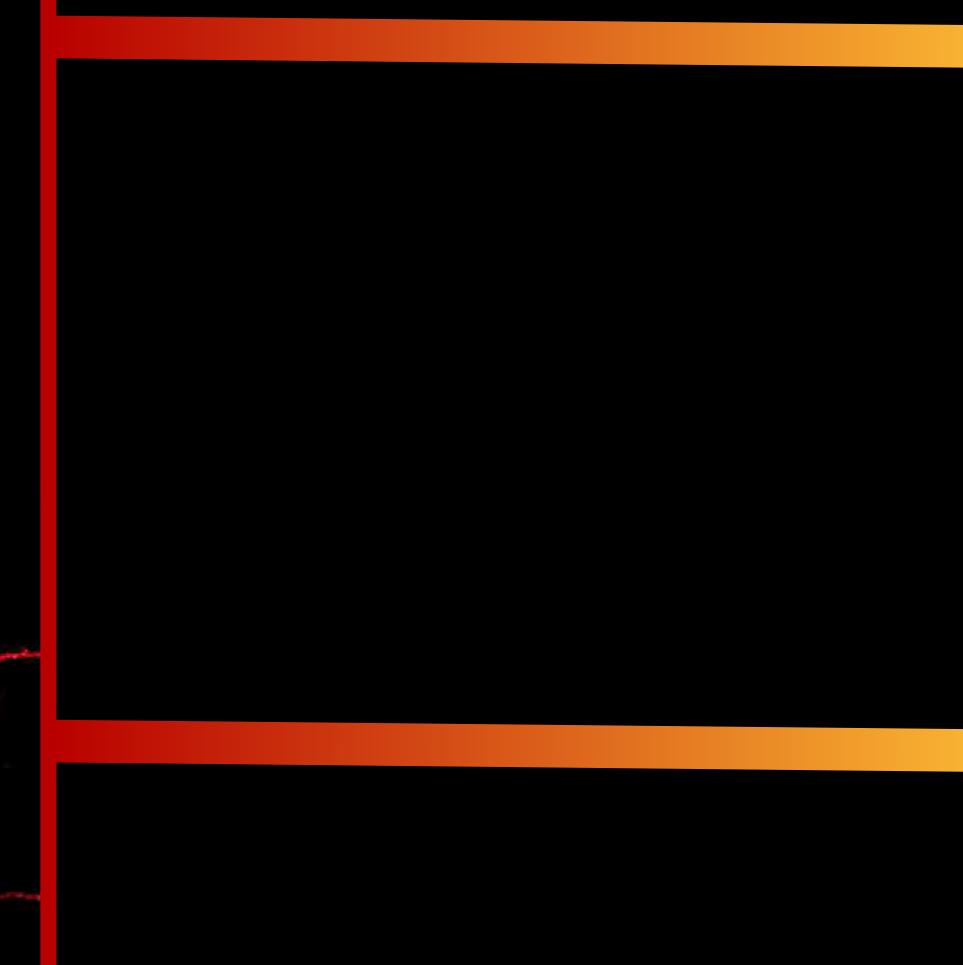
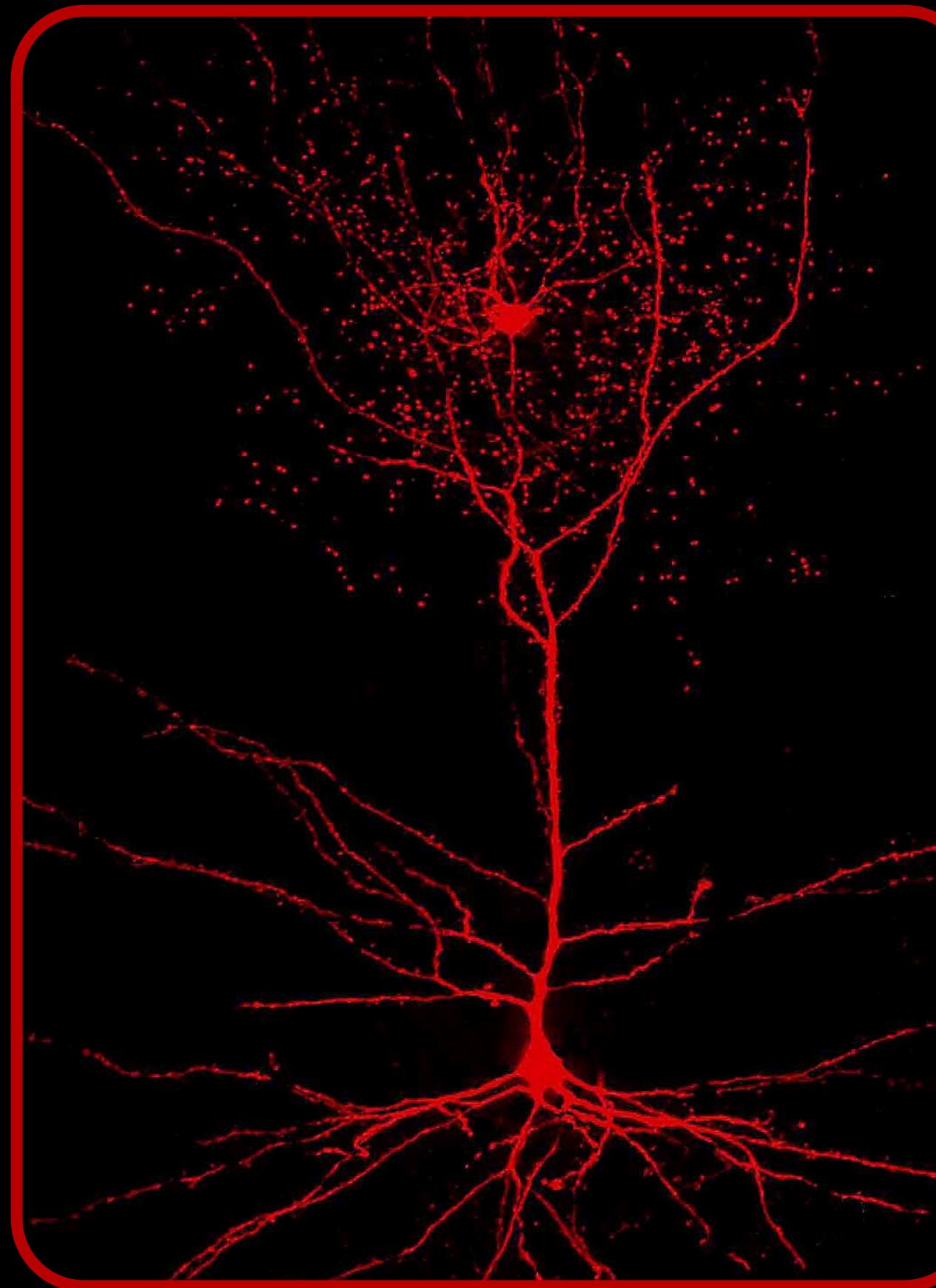
Canali K⁺ rimangono aperti → V discende sotto Vrest (-80mV)

5. PERIODO REFRAKTARIO:

Δt durante cui il neurone è “impermeabile” all’impulso.
Termina quando le pompe Na-K ripristinano le rispettive concentrazioni. Tale periodo è essenziale per la direzione di propagazione dell’output.



*LEAKY INTEGRATE-
AND-FIRE MODEL*

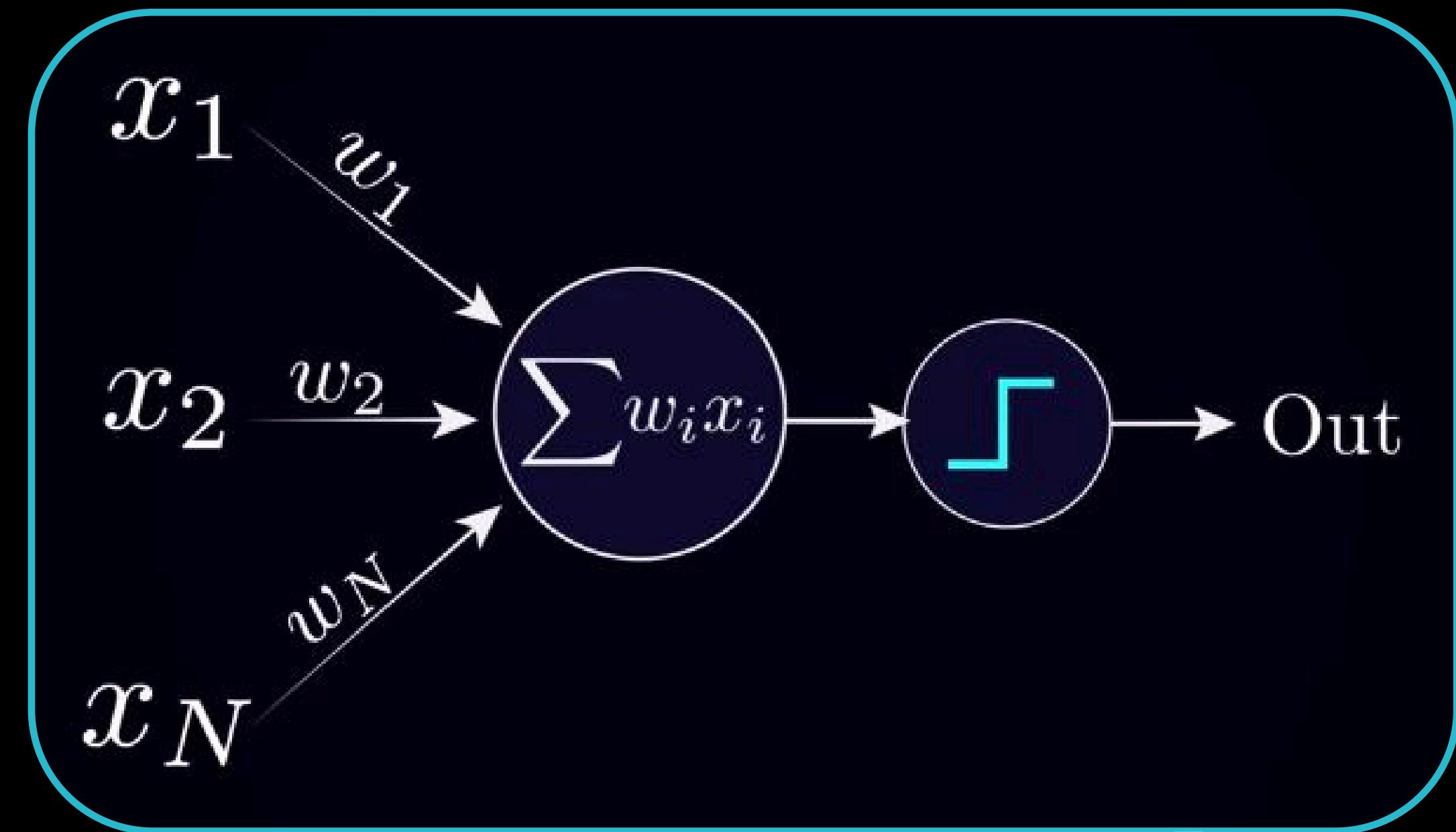


IL PERCETTRONE

In machine learning è definito come:
“un algoritmo per il learning
supervisionato di classificatori binari”.

w = **weight** della connessione

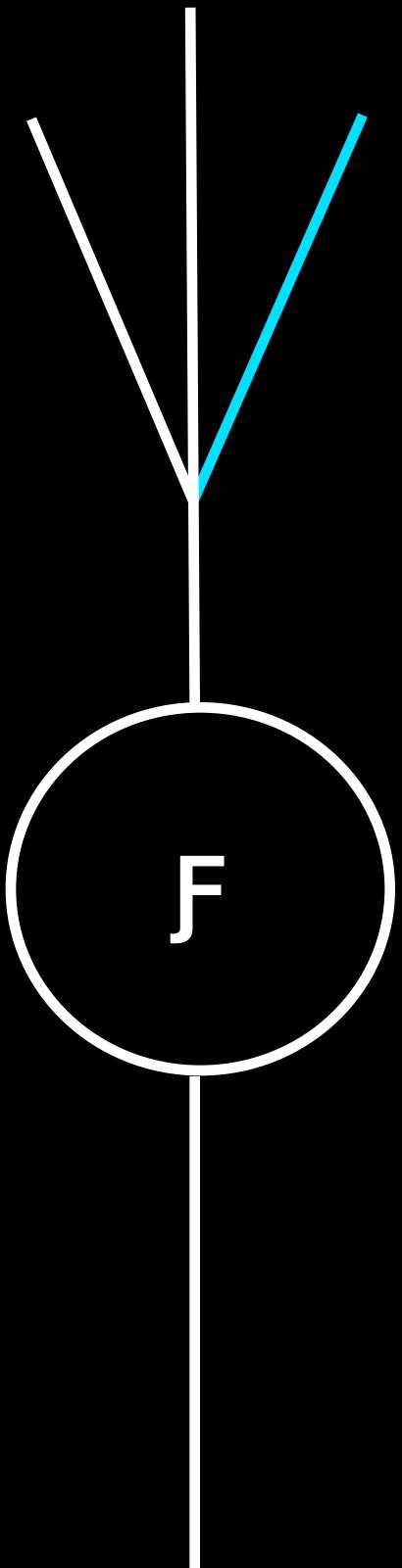
x = **state** della connessione



$$f \begin{cases} 1 & \text{if } h > \gamma \\ 0 & \text{if } h < \gamma \end{cases}$$

$$h = w_1 x_1 + w_2 x_2 + \dots + w_n x_n$$

CABLE THEORY (*Wilfred Rall*)



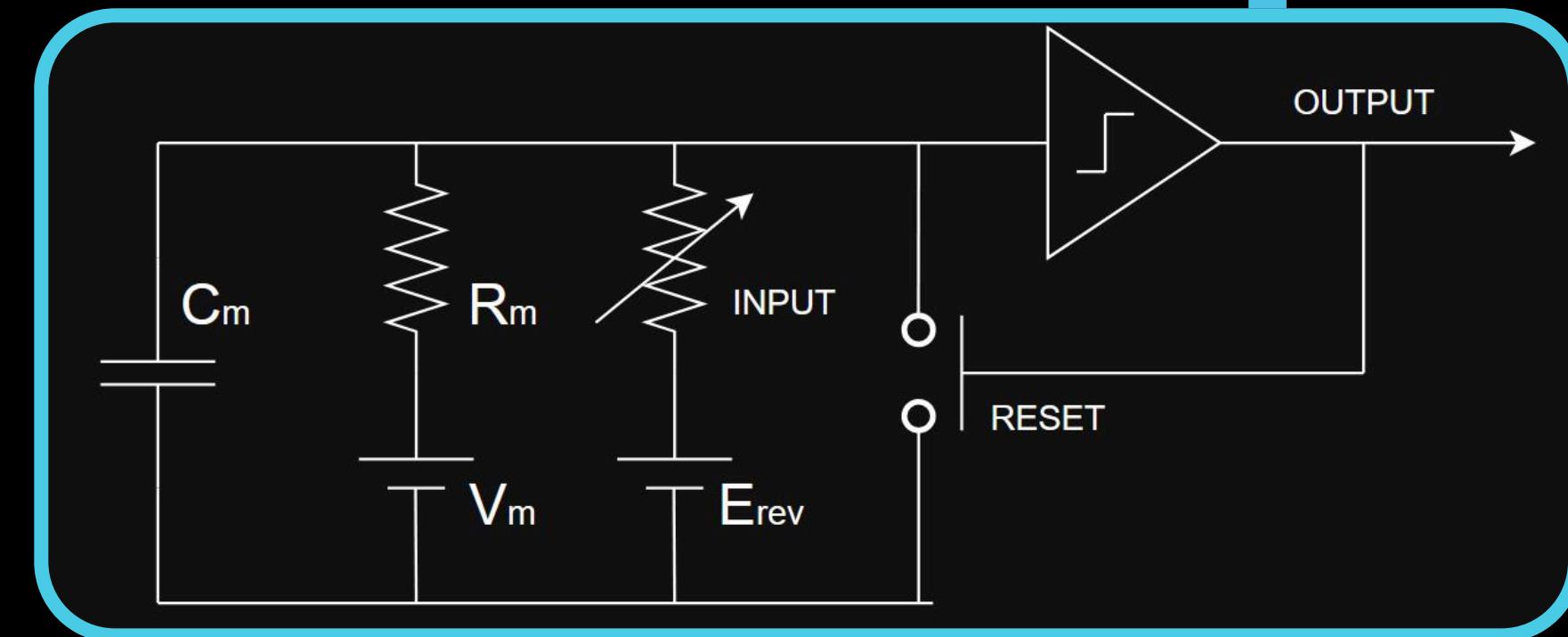
$V > \gamma_r \rightarrow \text{Spike}$

$V < V_{rest} \rightarrow \text{KIR (Potassium Inward Rectifier)}$

Dendrite = Cavo Leaky imperfetto.

“Leaky” in quanto V varia con Δx lungo di esso.

“Imperfetto” in quanto le sue caratteristiche fisiche variano lungo di esso.

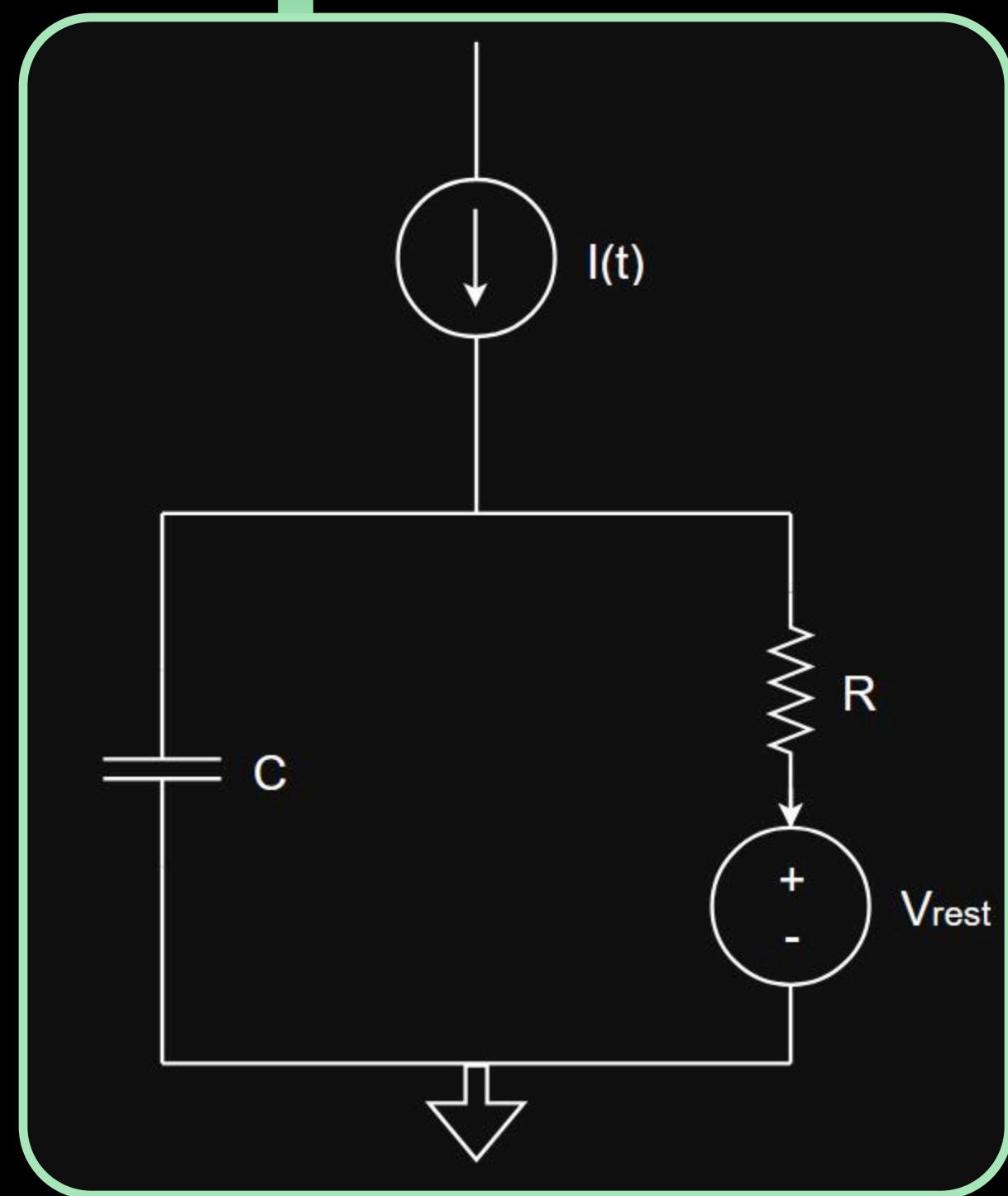


LEAKY INTEGRATE-AND-FIRE

Abbreviato come LIF, è il modello meno complesso per descrivere la variazione del potenziale di membrana nel tempo. In particolare può essere descritta attraverso la seguente equazione differenziale del primo ordine:

$$\tau \cdot \frac{dV}{dt} = V_{rest} - V + R \cdot I(t)$$

E' necessario precisare che il LIF descrive in maniera accurata il potenziale di membrana solo fino all'emissione della spike. In base a determinate condizioni si otterranno soluzioni diverse; Esaminiamo i diversi casi...



LEAKY INTEGRATE-AND-FIRE

$$\tau \cdot \frac{dV}{dt} = V_{rest} - V + R \cdot I(t)$$

Uno dei casi più semplici è: $I(t)=0$: ovvero non arriva alcun impulso al neurone.

In questo caso, il potenziale di membrana del neurone tenderà a raggiungere asintoticamente il potenziale di membrana a riposo, qualsiasi sia il potenziale di membrana all'istante iniziale.

$$\begin{cases} I(t) = 0 \\ V(t) = e^{-\frac{t}{\tau}} + V_{rest} \end{cases}$$

LEAKY INTEGRATE-AND-FIRE

$$\tau \cdot \frac{dV}{dt} = V_{rest} - V + R \cdot I(t)$$

Nel caso di un singolo impulso a corrente costante si ha che il potenziale di membrana tenderà a raggiungere asintoticamente $V_{rest} + R \cdot I$, secondo la funzione:

$$\begin{cases} I(t) = K \\ V(t) = V_{rest} + R \cdot K \left(1 - e^{-\frac{t}{\tau}}\right) \end{cases}$$

LEAKY INTEGRATE-AND-FIRE

$$\tau \cdot \frac{dV}{dt} = V_{rest} - V + R \cdot I(t)$$

Infine nel caso di una corrente arbitraria e quindi descritta da una funzione, il potenziale di membrana varierà secondo la funzione:

$$V(t) = V_{rest} + \int_{-\infty}^t \frac{1}{C} \cdot e^{-\frac{(t-t')}{\tau}} \cdot I(t') dt'$$

CODING A LIF MODEL - *LIF_Model.py*

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

class LIF_Neuron:
    def __init__(self, current_function=None, v_init=-56, v_rest=-65.0, R=10.0, threshold=-55.0, tau=10.0, refract_time=5.0, dt=0.1):
        self.v_rest = v_rest
        self.v = v_init
        self.tau = tau
        self.R = R
        self.dt = dt
        self.threshold = threshold
        self.refract_time = refract_time

        if current_function is None:
            self.current_function= lambda x:0
        else:
            self.current_function=current_function

        self.refract_counter = 0
        self.spikes = []
```

Costruttore dell'oggetto Neuron. Parametri importanti con valori di default:

- *una funzione che descrive la corrente in input*
- *il voltaggio di riposo*
- *la resistenza di membrana del neurone*
- *il tempo refrattario*

CODING A LIF MODEL - *LIF_Model.py*

```
def get_input_current(self, t):
    return self.current_function(t)

def reset(self):
    self.v = self.v_rest
    self.refract_counter = 0
    self.spikes.clear()

def step(self, t):
    if self.refract_counter > 0:
        self.refract_counter -= 1
    return self.v_rest

    I_t = self.get_input_current(t)
    dv = (-self.v - self.v_rest) + self.R * I_t * (self.dt / self.tau)
    self.v += dv

    if self.v >= self.threshold:
        self.v = self.v_rest
        self.refract_counter = int(self.refract_time / self.dt)
        self.spikes.append(t)

    return self.v

def simulate(self, T):
    time_points = np.arange(0, T, self.dt)
    voltages = [self.step(t) for t in time_points]
    return time_points, voltages, self.spikes
```

Metodi del oggetto:

get_input_current: restituisce la corrente calcolata al tempo t .

step:
restituisce il potenziale d'azione secondo l'equazione differenziale secondo il modello LIF

simulate: restituisce gli insiemi che andano a costituire le coordinate dei punti x, y del grafico

CODING A LIF MODEL - Main.py

```
def dynamic_plot(T_total:int, lif:LIF_Neuron):
    time, voltages, spikes = lif.simulate(T_total)
    fig, ax = plt.subplots(figsize=(10, 5))
    ax.set_xlim(0, T_total)
    ax.set_ylim(min(voltages) - 5, max(voltages) + 5)
    line, = ax.plot([], [], lw=2)
    threshold_line = ax.axhline(y=lif.threshold, color='gray', linestyle='--')
    scatter = ax.scatter([], [], color='red', marker='x')

    def init():
        line.set_data([], [])
        scatter.set_offsets(np.empty((0, 2)))
        return line, scatter

    def update(frame):
        x = time[:frame]
        y = voltages[:frame]
        line.set_data(x, y)
        .
        spike_pts = np.array([[t, lif.threshold] for t in spikes if t <= time[frame - 1]])
        if len(spike_pts) > 0:
            scatter.set_offsets(spike_pts)
        return line, scatter

    ani = animation.FuncAnimation(fig, update, frames=len(time), init_func=init,
                                  blit=True, interval=5, repeat=True)

    plt.title("LIF Neuron Membrane Potential Animation")
    plt.xlabel("Time (ms)")
    plt.ylabel("Membrane Potential (mV)")
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

Dynamic plot:

Dato un Δt , e un oggetto neurone, questa funzione, a seguito della simulazione del neurone, genera:

- Il grafico della variazione di potenziale d'azione del neurone
- Una retta indicante il threshold
- Punti rossi in corrispondenza dei momenti di generazione delle spikes.

CODING A LIF MODEL - Main.py

```
def no_current():
    def my_current(x):
        return 0

    lif = LIF_Neuron(lambda x:my_current(x))
    T_total = 100
    dynamic_plot(T_total, lif)
```

```
def impulse_current_no_spike():
    def my_current(x):
        if x<25:return 0
        else: return 0.8

    lif = LIF_Neuron(lambda x:my_current(x), -65)
    T_total = 100
    dynamic_plot(T_total, lif)
```

All'interno di queste funzioni:

- E' definita la funzione **my_current**, che in questo caso restituisce 0mA e 0.8 mA dopo 25 ms.
- Viene inizializzato un oggetto neurone e quindi li viene passata la funzione **my_current**
- Chiama la funzione **dynamic_plot**, che inizializza la simulazione e genera il grafico.

CODING A LIF MODEL - Main.py

```
def impulse_current_with_spike():
    def my_current(x):
        if x<25: return 0
        else: return 1.5

    lif = LIF_Neuron(lambda x:my_current(x), -65)
    T_total = 100
    dynamic_plot(T_total, lif)
```

Analogamente, sono inizializzati neuroni caratterizzati da diverse funzioni che descrivono la corrente.

Nel primo caso la corrente è 0 per $t=0\text{ms}$, 1.5 da $t=25\text{ms}$ in poi.

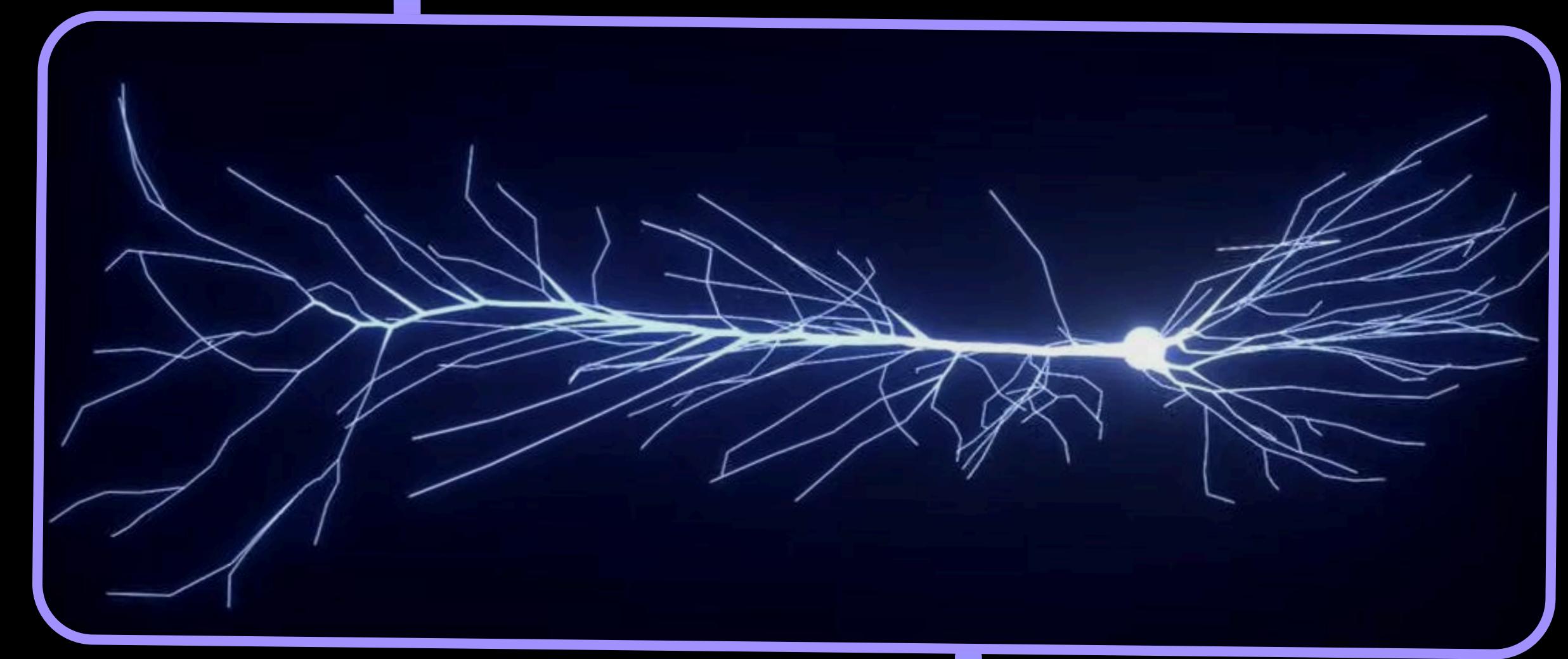
```
def variable_current():
    def my_current(x):
        return np.sin(x)**2 + 1.5

    lif = LIF_Neuron(lambda x:my_current(x), -65)
    T_total = 100
    dynamic_plot(T_total, lif)
```

Nel secondo caso, la funzione ha equazione:

$$y = \sin(x)^2 + 1.5$$

*PROPRIETA NON LINEARI
DEL NEURONE BIOLOGICO*



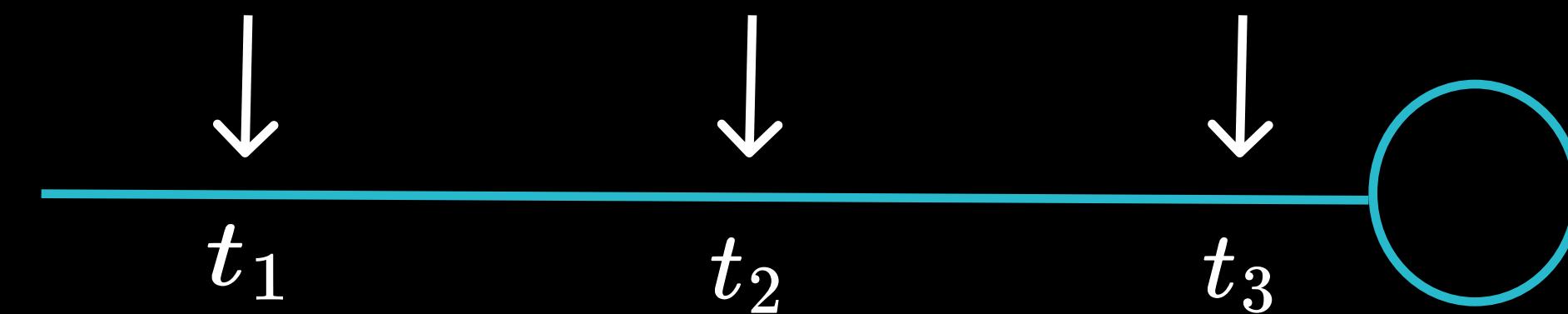
COMPUTAZIONE SPAZIO-TEMPORALE

Ovvero la soluzione di Rall alla “Sequenzialità Selettiva”.

Esempio: Sequenza di posizioni spazio-temporali di un oggetto percepito dai fotorecettori della retina.

Caso 1: $t_1 < t_2 < t_3 \rightarrow$ Incremento.

Caso 2: $t_3 < t_2 < t_1 \rightarrow$ Non-Incremento



In questo modo il dendrite stesso diviene una **unità logica**, un “sub-neurone”.

Il potenziale diminuisce muovendosi verso il Soma a causa di $R_{membrana}$.

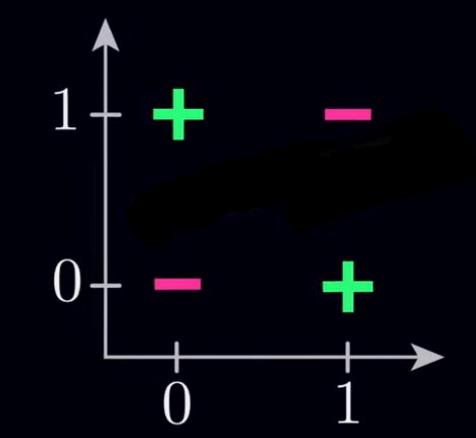
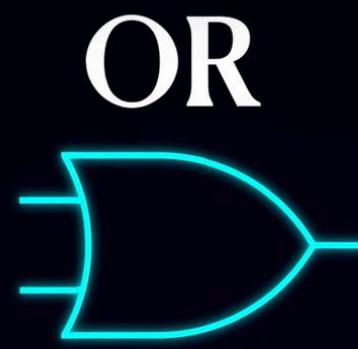
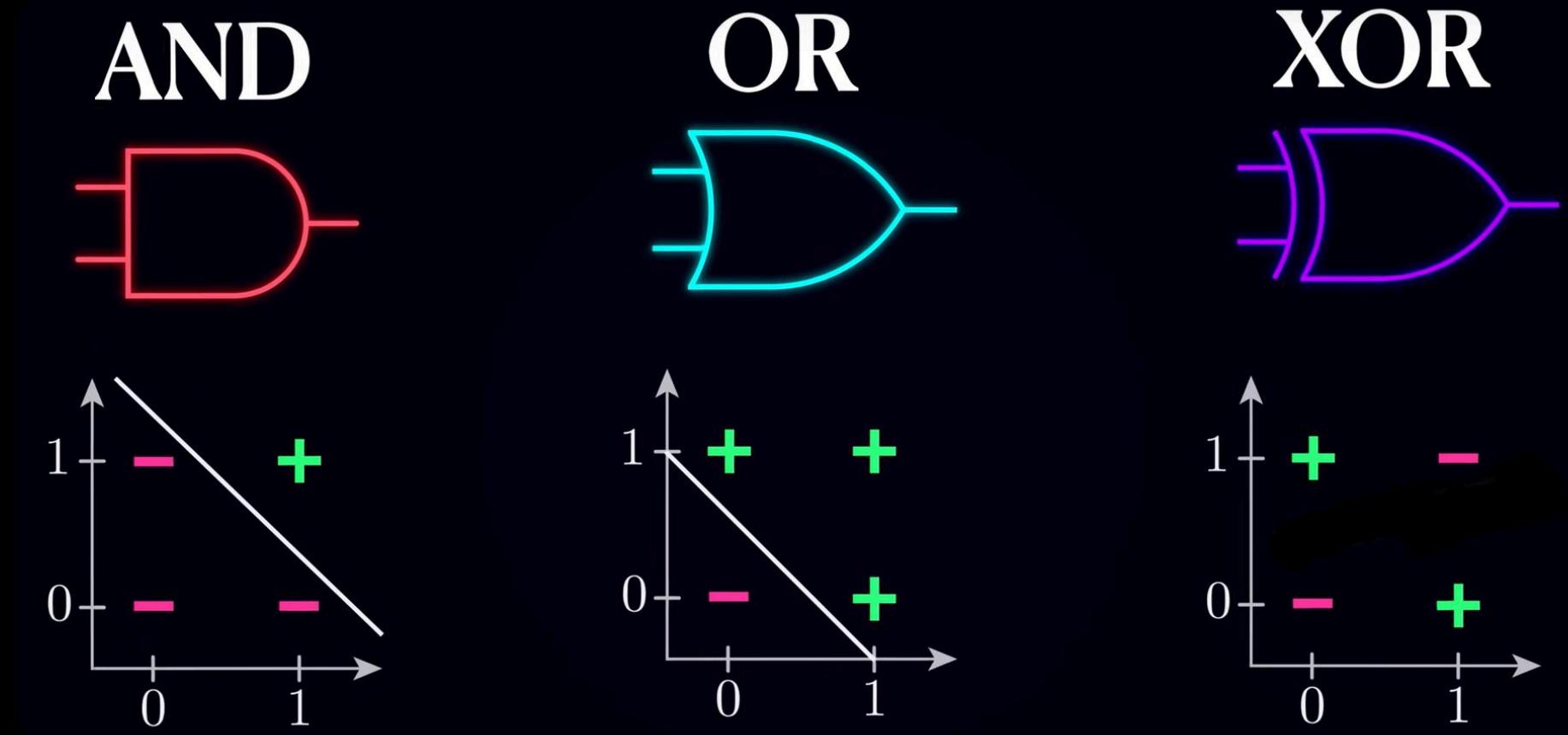
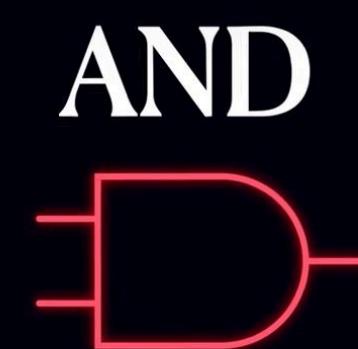
CANALI SELETTIVI: XOR

La porta logica XOR definisce una funzione **Non Linearmente separabile**

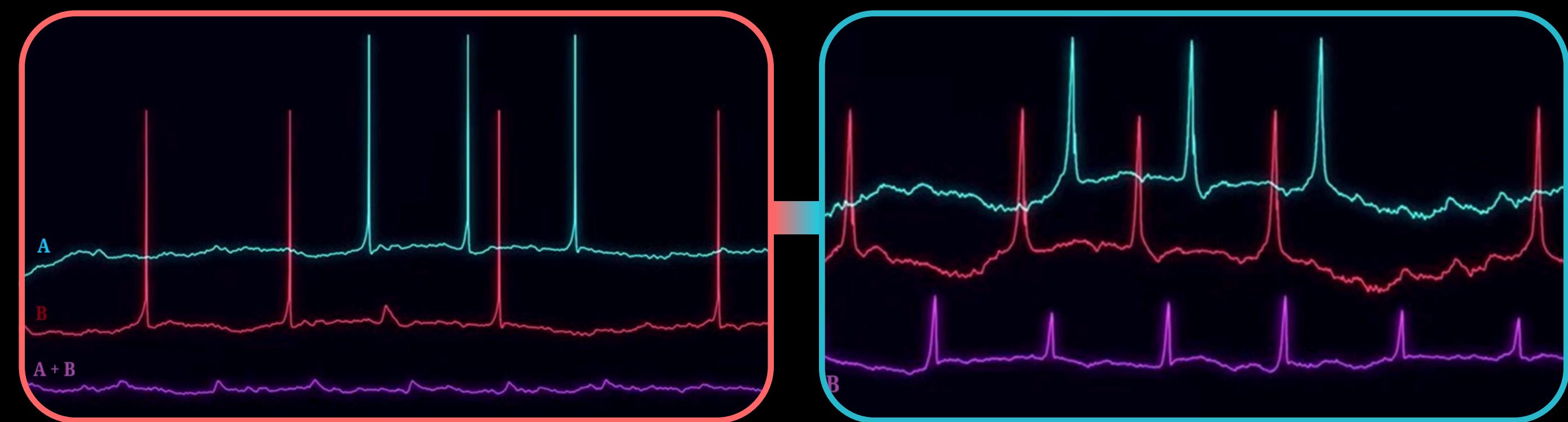
I dendriti presentano canali voltaggio-dipendenti (k^+ , Ca^{2+}) la cui apertura è limitata non solo da un V_{min} ma anche da un V_{max} .

Una rete neurale necessita $n > 1$ layers per computare una XOR.

Approfondimento: “*Simple Cortical Neurons...*” by Seggev & London



Dunque è possibile che dati 2 impulsi A e B, i canali siano attivati solo per A XOR B.



NMDA E BACKPROPAGATION

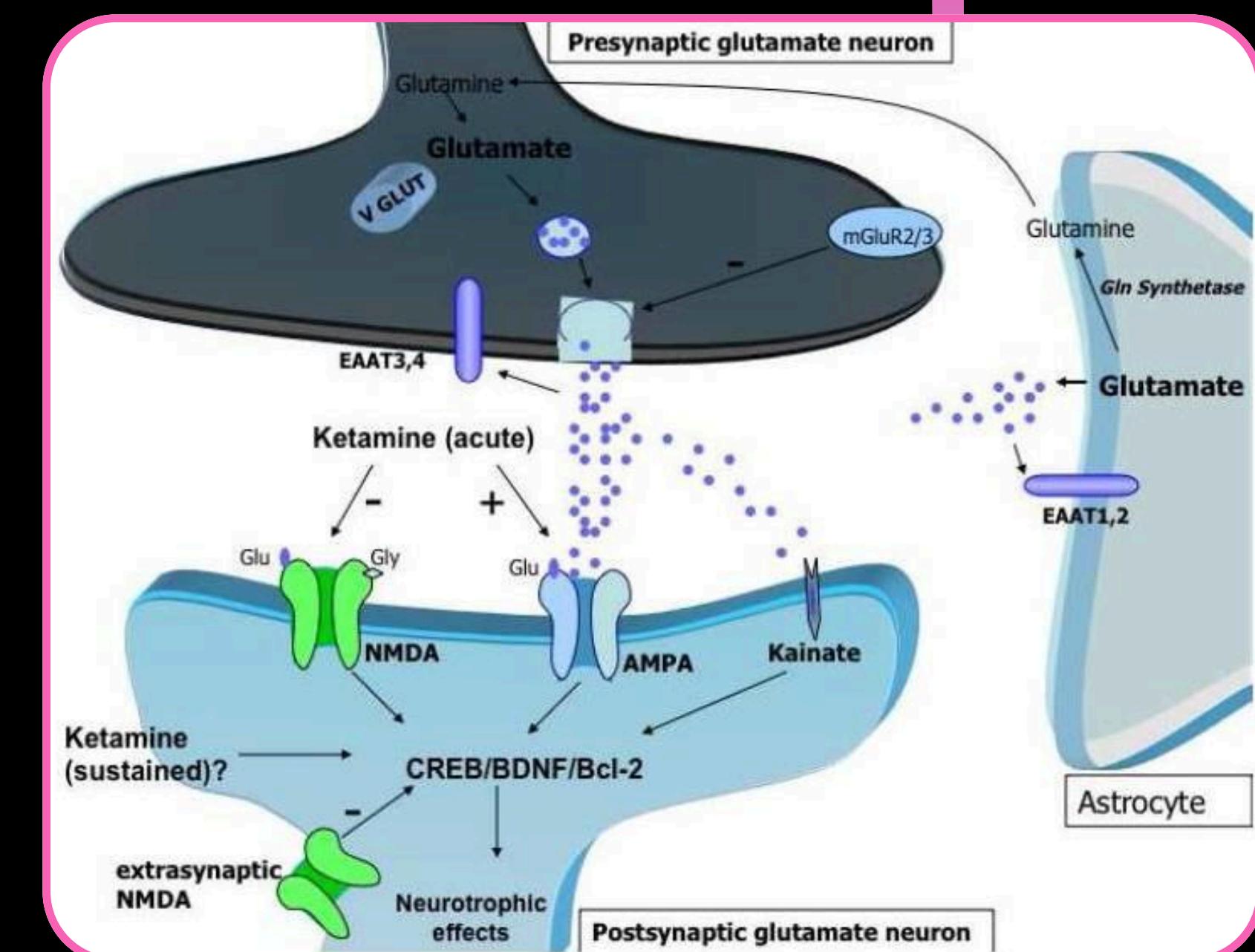
I recettori **N-Metil-D-Aspartato** costituiscono una porta **AND**:
la loro apertura richiede:

- *Presenza di neurotrasmettitori specifici.*
- *Depolarizzazione della membrana postsinaptica.*

Le NMDA sono attivate attraverso
il processo di **Backpropagation**:

1. Apertura canali a Na+ nella *Trigger Zone*.
2. *Propagazione a ritroso del potenziale.*
3. Apertura dei canali NMDA (ingresso Ca 2+).

Essenzialmente ciò permette al Soma di inviare un feedback i dendriti.



NEUROPLASTICITA'

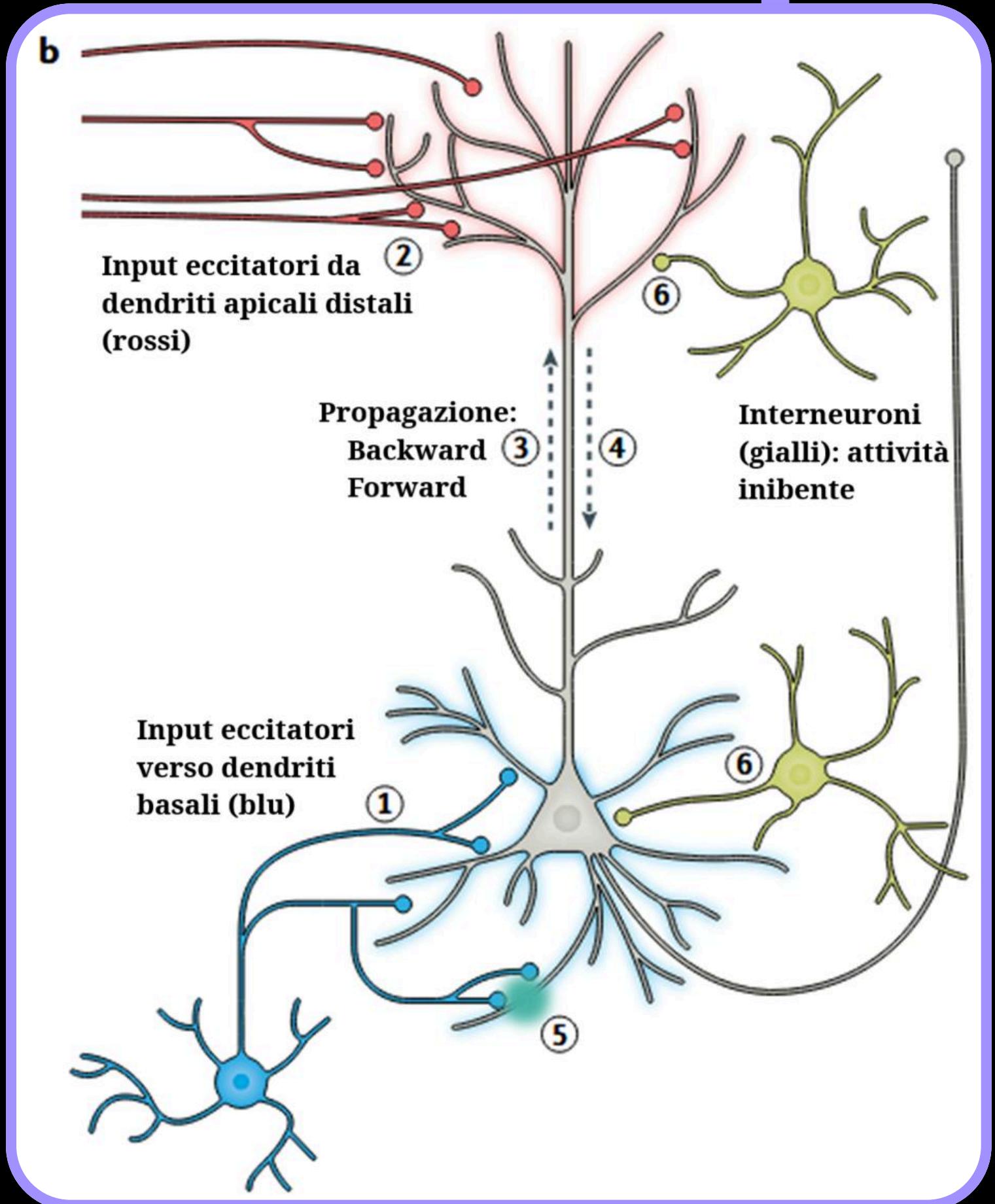
La Backpropagation è essenziale per la STDP:
Spike-Timing Dependent Plasticity:

Δt (tra $t_{impulso}$ e $t_{feedback}$) permette di distinguere se l'input ha contribuito all'output. Connessioni contribuenti saranno rafforzate (LTP), le altre indebolite (LTD).

Apertura dei canali NMDA (ingresso Ca 2+) →
→ Azione di proteine specifiche per LTP o LTD.

$$\Delta w = \begin{cases} A_+ \cdot e^{-\frac{\Delta t}{\tau_+}} & \text{se } \Delta t > 0 \quad (\text{LTP}) \\ -A_- \cdot e^{\frac{\Delta t}{\tau_-}} & \text{se } \Delta t < 0 \quad (\text{LTD}) \end{cases}$$

[Qui Dt è tra $t_{(arrivo dell'impulso al soma)}$ e t_{output}]



NON-LINEAR LIF MODEL:

Abbreviato “NLIF”, è un modello molto più accurato del *LIF* in quanto permette di descrivere il potenziale di membrana sia al di sotto del threshold, sia al di sopra della soglia, quando la V cresce bruscamente generando una “spike”.

Nel LIF in quanto modello lineare, la variazione della derivata di V in funzione di V è rappresentata da una retta:

$$\begin{cases} y = -\frac{1}{\tau} \cdot x + q \\ y = \frac{dV}{dt} \\ x = V \\ q = V_{rest} + R \cdot I(t) \end{cases}$$

Al contrario il NLIF non essendo un modello lineare la derivata di V in funzione di V viene descritta con la seguente equazione:

$$\tau \cdot \frac{dV}{dt} = F(V) + R \cdot I(t)$$

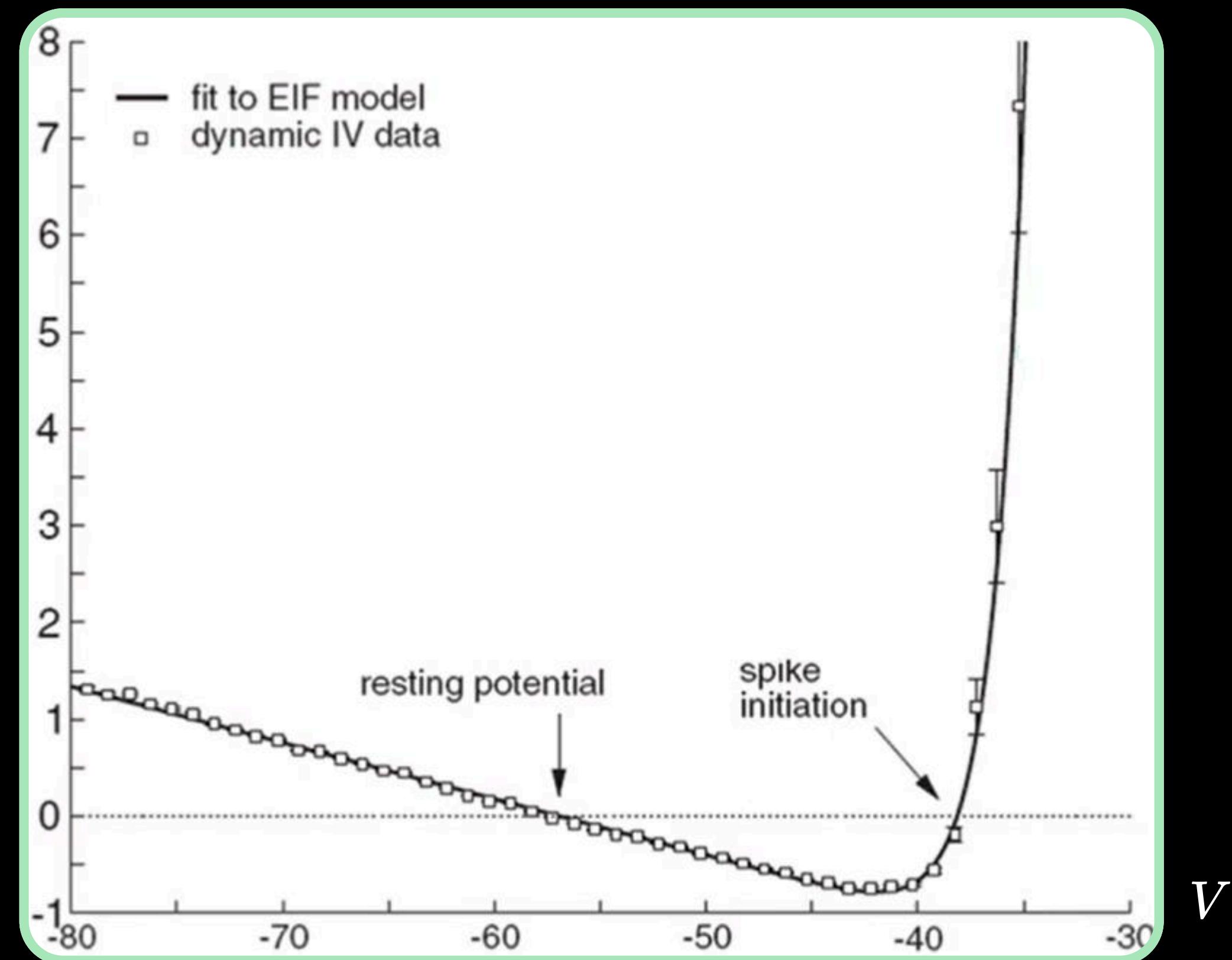
NON-LINEAR LIF MODEL:

Secondo dati sperimentali la funzione $F(V)$ che soddisfa l'equazione differenziale e al contempo approssima meglio i dati sperimentali è:

$$F(V) = - (V - V_{rest}) + c_0 \cdot e^{V-\delta}$$

Si noti come l'andamento dei punti antecedenti all'inizializzazione della spike può essere approssimato con una retta, in quanto il differenziale di V è linearmente dipendente da V , in accordo con il LIF model.

invece se V ha superato la soglia indicata, il differenziale di V cresce esponenzialmente dando origine così alla nota spike.



CODING A NLIF MODEL - NLIF_Model.py

```
def generate_spike_shape(self):  
  
    t = np.linspace(0, self.spike_duration, self.spike_steps)  
    return self.v_rest + 50 * np.exp(-(t - self.spike_duration / 3) ** 2) / (2 * (self.spike_duration / 10) ** 2)  
  
def get_input_current(self, t):  
    return self.current_function(t)  
  
def reset(self):  
    self.v = self.v_rest  
    self.refract_counter = 0  
    self.spike_index = 0  
    self.in_spike = False  
    self.spikes.clear()
```

Il NLIF model, a differenza del precedente, presenta una nuova funzione, **generate_spike_shape**, atta a descrivere il potenziale d'azione dopo il threshold, secondo l'equazione differenziale:

$$\tau \cdot \frac{dV}{dt} = F(V) + R \cdot I(t)$$

CODING A NLIF MODEL - Main.py

*Come in precedenza, per ogni funzione viene definita la funzione **my_current**, da “passare” all’oggetto neurone.*

Ma questa volta tale oggetto è simulato secondo il modello non lineare.

Nel primo caso si ha una corrente intermittente di 1.5 mA, invece nel secondo caso si ha una corrente del tipo:

$$y = \sin(x)^2 + 1.5$$

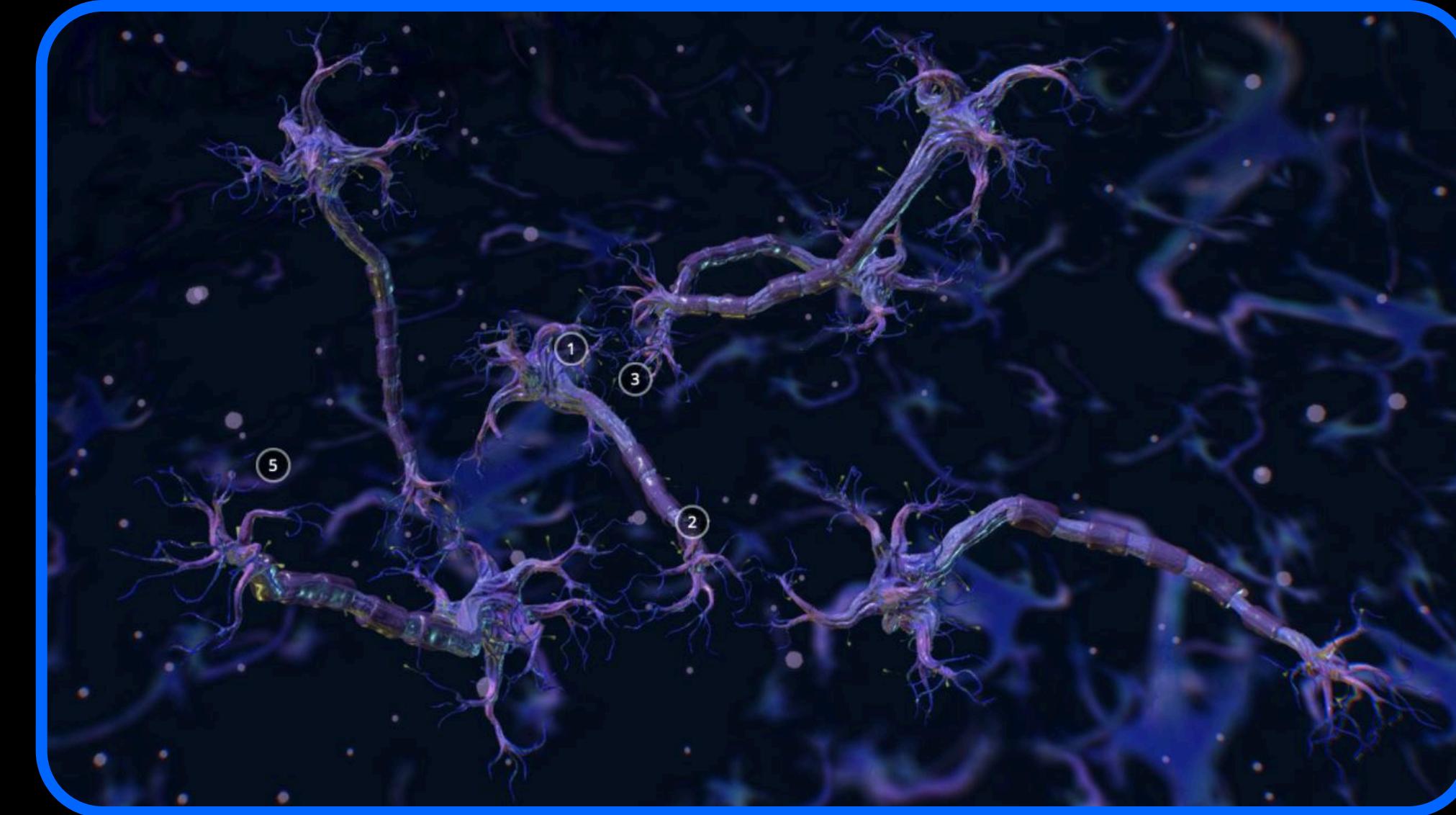
```
def impulse_current_with_spike_NL_Model():
    def my_current(x):
        if x<25 or (x>60 and x<80):return 0
        else: return 1.5

    lif = NLIF_Neuron(lambda x:my_current(x), -65)
    T_total = 100
    dynamic_plot_NL(T_total, lif)

def variable_current_NL_model():
    def my_current(x):
        return np.sin(x)**2 + 1.5

    lif = LIF_Neuron(lambda x:my_current(x), -65)
    T_total = 100
    dynamic_plot_NL(T_total, lif)
```

MEMORIA BIOLOGICA ASSOCIATIVA



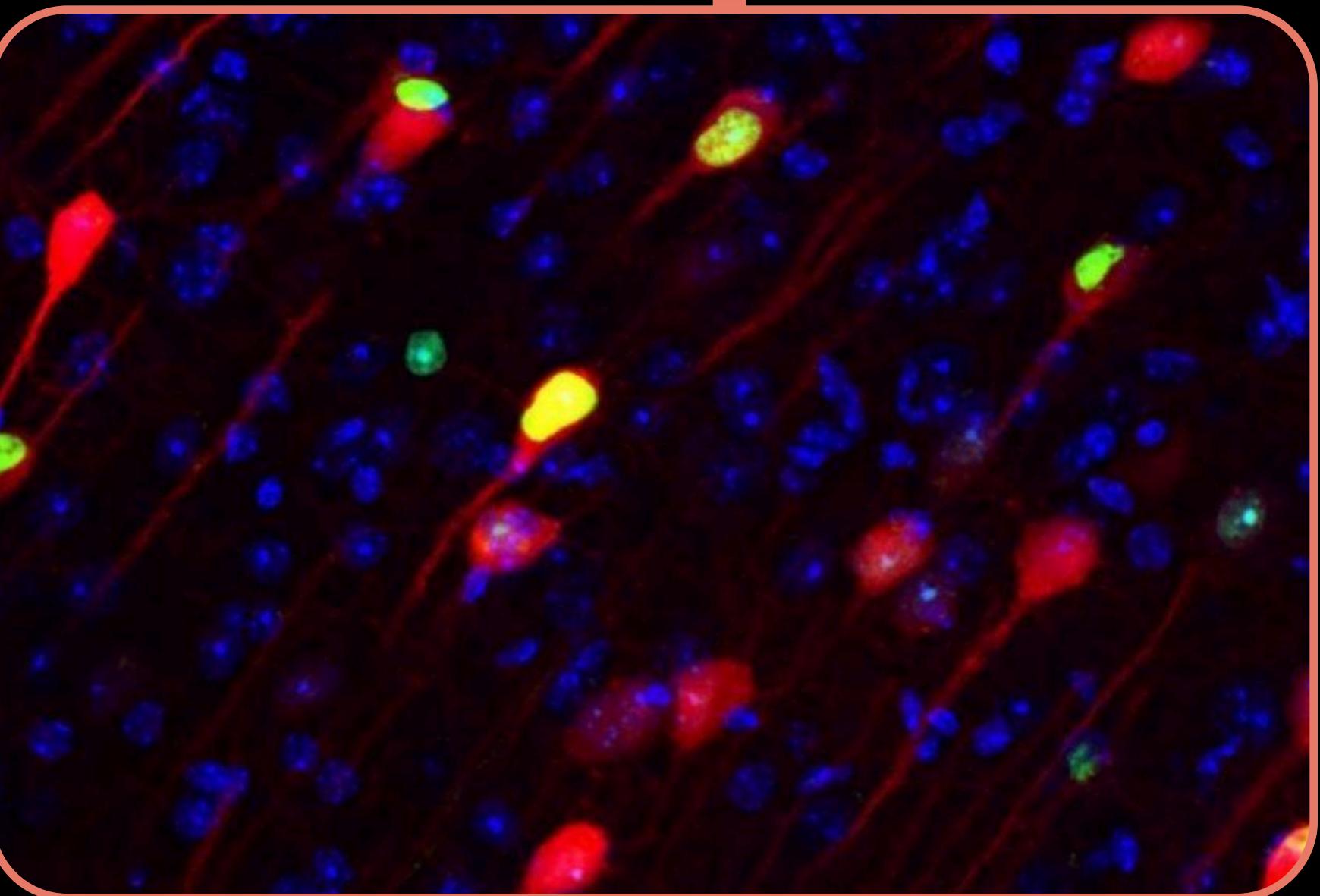
ENGRAM E FEAR CONDITIONING

Engramma = “rappresentazione dell'insieme delle modifiche strutturali che costituiscono la base fisica dell'immagazzinamento di una memoria.”

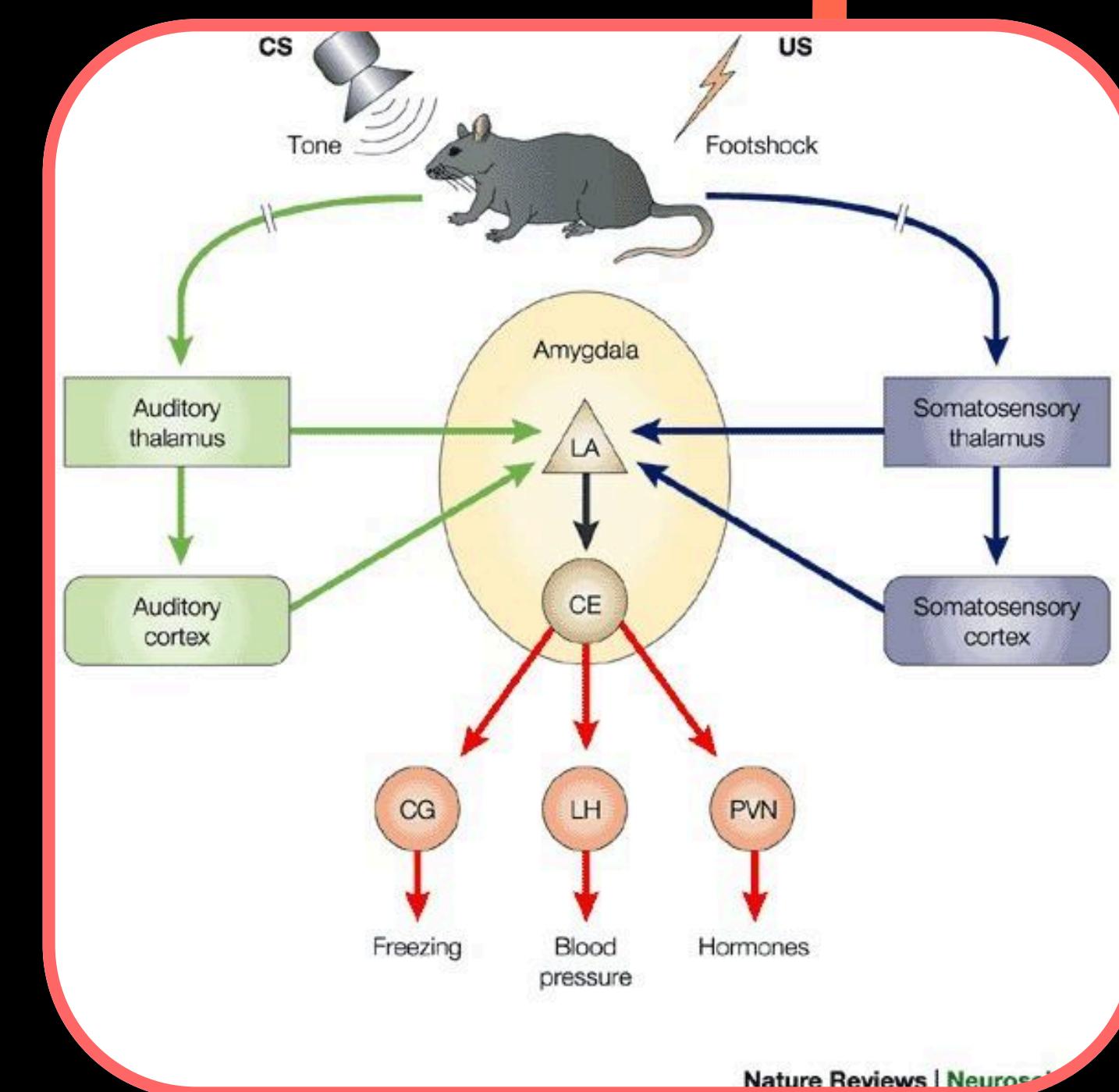
Il n di neuroni di un engramma non varia in base alla “intensità” della memoria:

$$N_e n \approx N_e m \forall \text{Engram}$$

Un engramma è distribuito in diverse regioni del cervello, corrispondenti a diversi dati della memoria (spaziali, temporali etc.)



ENGRAM E FEAR CONDITIONING



Osservazione Sperimentale (cavia):

1. *Iniezione di virus contenente geni per un marcatore.*
 2. *Attivazione virus (sostanza attivante).*
 3. *Fear Conditioning (formazione memoria).*

(2. e 3. avvengono in successione immediata)

I neuroni che prendono parte all'engramma esprimono
“geni precoci” (c-fos);

I geni del marcatore si legano ad essi → il marcatore è prodotto solo in neuroni coinvolti.

E' possibile sopprimere l'associazione senza sopprimere le memorie, interrompendo l'attività di determinati Interneuroni.

CO-ALLOCAZIONE E CO-RIEVOCAZIONE

Eccitabilità: determina la rapidità di risposta all'input ed il valore del threshold γ_r . Varia nel tempo.

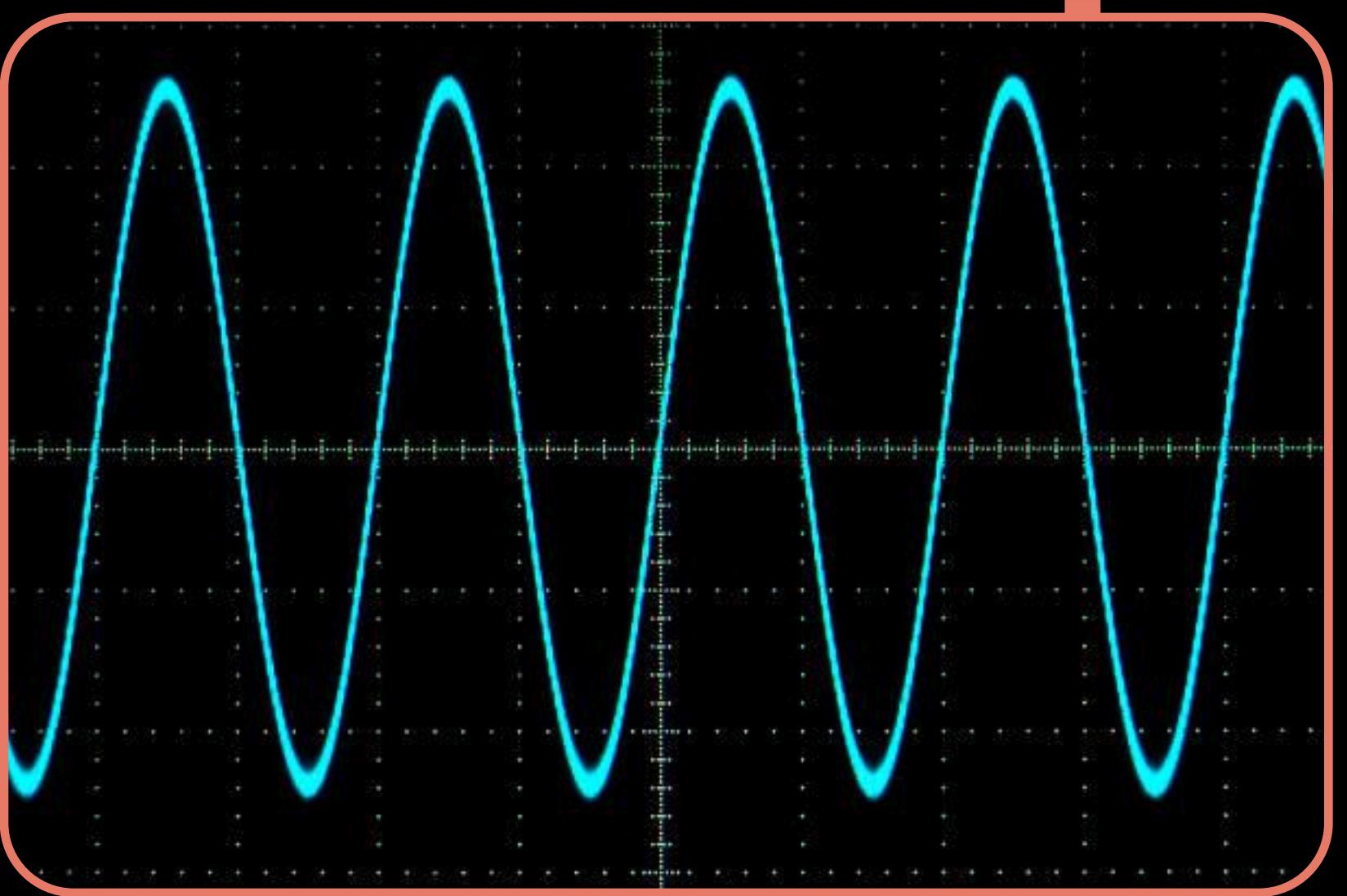
($T_{ecc} \approx 6h$).

Alta Eccitabilità al momento dell'associazione → → maggiore probabilità di formare l'Engramma. Neuroni eccitabili inibiscono l'attività di quelli adiacenti.

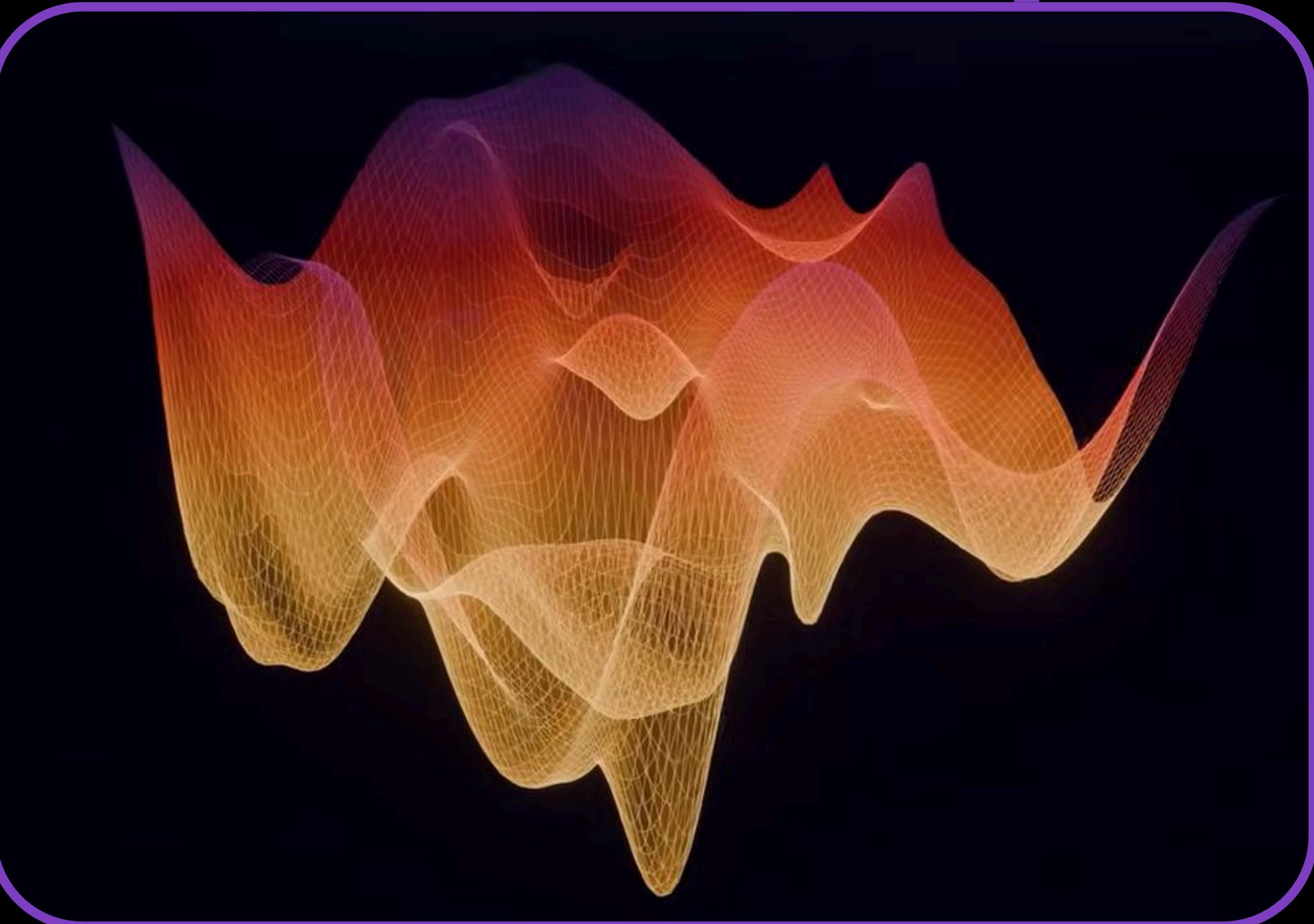
Associazione per Co-Allocazione: Dati 2 engrammi generati con $\Delta t < 6h$ → Sovrapposizione.

Associazione per Co-Rievocazione : Se 2 engrammi richiamati spesso insieme ad essi sono aggiunti dei neuroni di “Overlap” → Co-Allocazione.

“Ricordo” = stimolo di connessioni rafforzate in una qualsiasi sottosezione dell'Engramma.



MODELLO HOPFIELD PER I SISTEMI NEURALI



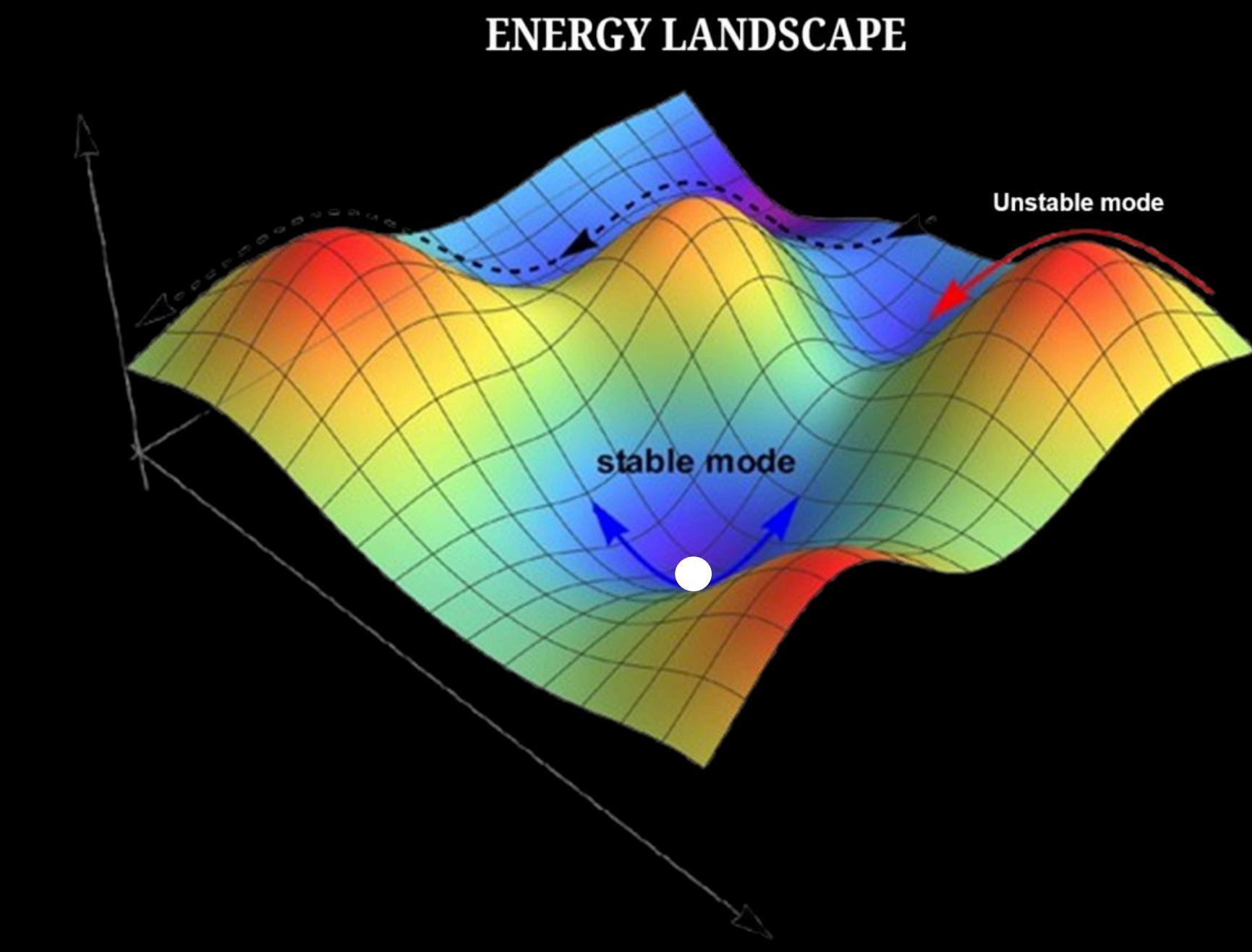
MEMORIA IN UN SISTEMA DINAMICO

Definiamo la “Memoria” come la capacità di un sistema che sia trovato in uno stato E, di tornare a tale stato.

Se una rete presenta un “Preferred State”, a cui torna quando perturbata, esso può dirsi una “memoria” della rete.

Il preferred state è anche detto **Stable State**.

E’ possibile visualizzare una memoria come un punto di minimo in un **Energy Landscape**.



Approfondimento: Il Paradosso di Levinthal

IL MODELLO DI HOPFIELD

Obiettivo → creare un equivalente dell'energy landscape in un sistema con N neuroni informatici.

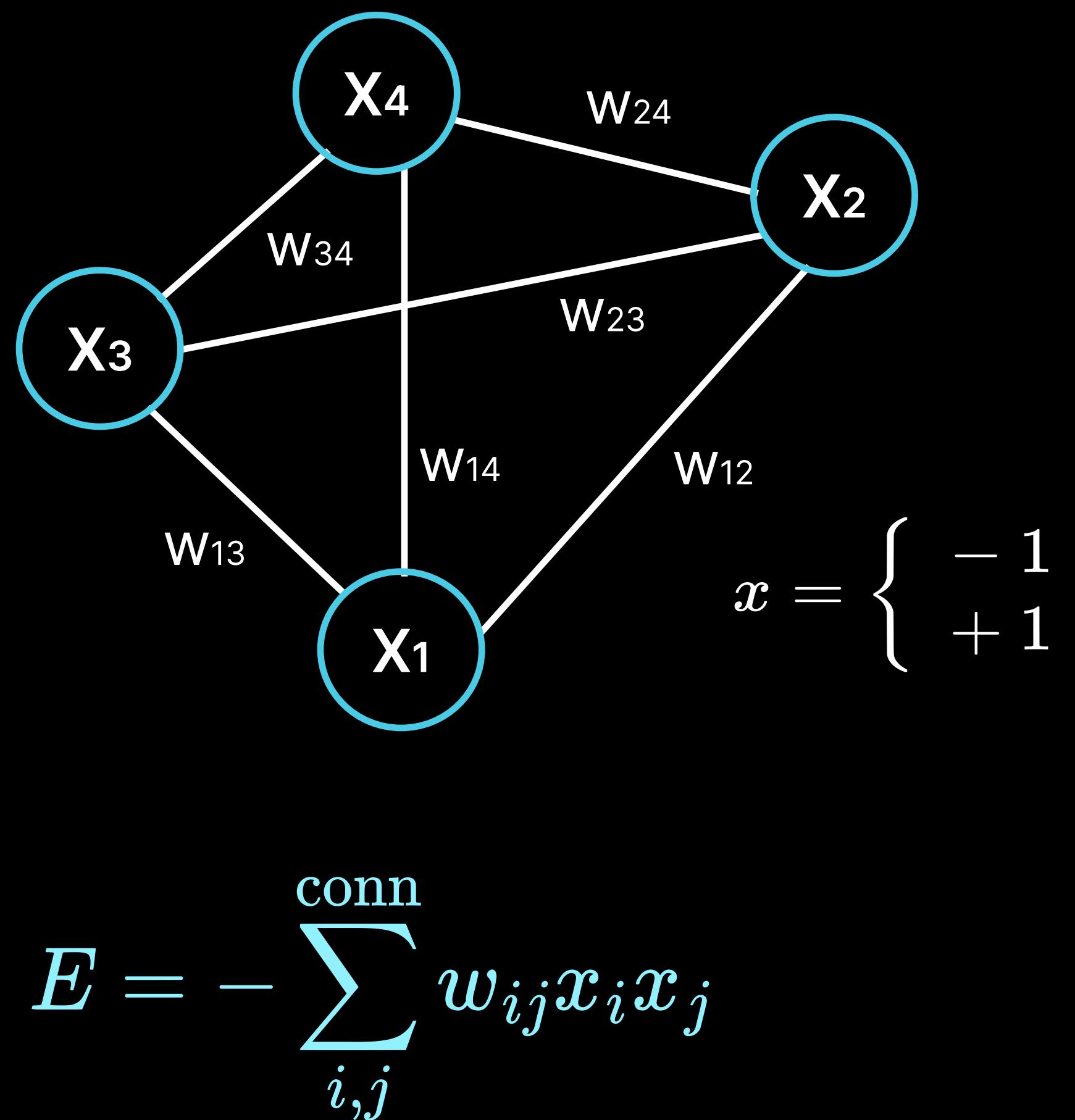
Si considera una rete Simmetrica ($W_{ij} = W_{ji}$).

$w_{ij} > 0 \rightarrow x_i = x_j$ (*Allineamento*)

$w_{ij} < 0 \rightarrow x_i \neq x_j$ (*Disallineamento*)

Gli stati i X_i e X_j possono essere in **Accordo** o in **Disaccordo** con il peso della connessione W_{ij} .

Definiamo lo stato di Disallineamento del sistema come:



INFERENCE: Arrivare alla memoria.

Data una rete che presenti già uno Stable State,
→ **Update Asincrono** dello stato di ogni nodo.

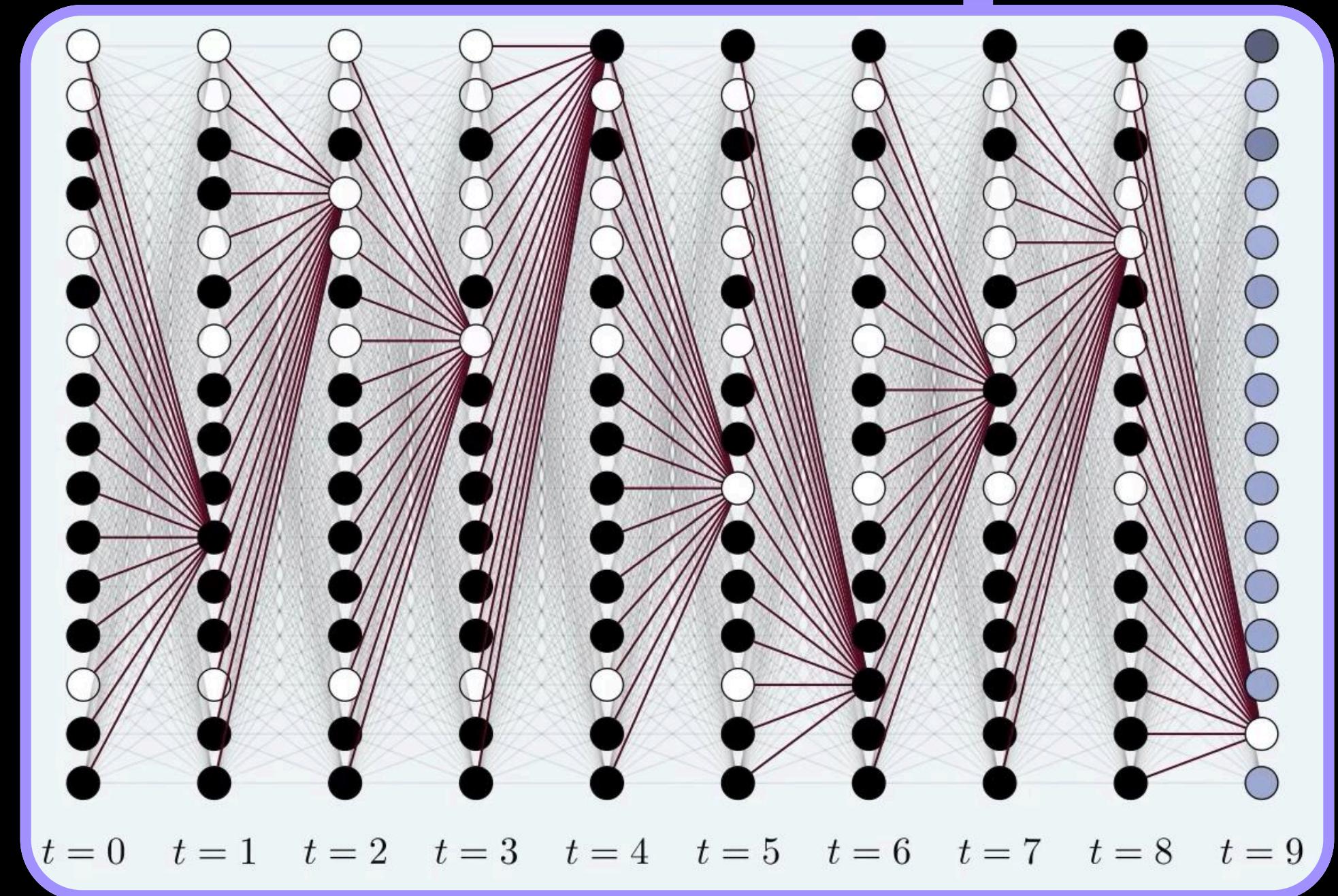
$$x_i(t) = f\left(\sum_{i=t}^{\text{conn}} w_i x_i(t-1)\right)$$

$$f(x) = \begin{cases} +1 & \text{se } x \geq 0 \\ -1 & \text{se } x < 0 \end{cases}$$

Ogni iterazione del processo diminuisce E.

Lo Stable State costituisce un “minimo locale”
dell’ Energy Landscape di E.

Per $t \rightarrow \infty$, il network **converge** al Stable State, o
allo stato opposto ad esso (antimemoria).



LEARNING: Generare la memoria

La **Regola Hebbiana** afferma: “*Neurons that fire together, wire together*”.

$$\Delta w_{ij} = \eta x_i x_j$$

η = Fattore d'apprendimento

RECALL: Plasticità.:

Se 2 neuroni sono attivi contemporaneamente il peso della loro connessione è rafforzato, se non lo sono è indebolito.

Definiamo i vettori che descrivono rispettivamente i pesi e gli stati della rete quando essa è nel pattern dello Stable State.

$$\varepsilon = \begin{bmatrix} \varepsilon_1 = \pm 1 \\ \varepsilon_2 = \pm 1 \\ \vdots \\ \varepsilon_n = \pm 1 \end{bmatrix}$$

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \cdots & w_{1m} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2m} \\ w_{31} & w_{32} & w_{33} & \cdots & w_{3m} \\ \vdots & \vdots & \vdots & & \vdots \\ w_{n1} & w_{n2} & w_{n3} & \cdots & w_{nm} \end{bmatrix}$$

Posto $w_{ii} = 0$ per evitare auto-reinforcemento

LEARNING: Generare la memoria

Poniamo: $w = \xi_i \xi_j$

Partendo da un pattern “vicino” a $\vec{\xi}$, attraverso l’Inference si arriverà ad esso.

Per memorizzare “p” pattern:

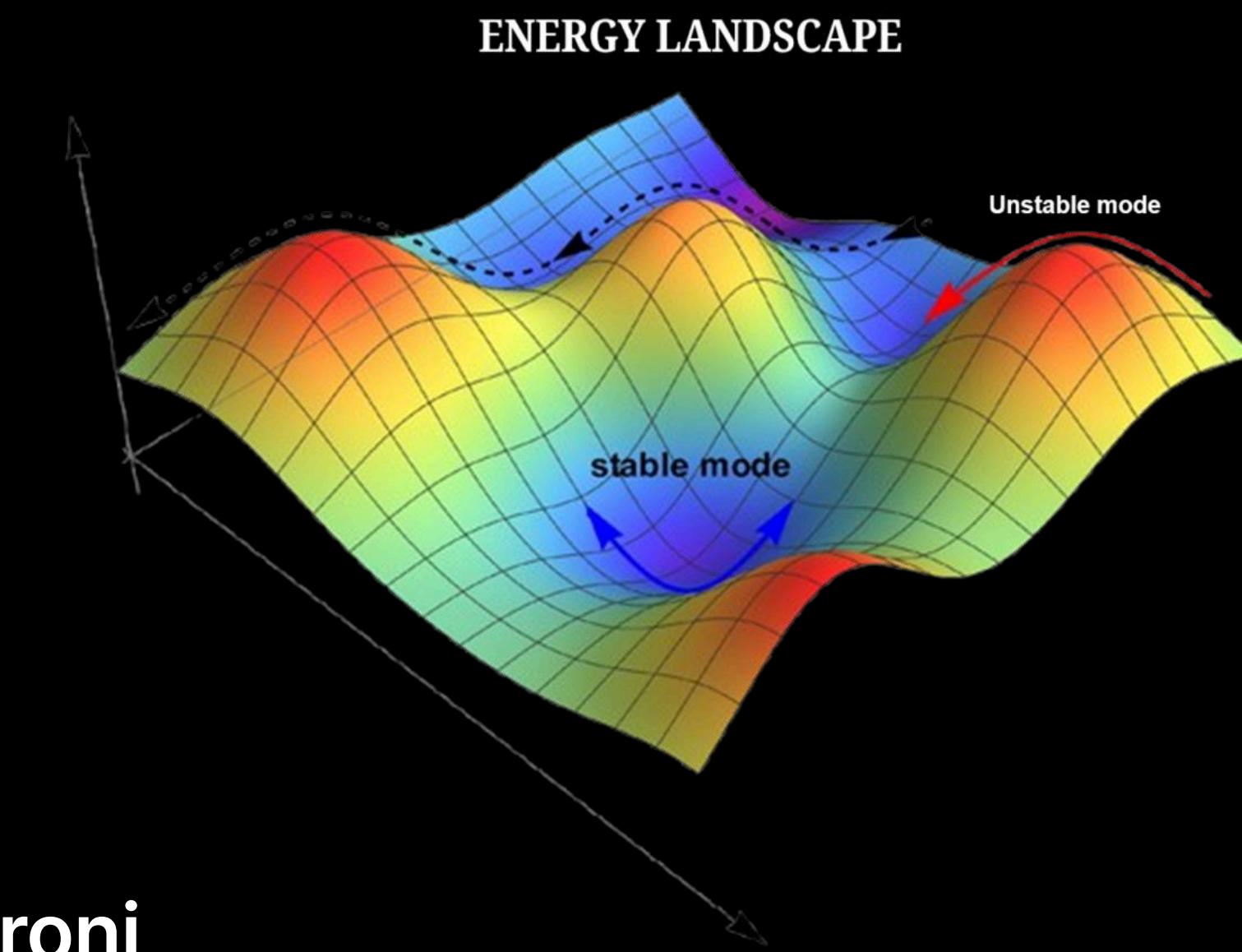
$$w_{ij} = \sum_{\mu=1}^p x_i^\mu x_j^\mu$$

(Il numero di valori che W può assumere è $n = p+1$)

Il modello supporta fino a $p = 0.14 * N$ neuroni

Per evitare w troppo grandi si usa la formula normalizzata:

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^p x_i^\mu x_j^\mu$$



GRAZIE PER L'ATTENZIONE

Per ulteriori approfondimenti è possibile accedere alla nostra dettagliata relazione riguardo al LIF ed al relativo modello informatico al seguente link:

<https://github.com/y4sss3r/Neuronal-Dynamics/>

