

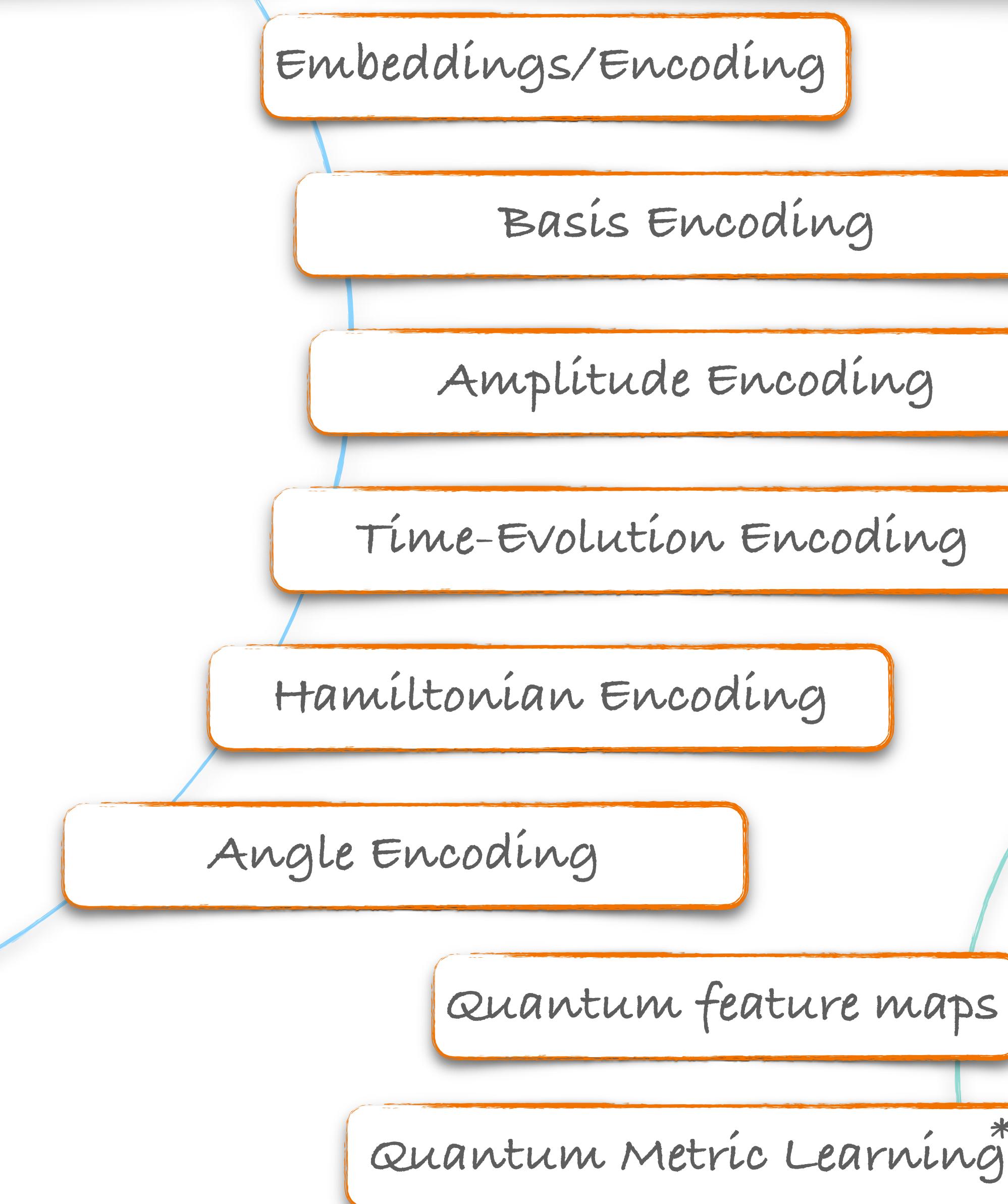
# Applied QML

*Lecture 4: Data embeddings*

Christophe Pere

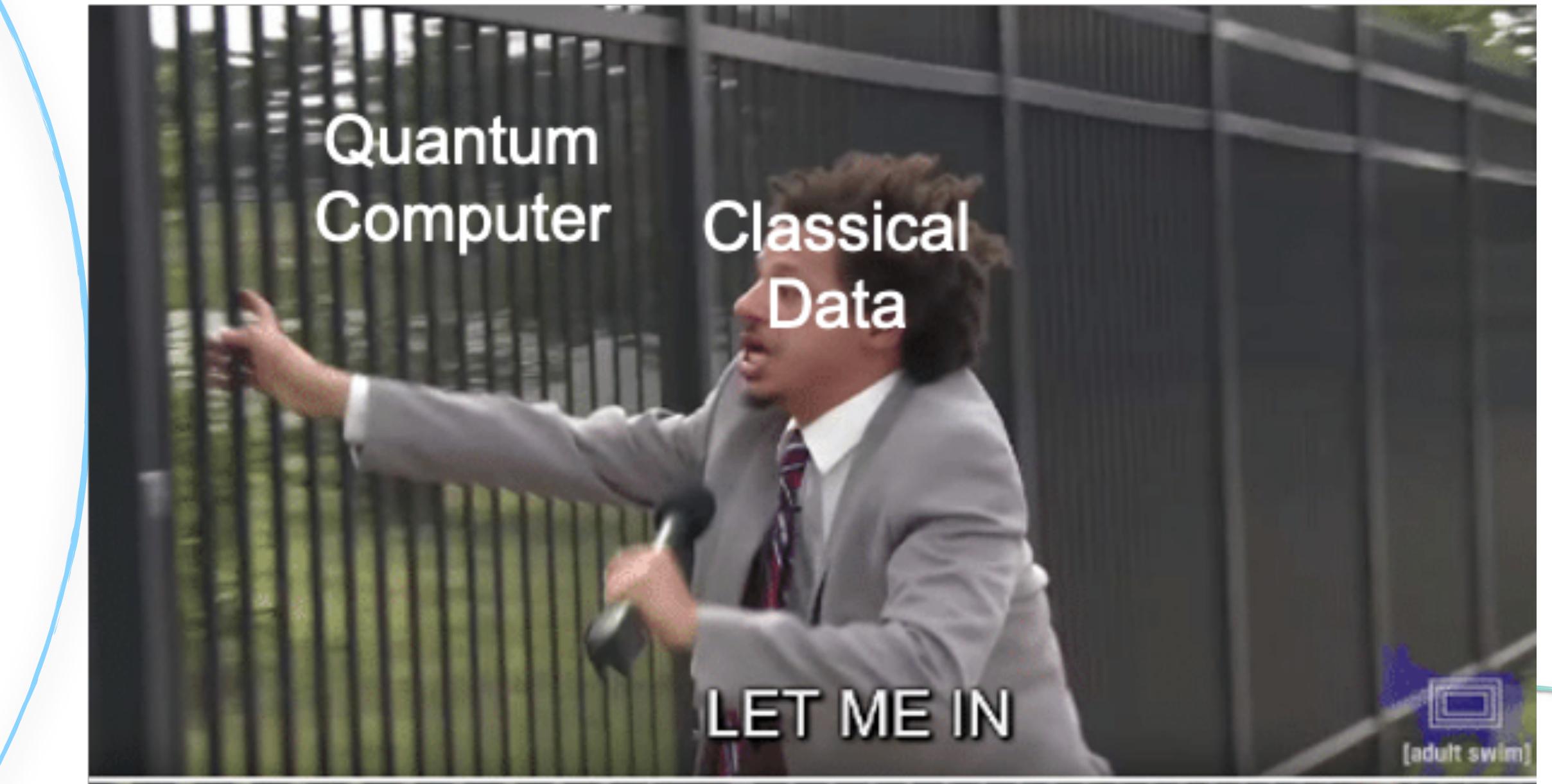
2024-01-xx

# Table des matières



# Introduction

Qu'est-ce que l'encodage de données?  
Pourquoi est-ce important?

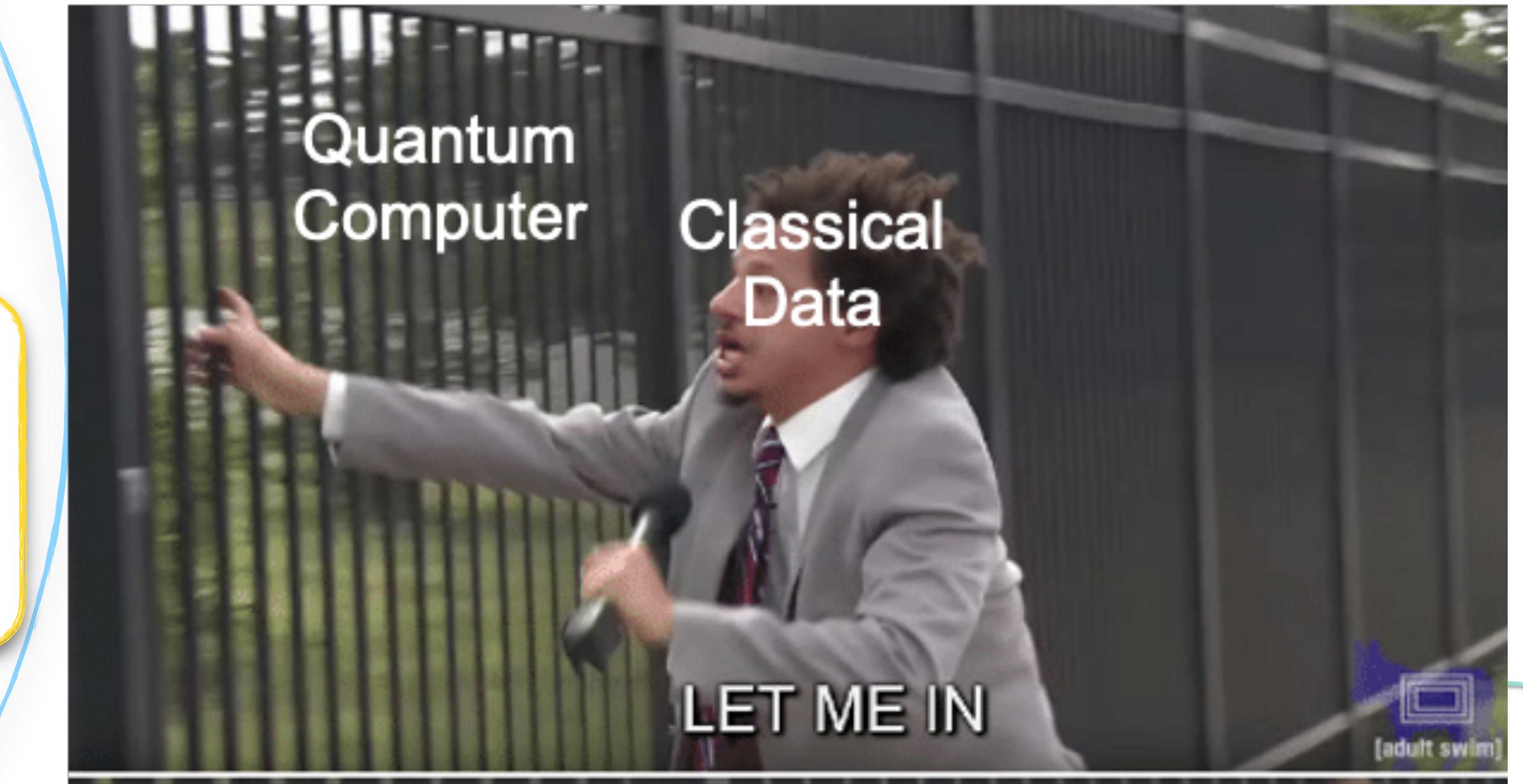


# Introduction

Qu'est-ce que l'encodage de données?  
Pourquoi est-ce important?

Motivation

Comment peut-on utiliser  
des données classiques  
avec un QC? Comment les  
encoder?



# Introduction

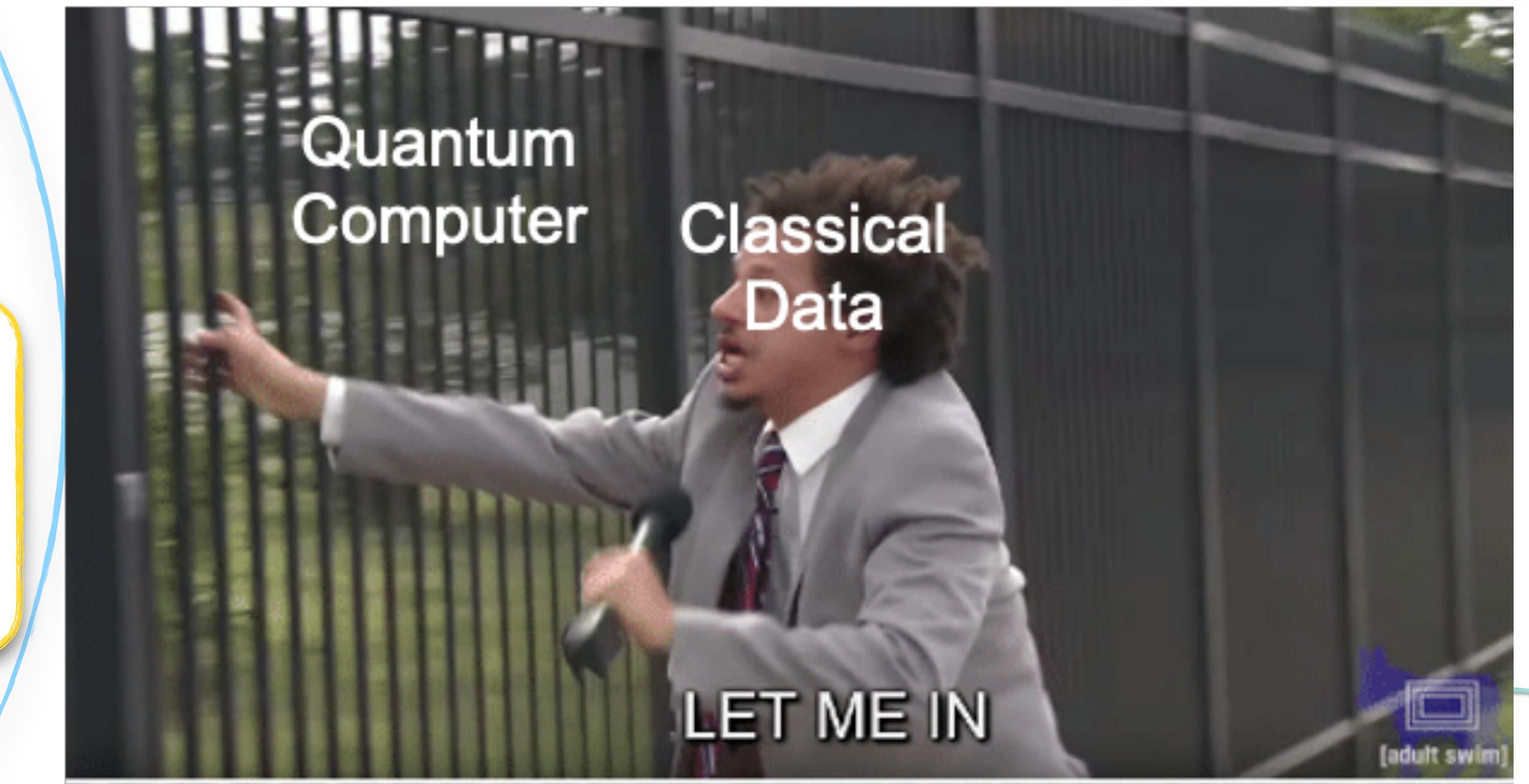
Qu'est-ce que l'encodage de données?  
Pourquoi est-ce important?

Motivation

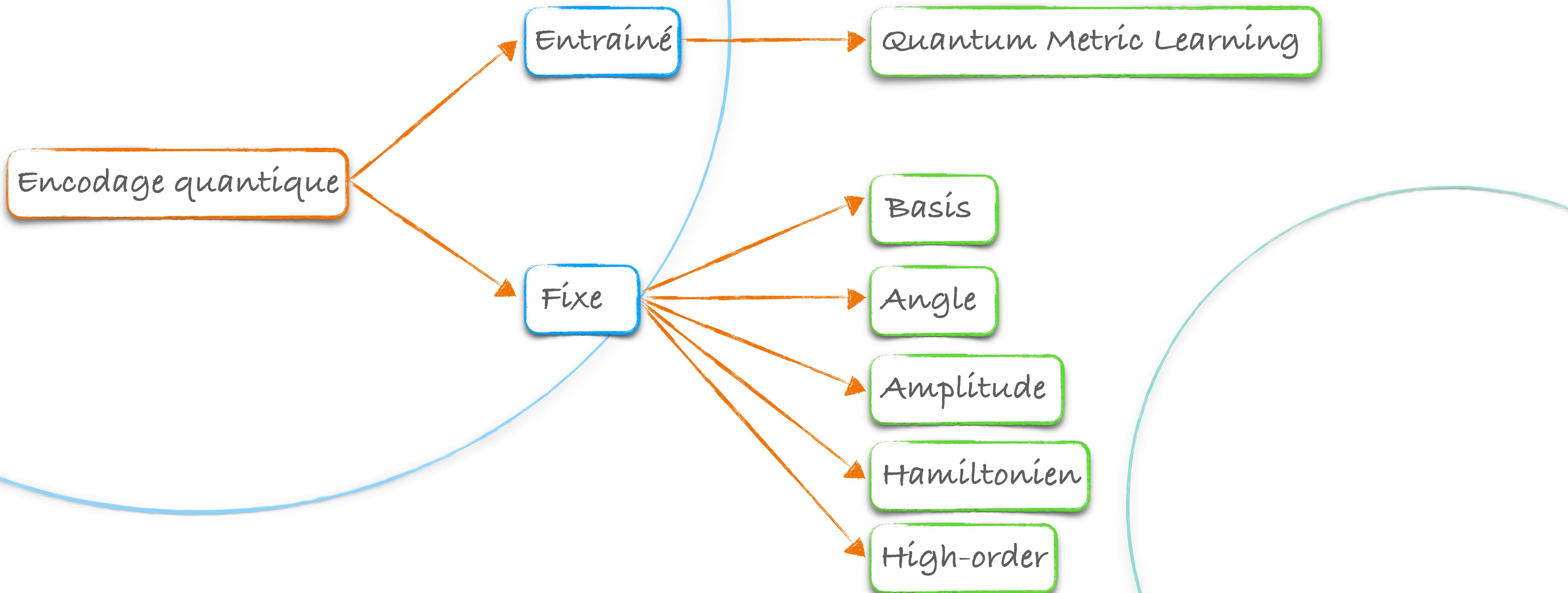
Comment peut-on utiliser  
des données classiques  
avec un QC? Comment les  
encoder?

Embeddings/  
encodings

Série d'algorithmes et  
stratégies pour associer de  
la donnée classique a des  
états quantiques



# Introduction



# Représentation

Encodage de base d'une chaîne binaire (1,0)  
i.e. représentation de l'entier 2

$$|\psi\rangle = \alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|10\rangle + \alpha_3|11\rangle$$

Encodage des amplitudes d'un vecteur complexe de longueur unitaire  
 $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$

Encodage par Hamiltonian d'une matrice A

$$e^{-iH_A t}$$

Encodage par évolution-temporelle d'un scalaire t

# BASIS Encoding

Représentation sous la forme de chaînes binaires

Nombre réels à manipuler

$$x^{(m)} = (b_1, \dots, b_N) \quad \text{with} \quad b_i \in \{0,1\} \text{ for } i = 1, \dots, N$$

$x^{(m)}$  peut être directement mappé en  $|x^{(m)}\rangle$

Superposition

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^M |x^{(m)}\rangle$$

Dataset

Signe +/-  
ajout d'un bit  
supplémentaire  
0 pour +, 1 pour -

Avantage

un dataset peut être mis en superposition

Inconvénient

1 bit string est encodé dans un qubit

$N$  bits correspondent à  $2^N$  possible états de base

Comme  $M \ll 2^N$  L'encodage de  $\mathcal{D}$  sera sparse

Exemple 1:

$$|x^{(1)}\rangle = |01\rangle$$

$$|x^{(2)}\rangle = |11\rangle$$

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{2}} |01\rangle + \frac{1}{\sqrt{2}} |11\rangle$$

Exemple 2:

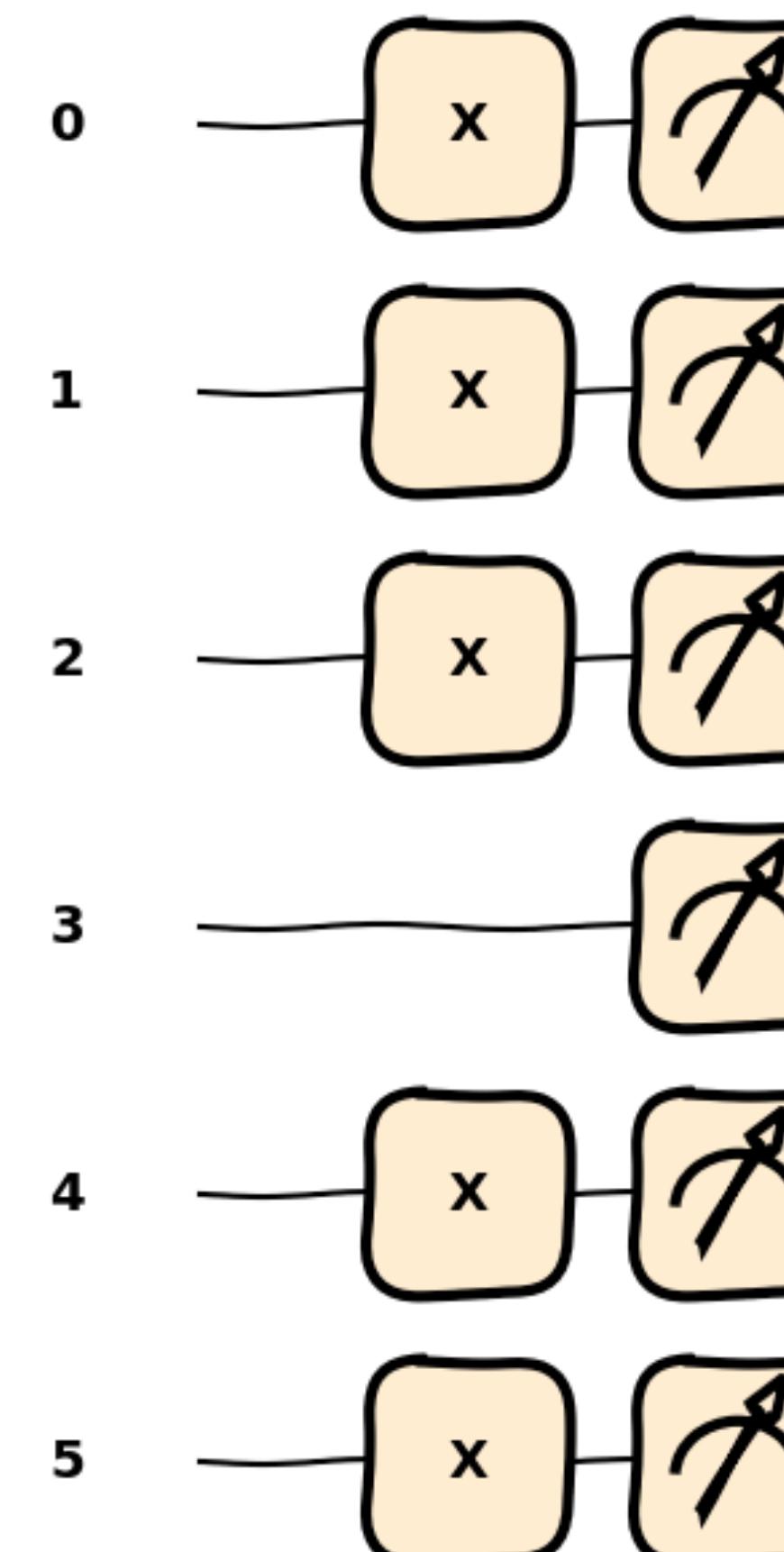
$$\begin{aligned} x &= 1001 & 4 \text{ qubits} \\ x &= 1011 & \rightarrow |1001\rangle \\ x &= 0011 & \rightarrow |1011\rangle \\ & & \rightarrow |0011\rangle \end{aligned}$$

# Basis Encoding

Code!!

$x = [1, 1, 1, 0, 1, 1]$

```
dev = qml.device('default.qubit',
wires=len(x))
def circuit(vector):
    for i in range(len(vector)):
        if vector[i] == 1:
            qml.PauliX(wires=i)
    return qml.state()
print(qml.draw_mpl(circuit,
expansion_strategy="device",
style="sketch")(x))
```



$x = [1, 1, 1, 0, 1, 1]$

```
dev = qml.device('default.qubit',
wires=len(x))
def circuit(vector):
    qml.BasisEmbedding(feature
s=vector, wires=len(vector))
    return qml.state()
print(qml.draw_mpl(circuit,
expansion_strategy="device",
style="sketch")(x))
```

# Amplitude Encoding

Représente la donnée directement

vecteur de nombres réels

Représentation vectorielle

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{2^n} \end{pmatrix} \leftrightarrow |\psi_x\rangle = \sum_{j=0}^{2^n-1} x_j |j\rangle$$

valeurs normalisées et normées  
utilisées comme amplitude

Représentation matricielle

$$A \in \mathbb{C}^{2^n \times 2^n}; \sum_{ij} |\alpha_{ij}|^2 = 1$$

$$|\psi_A\rangle = \sum_{i=0}^{2^m-1} \sum_{j=0}^{2^n-1} \alpha_{ij} |i\rangle |j\rangle$$

Complex

$$x \in \mathbb{C}^{2^n}; \sum_k |x_k|^2 = 1$$

valeurs normalisées

ième ligne

jème colonne

# Amplitude Encoding

vecteur de nombres réels

Exemple

$$x = (0.073, -0.438, 0.730, 0.000) \text{ vecteur normalisé}$$

Encodage dans les amplitudes

$$|\psi_x\rangle = 0.073|00\rangle - 0.438|01\rangle + 0.730|10\rangle + 0.000|11\rangle$$

Matrice correspondante

$$A = \begin{pmatrix} |0\rangle & |1\rangle \\ 0.073 & -0.438 \\ 0.730 & 0.000 \end{pmatrix} \begin{pmatrix} |0\rangle \\ |1\rangle \end{pmatrix}$$

Rappel  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$

$$|\psi^*\rangle = \frac{\alpha}{\sqrt{|\alpha|^2 + |\beta|^2}}|0\rangle + \frac{\beta}{\sqrt{|\alpha|^2 + |\beta|^2}}|1\rangle$$

2 qubits pour encoder 4 features

8 qubits pour 256 features

Gain en qubits comparé à l'encodage binaire  
Représentation plus compacte mais qui est plus couteuse en terme de temps de calcul

# Amplitude Encoding

vecteur de nombres réels

Exemple

$$x = (0.073, -0.438, 0.730, 0.000) \text{ vecteur normalisé}$$

Encodage dans les amplitudes

$$|\psi_x\rangle = 0.073|00\rangle - 0.438|01\rangle + 0.730|10\rangle + 0.000|11\rangle$$

Matrice correspondante

$$A = \begin{pmatrix} |0\rangle & |1\rangle \\ 0.073 & -0.438 \\ 0.730 & 0.000 \end{pmatrix} \begin{pmatrix} |0\rangle \\ |1\rangle \end{pmatrix}$$

Rappel  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$

$$|\psi^*\rangle = \frac{\alpha}{\sqrt{|\alpha|^2 + |\beta|^2}}|0\rangle + \frac{\beta}{\sqrt{|\alpha|^2 + |\beta|^2}}|1\rangle$$

2 qubits pour encoder 4 features

8 qubits pour 256 features

Comment encoder un dataset?

# Amplitude Encoding

Comment encoder un dataset?

$$\text{Dataset: } \mathcal{D} = \{x^1, x^2, \dots, x^M\}$$

$$\text{Amplitudes: } \alpha = C_{\text{norm}} x_1^{(1)}, \dots, x_N^{(1)}, x_1^{(2)}, \dots, x_N^{(2)}, \dots, x_1^{(M)}, \dots, x_N^{(M)}$$

critère de normalisation

$$|\alpha|^2 = 1$$

Dataset en entrée

$$|\mathcal{D}\rangle = \sum_{i=1}^{2^n} \alpha_i |i\rangle$$

Éléments du vecteur amplitude

États de base

Constante de normalisation

On pourrait stocker  
un volume  
exponentiellement  
plus grand de données

Nombre d'amplitudes  
 $N \times M$   
L'encodage requiert  
 $n \geq \log_2 (N \times M)$   
qubits

# Amplitude Encoding

Code!!

```
import pennylane as qml  
dev = qml.device('default.qubit', wires=2)  
@qml.qnode(dev)  
  
def circuit(f=None):  
    qml.AmplitudeEmbedding(features=f, wires=range(2),  
                           normalize=True)  
    return qml.expval(qml.Pauliz(0)), qml.state()
```

Permet de donner un vecteur sans qu'il soit normalisé

```
data = [15, 15, 15, 15]  
res, state = circuit(f=data)
```

```
>>> state
```

```
tensor([0.5+0.j, 0.5+0.j, 0.5+0.j, 0.5+0.j], requires_grad=True)
```

# Amplitude Encoding

Code!!

Custom?

Il suffit de préparer un état quantique

```
data = [15, 15, 15, 15]
data_norm = np.array(data)/np.linalg.norm(data)
state = prep_circuit(state=data_norm)
```

>>> state

```
tensor([0.5+0.j, 0.5+0.j, 0.5+0.j, 0.5+0.j], requires_grad=True)
```

```
import pennylane as qml
dev = qml.device('default.qubit', wires=2)
```

```
@qml.qnode(dev)
```

```
def prep_circuit(state=None):
    qml.StatePrep(state, wires=range(2))
    return qml.state()
```

```
from pennylane import numpy as np
```

Norme d'un vecteur ou matrice

# Amplitude Encoding

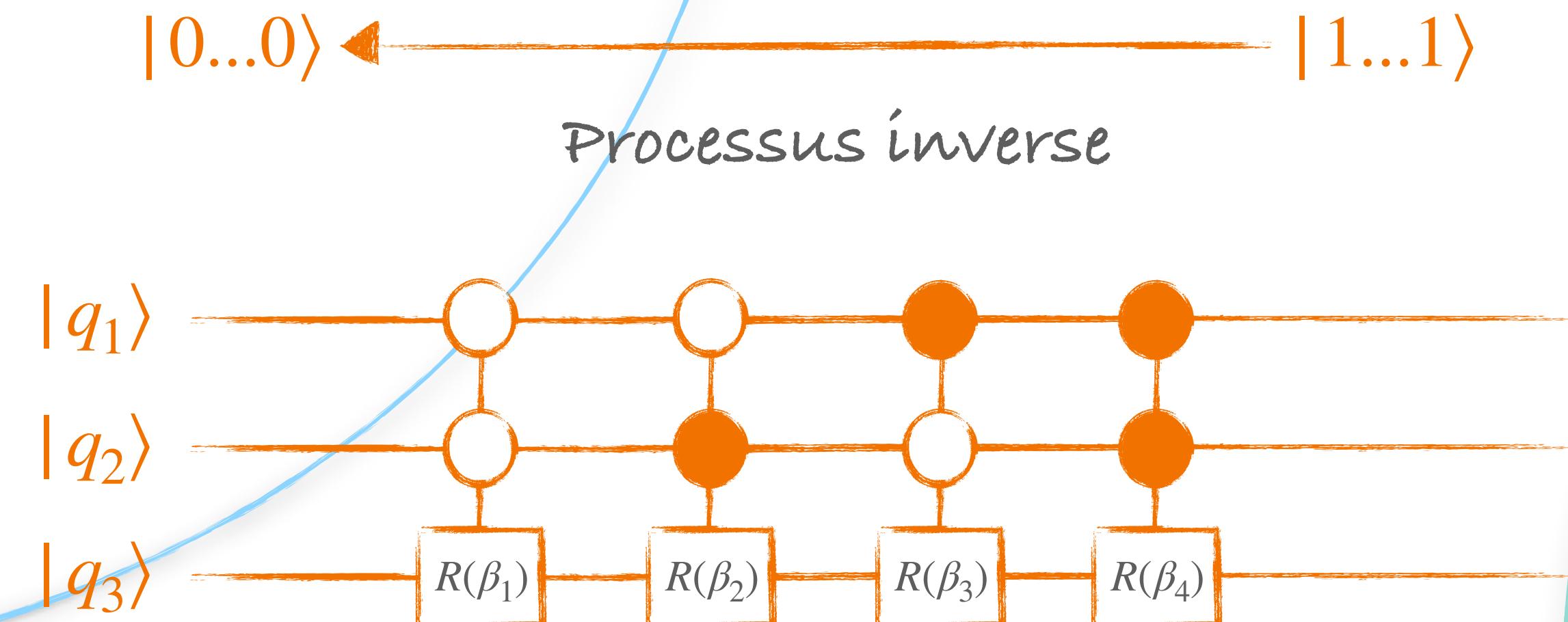
Amplitude-Efficient State Preparation

Rotation sur toutes les branches en superposition

Chacune différente sur chacune des branches

Multi-controlled rotations

une cascade de  $R_z$   
Égalise le signe de l'amplitude pour faire une phase globale



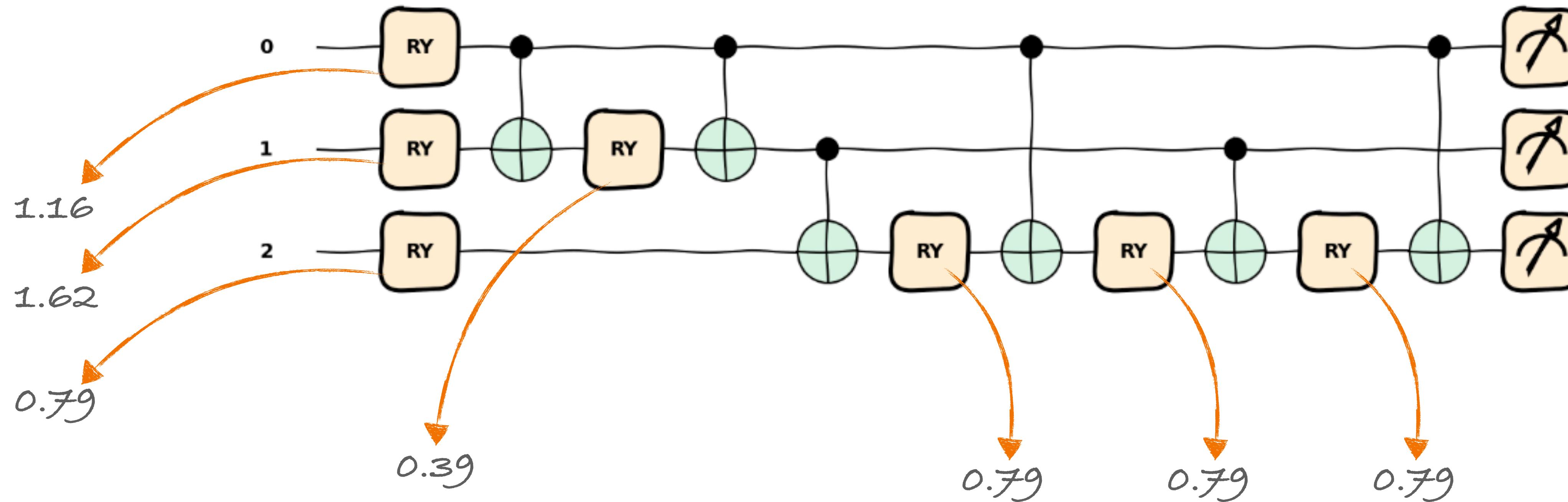
suivi d'une cascade de  $R_y$   
Égalise l'amplitude pour être dans l'état fondamental

Très gourmand en resources

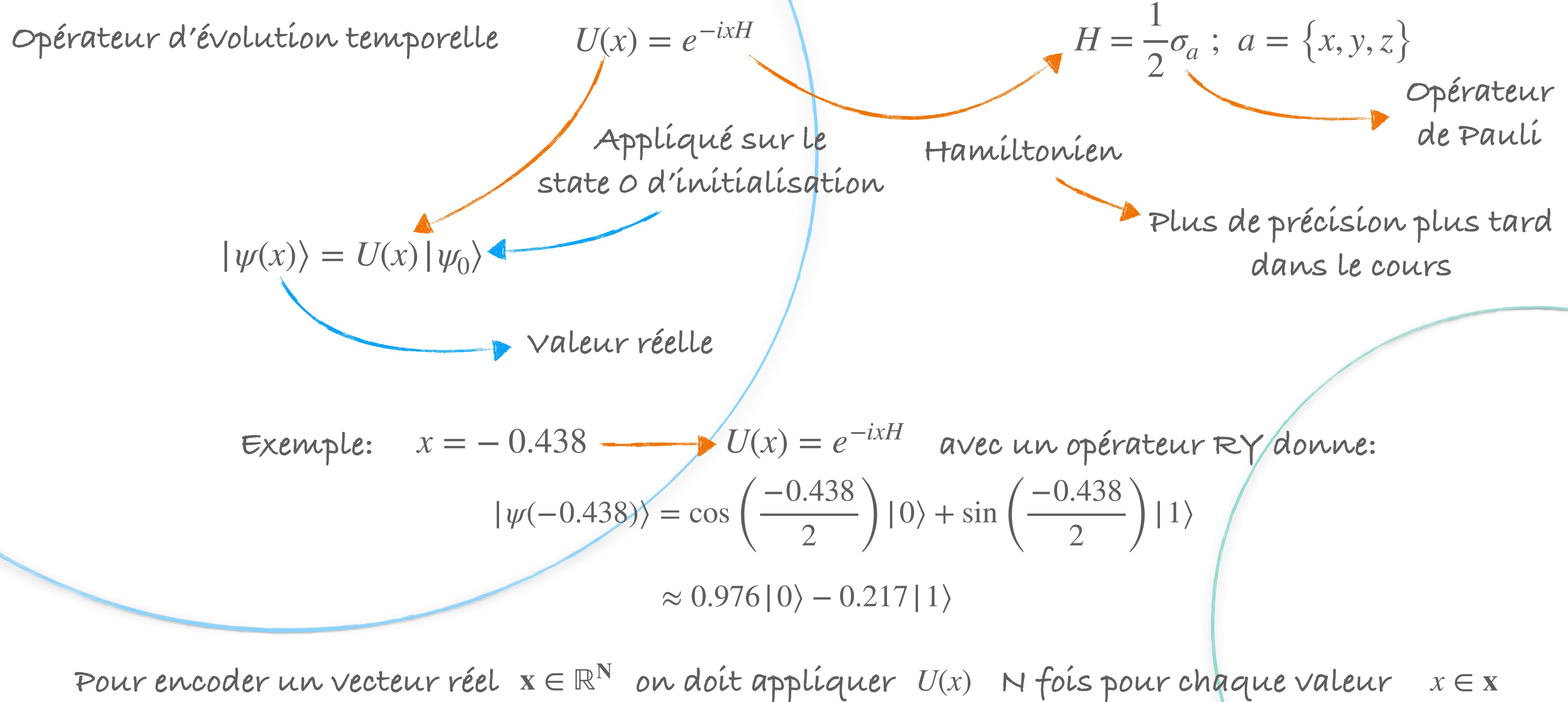
# Amplitude Encoding

Amplitude-Efficient State Preparation

$$|\psi\rangle = \sqrt{0.2} |000\rangle + \sqrt{0.5} |010\rangle + \sqrt{0.2} |110\rangle + \sqrt{0.1} |111\rangle + 0|100\rangle + 0|001\rangle + 0|011\rangle + 0|101\rangle$$

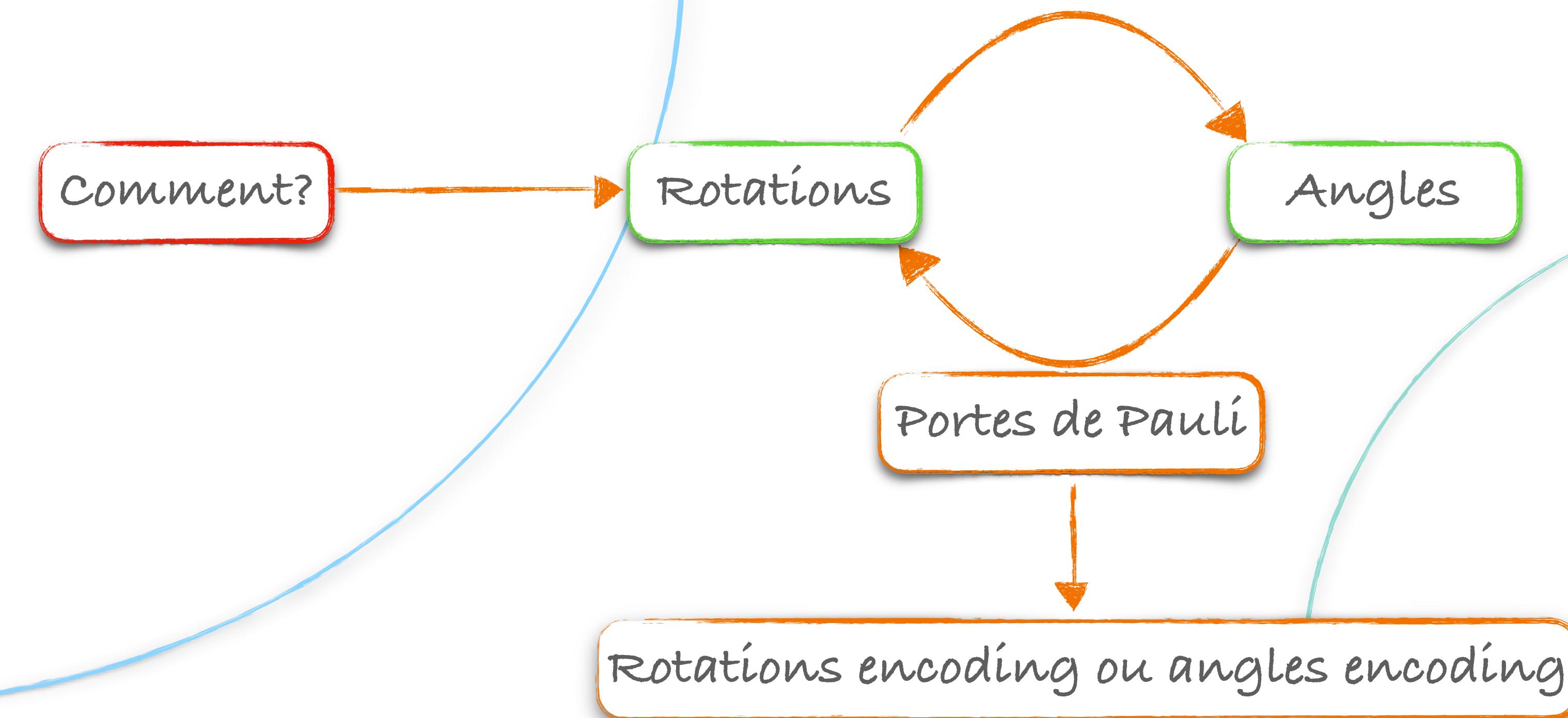


# Time-Evolution Encoding



# Time-Evolution Encoding

Code!!



# Angle Encoding

Et si on essayait avec les phases?

Phase shift -> Rotation dans le plan complexe

Angles

Rotation (X, Y ou Z)

$$R_x(\theta) = e^{-i\theta X/2} = \begin{bmatrix} \cos(\theta/2) & -i \sin(\theta/2) \\ -i \sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$$

$$R_y(\theta) = e^{-i\theta Y/2} = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$$

$$R_z(\theta) = e^{-i\theta Z/2} = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}$$

1 feature / qubit

# Angle Encoding

Et si on essayait avec les phases?

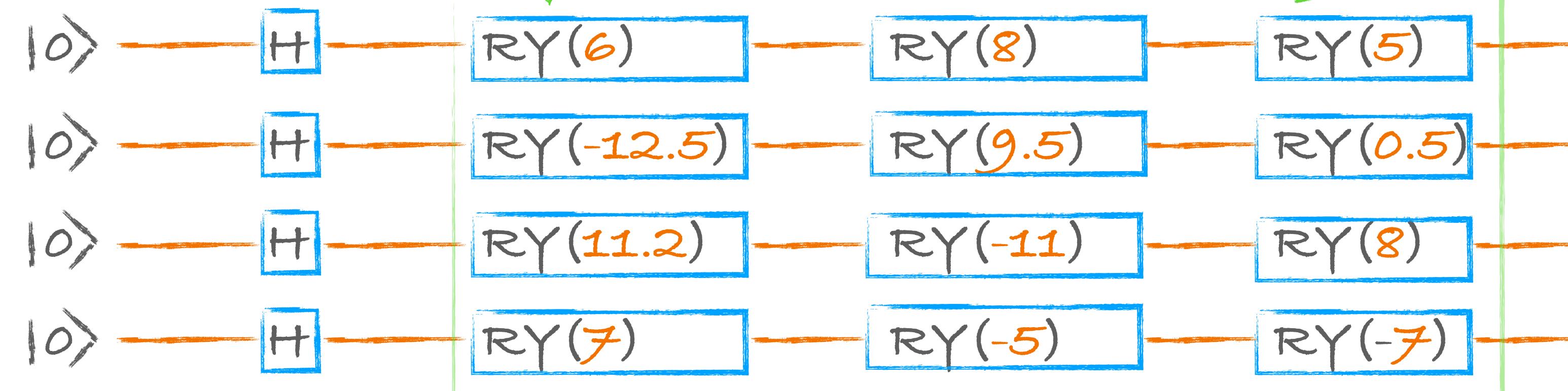
Phase shift -> Rotation dans le plan complexe

Angles

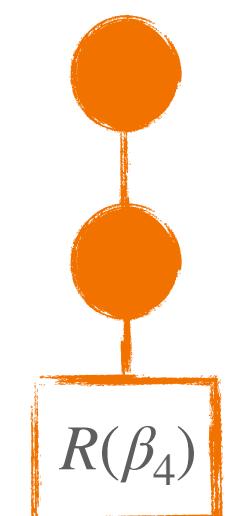
Rotation (X, Y ou Z)

$$data = \begin{bmatrix} 6 & -12.5 & 11.15 & 7 \\ 8 & 9.5 & -11 & -5 \\ 5 & 0.5 & 8 & -7 \end{bmatrix}$$

1 feature / qubit



Portes  
Multi-controle



$R(\beta_4)$

Ansatz

VQA

Temps linéaire par variable et qubits

# Hamiltonian Encoding

Dans la suite du "time-evolution"

Pour chaque matrice unitaire, il y a un vecteur réel tel que:

$$\mathbf{A} = \alpha + \beta\mathbf{X} + \delta\mathbf{Y} + \gamma\mathbf{Z}$$

$$e^{i\mathbf{At}} = e^{i(\alpha + \beta X + \delta Y + \gamma Z)t}$$

$$= e^{iatI} e^{i\beta Xt} e^{i\delta Yt} e^{i\gamma Zt}$$

$$= R_x(\beta t) R_y(\delta t) R_z(\gamma t)$$

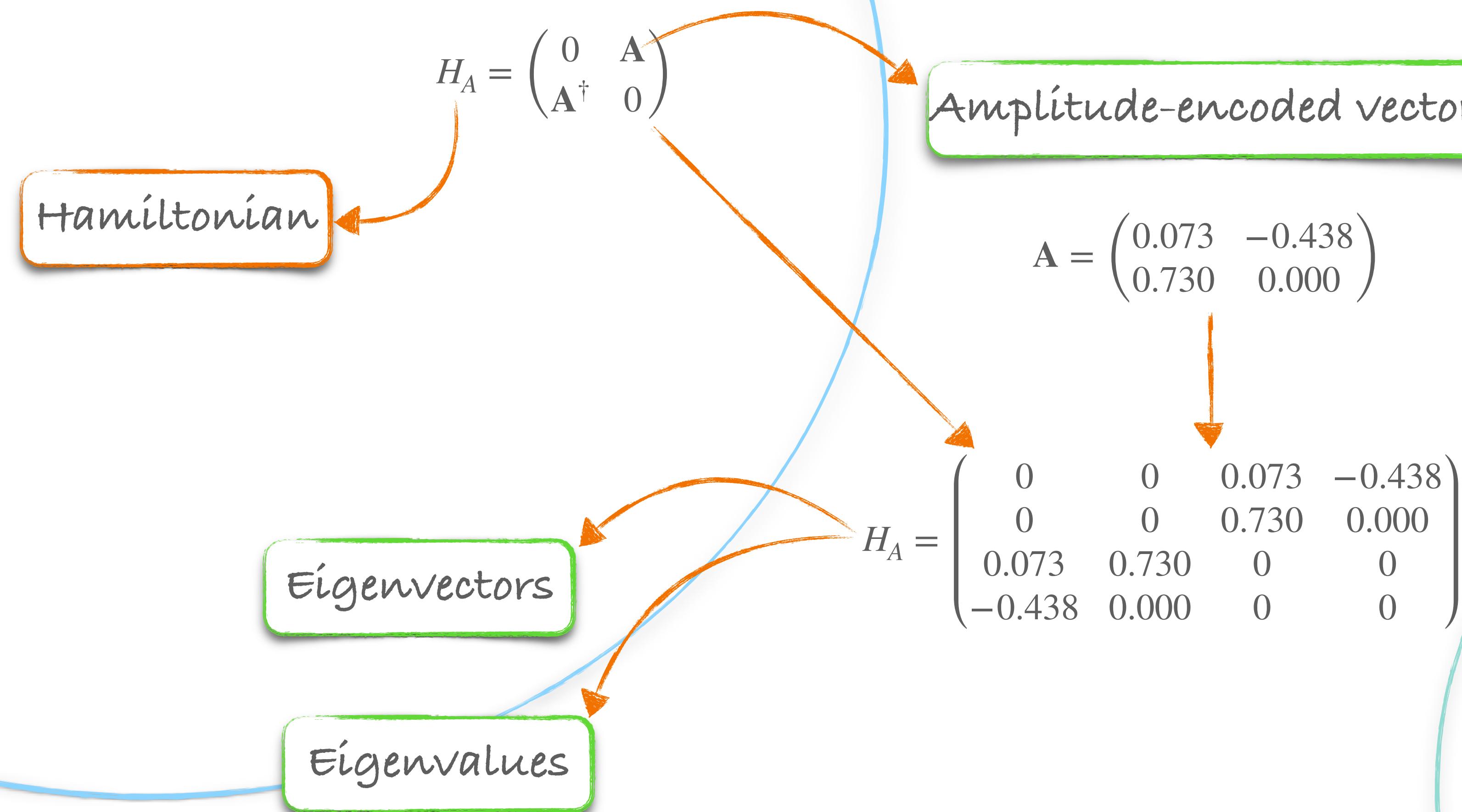
Pauli X

Pauli Y

Pauli Z

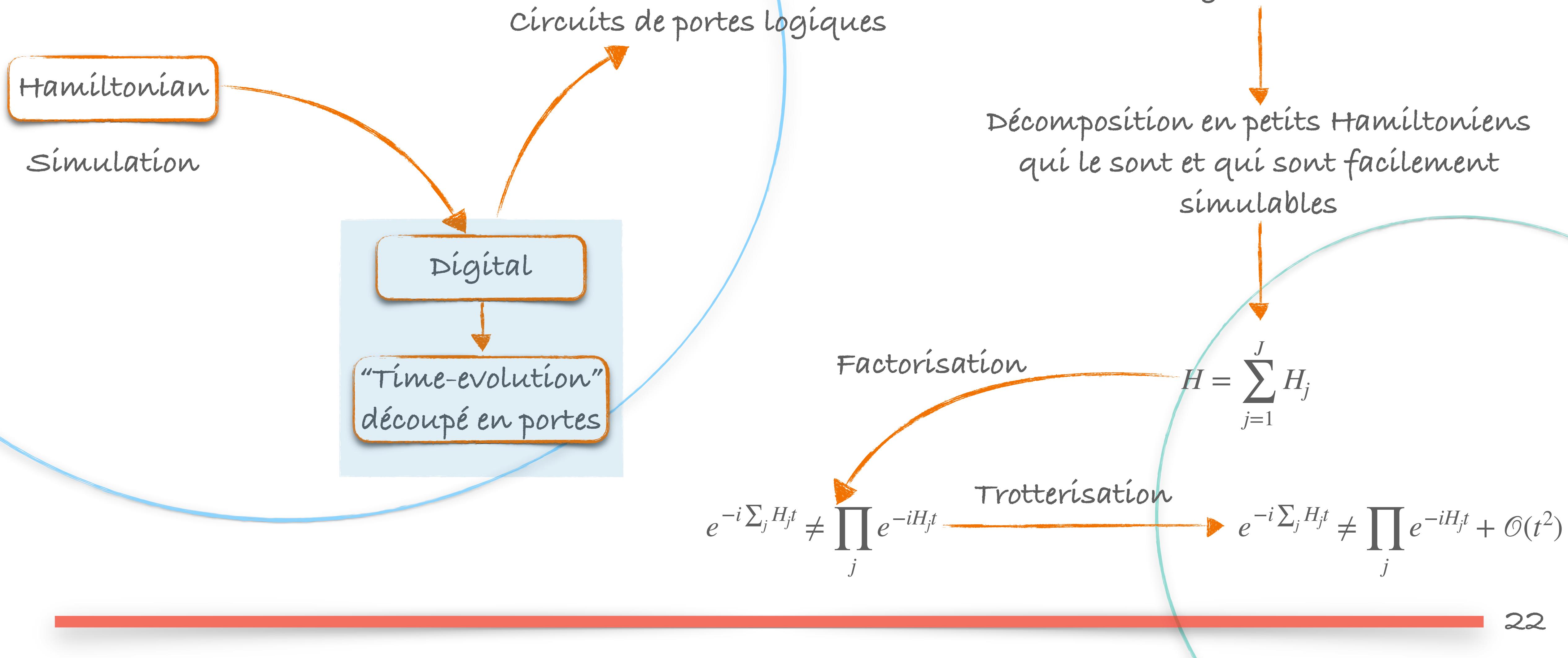
# Hamiltonian Encoding

Dans la suite du "time-evolution"



# Hamiltonian Encoding

Dans la suite du "time-evolution"



# Hamiltonian Encoding

Dans la suite du "time-evolution"

La plupart du temps  $H$  n'est pas diagonalisable

Décomposition en petits Hamiltoniens qui le sont et qui sont facilement simulables

Factorisation

$$H = \sum_{j=1}^J H_j$$

Permet d'approximer quand  $t$  est très petit

$$e^{-i \sum_j H_j t} \neq \prod_j e^{-i H_j t}$$

Trotterisation

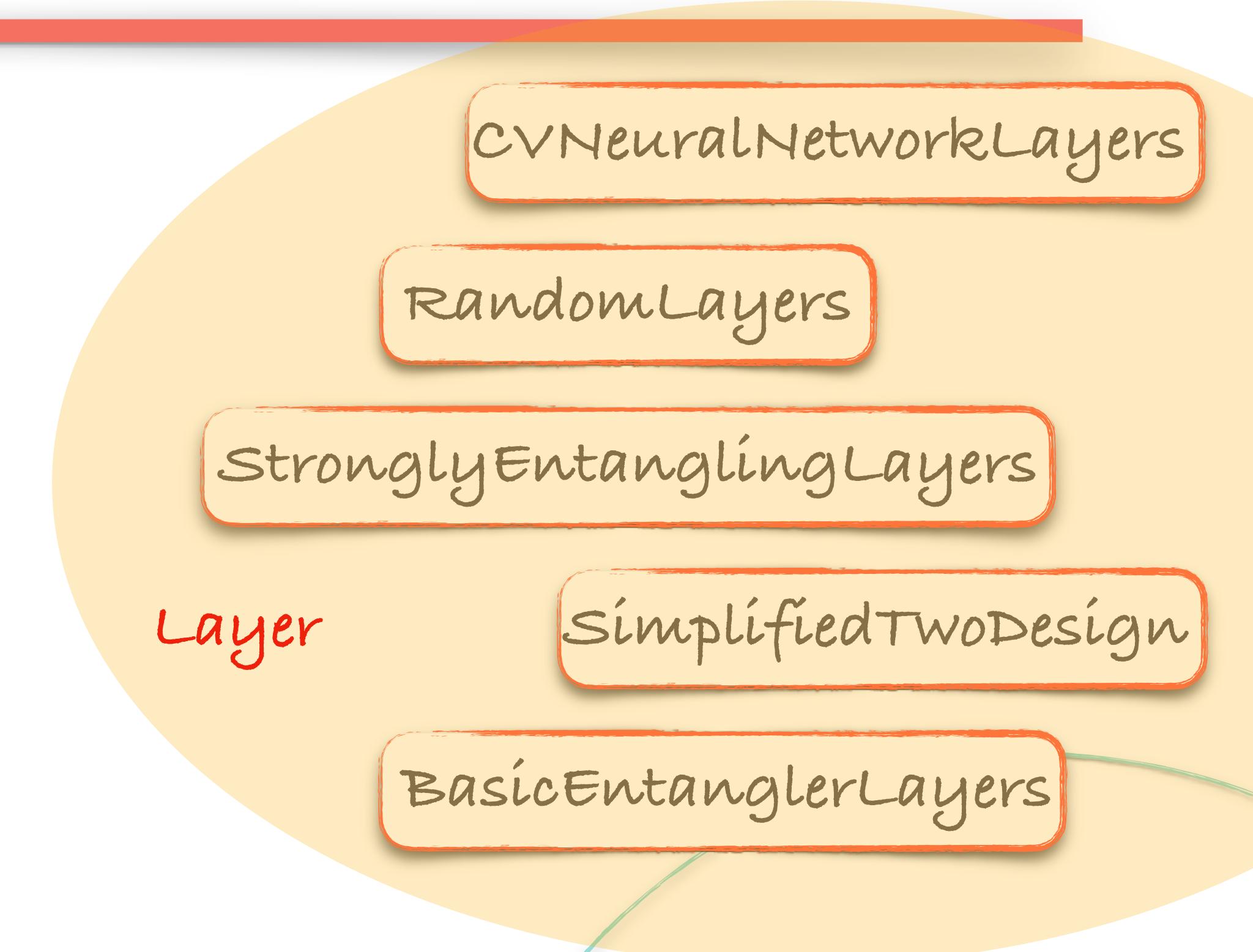
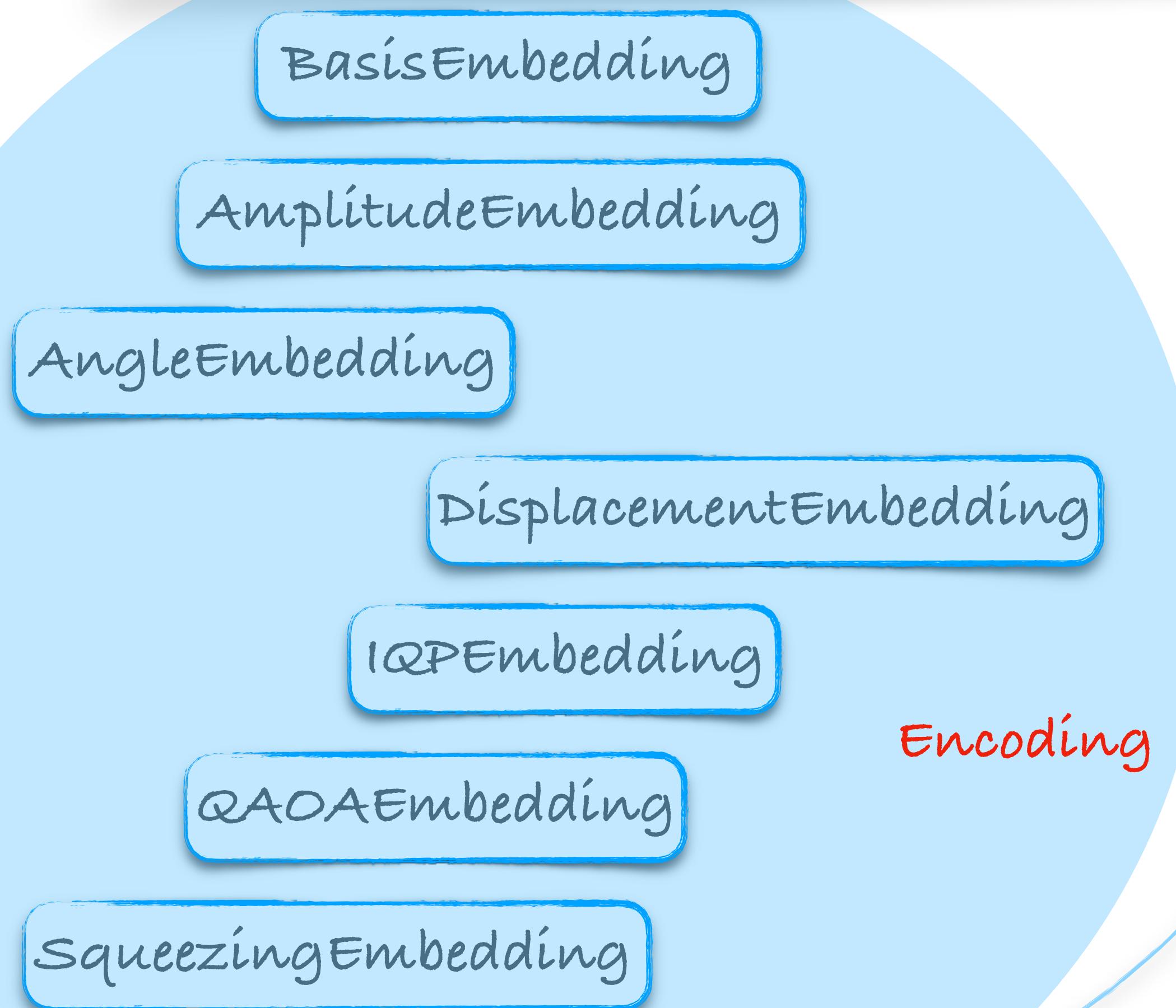
$$e^{-i \sum_j H_j t} \neq \prod_j e^{-i H_j t} + \mathcal{O}(t^2)$$

$$\begin{aligned} H &= X\text{I}\text{I}\text{I}\text{I} \\ &+ \text{I}X\text{I}\text{I}\text{I} \\ &+ \text{I}\text{I}X\text{I}\text{I} \\ &+ \text{I}\text{I}X\text{Z}\text{I} \\ &+ \text{I}\text{I}\text{I}\text{I}\text{Y} \end{aligned}$$

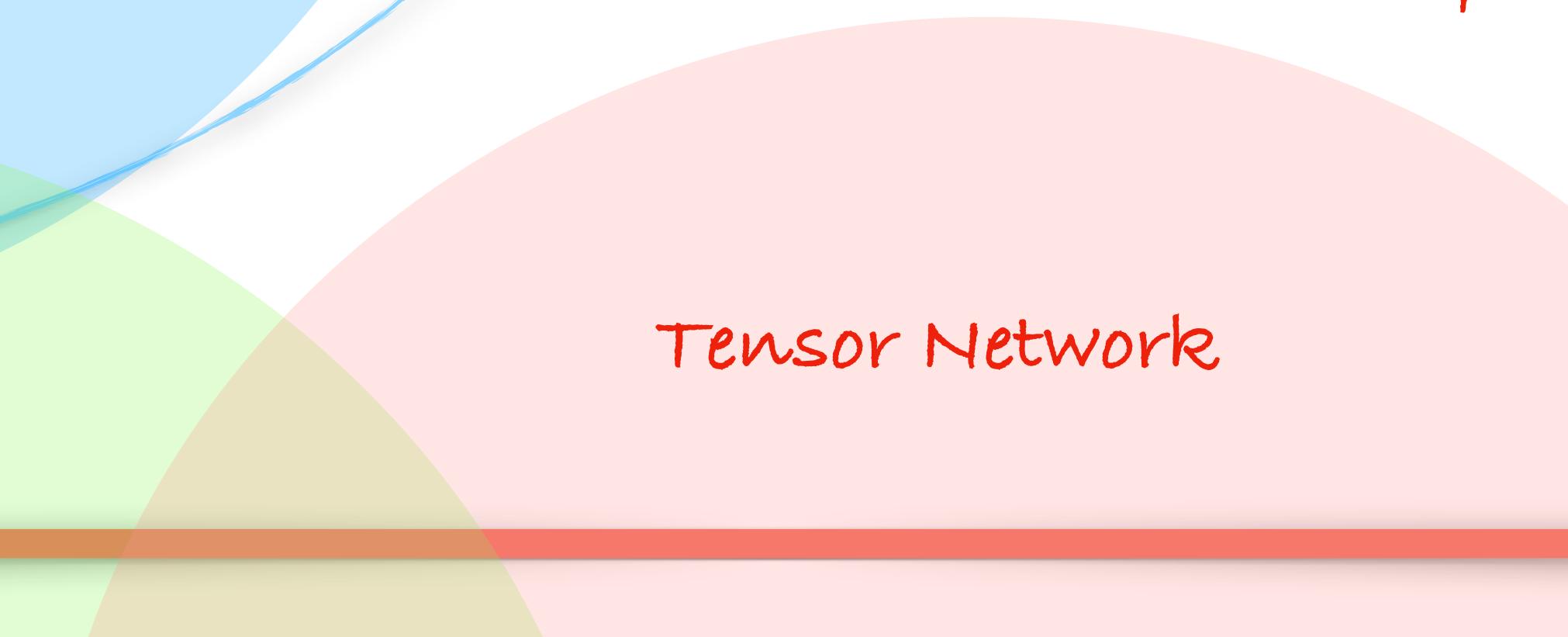
Opérateurs de Pauli

Équivaut à ramener le problème à un problème local avec peu d'interactions

# Autres Encodages?



Et plein d'autres...

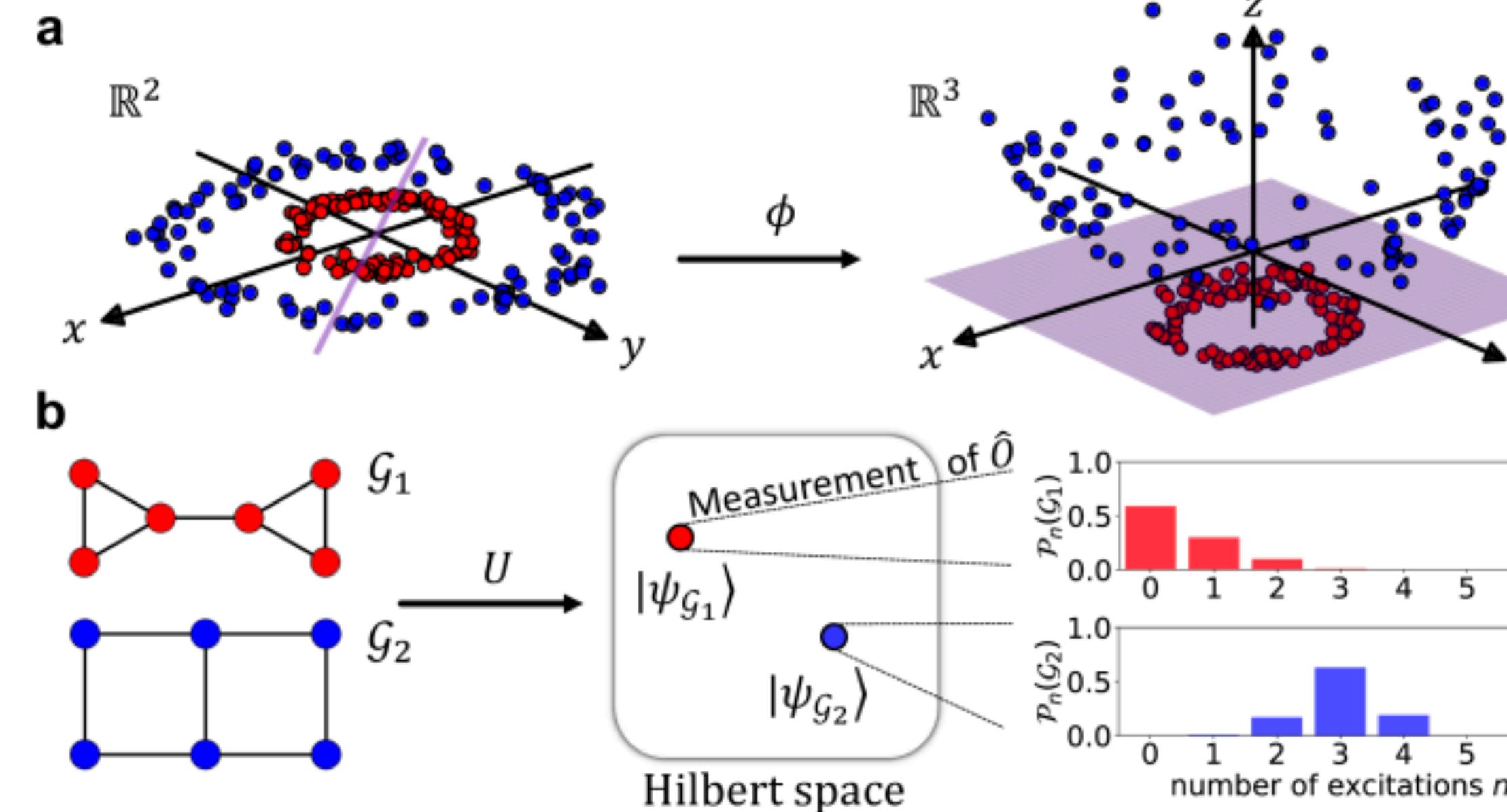


# Quantum Feature Maps

Données dans un espace 2D

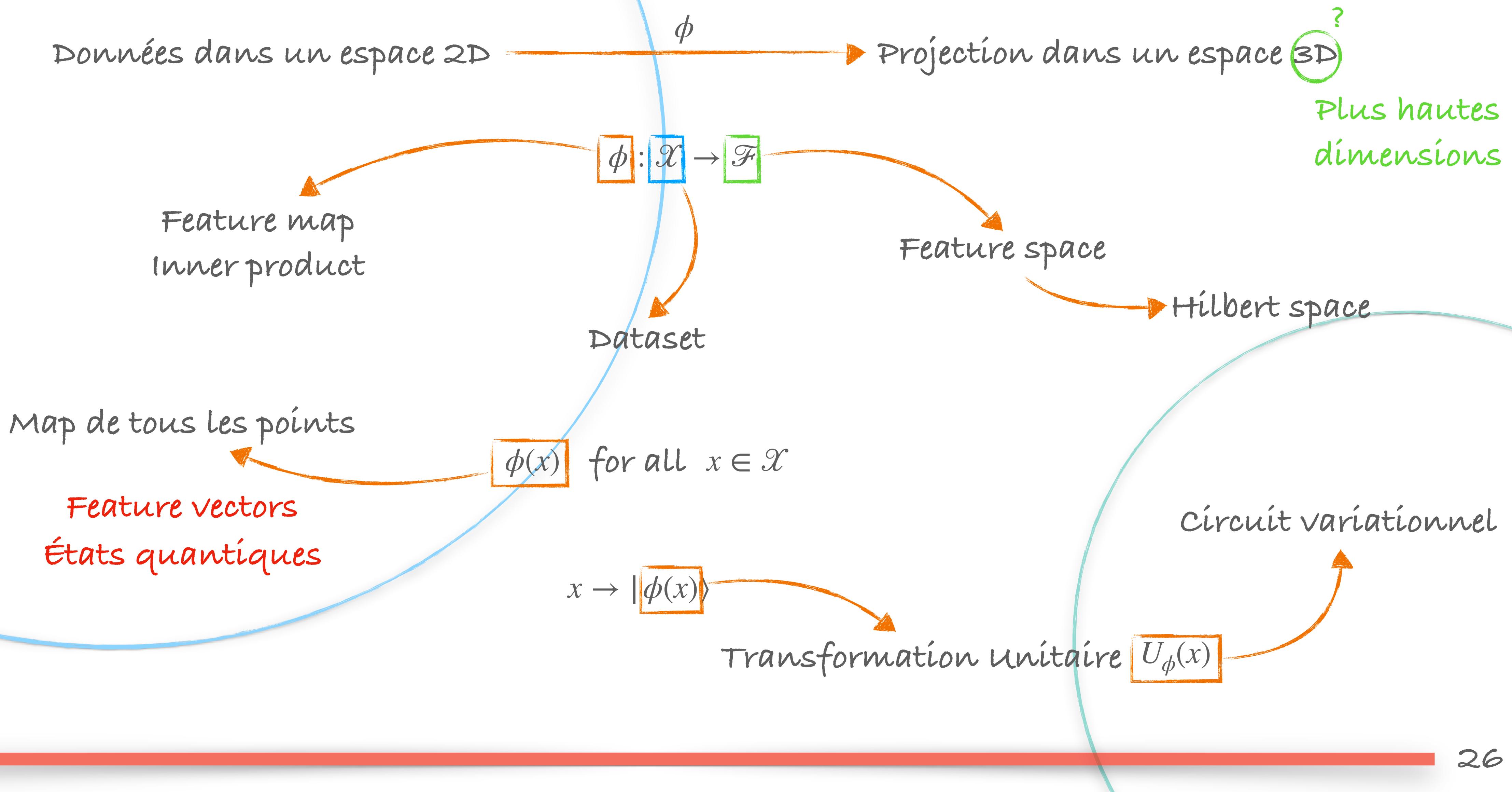
$\phi$

Projection dans un espace 3D

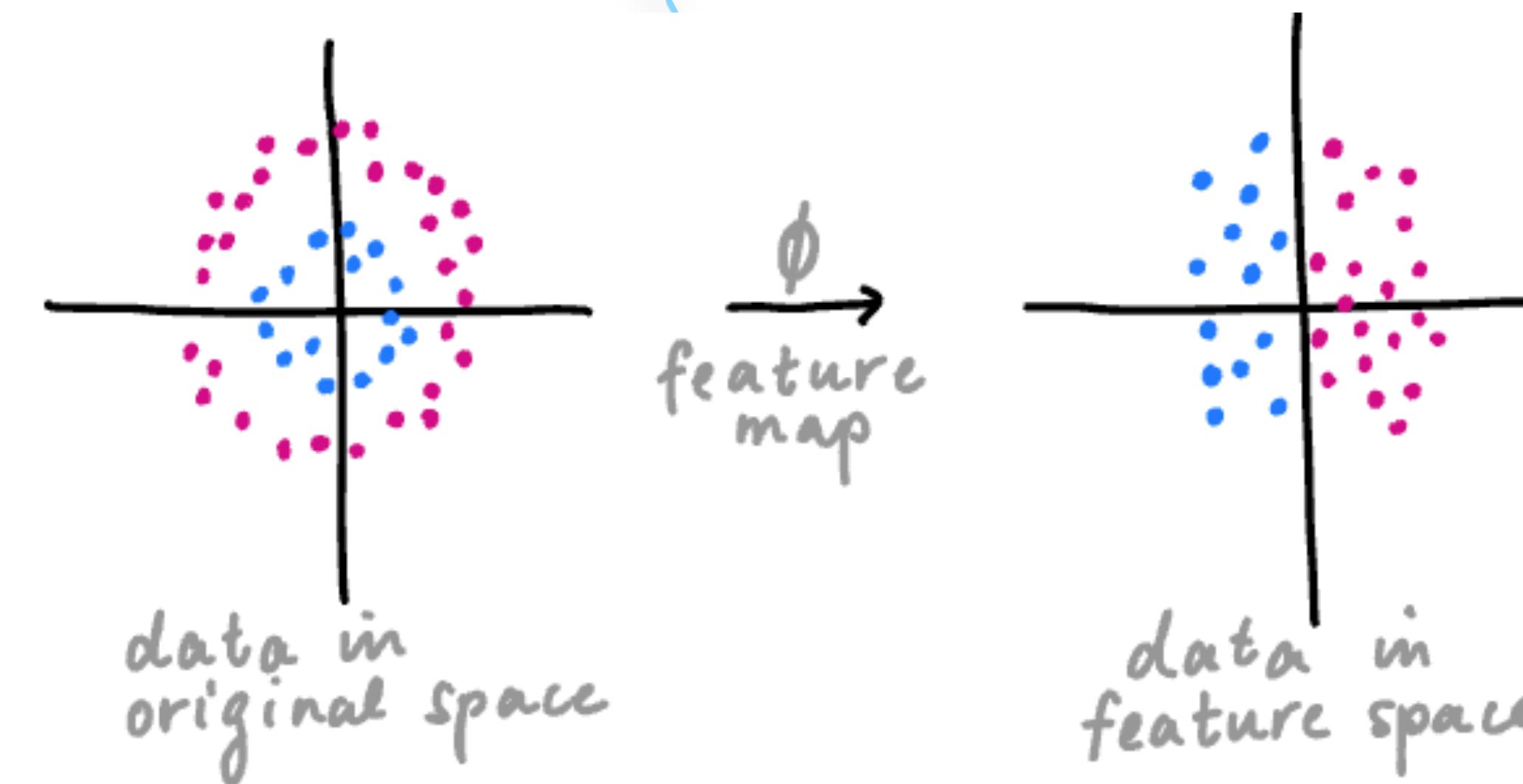


Principe de transformation dans un espace à plus haute dimension

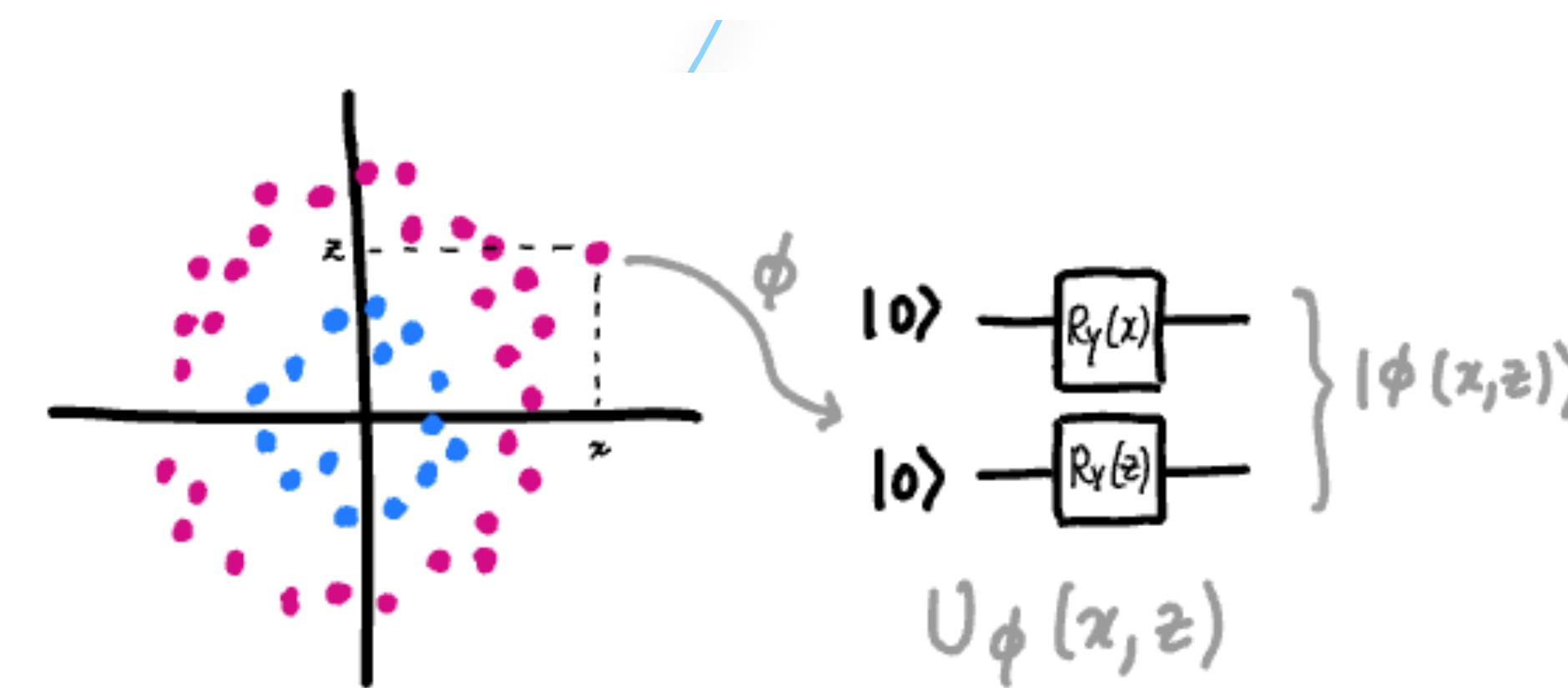
# Quantum Feature Maps



# Quantum Feature Maps

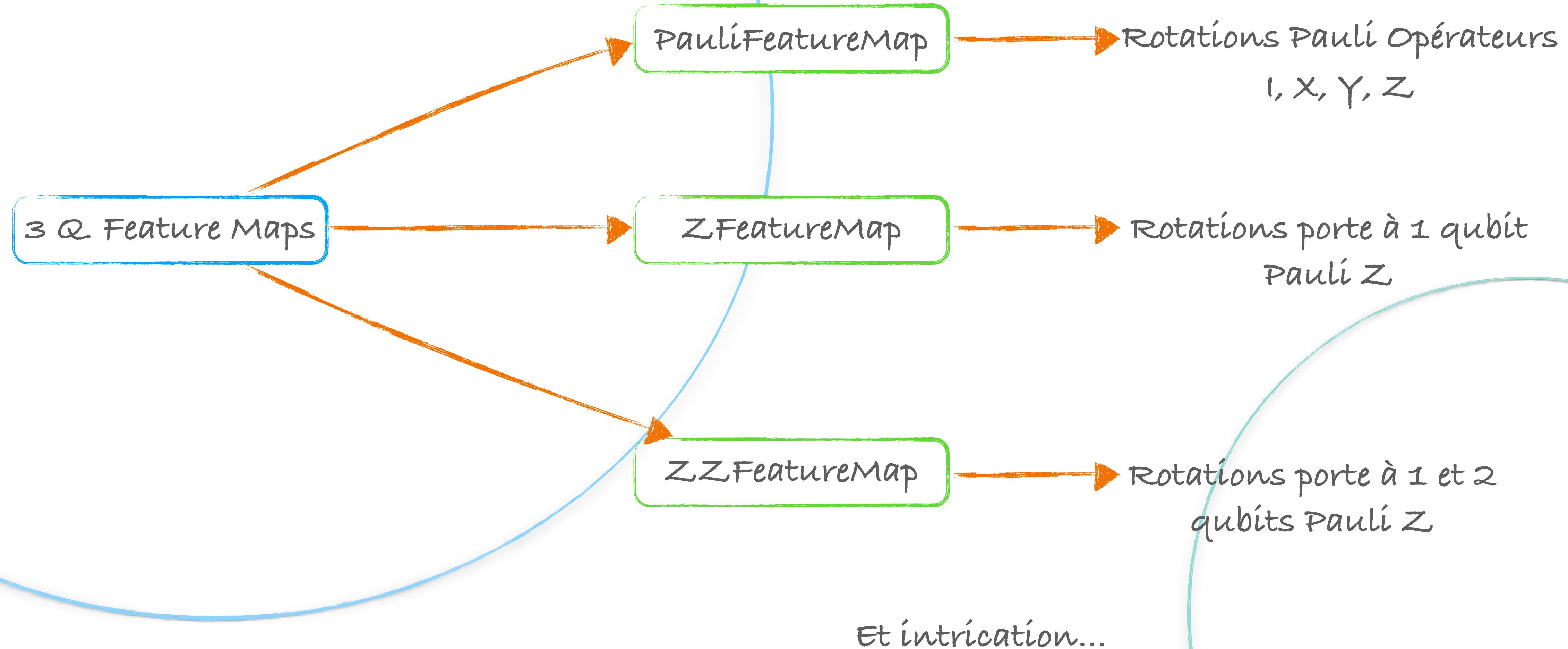


A feature map can transform data into a space where it is easier to process.



# Quantum Feature Maps

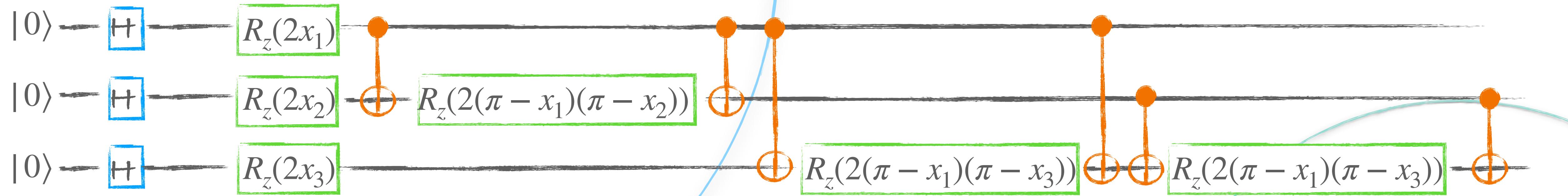
Qiskit



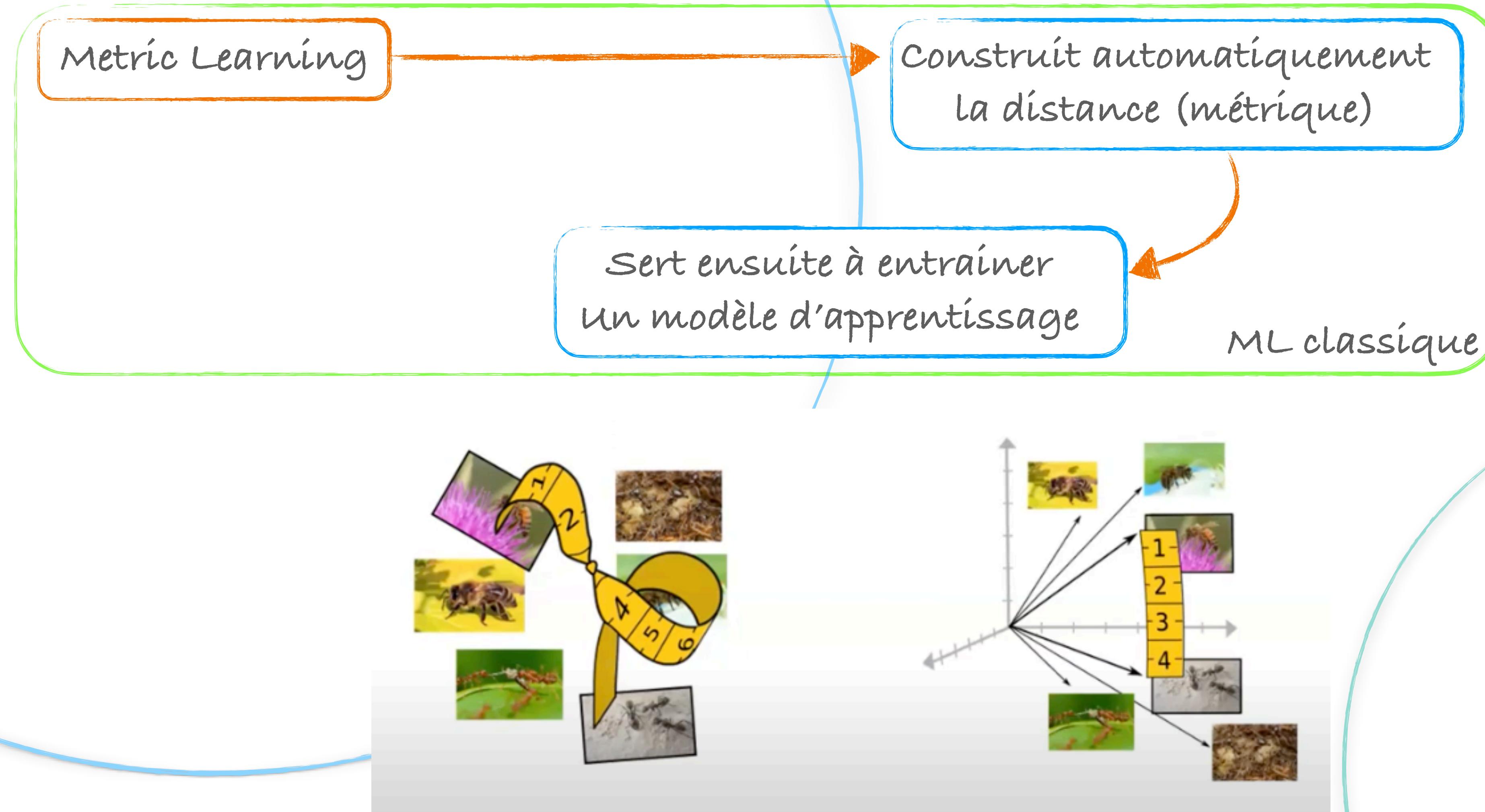
# Quantum Feature Maps

ZZFeatureMap

3 qubits 3 features  
1 rep  
Full entanglement



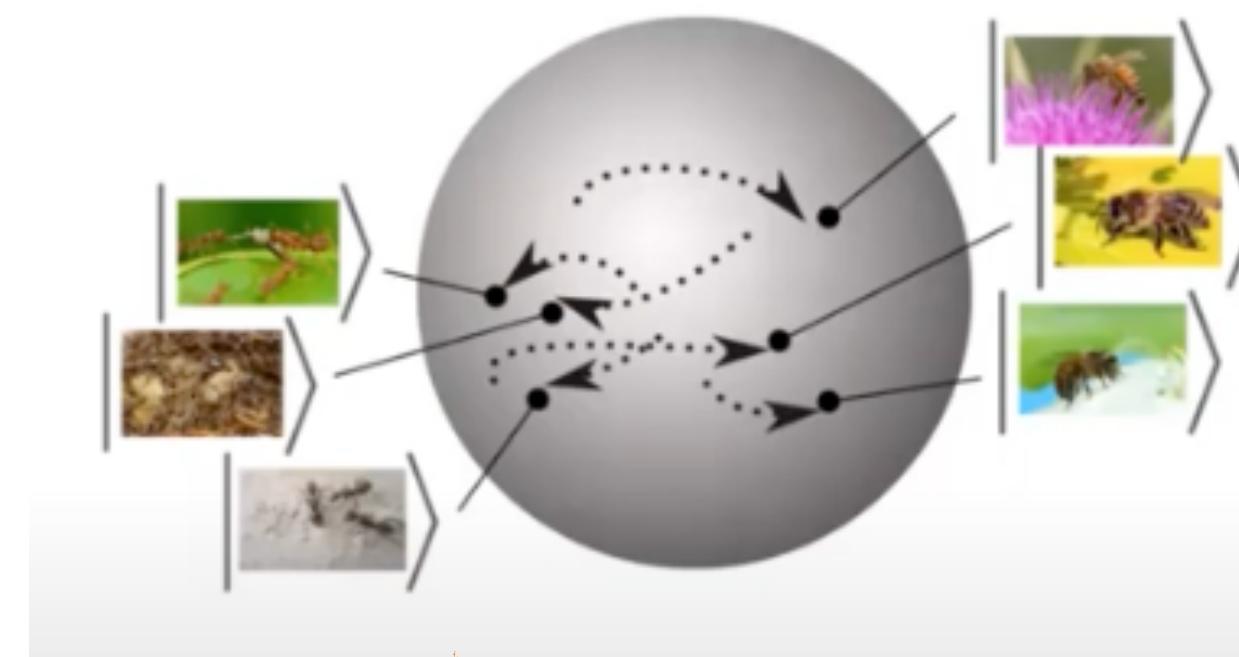
# Quantum Metric Learning



# Quantum Metric Learning

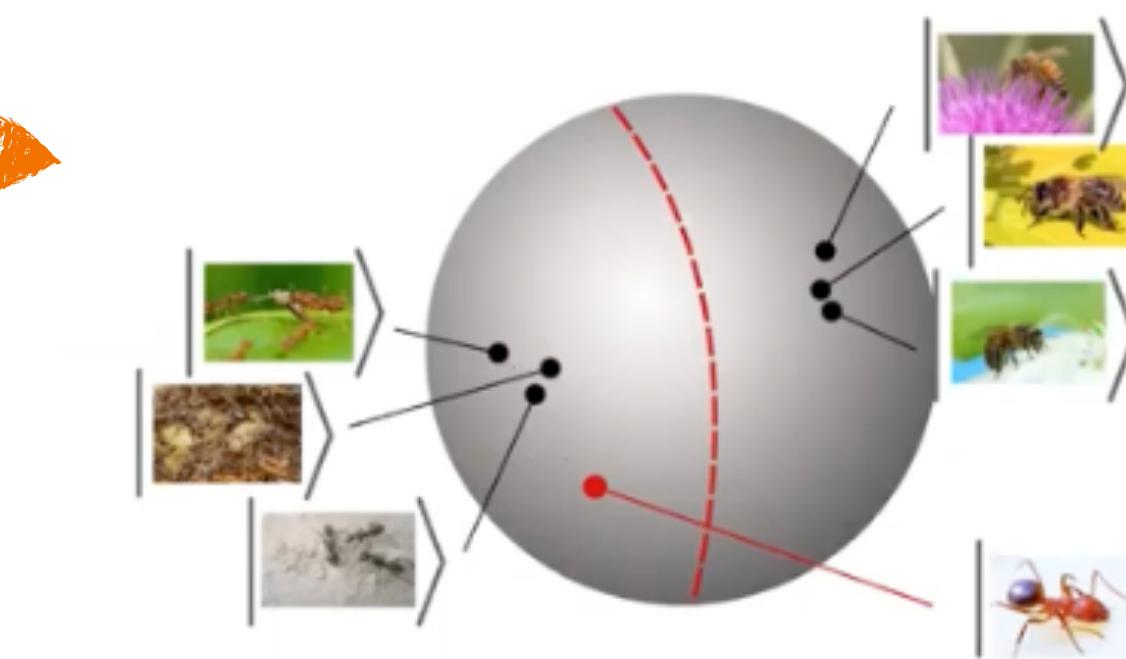
Quantum Metric Learning

Random states pour encoder  
Les données



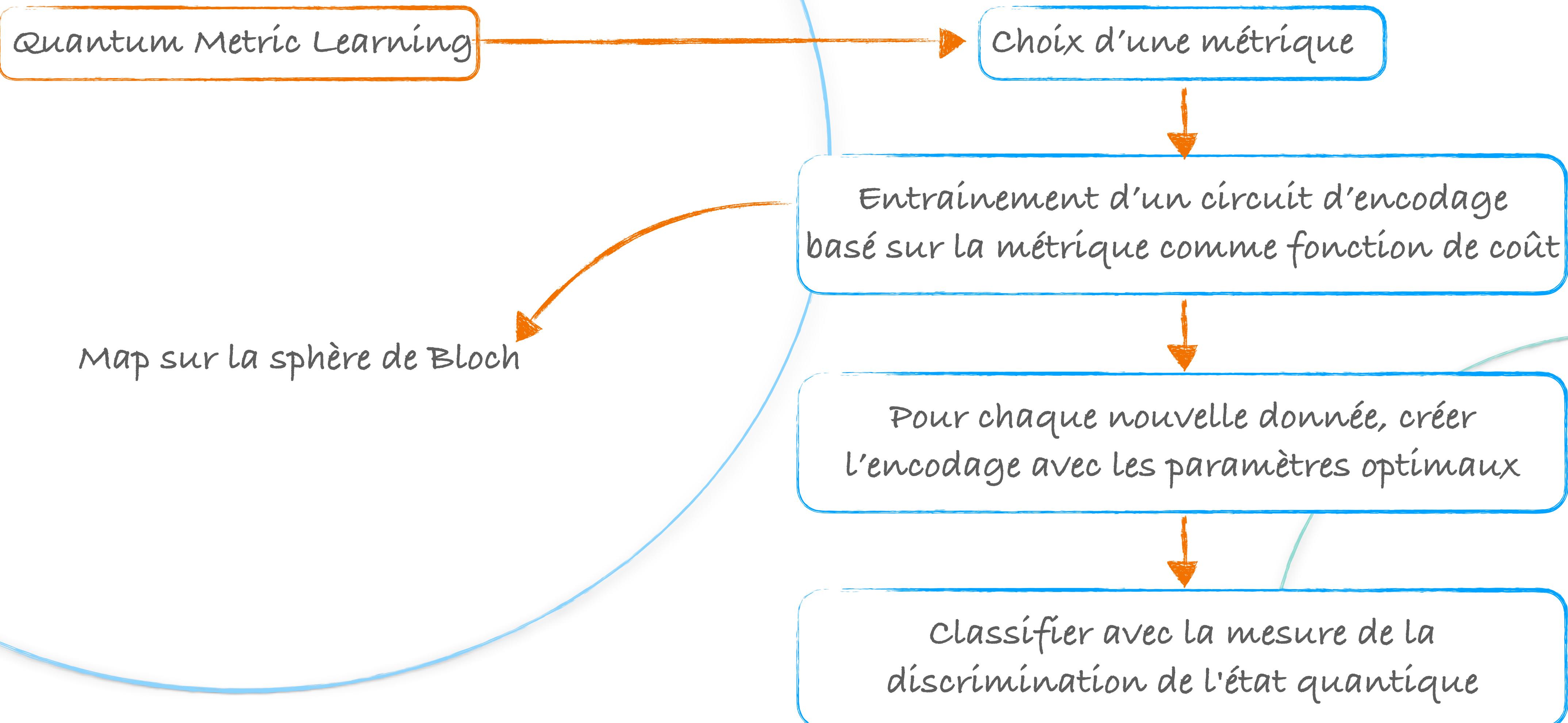
Patterns? Differences?

Apprentissage variationnel



L'algorithme apprend à séparer les embeddings  
selon leur distance ou similarité

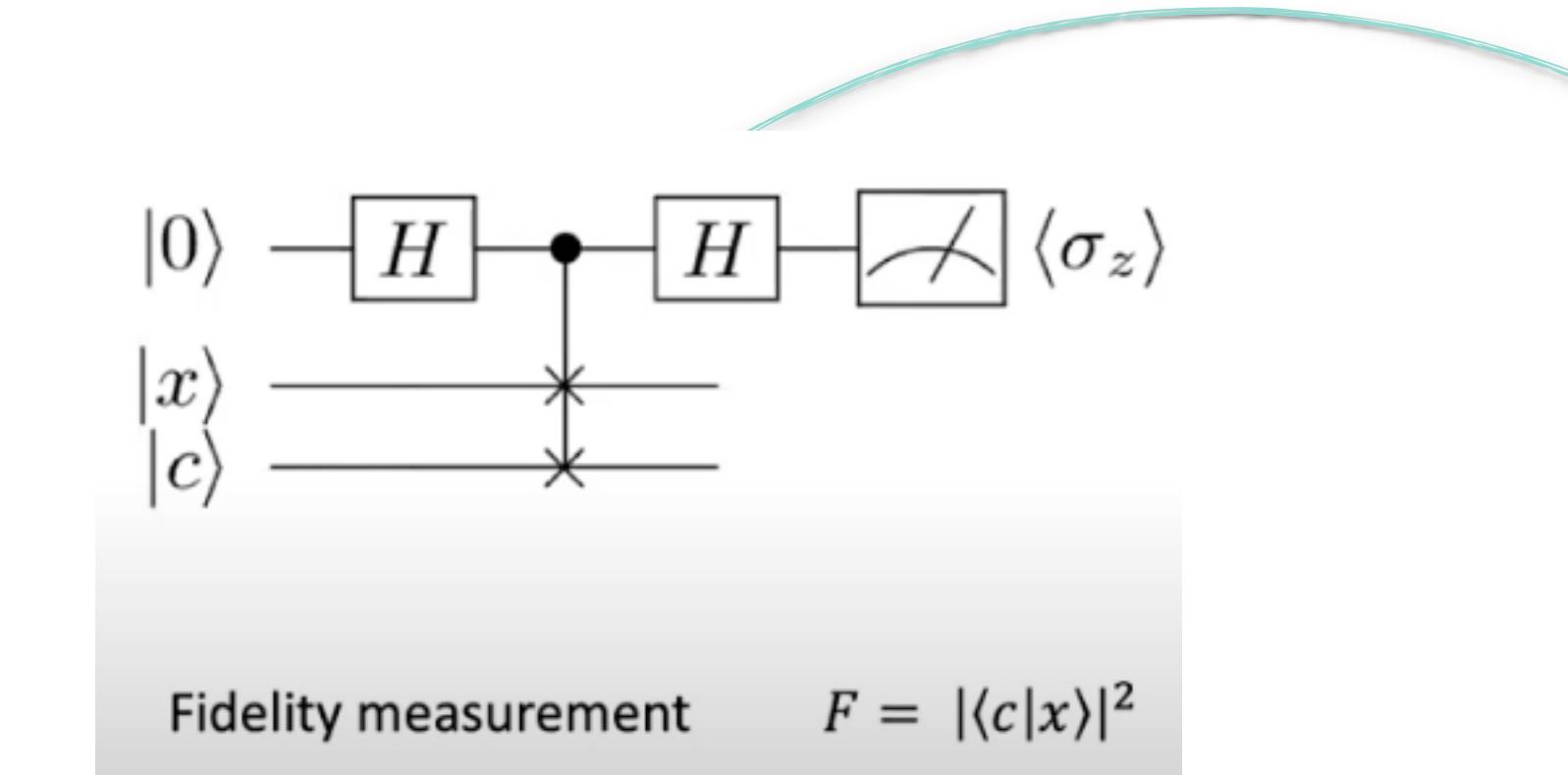
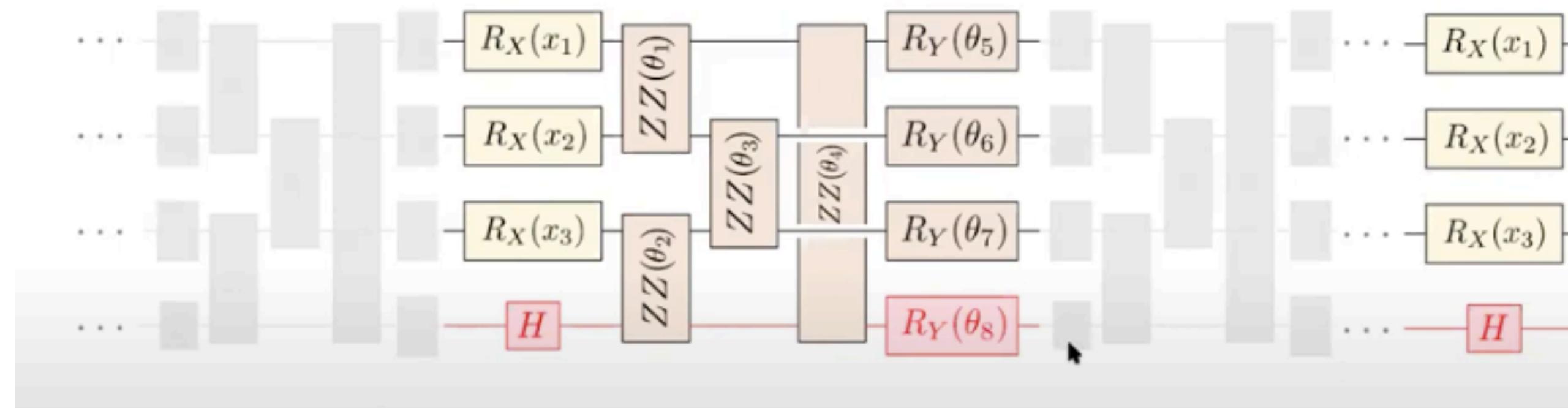
# Quantum Metric Learning



# Quantum Metric Learning

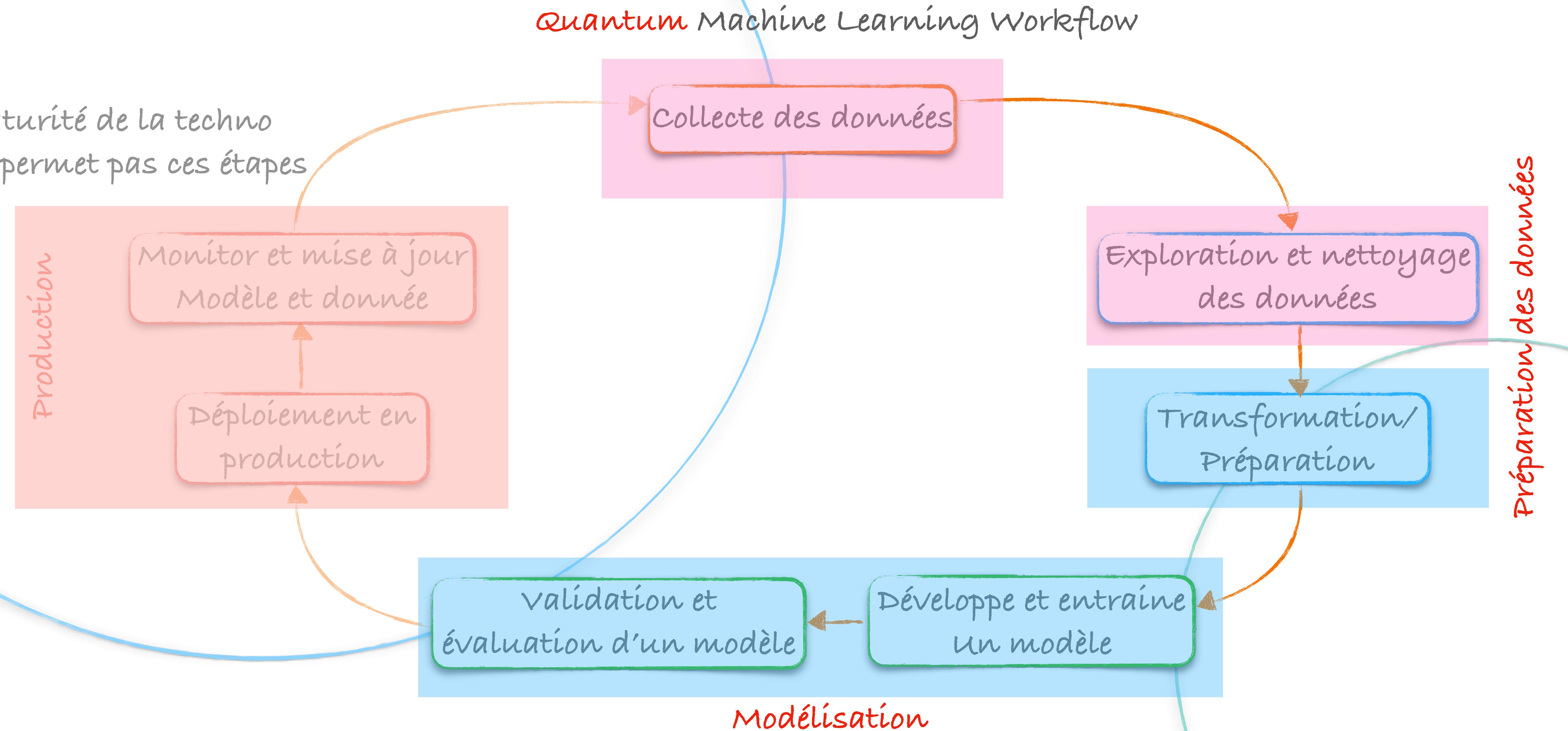
Quantum Metric Learning

QAOA Embedding Ansatz



# Pourquoi est-ce essentiel?

Maturité de la techno  
ne permet pas ces étapes



# Références

---

- [1] Lloyd et al., 2020, Quantum embeddings for machine learning, <https://arxiv.org/pdf/2001.03622.pdf>
- [2] Quantum Embeddings by Aroosa Ijaz - <https://www.youtube.com/watch?v=mNR-70millo>
- [3] <https://towardsdatascience.com/the-machine-learning-workflow-explained-557abf882079>
- [4] Albrecht et al. 2023, Quantum Feature Maps for Graph Machine Learning on a Neutral Atom Quantum Processor,  
<https://www.arxiv-vanity.com/papers/2211.16337/>
- [5] Quantum embedding - Pennylane blog - [https://pennylane.ai/qml/glossary/quantum\\_embedding/](https://pennylane.ai/qml/glossary/quantum_embedding/)