Atos QLM Fast-Start

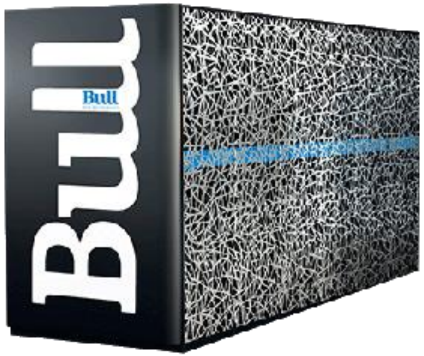
Introduction to quantum computing

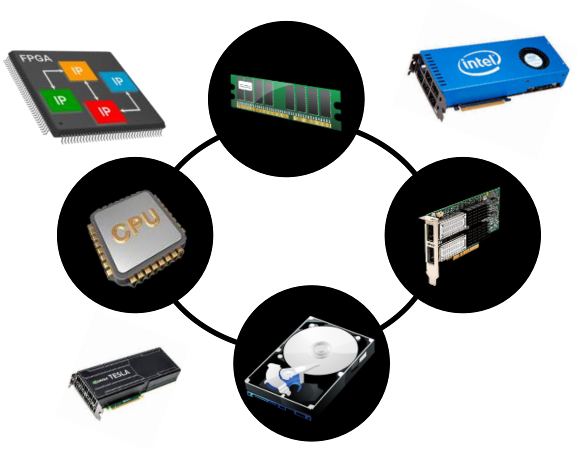
2021/12/02

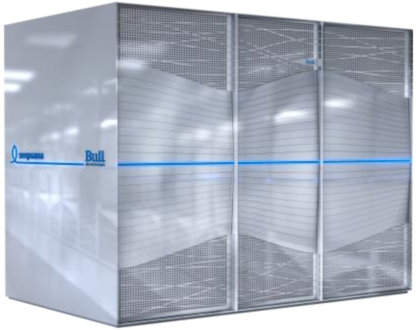
Gaëtan Rubez
Quantum Computing Expert for the CEPP
gaetan.rubez@atos.net

Trusted partner for your **Digital Journey**

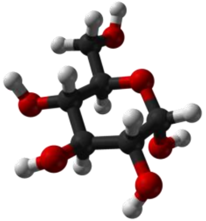
HPC, Driving innovation







CyberSecurity



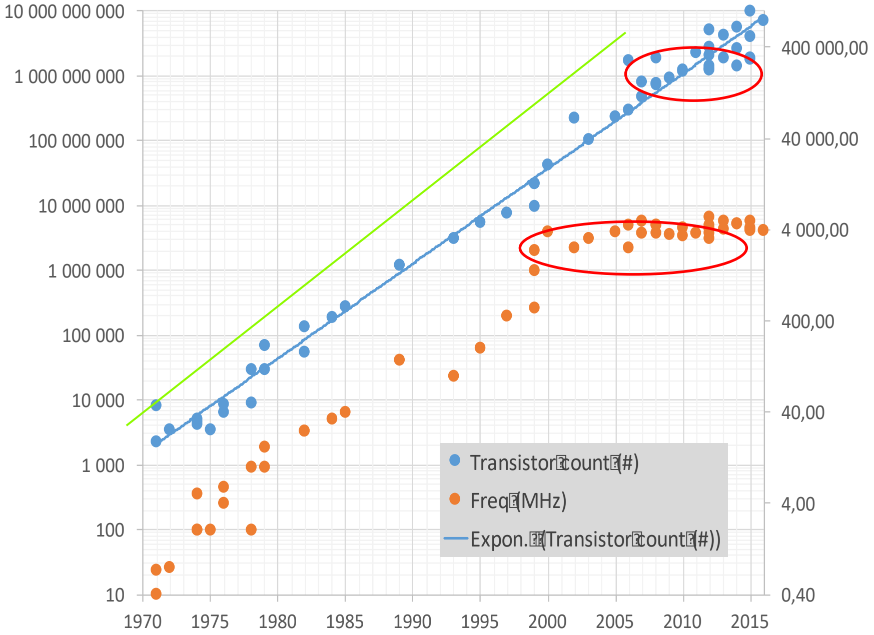
Medical, Chemistry

Weather Forecast

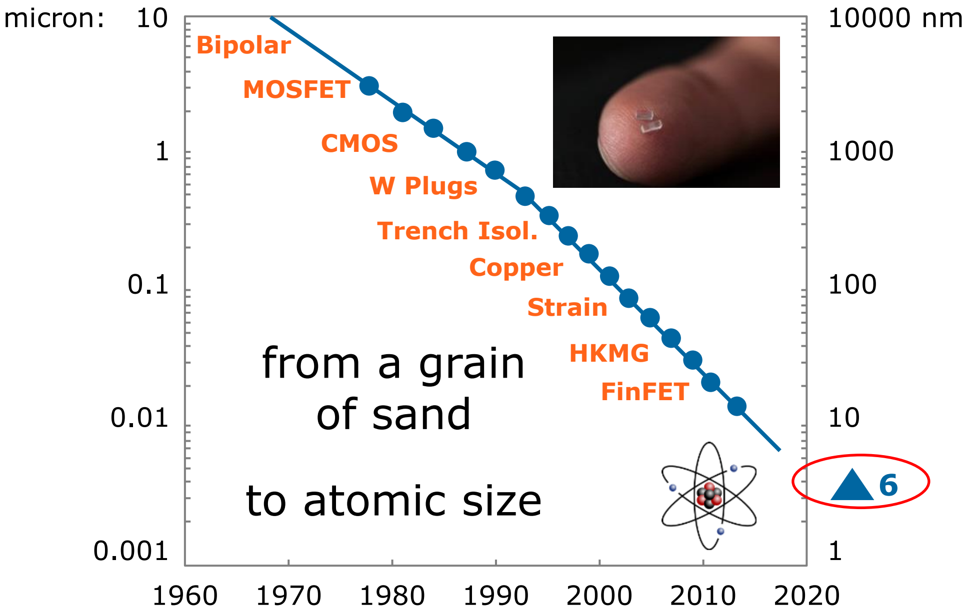
Artificial Intelligence

End Of Moore’s Law

More transistors, higher frequencies



New technologies for thinner chips



Second Quantum revolution

First Quantum revolution

☑

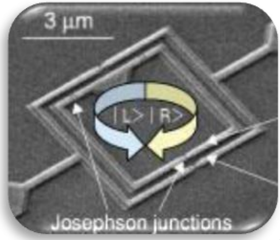
Quantum Processors

Second Quantum revolution

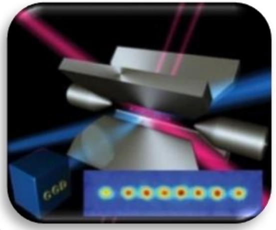
IN PROGRESS

Quantum Hardware Technologies

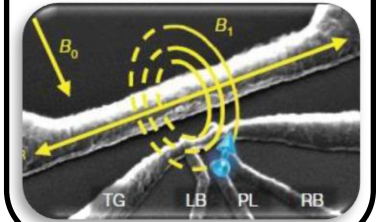
Superconducting loops



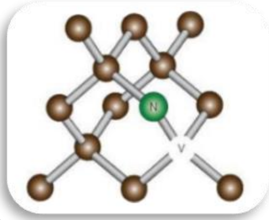
Ion traps



Silicon spin qubits

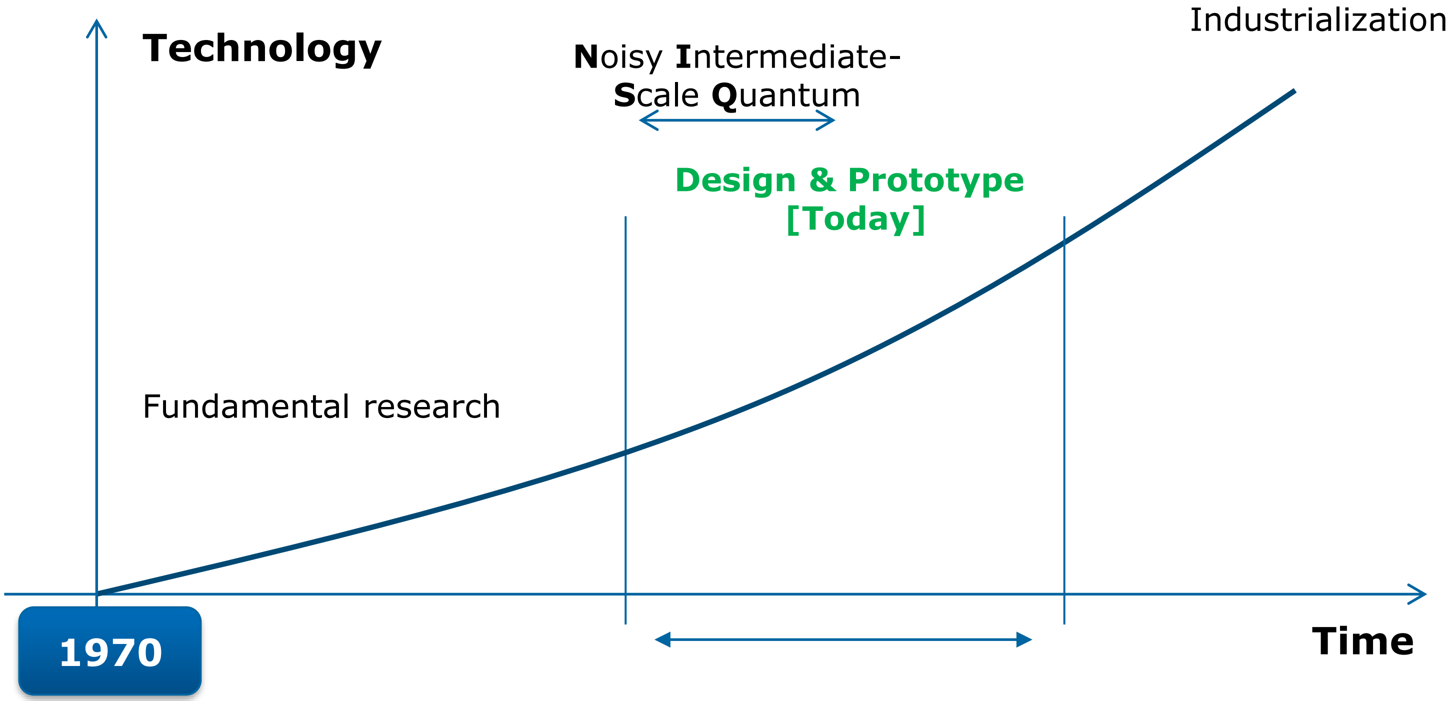


Diamond vacancies

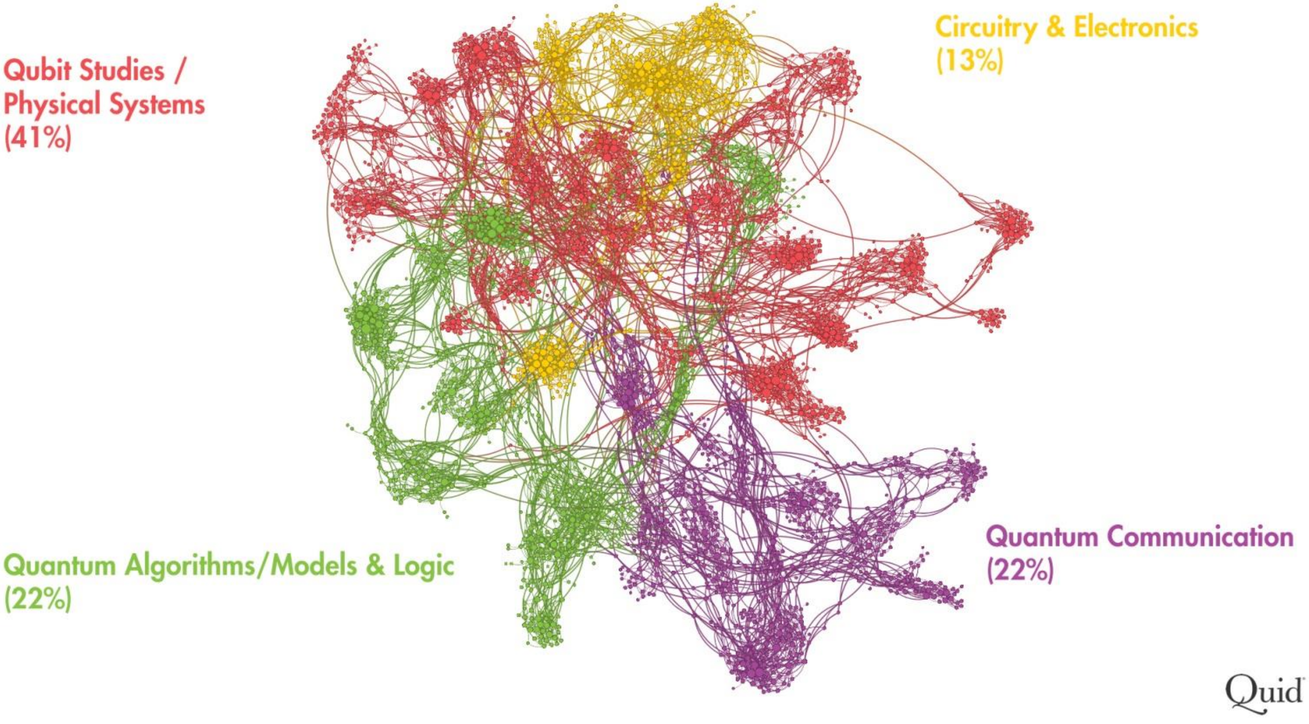


Topological qubits

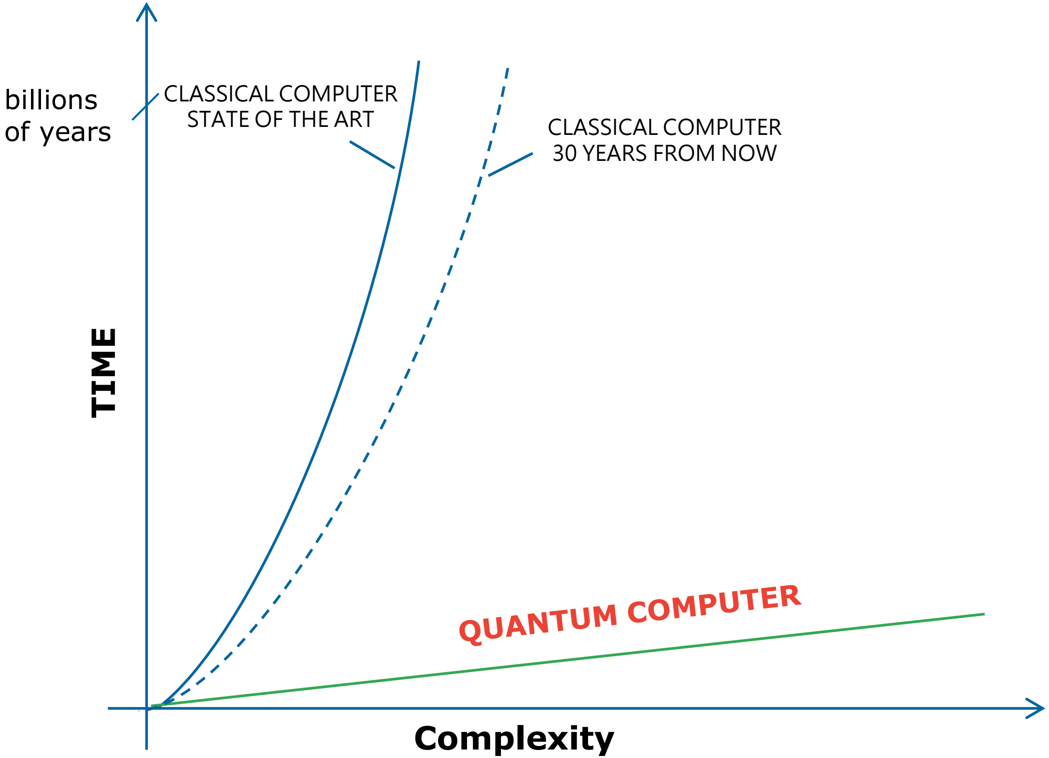
Real Quantum technologies



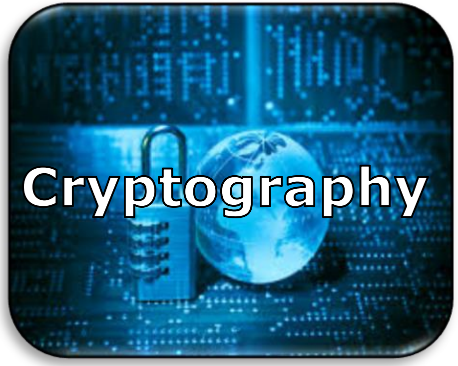
Quantum computing research areas

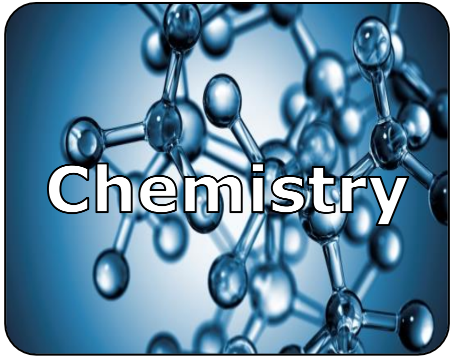


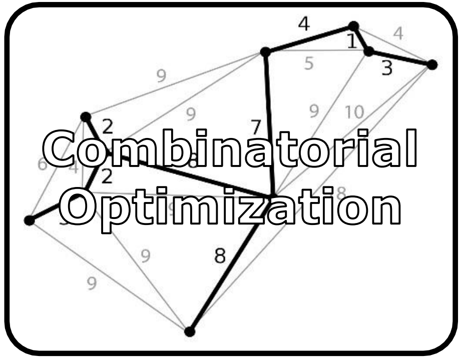
Quantum Computing Speedup

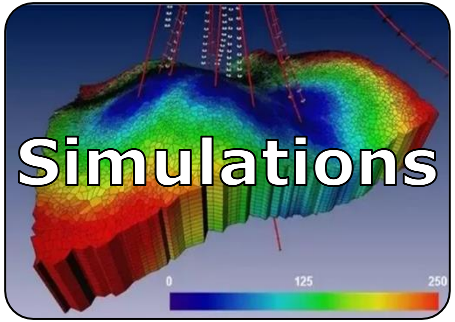


Quantum Speedup expected

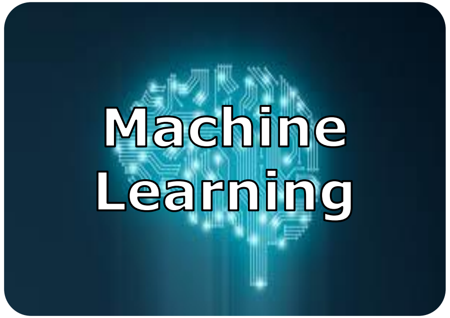

Cryptography


Chemistry


Combinatorial Optimization


Simulations


Finance


Machine Learning

The Atos Quantum learning machine

Stanford University

Oak Ridge National Laboratory

Argonne National Laboratory

Total

cea

ccrt

Université de Reims Champagne-Ardenne

CSC

DTU

Hartree Centre
Science & Technology Facilities Council

Quantum Learning machines
from 30 to 40 qubits
simulation power capabilities

Success around the world

Jülich Forschungszentrum

lrz

Bayer

fh Oberösterreich
University of Applied Sciences Upper Austria

SURF SARA

सी-डैक
CDAC
Center for Development of Advanced Computing

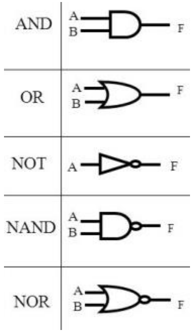
Classical and Quantum Computing

Classical Computer

bits (0,1)

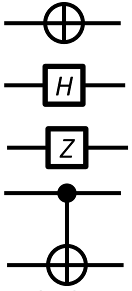
Logic, ***boolean operators***,

to represent ***boolean gates***



Output 01011010001…

Quantum Computer



CNOT

quantum bits : qubits (0 and 1)

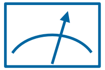
qubit is a quantum system with 2 states

Linear algebra, vectors and matrices

to represent ***quantum gates***

(e.g; NOT, Hadamard, phase shift, CNOT)

Measurement



Superposition and measurement

- 1 classical bit: 0 or 1, white or black
- 1 quantum bit (qubit): superposition of white |0> and black |1>

Measurement

Initialize your qubit :
for example we choose to be in
30% white i.e |0>
70% black, i.e |1>

The qubit is in a gray state

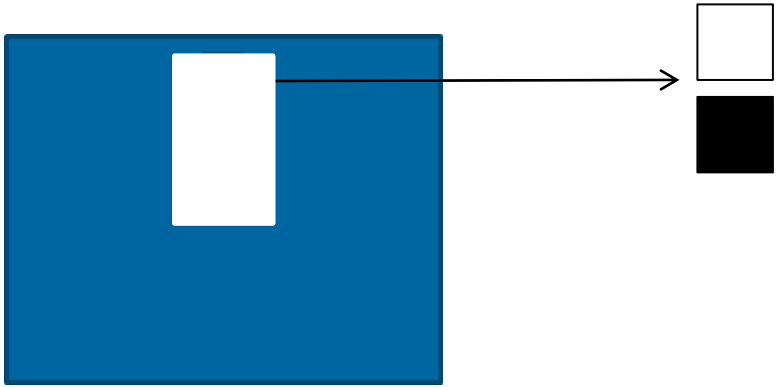
Don't look at it

When you look at it, the qubit
takes the color seen

If you want to get another result :
try again from the beginning

Superposition and measurement

- 1 classical bit: 0 or 1, white or black
- 1 quantum bit (qubit): superposition of white |0> and black |1>



Measurement

you see white in 30% of the cases

you see black in 70% of the cases

Superposition

Vector Notation

A *single qubit* is a **complex vector**:

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

Superposition & Measurement

Vector Notation

A *single qubit* is a **complex vector**:

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

But a *measurement* only gives **a 0 or a 1**

with a certain *probability:*

Superposition & Measurement

Vector Notation

A *single qubit* is a **complex vector**:

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

But a *measurement* only gives **a 0 or a 1** with a certain *probability:*

0 with probability $|\alpha|^2$

1 with probability $|\beta|^2$

Superposition & Measurement

Vector Notation

A *single qubit* is a **complex vector**:

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

But a *measurement* only gives **a 0 or a 1**

with a certain *probability:*

0 with probability $|\alpha|^2$

1 with probability $|\beta|^2$

so: $|\alpha|^2 + |\beta|^2 = 1$

Max Born in 1926

Computational basis

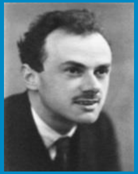
Vector Notation

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Computational basis for one qubit

Computational basis

Dirac notation



$$\begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \alpha\begin{pmatrix}1\\ 0\end{pmatrix} + \beta\begin{pmatrix}0\\ 1\end{pmatrix}$$

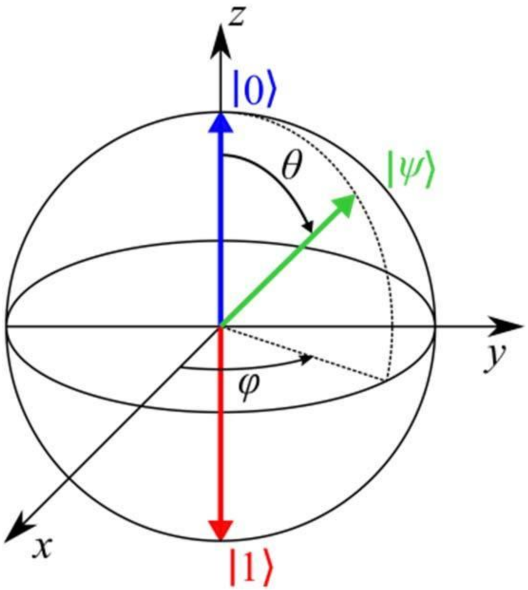
Computational basis for one qubit

$$\begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \alpha|0 > + \beta|1 >$$

« ket » notation

Single qubit representation

Bloch sphere



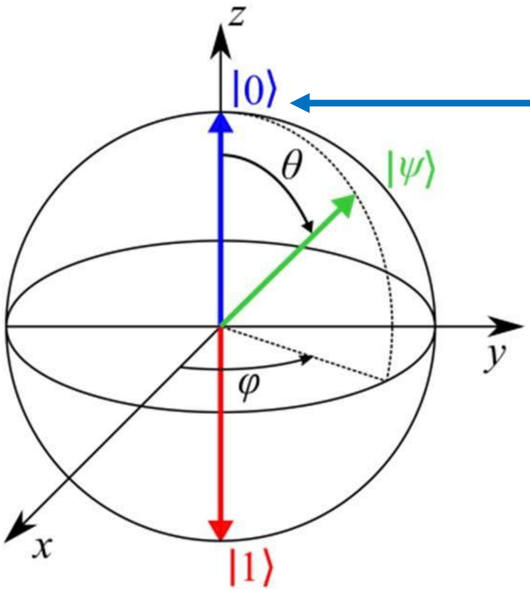
author: Fabio Sebastiano

$$|\psi> = \alpha\,|0> \; + \beta\,|1>$$

Single qubit representation

Bloch sphere

$|\psi> = \alpha\ |0>\ + \beta\ |1>$



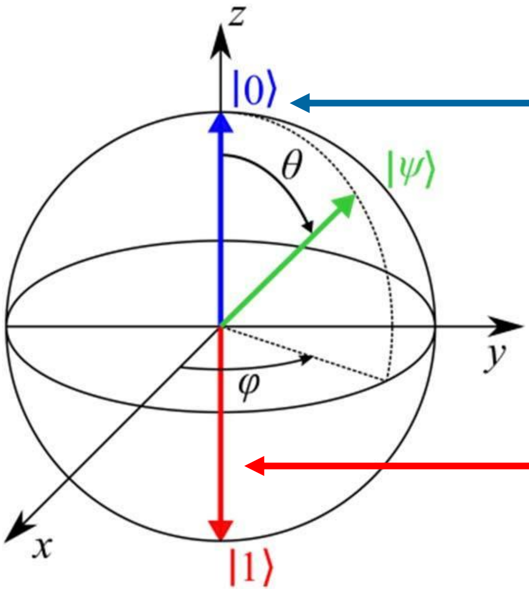
Ground state

author: Fabio Sebastiano

Single qubit representation

Bloch sphere

$$|\psi > = \alpha\,|0 > + \beta\,|1 >$$



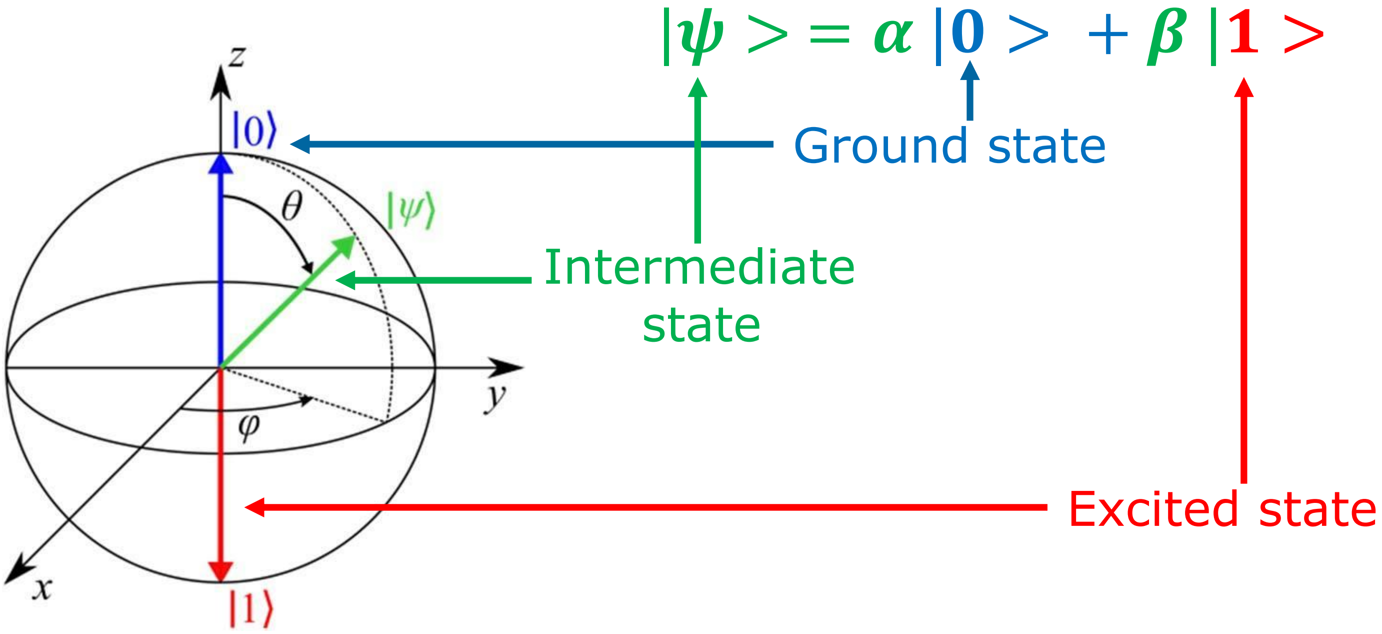
Ground state

Excited state

author: Fabio Sebastiano

Single qubit representation

Bloch sphere



$$|\psi> = \alpha\,|0> + \beta\,|1>$$

Ground state

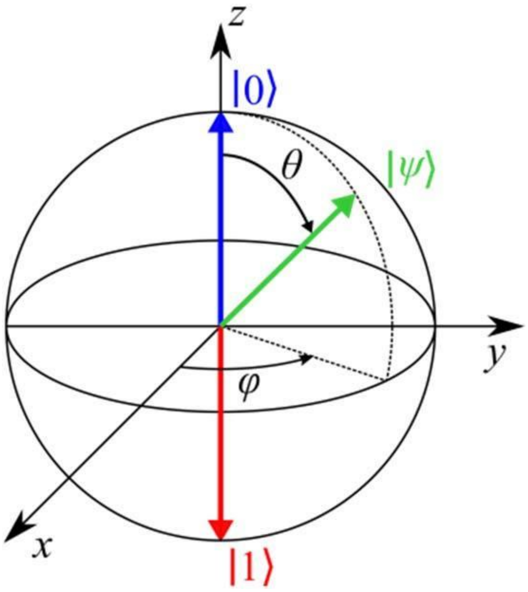
Intermediate state

Excited state

author: Fabio Sebastiano

Linking angles and vector

Bloch sphere



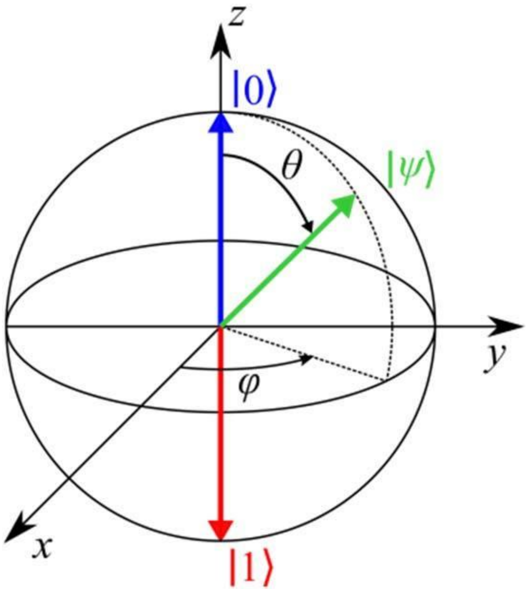
author: Fabio Sebastiano

$$|\psi> = \alpha\,|0> \; + \beta\,|1>$$

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

Linking angles and vector

Bloch sphere



author: Fabio Sebastiano

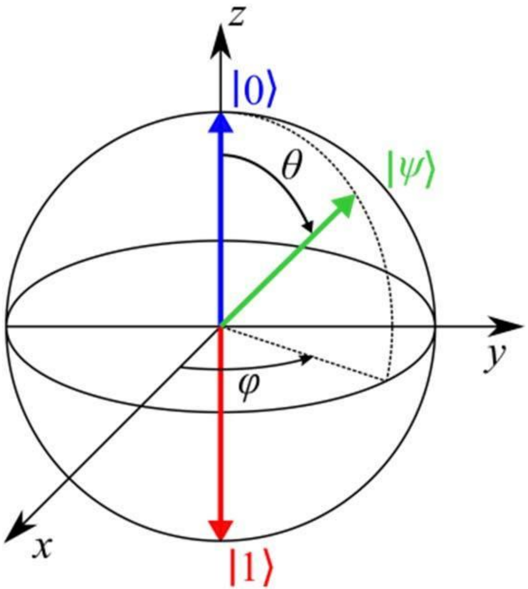
$$|\psi> = \alpha\,|0> + \beta\,|1>$$

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

$$|\psi> = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

Linking angles and vector

Bloch sphere



author: Fabio Sebastiano

$$|\psi> = \alpha\,|0> + \beta\,|1>$$

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

$$|\psi> = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \qquad \alpha \rightarrow \cos\left(\frac{\theta}{2}\right) \qquad \beta \rightarrow e^{i\varphi}\sin\left(\frac{\theta}{2}\right)$$

$$e^{i\varphi} = \cos(\varphi) + i\,\sin(\varphi)$$

Measurement

Measurement

OR

Projection on 0:

$$|\boldsymbol{\psi} > \;\rightarrow\; \frac{\boldsymbol{\alpha}}{|\boldsymbol{\alpha}|}\,|\mathbf{0} >$$

with probability $\mathbf{p(0)}$:

$$\mathbf{p(0)} = |<\mathbf{0}|\boldsymbol{\psi}>|^2 = |\boldsymbol{\alpha}|^2$$

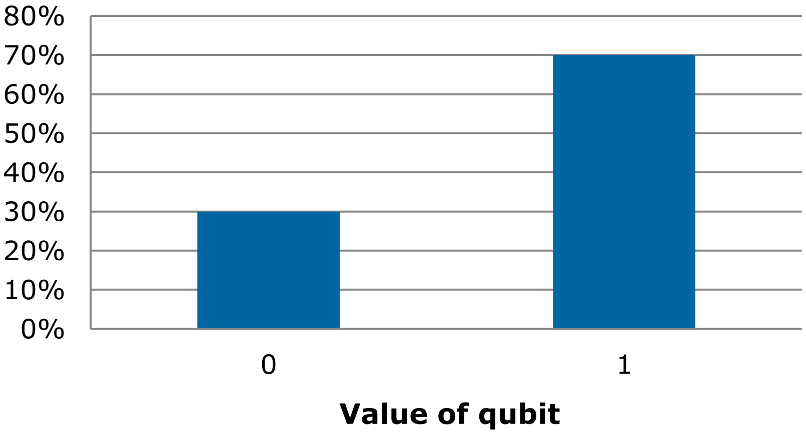
Projection on 1:

$$|\boldsymbol{\psi} > \;\rightarrow\; \frac{\boldsymbol{\beta}}{|\boldsymbol{\beta}|}\,|\mathbf{1} >$$

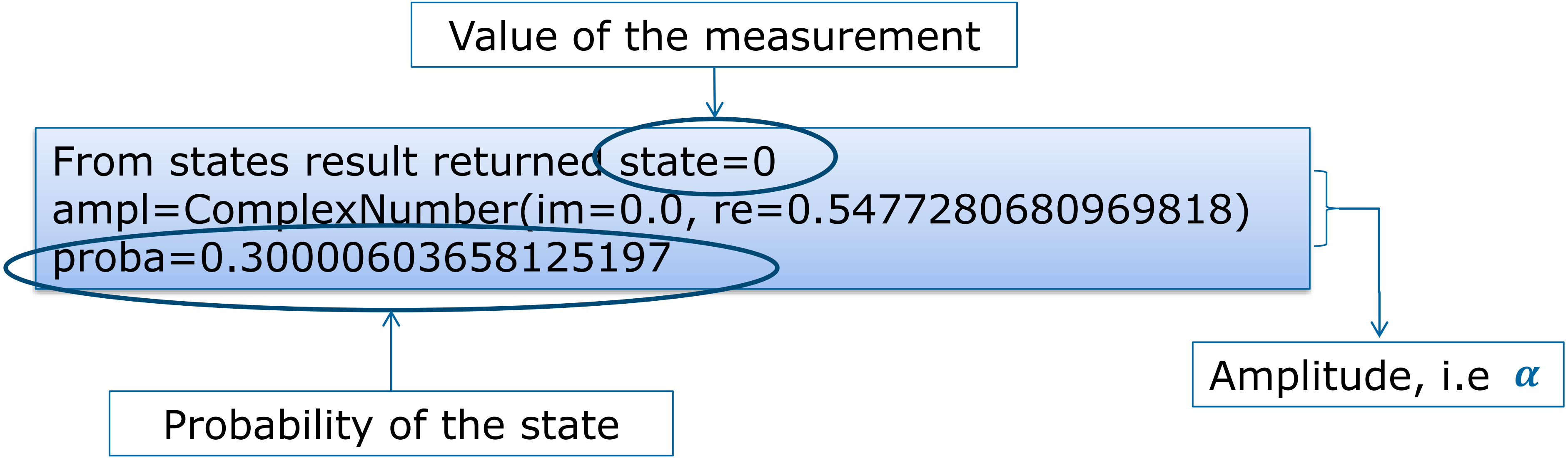
with probability $\mathbf{p(1)}$:

$$\mathbf{p(1)} = |<\mathbf{1}|\boldsymbol{\psi}>|^2 = |\boldsymbol{\beta}|^2$$

Superposition & Measurement



On a QLM



```
From states result returned state=0
ampl=ComplexNumber(im=0.0, re=0.5477280680969818)
proba=0.30000603658125197
```

Value of the measurement

Amplitude, i.e α

Probability of the state

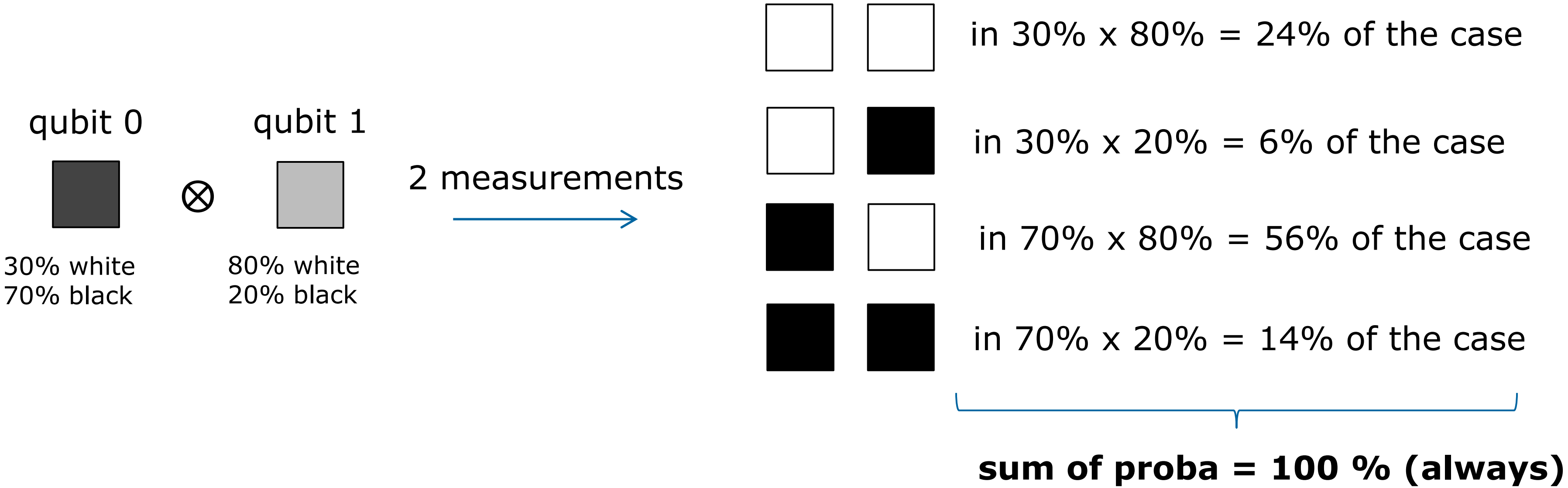
On a QLM

- Output of the full state:

```
From states result returned state=0
ampl=ComplexNumber(im=0.0, re=0.5477280680969818)
proba=0.3000060365812519
From states result returned state=1
ampl=ComplexNumber(im=-0.8366564189789905, re=0.0)
proba=0.699993963418748
```

Two qubits

- 2 qubits = 4 states



qubit 0 ⊗ qubit 1

30% white
70% black

80% white
20% black

2 measurements →

⬜ ⬜ in 30% x 80% = 24% of the case

⬜ ⬛ in 30% x 20% = 6% of the case

⬛ ⬜ in 70% x 80% = 56% of the case

⬛ ⬛ in 70% x 20% = 14% of the case

sum of proba = 100 % (always)

Two qubits

qubit 0 qubit 1

☐ ☐ = 00 = 0

☐ ■ = 01 = 1

■ ☐ = 10 = 2

■ ■ = 11 = 3

$$|\boldsymbol{\psi_0} > = \boldsymbol{\alpha_0}\,|\mathbf{0} > \; + \boldsymbol{\beta_0}\,|\mathbf{1} >$$

$$|\boldsymbol{\psi_1} > = \boldsymbol{\alpha_1}\,|\mathbf{0} > \; + \boldsymbol{\beta_1}\,|\mathbf{1} >$$

Two qubits

qubit 0 qubit 1

□ □ = 00 = 0

□ ■ = 01 = 1

■ □ = 10 = 2

■ ■ = 11 = 3

$$|\psi_0 > = \alpha_0 \, |0 > \; + \beta_0 \, |1 >$$

$$|\psi_1 > = \alpha_1 \, |0 > \; + \beta_1 \, |1 >$$

$$|\psi_{tot} > = |\psi_0 > \otimes |\psi_1 >$$

Two qubits

qubit 0	qubit 1	
☐	☐	= 00 = 0
☐	■	= 01 = 1
■	☐	= 10 = 2
■	■	= 11 = 3

$$|\psi_0 > = \alpha_0 \, |0 > \; + \beta_0 \, |1 >$$

$$|\psi_1 > = \alpha_1 \, |0 > \; + \beta_1 \, |1 >$$

$$\begin{aligned} |\psi_{tot} > &= |\psi_0 > \otimes |\psi_1 > \\ &= |\psi_0 > |\psi_1 > \end{aligned}$$

Two qubits

qubit 0 qubit 1

☐ ☐ = 00 = 0

☐ ■ = 01 = 1

■ ☐ = 10 = 2

■ ■ = 11 = 3

$$|\psi_0> = \alpha_0\,|0> \;+ \beta_0\,|1>$$

$$|\psi_1> = \alpha_1\,|0> \;+ \beta_1\,|1>$$

$$\begin{aligned} |\psi_{tot}> &= |\psi_0> \otimes |\psi_1> \\ &= |\psi_0>\,|\psi_1> \\ &= |\psi_0\psi_1> \end{aligned}$$

Two qubits

qubit 0	qubit 1	
☐	☐	= 00 = 0
☐	■	= 01 = 1
■	☐	= 10 = 2
■	■	= 11 = 3

$$|\boldsymbol{\psi}_{\boldsymbol{tot}} > = \boldsymbol{\alpha}_0\boldsymbol{\alpha}_1|\mathbf{00} > \; +\boldsymbol{\alpha}_0\boldsymbol{\beta}_1|\mathbf{01} > \\ +\boldsymbol{\beta}_0\boldsymbol{\alpha}_1|\mathbf{10} > \; +\boldsymbol{\beta}_0\boldsymbol{\beta}_1|\mathbf{11} >$$

$$= \; \boldsymbol{\alpha}_0\boldsymbol{\alpha}_1|\mathbf{0} > \; +\boldsymbol{\alpha}_0\boldsymbol{\beta}_1|\mathbf{1} > \\ +\boldsymbol{\beta}_0\boldsymbol{\alpha}_1|\mathbf{2} > \; +\boldsymbol{\beta}_0\boldsymbol{\beta}_1|\mathbf{3} >$$

Computational basis

$$|\mathbf{0}> = \begin{pmatrix}1\\0\end{pmatrix}$$

$$|\mathbf{1}> = \begin{pmatrix}0\\1\end{pmatrix}$$

Computational basis

$$|\mathbf{0}> = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$|\mathbf{0}> \otimes |\mathbf{1}> = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$|\mathbf{1}> = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Computational basis

$$|\mathbf{0}> = \begin{pmatrix}1\\0\end{pmatrix} \qquad |\mathbf{0}> \otimes |\mathbf{1}> = \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}1 * \begin{pmatrix}0\\1\end{pmatrix}\\ 0 * \begin{pmatrix}0\\1\end{pmatrix}\end{pmatrix}$$

$$|\mathbf{1}> = \begin{pmatrix}0\\1\end{pmatrix}$$

Computational basis

$$|\mathbf{0}> = \begin{pmatrix}1\\0\end{pmatrix}$$

$$|\mathbf{0}> \otimes |\mathbf{1}> = \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}1 * \begin{pmatrix}0\\1\end{pmatrix}\\ 0 * \begin{pmatrix}0\\1\end{pmatrix}\end{pmatrix} = \begin{pmatrix}\mathbf{0}\\\mathbf{1}\\\mathbf{0}\\\mathbf{0}\end{pmatrix}$$

$$|\mathbf{1}> = \begin{pmatrix}0\\1\end{pmatrix}$$

Computational basis

$$|\mathbf{0}> = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$|\mathbf{0}> \otimes |\mathbf{1}> = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 * \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ 0 * \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{1} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix} = |1>$$

$$|\mathbf{1}> = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Computational basis

$$|\mathbf{0}> = \begin{pmatrix}1\\0\end{pmatrix}$$

$$|\mathbf{0}> \otimes |\mathbf{1}> = \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}1 * \begin{pmatrix}0\\1\end{pmatrix}\\ 0 * \begin{pmatrix}0\\1\end{pmatrix}\end{pmatrix} = \begin{pmatrix}\mathbf{0}\\\mathbf{1}\\\mathbf{0}\\\mathbf{0}\end{pmatrix} = |1>$$

$$|\mathbf{1}> = \begin{pmatrix}0\\1\end{pmatrix}$$

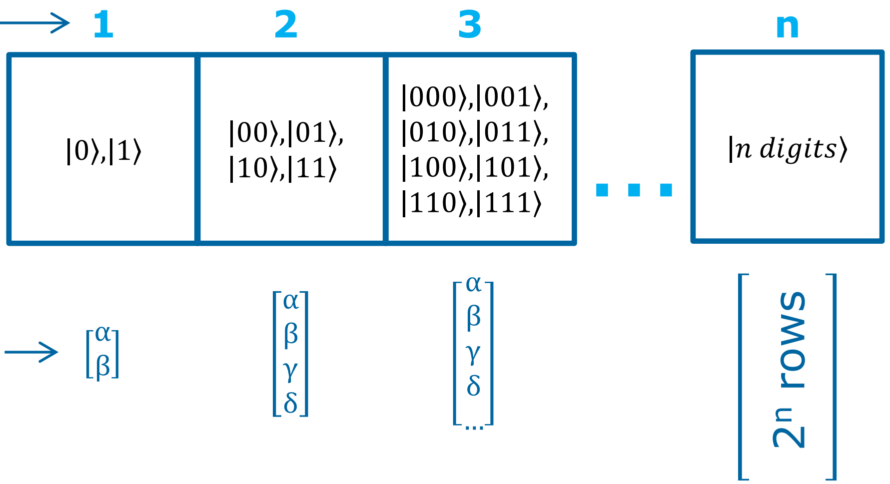
```
State |00>; int 0; probability 0.24000485542571934
State |01>; int 1; probability 0.5599952317749185
State |10>; int 2; probability 0.06000118115532664
State |11>; int 3; probability 0.13999873164382956
```

On a QLM

- Output of the full state:

```
From states result returned state=0
ampl=ComplexNumber(im=0.0, re=0.4899035164566278)
proba=0.2400054543656937
From states result returned state=1
ampl=ComplexNumber(im=-0.7483292268513849, re=0.0)
proba=0.5599966317599915
From states result returned state=2
ampl=ComplexNumber(im=-0.24495016053206137, re=0.0)
proba=0.0600058114468263
From states result returned state=3
ampl=ComplexNumber(im=0.0, re=-0.3741621729394309)
proba=0.13999733165875658
```

More Qubits



Number of qubits → **1** **2** **3** ... **n**

$|0\rangle, |1\rangle$

$|00\rangle, |01\rangle,$
$|10\rangle, |11\rangle$

$|000\rangle, |001\rangle,$
$|010\rangle, |011\rangle,$
$|100\rangle, |101\rangle,$
$|110\rangle, |111\rangle$

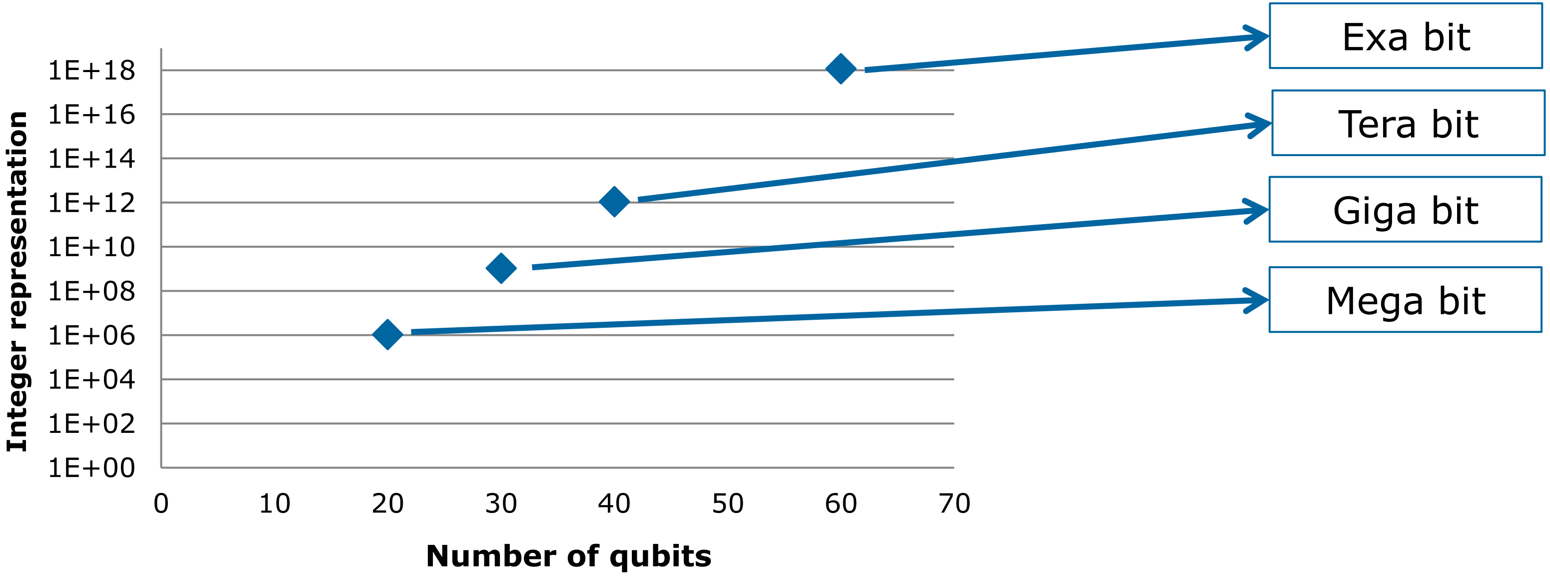
$|n\ digits\rangle$

qubit vector → $\begin{bmatrix}\alpha\\\beta\end{bmatrix}$ $\begin{bmatrix}\alpha\\\beta\\\gamma\\\delta\end{bmatrix}$ $\begin{bmatrix}\alpha\\\beta\\\gamma\\\delta\\\dots\end{bmatrix}$ $\begin{bmatrix}2^n \text{ rows}\end{bmatrix}$

tensor product $|\psi_1\rangle$ $|\psi_1\rangle \otimes |\psi_2\rangle$ .... $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle$

Scalability

- N qubits = **2^N states**



Entanglement

- Let’s take 2 unentangled qubits. A classical preparation would give us:

$$|\psi_0 > = \alpha_0 \, |0 > \; + \beta_0 \, |1 >$$

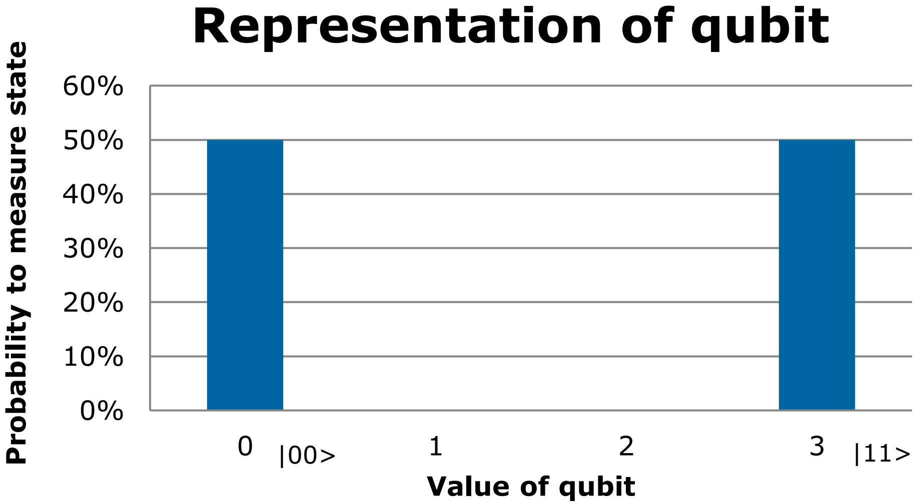
$$|\psi_1 > = \alpha_1 \, |0 > \; + \beta_1 \, |1 >$$

- And consider their joint state:

$$|\psi_0 > |\psi_1 >= \alpha_0\alpha_1|00 > \; +\alpha_0\beta_1|01 > \\ +\beta_0\alpha_1|10 > \; +\beta_0\beta_1|11 >$$

Entanglement

- OK, great but what if I want to prepare my 2 qubits to obtain this :



Previously explained

$$|\psi_0\psi_1> = \alpha_0\alpha_1|00> + \alpha_0\beta_1|01> + \beta_0\alpha_1|10> + \beta_0\beta_1|11>$$

For example:

- need to have $\alpha_0 \neq 0$ for |00>
- need to have $\beta_1 \neq 0$ for |11>
- but need $\alpha_0\beta_1 = 0$ for |01>

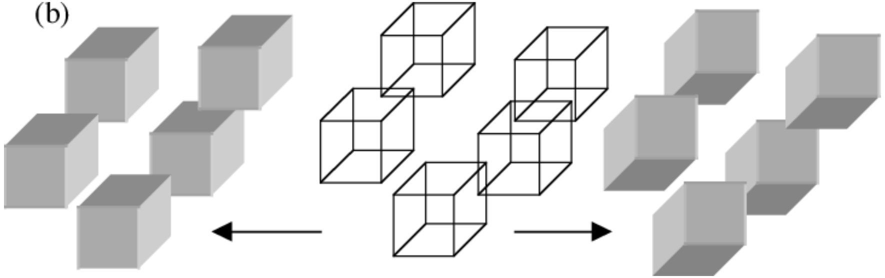
This cannot be prepared locally

Entanglement

- To get previous state : **entangle your qubits.** This means that the qubits need to **interact. For instance, you could conditionally flip the qubit.**

- The previous state $(|00> + |11>)$ is known as an EPR pair.

- This state obtained of q_1 and q_0 can no longer be separated as $|q_0> \otimes |q_1>$



Entanglement

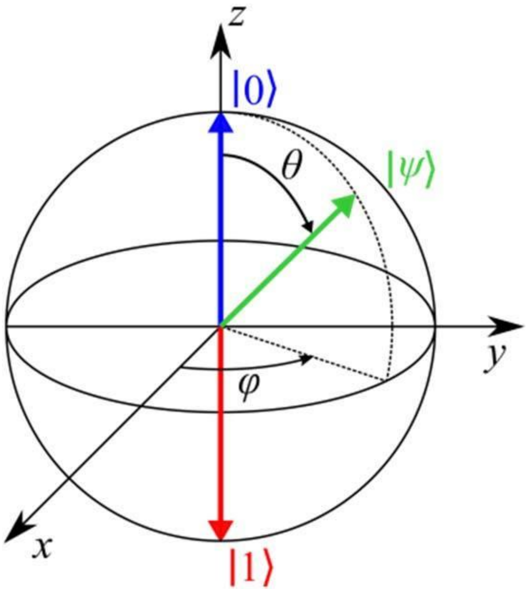
- Entanglement is a key point for quantum computing:
  - Without entanglement, the register can be described using 2N complex numbers.
  - General states are entangled, and one needs up to 2^N complex numbers to describe them.

Summary

- A qubit is a superposition of 0 and 1
- Measurement gives only access to 0 OR 1
- N qubits correspond to 2^N states
- Entanglement is mandatory

Back to the representation

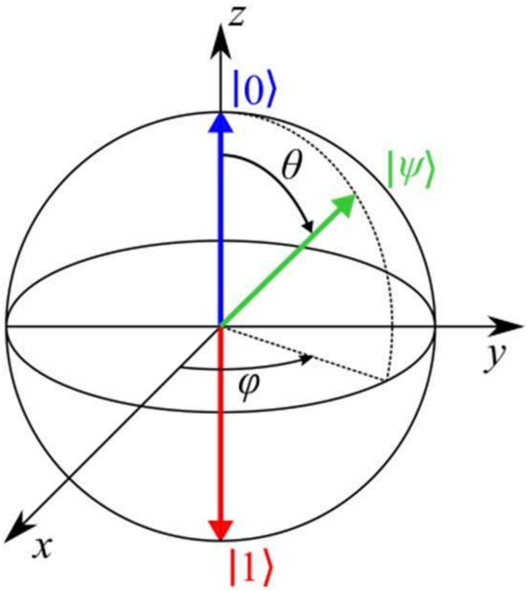
Bloch sphere



author: Fabio Sebastiano

Back to the representation

Bloch sphere

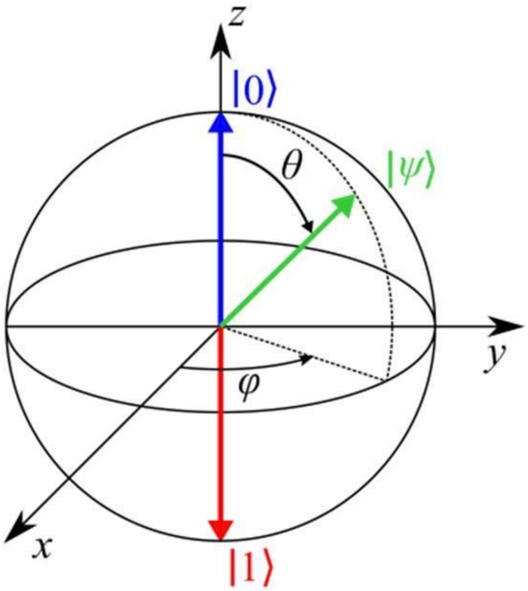


author: Fabio Sebastiano

- reflection/rotation

Back to the representation

Bloch sphere

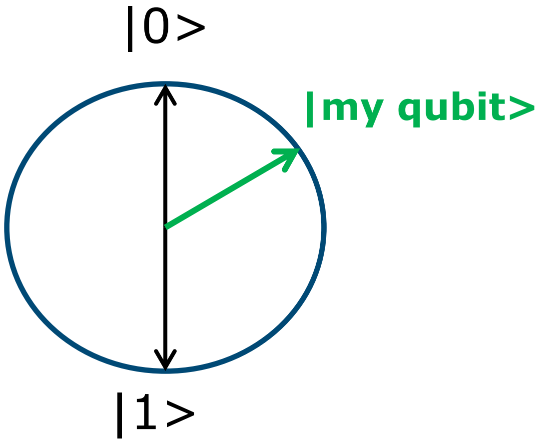


author: Fabio Sebastiano

- reflection/rotation
- unitary matrices :

$$MM^{\dagger}=I$$

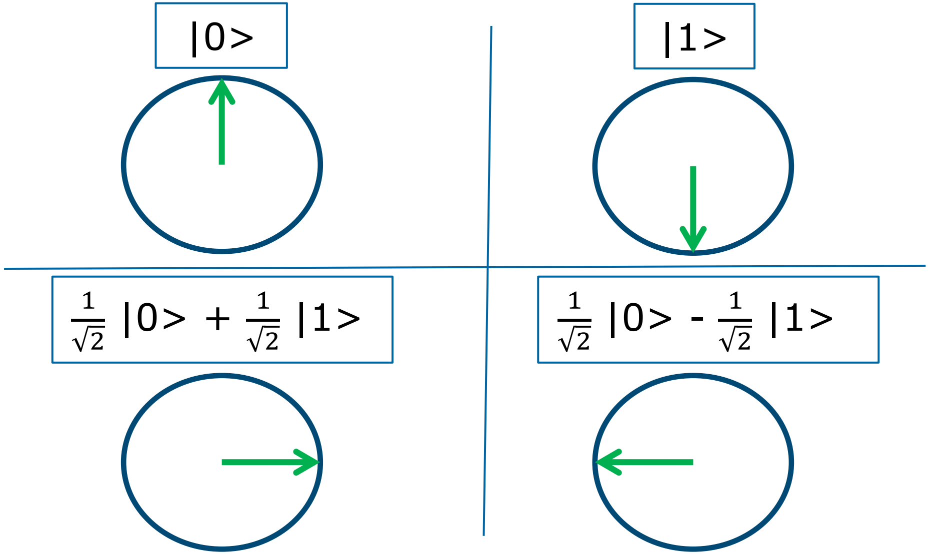
Another view



Will give
70% |0>

Will give
30% |1>

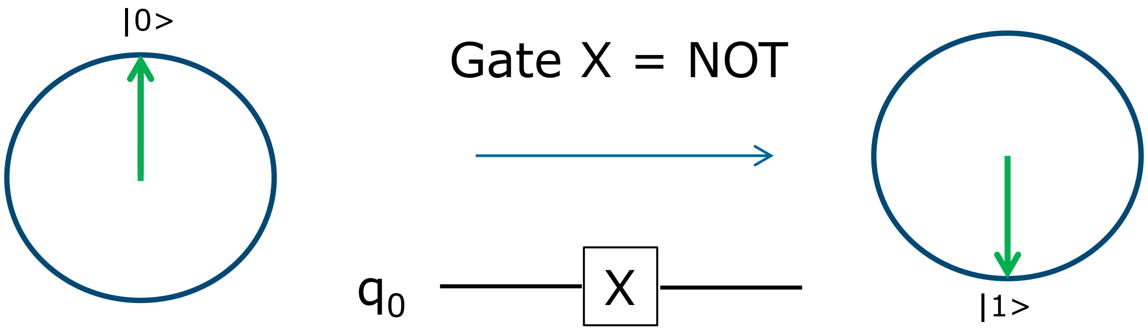
Measurement = project + renormalize



$\frac{1}{\sqrt{2}}$ |0> + $\frac{1}{\sqrt{2}}$ |1>

$\frac{1}{\sqrt{2}}$ |0> - $\frac{1}{\sqrt{2}}$ |1>

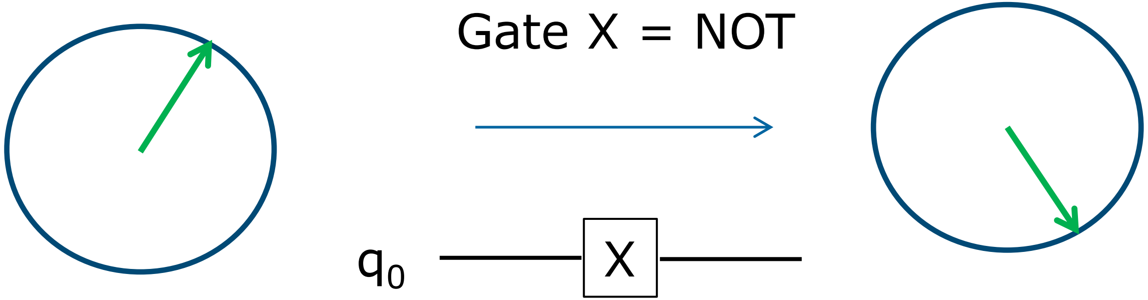
Quantum Gates



$$\begin{array}{cc} \begin{pmatrix} 1 \\ 0 \end{pmatrix} & \begin{array}{l} |0> \\ |1> \end{array} \end{array}$$

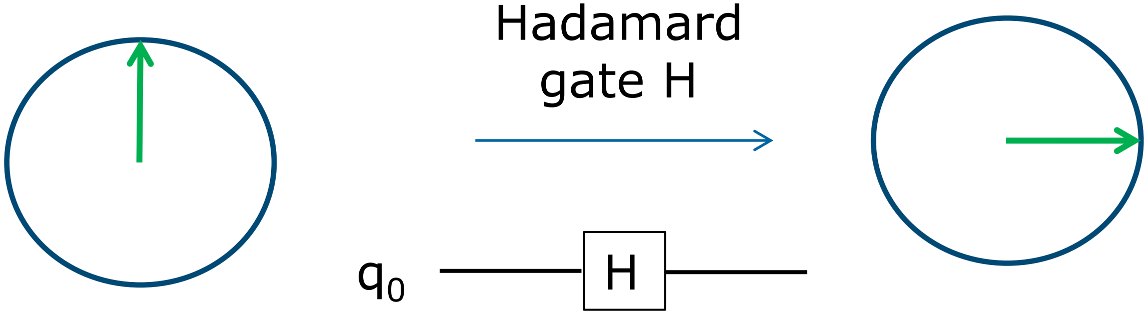
$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Quantum Gates



$$\begin{pmatrix}\alpha \\ \beta\end{pmatrix}\begin{matrix}|0> \\ |1>\end{matrix} \qquad \begin{pmatrix}0 & 1 \\ 1 & 0\end{pmatrix} \quad \rightarrow \quad \begin{pmatrix}\beta \\ \alpha\end{pmatrix}\begin{matrix}|0> \\ |1>\end{matrix}$$

Quantum Gates


Hadamard gate H

$$\begin{pmatrix}1\\0\end{pmatrix}\begin{array}{l}|0>\\|1>\end{array} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix} \rightarrow \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}$$

Main Quantum Gates of arity 1

Pauli-X	$\boxed{\mathrm{X}}$	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Pauli-Y	$\boxed{\mathrm{Y}}$	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Pauli-Z	$\boxed{\mathrm{Z}}$	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
Phase	$\boxed{\mathrm{S}}$	$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$
Phase Shift	$\boxed{\mathrm{PH}[\theta]}$	$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$

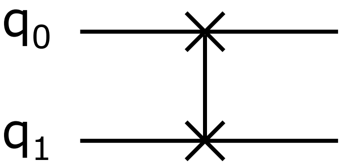
X Rotation	$\boxed{\mathrm{RX}[\theta]}$	$\begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$
Y Rotation	$\boxed{\mathrm{RY}[\theta]}$	$\begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$
Z Rotation	$\boxed{\mathrm{RZ}[\theta]}$	$\begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}$

A Quantum Gate of arity 2

$$|q_0> = \alpha_0 \,|0> \; + \beta_0 \,|1>$$

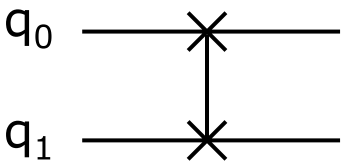
$$|q_1> = \alpha_1 \,|0> \; + \beta_1 \,|1>$$

SWAP
Gate



A Quantum Gate of arity 2

SWAP Gate



$|q_0 >= \alpha_0\,|0> \;+\beta_0\,|1>$

$|q_1 >= \alpha_1\,|0> \;+\beta_1\,|1>$

$|q_0 >= \alpha_1|0> \;+\beta_1|1>$

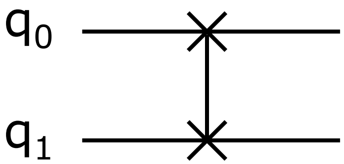
$|q_1 >= \alpha_0|0> \;+\beta_0|1>$

A Quantum Gate of arity 2

SWAP
Gate

$|q_0> = \alpha_0\,|0> \;+ \beta_0\,|1>$

$|q_1> = \alpha_1\,|0> \;+ \beta_1\,|1>$

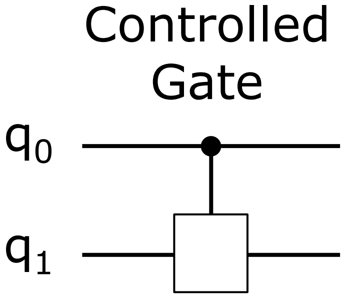


$|q_0> = \alpha_1|0> \;+ \beta_1|1>$

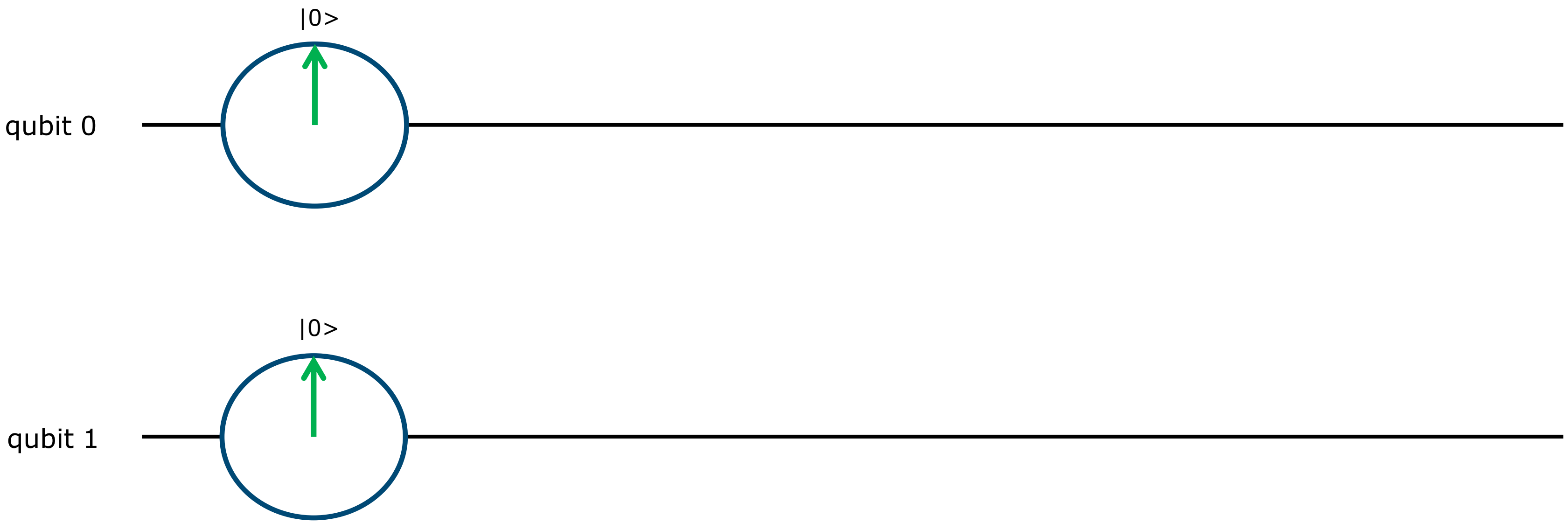
$|q_1> = \alpha_0|0> \;+ \beta_0|1>$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha_0\alpha_1 \\ \alpha_0\beta_1 \\ \beta_0\alpha_1 \\ \beta_0\beta_1 \end{pmatrix} = \begin{pmatrix} \alpha_1\alpha_0 \\ \alpha_1\beta_0 \\ \beta_1\alpha_0 \\ \beta_1\beta_0 \end{pmatrix}$$

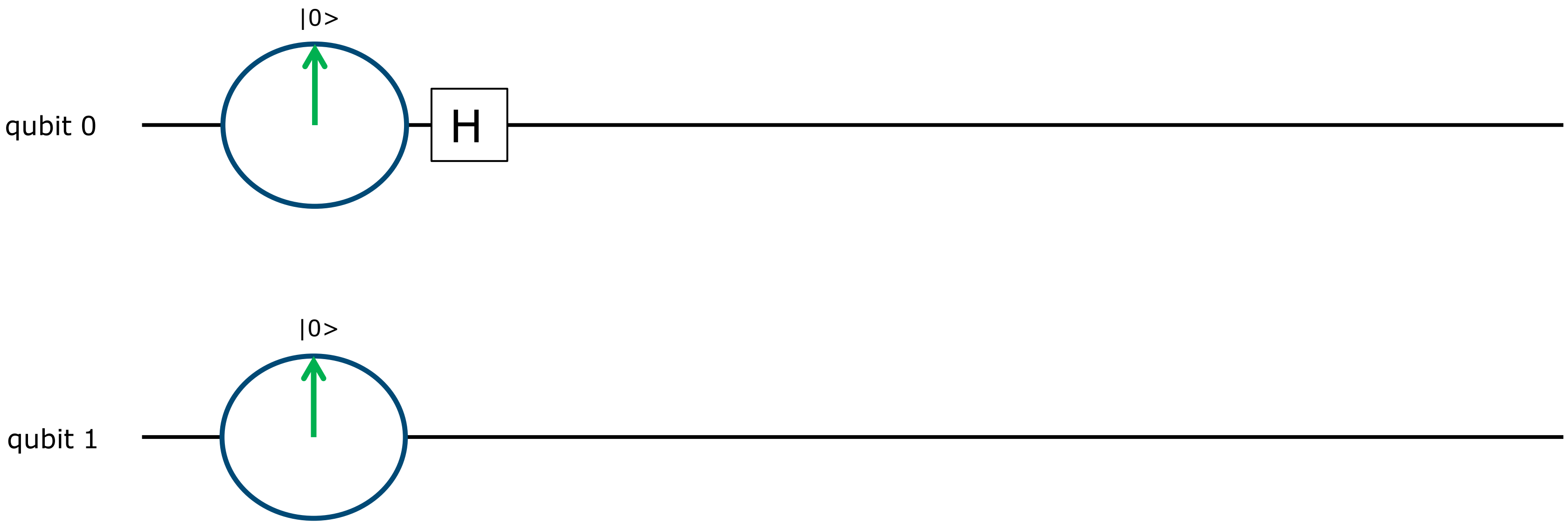
Controlled Quantum Gates



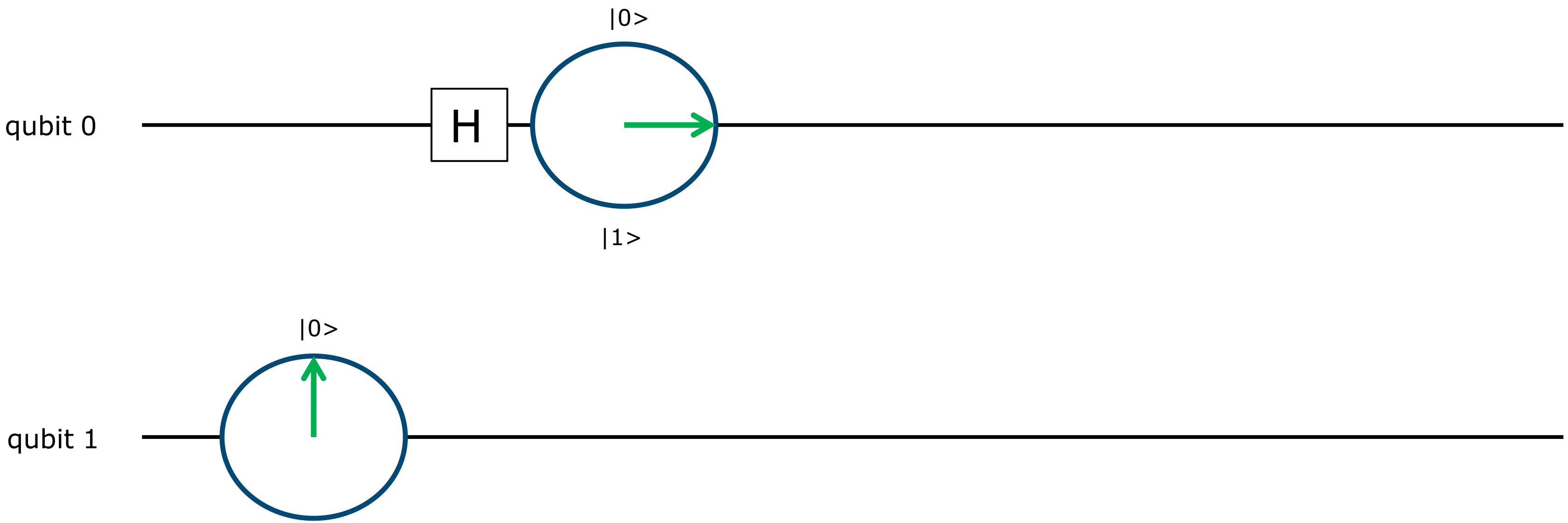
The C-NOT Gate



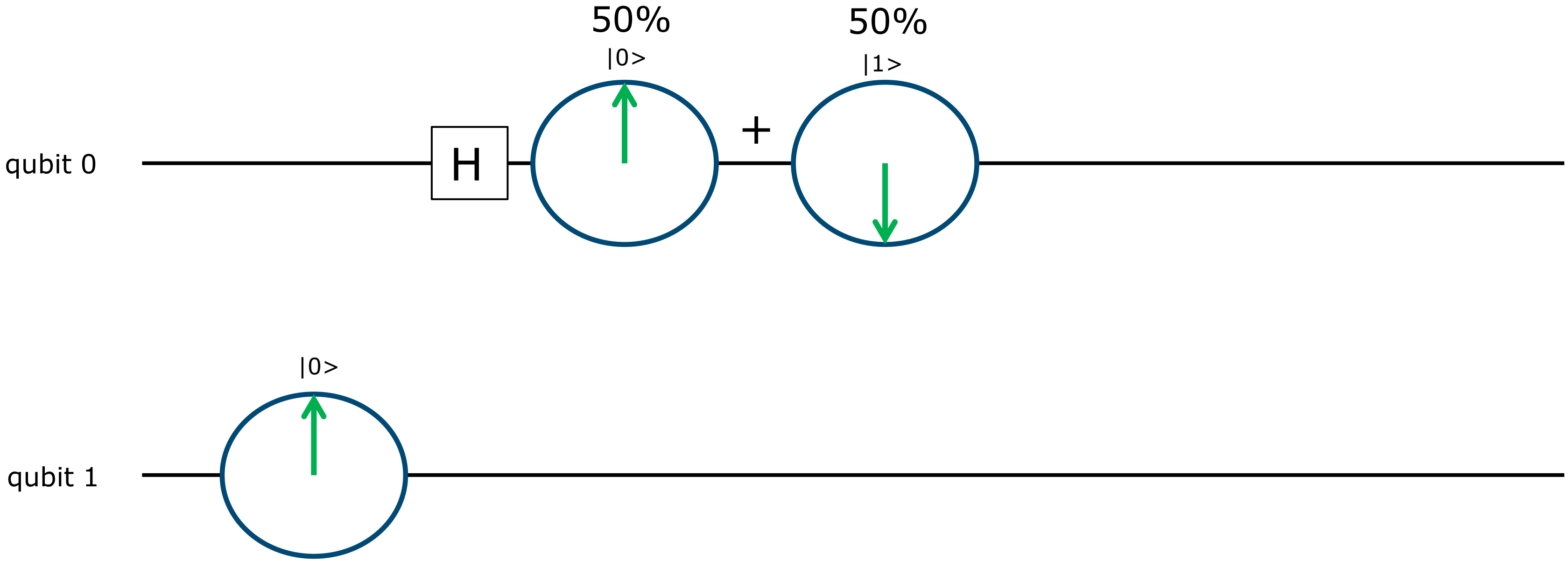
The C-NOT Gate



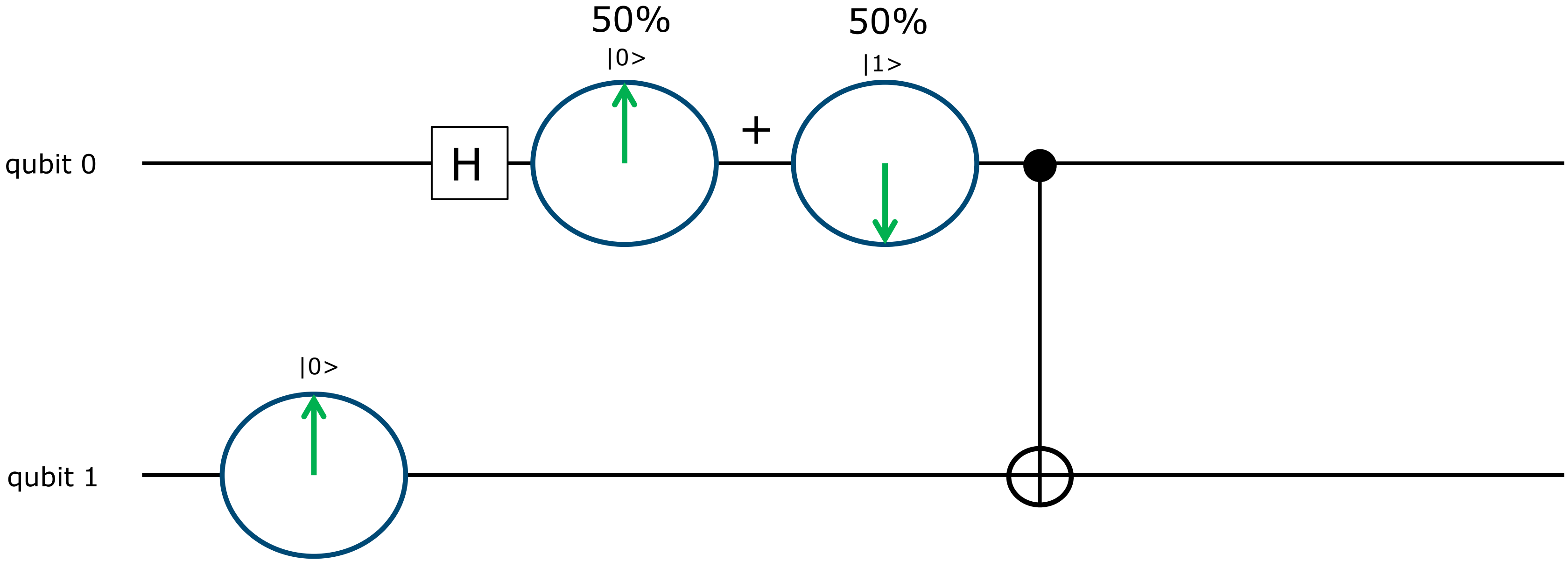
The C-NOT Gate



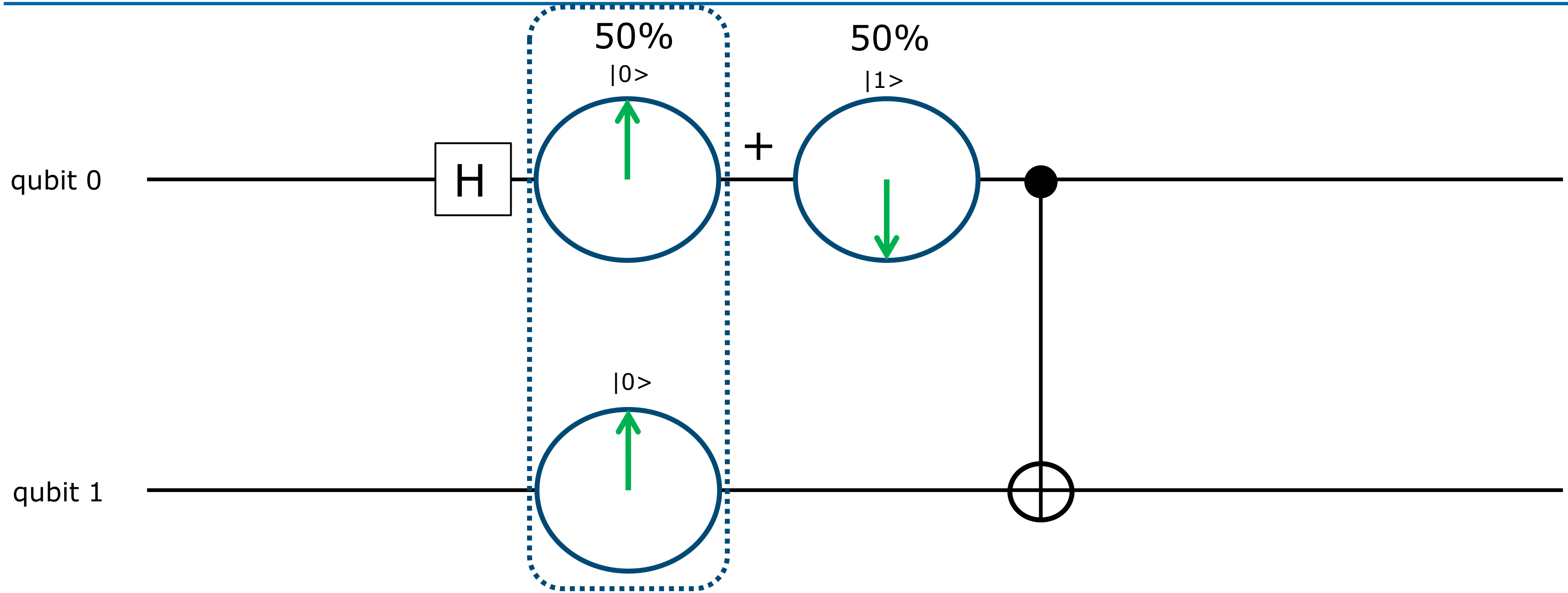
The C-NOT Gate



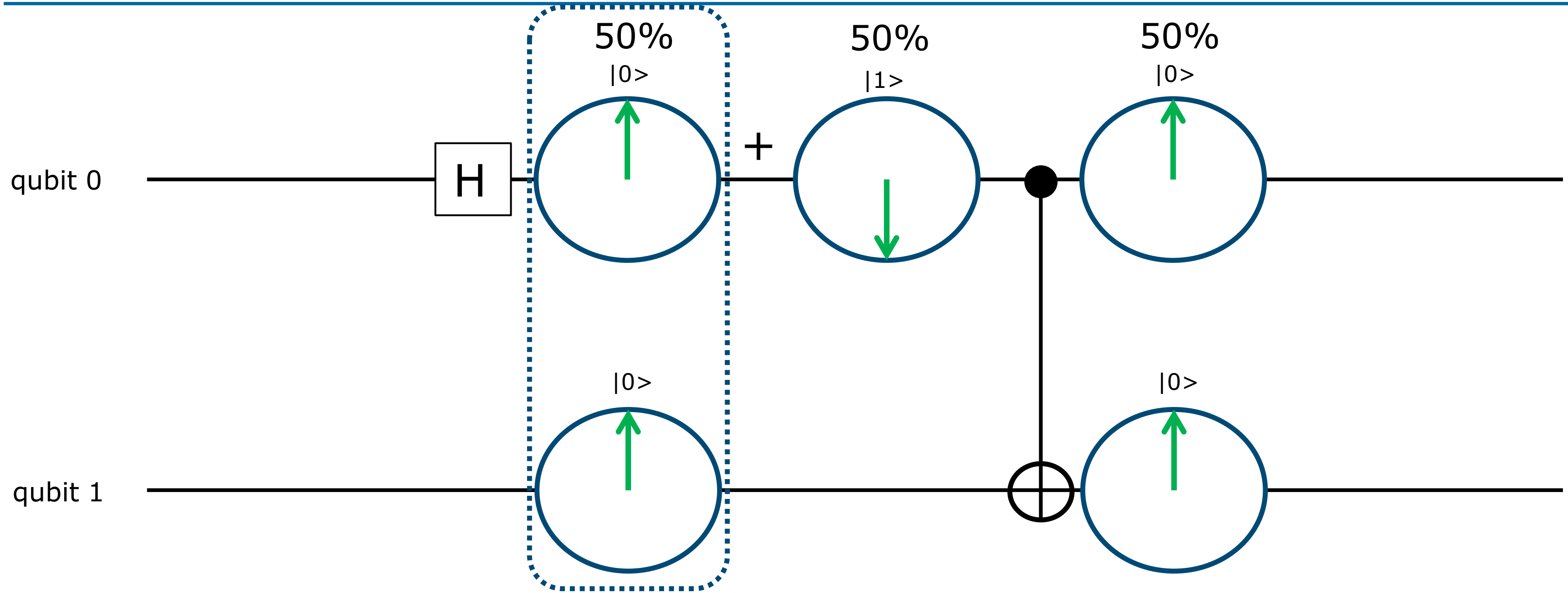
The C-NOT Gate



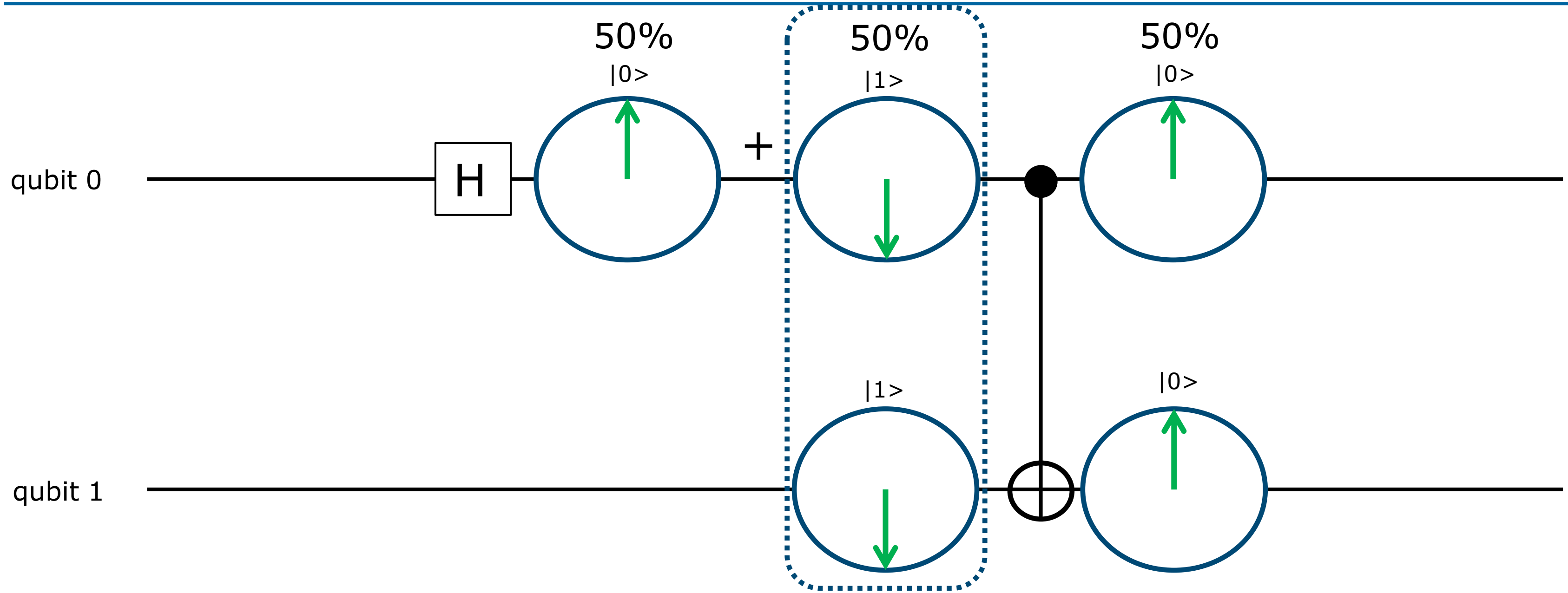
The C-NOT Gate



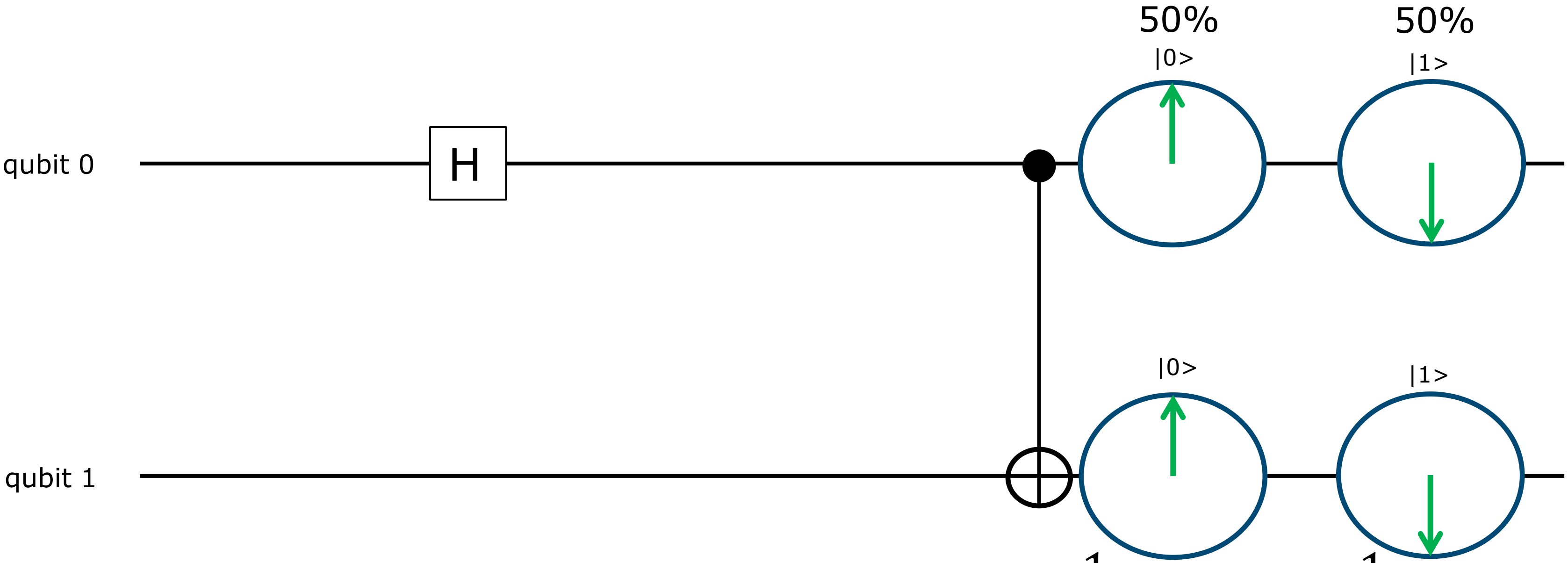
The C-NOT Gate



The C-NOT Gate



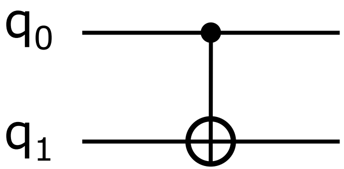
The C-NOT Gate



$$\frac{1}{\sqrt{2}}|\mathbf{00}> + \frac{1}{\sqrt{2}}|\mathbf{11}>$$

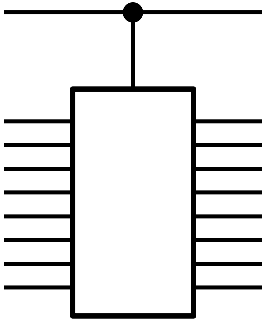
Controlled Quantum Gates

CNOT



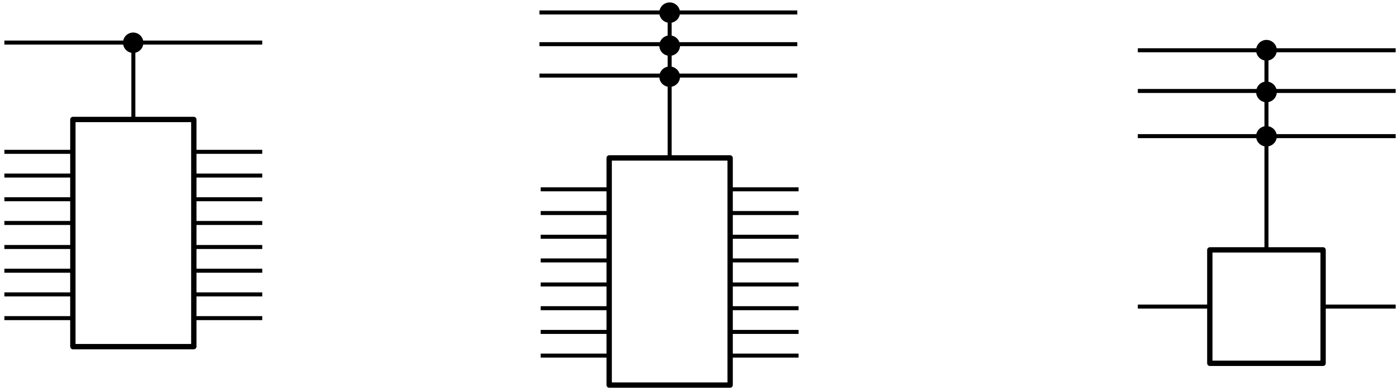
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha_0\alpha_1 \\ \alpha_0\beta_1 \\ \beta_0\alpha_1 \\ \beta_0\beta_1 \end{pmatrix} = \begin{pmatrix} \alpha_0\alpha_1 \\ \alpha_0\beta_1 \\ \beta_0\beta_1 \\ \beta_0\alpha_1 \end{pmatrix}$$

Controlled Quantum Gates



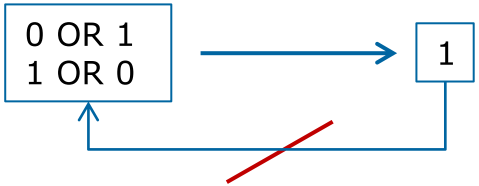
Controlled Quantum Gates

Controlled Quantum Gates



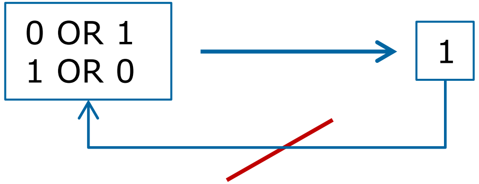
Quantum Gates

- All operations must be reversible.



Quantum Gates

- All operations must be reversible.



$q0$	$q1$	$q2$	$q0$	$q1$	$q2$
$\vert 0\rangle$	$\vert 0\rangle$	$\vert 0\rangle$	$\vert 0\rangle$	$\vert 0\rangle$	$\vert 0\rangle$
$\vert 0\rangle$	$\vert 0\rangle$	$\vert 1\rangle$	$\vert 0\rangle$	$\vert 0\rangle$	$\vert 1\rangle$
$\vert 0\rangle$	$\vert 1\rangle$	$\vert 0\rangle$	$\vert 0\rangle$	$\vert 1\rangle$	$\vert 1\rangle$
$\vert 0\rangle$	$\vert 1\rangle$	$\vert 1\rangle$	$\vert 0\rangle$	$\vert 1\rangle$	$\vert 0\rangle$
$\vert 1\rangle$	$\vert 0\rangle$	$\vert 0\rangle$	$\vert 1\rangle$	$\vert 0\rangle$	$\vert 1\rangle$
$\vert 1\rangle$	$\vert 0\rangle$	$\vert 1\rangle$	$\vert 1\rangle$	$\vert 0\rangle$	$\vert 0\rangle$
$\vert 1\rangle$	$\vert 1\rangle$	$\vert 0\rangle$	$\vert 1\rangle$	$\vert 1\rangle$	$\vert 1\rangle$
$\vert 1\rangle$	$\vert 1\rangle$	$\vert 1\rangle$	$\vert 1\rangle$	$\vert 1\rangle$	$\vert 0\rangle$

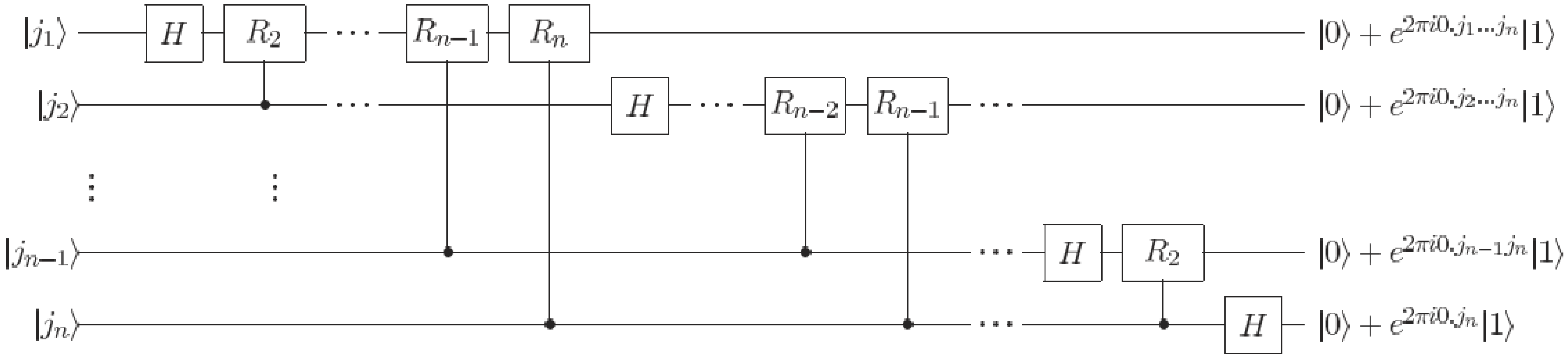
Target qubit

$$|q_0 > |q_1> |q_2 > \rightarrow |q_0 > |q_1> |q_2 + q_0 \wedge q_1 >$$

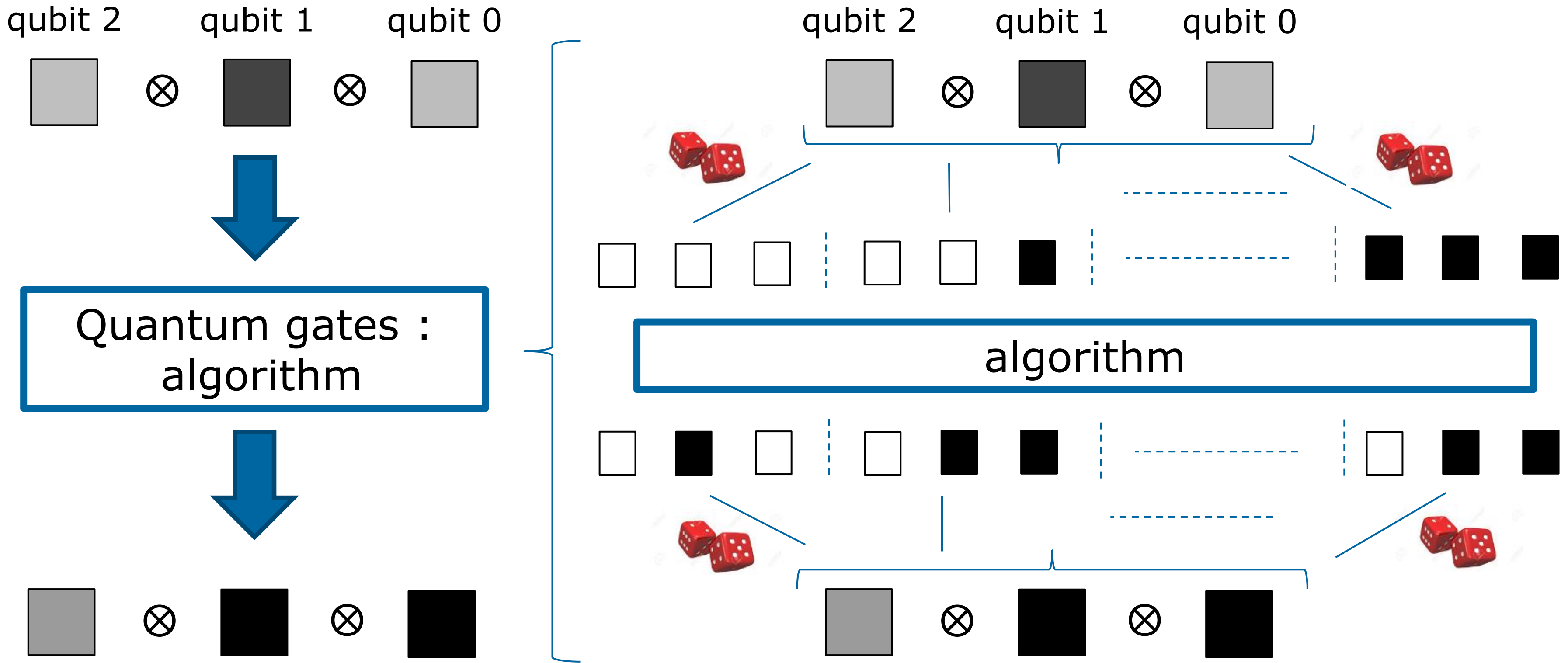
Quantum Circuits

- Gates are building blocks for circuits

Example of circuit (Quantum Fourier Transform)



Quantum Algorithm



qubit 2 qubit 1 qubit 0

⊗ ⊗

Quantum gates : algorithm

⊗ ⊗

qubit 2 qubit 1 qubit 0

⊗ ⊗

algorithm

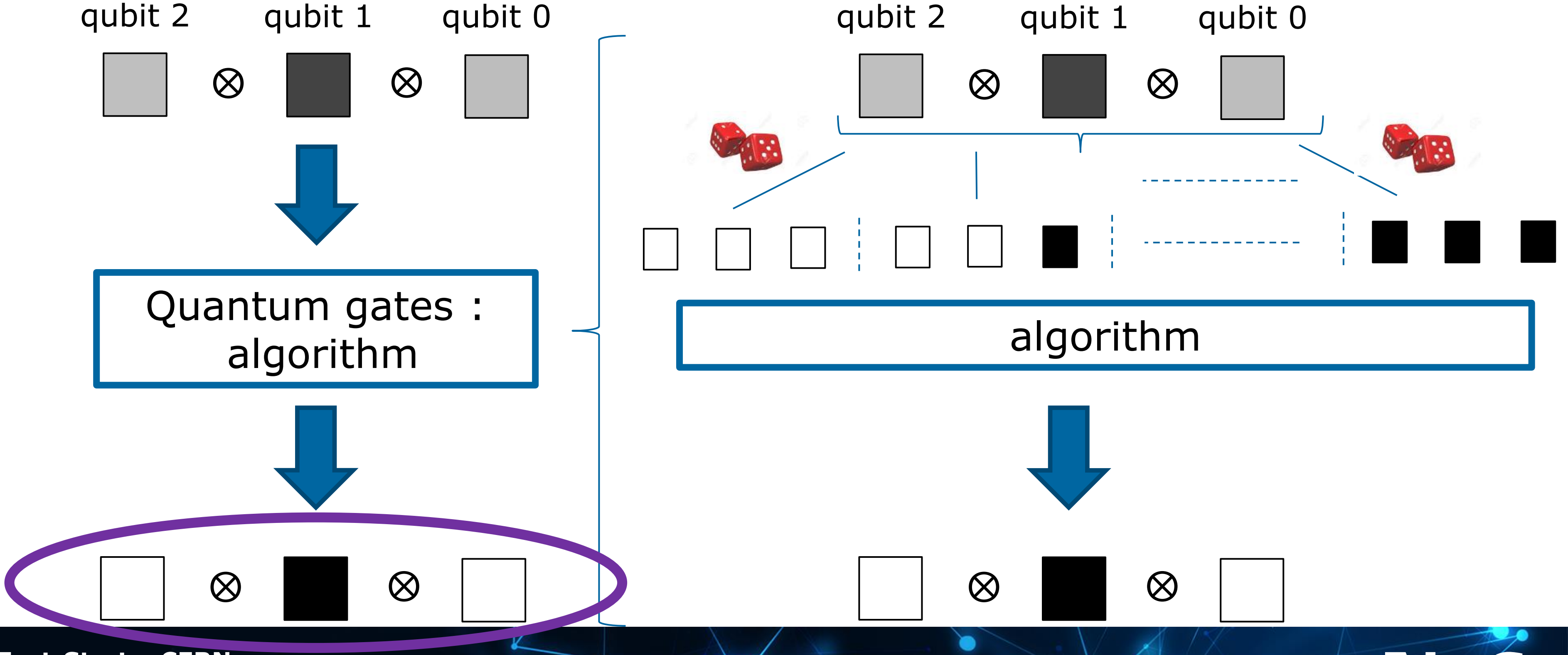
⊗ ⊗

Quantum Algorithm

…

Accumulate results

Quantum Algorithm



qubit 2 qubit 1 qubit 0

Quantum gates : algorithm

qubit 2 qubit 1 qubit 0

algorithm

Quantum Algorithm

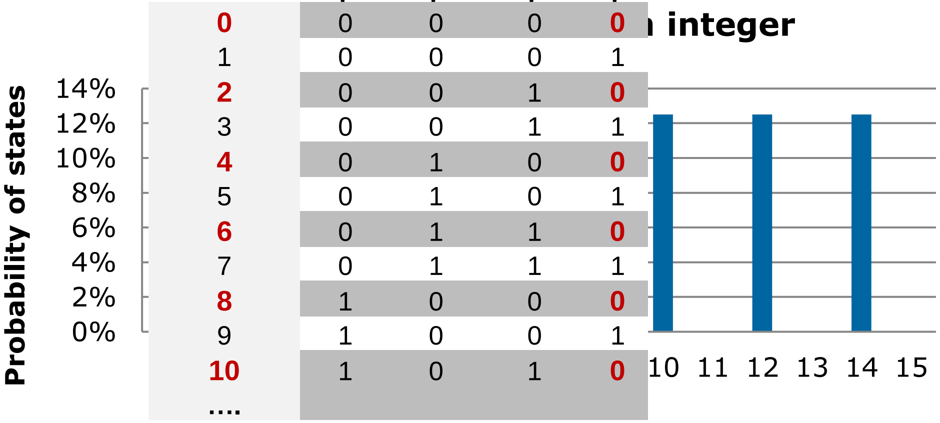
Algorithm designed to get answers (with high probability)

Ex :

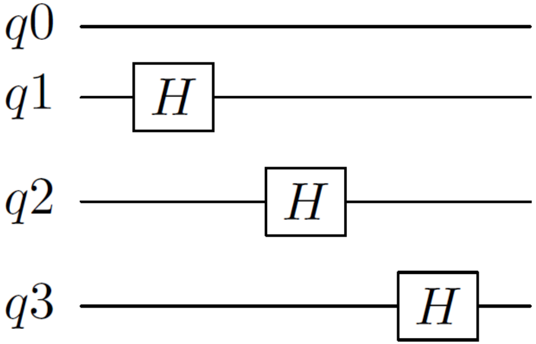
Shor
Grover

Quantum Parallelism

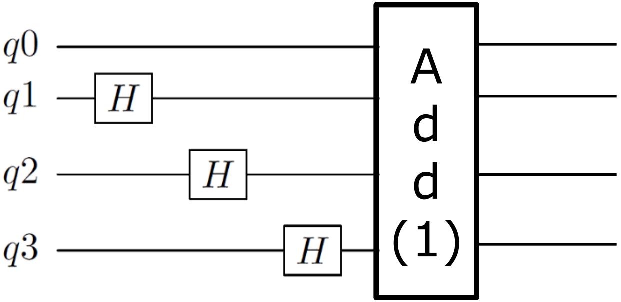
Initialization : superposition of all even numbers



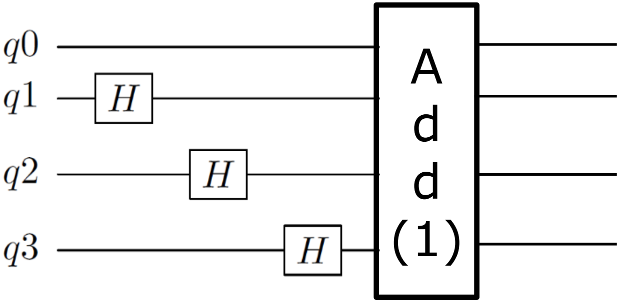
	q3	q2	q1	q0
0	0	0	0	**0**
1	0	0	0	1
2	0	0	1	**0**
3	0	0	1	1
4	0	1	0	**0**
5	0	1	0	1
6	0	1	1	**0**
7	0	1	1	1
8	1	0	0	**0**
9	1	0	0	1
10	1	0	1	**0**
….				

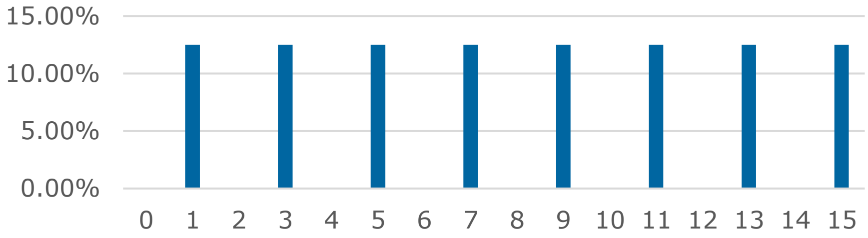


Quantum Parallelism

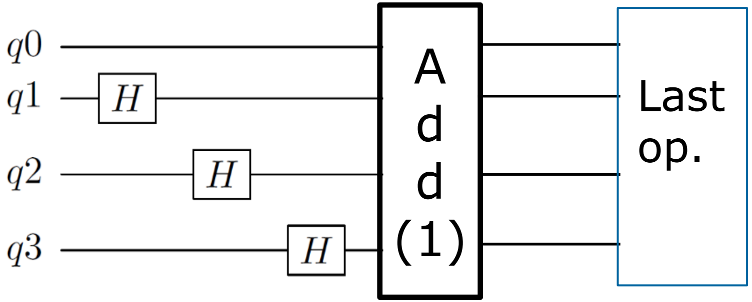


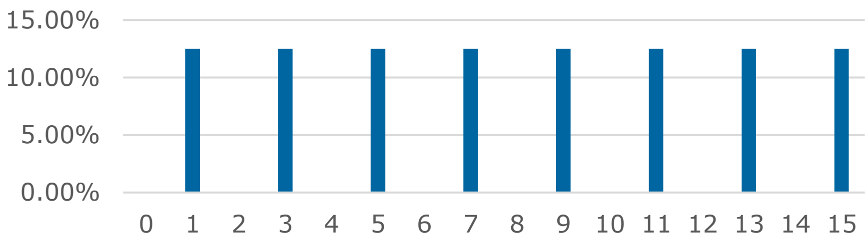
Quantum Parallelism



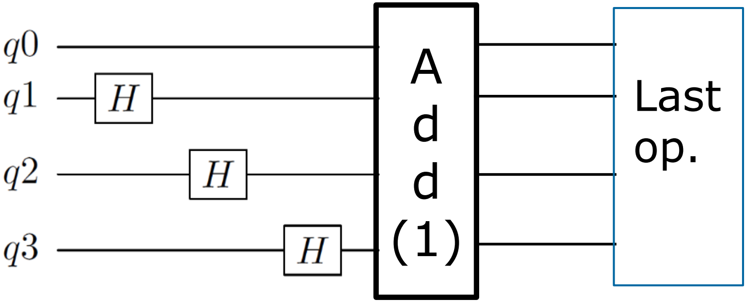


Quantum Parallelism

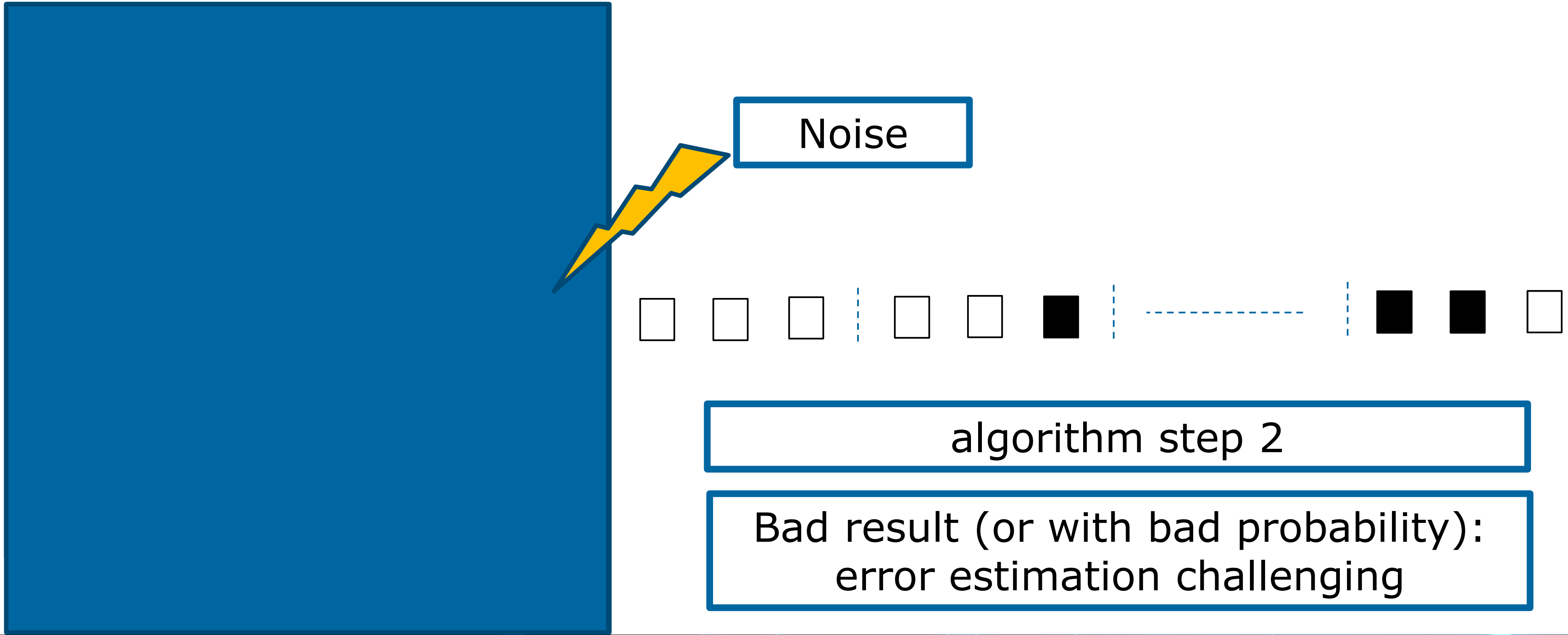




Quantum Parallelism



The Noise



Noise

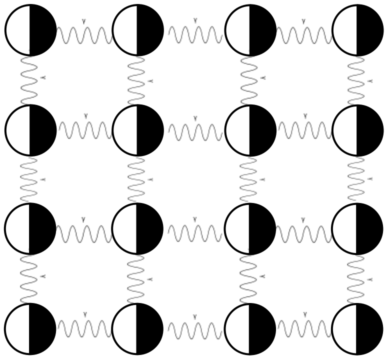
algorithm step 2

Bad result (or with bad probability):
error estimation challenging

Quantum Computing

- Available gates depend on quantum hardware

- Some challenges of real quantum hardware
  - Maintain coherence between all qubits
  - Correct quantum error



- Adapt algorithm for existing quantum hardware
  - Use available gates
  - Manage quantum hardware topology constraints
  - Optimize circuits (gates)
  - Simulate quantum errors

Overview of the QLM and myQLM

Quantum Computing

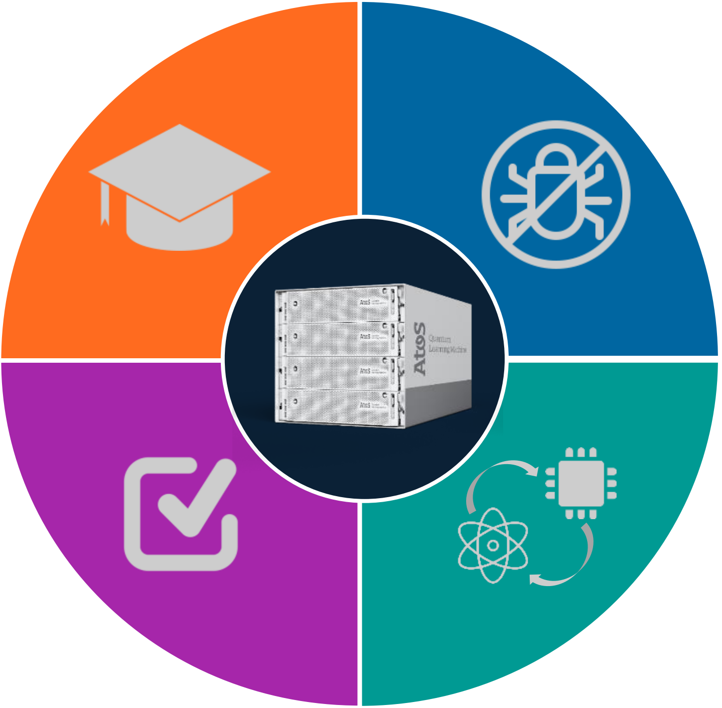
Hardware Approach

- **Pros**
  - Real Quantum speedup
- **Cons**
  - Heavy environmental constraints
  - Technology uncertainty
  - Probabilistic output makes it hard to develop algorithms

Simulation Approach

- **Pros**
  - Speeds up the quantum algorithm development phase
  - Possibility to allow quantum algorithms development without quantum hardware constraints
  - Assessing different hardware/environments for an algorithm of interest
- **Cons**
  - **No Quantum speedup**

The Atos Quantum Learning Machine



LEARN

Get acquainted with quantum computing

TEST

Conceive new programs …
… and debug them

OPTIMIZE

Select the best quantum technology to solve your problem

RUN HYBRID CODE

Off-load the quantum-accelerable parts to the simulated QPU

Overview of the QLM

PROGRAMMING

AQASM
Assembly language to build quantum circuits

pyAQASM
Python extension to AQASM

CIRC
Binary format of quantum circuits

QUANTUM ALGORITHMS
QFT, Grover's search, QAOA, VQE…

JOB
Object to submit for simulation

PLUGINS
Modify circuit before execution or gathering results after execution

OPTIMIZATION

PBO
Pattern based optimizer

NNIZER
Topology constraint solver

Circuit Optimizer
Generic circuit optimizer

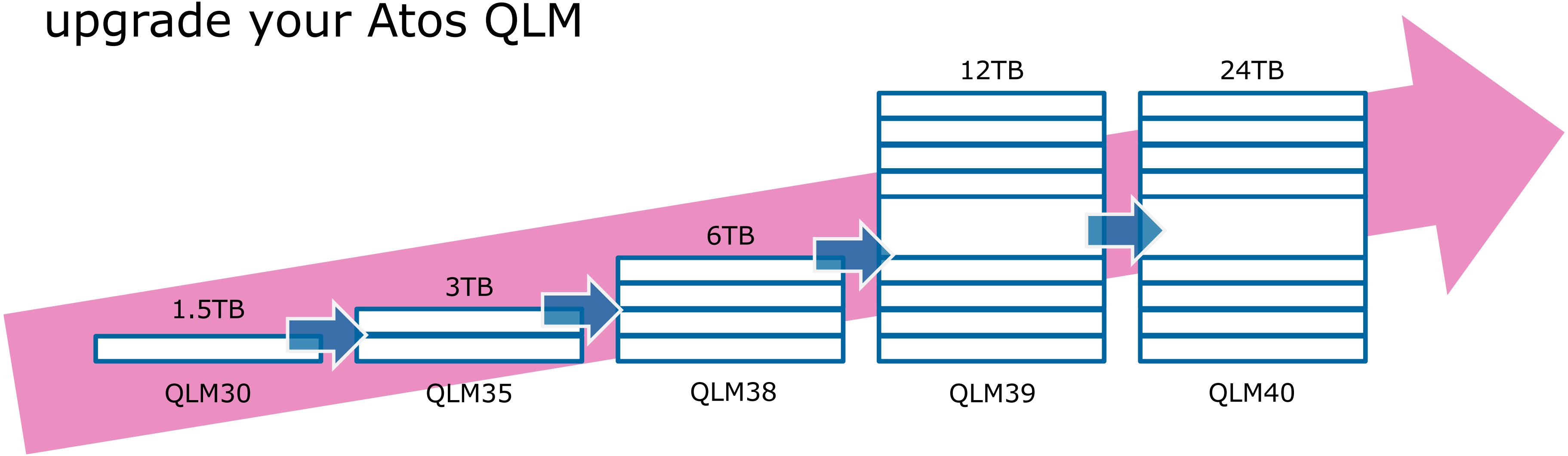
SIMULATION

PERFECT SIMULATION

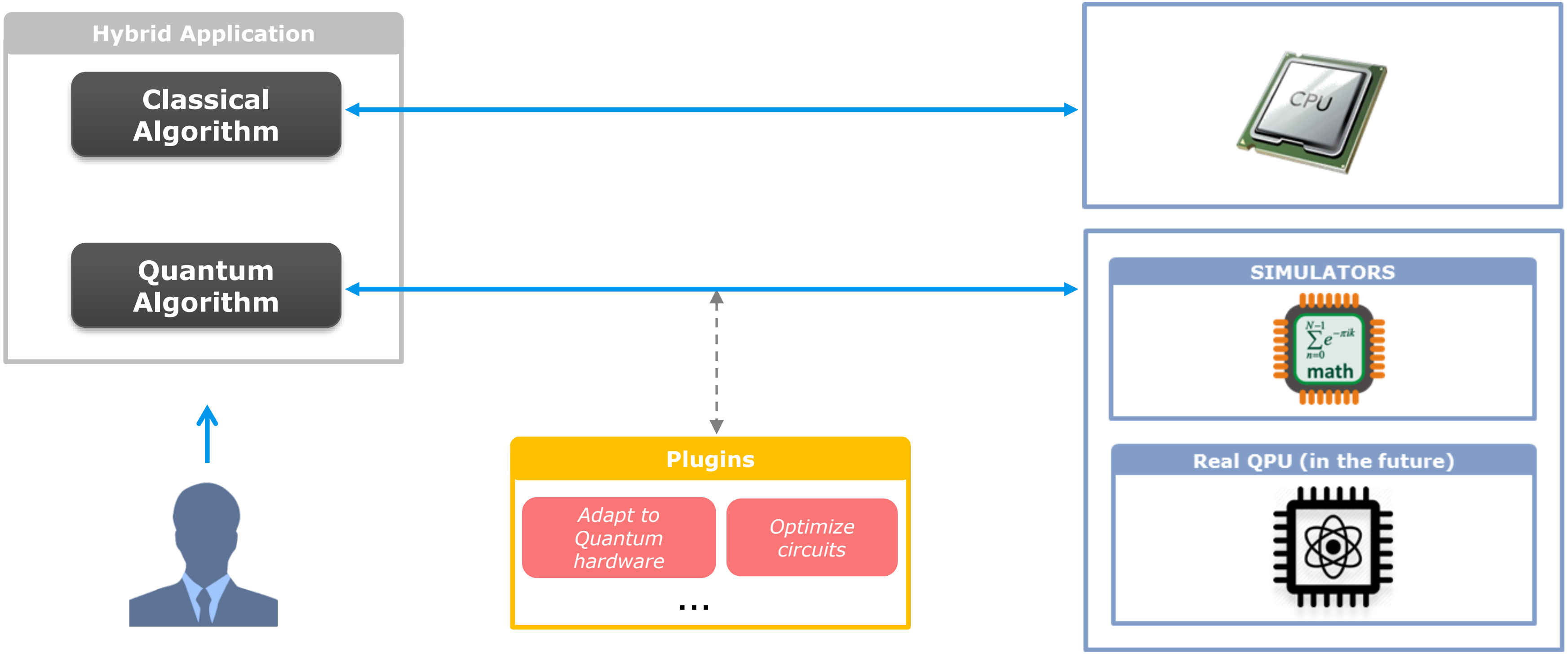
NOISY SIMULATION

A modular and scalable solution

Expand your simulation capabilities,
upgrade your Atos QLM



Hybrid application workflow on a QLM



Hybrid Application

Classical Algorithm

Quantum Algorithm

Plugins

Adapt to Quantum hardware

Optimize circuits

...

SIMULATORS

Real QPU (in the future)

myQLM

Discover AQASM and pyAQASM

WRITE

your own quantum algorithms

Explore Jupyter Notebook tutorials Adapt QLIB algorithms

On your laptop using pyLinalg or your own simulator

RUN & TEST

your quantum circuits

On your Atos QLM

Create myQLM user communities

SHARE

tips and codes with the community

Collaborate with other frameworks' users

Overview of myQLM

PROGRAMMING

AQASM
Assembly language to build quantum circuits

pyAQASM
Python extension to AQASM

CIRC
Binary format of quantum circuits

QLIB
AQASM & pyAQASM libraries
QRAM, oracle emulator, arithmetic libraries and more!

rigetti

Qiskit

Open Source

INTEROP
Connectors' source codes: build your own!

ProjectQ

Cirq

SIMULATION

Open Source

SIMULATOR
pyLinalg
Source code of this simulator: build your own!

Where can you get myQLM?

The myQLM website:

https://atos.net/en/lp/myqlm

The instructions to install myQLM:

https://myqlm.github.io/myqlm_specific/install.html

pyAQASM
part 1

Overview pyAQASM

Python

\+ AQASM

= Ease the developpement

Writing circuits in pyAQASM

```
from qat.lang.AQASM import Program, X, H, CNOT, SWAP
#Create a Program
my_program = Program()
#Allocate some qubits
qubits_reg = my_program.qalloc(4)
#Apply some quantum Gates
my_program.apply(X, qubits_reg[0])
my_program.apply(H, qubits_reg[1])
my_program.apply(SWAP, qubits_reg[1], qubits_reg[2])
my_program.apply(CNOT, qubits_reg[2], qubits_reg[3])
#Export this program into a quantum circuit
my_circuit = my_program.to_circ()
#And display it!
%qatdisplay my_circuit
```

Writing circuits in pyAQASM

Import functions

```
from qat.lang.AQASM import Program, X, H, CNOT, SWAP
#Create a Program
my_program = Program()
#Allocate some qubits
qubits_reg = my_program.qalloc(4)
#Apply some quantum Gates
my_program.apply(X, qubits_reg[0])
my_program.apply(H, qubits_reg[1])
my_program.apply(SWAP, qubits_reg[1], qubits_reg[2])
my_program.apply(CNOT, qubits_reg[2], qubits_reg[3])
#Export this program into a quantum circuit
my_circuit = my_program.to_circ()
#And display it!
%qatdisplay my_circuit
```

Writing circuits in pyAQASM

Import functions

Create your **program**

```
from qat.lang.AQASM import Program, X, H, CNOT, SWAP
#Create a Program
my_program = Program()
#Allocate some qubits
qubits_reg = my_program.qalloc(4)
#Apply some quantum Gates
my_program.apply(X, qubits_reg[0])
my_program.apply(H, qubits_reg[1])
my_program.apply(SWAP, qubits_reg[1], qubits_reg[2])
my_program.apply(CNOT, qubits_reg[2], qubits_reg[3])
#Export this program into a quantum circuit
my_circuit = my_program.to_circ()
#And display it!
%qatdisplay my_circuit
```

Writing circuits in pyAQASM

Import functions

Create your **program**

Allocate registers of qubits

```
from qat.lang.AQASM import Program, X, H, CNOT, SWAP
#Create a Program
my_program = Program()
#Allocate some qubits
qubits_reg = my_program.qalloc(4)
#Apply some quantum Gates
my_program.apply(X, qubits_reg[0])
my_program.apply(H, qubits_reg[1])
my_program.apply(SWAP, qubits_reg[1], qubits_reg[2])
my_program.apply(CNOT, qubits_reg[2], qubits_reg[3])
#Export this program into a quantum circuit
my_circuit = my_program.to_circ()
#And display it!
%qatdisplay my_circuit
```

Writing circuits in pyAQASM

Import functions

Create your **program**

Allocate registers of qubits

Apply gates

```
from qat.lang.AQASM import Program, X, H, CNOT, SWAP
#Create a Program
my_program = Program()
#Allocate some qubits
qubits_reg = my_program.qalloc(4)
#Apply some quantum Gates
my_program.apply(X, qubits_reg[0])
my_program.apply(H, qubits_reg[1])
my_program.apply(SWAP, qubits_reg[1], qubits_reg[2])
my_program.apply(CNOT, qubits_reg[2], qubits_reg[3])
#Export this program into a quantum circuit
my_circuit = my_program.to_circ()
#And display it!
%qatdisplay my_circuit
```

Writing circuits in pyAQASM

Import functions

Create your **program**

Allocate registers of qubits

Apply gates

Create your **circuit**

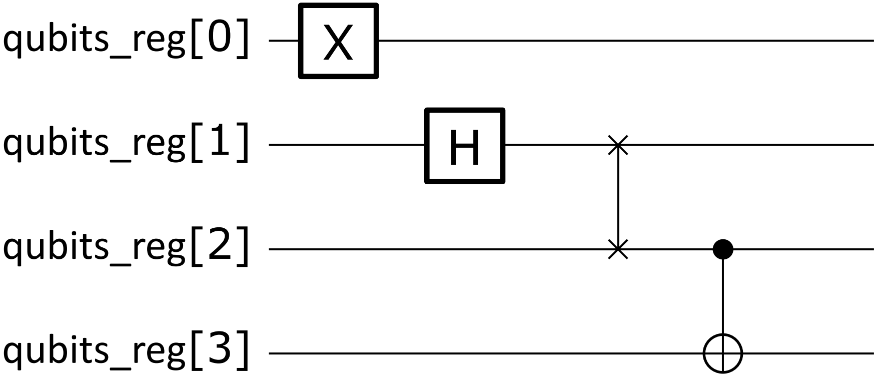
```
from qat.lang.AQASM import Program, X, H, CNOT, SWAP
#Create a Program
my_program = Program()
#Allocate some qubits
qubits_reg = my_program.qalloc(4)
#Apply some quantum Gates
my_program.apply(X, qubits_reg[0])
my_program.apply(H, qubits_reg[1])
my_program.apply(SWAP, qubits_reg[1], qubits_reg[2])
my_program.apply(CNOT, qubits_reg[2], qubits_reg[3])
#Export this program into a quantum circuit
my_circuit = my_program.to_circ()
#And display it!
%qatdisplay my_circuit
```

Writing circuits in pyAQASM

- **Import** functions
- **Create** your **program**
- **Allocate** registers of qubits
- **Apply** gates
- **Create** your **circuit**
- **Display** your circuit

```
from qat.lang.AQASM import Program, X, H, CNOT, SWAP
#Create a Program
my_program = Program()
#Allocate some qubits
qubits_reg = my_program.qalloc(4)
#Apply some quantum Gates
my_program.apply(X, qubits_reg[0])
my_program.apply(H, qubits_reg[1])
my_program.apply(SWAP, qubits_reg[1], qubits_reg[2])
my_program.apply(CNOT, qubits_reg[2], qubits_reg[3])
#Export this program into a quantum circuit
my_circuit = my_program.to_circ()
#And display it!
%qatdisplay my_circuit
```

Writing circuits in pyAQASM



```
from qat.lang.AQASM import Program, X, H, CNOT, SWAP
#Create a Program
my_program = Program()
#Allocate some qubits
qubits_reg = my_program.qalloc(4)
#Apply some quantum Gates
my_program.apply(X, qubits_reg[0])
my_program.apply(H, qubits_reg[1])
my_program.apply(SWAP, qubits_reg[1], qubits_reg[2])
my_program.apply(CNOT, qubits_reg[2], qubits_reg[3])
#Export this program into a quantum circuit
my_circuit = my_program.to_circ()
#And display it!
%qatdisplay my_circuit
```

List of constant available gates

Gate name	Keyword	Arity
Hadamard	H	1
Pauli X	X	1
Pauli Y	Y	1
Pauli Z	Z	1
Identity	I	1
S gate	S	1
T gate	T	1

Gate name	Keyword	Arity
Controlled NOT	CNOT	2
SWAP	SWAP	2
iSWAP	iSWAP	2
$\sqrt{SWAP}$	SQRTSWAP	2
Toffoli	CCNOT	3

Operations on gates

Operation name	Keyword	Example	Note
control	ctrl()	X.ctrl()	The first qubit of the list is the controller
dagger	dag()	S.dag()	Gates are unitary matrices

Executing circuits in pyAQASM

```
#import one Quantum Processor Unit Factory
from qat.qpus import LinAlg
#Create a Quantum Processor Unit
linalgqpu = LinAlg()
#Create a job
job = my_circuit.to_job()
#Submit the job to the QPU
result = linalgqpu.submit(job)
#Iterate over the final state vector to get all final components
for sample in result:
    print("State %s amplitude %s" % (sample.state, sample.amplitude))
```

Executing circuits in pyAQASM

Import functions

```
#import one Quantum Processor Unit Factory
from qat.qpus import LinAlg
#Create a Quantum Processor Unit
linalgqpu = LinAlg()
#Create a job
job = my_circuit.to_job()
#Submit the job to the QPU
result = linalgqpu.submit(job)
#Iterate over the final state vector to get all final components
for sample in result:
    print("State %s amplitude %s" % (sample.state, sample.amplitude))
```

Executing circuits in pyAQASM

Import functions

Get a simulator

```
#import one Quantum Processor Unit Factory
from qat.qpus import LinAlg
#Create a Quantum Processor Unit
linalgqpu = LinAlg()
#Create a job
job = my_circuit.to_job()
#Submit the job to the QPU
result = linalgqpu.submit(job)
#Iterate over the final state vector to get all final
components
for sample in result:
    print("State %s amplitude %s" % (sample.state,
sample.amplitude))
```

Executing circuits in pyAQASM

Import functions

Get a simulator

Create your job

```
#import one Quantum Processor Unit Factory
from qat.qpus import LinAlg
#Create a Quantum Processor Unit
linalgqpu = LinAlg()
#Create a job
job = my_circuit.to_job()
#Submit the job to the QPU
result = linalgqpu.submit(job)
#Iterate over the final state vector to get all final components
for sample in result:
   print("State %s amplitude %s" % (sample.state, sample.amplitude))
```

Executing circuits in pyAQASM

- **Import** functions
- **Get** a simulator
- **Create** your job
- **Submit** your job

```
#import one Quantum Processor Unit Factory
from qat.qpus import LinAlg
#Create a Quantum Processor Unit
linalgqpu = LinAlg()
#Create a job
job = my_circuit.to_job()
#Submit the job to the QPU
result = linalgqpu.submit(job)
#Iterate over the final state vector to get all final components
for sample in result:
   print("State %s amplitude %s" % (sample.state, sample.amplitude))
```

Executing circuits in pyAQASM

Import functions

Get a simulator

Create your job

Submit your job

Print the result

```
#import one Quantum Processor Unit Factory
from qat.qpus import LinAlg
#Create a Quantum Processor Unit
linalgqpu = LinAlg()
#Create a job
job = my_circuit.to_job()
#Submit the job to the QPU
result = linalgqpu.submit(job)
#Iterate over the final state vector to get all final components
for sample in result:
   print("State %s amplitude %s" % (sample.state, sample.amplitude))
```

Executing circuits in pyAQASM

```
State |1000> amplitude
(0.7071067811865475+0j)
State |1011> amplitude
(0.7071067811865475+0j)
```

```python
#import one Quantum Processor Unit Factory
from qat.qpus import LinAlg
#Create a Quantum Processor Unit
linalgqpu = LinAlg()
#Create a job
job = my_circuit.to_job()
#Submit the job to the QPU
result = linalgqpu.submit(job)
#Iterate over the final state vector to get all final components
for sample in result:
   print("State %s amplitude %s" % (sample.state, sample.amplitude))
```

pyAQASM: Job

```
job = circuit.to_job(*options*) #creating job from circuit.
results = qpu.submit(job)       #submitting job to QPU instance, getting results.
```

Circuit execution modes:

- Full distribution (*default case*)
- Strictly emulate
- Directly compute observable averages *(Advance topic)*

pyAQASM: Job

- Full distribution (*default case*):

```
job = circuit.to_job()            #creating job from circuit to get full distribution.
results = qpu.submit(job)         #submitting job to QPU instance, getting results.
```

results contains **all states with non-zero amplitude**.

The job was created with *default arguments*:

for example **nbshots = 0**, by convention, it means the **qpu** returns the **best it can do**.

pyAQASM: Job

- Strictly emulate:

```
job = circuit.to_job(nbshots = 6, aggregate_data=False)#creating job from circuit to get 6 measures.
results = qpu.submit(job)                              #submitting job to QPU instance, getting results.
```

results contains **6 measurements** that you can print:

```
for state in results:
    print(state)
```

```
Sample(state=|00>, probability=None, amplitude=None, intermediate_measurements=None, err=None)
Sample(state=|00>, probability=None, amplitude=None, intermediate_measurements=None, err=None)
Sample(state=|11>, probability=None, amplitude=None, intermediate_measurements=None, err=None)
Sample(state=|11>, probability=None, amplitude=None, intermediate_measurements=None, err=None)
Sample(state=|00>, probability=None, amplitude=None, intermediate_measurements=None, err=None)
Sample(state=|11>, probability=None, amplitude=None, intermediate_measurements=None, err=None)
```

pyAQASM: Job

- Strictly emulate (with *aggregate_data*):

```
job = circuit.to_job(nbshots = 6)        #creating job from circuit to aggregate 6 measures.
results = qpu.submit(job)                #submitting job to QPU instance, getting results.
```

results contains **one unique sample** per possible output with an **empirical estimation of the probability**:

```
for state in results:
    print(state)
```

```
Sample(state=|00>, probability=0.5, amplitude=None, intermediate_measurements=None, err=0.16666666666666666)
Sample(state=|11>, probability=0.5, amplitude=None, intermediate_measurements=None, err=0.16666666666666666)
```

pyAQASM: Job

- Subset of qubits:

```
job = circuit.to_job(qubits=[0])          #creating job from circuit only on the first qubit
results = qpu.submit(job)                  #submitting job to QPU instance, getting results.
```

results contains **all possible states with a non-zero probability** for the subset of qubits.

```
for state in results:
    print(state)
```

```
Sample(state=|0>, probability=0.4999999999999999, amplitude=None, intermediate_measurements=None, err=None)
Sample(state=|1>, probability=0.4999999999999999, amplitude=None, intermediate_measurements=None, err=None)
```

pyAQASM: Job

```
help(circ.to_job)
```

```
Help on method to_job in module qat.core.wrappers.circuit:

to_job(job_type='SAMPLE', qubits=None, nbshots=0, aggregate_data=True, amp_threshold=9.094947017729282e-13,
**kwargs) method of qat.core.wrappers.circuit.Circuit instance
    Generates a Job containing the circuit and some post
    processing information.

    Args:
        job_type (str): possible values are "SAMPLE" for computational basis sampling
            of some qubits, or "OBS" for observable evaluation (see :py:mod:`qat.application.observables`
            for more information about this mode).
        qubits (optional, list<int>, list<QRegister>): the list of qubits to measure (in "SAMPLE" mode).
            If some quantum register is passed instead, all the qubits of the register will be…
```

pyAQASM: Simulators

Perfect simulators:

- **LinAlg**: Linear-algebra simulator
- **MPS**: Matrix Product State
- **Stabs**: Stabilizer formalism
- **Feynman**: Path integral formulation
- **BDD**: Binary Decision Diagrams

Noisy simulators *(Advance topic)*

How to log on a QLM?

Hands-on 1:
EPR pair

Hands-on 1: EPR pair

- Log on the QLM

- Go to Hands-on 1 directory :

  http://127.0.0.1:8888/tree/notebooks/Hands-on1

- Open and complete the notebook *Helloworld.ipynb*

Thank you.

Gaëtan Rubez

Quantum Computing Expert for the CEPP

gaetan.rubez@atos.net