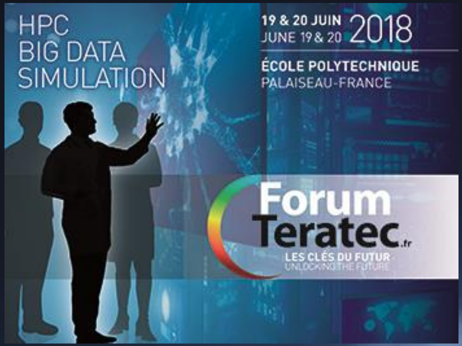
Atos QLM, a future-proof approach to quantum computing

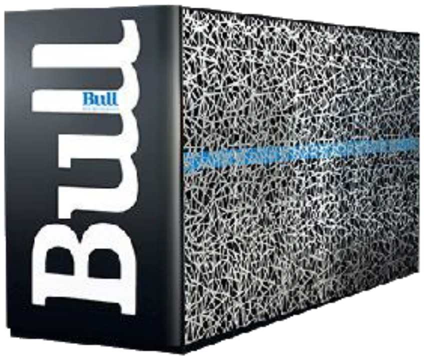


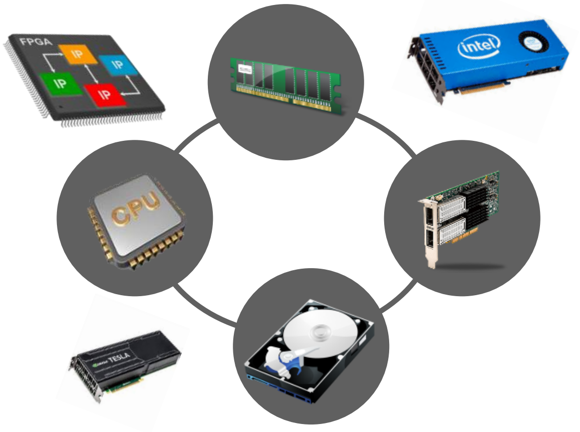
Christelle Piechurski, Atos
HPC Principal Architect &
Atos Quantum Computing Solution Product Manager

Thomas Ayral, PhD
Research Engineer in the Atos Quantum lab.

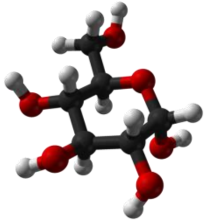
HPC, Driving innovation

More Computing Power To Address New Applications





CyberSecurity



Medical, Chemistry

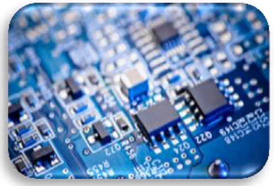
Weather Forecast

Artificial Intelligence

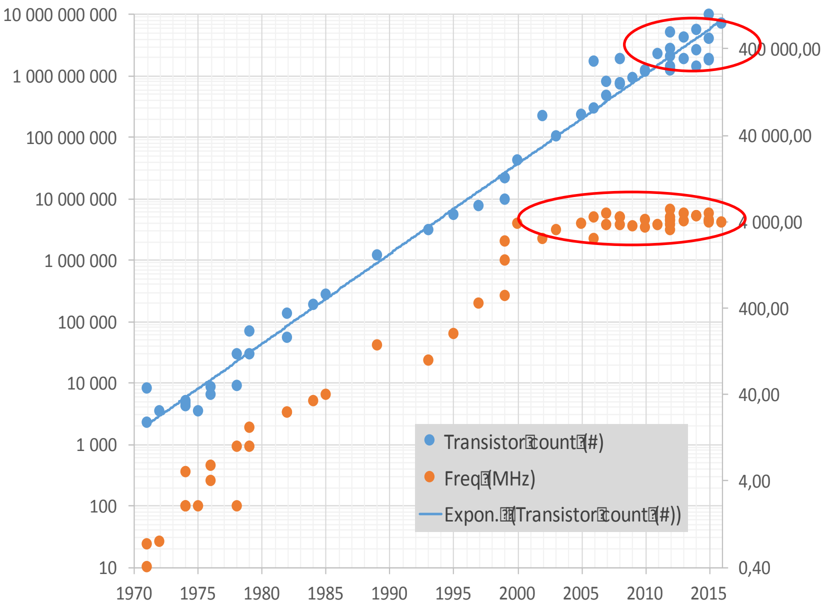
Atos

End Of Moore's Law

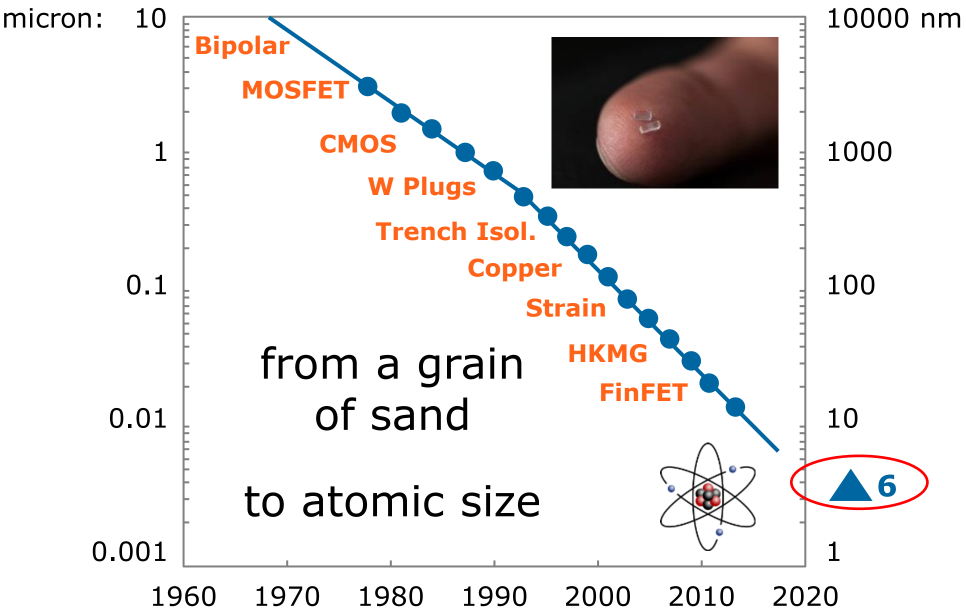
Limits Of Miniaturization Reaching Classical Computers



More transistors, higher frequencies



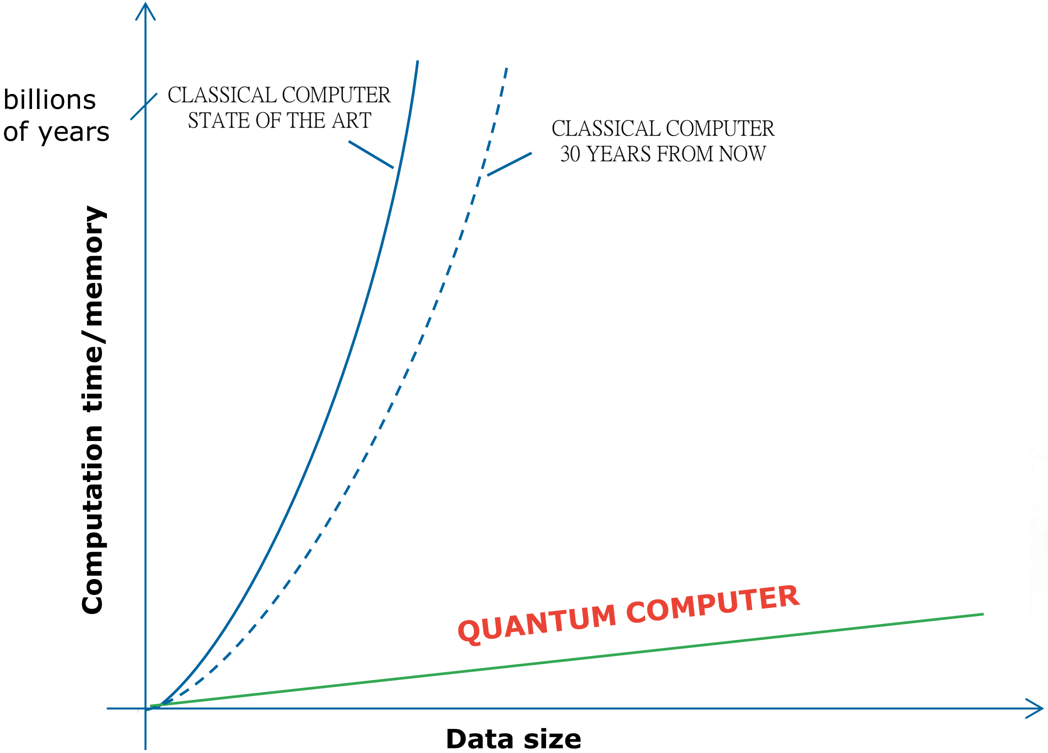
New technologies for thinner chips



Atos

Quantum Computing Speedup

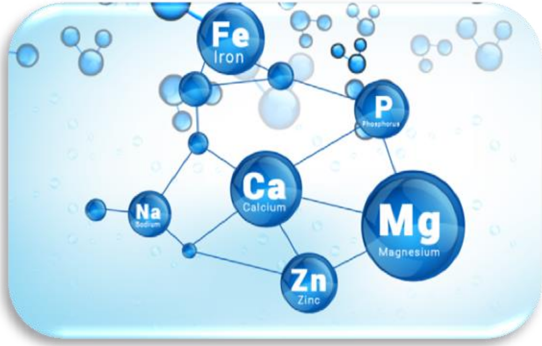
Applications focus



Some of the Applications Domain

Quantum Speedup expected

Cryptography
Integer factorization
Shor algorithm & derivatives
exponential speedup



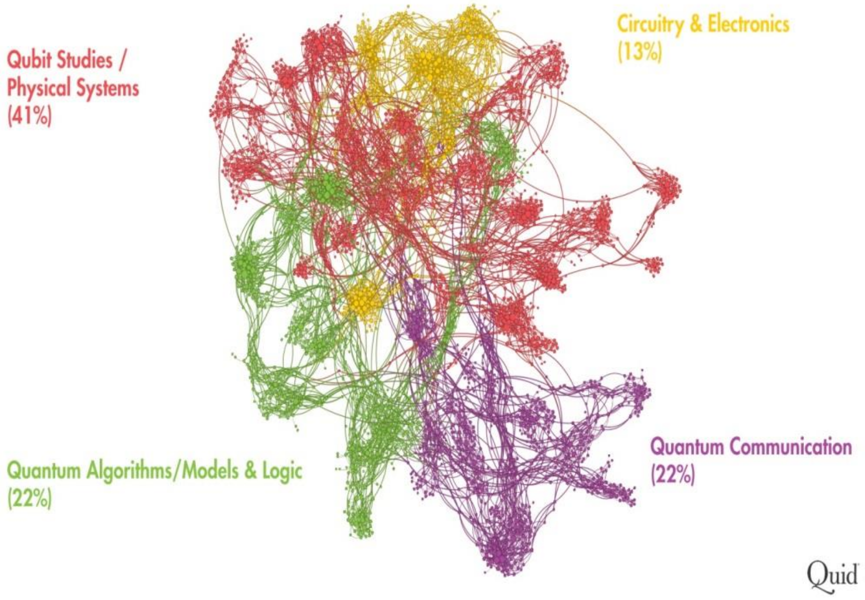
Chemistry, science of materials
Hamiltonian Simulation

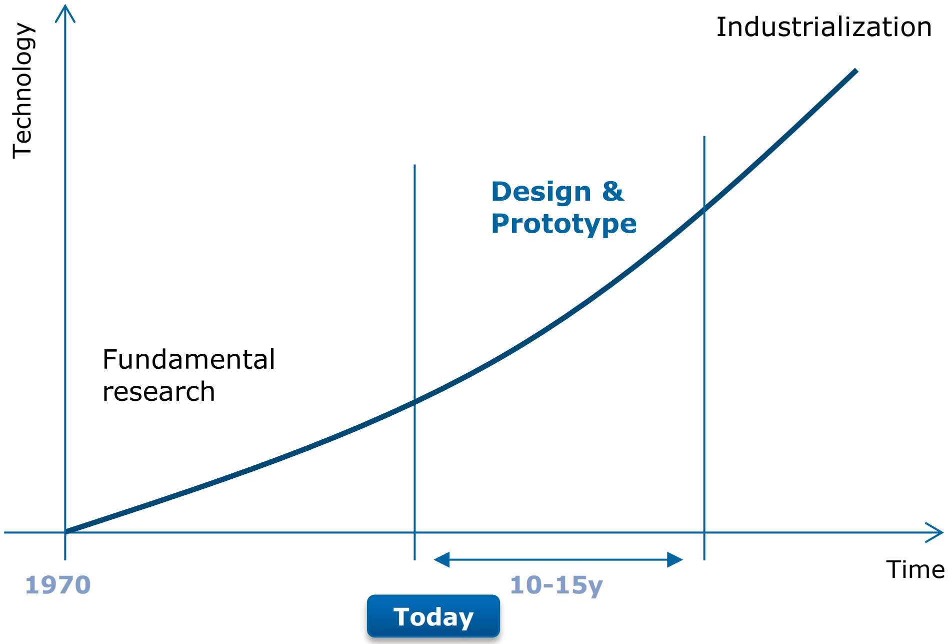
Quantum database search
Grover algorithm and affiliates – polynomial speedup

Statistical Analysis
Mathematical computation
exponential speedup

Quantum computing research areas

Where are we now?

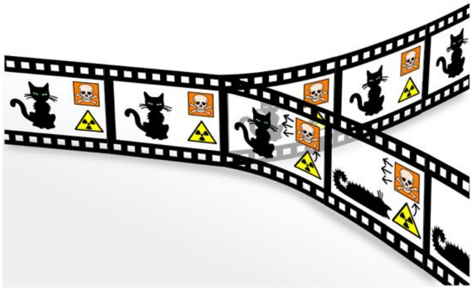




Why don't we have a Quantum Computer today?

- Only Labs are working on experimentation to create physical qubits with HUGE constraints:

  – Most of the experimentation only work near absolute zero (-273.15°C)
  – Quantum stability last only few seconds
  – Quantum entanglement and probability generate noise, so data is not reliable
  – etc.

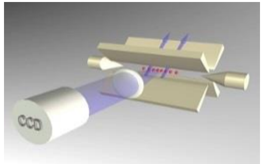


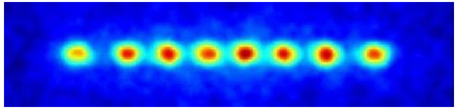
Quantum Current Leading Technologies

Not Any Standard Implementation

Trapped Ions

- \+ High level of control, rather long coherence (s), operation conditions 5Kelvin even higher)
- − slow (µs) , scalability
- − current state of the art: 14 qubits





Superconducting

- \+ speed (ns), size, electronic
- − coherence time (<100 µs), low fidelity, conditions (30 mK)
- − current state of the art : 20 qubits IBM, 49 qubits Intel, 72 qubits Google

Atos QLM
A quantum simulator as an appliance

- Enabling end-users preparing themselves now for the arrival of the first generation of GPQPU
- Allowing quantum algorithms development without quantum hardware constraints
- Offering a unique software environment for users without having to modify quantum algo. when real GPQPU available

The Atos Quantum Learning Machine

Unique Features

1 **Universal technology quantum language**

2 **Genuine hybrid classic-quantum programming**

3 **Powerful simulation**

4 **Modular and Scalable on premises appliance**

5 **High extensibility and interoperable**

6 **Hardware agnostic**

Atos QLM Software stack

Functional Scope

PROGRAMMING

AQASM
Assembly language to build quantum circuits

pyAQASM
Python extension to AQASM

CIRC
Binary format of quantum circuits

QLIB
AQASM & pyAQASM libraries

INTEROP
Connectors with other frameworks

INTERFACE

QPU
Quantum processing unit interface

OPTIMIZATION

RBO
Rule based optimizer

Circuit Optimizer
Generic circuit optimizer

NNIZER
Topology constraint solver

SIMULATION

SIMULATORS
Simulation modules

PHYSICS
Physical Noise models

SIM OPTIMIZER
Best Simulator dynamic selection

A Noisy QFT on the Quantum Learning Machine
Demonstration

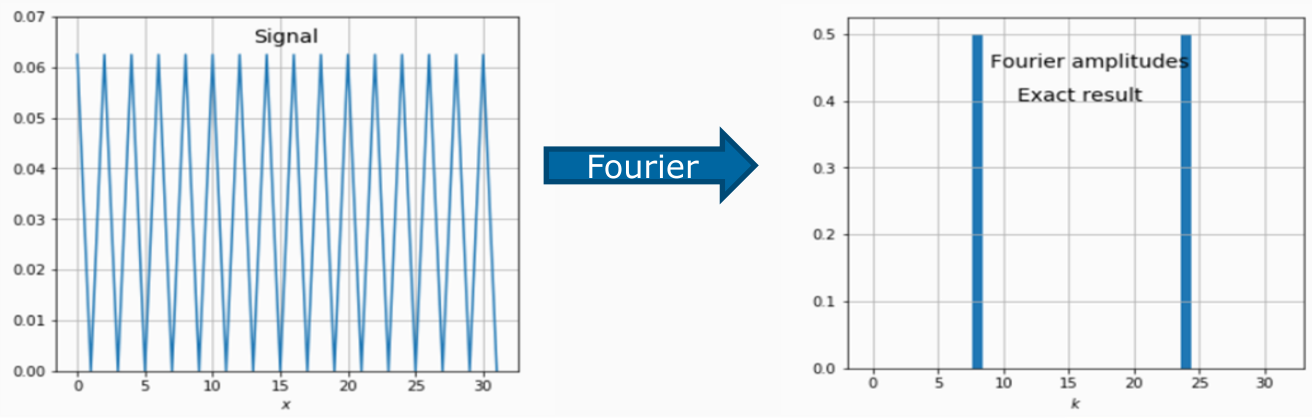
- Simulate the (noisy) IBM processor on the QLM with a Jupyter notebook
- IBM Quantum Experience: 5 superconducting qubits

Atos

Part I: Writing and optimizing a quantum program

Example: Quantum Fourier transform, $QFT(|x\rangle) = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} \left(e^{\frac{2i\pi}{2^n}}\right)^{xk} |k\rangle$

Key ingredient to e.g **Shor's factoring algorithm**.



Atos

Writing the quantum program

The QLM provides usual quantum gates + libraries of quantum routines

In [1]:
```
from qat.lang.AQASM import *
from qat.lang.AQASM.qftarith import QFT_rev
from demo_init import init_routine
nqbits = 5
prog = Program()
reg = prog.qalloc(nqbits)
prog.apply(init_routine.gate(5),reg)
prog.apply(QFT_rev(nqbits), reg)
prog.apply(SWAP,reg[0],reg[4])
prog.apply(SWAP,reg[1],reg[3])
qft_circuit = prog.to_circ(keep="Init") #convert program to circuit
%qatdisplay qft_circuit
```

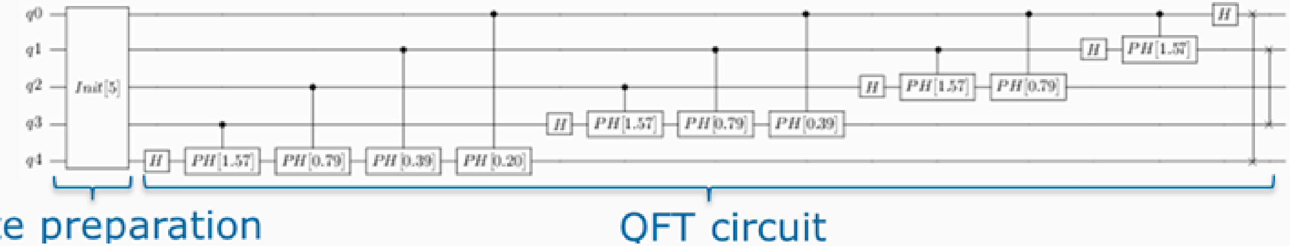

Atos

Writing the quantum program

The QLM provides usual quantum gates + libraries of quantum routines

In [1]:
```
from qat.lang.AQASM import *
from qat.lang.AQASM.qftarith import QFT_rev
from demo_init import init_routine
nqbits = 5
prog = Program()
reg = prog.qalloc(nqbits)
prog.apply(init_routine.gate(5),reg)
prog.apply(QFT_rev(nqbits), reg)
prog.apply(SWAP,reg[0],reg[4])
prog.apply(SWAP,reg[1],reg[3])
qft_circuit = prog.to_circ(keep="Init") #convert program to circuit
%qatdisplay qft_circuit
```

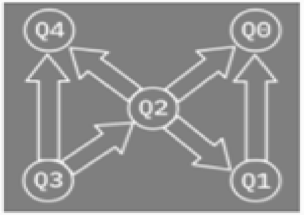
Out[1]:



Atos

Optimization

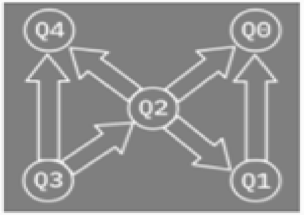
IBM QX4 has a limited qubit connectivity, and accepts only CNOT gates for two-qubit operations:



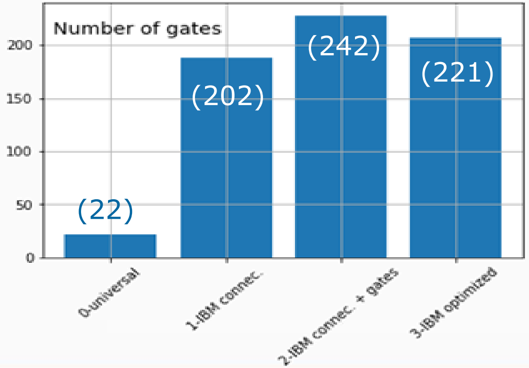
In [2]:
```
from qat.core.simutil import optimize_circuit ; import qat.nnize, qat.graphopt
qft_circuit = prog.to_circ()
qft_conn_circ = optimize_circuit(qft_circuit, qat.nnize.Nnizer(directed=True, topology="graph_ibmqx4.json"))
qft_conn_gates_circ = optimize_circuit(qft_conn_circ, qat.graphopt.Graphopt(expandonly=True))
qft_optimized_circ = optimize_circuit(qft_conn_gates_circ, qat.graphopt.Graphopt(directed=True))

%matplotlib inline
import numpy as np, matplotlib.pyplot as plt
plt.bar(["0-universal", "1-IBM connec.", "2-IBM connec. + gates", "3-IBM optimized"],  [len(c.ops) for c in [qft_circuit, qf
plt.xticks(rotation=45); plt.text(-0.5,210,"Number of gates", size=14); plt.grid();
```

Optimization

IBM QX4 has a limited qubit connectivity, and accepts only CNOT gates for two-qubit operations:




```
In [2]: from qat.core.simutil import optimize_circuit ; import qat.nnize, qat.graphopt
qft_circuit = prog.to_circ()
qft_conn_circ = optimize_circuit(qft_circuit, qat.nnize.Nnizer(directed=True, topology="graph_ibmqx4.json"))
qft_conn_gates_circ = optimize_circuit(qft_conn_circ, qat.graphopt.Graphopt(expandonly=True))
qft_optimized_circ = optimize_circuit(qft_conn_gates_circ, qat.graphopt.Graphopt(directed=True))

%matplotlib inline
import numpy as np, matplotlib.pyplot as plt
plt.bar(["0-universal", "1-IBM connec.", "2-IBM connec. + gates", "3-IBM optimized"],  [len(c.ops) for c in [qft_circuit, qt
plt.xticks(rotation=45); plt.text(-0.5,210,"Number of gates", size=14); plt.grid();
```



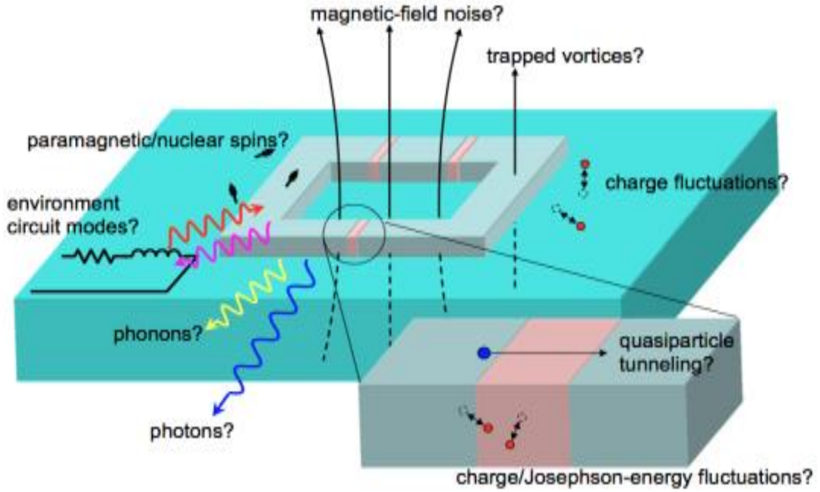
- 9% gates

Atos

Part II: Noisy simulation

Live simulation of noisy simulation on Atos QLM, with **simplified hardware model**:

- "Idle" qubits suffer from decoherence: **amplitude damping** (A.D) and **pure dephasing** (P.D).
- Relaxation and dephasing times from constructor: T_1 and T_2



Atos

Defining a noise model

In [3]:
```python
from qat.hardwares.default import DefaultGatesSpecification
from qat.quops.quantum_channels import ParametricPureDephasing, ParametricAmplitudeDamping
from qat.hardwares.default import HardwareModel

gate_durations = {"H":60, "X":120, "Y":120, "S":1,"T":1, "DAG(T)":1, "Z":1, "RZ":lambda angle : 1,
                  "PH": lambda angle : 1, "CNOT":386}

ibm_gates_spec = DefaultGatesSpecification(gate_durations)

T1, T2 = 44000, 38900 #nanosecs

amp_damping = ParametricAmplitudeDamping(T_1 = T1)
pure_dephasing = ParametricPureDephasing(T_phi = 1/(1/T2 - 1/(2*T1)))

ibm_hardware = HardwareModel(ibm_gates_spec, idle_noise = [amp_damping, pure_dephasing])
```

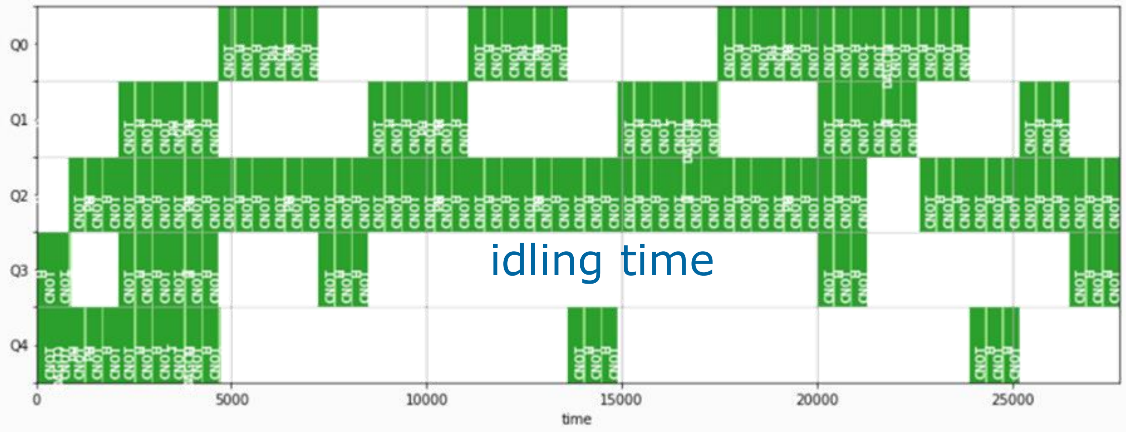

Atos

Temporal representation of the quantum algorithm

```
In [4]: %qatdisplay qft_optimized_circ ibm_hardware
```

Temporal representation of the quantum algorithm

In [4]:
```
%qatdisplay qft_optimized_circ ibm_hardware
```



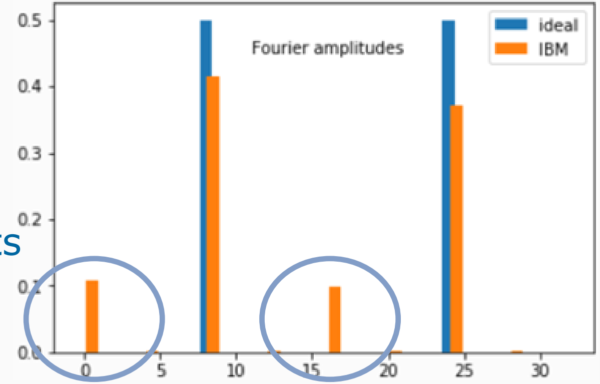
Result: Noisy Fourier spectrum

In [5]:
```
from qat.core.task import Task
from qat.mps import get_qpu_server
from qat.noisy import get_qpu_server as get_noisy_qpu_server
ideal_qpu = get_qpu_server(lnnize=True)
noisy_qpu = get_noisy_qpu_server(hardware_model = ibm_hardware, sim_method = "deterministic")

for nqpu, (qpu, label) in enumerate([(ideal_qpu, "ideal"), (noisy_qpu, "IBM")]):
    task = Task(qft_optimized_circ, qpu)
    allprob=np.zeros(shape=(2**nqbits))
    for result in task.states():
        allprob[result.state]=result.probability
    plt.bar(np.arange(2**nqbits)+0.5*nqpu,allprob, label = label);
plt.legend(); plt.xlabel("x"); plt.text(11, 0.45, "Fourier amplitudes");
```

Result: Noisy Fourier spectrum

In [5]:
```
from qat.core.task import Task
from qat.mps import get_qpu_server
from qat.noisy import get_qpu_server as get_noisy_qpu_server
ideal_qpu = get_qpu_server(lnnize=True)
noisy_qpu = get_noisy_qpu_server(hardware_model = ibm_hardware, sim_method = "deterministic")

for nqpu, (qpu, label) in enumerate([(ideal_qpu, "ideal"), (noisy_qpu, "IBM")]):
    task = Task(qft_optimized_circ, qpu)
    allprob=np.zeros(shape=(2**nqbits))
    for result in task.states():
        allprob[result.state]=result.probability
    plt.bar(np.arange(2**nqbits)+0.5*nqpu,allprob, label = label);
plt.legend(); plt.xlabel("x"); plt.text(11, 0.45, "Fourier amplitudes");
```

spurious components



Our customers

Atos Quantum Computing Scientific Community (aQCSC)

Oak Ridge National Laboratory

Jülich Forschungszentrum

SURF SARA

Université de Reims Champagne-Ardenne

Argonne National Laboratory

fh Oberösterreich University of Applied Sciences Upper Austria

Atos

Thanks

Bull
atos technologies

Atos Quantum Assembly Language

In [7]:
```
prog.export("qft.aqasm")
%cat qft.aqasm
```

```
BEGIN
qubits 5
cbits 5

Init[5] q[0],q[1],q[2],q[3],q[4]
H q[4]
CTRL(PH[1.5707963267948966]) q[3],q[4]
CTRL(PH[0.7853981633974483]) q[2],q[4]
CTRL(PH[0.39269908169872414]) q[1],q[4]
CTRL(PH[0.19634954084936207]) q[0],q[4]
H q[3]
CTRL(PH[1.5707963267948966]) q[2],q[3]
CTRL(PH[0.7853981633974483]) q[1],q[3]
CTRL(PH[0.39269908169872414]) q[0],q[3]
H q[2]
CTRL(PH[1.5707963267948966]) q[1],q[2]
CTRL(PH[0.7853981633974483]) q[0],q[2]
H q[1]
CTRL(PH[1.5707963267948966]) q[0],q[1]
H q[0]
SWAP q[0],q[4]
SWAP q[1],q[3]
END
```