Quantum Computing

Opportunities and Challenges for High Performance Computing

Simon Martiel

Bull
atos technologies

ATOS & Bull

- Our core business at Bull:

Bull
atos technologies

ATOS & Bull

- Our main interest: next generation of quantum supercomputers

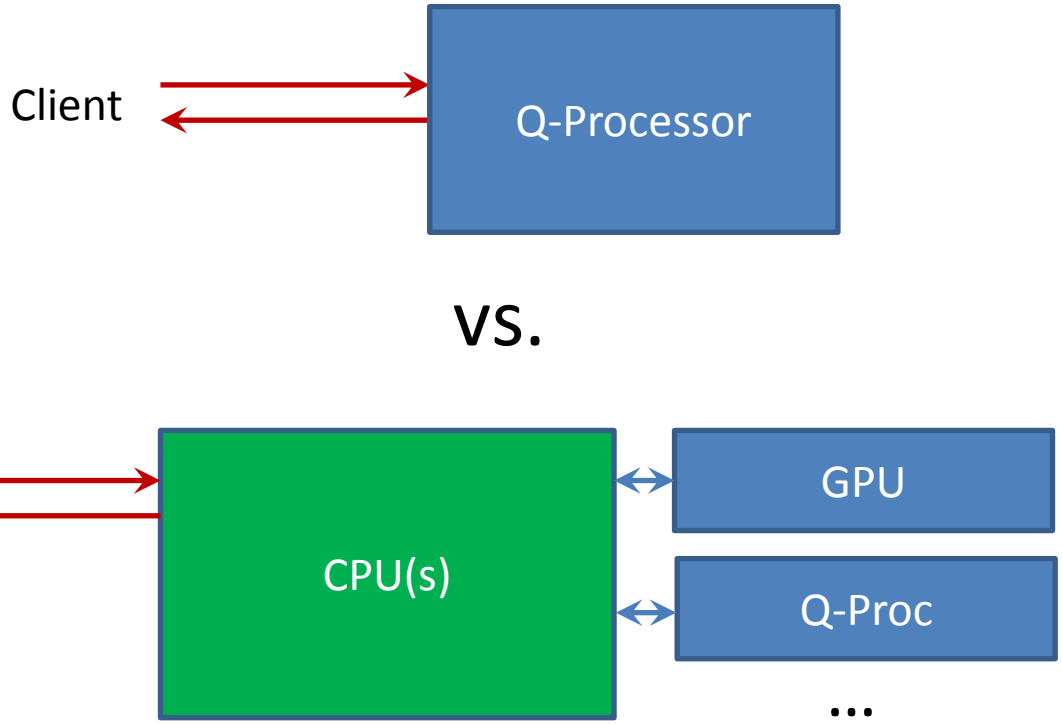
- The goal of the quantum project:
  - Quantum Software
    - Developing complete quantum programming platform
    - HPC-powered classical quantum simulation
  - Quantum Hardware
    - Qualifying qubit technologies for software purposes
    - Partnering with best European labs on each qubit technology
    - Investigating next-generation hybrid system architectures

And many collaborations

- Quantum computing hardware : modelling and experimenting of physical implementations of algorithms of interest (machine learning, HPC)
- Error correction & error tolerant computing capacity of qubit technologies
- Distributed quantum computing
- Applications of quantum simulation to numerical optimizations
- … supporting any other topic with valuable adherence to our objectives

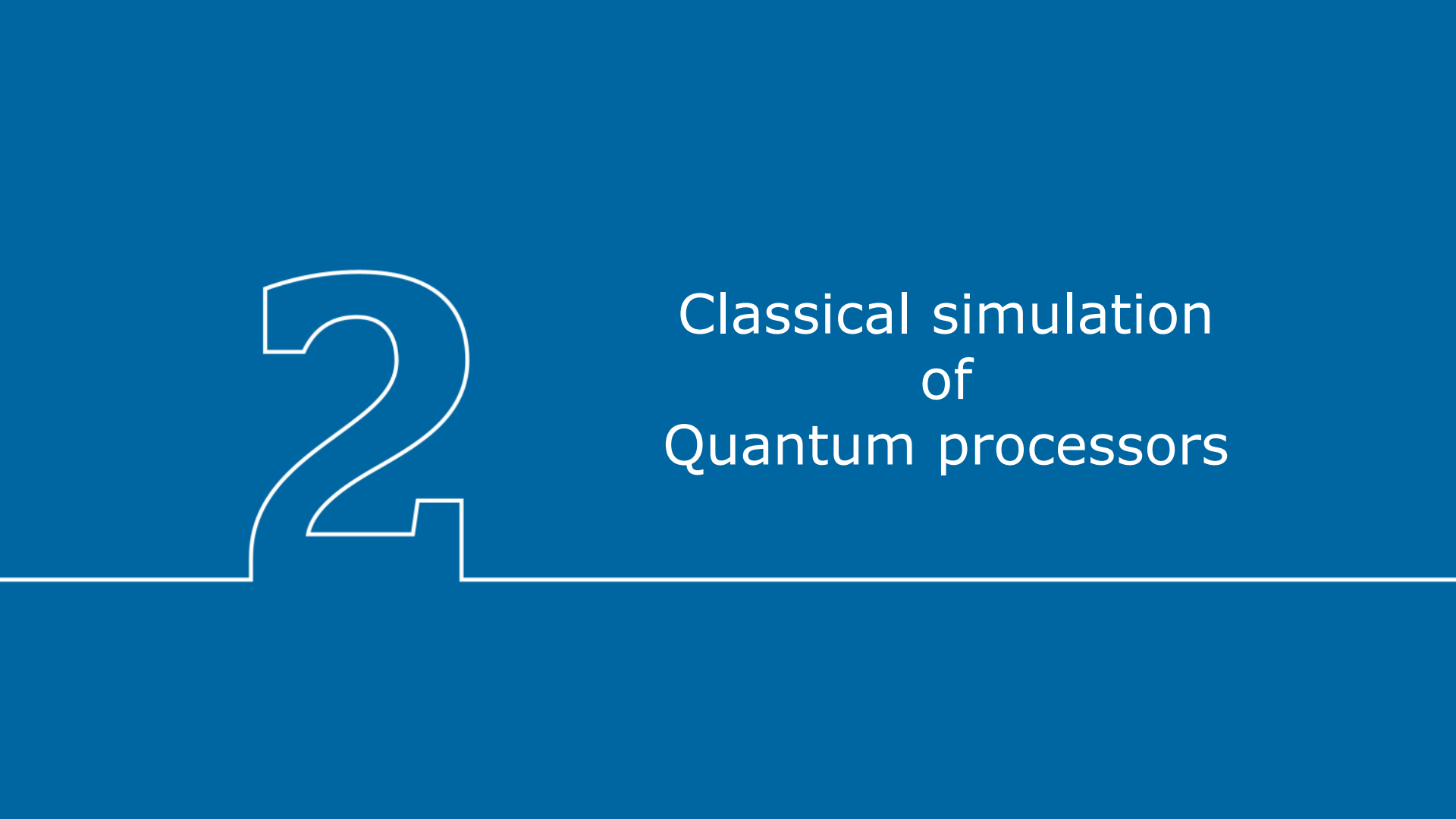
Target Architecture

Target architecture



Target architecture

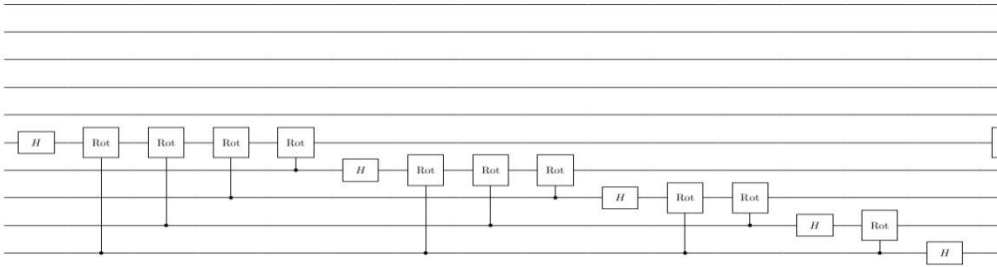
- Quantum processors as highly specialized co-processors
- Requires a proper environment:
  - to generate quantum programs (programmation)
  - to test/debug those programs
  - to simulate/predict their behaviour/performances
- All that without committing to a particular hardware architecture:
  - trapped ions?
  - quantum optics?
  - supra conductors?


2
Classical simulation
of
Quantum processors

Why do we want simulation?

- For debugging purpose:
  - is my algorithm correct?
  - does the 13^{th} ancillary qubit go back to $|0\rangle$?
- And for more long term reasons:
  - Validating a “real” quantum processor?
  - Calibrating it? (noise models)
  - Some programs might be easier to simulate classically (why do it quantumly?)

What are quantum programs?



- Memory/internal state:
  - a quantum state
  (i.e a normed vector)
  - over a finite number of qbits
  (i.e Hilbert space of finite dim.)
- Program/instructions:
  - unitary operators (i.e norm preserving matrices)
  - applying an op. = multiplying the state vector by the matrix

So, lets write a simulator!

- Naïve approach:
  - memory: the state vector (array of amplitudes)
  - program: a sequence of matrices (a list of arrays of complex numbers)
  - simulation: matrix multiplication
- “Small” issue:
  - state of 1 qbit = a normed vector in $\mathbb{C}^2$ -> 2 complex amplitudes to store
  - state of 2 qbits = a normed vector in $\mathbb{C}^2 \otimes \mathbb{C}^2$ -> 4 amplitudes to store
  - n qbits -> 2^n amplitudes to store
  - 40 qbits -> $\sim 10^{12}$ amplitudes -> terabyte range
  - n qbit gate -> $2^n \times 2^n$ matrix to apply ($\sim 2^{2n}$ operations)
- Can we do better?

Efficient simulation?

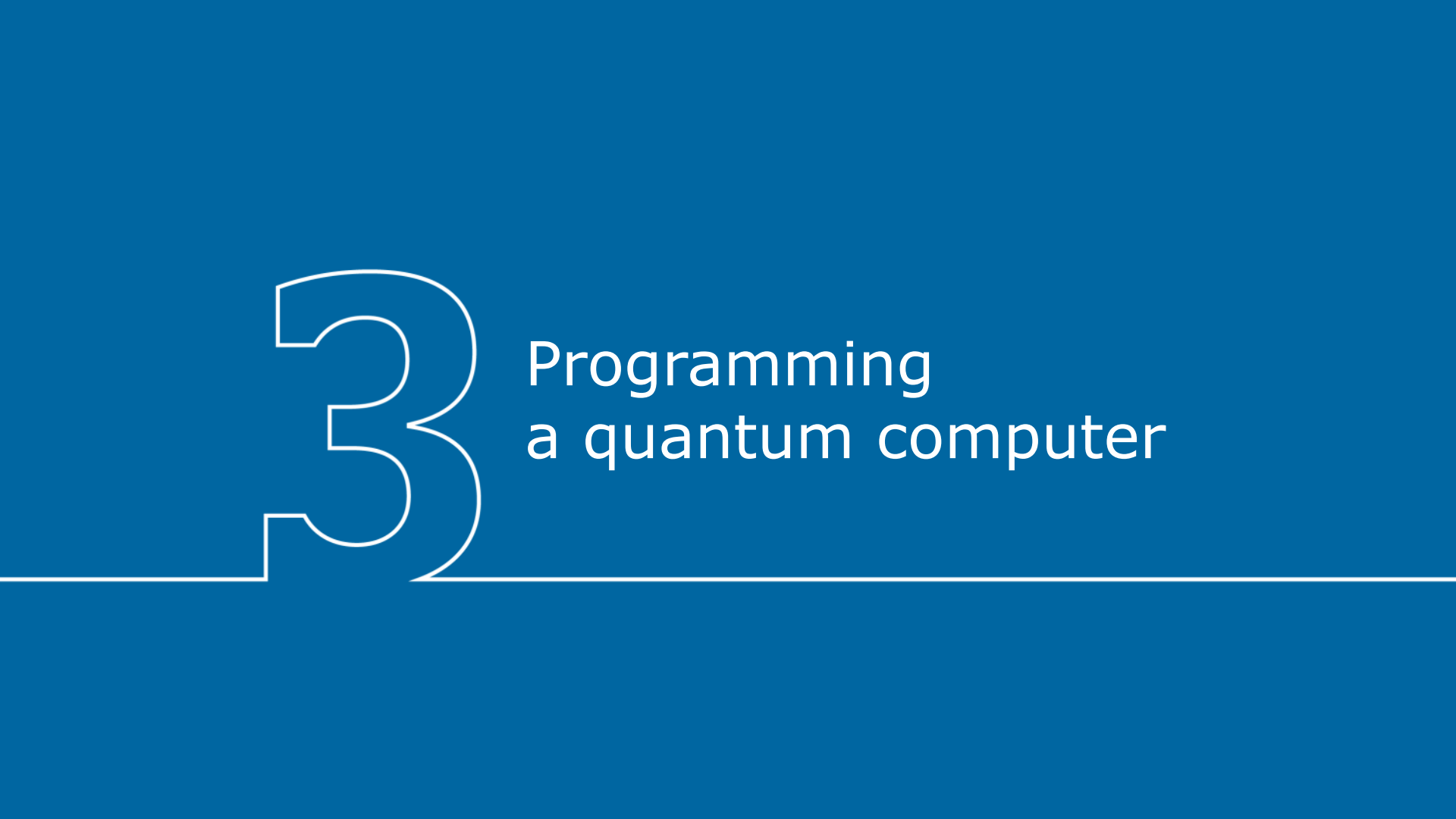
- Not in the general case!
- Clifford operators: a subset of gates that stabilize the Pauli group
  - O(n^2) to store a stabilizer state
  - Clifford gate application is polynomial
  - non-clifford is exponential (creates superpositions of stablizer states)
- MPS: a data structure to store a state
  - small size if non-entangled (down to linear if no entanglement)
  - exponential if maximally entangled
  - efficient for low entanglement simulation
- Of course:
  - Clifford operators are not universal (conjectured not even classically)
  - MPS structure explodes with entanglement

In practice

- Currently, we have:
  - A general simulator (optimized Linear algebra/matrix mult.)
  - A MPS-based simulator (low entanglement)
  - a path integral simulator (highly parallel algorithm)
  - a stabilizer-based simulator (Clifford gates)
- All simulators are behind an abstract interface (Thrift generated):
  - the programmer cannot see the simulation
  - faithful emulation of a quantum co-processor

Long term goals

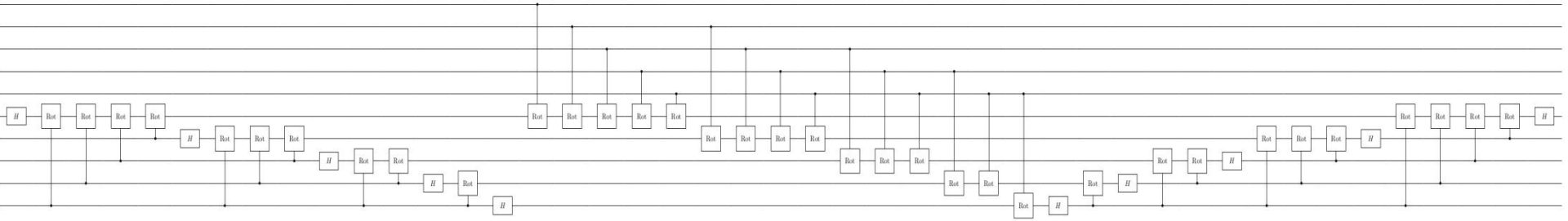
- Deciding which simulator to use:
  - feed the simulators a lot of circuits
  - record time and memory usage
  - run a machine learning algorithm to decide what simulator to use
- What are the criterions to track?
  - number of qbits?
  - type of operators (“classical”, dense, sparse, clifford, non-clifford,etc…)
  - entanglement? (can we predict it?)
  - ??
- Ideally, everything should be automated.


3
Programming
a quantum computer

What is difficult?

- Current state:
  - Programs are sequences of gates (basic operations) applied on qbits (quantum circuit)
  - Think assembly level of the 70's
  - 30 years of compilation theory to reinvent!

  (GCC is 30 years old / last critical improvement ~5 years ago)



This is a modular addition of two 5-(q)bits integers (using QFTs)

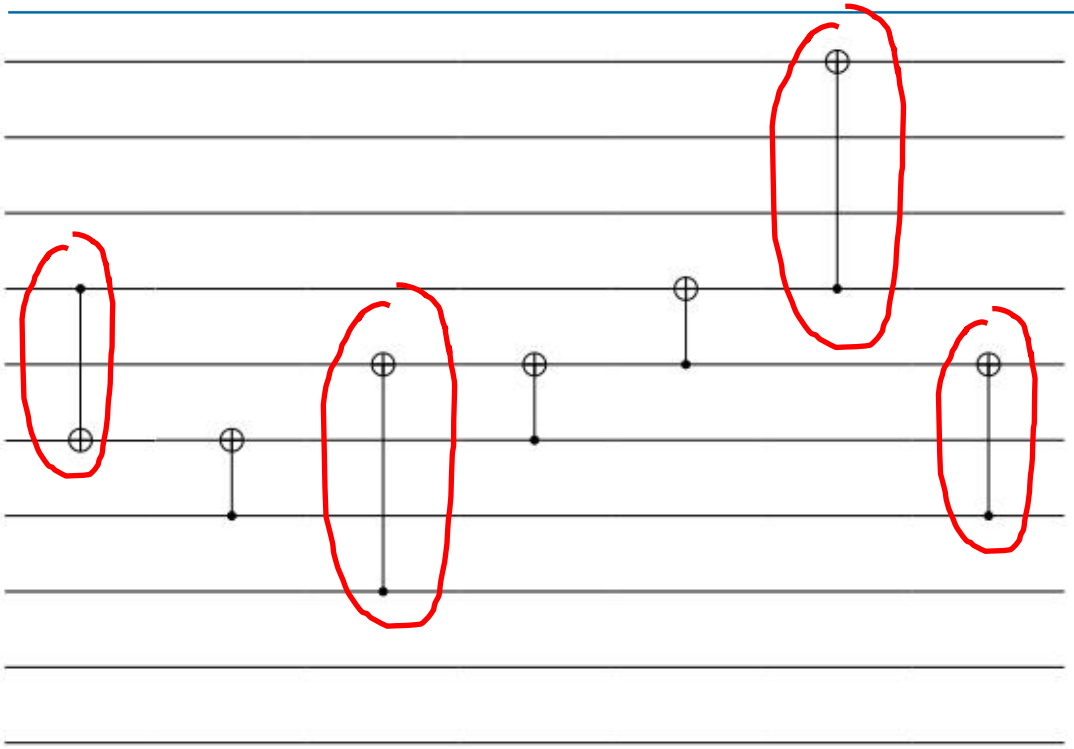
What is difficult?

- We need higher level programming language:
  - Already some languages/environments
  - Some are proprietary and bound to a precise simulator (e.g Microsoft liquiD)
  - Some are open but not optimized (e.g Quipper)

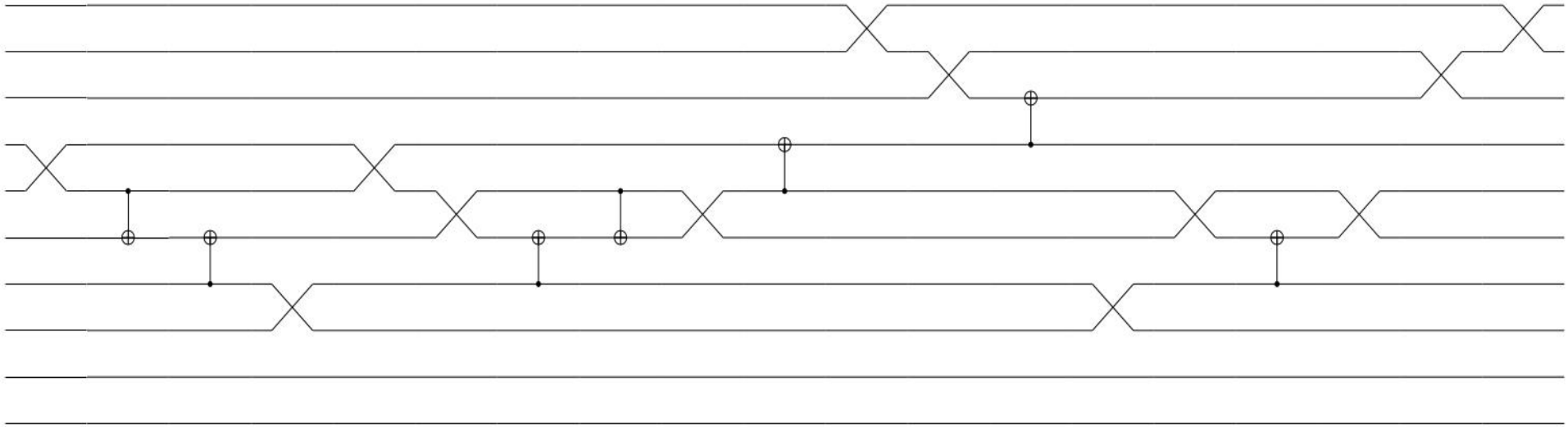
- Our approach:
  - a low level assembly language (quantum circuits)
  - generated by higher level languages (python libraries)
  - everything that is produced can be simulated (in terms of functionalities)
  - circuit generation, optimization/compilation, and simulation are separated

Circuit compilation and optimization

- Targeting particular hardware:
  - each hardware has its own (universal?) set of gates
  - how to decompose a circuit in a particular set of gates?
  - efficiently?
- Another example of constraint: locality
  - qbits are physically place on a line
  - only gates on neighboring qbits
  - compiling = inserting swaps to « localize » the circuit



Not local!



Circuit compilation and optimization

- All of this must be done efficiently:
  - use of heuristics/approximation algorithms
  - distributed if possible (GPU/MPI)
- Those are purely classical problems!
- Already some interests:
  - MPS simulation -> only local gates
  - some simulator only accept « small » gates
  - we need to break down large gates into smaller ones

Long term goals

- Compilation toward various hardwares
  - communicate with experimentalist
  - understand the basic operation that they implement
  - compile efficiently toward them
- Theoretical benchmarks:
  - gate passing time vs. circuit length & decoherence time
  - algorithm vs. architectures?
  - optimize the calls to quantum proc.
- Noise models?

Conclusion

- We need « classical » competences:
  - HPC powered simulation
  - optimization and compilation are classical problems
- But also « quantum » competences:
  - implementing quantum algorithms
  - build high level libraries
  - interacting with labs (experimental & theoretical)

Thanks

For more information please contact:

simon.martiel@atos.net

Bull
atos technologies