Neutral-atom quantum computing
Gates and moves - tutorial

Pedro Lopes

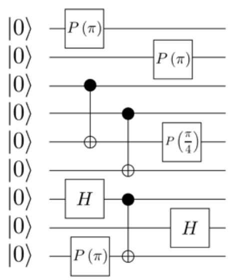
Casey Duckering

Yelissa Lopez

QuEra Computing Inc.

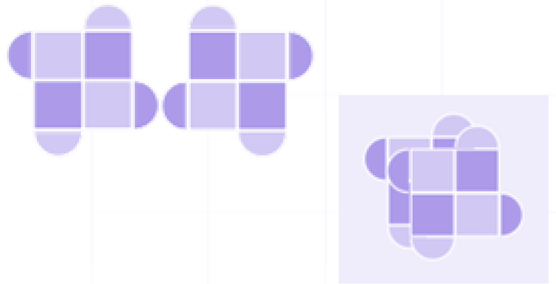
YQuantum 2025

Main theme



Algorithmic pipeline

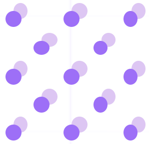
Error-correcting code choice
Compilation
Gate design
Protocol layout + spacetime optimization

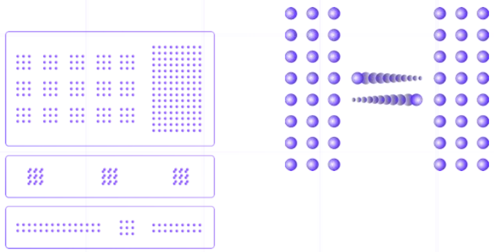


Co-Design

Native hardware capabilities

Speed
Qubit connectivity
Parallelization
Universal gate-set
Biased noise & erasure



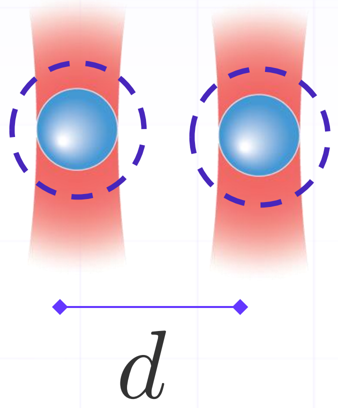


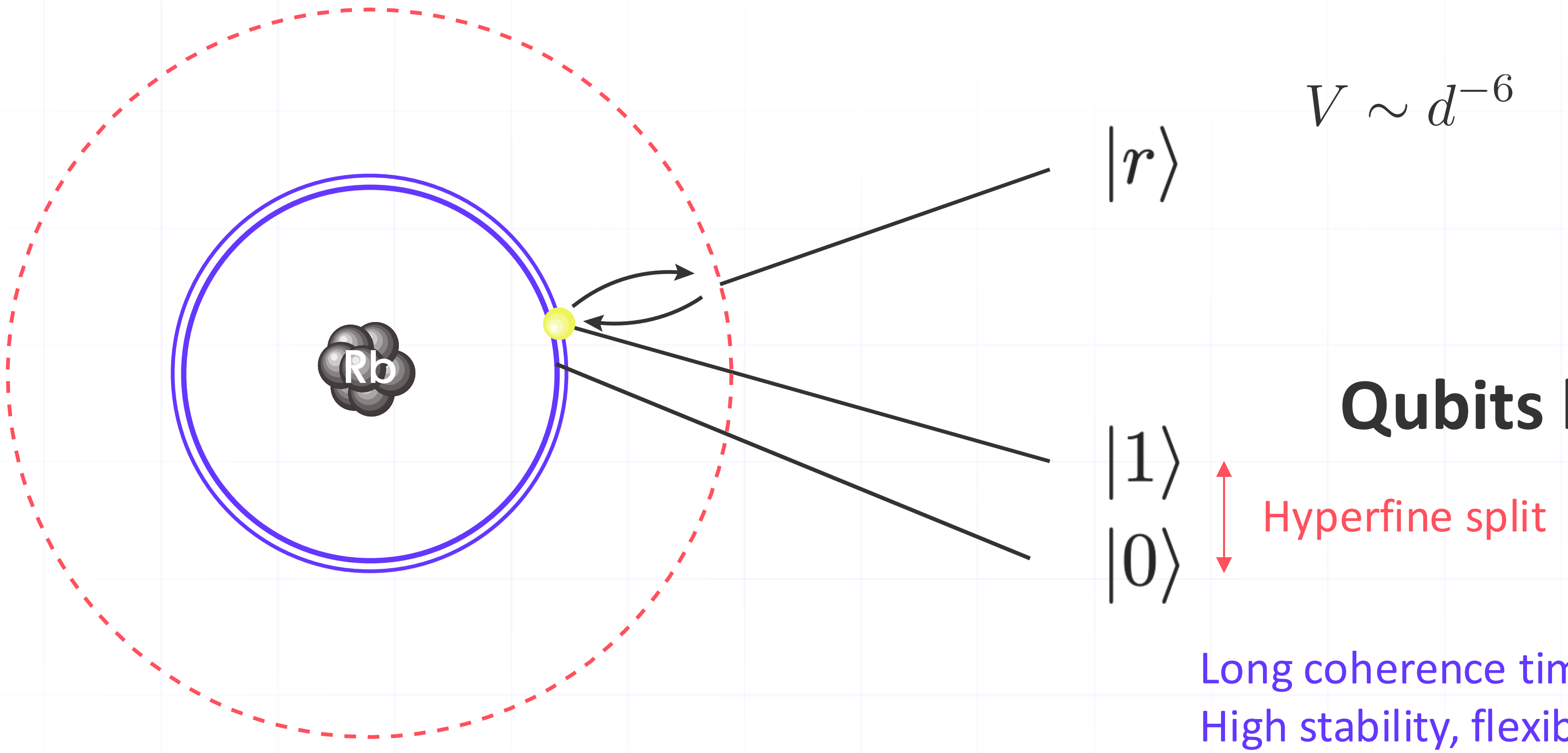
How can we leverage neutral atoms’ strengths to design efficient algorithms?

Digital: Entanglement mediated by puffing-up atoms

Strong interactions
Controlled interactions

$V \sim d^{-6}$

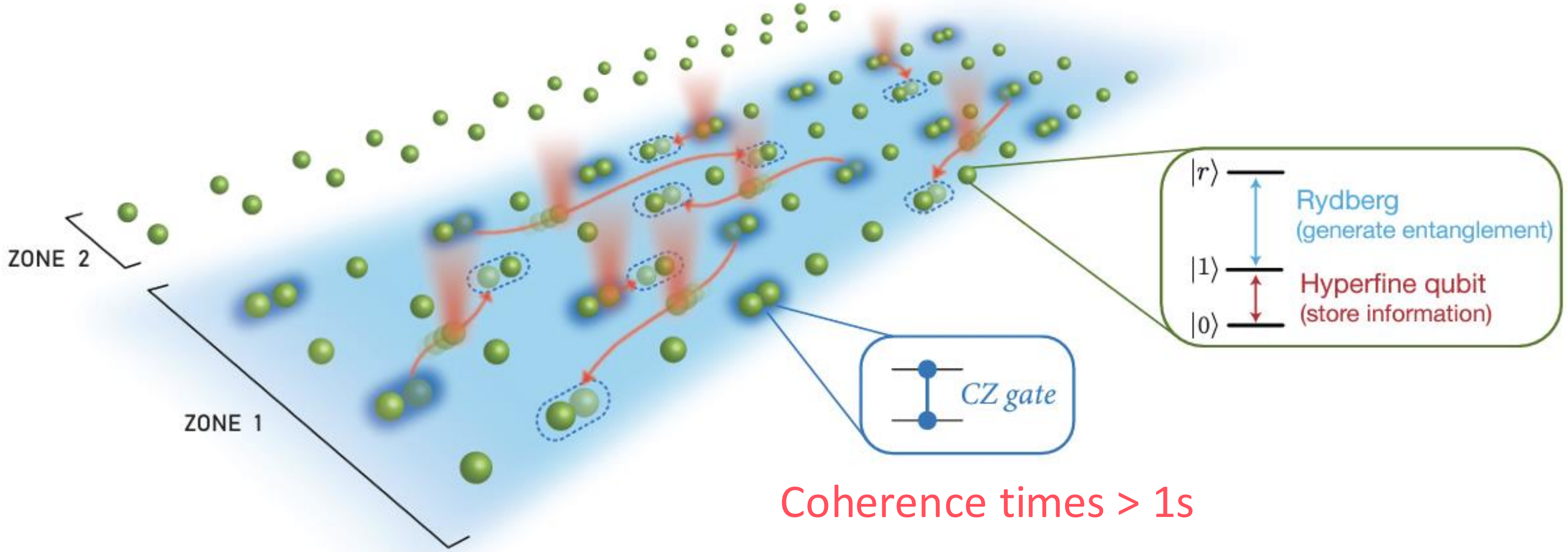




Qubits by isolated levels choice

Long coherence times
High stability, flexibility

Basic architecture: mid-circuit reconfigurability



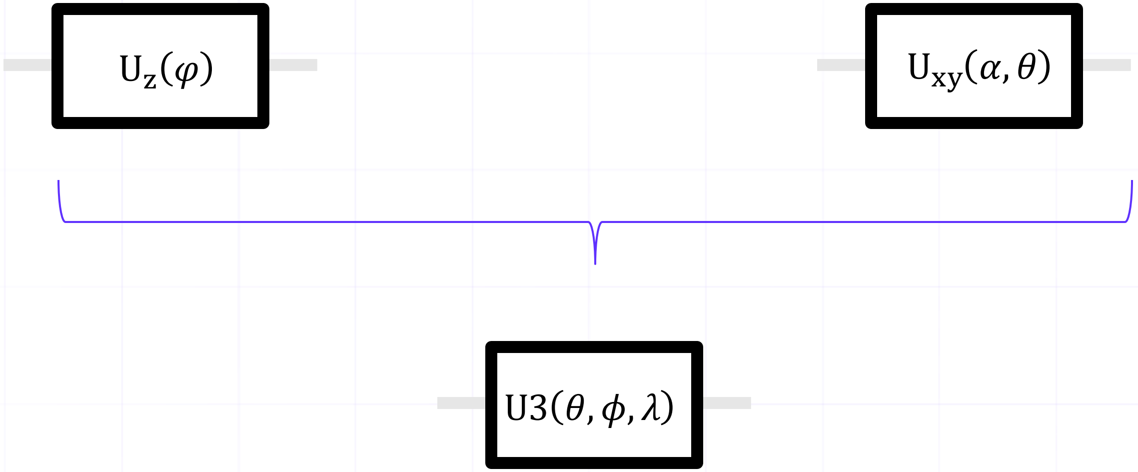
Coherence times > 1s

Bluvstein et al., Nature (2022)

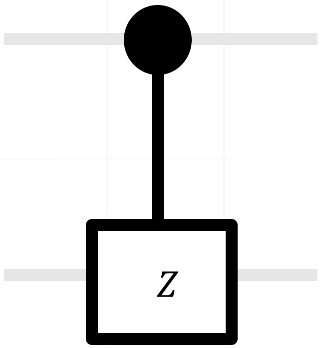
Native gate set (for our purposes)

Arbitrary 1 qubit Z rotations

Arbitrary 1 qubit rotations
With axis in the XY plane

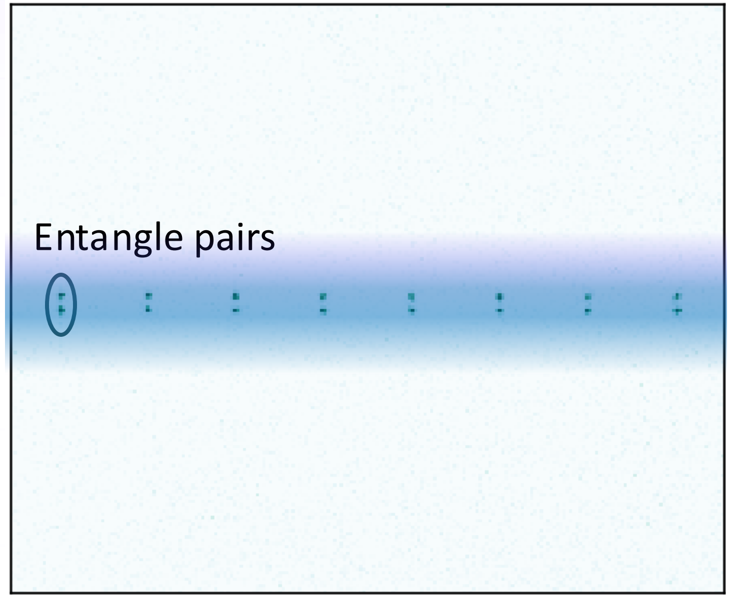


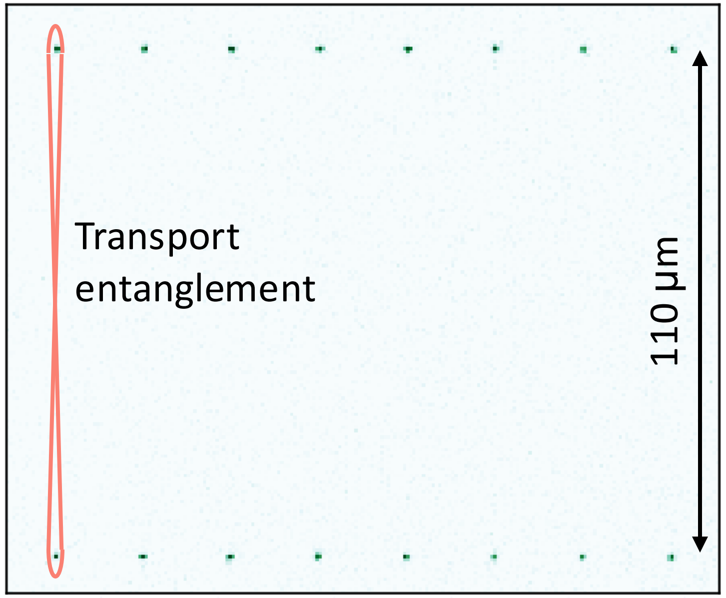
Controlled-Z gates

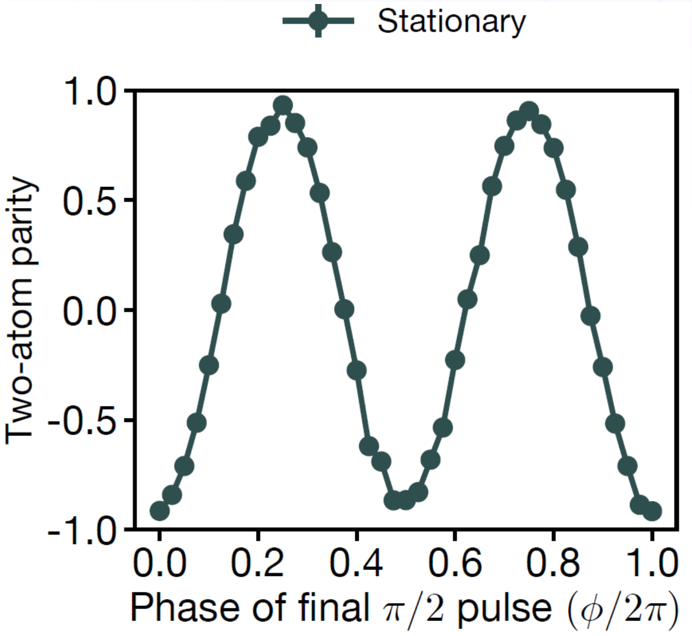


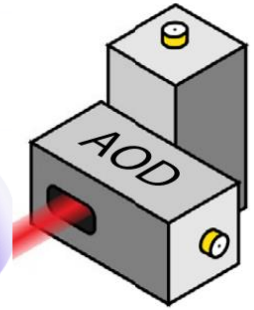
Entanglement transport

<300 μs to move across entire array (T_2 ~ 1.5 s)









Atom-atom spacing of ~3 μm

→ transport across array of ~2000 qubits in a time of < 10^{-3} T_2

Bluvstein et al., Nature 2022

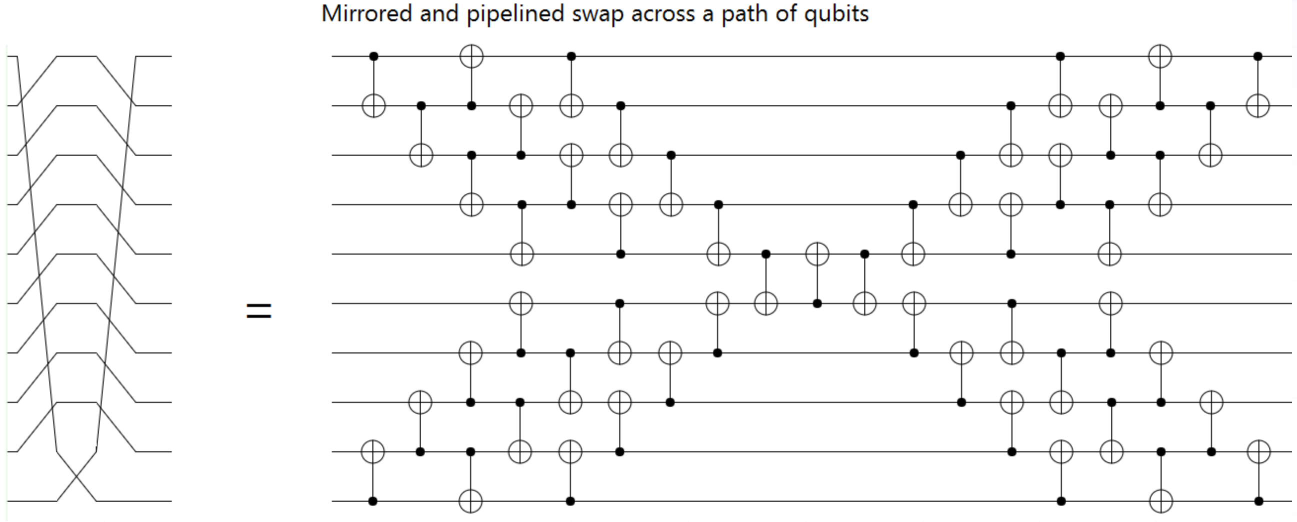
Beugnon et al *Nat Phys* 2007; T. Đorđević et al *Science* 2021

Atom shuttling rules!

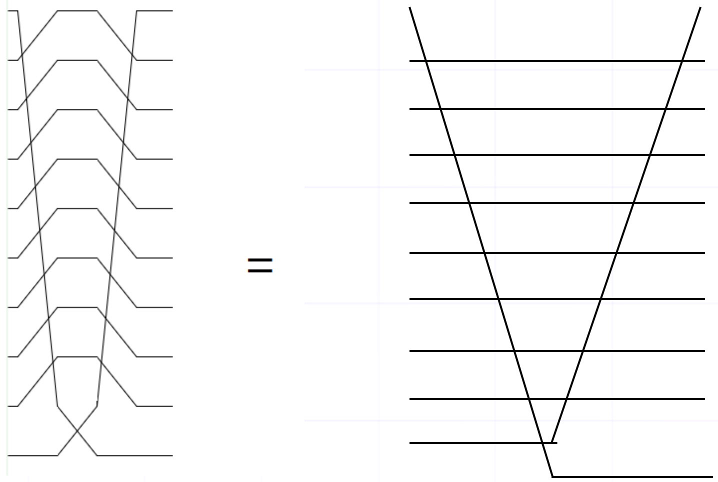
Long-range/arbitrary connectivity

Source: Craig Gidney's blog

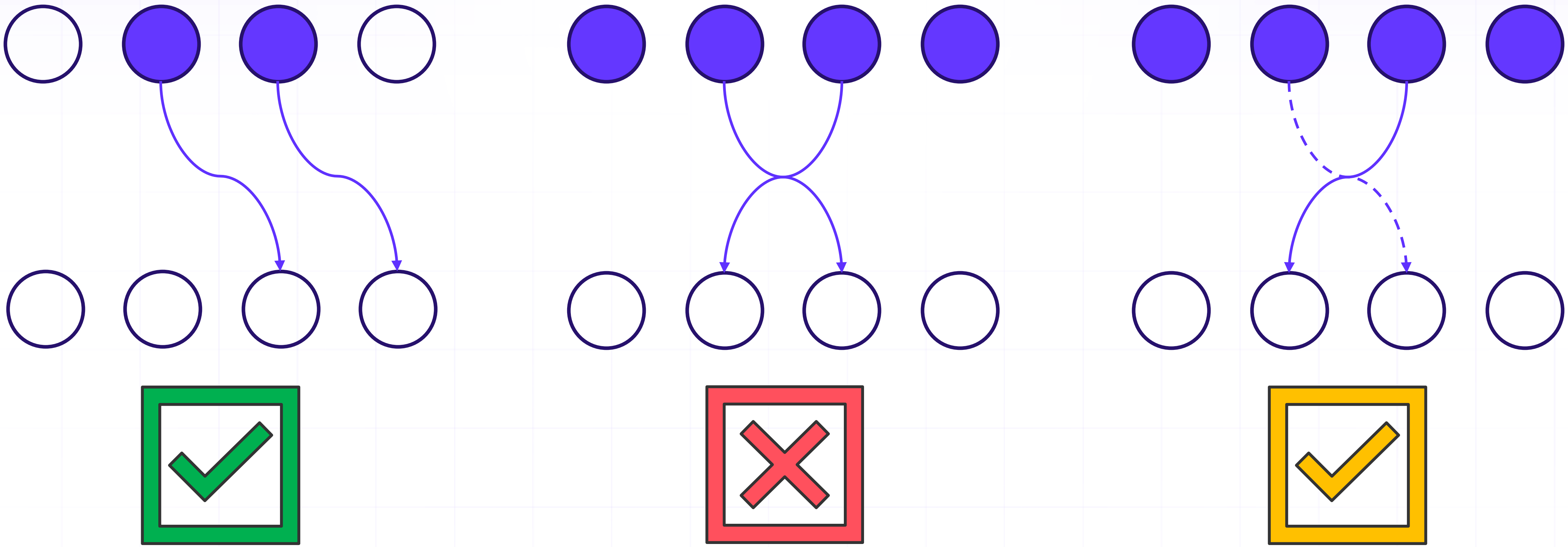
Nearest-neighbor connectivity



Reconfigurable connectivity

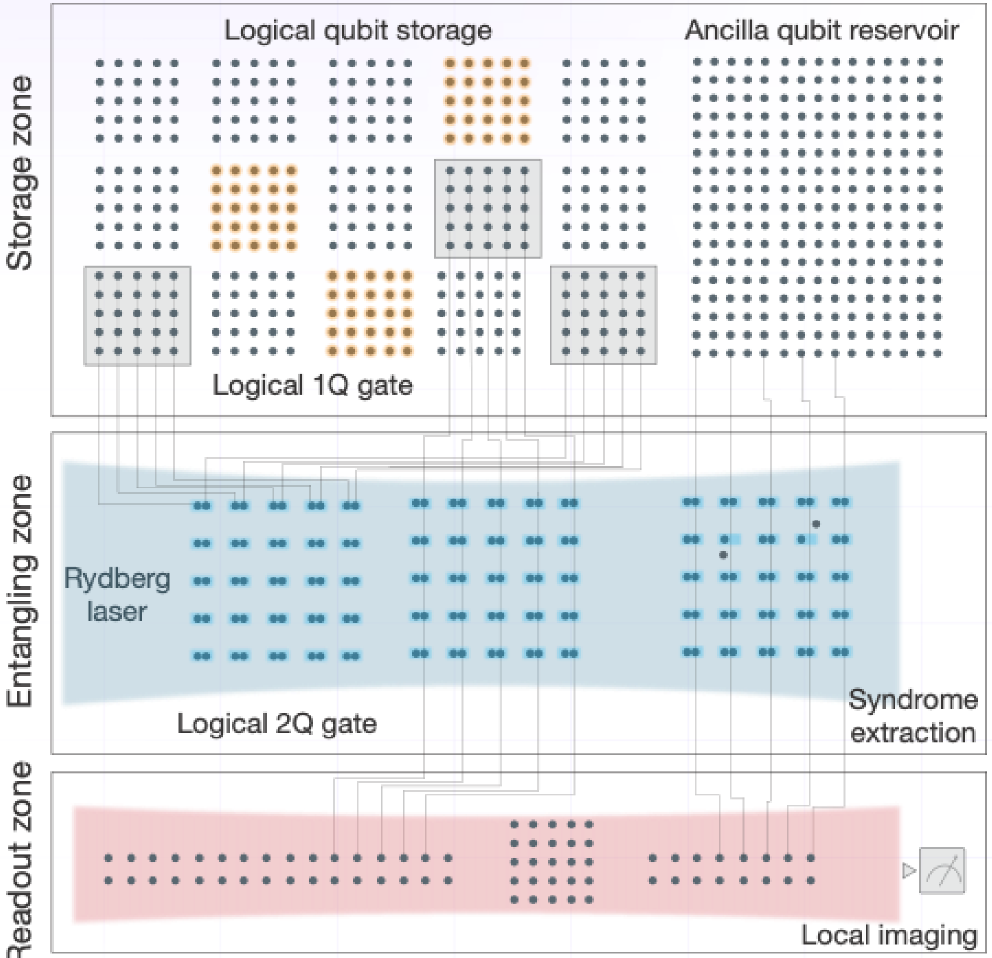


Atom shuttling rules?



“atoms cannot collide”
“atoms cannot change order in a single move”

Sandbox Model for Current Gen. Quantum Computer



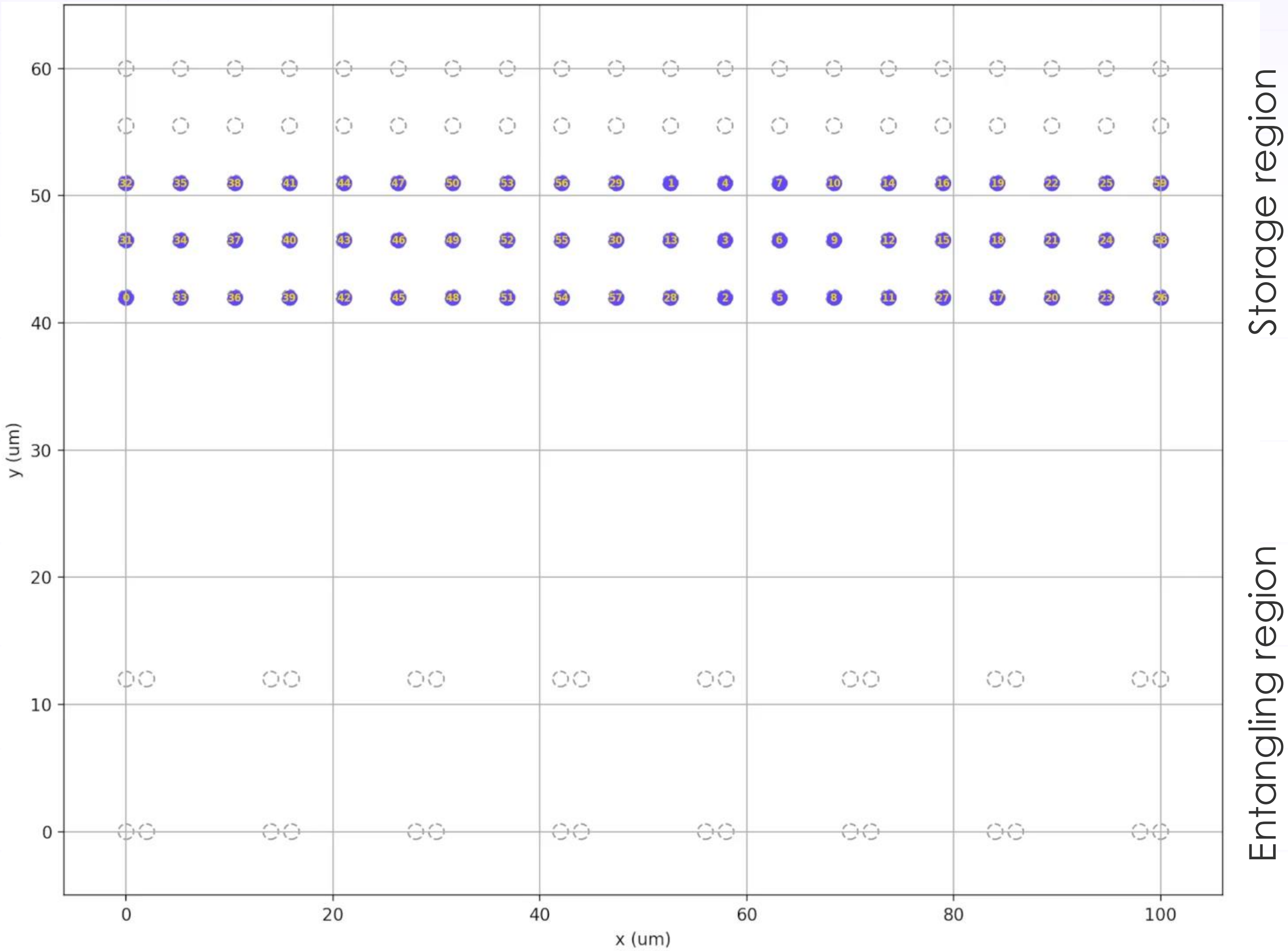
Bluvstein et al., Nature 2024

Keep in mind: the technology is still rapidly developing, and **tomorrow's systems** may look **very different**!

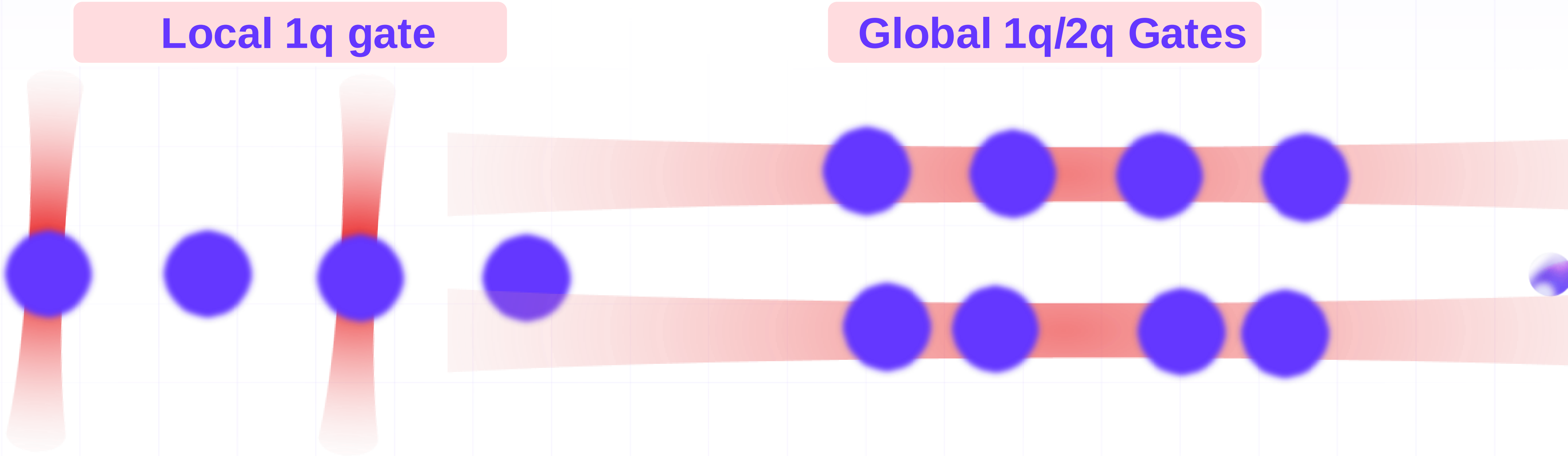
- Hundreds to a thousand qubits
- High-fidelity parallel gate operation, with long coherence times
- Parallel movement of qubits on a grid
- Mid-circuit measurement and feedforward
- Some analogies to classical RAMs

|QuEra>

Sandbox Model for Current Gen. Quantum Computer

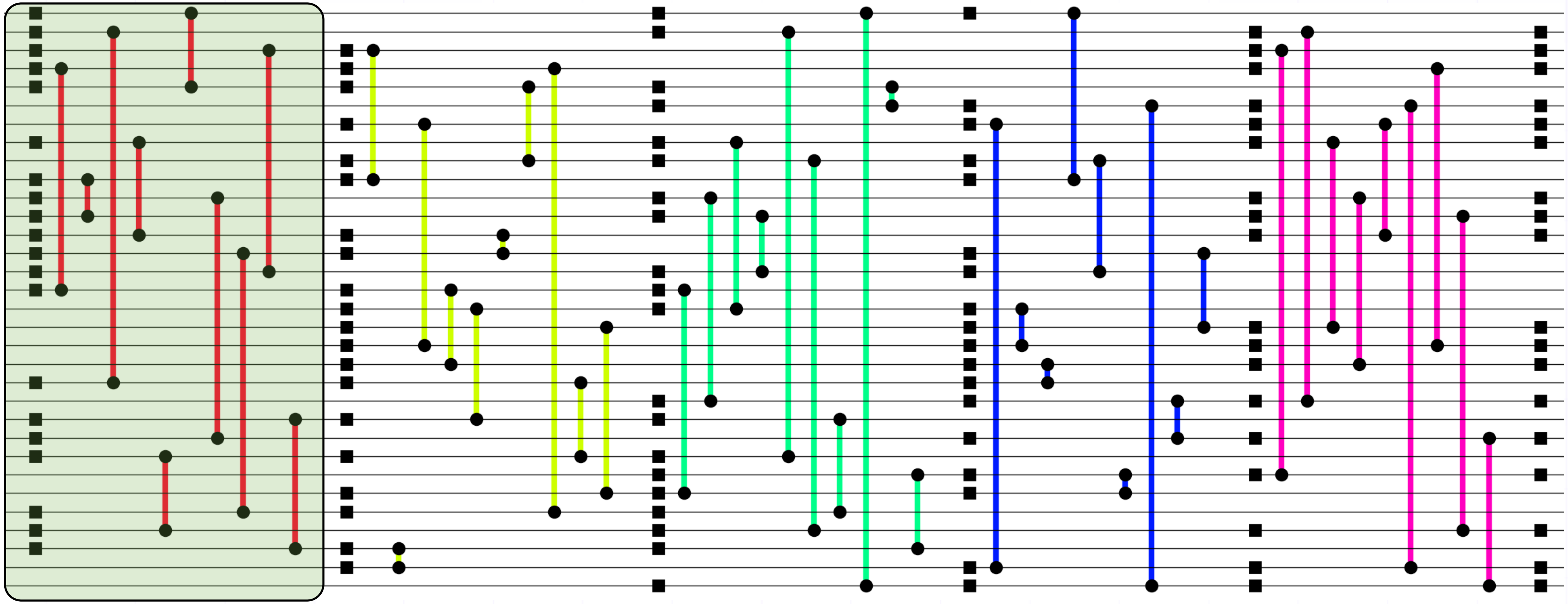


Local gates vs global gates



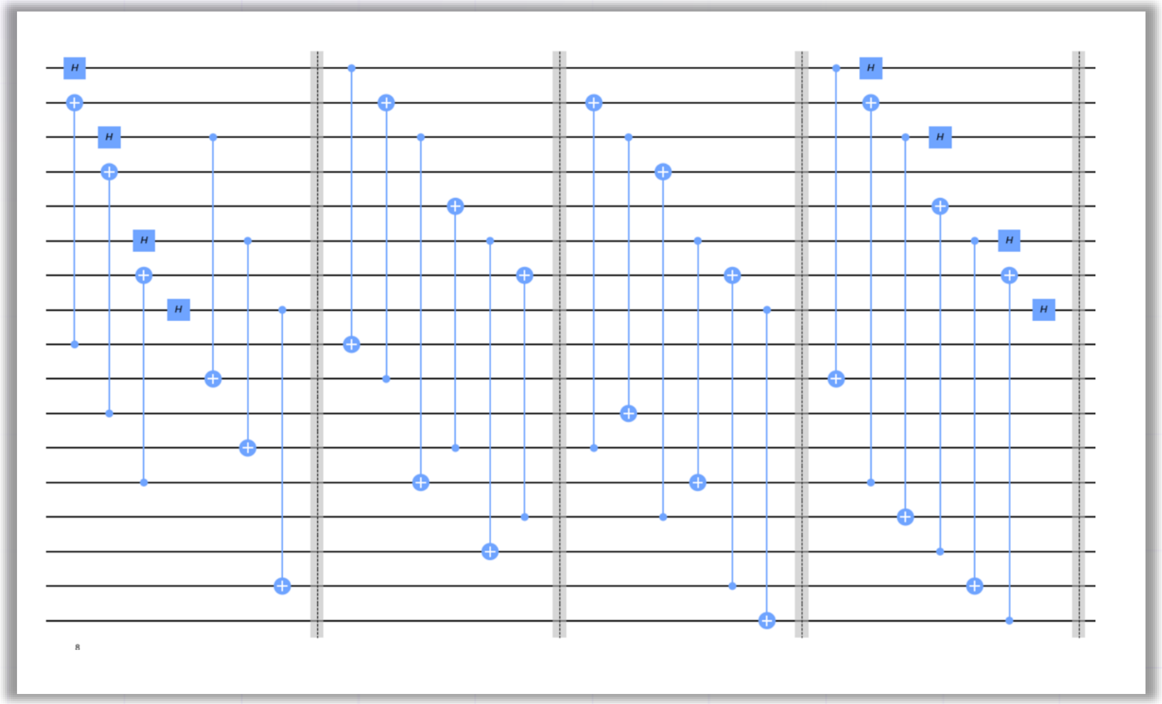
Global gates and native parallelism

Key notion: The same gate is applied on many qubits in parallel

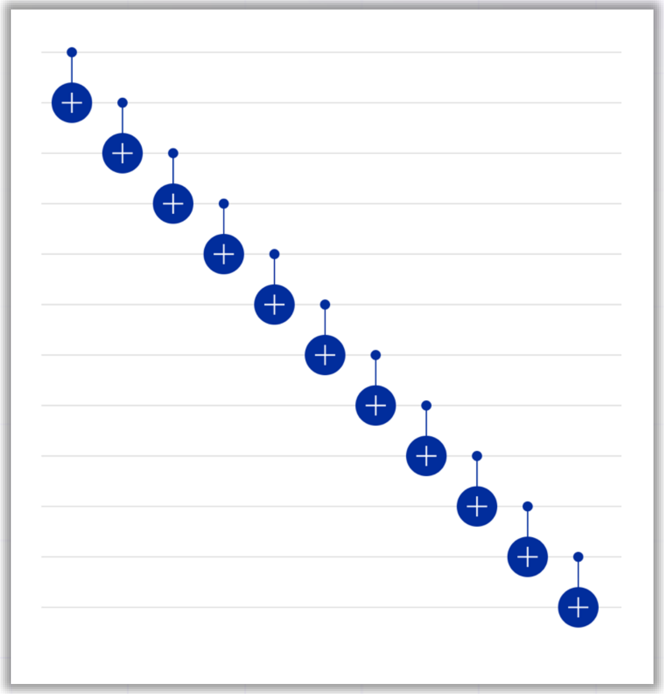


A fundamental block: 1q gates plus a set of cliques representing multi-qubit gates

Parallelism is key



A round of syndrome extraction for the surface code

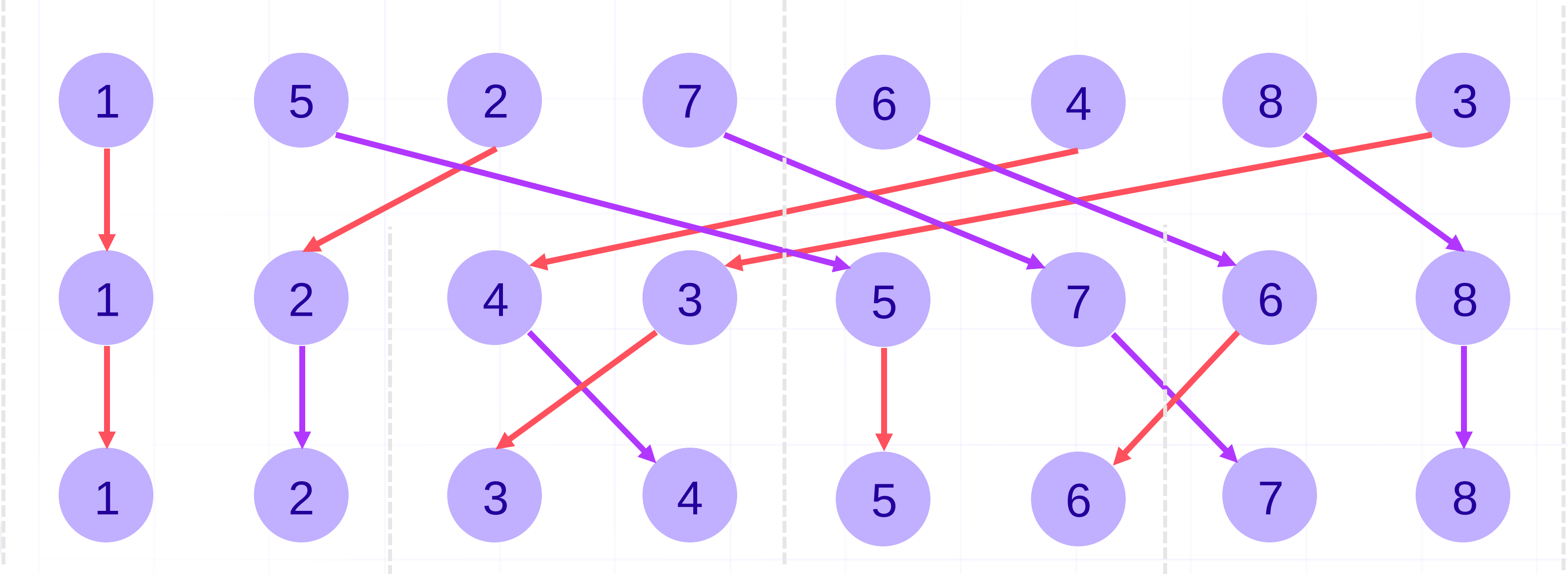


A staircase circuit

A Co-designed Compilation Mindset

~~"All to All"~~ ⇒ **Efficient parallel swap**

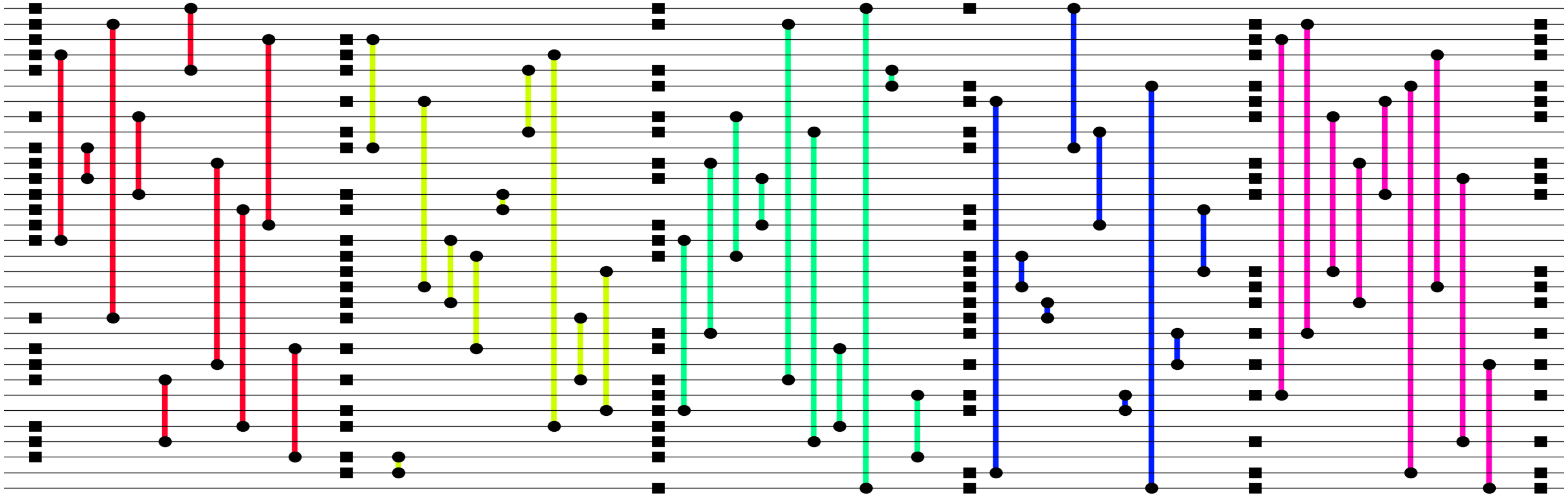
Atoms can **be efficiently sorted in $log(N)$** parallel moves.



A Co-designed Compilation Mindset

"All to All" ⇒ Efficient parallel swap

Sequential gates ⇒ Parallel layers



|QuEra>

Programming neutral-atom quantum computers

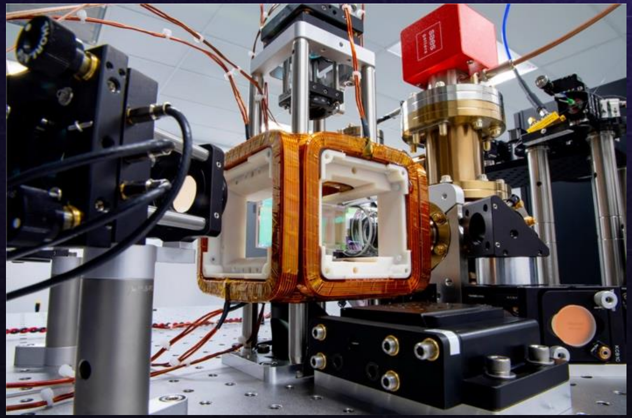
Bloqade

Bloqade is QuEra Computing's software development kit (SDK) for neutral atom quantum computers. It is designed to be a hub of embedded domain-specific languages (eDSLs) for neutral atom quantum computing. Bloqade is built on top of Kirin, the Kernel Intermediate Representation Infrastructure.

Kirin

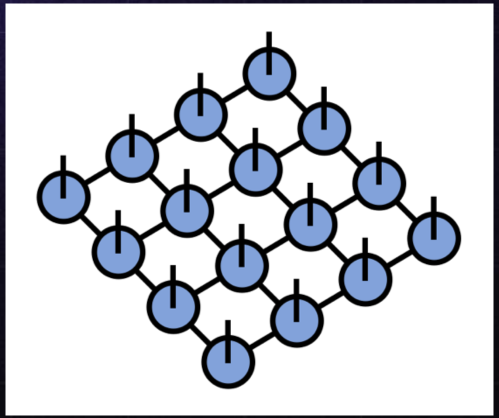
Kirin is the **K**ernel **I**ntermediate **R**epresentation **In**frastructure developed. It is a compiler infrastructure for building compilers for embedded domain-specific languages (eDSLs) that target scientific computing kernels especially for quantum computing use cases where domain-knowledge in quantum computation is critical in the implementation of a compiler.

The growing need of compiler engineering



novel hardware

(the Inside of Aquilla)



complicated multi-purpose simulation software

(A PEPS tensor network from tensornetwork.org)

Scientists start touching compiler engineering not only in the field of quantum computing, e.g., ModelingToolkit, Modelica, numericalEFT

What are scientists looking for?

- something not in C++, ideally in Python
- not aiming for compiling millions of IR nodes
- a low-effort frontend with customizable semantics
- common compiler passes such as constant propagation
- composability for fast prototyping

An example for the "candy" language

Decorator marking
kernel function for compilation

Custom statement with
default python syntax

Regular Python variable
with scoping analysis etc.

Native control flows
but in this case, execute randomly

```python
@candy
def main(x):
    candy = NewCandy("chocolate")
    Eat(candy)
    Unwrap(candy, 12)
    Crash()
    if x > 1:
        Eat(NewCandy("chocolate"))
    else:
        Eat(NewCandy("bubble_gum"))
    return x + 1
```

The next-gen SDK of QuEra – bloqade-circuit

A set of dialects including

- Structural gate dialect adopted from Yao
- A QASM2-like dialect as compilation target

A fully-featured (gate subroutines, opaque commands) QASM2 parser and tooling

- Parser
- Python-based AST objects with standard visitor pattern
- Pretty printing
- Lowering pass to Kirin QASM2 dialect as SSA IR

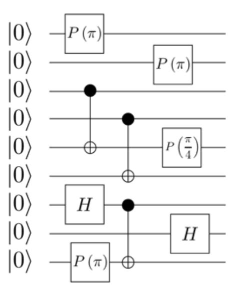
Allow QuEra-backed extension of QASM2

```
lines = textwrap.dedent(
    """
    OPENQASM 2.0;

    qreg q[2];
    creg c[2];

    h q[0];
    CX q[0], q[1];
    barrier q;
    CX q[0], q[1];
    rx(pi/2) q[0];
    """
)

@qasm
def qasm2_inline_code():
    core.InlineQASM(lines)
    qreg = core.QRegNew(4)
    RX(qreg[0], 2.2)
```

bloqade-circuit
(Kirin-based circuit SDK)

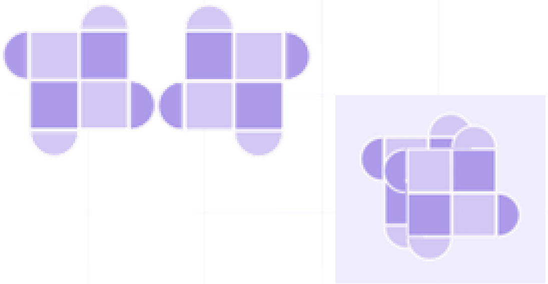
|QuEra>

Final words



Algorithmic pipeline

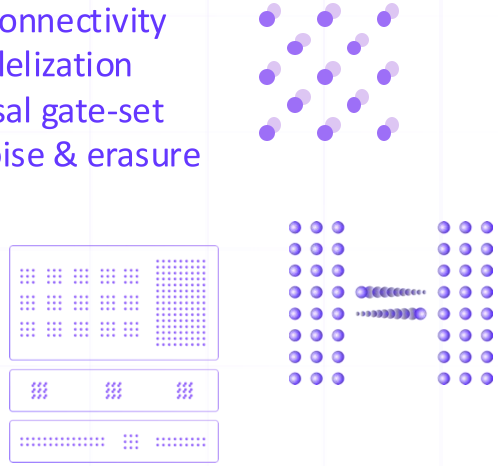
Error-correcting code choice
Compilation
Gate design
Protocol layout + spacetime optimization



Co-Design

Native hardware capabilities

Speed
Qubit connectivity
Parallelization
Universal gate-set
Biased noise & erasure



How can we leverage neutral atoms’ strengths to design efficient algorithms?