



Programming neutral-atom devices with Bloqade

Arinjoy De

Jonathan Wurtz

Kai Wu

Pedro Lopes

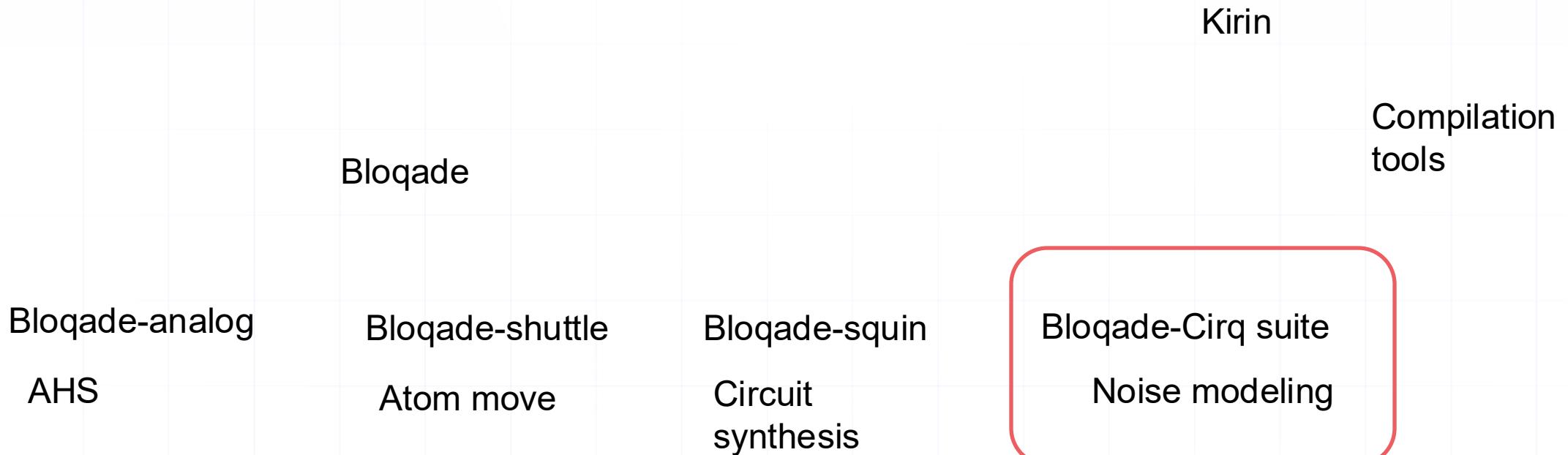
Tyler Cochran

QuEra Computing Inc.



Programming tools hierarchy

Circuit level, QEC/physical level, hardware level, [signals level]



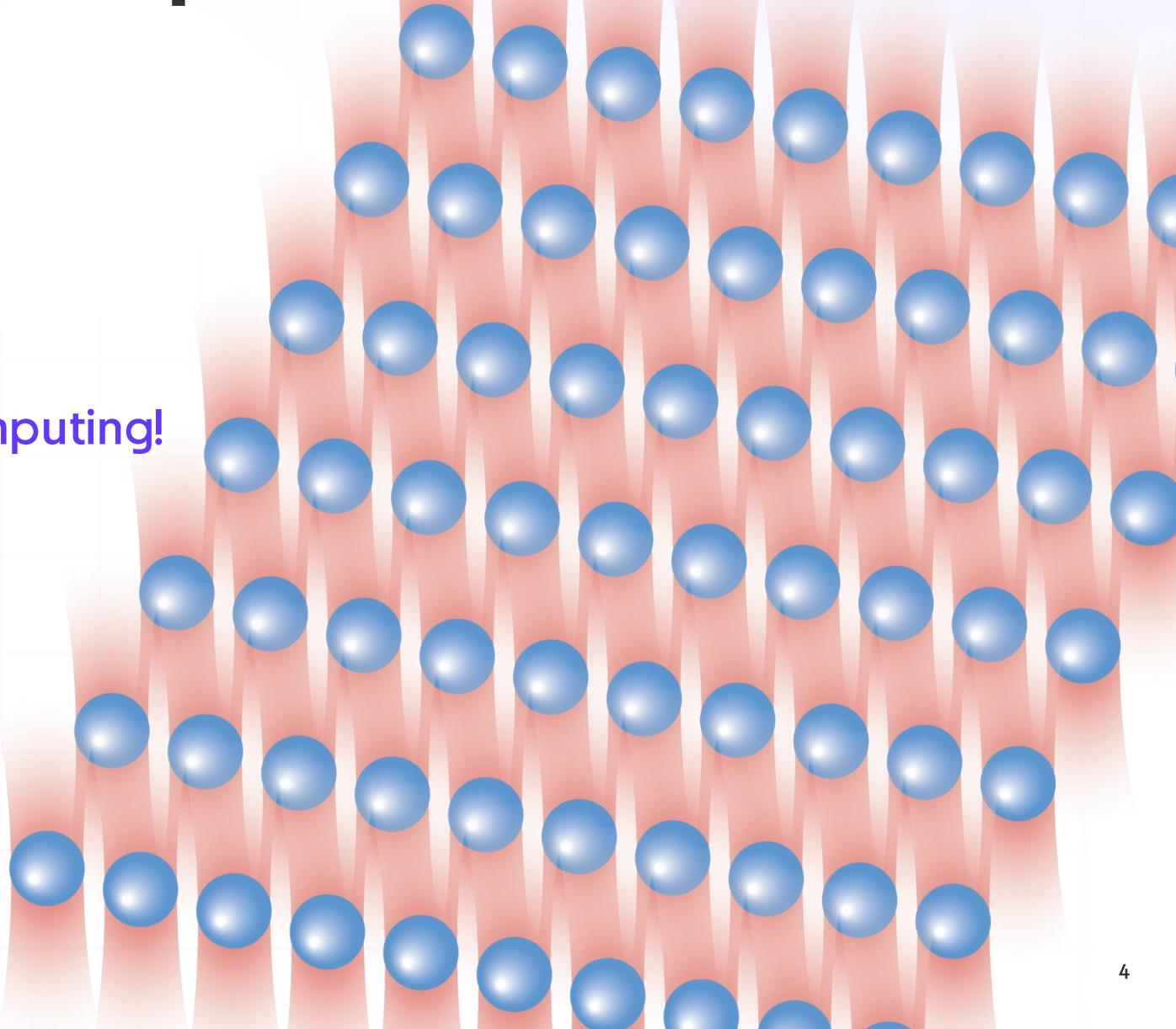
Learning objectives

By the end of the session, you will be able to:

- **Describe** the baseline **functionality** of a digital neutral-atom quantum computer architecture based on zoned architectures
- **State constraints for atom shuttling operations**, register sizes, and geometrical and temporal scales
- **Write down** a phenomenological **hierarchy of the errors** for different operations during a circuit execution
- **Exemplify** design rules for hardware-aware implementation of quantum circuits
- **Validate** strategies of hardware-aware **heuristic noisy** circuit simulation

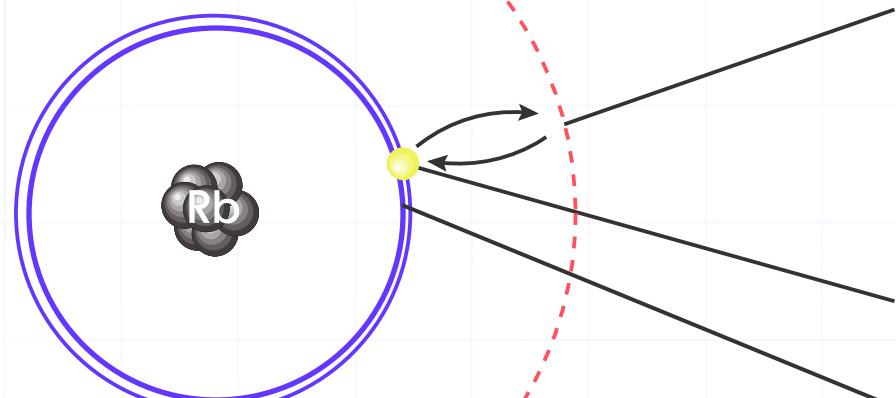
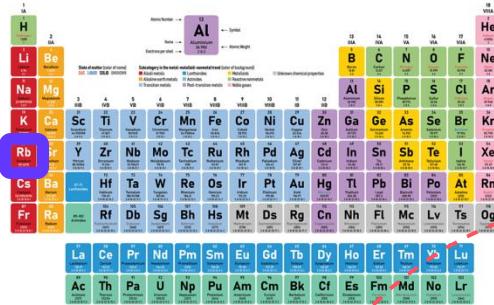
Neutral-atom quantum processor

- Densely packed qubits (atoms)
- Efficient qubit control
- Flexible problem encoding
- New ways to think quantum computing!



Entanglement mediated by puffing atoms up

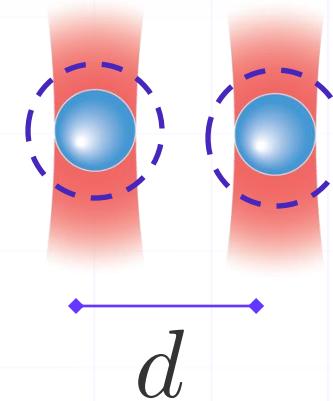
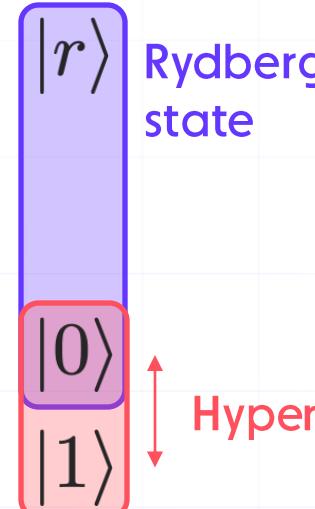
Periodic Table of the Elements



Entanglement via high-energy levels

Strong interactions
Controlled interactions

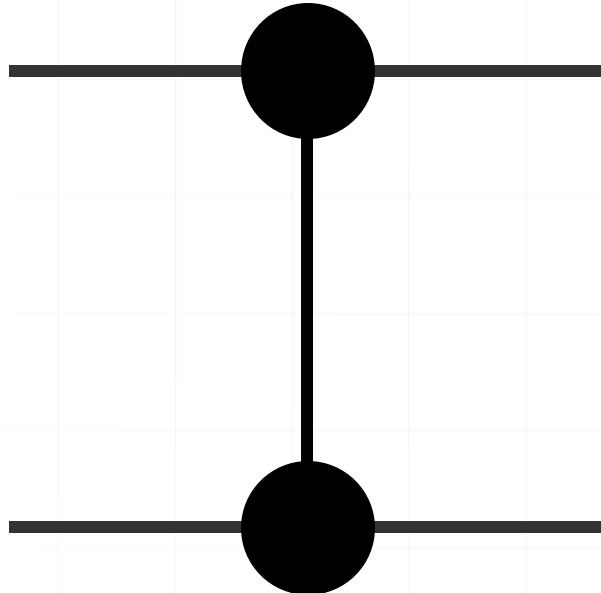
$$V \sim d^{-6}$$



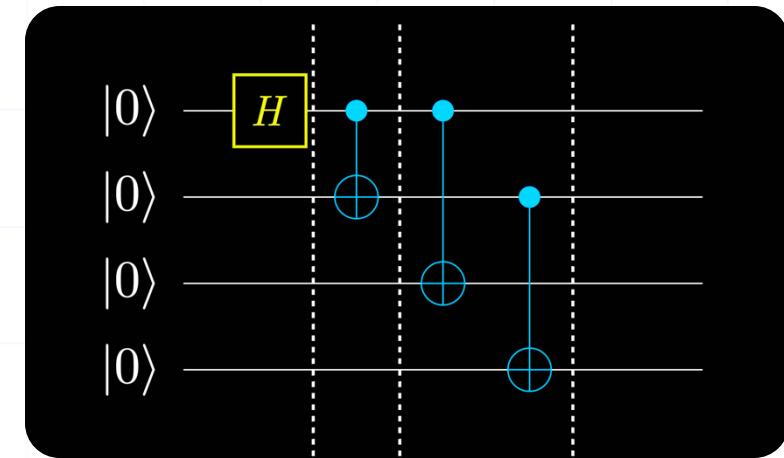
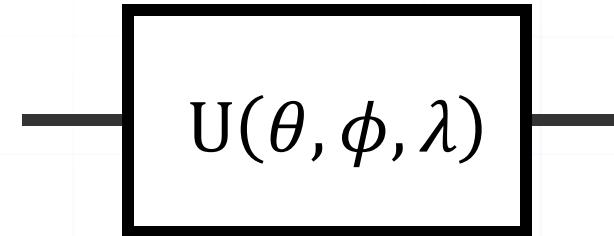
Quantum information in low-energy levels

Programming constraints – native gate set

Controlled Z gates



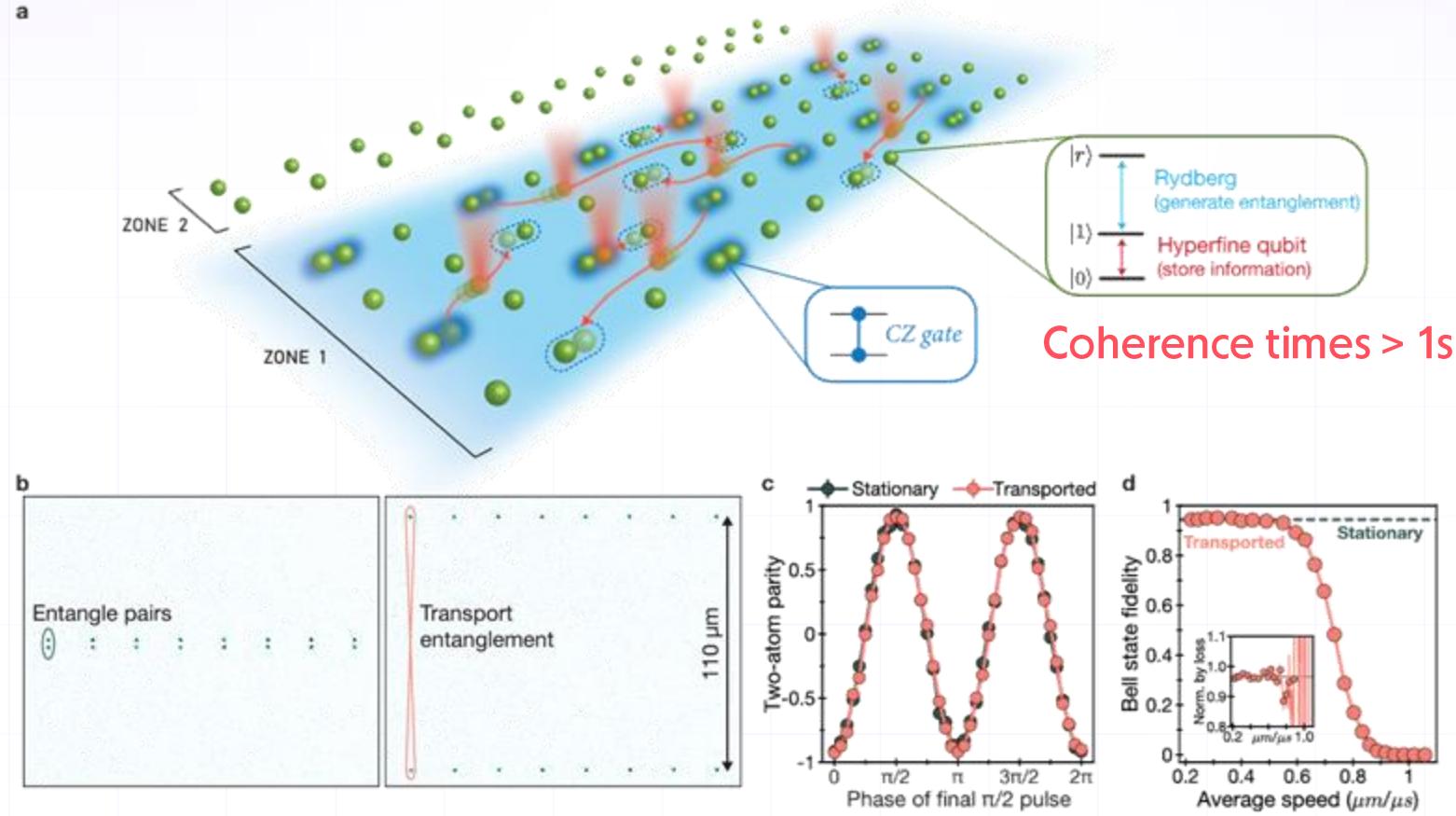
Arbitrary 1-qubit rotations
(favoring Z separate from XY-plane
rotations)



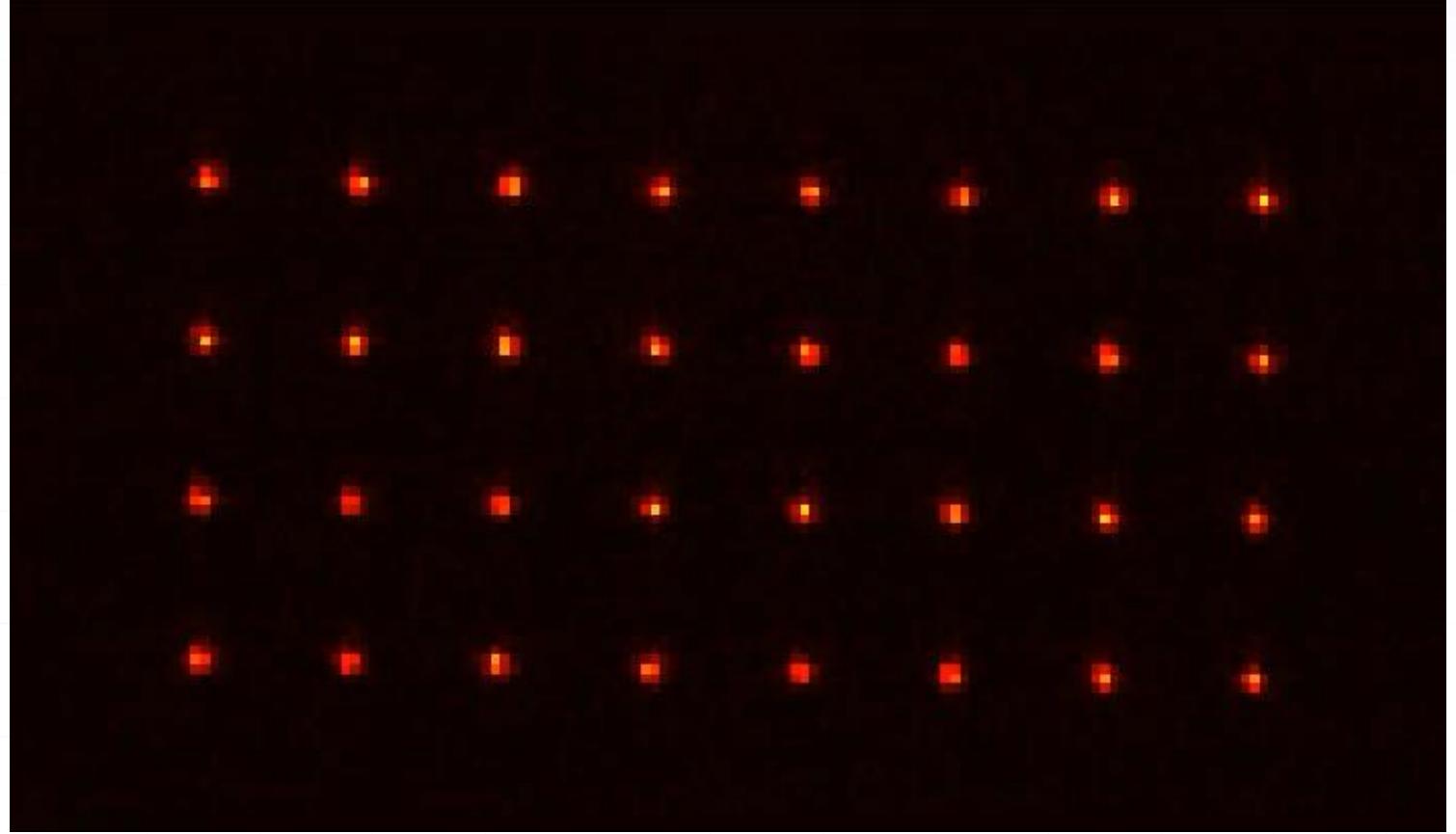
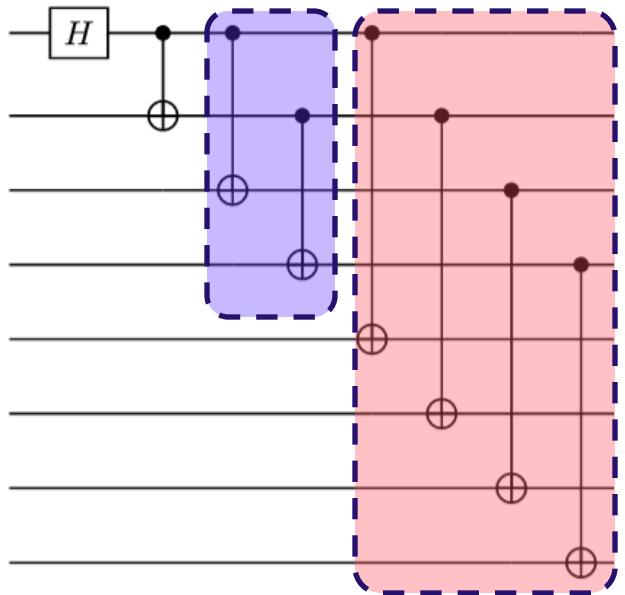
| Gemini-class: digital-mode computing



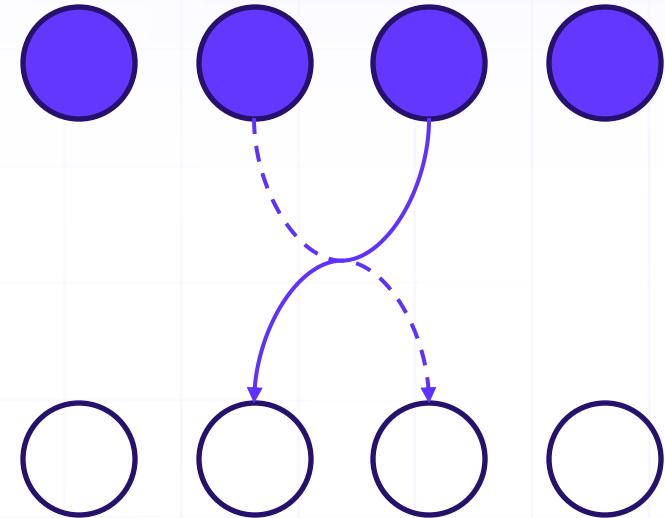
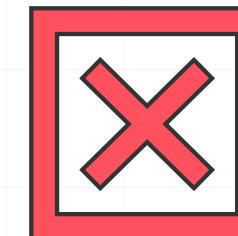
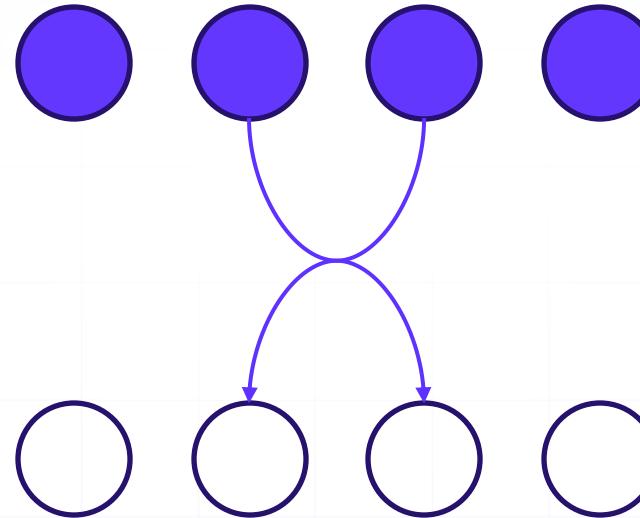
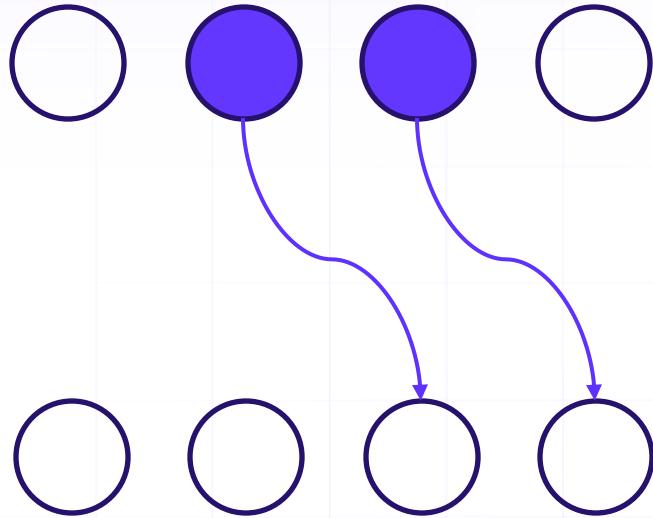
Basic architecture: mid-circuit reconfigurability



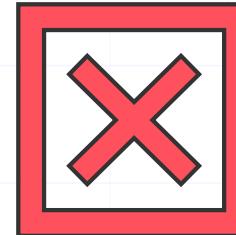
Multi-qubit gates through reconfiguration



Atom shuttling rules #1 – crossing conflict



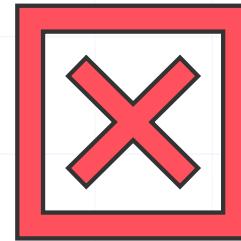
Atom shuttling rules #1 – crossing conflict



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

Activity: why these rules?

Atom shuttling rules #2 – “many-to-one” conflict (bonus)



Preview: Programming moves with Bloqade

Executing the **QCrack** algorithm arxiv:2507.10699

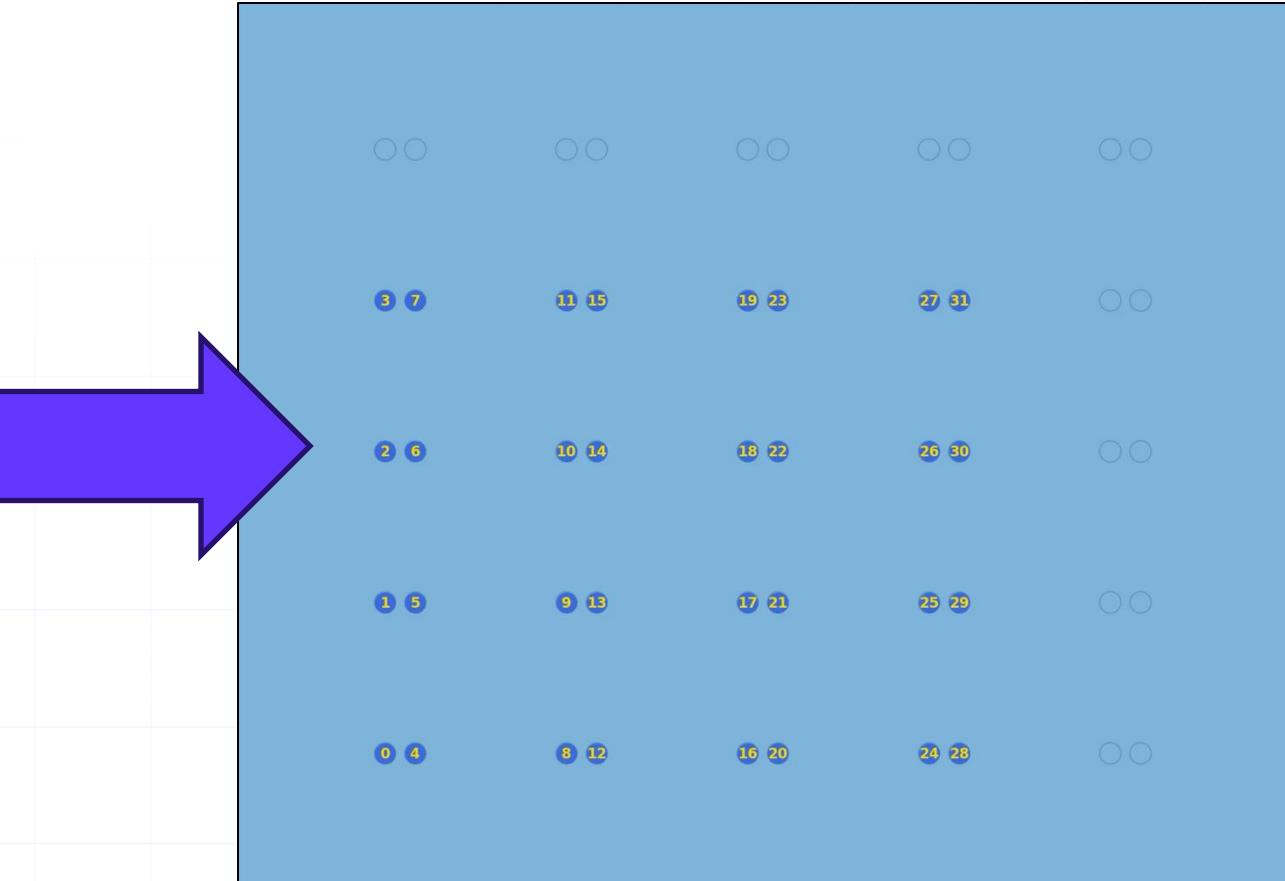
```
@move
def main():
    traps = spec.get_static_trap(zone_id="traps")
    sub_view = grid.sub_grid(traps, [0, 1, 2, 3, 4, 5, 6, 7], [0, 1, 2, 3])
    init.fill([sub_view])

    mover_ids = ilist.IList([0, 2, 4, 6])
    stationary_ids = ilist.IList([1, 3, 5, 7])
    rows = ilist.IList([0, 1, 2, 3])

    def single_qubit_gates():
        gate.global_r(0.1, 1.0)
        gate.local_rz(0.5, grid.sub_grid(traps, stationary_ids, rows))
        gate.local_rz(-0.5, grid.sub_grid(traps, stationary_ids, rows))
        gate.global_r(0.1, -1.0)

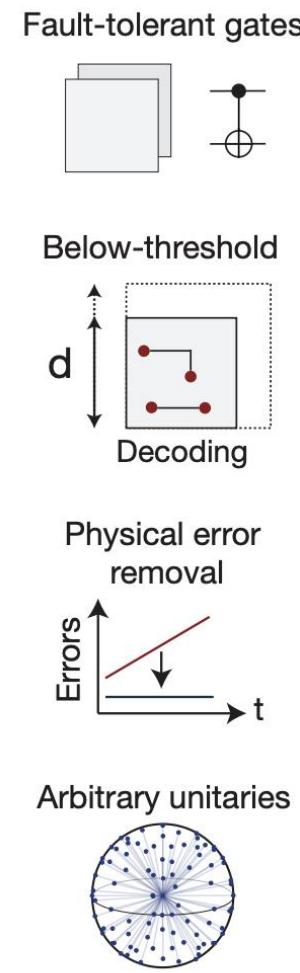
    single_qubit_gates()
    for _ in range(len(mover_ids)):
        for _ in range(len(rows)):
            gate.top_hat_cz(traps)
            row_aux = ilist.IList([rows[-1] + 1])
            rearrange(mover_ids, rows[0:1], mover_ids, row_aux)
            rearrange(mover_ids, rows[1:] + row_aux, mover_ids, rows)

            col_aux = ilist.IList([mover_ids[-1] + 2])
            rearrange(mover_ids[0:1], rows, col_aux, rows)
            rearrange(mover_ids[1:] + col_aux, rows, mover_ids, rows)
            single_qubit_gates()
```

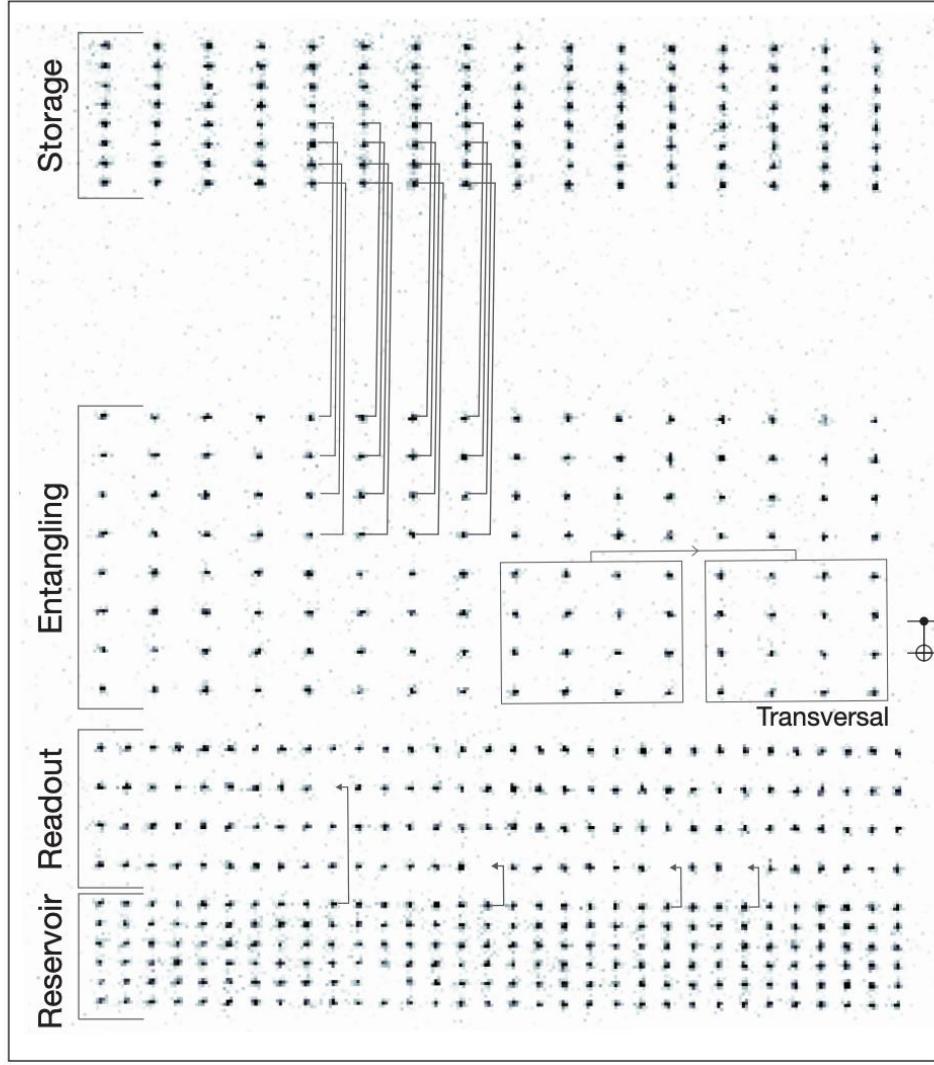


Zoned architectures for fault tolerant computing

a Fault-tolerant processing

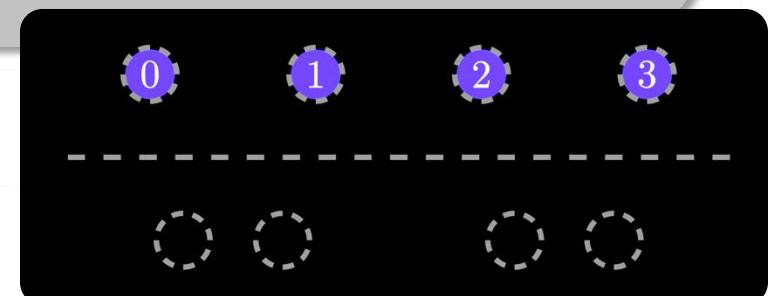
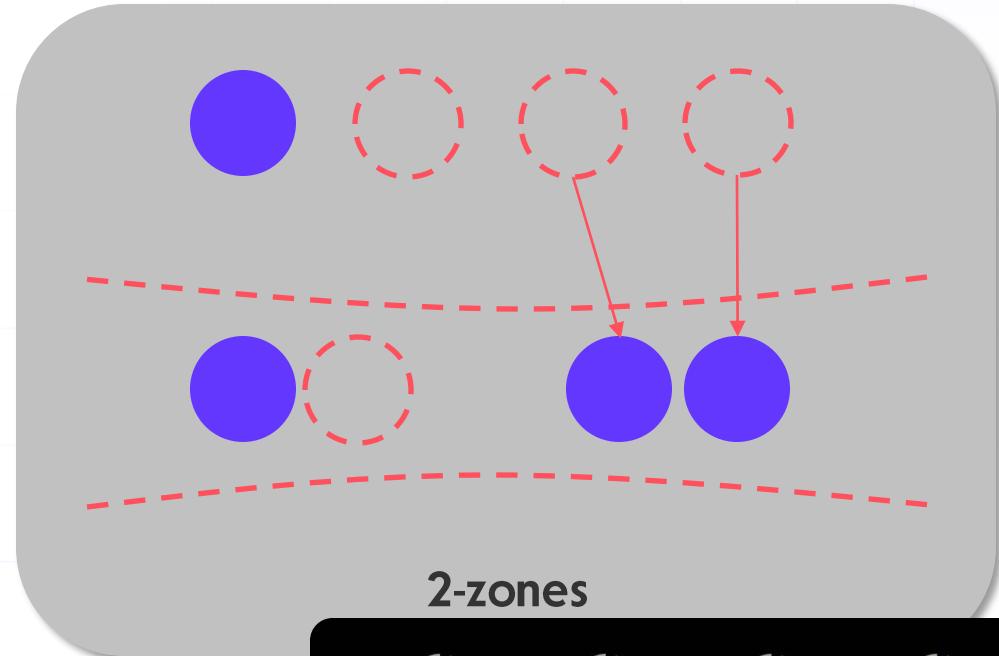
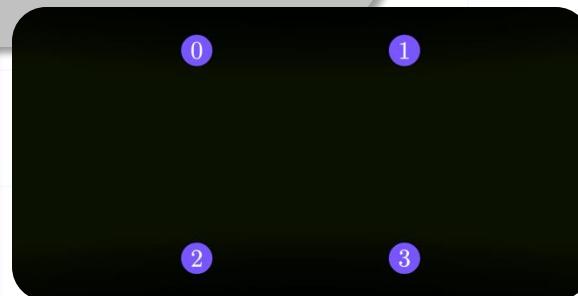
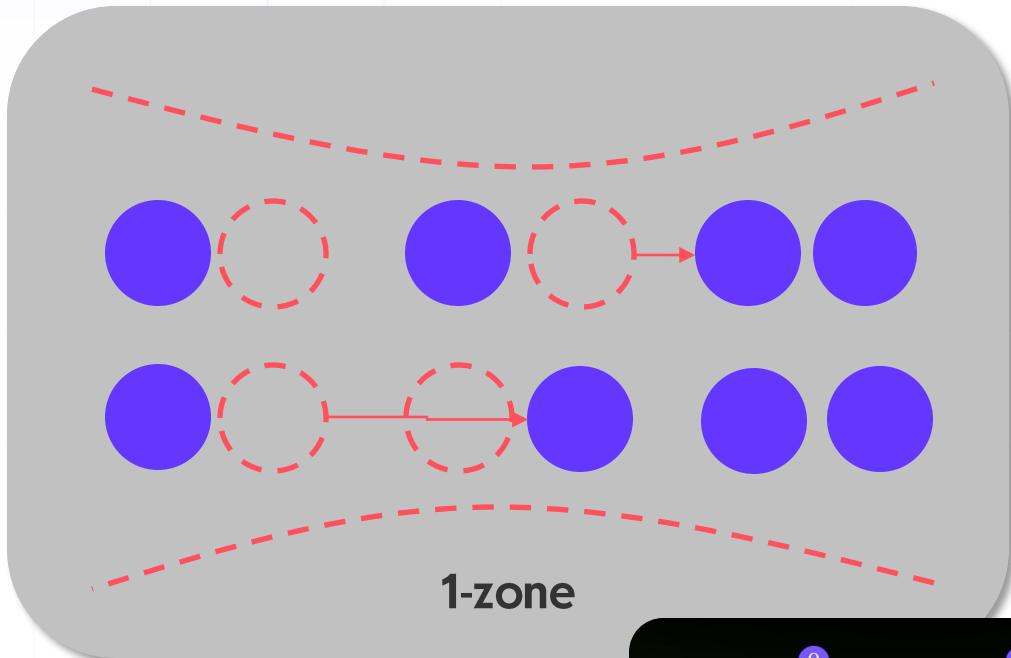
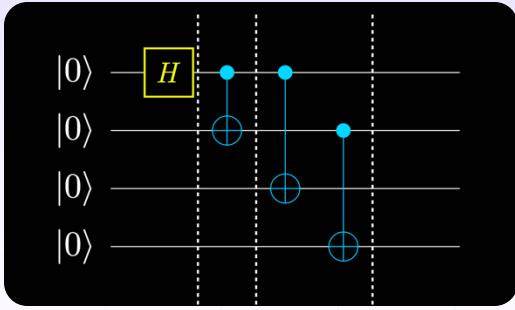


Architecture



Zoned architectures

Qubit count ~ 256



Programming pipeline

Squin: Bloqade's circuit composition dialect

```
from bloqade import squin

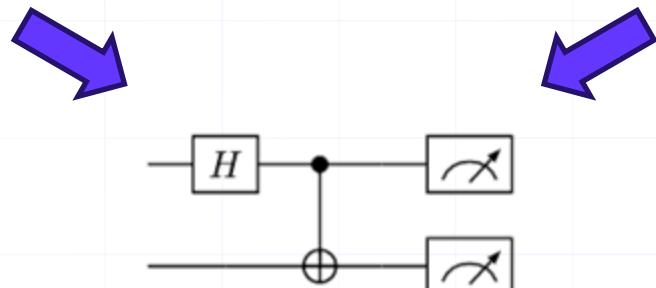
@squin.kernel
def main():
    q = squin.qubit.new(2)
    H = squin.op.h()
    CX = squin.op.cx()
    squin.qubit.apply(H, q[0])
    squin.qubit.apply(CX, q)
    squin.qubit.measure(q)
```

Full interoperability with Cirq

```
import cirq

qubits = cirq.LineQubit.range(2)
circuit = cirq.Circuit(
    cirq.H(qubits[0]),
    cirq.CX(qubits[0], qubits[1]),
    cirq.measure(qubits)
)

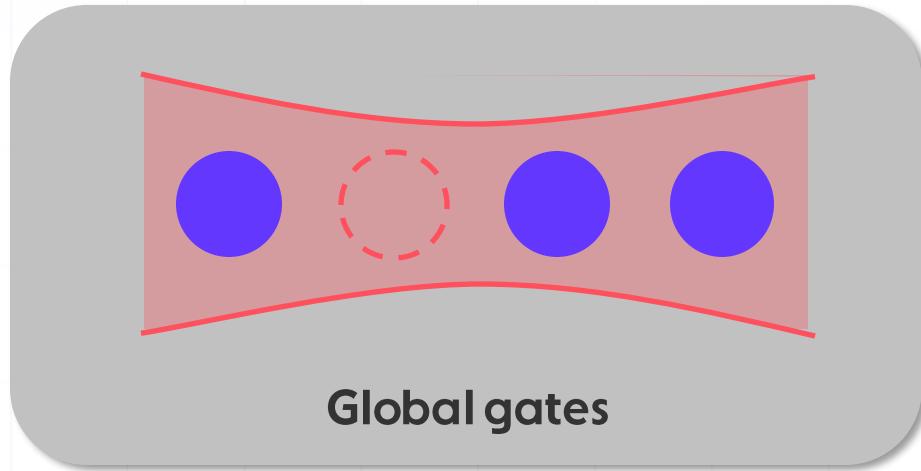
main_loaded = squin.cirq.load_circuit(circuit, kernel_name="main_loaded")
```



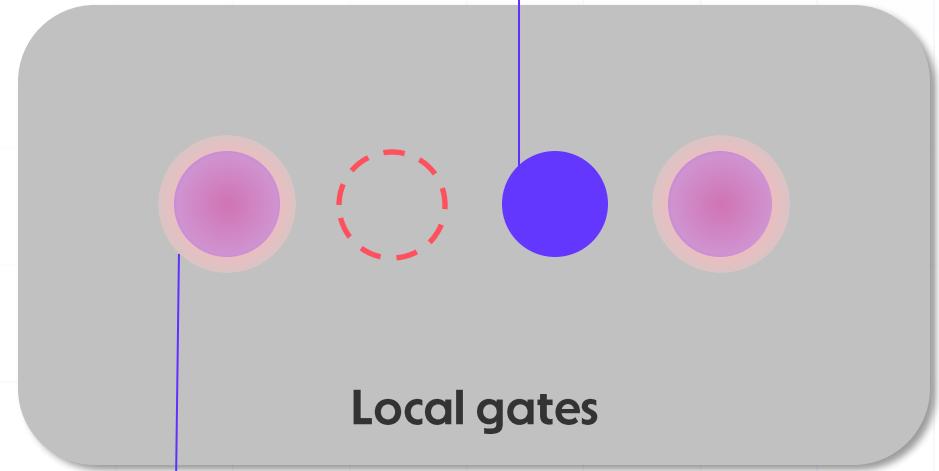
Atom moving composition (bloqade-shuttle)
Simulation (pyqrack)

Noisy circuit analysis (more in what follows!)

Error channels – single qubit operations



Error budget $\sim 10^{-5}$

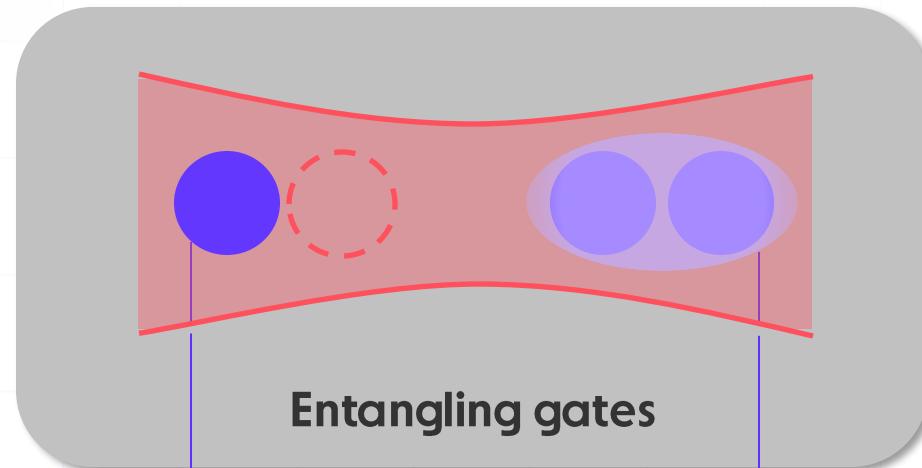


Addressed atom

Error budget $\sim 10^{-4}$

Bluvstein et al., Nature '23,
'24,'25
Evered et al., Nature '23
Rodriguez et al. Nature '25

Error channels – two-qubit operations



Unpaired/spectator atom

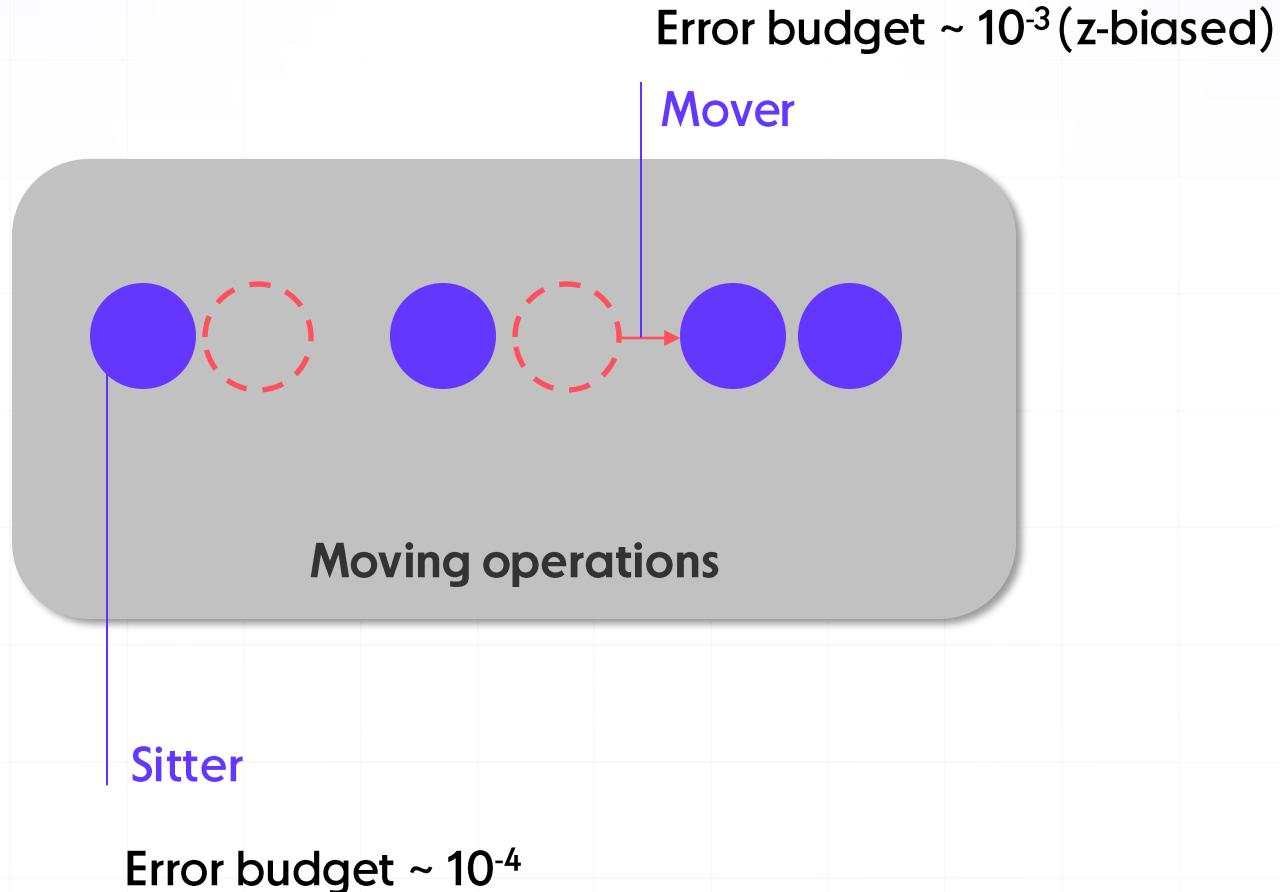
Error budget $\sim 10^{-3}$ (z-biased)

Paired/entangled atoms

Error budget $\sim 10^{-3}$ (z-biased)

Bluvstein et al., Nature '23,
'24, '25
Evered et al., Nature '23
Rodriguez et al. Nature '25

Error channels – shuttling



Bluvstein et al., Nature '23,
'24, '25
Evered et al., Nature '23
Rodriguez et al. Nature '25

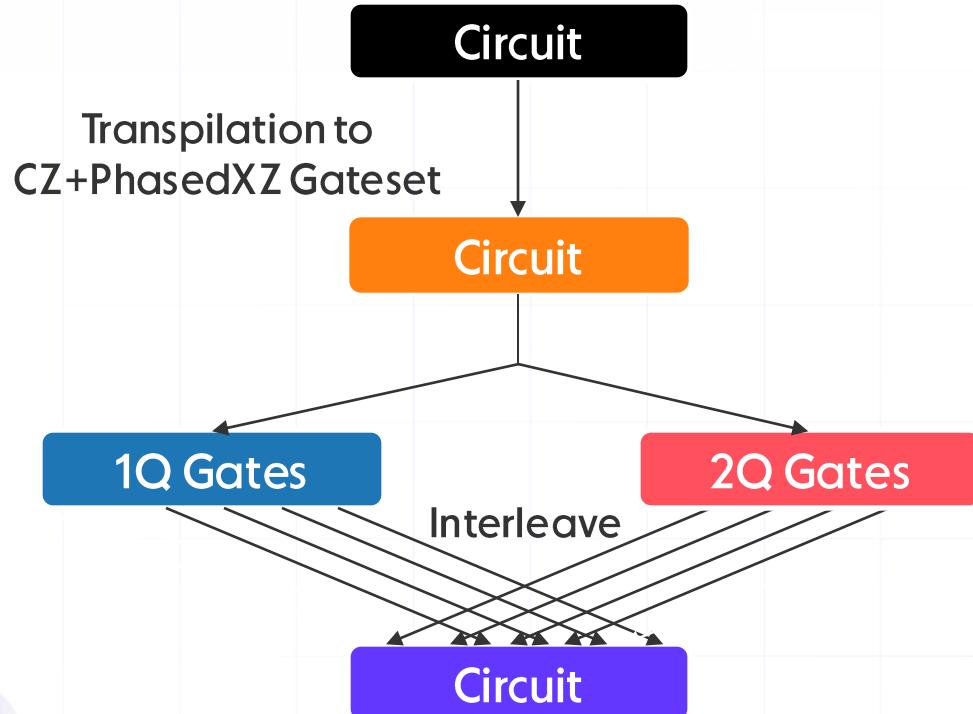
Noise hierarchy

$$E_{CZ} \gtrsim E_{mover} \sim E_{unpaired} > E_{sitter} \sim E_{1-qubit,local} > E_{1-qubit,global}$$

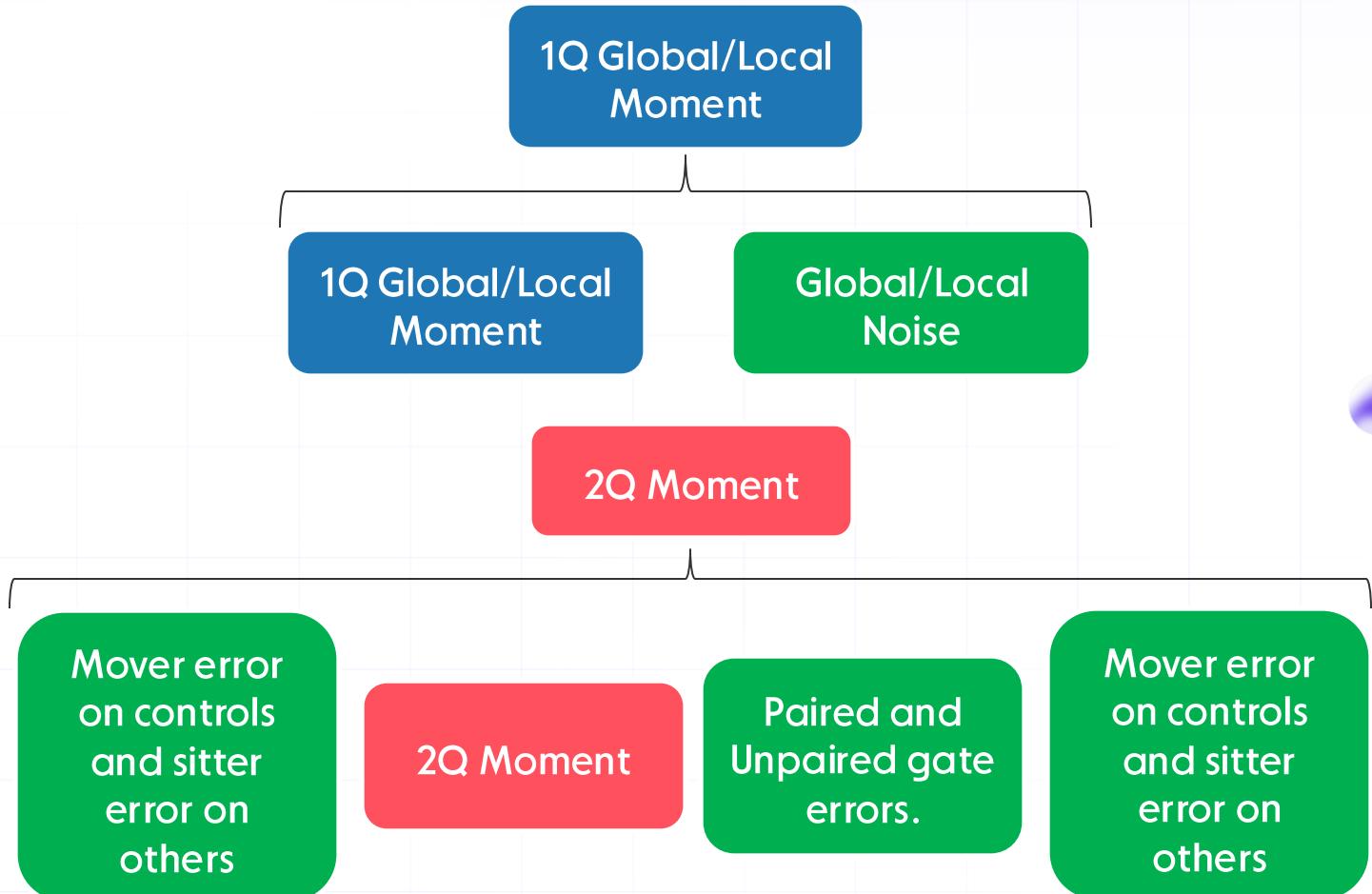
Z-biased

Heuristic noise model - one-zone logic

Step 1
`cirq_utils.transform_to_noisy_one_zone_circuit`

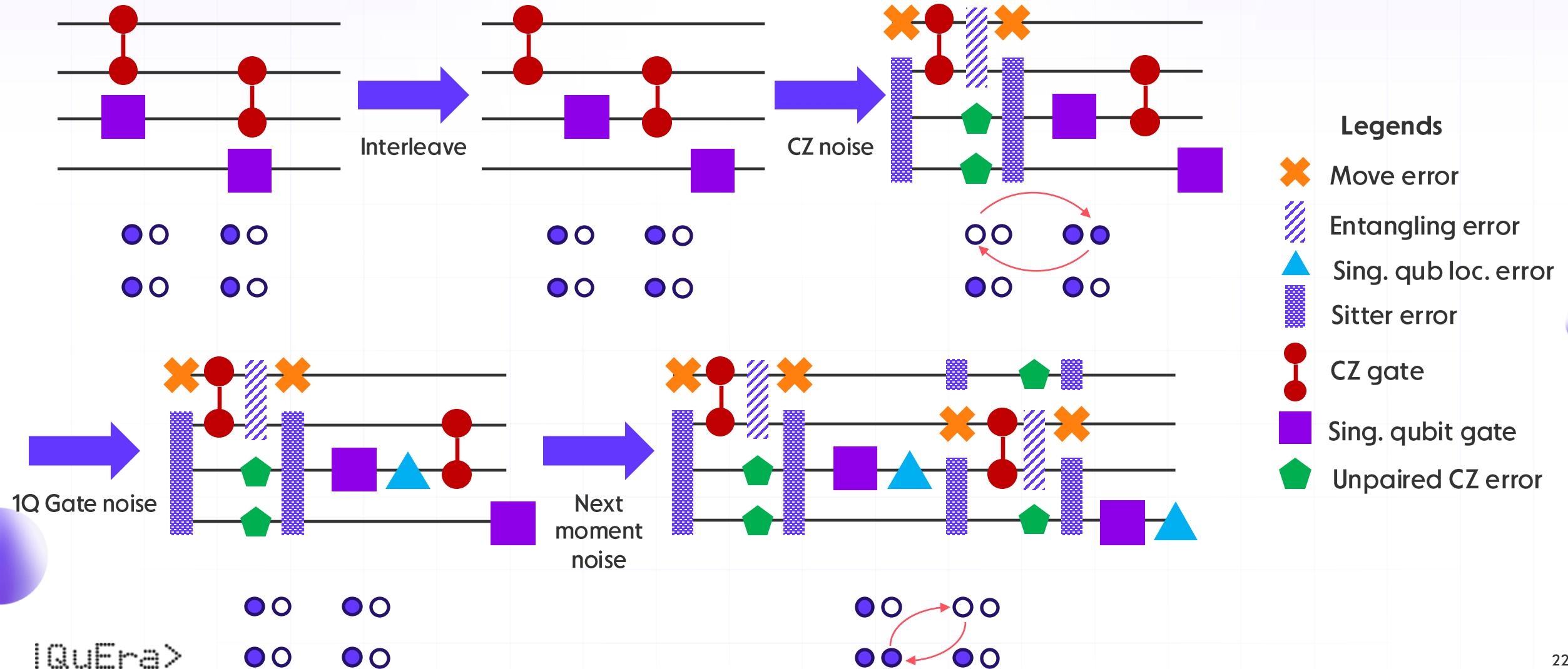


Step 2
`circuit.with_noise(GeminiOneZoneNoiseModel())`



Heuristic noise model - one-zone logic

Example

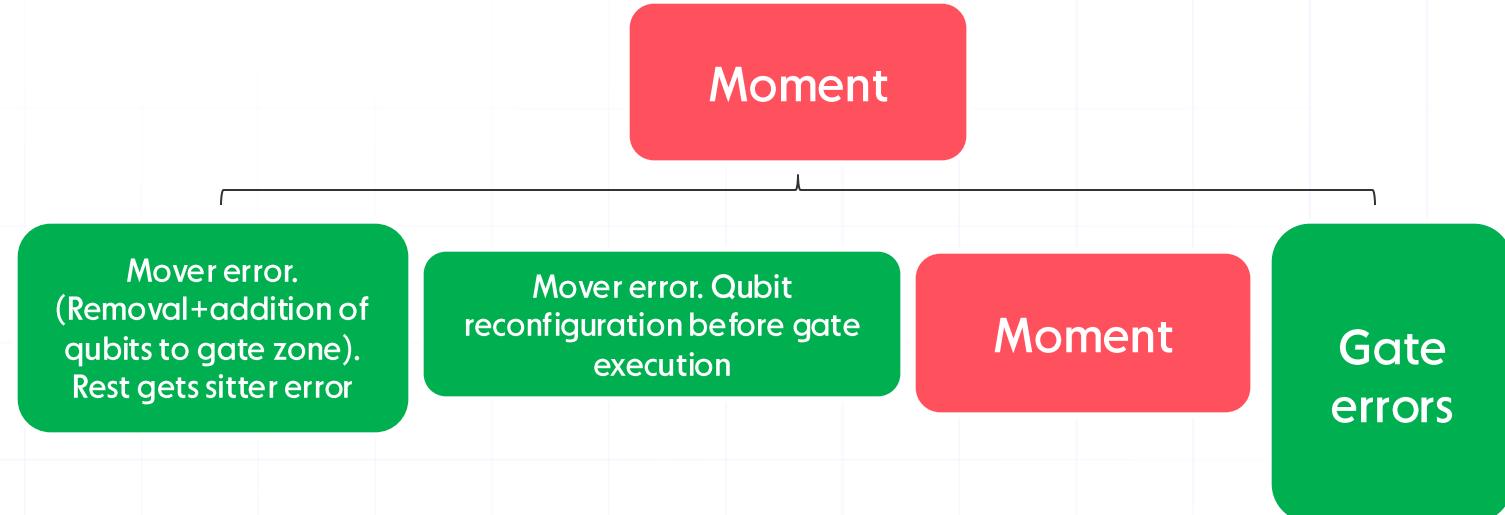


Heuristic noise model - two-zone logic

Step 1.
Transpilation to U3 (or PhasedXZ)+CZ gateset



Step 2.
`noisy_circuit = get_two_zoned_noisy_circ(circuit)`
Main assumption: all gates in a moment are executed in the gate zone

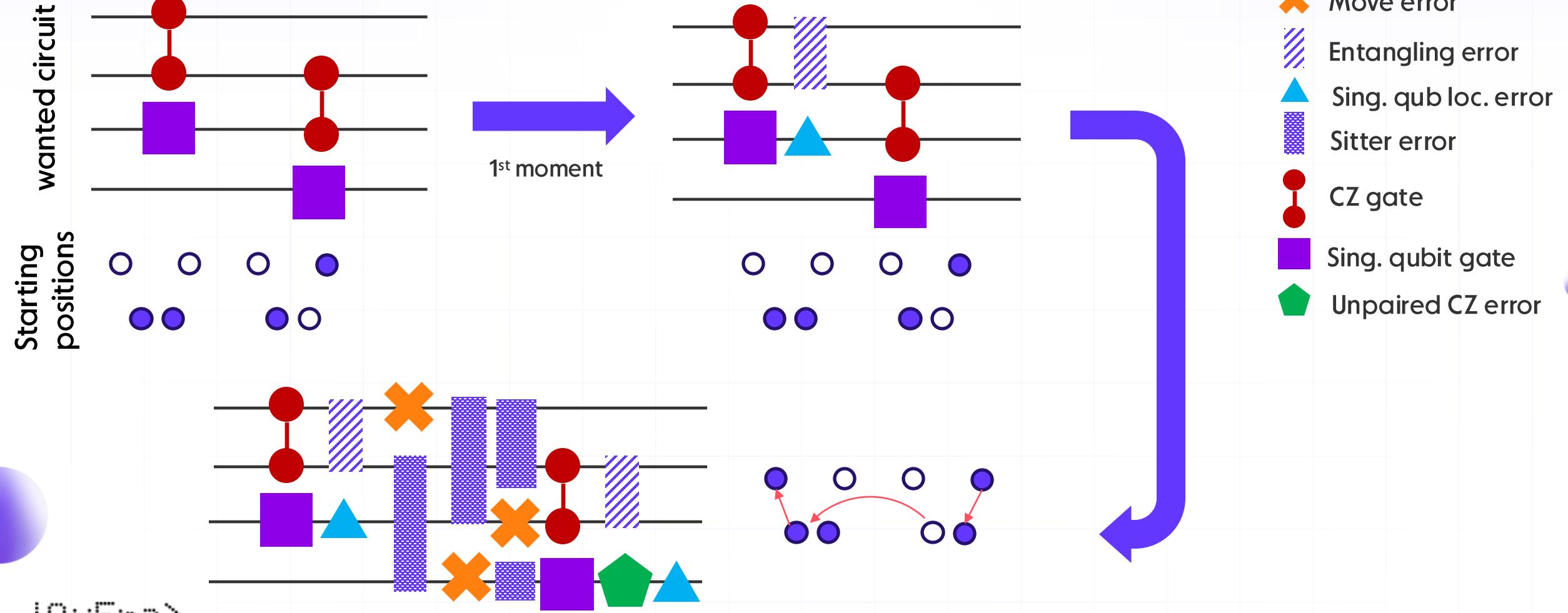


Caveats:

- Assumes qubits **must** be in entangling zone to be operated on
- Gates in a moment are performed **together**

Heuristic noise model - two-zone logic

Example/Exercise



Example calls

Define your circuit (say in Cirq)

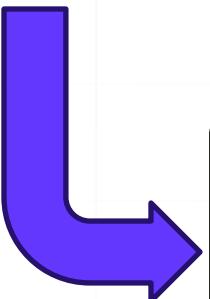
```
import cirq
from bloqade.cirq_utils import noise

def ghz_circuit(n: int) -> cirq.Circuit:
    qubits = cirq.LineQubit.range(n)

    # Step 1: Hadamard on the first qubit
    circuit = cirq.Circuit(cirq.H(qubits[0]))

    # Step 2: CNOT chain from qubit i to i+1
    for i in range(n - 1):
        circuit.append(cirq.CNOT(qubits[i], qubits[i + 1]))

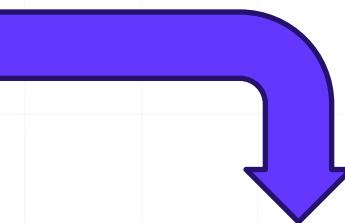
    return circuit
```



```
ghz_circuit_3 = ghz_circuit(3)
print(ghz_circuit_3)
✓ 0.0s
0: —H @—
      |
1: —X @—
      |
2: —X —
```

Pick up your noise model of interest

```
noise_model = noise.G
└── GeminiTwoZoneNoiseModel
└── GeminiOneZoneNoiseModel
└── GeminiOneZoneNoiseModelABC
└── GeminiOneZoneNoiseModelConflictGraphMoves
└── GeminiOneZoneNoiseModelCorrelated
└── {} conflict_graph
└── OneZoneConflictGraph
```

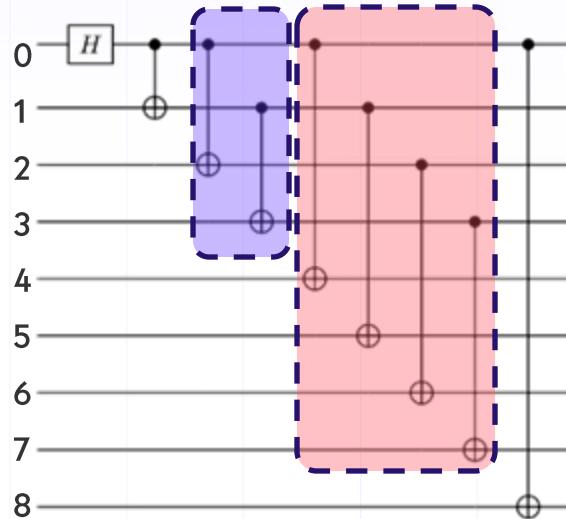


Generate your noisy circuit! (and simulate, etc...)

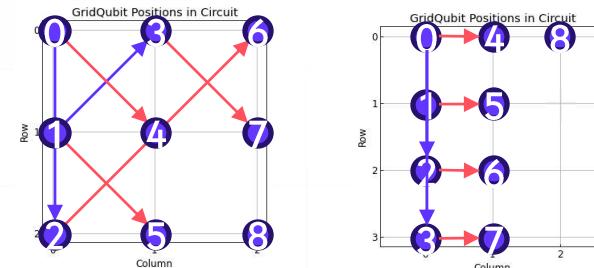
```
noise_model = noise.GeminiOneZoneNoiseModel()
noisy_ghz_circuit_3 = noise.transform_circuit(ghz_circuit_3, model=noise_model)
print(noisy_ghz_circuit_3)
✓ 0.0s
0: —PhXZ(a=0.5,x=0.5,z=0)—A(0.00041,0.00041,0.000411)—A(0.000806,0.000806,0.002
1: —PhXZ(a=0.5,x=0.5,z=0)—A(0.00041,0.00041,0.000411)—A(0.000307,0.000307,0.000
2: —A(0.000307,0.000307,0.000
```

Circuit design practices

Evaluating move error



1-zone routing examples



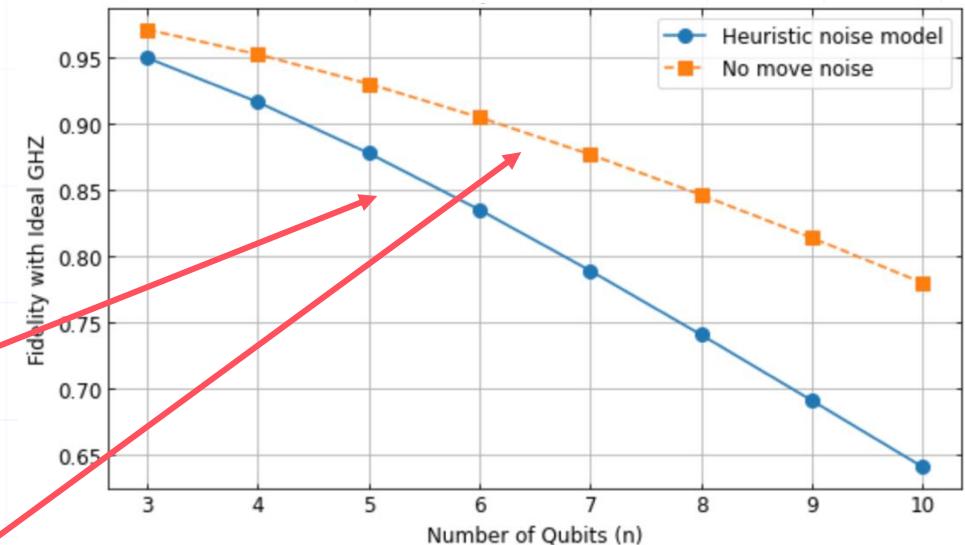
Atom-moving No atom-moving
conflicts conflicts

```
# standard one zone noise model
one_zone_std = noise.GeminiOneZoneNoiseModel()
# define your own adjusted model
onezone_no_move_noise = noise.GeminiOneZoneNoiseModel(
    mover_px=0, mover_py=0, mover_pz=0,
    sitter_px=0, sitter_py=0, sitter_pz=0
)
full_noise_ghz = noise.transform_circuit(ghz_circuit(n_qubits), model=noise_model)
noiseless_move_ghz = noise.transform_circuit(ghz_circuit(n_qubits), model=onezone_no_move_noise)
```

simulate

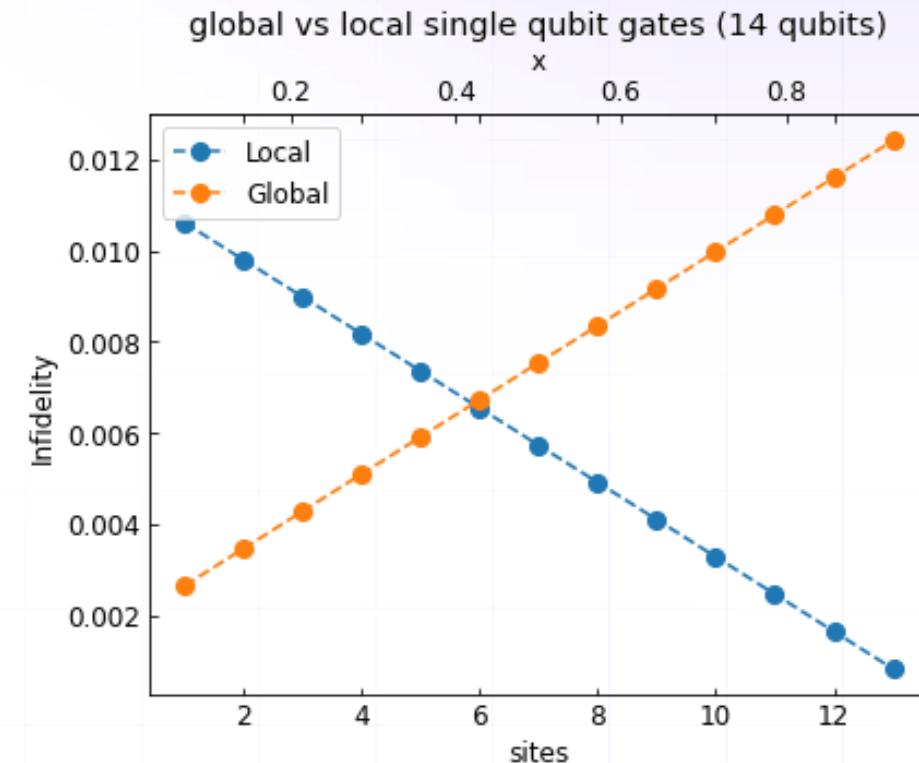
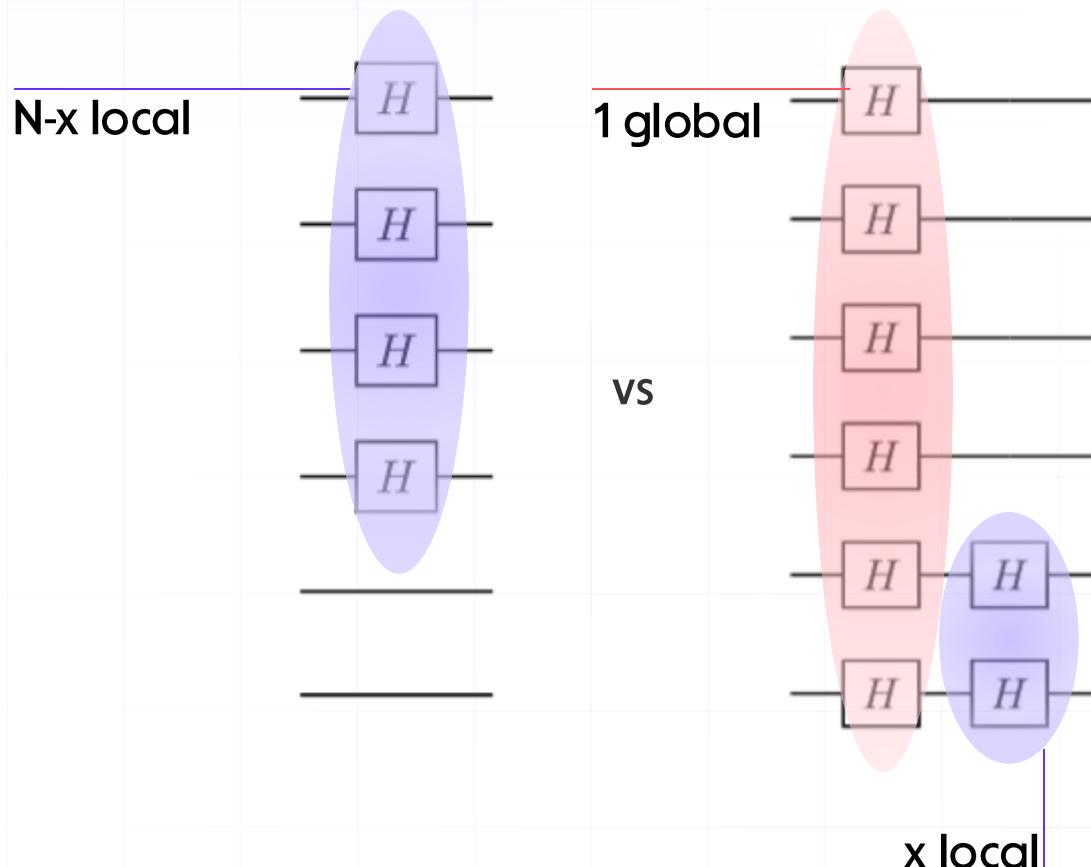
Lesson:

The heuristic noise model overestimates moves and different routing compilations will lead to different performance predictions. Bound realistic performance by running simulations with suppressed move errors!



Circuit design practices

Global-local balance



$$F = (1 - \epsilon_L)^{N-x}$$

$$F = (1 - \epsilon_G)^N (1 - \epsilon_L)^x$$

$$x_c \approx 0.421N$$

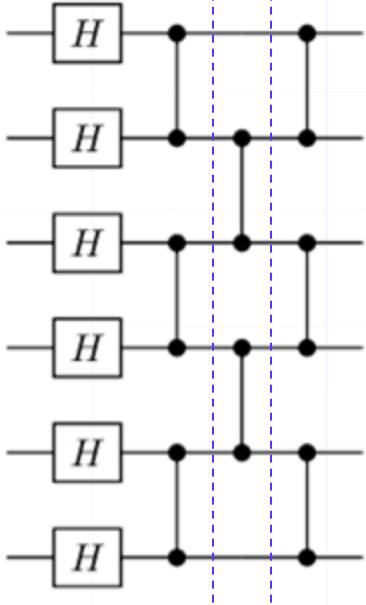
Lesson:

If circuit has a moment with more than 60% equal 1-qubit gates, it is better to exchange for a global and undo the undesired gates

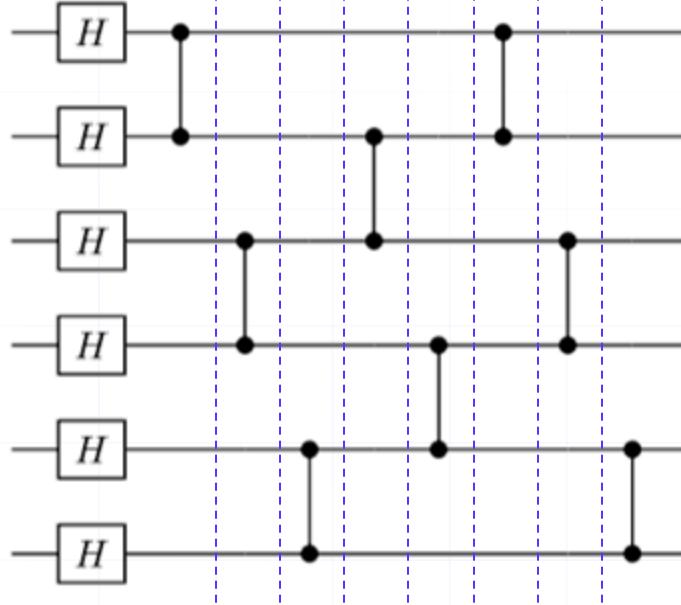
Circuit design practices

Parallelism and zones

Parallel
brick wall



Series
brick wall

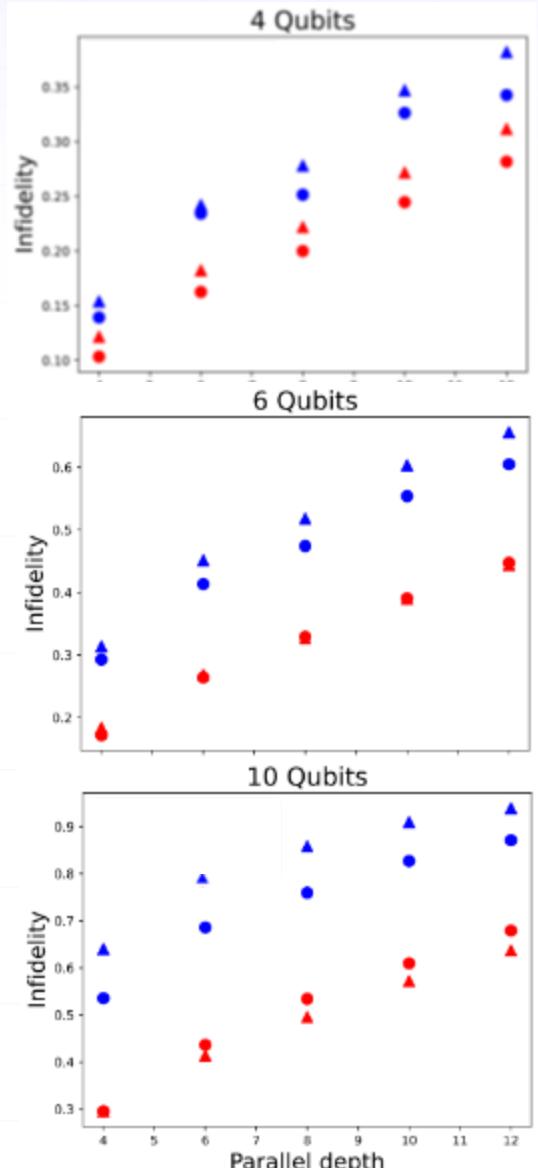


Brick wall circuit Loschmidt echo

Lessons:

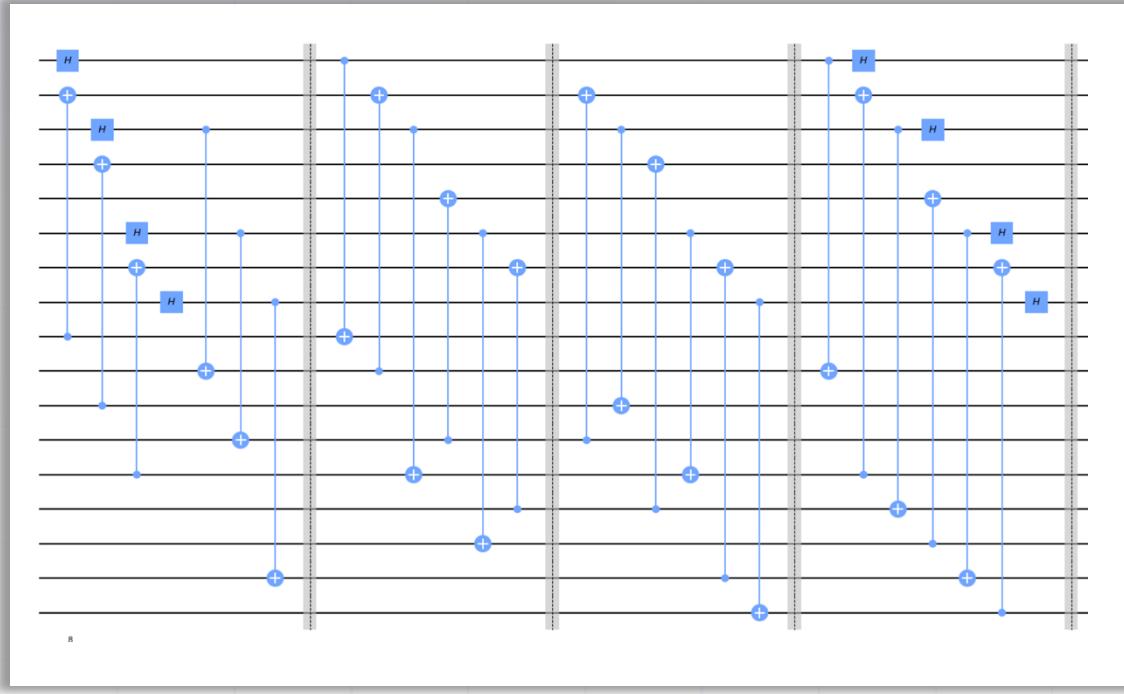
- Parallelization is always better!
- Zones \Rightarrow performance crossover
- Similar performance in zones due to high gate density and similar move dynamics

- ▲ Parallel One Zone
- ▲ Non Parallel One Zone
- Parallel Two Zones
- Non Parallel Two Zones

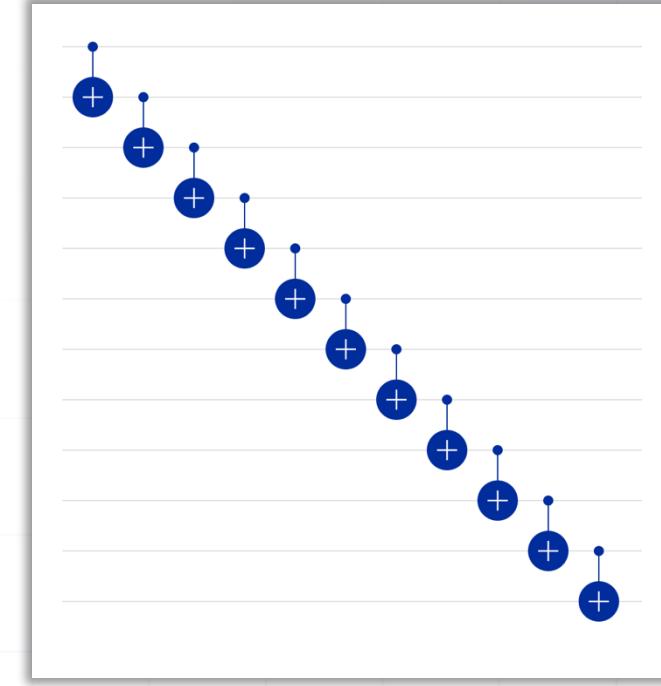


| Summary: what makes a good neutral-atoms program?

Parallelism is key



A round of syndrome extraction
for the surface code



A staircase circuit



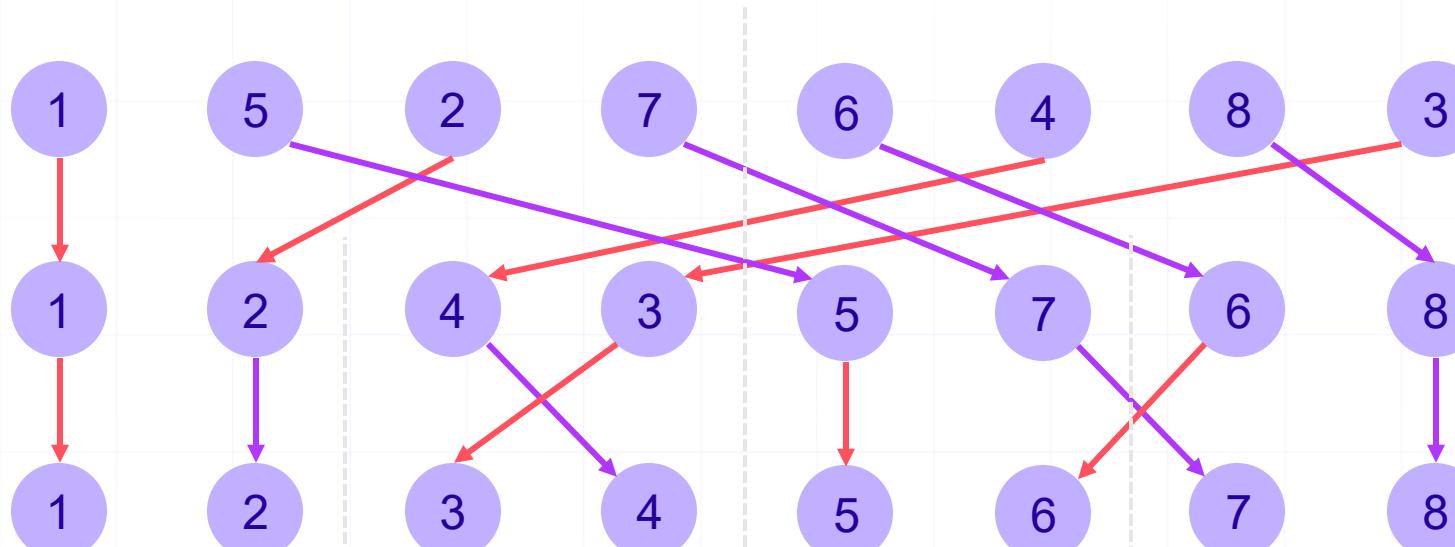
Careful choice of atom layout

~~"All-to-all"~~

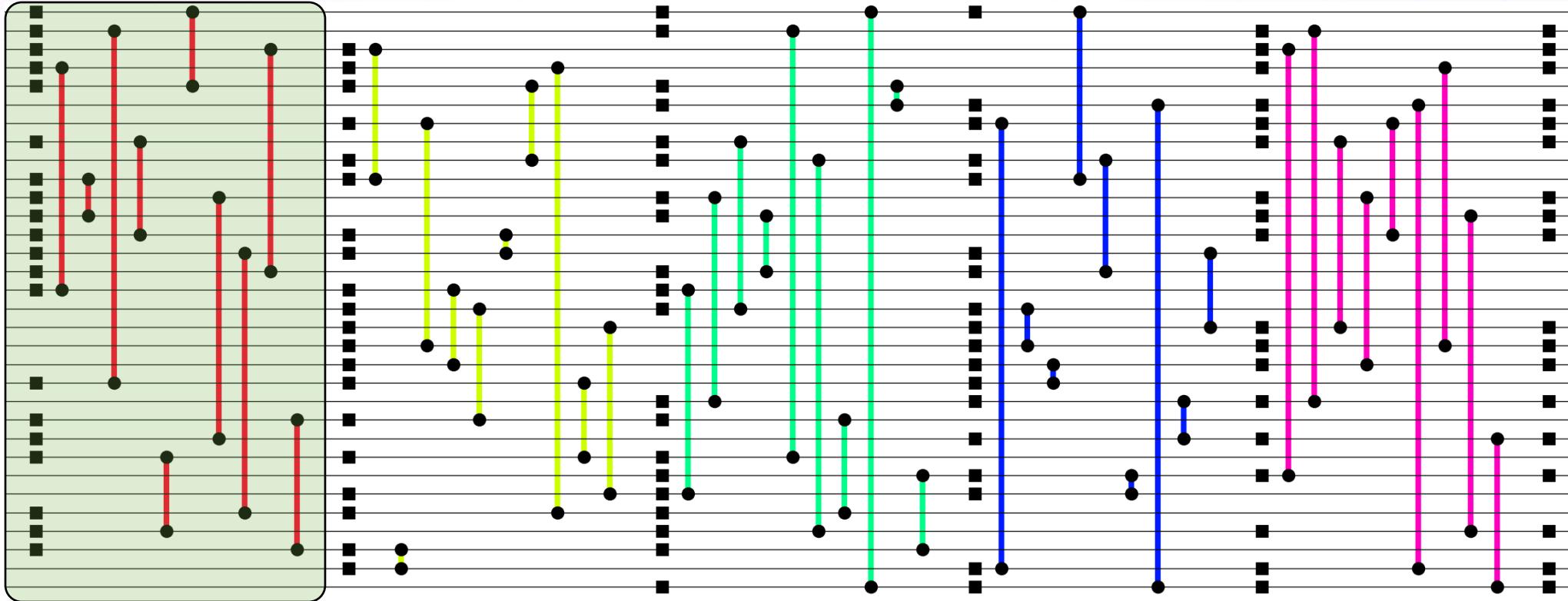


Efficient parallel swap

Atoms can be efficiently sorted in $\sim \log_2(N)$ parallel moves.



A healthy circuit: an overview



Learning objectives

Now you are be able to:

- **Describe** the baseline functionality of a digital neutral-atom quantum computer architecture based on zoned architectures
- **State** constraints for atom shuttling operations, register sizes, and geometrical and temporal scales
- **Write down** a phenomenological hierarchy of the errors for different operations during a circuit execution
- **Exemplify** design rules for hardware-aware implementation of quantum circuits

- **Hierarchy of eDSL/dialects/languages in Bloqade**

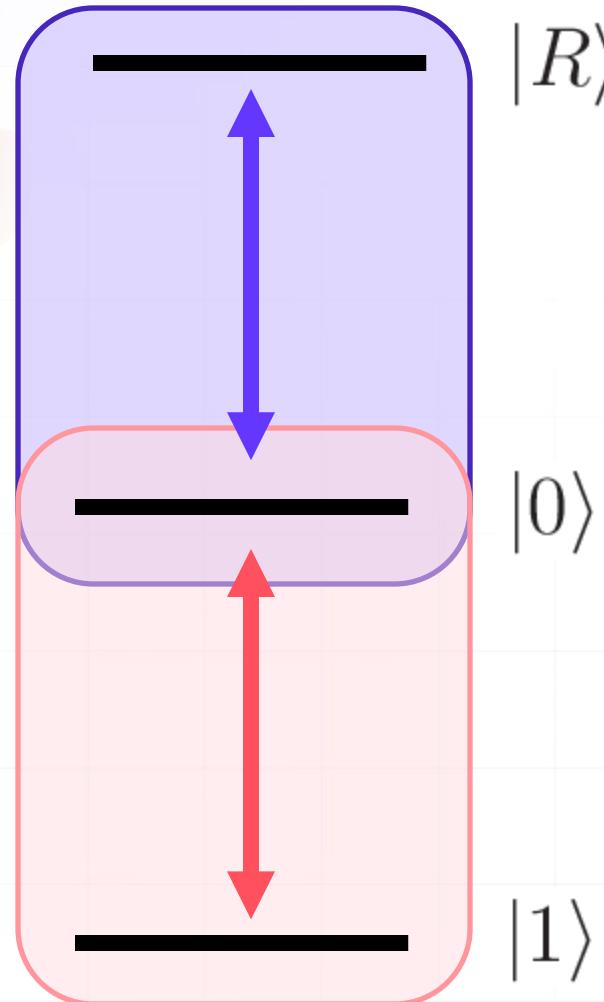
An example and more activities:

Appendix

Digital mode qubits

Analog (Rydberg)
qubit: $|0\rangle$ and $|R\rangle$

Digital (Hyperfine)
qubit: $|0\rangle$ and $|1\rangle$



Interactions and entanglement:

Coherently drive dynamics
with the Rydberg state using
a 2-photon transition

Single-qubit gates:

Coherently drive dynamics
with the hyperfine states using
a Raman transition

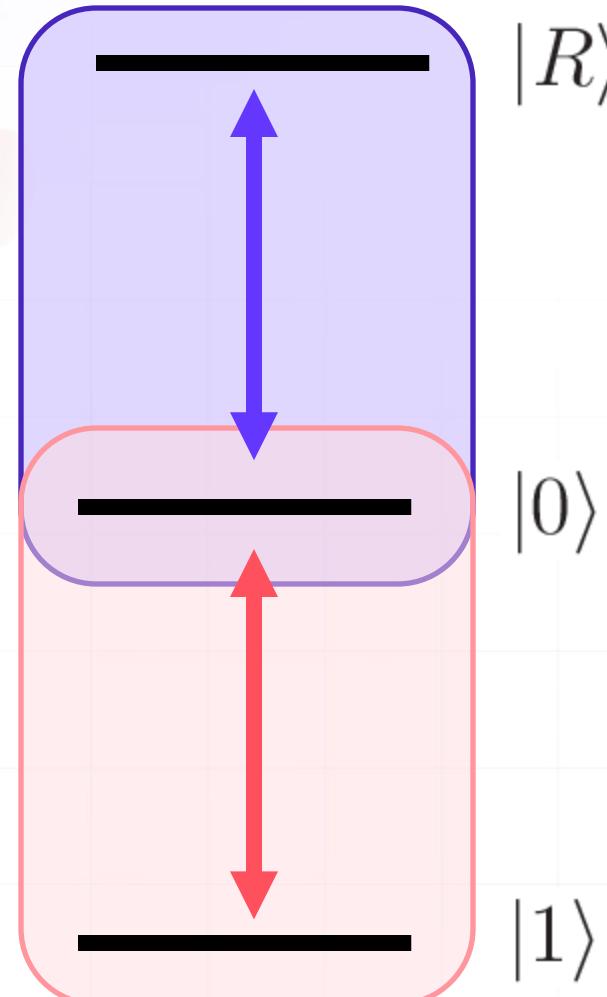
Hyperfine states

Insensitive to the environment
with a very long coherence time

Digital mode qubits

Analog (Rydberg)
qubit: $|0\rangle$ and $|R\rangle$

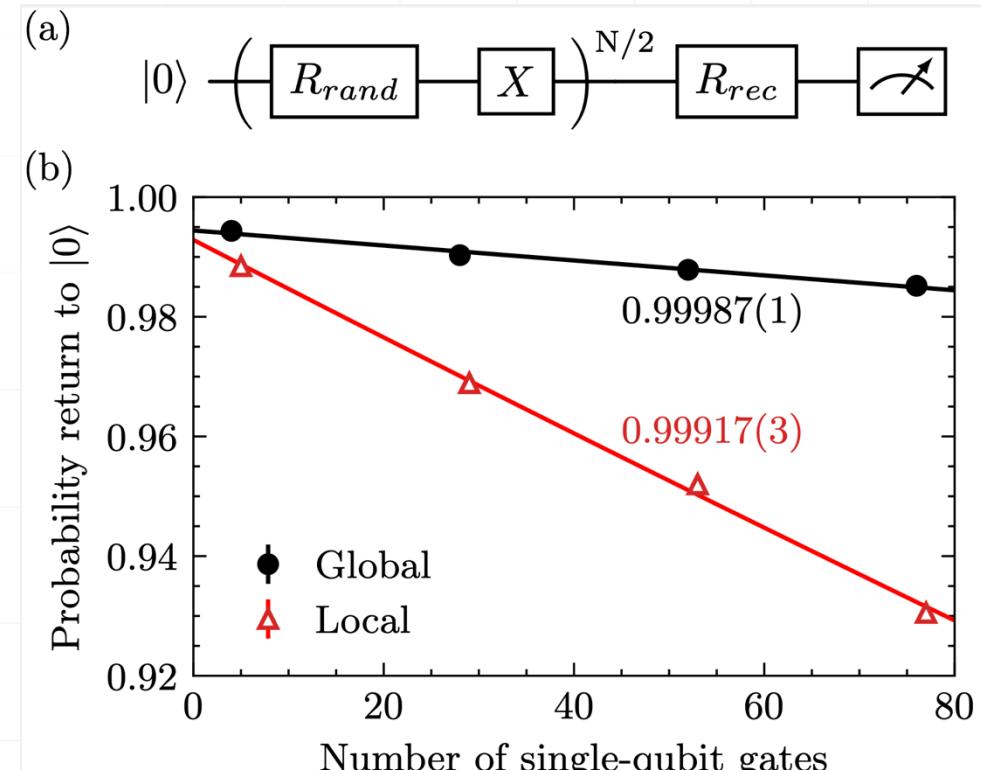
Digital (Hyperfine)
qubit: $|0\rangle$ and $|1\rangle$



Single-qubit gates:

99.917% fidelity local gate

99.987% fidelity global gate



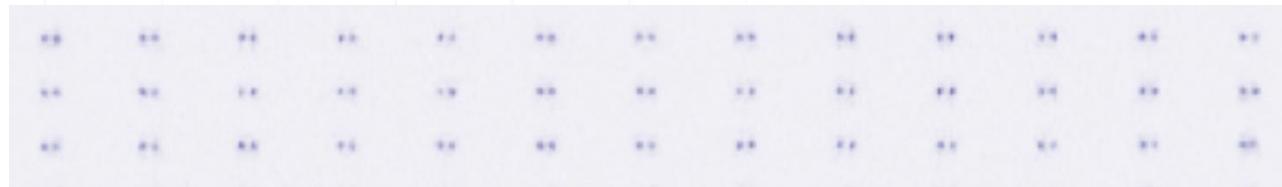
Entangling gates by moving atoms

^{87}Rb

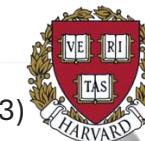
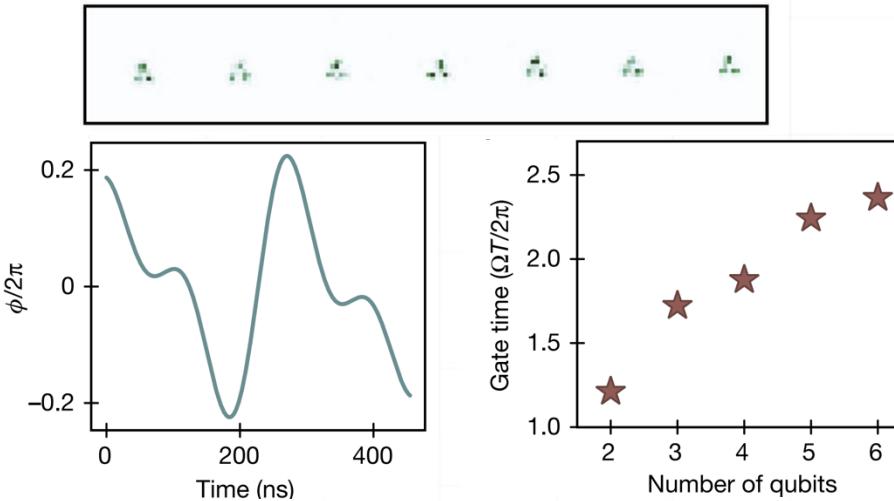
Entangling gates with the Rydberg blockade

Robust entangling gates enabled by the Rydberg blockade

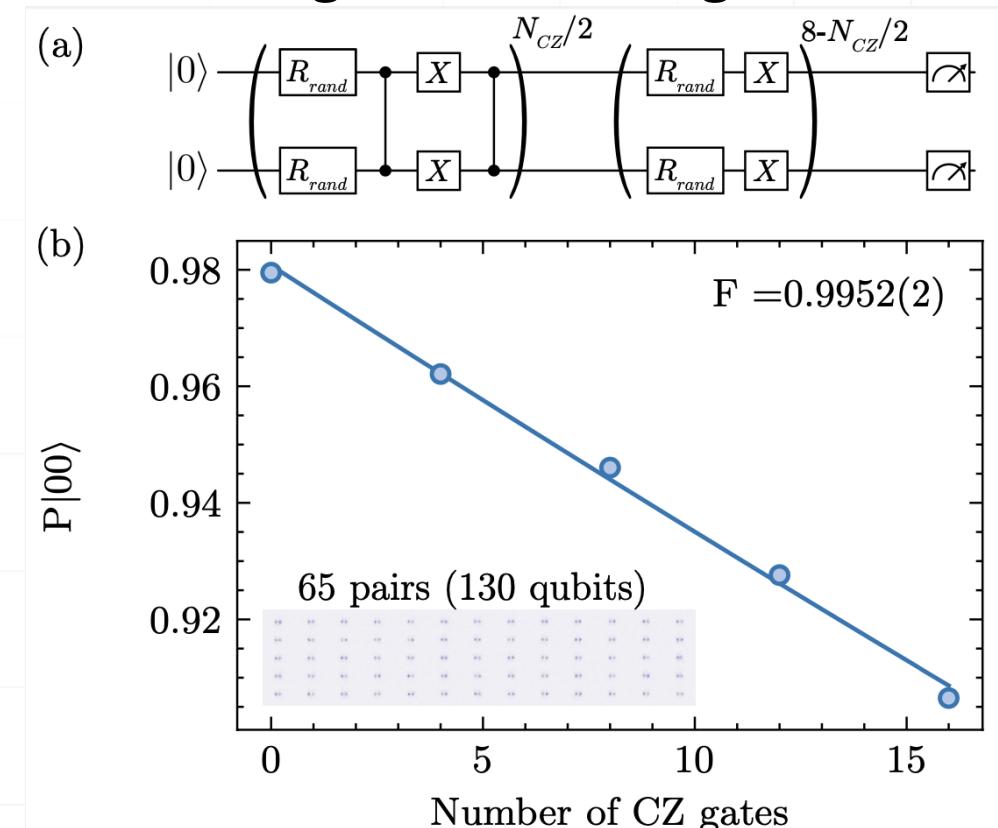
Rydberg blockade: only 1 atom can be in the Rydberg state. Multi-Rydberg states can be energetically excluded from dynamics. Move atoms together to do a gate



Blockade naturally extends to **multi-qubit gates**



Evered (Harvard) et al. *Nature* volume 622, pages 268–272 (2023)



From Gemini-class hardware in QuEra Boston HQ

| Meeting QuEra at IEEE QCE 2026