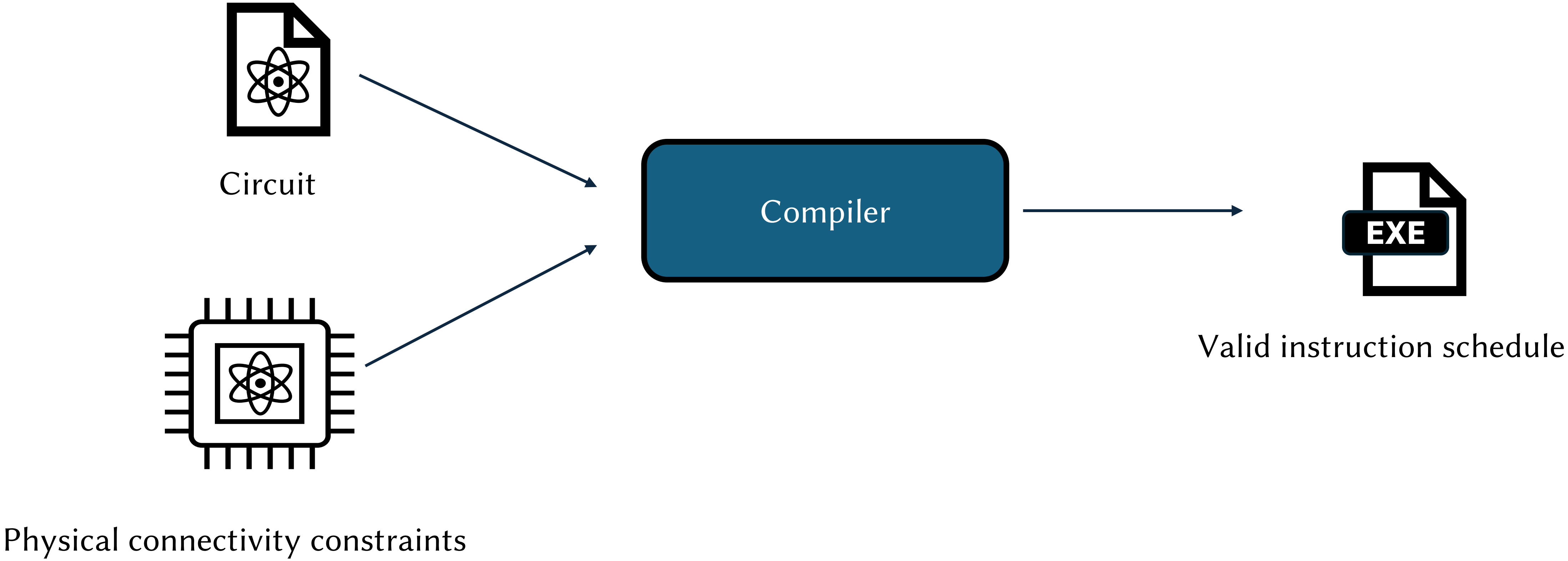
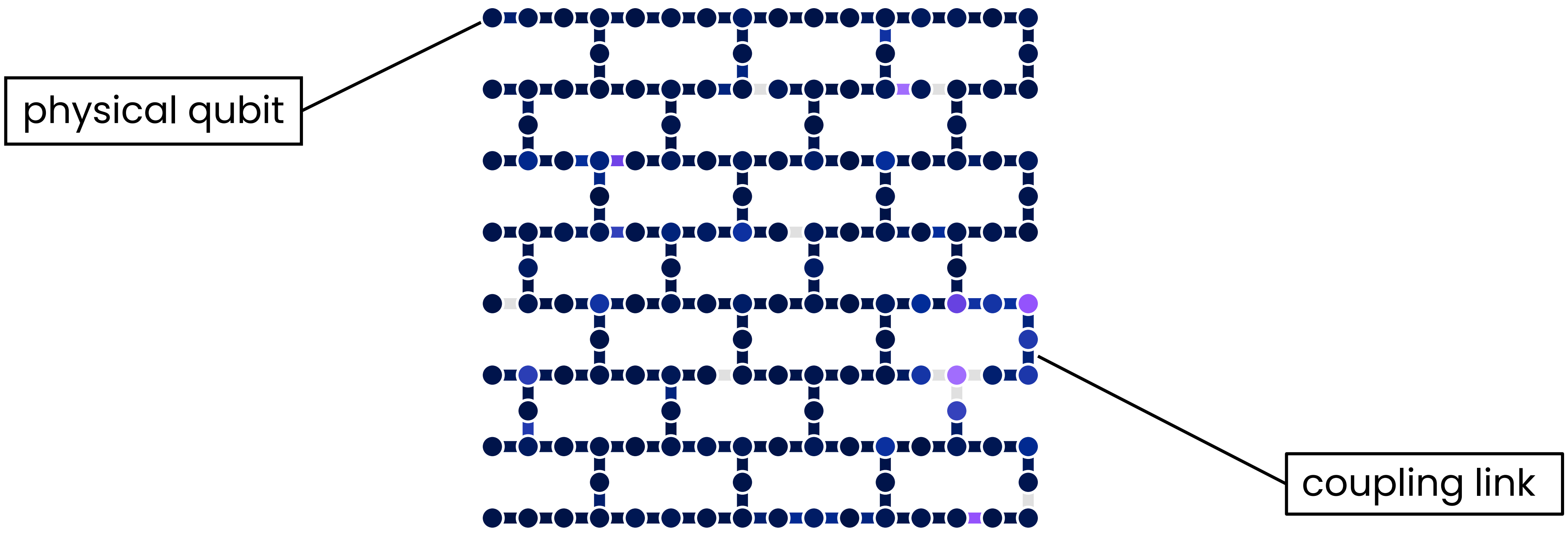
Qubit Mapping and Routing

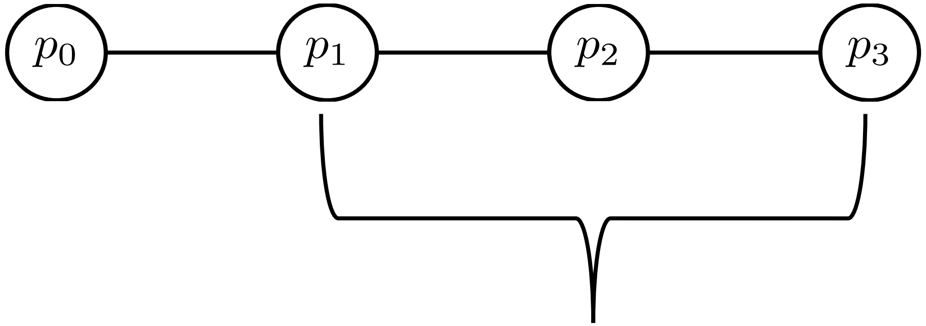

Circuit
Physical connectivity constraints
Compiler
EXE
Valid instruction schedule

NISQ Qubit Mapping and Routing

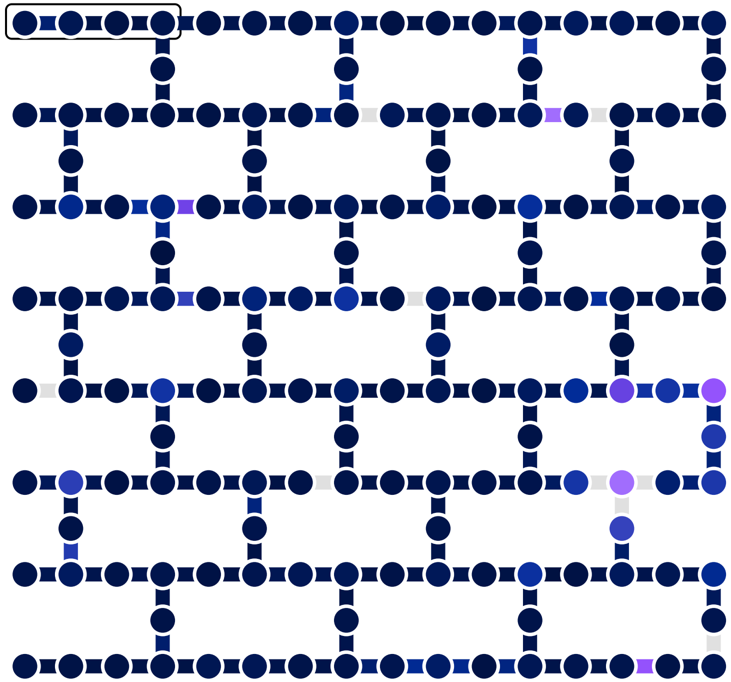
Connectivity is limited



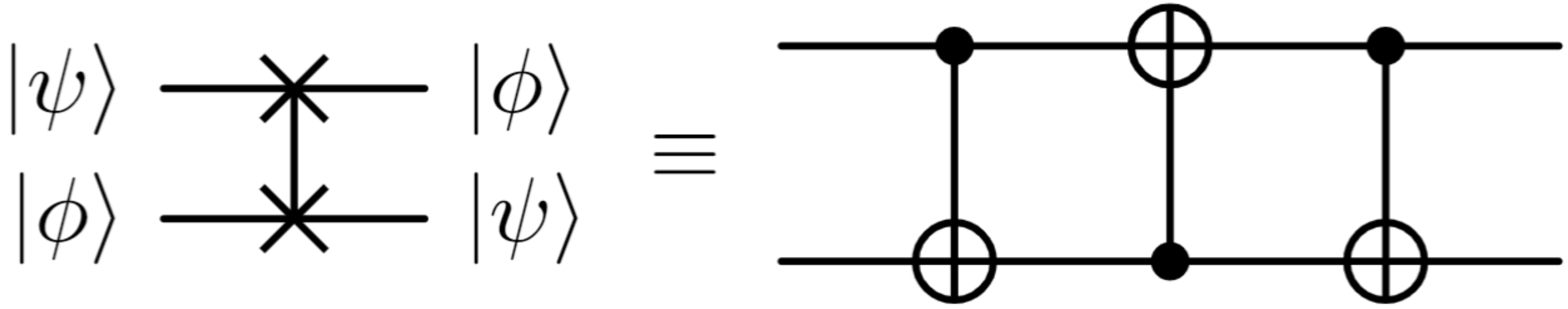
Connectivity is limited



CNOT p_1, p_3



The SWAP gate

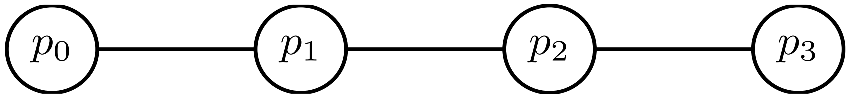


Routing with added SWAPs

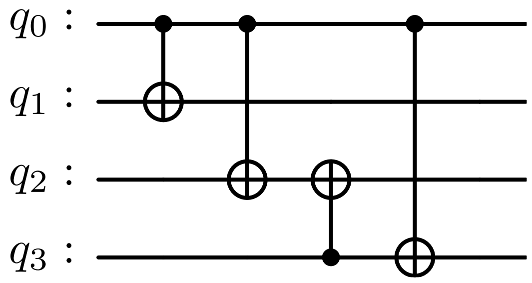
CNOT p_1, p_3

SWAP p_2, p_3
CNOT p_1, p_2

NISQ Qubit Mapping and Routing



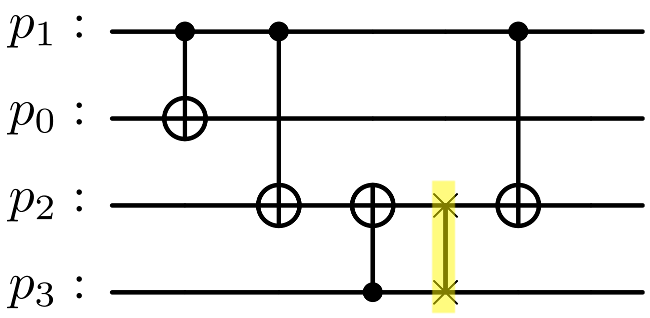
Architecture



Circuit

$$q_0 \mapsto p_1$$
$$q_1 \mapsto p_0$$
$$q_2 \mapsto p_2$$
$$q_3 \mapsto p_3$$

Initial Mapping



Circuit with inserted SWAPs

Extra two-qubit gates are costly!

ibm_**marrakesh**

QPU status: Online

Processor type: Heron r2

Qubits	2Q error (best/layered)	CLOPS
156	1.23e-3/3.88e-3	195K

Willow System Metrics	
Number of qubits	105
Average connectivity	3.47 (4-way typical)
Quantum Error Correction (Chip 1)	
Single-qubit gate error[1] (mean, simultaneous)	0.035% ± 0.029%
Two-qubit gate error[1] (mean, simultaneous)	0.33% ± 0.18% (CZ)
Measurement error (mean, simultaneous)	0.77% ± 0.21% (repetitive, measure qubits)
Reset options	Multi-level reset (|1⟩ state and above) Leakage removal (|2⟩ state only)
T_1 time (mean)	68 µs ± 13 µs[2]
Error correction cycles per second	909,000 (surface code cycle = 1.1 µs)
Application performance	$\Lambda_{3,5,7} = 2.14 \pm 0.02$

A spectrum of approaches

Heuristic-based solvers
(MQT-A*, tket, **SABRE**, …)

Constraint-based solvers
(OLSQ, MQT-Exact, **SATMAP**, …)

Scale to large instances
Solutions can be far from optimal

Optimal or near optimal solutions
Cannot handle large inputs

SABRE

“**S**W**A**P-based **B**idi**RE**ctional Search”

Method of choice for the IBM `qiskit` compiler

Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices

Gushu Li, Yufei Ding, and Yuan Xie
University of California, Santa Barbara, CA, 93106, USA
{gushuli,yufeiding,yuanxie}@ucsb.edu

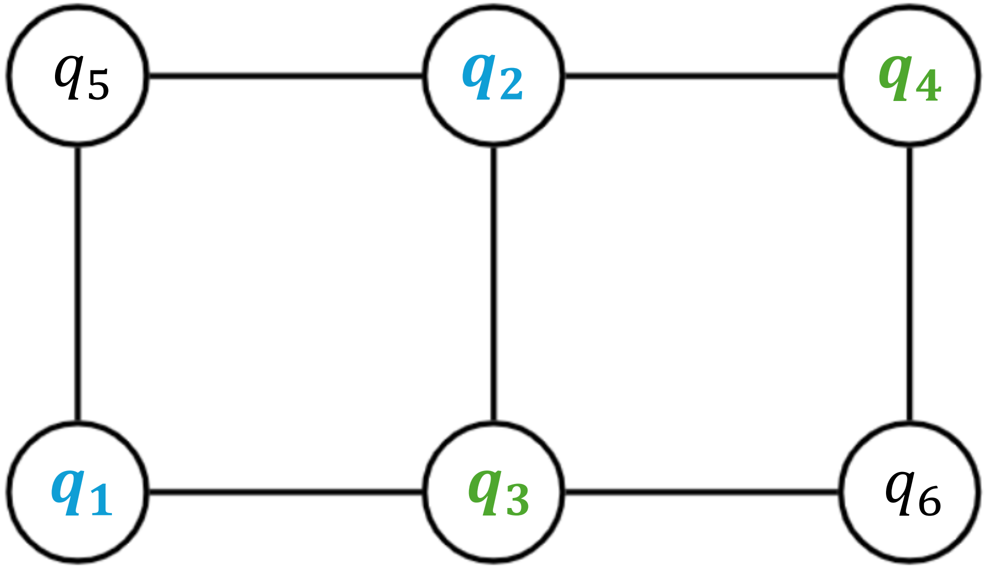
The SABRE routing loop

1. Execute any gates that we can
2. Compute the front layer of non-executable gates *F*
3. Generate a set of candidate SWAPs
   - Those acting on qubits in *F*
4. Choose the candidate with best heuristic cost

repeat until
all gates executed

SABRE Routing in Action

```
CNOT q1,q2
CNOT q3,q4
...
```



Candidates	Distance
SWAP q_1, q_5	1+2=3
SWAP q_1, q_3	1+3=4
SWAP q_2, q_4	3+1=4
SWAP q_2, q_5	1+2=3
SWAP q_2, q_3	1+1=2
SWAP q_4, q_6	2+1=3
SWAP q_3, q_6	2+1=3

Choosing a heuristic

How we define the “best” SWAP is crucial to the algorithm

The cost from the last slide is called the *basic* heuristic

SABRE also includes a *lookahead* and *decay* variant

Lookahead heuristic

Also consider the immediate successors of the front layer, *E*

adjustable weight

$$H = \underbrace{\frac{1}{|F|}\sum_{(i,j)\in F}\mathrm{dist}(i,j)}_{\texttt{basic component}} + \underbrace{\frac{k}{|E|}\sum_{(i,j)\in E}\mathrm{dist}(i,j)}_{\texttt{lookahead component}},$$

Decay heuristic

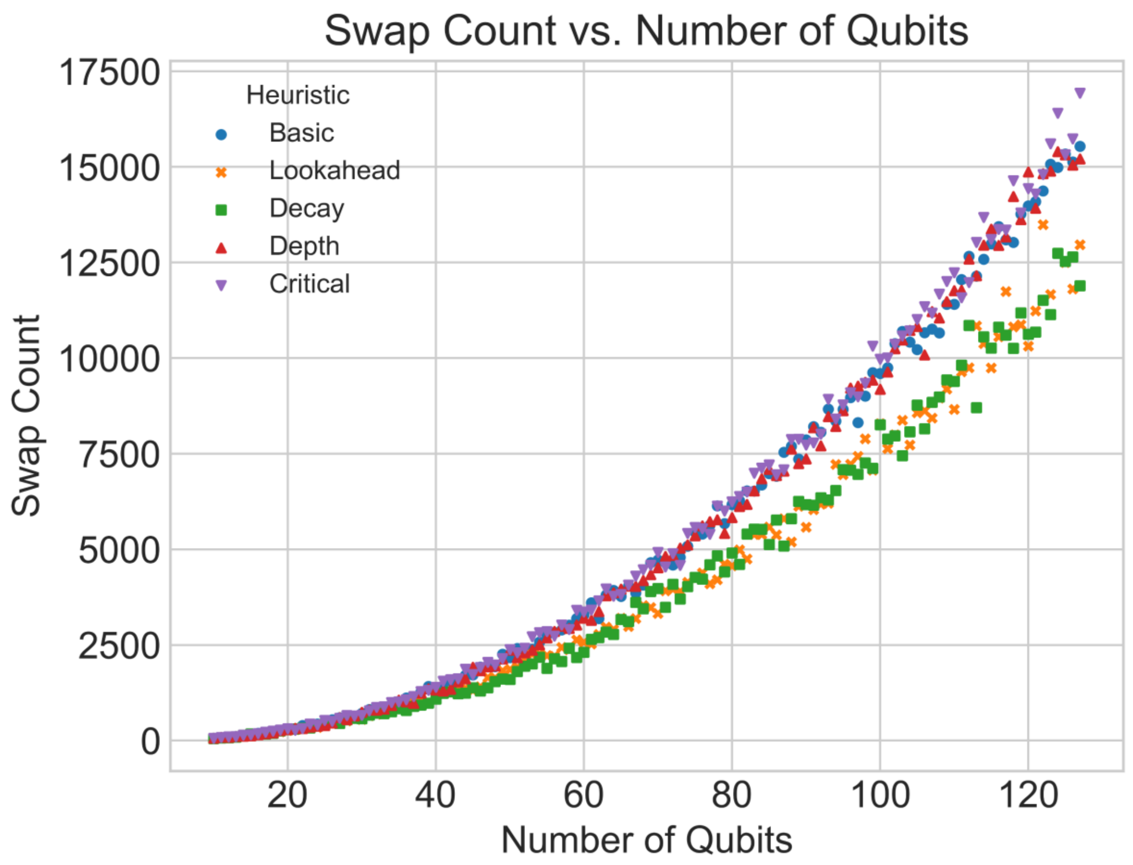
Avoid repeated SWAPs on the same qubit

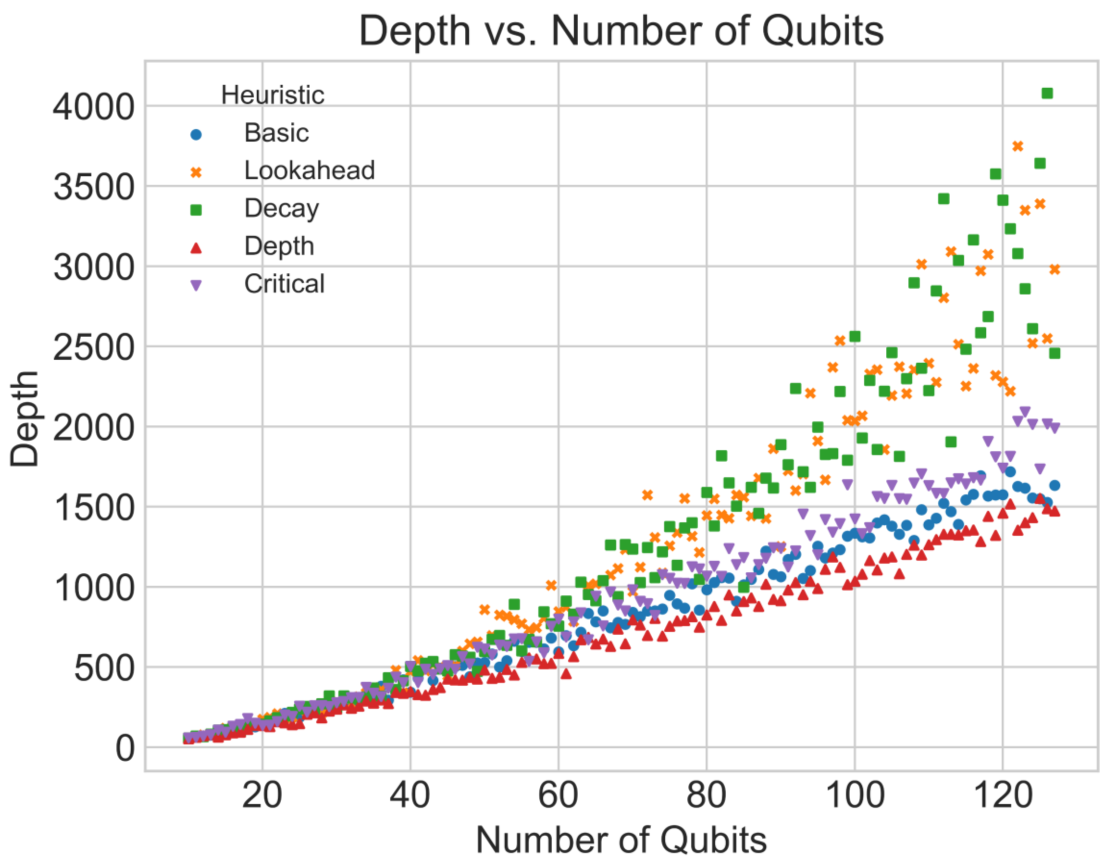
Apply a decay to each qubit, increases with every SWAP

$$H = max(decay(SWAP.q_1), decay(SWAP.q_2)) * \left(\frac{1}{|F|} \sum_{(i,j) \in F} \mathrm{dist}(i, j) + \frac{k}{|E|} \sum_{(i,j) \in E} \mathrm{dist}(i, j) \right)$$

Comparing heuristics

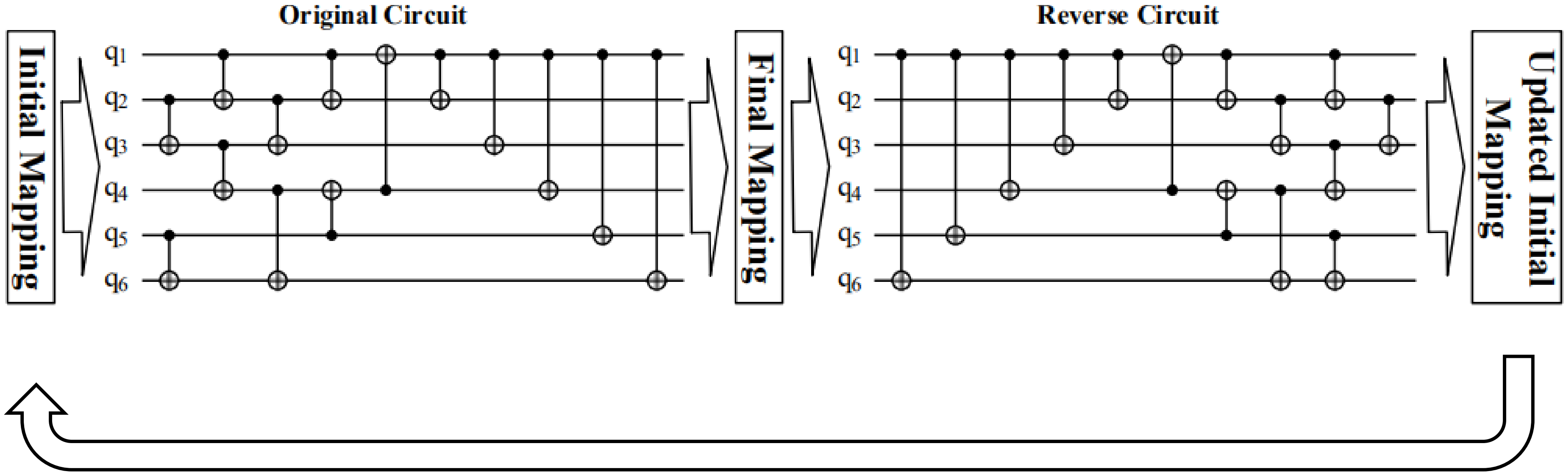
Best choice depends on your objective!





“LightSABRE: A Lightweight and Enhanced SABRE Algorithm” Zou, Treinish, Hartman, Ivrii, Lishman

BiDiREctional Mapping



Final mapping for a reversed circuit is a good initial mapping for the original

Enables a mapping refining loop: solve the forward problem, then reverse, then repeat

SATMAP

Constraint-based approaches: **encode** a problem in the form:

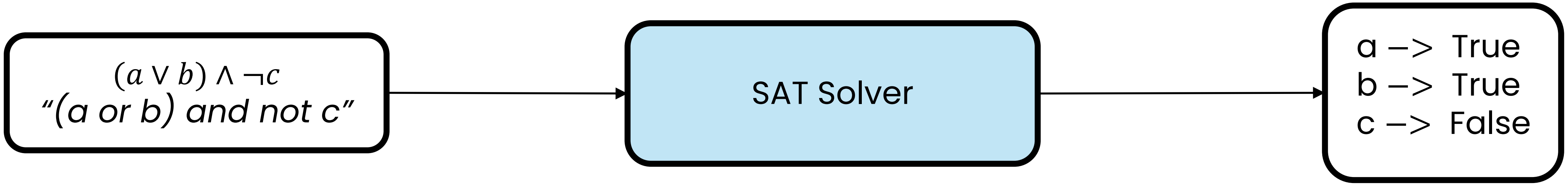
Optimize **objective function**
subject to **constraints**

Qubit Mapping and Routing via MaxSAT

Abtin Molavi, Amanda Xu, Martin Diges, Lauren Pick, Swamit Tannu, Aws Albarghouthi
University of Wisconsin-Madison, Madison, WI, USA
{amolavi, axu44, mdiges, lpick2, stannu, albarghouthi}@wisc.edu

Satisfiability (SAT) solving

Goal: find an assignment of Boolean variables such that the full formula evaluates to *true*



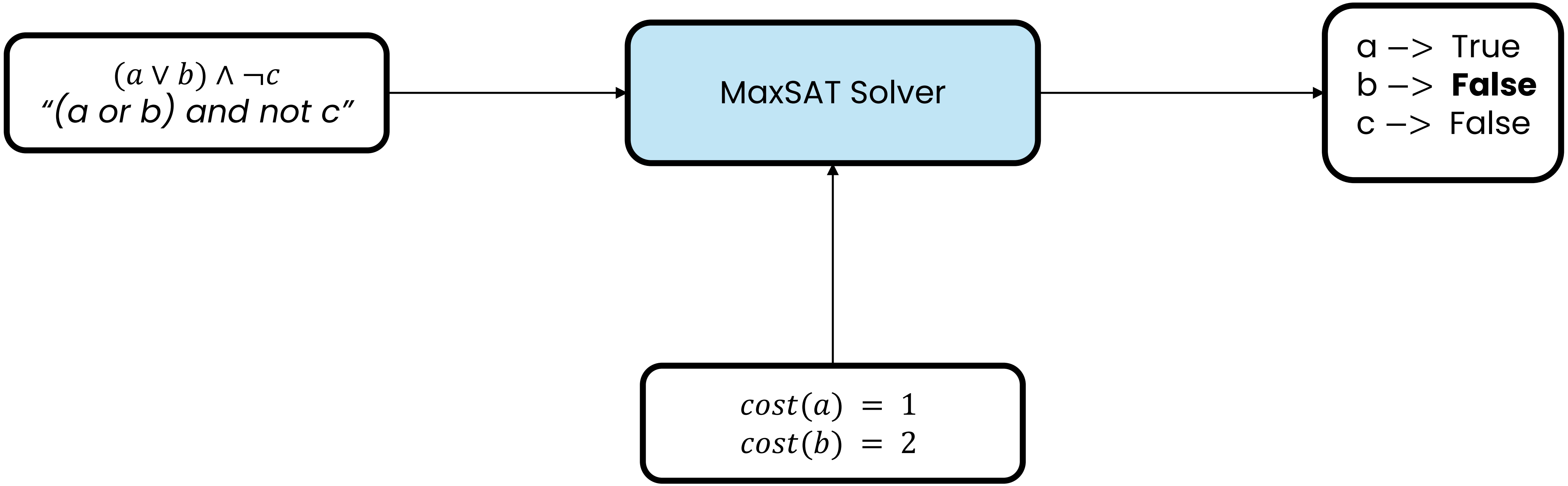
NP-complete, but often tractable in practice!

Many years of solver engineering: https://satcompetition.github.io/

MaxSAT solving

The optimization analogue of satisfiability solving

Express solution cost with soft constraints



Encoding overview

Introduce the following sets of Boolean variables:

- $map(q, p, t)$: Logical qubit q is mapped to physical qubit p at step *t*
- $swap(p, p', t)$: A SWAP operation is executed on edge (p, p') at step *t*

Constraints encode

- The maps are injective functions
- Two-qubit gates are executable
- SWAPs transform maps

Pay a cost of 1 for each swap variable set to *true*

Maps are injective functions

For each pair of distinct circuit qubits q and q’

$$map(q, p, t) \rightarrow \neg map(q', p, t)$$

For each pair of distinct physical qubits p and p'

$$map(q, p, t) \rightarrow \neg map(q, p', t)$$

Two qubit gates are executable

For each two-qubit gate $g_k(q, q')$

$$\bigvee_{(p,p')\in Edges} map(q,p,k) \wedge map(q',p',k)$$

SWAPs transform maps

“no-op” SWAP

For each step t and swap (p_1, p_2) in $Edges \cup \{(0,0)\}$

$$swap(p_1, p_2, t) \rightarrow (map(q, p, t-1) \leftrightarrow map(q, \pi_e(p), t))$$

where

$$\pi_e(p_1) = p_2 \qquad \pi_e(p_2) = p_1$$

$$\pi_e(p) = p \text{ otherwise}$$

Minimize SWAPs

Soft constraint for each step *t* and edge (p, p')

$$cost\big(swap(p, p', t)\big) = 1$$

Note: “No-op” swap is free

Challenge: scaling with gate count

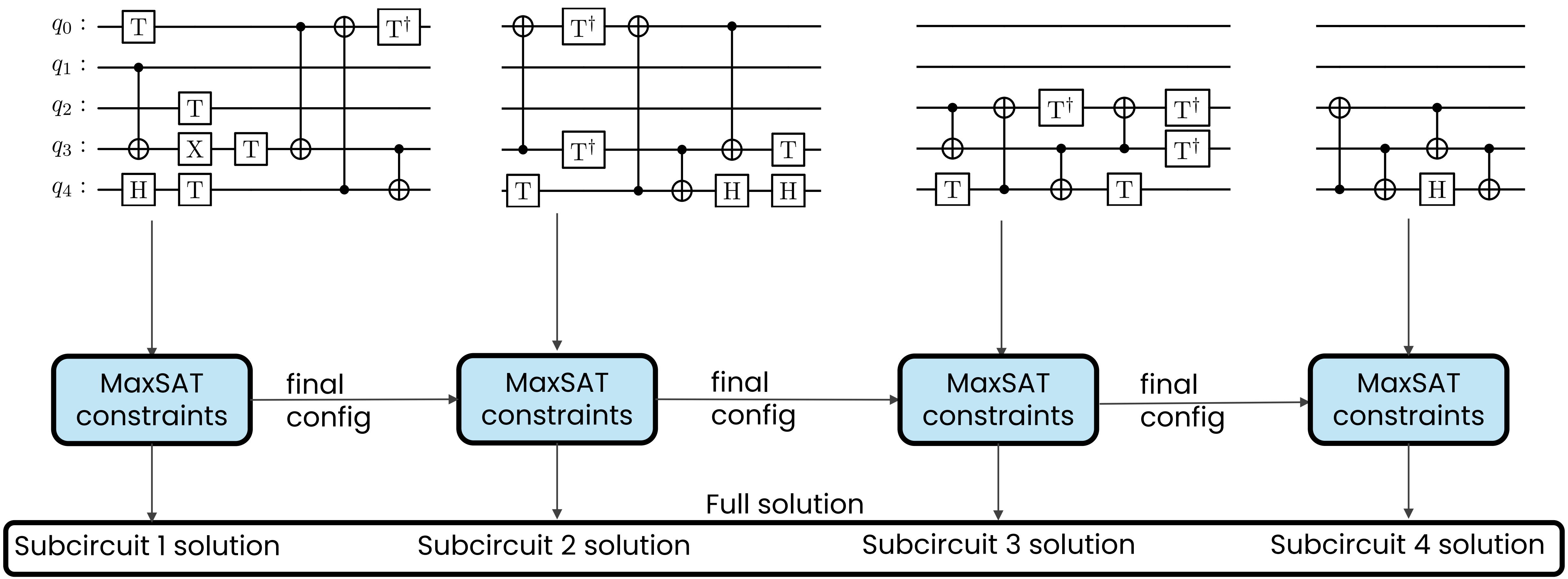
Search space grows exponentially with two-qubit gates count

Global constraint-solving is infeasible for large circuits

SATMAP approach: take a more “local” view

Circuit slicing

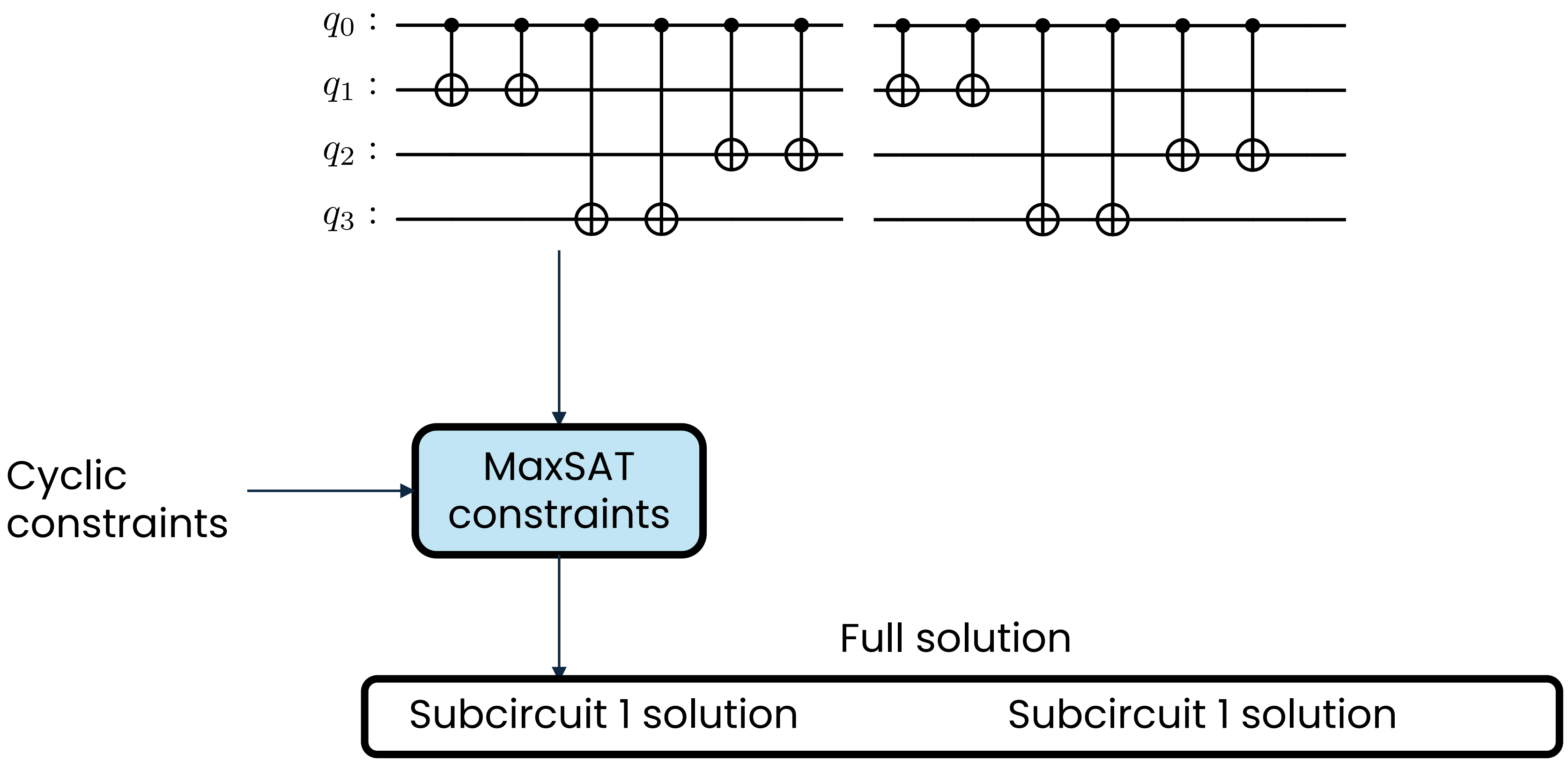
Idea: Solve one subproblem at a time and stitch together



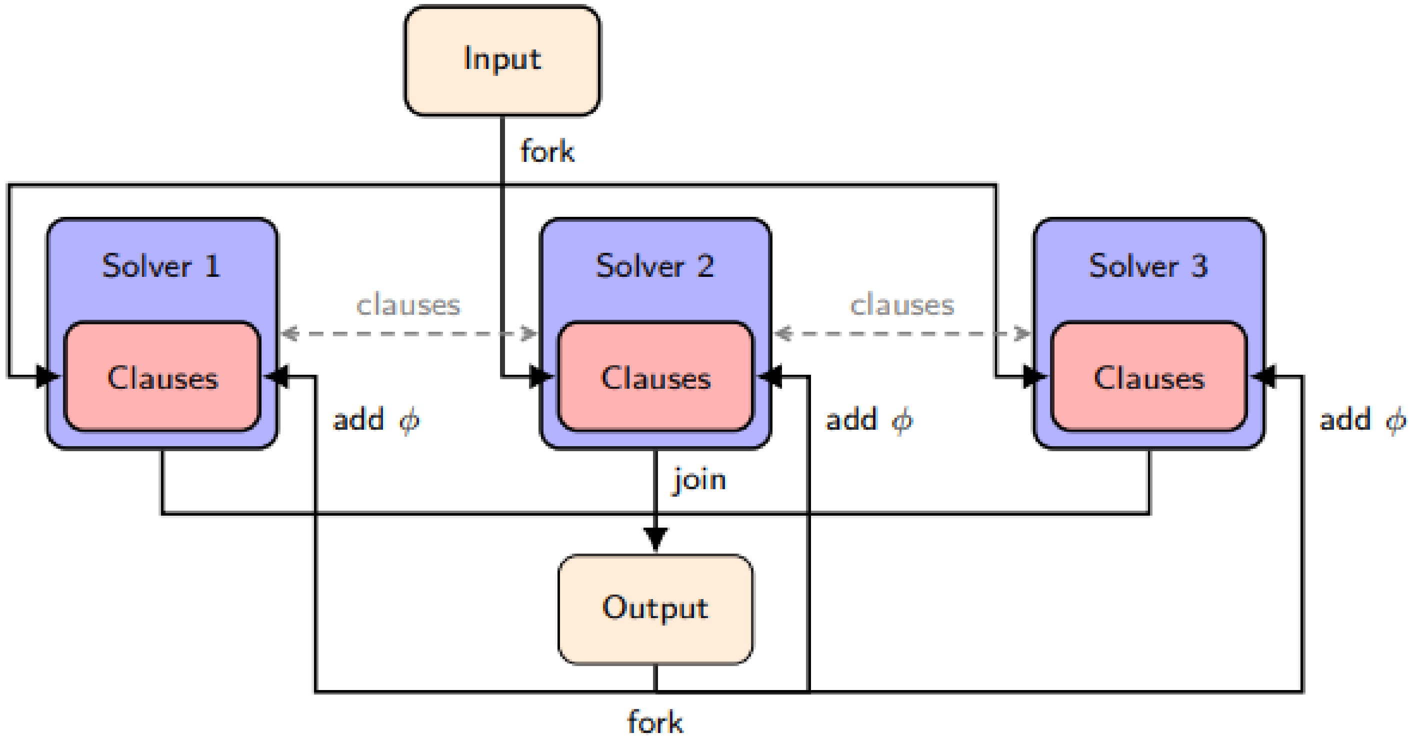
Slicing with reuse

Some applications, like QAOA, have repeating subcircuits

Just need to solve one of these

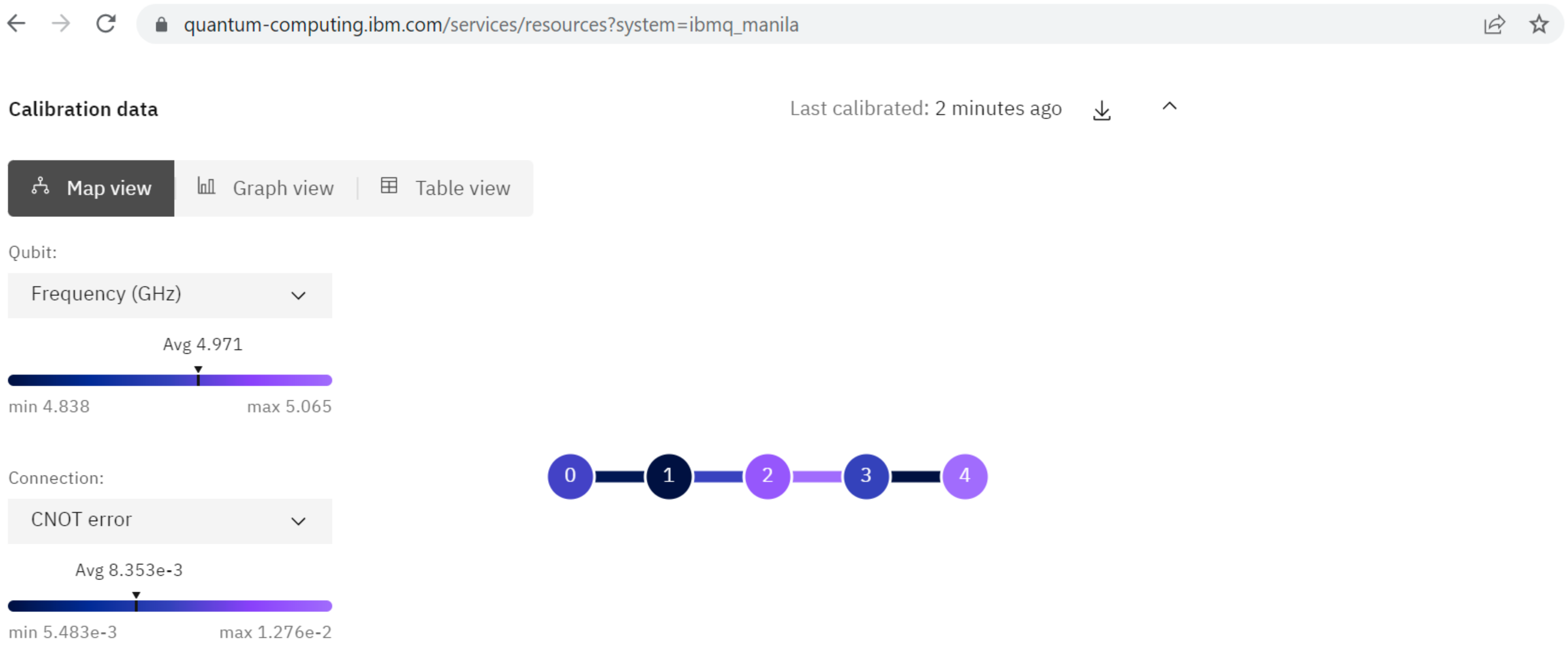


Scaling with parallel solvers



“Quantum Circuit Mapping Based on Incremental and Parallel SAT Solving” Yang et al.
International Conference on Theory and Applications of Satisfiability Testing (2024)

Not all SWAPs have the same cost



Variation-aware approaches

Not All Qubits Are Created Equal

A Case for Variability-Aware Policies for NISQ-Era Quantum Computers

Swamit S. Tannu
swamit@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia

Moinuddin K. Qureshi
moin@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia

Heuristic solver

Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers

Prakash Murali*
Princeton University

Jonathan M. Baker
University of Chicago

Ali Javadi Abhari
IBM T. J. Watson Research Center

Frederic T. Chong
University of Chicago

Margaret Martonosi
Princeton University

Heuristic solver & constraint-based solver

Surface Code Mapping and Routing

Quantum Error Correction

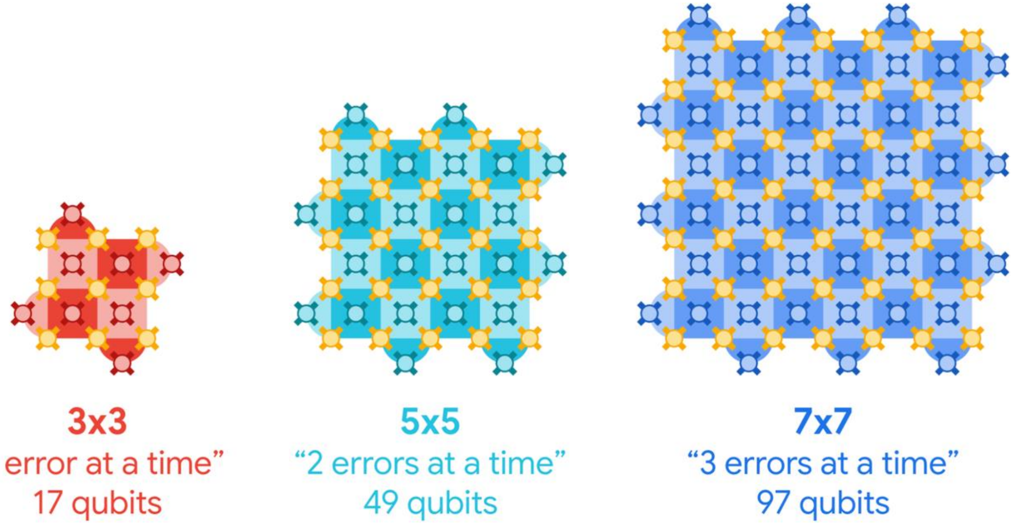
Encode a logical qubit into several physical qubits

Reduce error by scaling the logical qubit

Prerequisite for exciting applications

Shor’s algorithm,

Quantum simulation

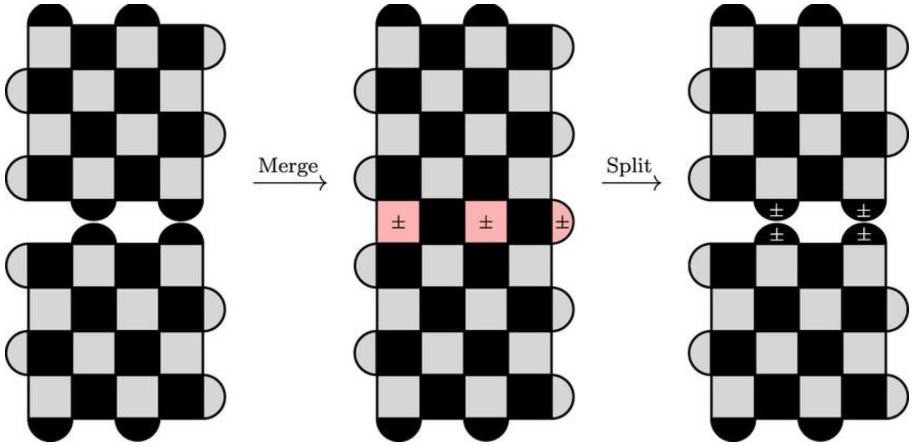


https://research.google/blog/making-quantum-error-correction-work/

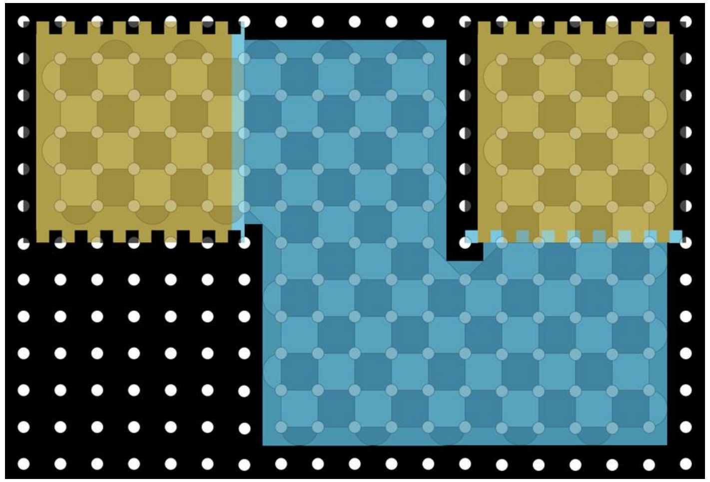
Two-qubit gates via lattice surgery

Lattice surgery: Logical qubits can be *merged* and *split*

Two qubit gates require lattice surgery with an intermediary

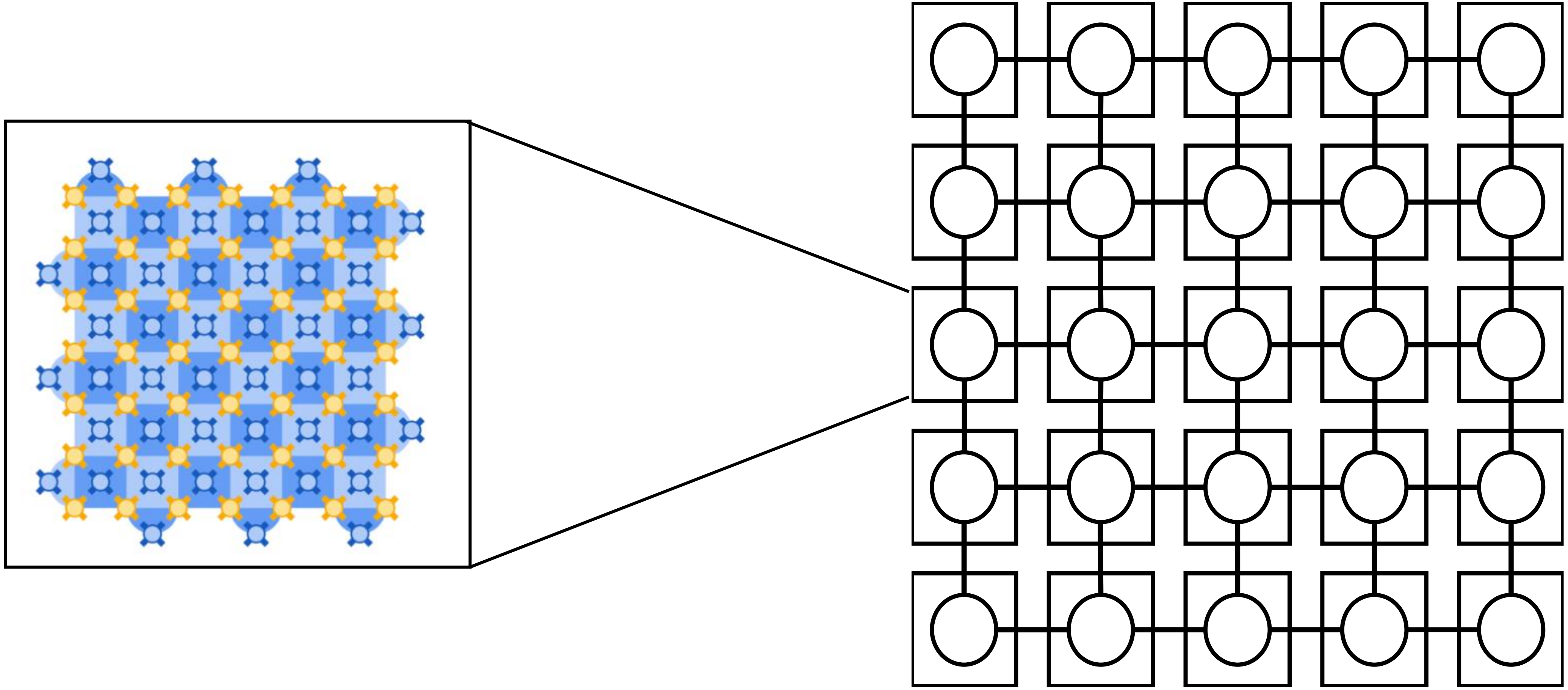


“Code Deformation and Lattice Surgery are Gauge Fixing”
Vuillot et al. New J. Phys. 21 (2019)

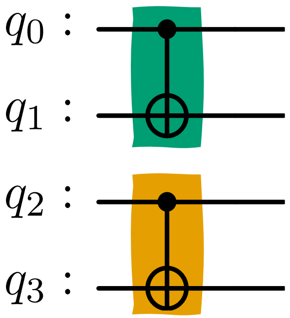


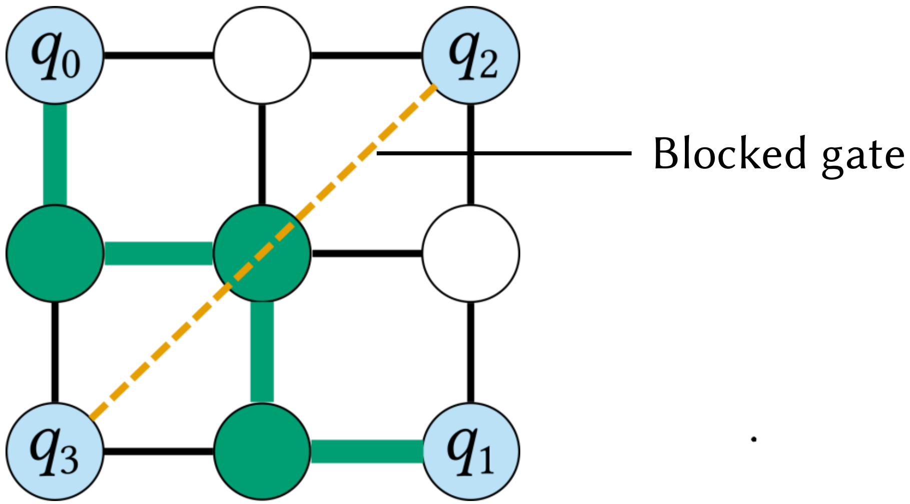
“A High Performance Compiler for Very Large Scale Surface Code Computations” Watkins et al. *Quantum* 8, 1354 (2024).

A graph model

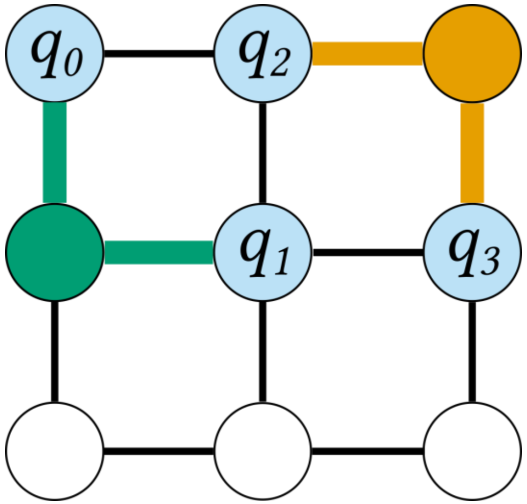


Preserving parallelism





Two step execution



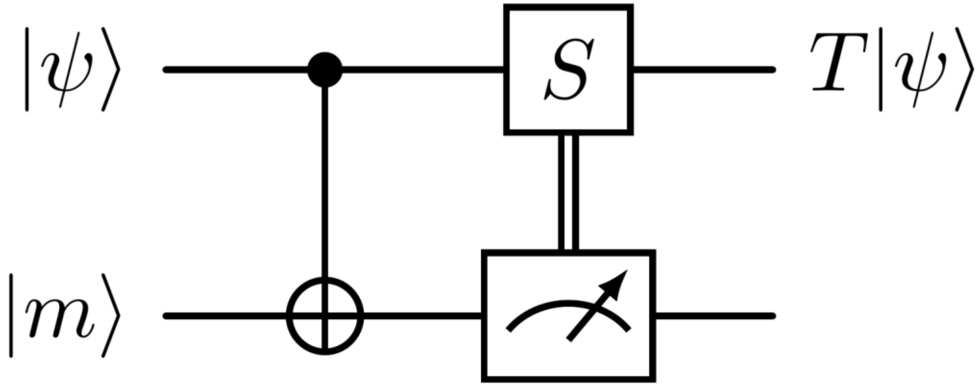
Single step execution

We need to choose our map and gate routes carefully to avoid serializing parallel gates

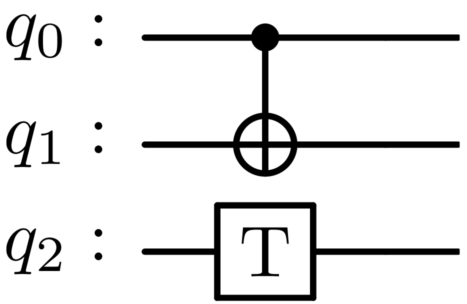
Magic state T gates

We cannot directly apply a T gate to encoded logical qubits

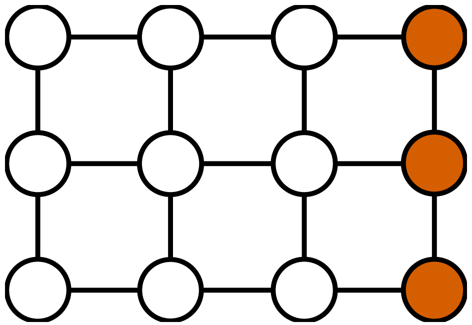
T gate is implemented via a CNOT gate with a “magic state”



Surface Code Mapping and Routing



Circuit

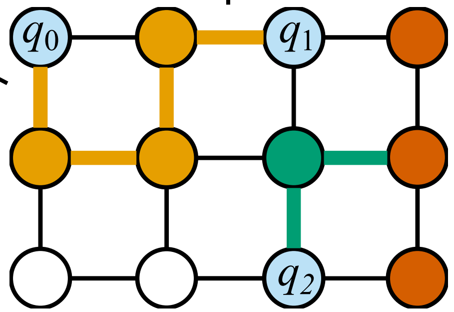


Fault-tolerant architecture

Magic state qubits for T gates

Must be vertical

Must be horizontal



Goal: Minimize # of steps

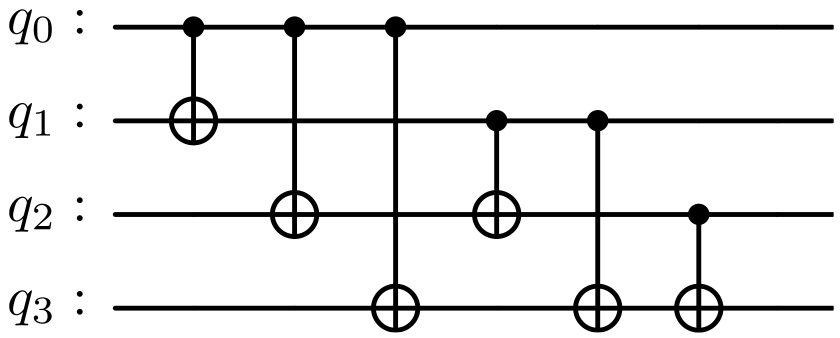
The DASCOT approach

Mapping takes dependency-aware view
minimizes conflicts between *parallel* gates

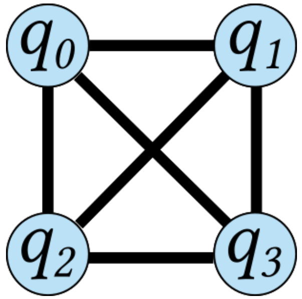
Routing searches for a strategy to maximize criticality
better than a fixed heuristic

Both powered by
simulated annealing

Mapping via interaction graphs

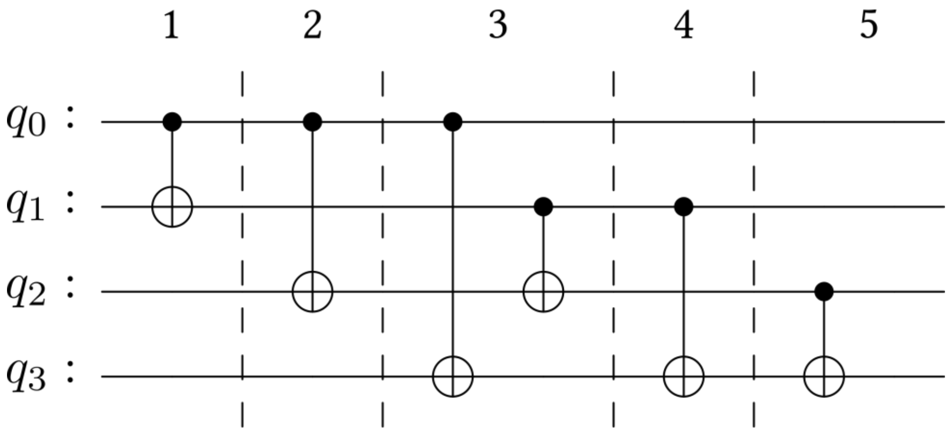


Interaction graph
(prior work)

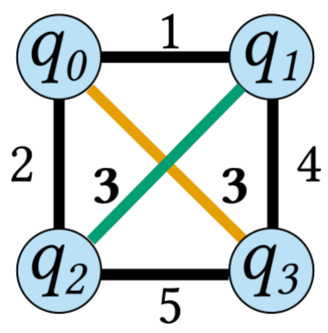


No useful information for choosing a mapping

Dependency-aware mapping



Layered interaction graph (DASCOT)

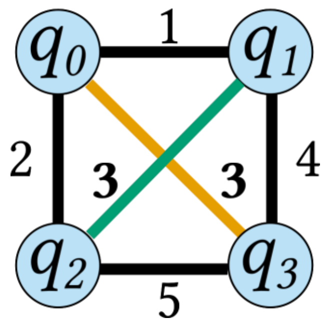


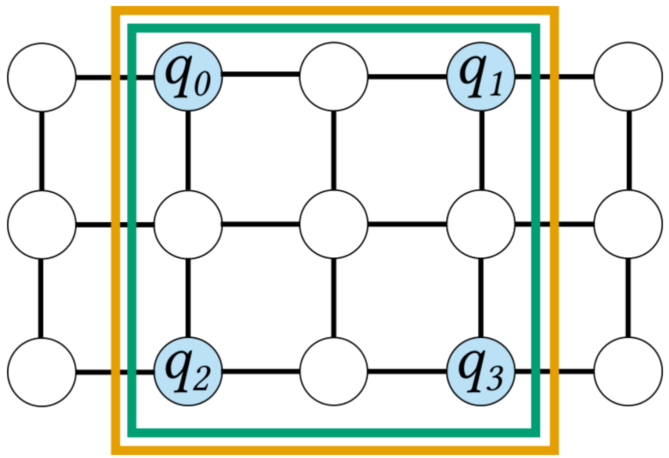
Captures the pair with the potential for blocking

Dependency-aware mapping

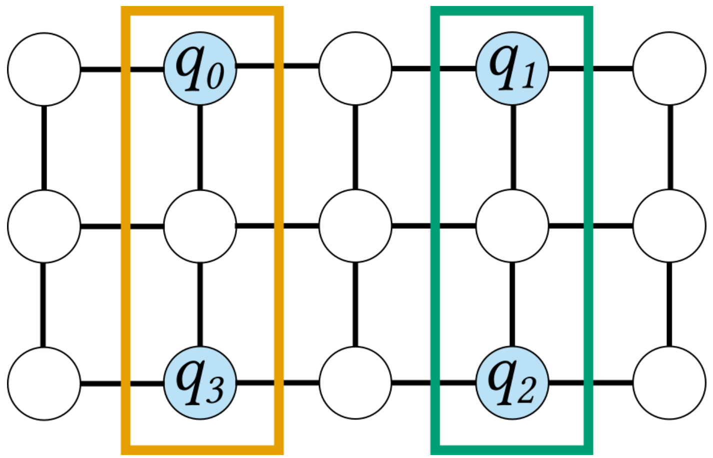
Conflict: Edge pair sharing label and overlapping bounding boxes

Search for a map that minimizes conflicts with simulated annealing





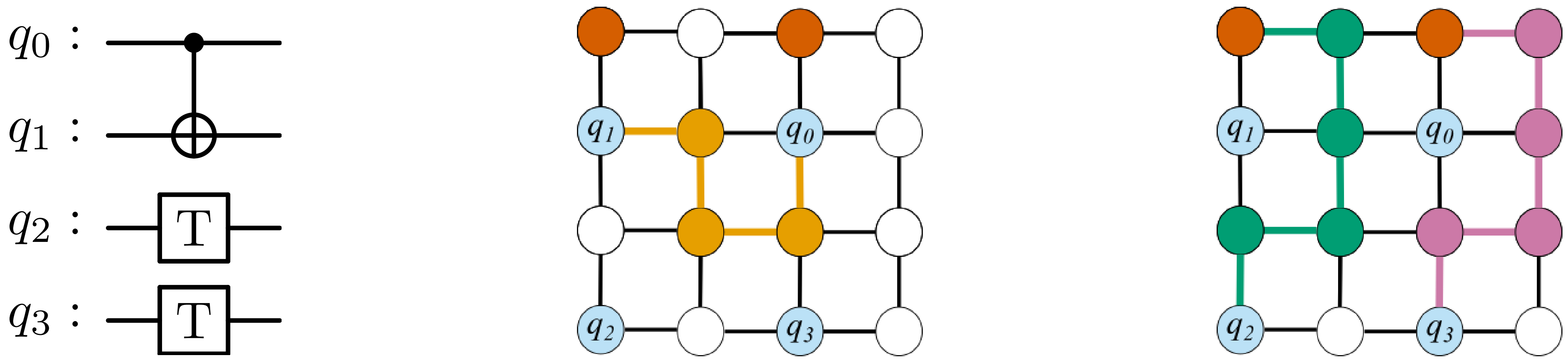
Conflicts: 1



Conflicts: 0

Searching the space of gate orders

Greedily routing the “best” gate is suboptimal



Search the space of routing orders with simulated annealing

Dependency-aware: weigh gates by **criticality**

Optimal SCMR

We have also encoded the SCMR problem into SAT

Increment the number of steps until satisfiable

Many of the same ideas as the NISQ case,
with new formulas to represent the no-crossing constraint

Feasible for circuits with tens of gates and qubits

What’s changed?

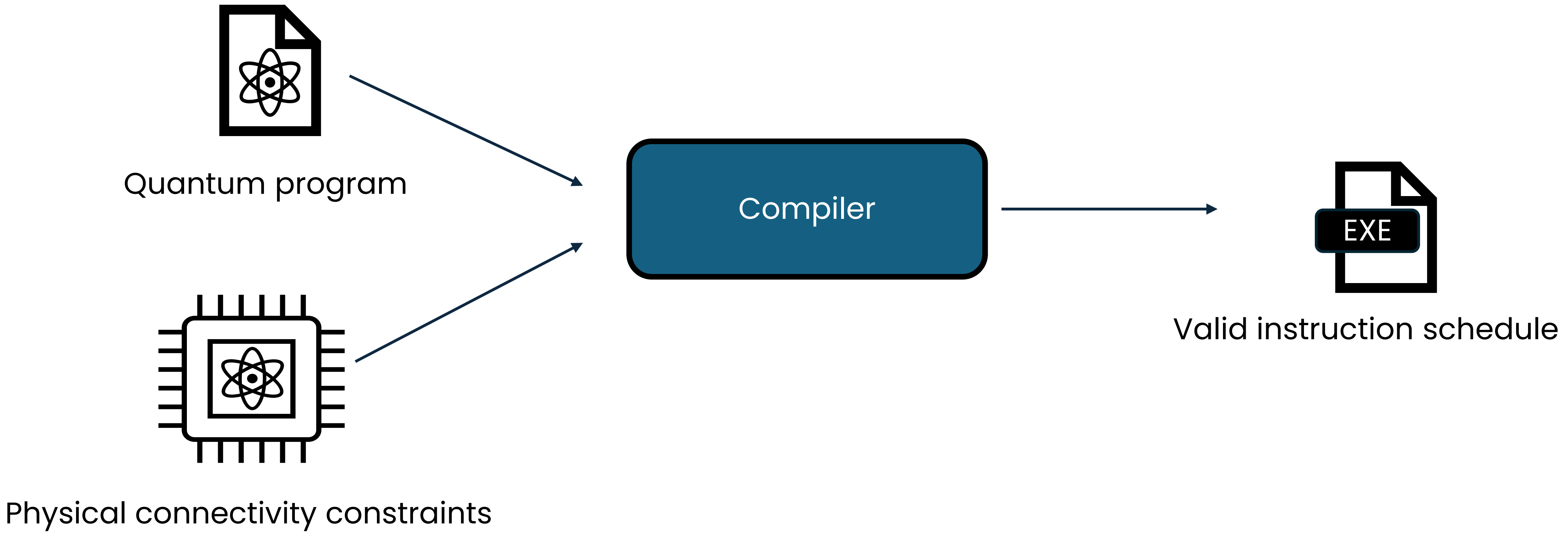
Mapping: distance doesn’t matter (directly), focus on conflicts

Routing: conflicts between gates mean that order matters

No added gates; execution time is primary objective

A Specification Language for Qubit Mapping and Routing

Returning to our Abstract Picture



Variations on a theme

We have seen two examples, but there are many more

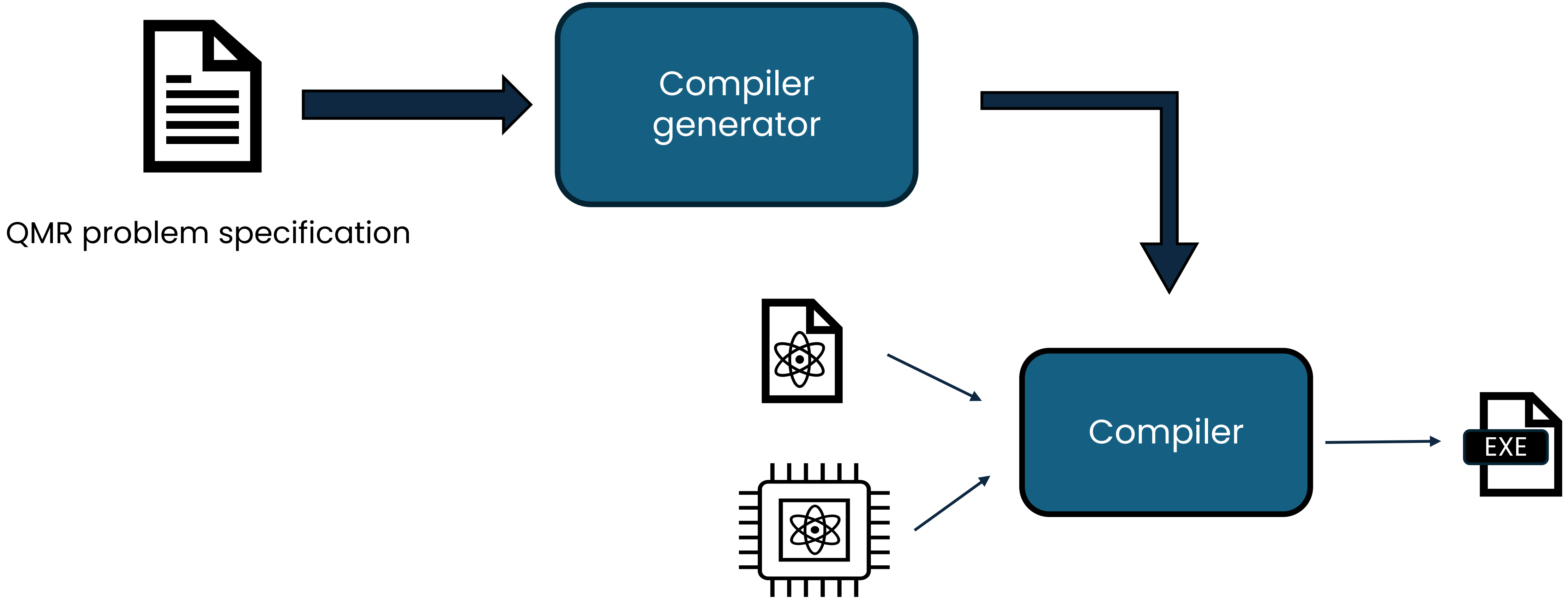
What if some qubits are more reliable that others? [Tannu19]

What if qubits can physically move during execution? [Wang24]

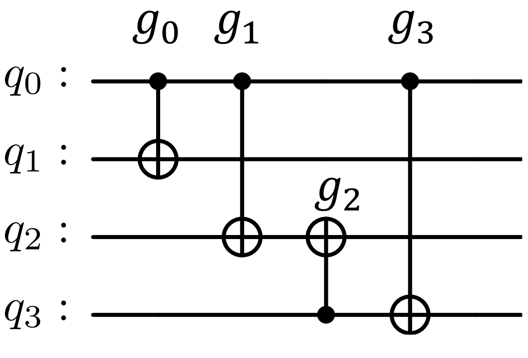
What if we can natively execute gates over >2 qubits? [Silva24]

What if…

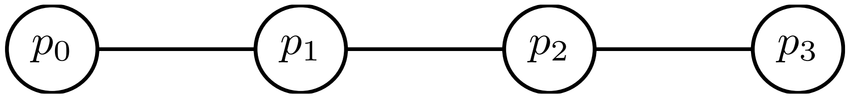
A “compiler generator” for QMR



Another look at a QMR solution



Circuit



Architecture

Step	Gates	Qubit Map
1	$g_0 \mapsto (p_0, p_1)$	$q_0 \mapsto p_1$ $q_1 \mapsto p_0$ $q_2 \mapsto p_2$ $q_3 \mapsto p_3$
2	$g_1 \mapsto (p_1, p_2)$	--
3	$g_2 \mapsto (p_3, p_2)$	--
4	$g_3 \mapsto (p_1, p_2)$	$q_0 \mapsto p_1$ $q_1 \mapsto p_0$ $\boldsymbol{q_2} \mapsto \boldsymbol{p_3}$ $\boldsymbol{q_3} \mapsto \boldsymbol{p_2}$

QMR generically

Shared between problems
Data types:
    Step, Gate, Arch, Map

Constraints:
    Steps respect dependency
    Maps are injective functions

Unique to each problem
How do I implement a gate?
How can I transition between steps?

Step	Gates	Qubit Map
1	$g_0 \mapsto (p_0, p_1)$	$q_0 \mapsto p_1$ $q_1 \mapsto p_0$ $q_2 \mapsto p_2$ $q_3 \mapsto p_3$
2	$g_1 \mapsto (p_1, p_2)$	--
3	$g_2 \mapsto (p_3, p_2)$	--
4	$g_3 \mapsto (p_1, p_2)$	$q_0 \mapsto p_1$ $q_1 \mapsto p_0$ $\boldsymbol{q_2 \mapsto p_3}$ $\boldsymbol{q_3 \mapsto p_2}$

```
GateRealization[
    name = ‘PhysicalCnot'
    data = (u : Location, v : Location)
    realize_gate = if Arch.contains_edge((Step.map[Gate.qubits[0]],Step.map[Gate.qubits[1]]))
            then Some(GateRealization{u=Step.map[Gate.qubits[0]],v=Step.map[Gate.qubits[1]]})
            else None
]

Transition[
    name = 'Swap'
    data = (edge : (Location,Location))
    get_transitions = (map(|x| -> Transition{edge = x}, Arch.edges()))
                      .push(Transition{edge = (Location(0),Location(0))})
    apply = value_swap(Transition.edge.0, Transition.edge.1)
    cost = if (Transition.edge)==(Location(0), Location(0)) then 0.0 else 1.0
]
```

The full spec in our language