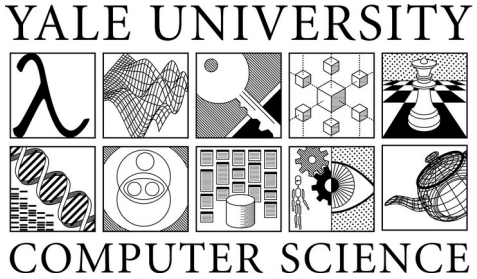
PRISM: A Logical-Physical Boundary Compiler for Dual-Rail Qubits

Evan McKinney
Kun Liu, Harshvardhan K Babla, Robert J Schoelkopf, Yongshan Ding

Monday Lunch Seminar (Oct 2026)



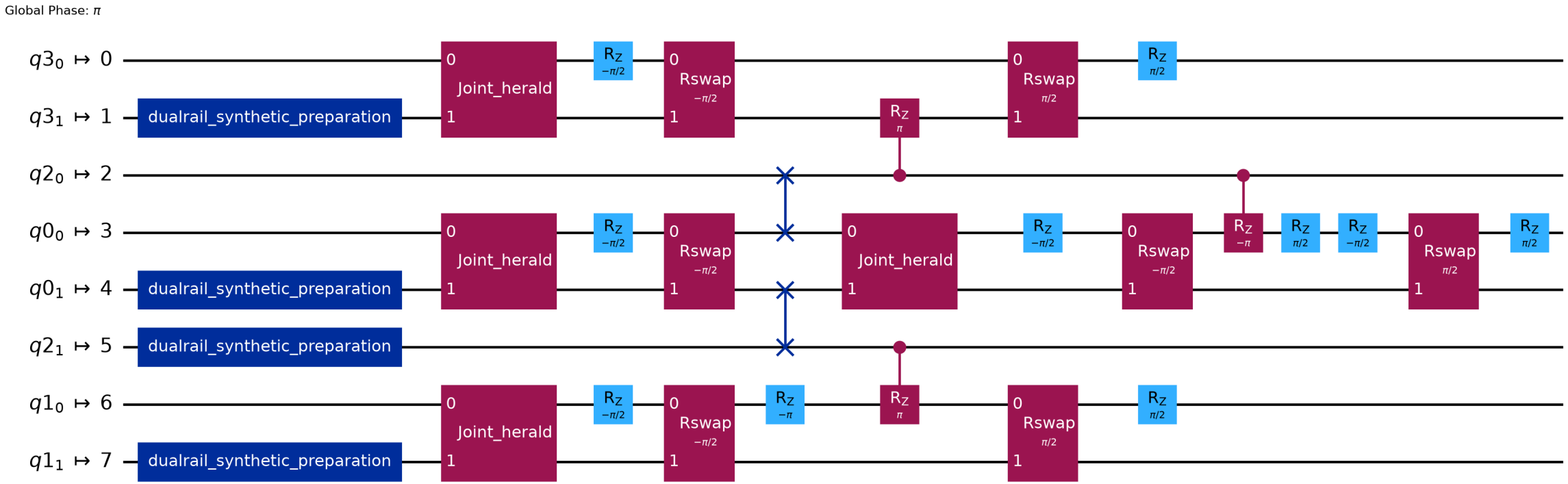




Dual-rail compilation in Qiskit?

- Given a quantum circuit how do I compile it (a) into a dual-rail encoding; (b) onto a device?

```
transpiled_qc = transpile(input_qc, basis_gates=["cx", "u3"])
transpiled_qc.draw("mpl")
```



A typical quantum compiler stack

- **Compiler**: lowers the program **down** one layer
  - Rewrite each gate in the next layer's gate set
- **Transpiler**: rewrites **within** one layer
  - Cancel, route, resynthesize

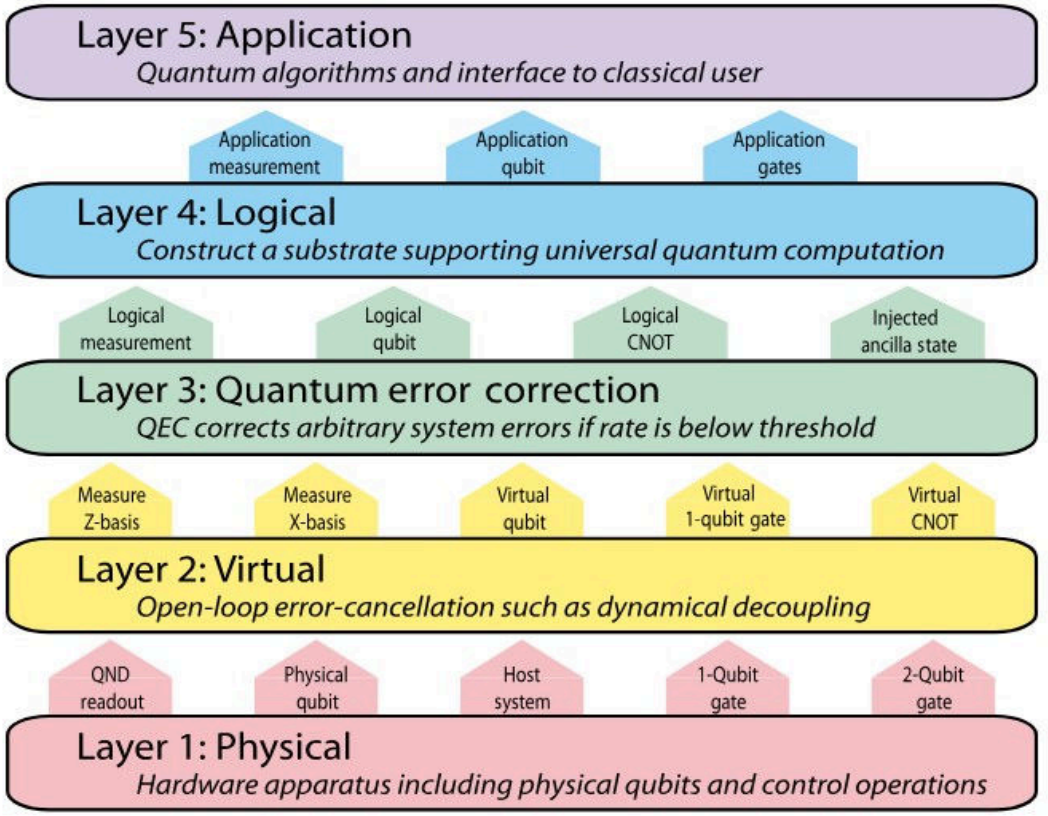
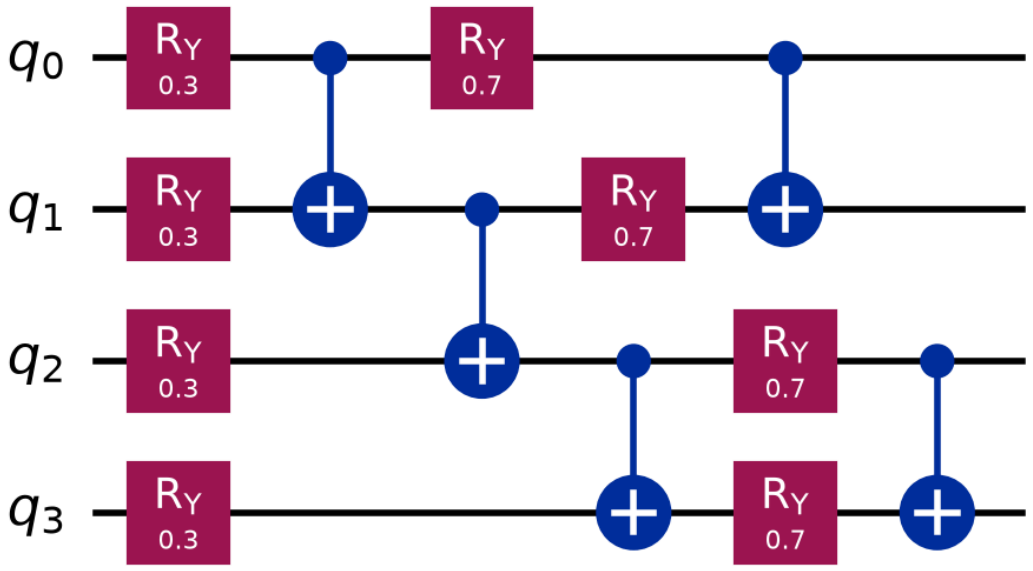
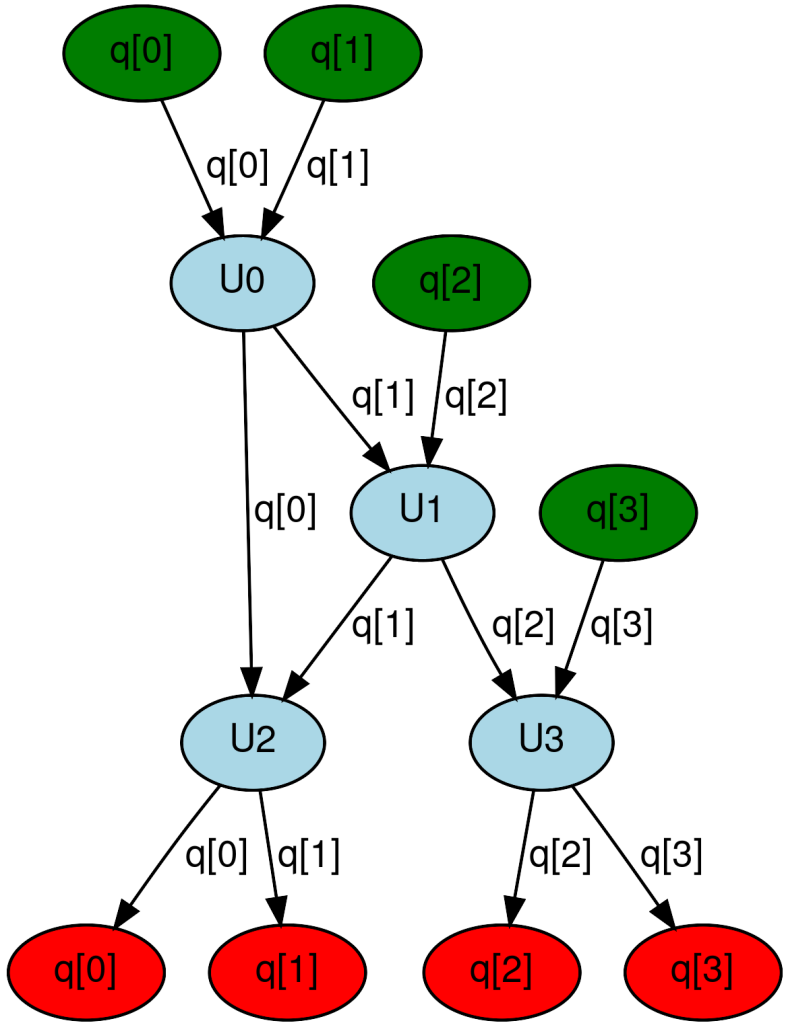


FIG. 1. Layered control stack that forms the framework of a quantum-computer architecture. Vertical arrows indicate services provided to a higher layer.

Jones, et al. **Physical Review X 2.3** (2012)

The compiler works on the gate dependency DAG





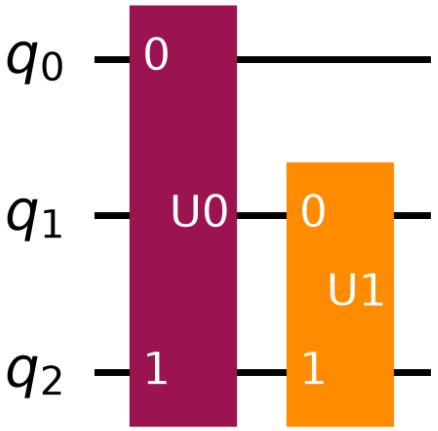


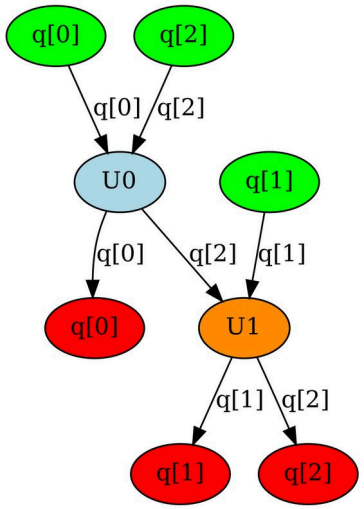
Passes rewrite the DAG, not what it computes

Stage	Pass
Init	HighLevelSynthesis, Unroll3qOrMore, ElidePermutations
Layout	TrivialLayout, DenseLayout, VF2Layout, SabreLayout, SetLayout, FullAncillaAllocation, EnlargeWithAncilla, ApplyLayout
Routing	SabreSwap, BasicSwap, LookaheadSwap
Translation	BasisTranslator, UnitarySynthesis, Decompose, GateDirection
Optimization	Optimize1qGatesDecomposition, InverseCancellation, CommutativeCancellation, Collect2qBlocks, ConsolidateBlocks, RemoveDiagonalGatesBeforeMeasure, OptimizeSwapBeforeMeasure, RemoveResetInZeroState
Scheduling	ALAPScheduleAnalysis, ASAPScheduleAnalysis, TimeUnitConversion, PadDelay, PadDynamicalDecoupling
Analysis	Depth, Size, Width, CountOps, CheckMap, GatesInBasis, DAGLongestPath

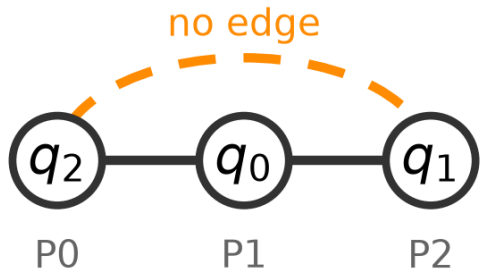
Javadi-Abhari, et al. **arXiv** (2024)

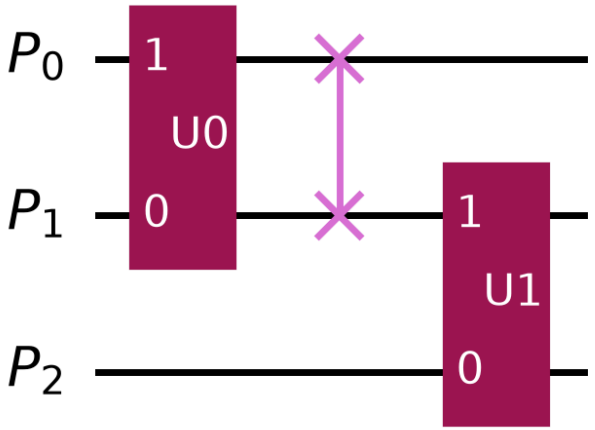
SABRE routing loop

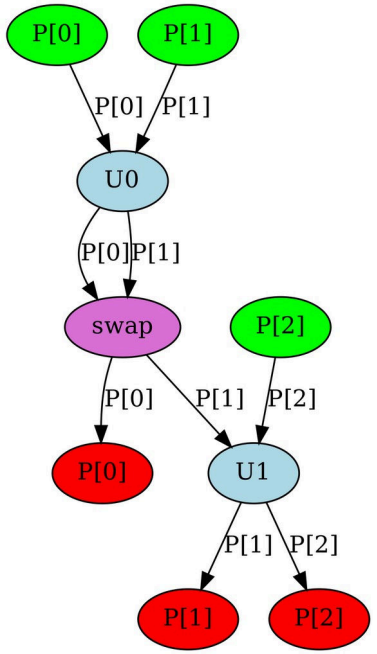




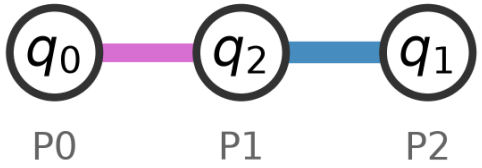
U1 on (q_1, q_2): stalled







SWAP on P0–P1, then U1 is adjacent



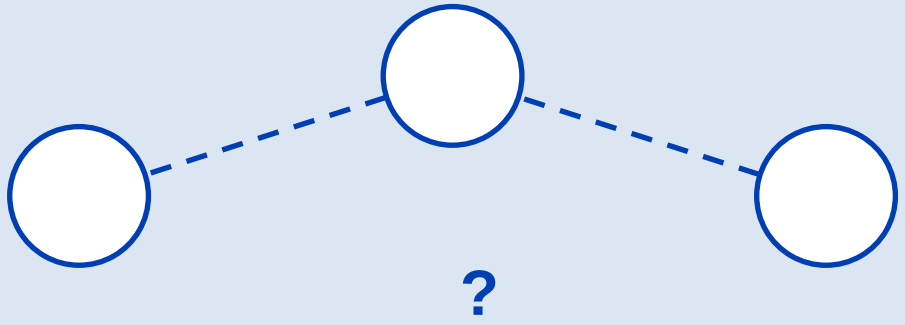
Li, et al. **ASPLOS** (2019)

Fault-tolerant encoded qubits: stack two transpilers?

Logical transpiler

routes logical qubits

…on which graph?

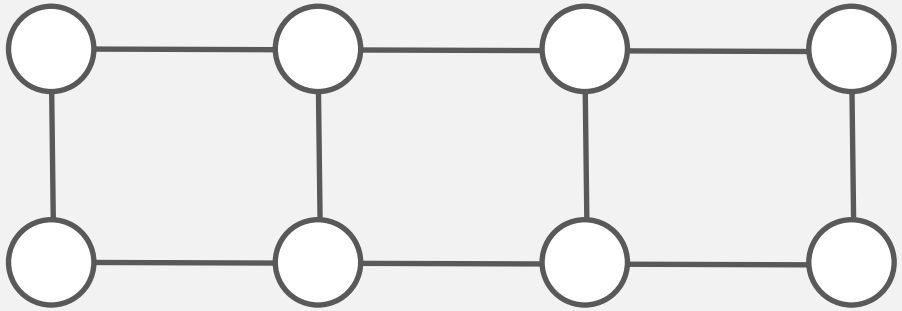


lower: replace each logical qubit by its encoding

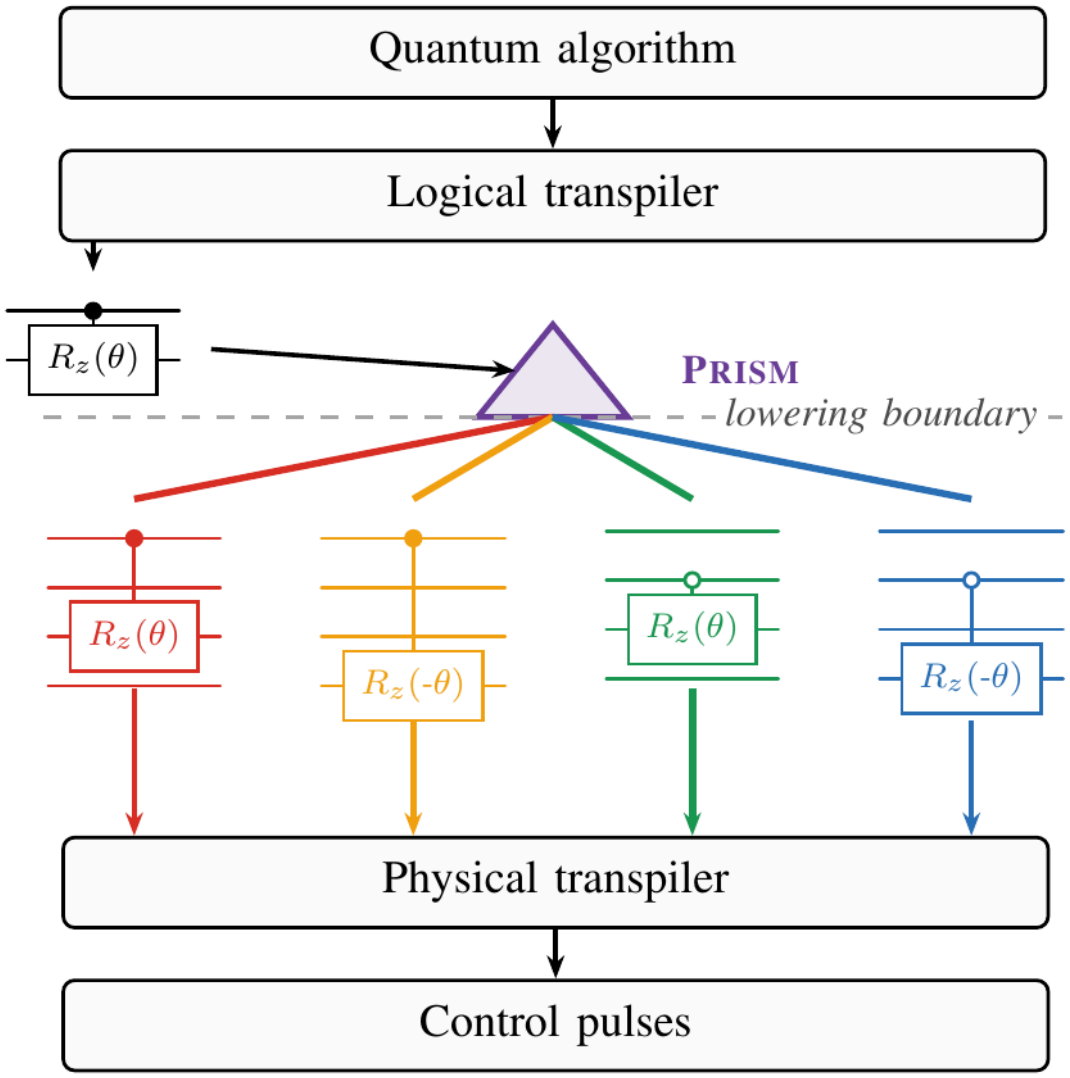
Physical transpiler

routes physical qubits

knows nothing about the code



PRISM as a compiler lowering stage



what is **logical connectivity** when one gate has several physical circuits?

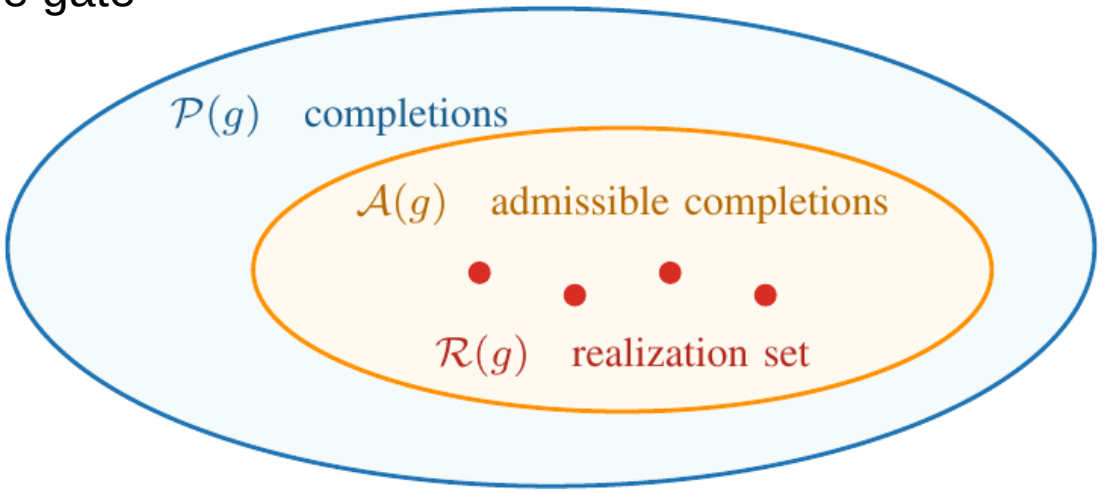
where do those **circuits** come from, for any native gate set?

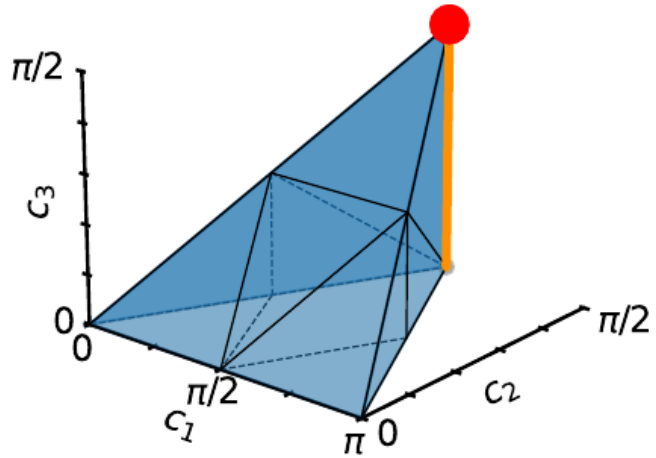
when should the compiler commit to one?

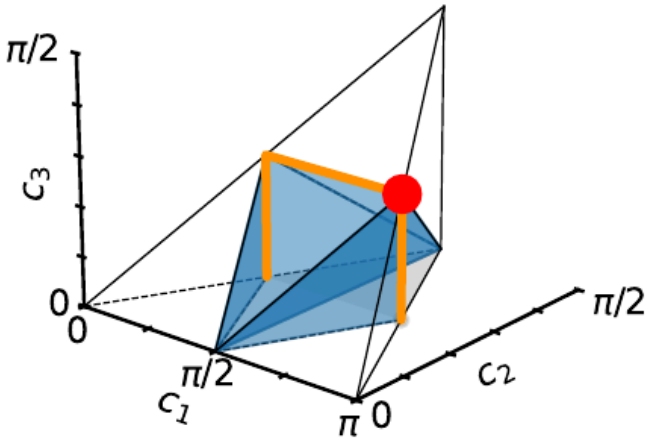
Fault model and physical ISA define realization set

- Code-preserving gate **M(A, B)**
- Logical action fixed by B; **any A** gives the same gate
- Loss model: A must not send **|00⟩ → |11⟩**

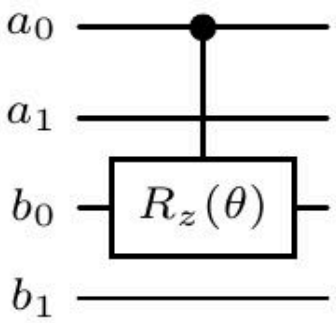
	|00⟩	|01⟩	|10⟩	|11⟩
|00⟩	A	0	0	A
|01⟩	0	**B**	**B**	0
|10⟩	0	**B**	**B**	0
|11⟩	A	0	0	A



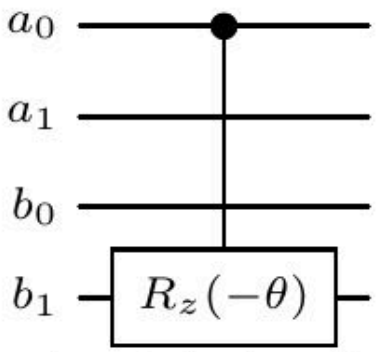




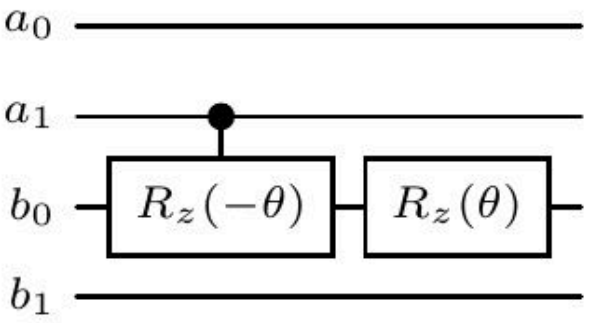
Four physical realizations of one logical CRz



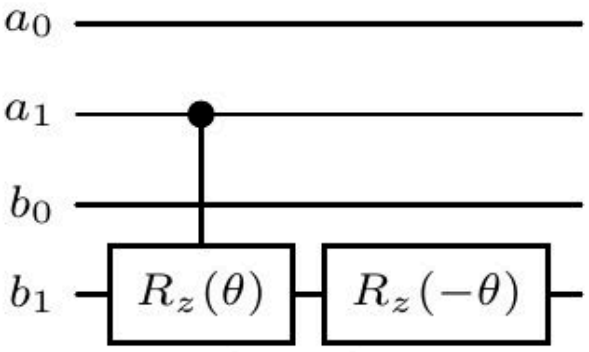
(a) • (a_0, b_0)



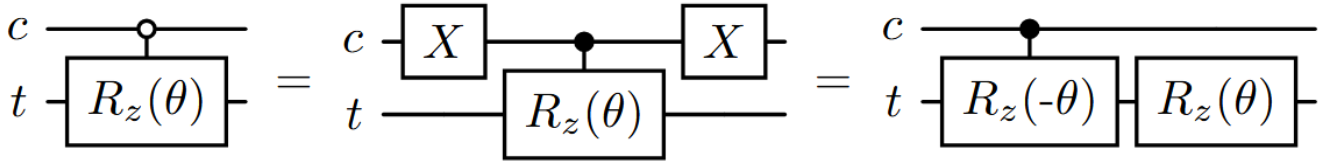
(b) • (a_0, b_1), sign flipped



(c) • (a_1, b_0), anti-control



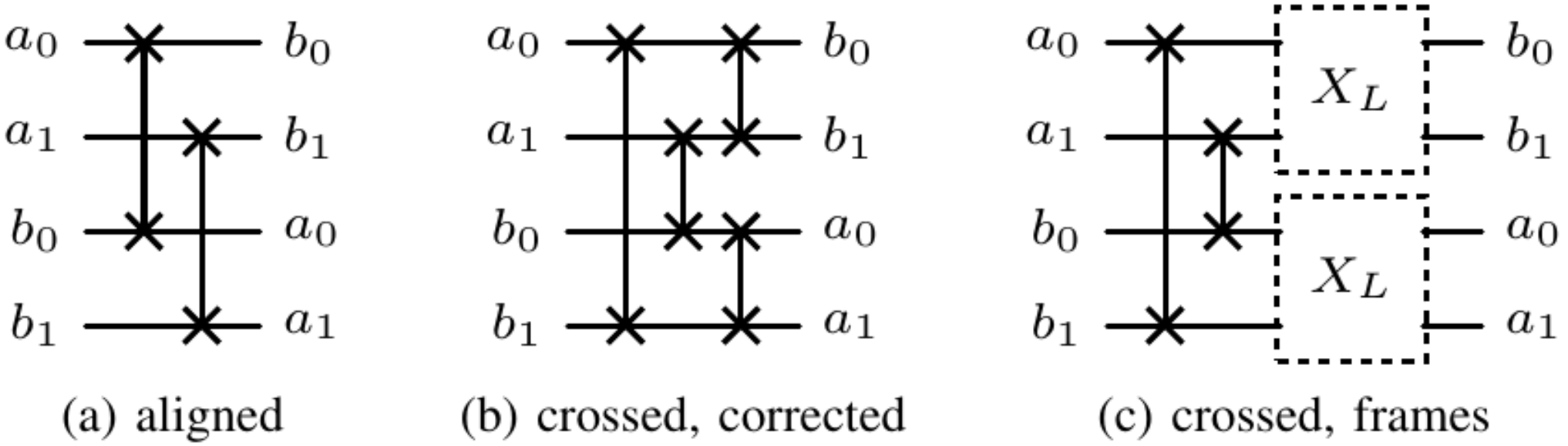
(d) • (a_1, b_1), both



- One SWS pulse between **any** rail of A and **any** rail of B
- Four circuits in the **realization set** R(CRz)
- Pairing fixes only the **sign** of θ and a local Rz

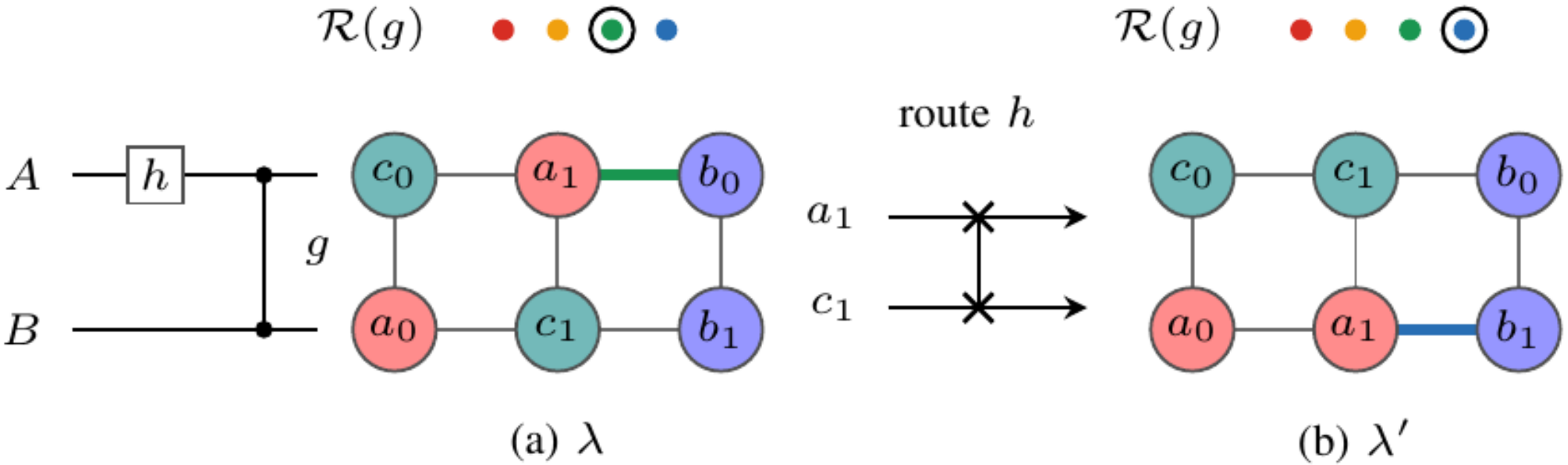
Two physical realizations of one logical SWAP

- Aligned logical SWAP (a): **2 exchanges**; crossed (b): 2 more to restore roles, or software frames (c)
- Routing on a fixed logical graph must move **both rails** even when one pending realization needs one



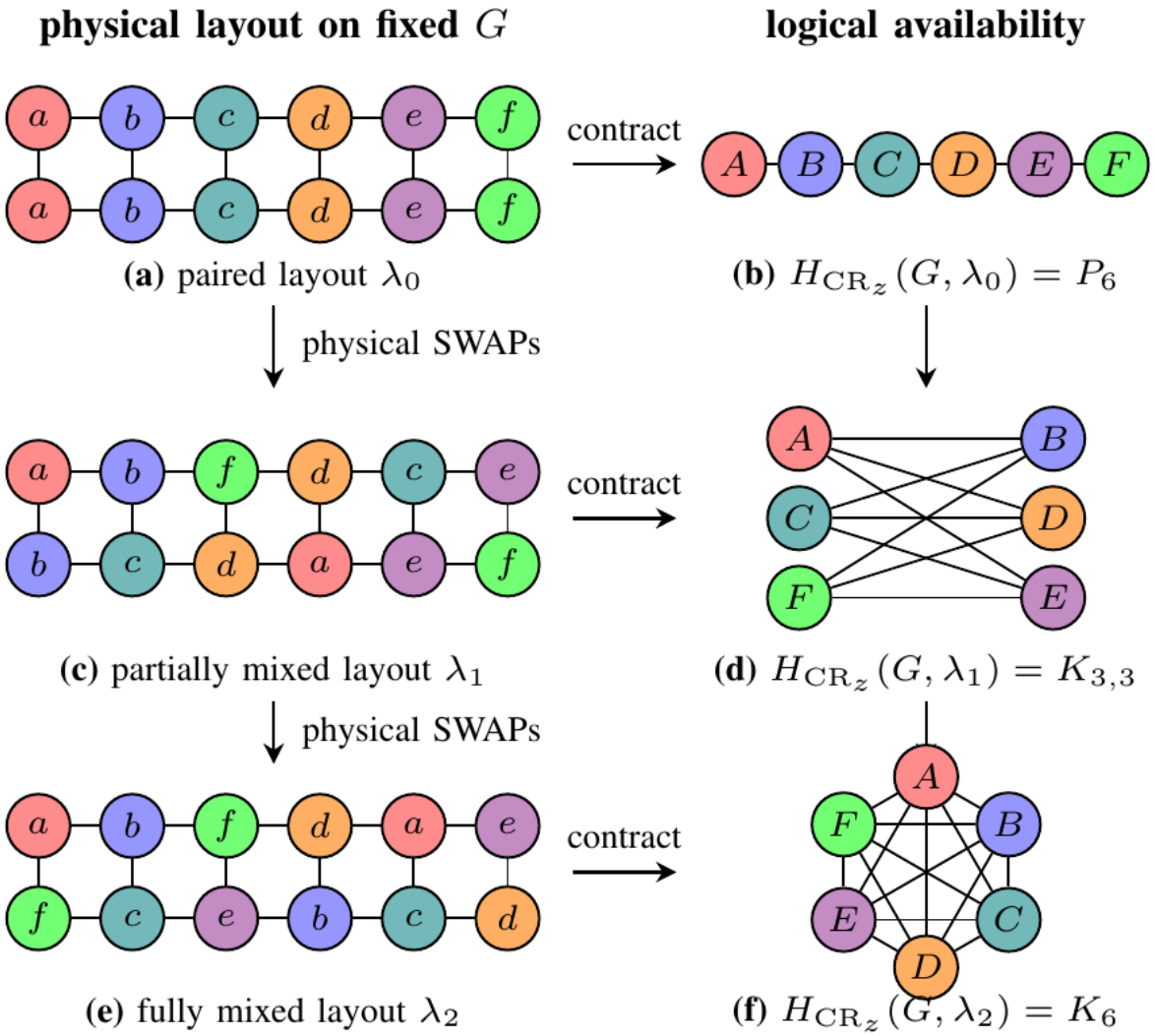
(a) aligned (b) crossed, corrected (c) crossed, frames

Routing changes which realization is available



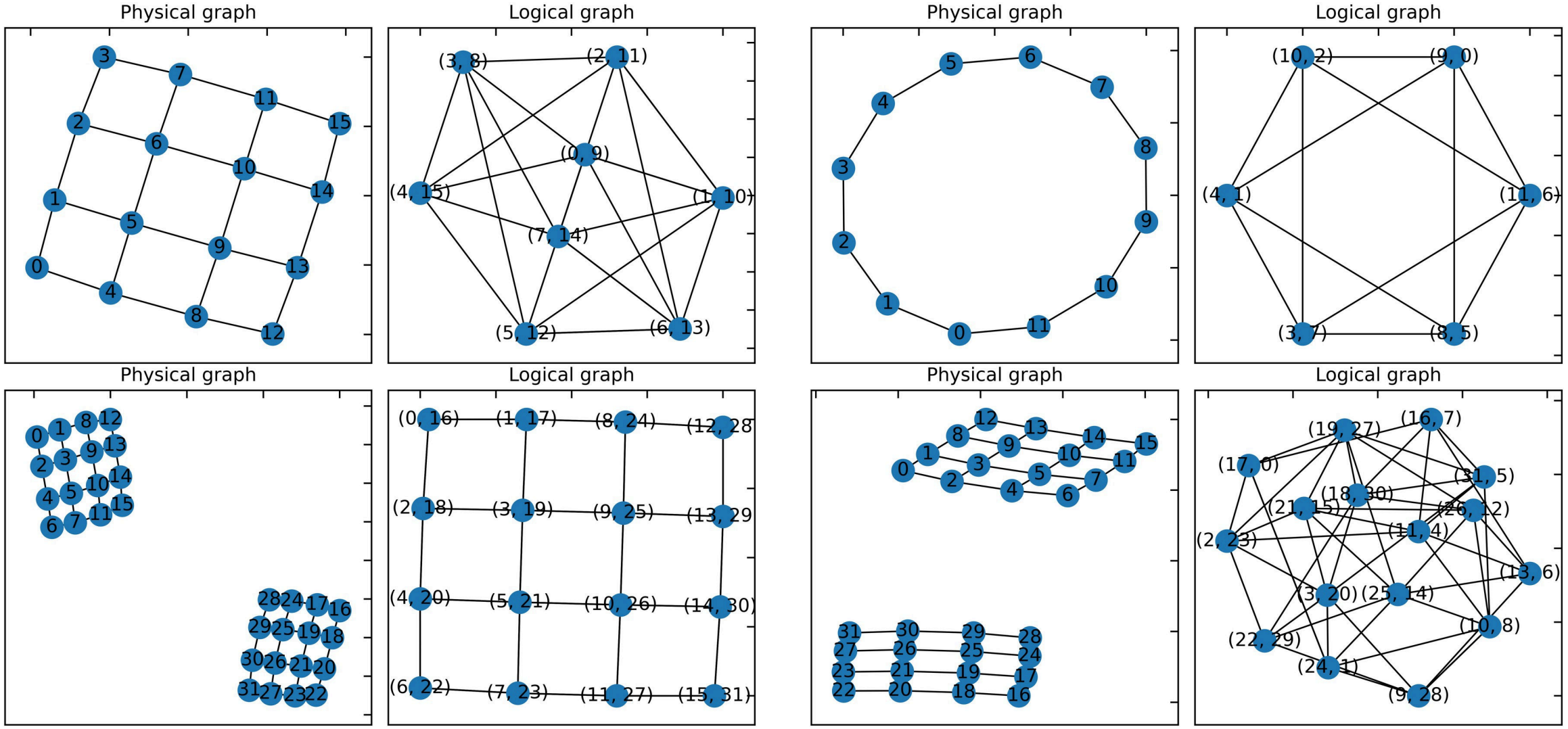
- Under λ: only the **green** pairing (a_1, b_0) is adjacent
- Route the earlier gate h: $a_1 \leftrightarrow c_1$; now only **blue** (a_1, b_1) is adjacent
- Commit to green under λ → pay a SWAP under λ′ that blue never needed

The logical graph is the device graph modulo rail pairs



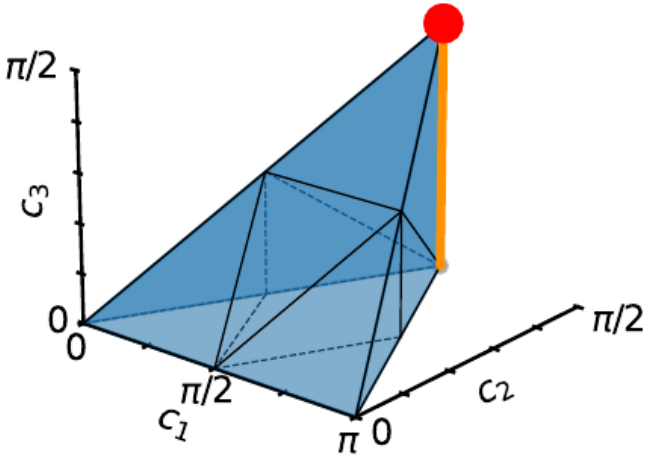
- Contract each rail pair to **one logical vertex**
- Edge ⇔ a CRz circuit runs **without routing**
- Paired: P_6; one exchange: $K_{3,3}$; two: **K_6**
- Physical SWAP → **new quotient graph**: compiler state, not architecture
- Gate → rail pairing **not unique**
- Breaks the transpiler API: **one coupling map**, one decomposition per gate

Delocalized dual-rail quotient connectivies

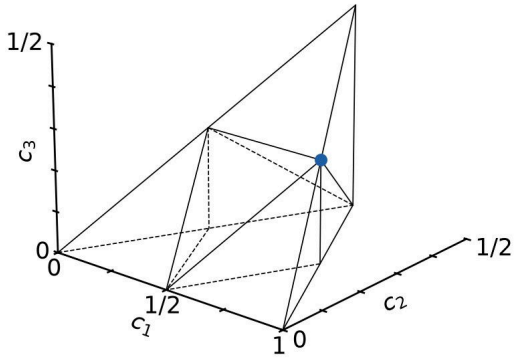


Reachability of sqrtSWAP sentences

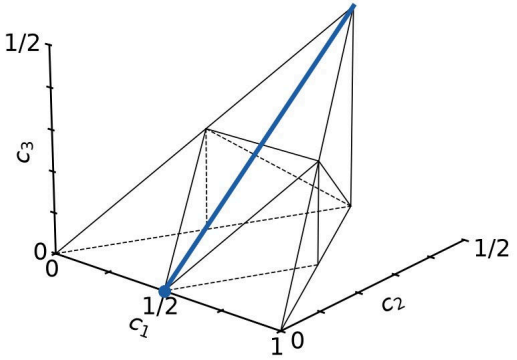
How do we pick the realization in the admissible set?



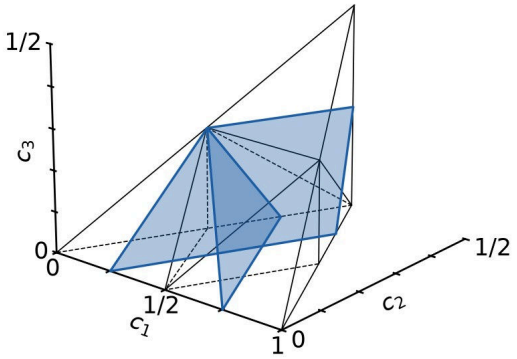
S | cost 0.5
new 0.00% | cumulative 0.00%



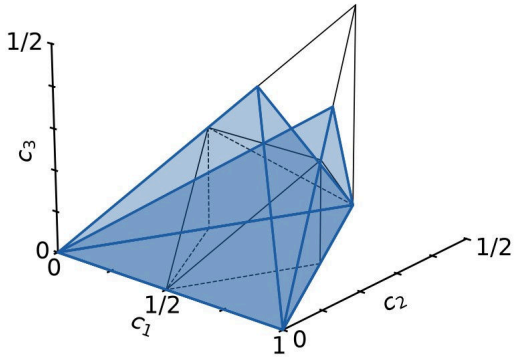
SS | cost 1
new 0.00% | cumulative 0.00%



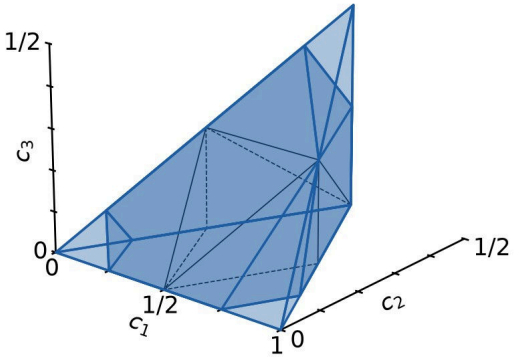
SSS | cost 1.5
new 0.00% | cumulative 0.00%



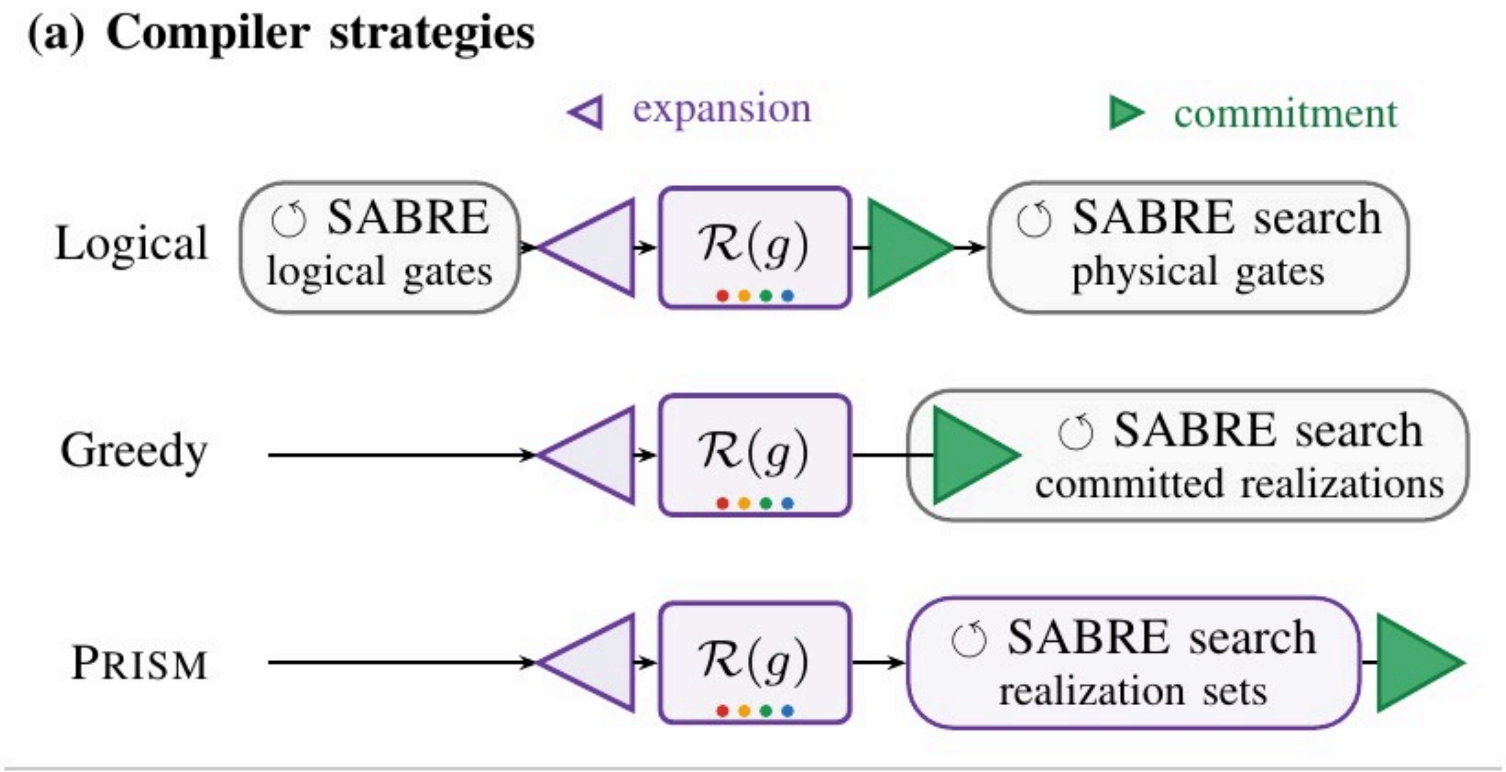
SSSS | cost 2
new 95.99% | cumulative 95.99%



SSSSS | cost 2.5
new 4.01% | cumulative 100.00%

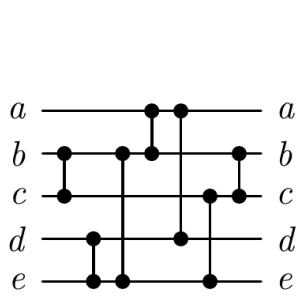


Three places to commit to a realization

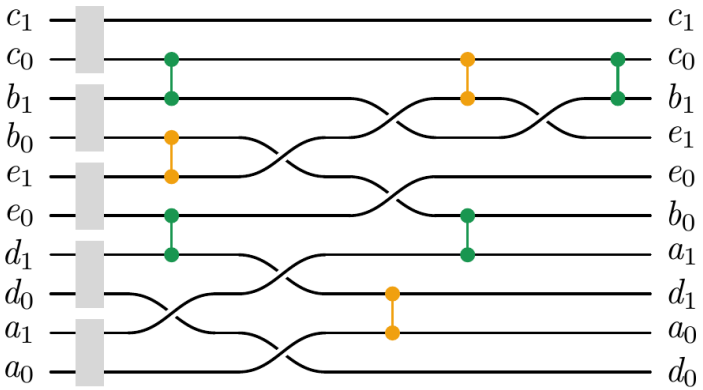


- **Logical-SABRE**: at code expansion; routes whole rail pairs
- **Greedy**: at router entry; nearest pairing under the input layout
- **PRISM**: when the gate executes; the set crosses the boundary

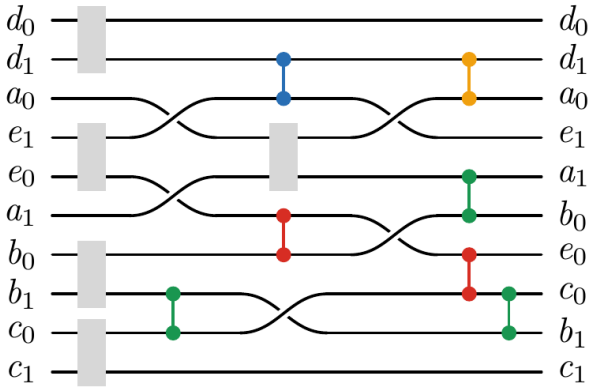
Contrasting commitment strategies



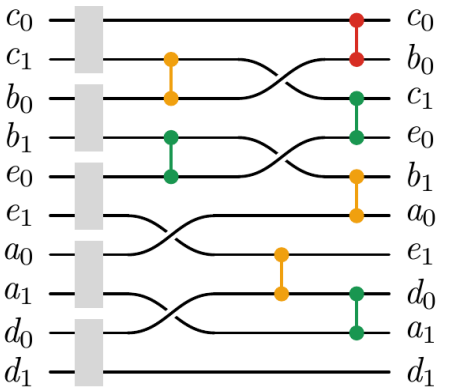
(a) logical input



(b) Logical-SABRE (7 SWAPs, depth 6)



(c) Greedy (5 SWAPs, depth 5)



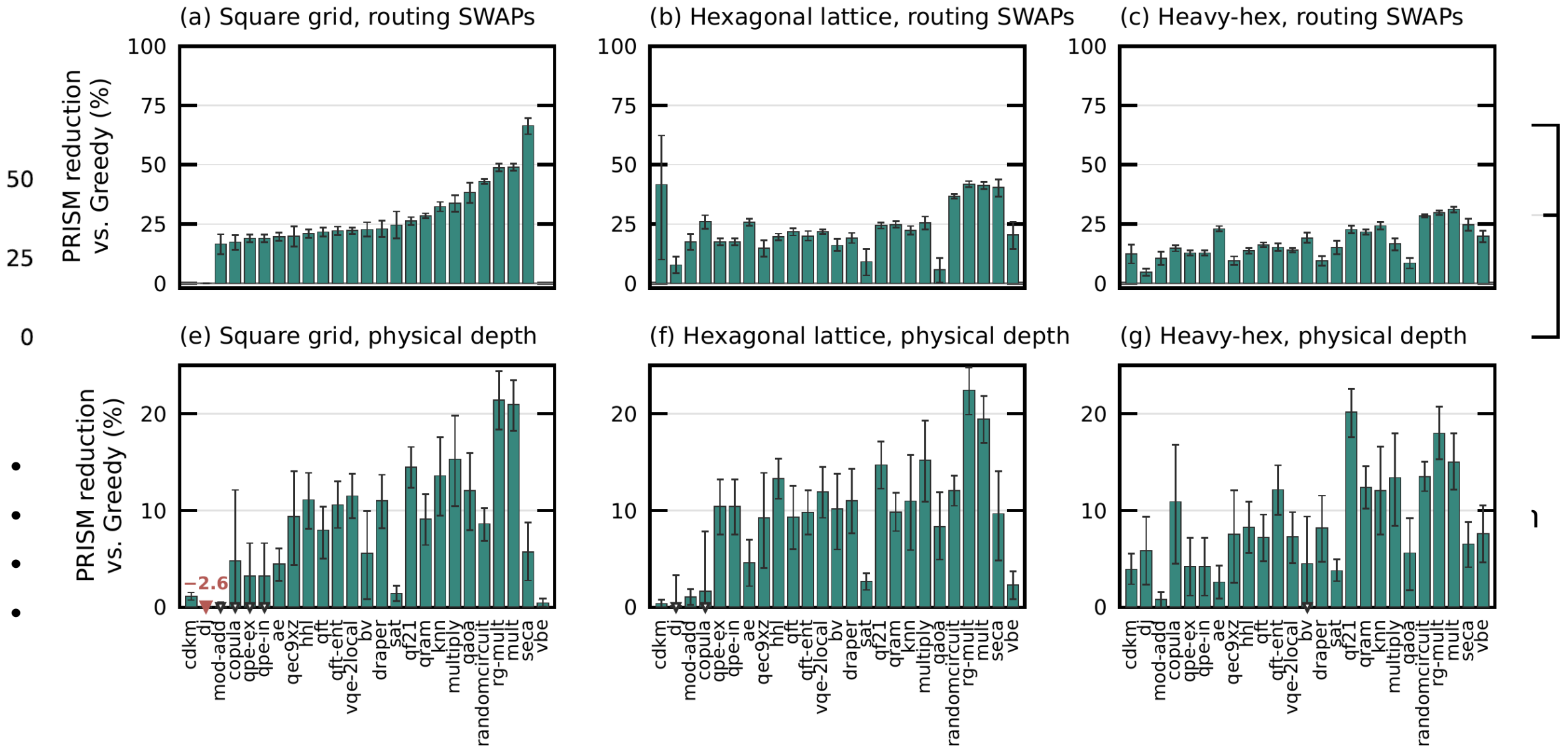
(d) PRISM (4 SWAPs, depth 3)

- **Logical-SABRE**: 7 SWAPs
- Commit at expansion
- Routes whole rail pairs

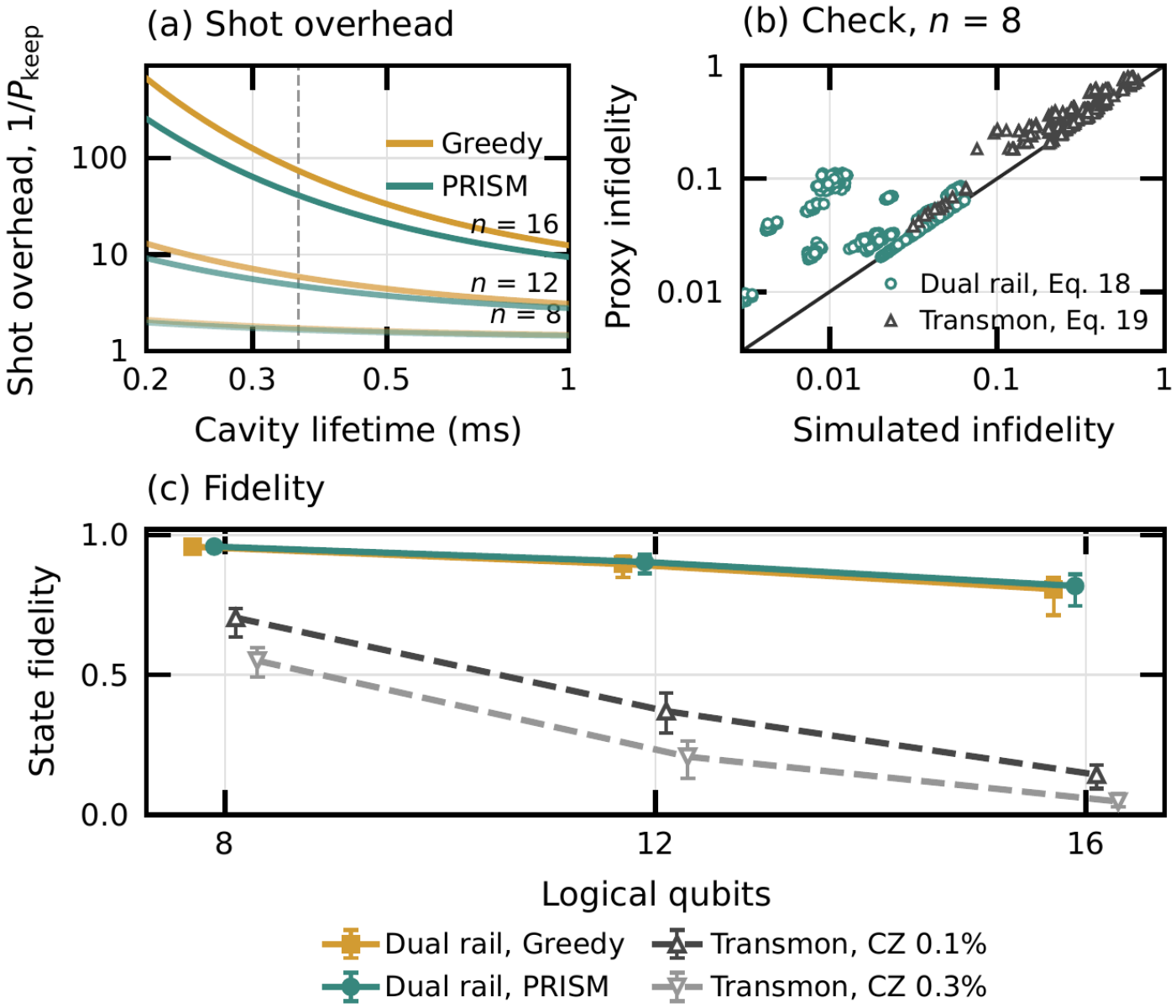
- **Greedy**: 5 SWAPs
- Commit at router entry
- Nearest pairing, then fixed

- **PRISM**: 4 SWAPs
- Commit when **executable**
- Keeps all four pairings

PRISM greatly reduces routing overhead



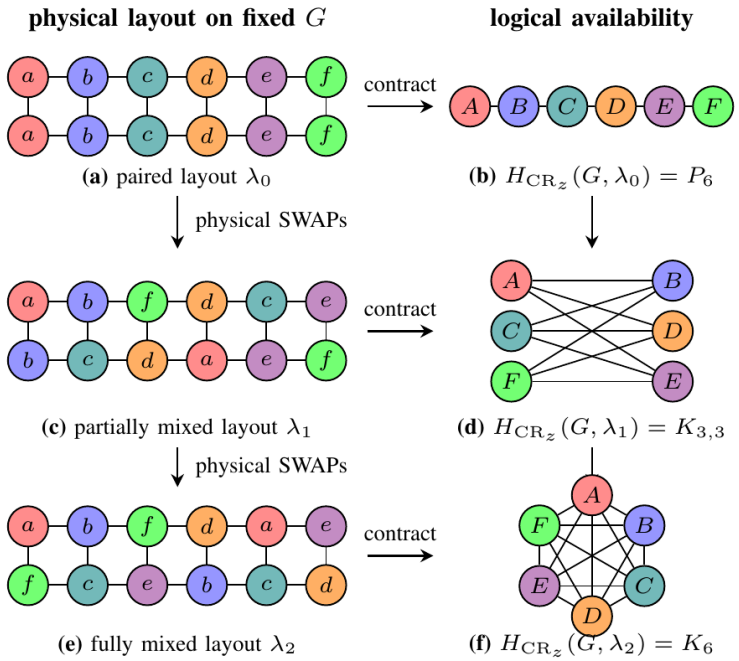
Shorter routes compound under erasure postselection



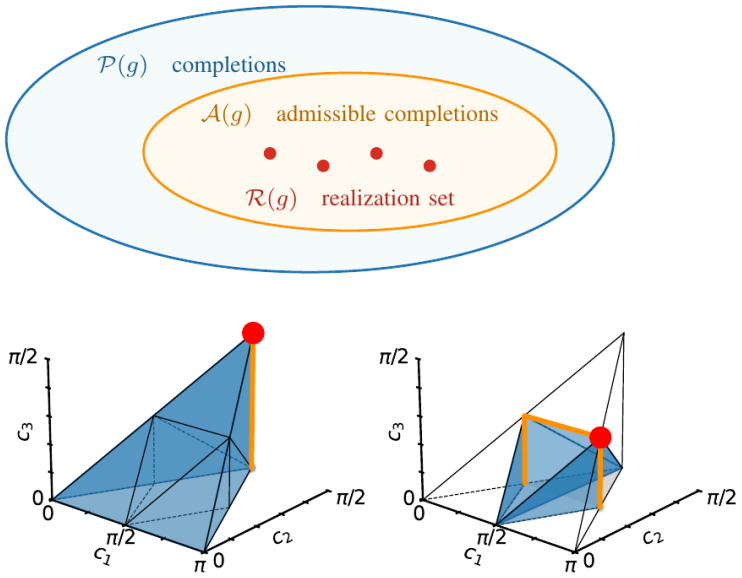
At 16 logical qubits:

- Duration **−15% / −14% / −16%**
- Shot overhead **−44%**
- Conditional infidelity **−7% / −9% / −10%**
- Fidelity, 8 → 16 qubits: **0.96 → 0.82** dual rail, 0.70 → 0.14 transmon

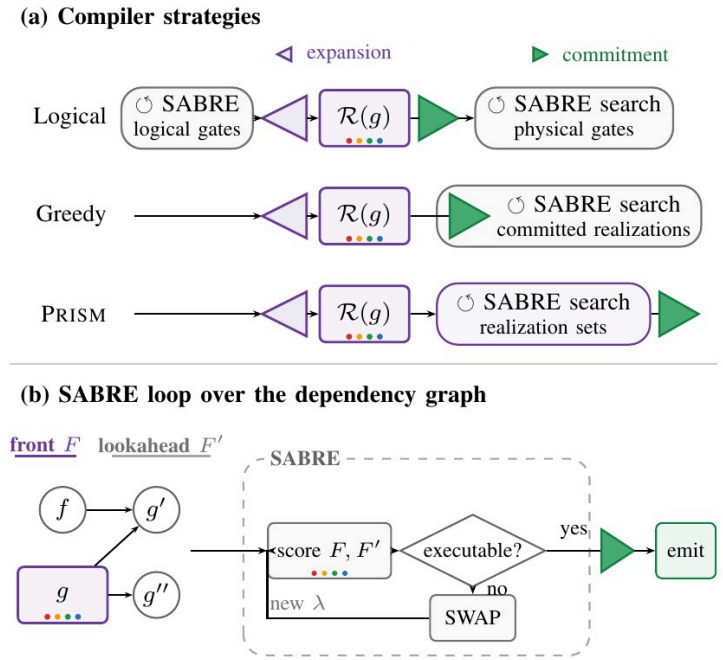
Conclusion



Compiler state: the device graph quotiented by the current rail pairs layout



A set of circuits per gate, synthesized by GULPS for any native gate set



While routing: −20% to −31% SWAPs vs. Greedy, −44% shot overhead