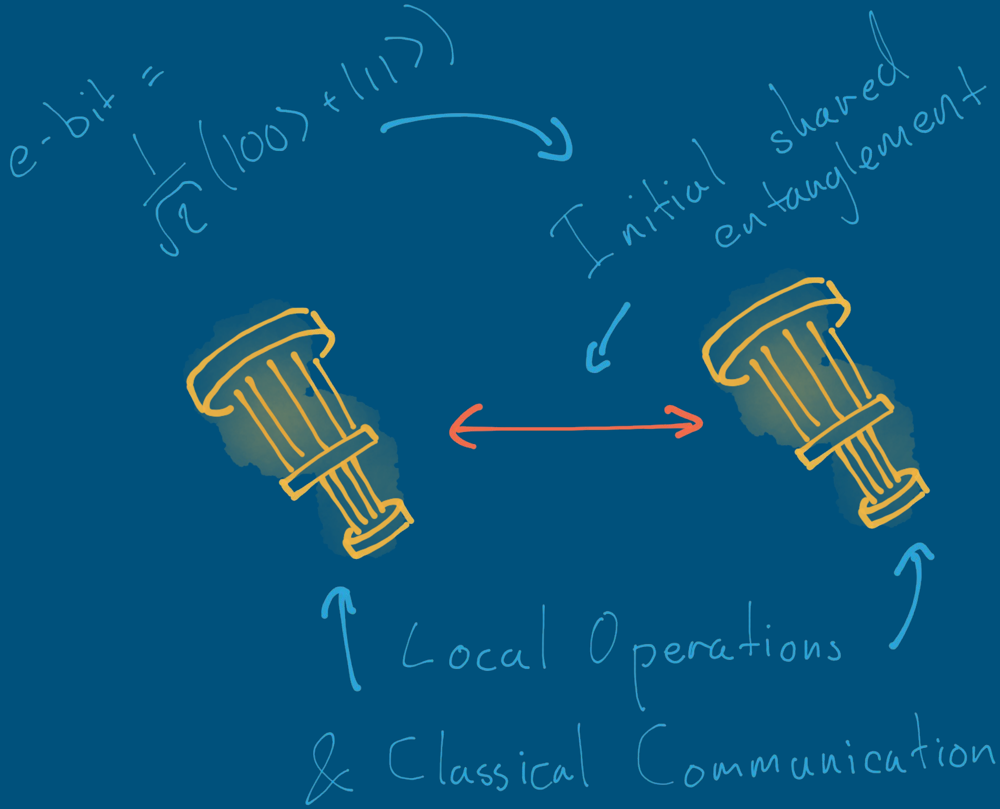
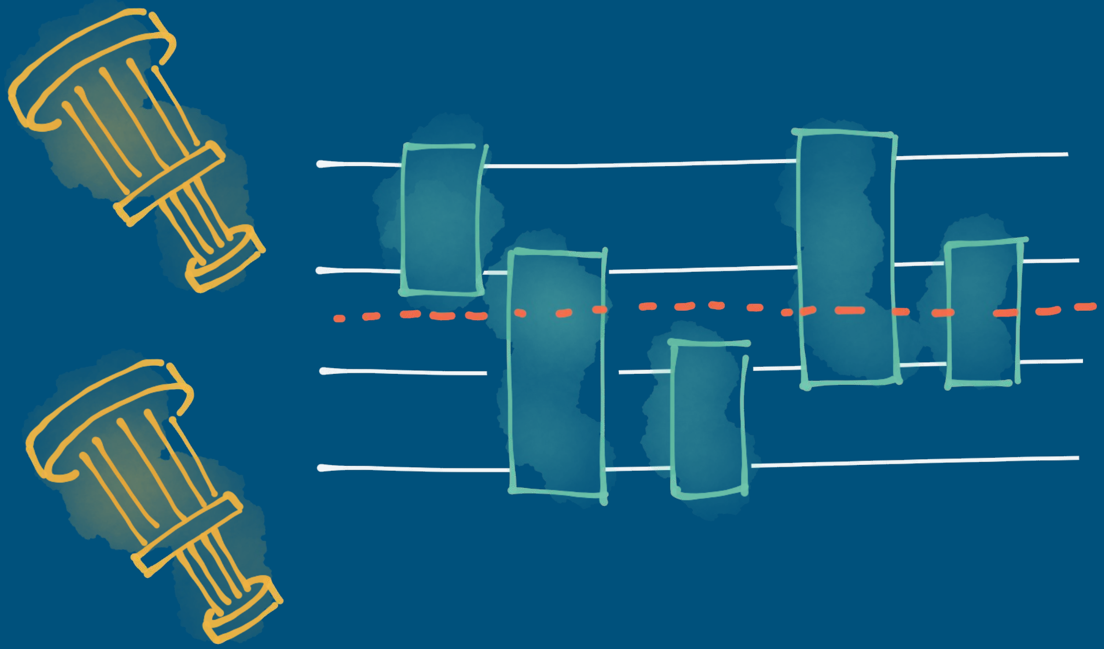
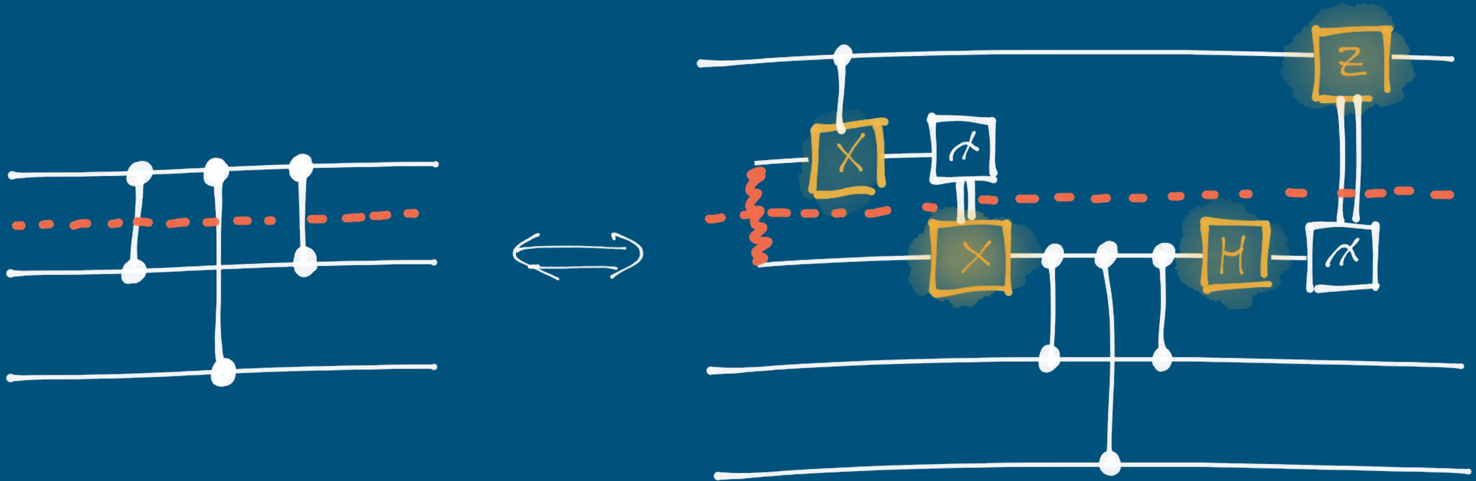
Distributing Circuits Over Heterogeneous, Modular Quantum Computing Networks

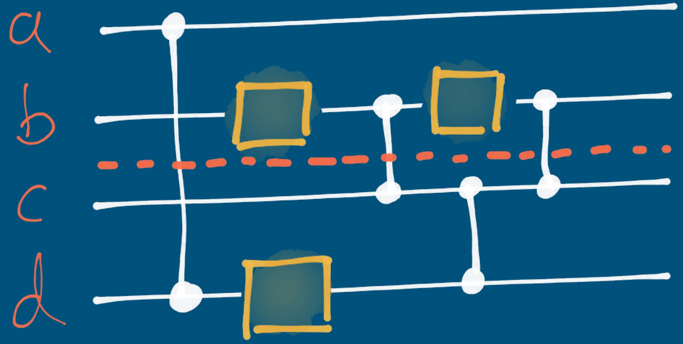
Pablo Andres-Martinez, Tim Forrer, **Dan Mills**, Jun-Yi Wu, Luciana Henaut, Kentaro Yamamoto, Mio Murao, Ross Duncan

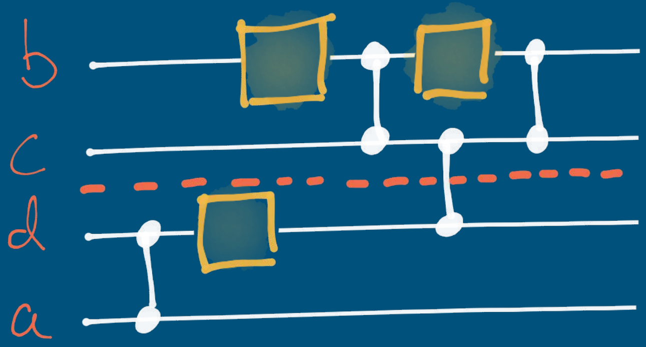

e-bit = $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$
Initial shared entanglement
Local Operations
& Classical Communication

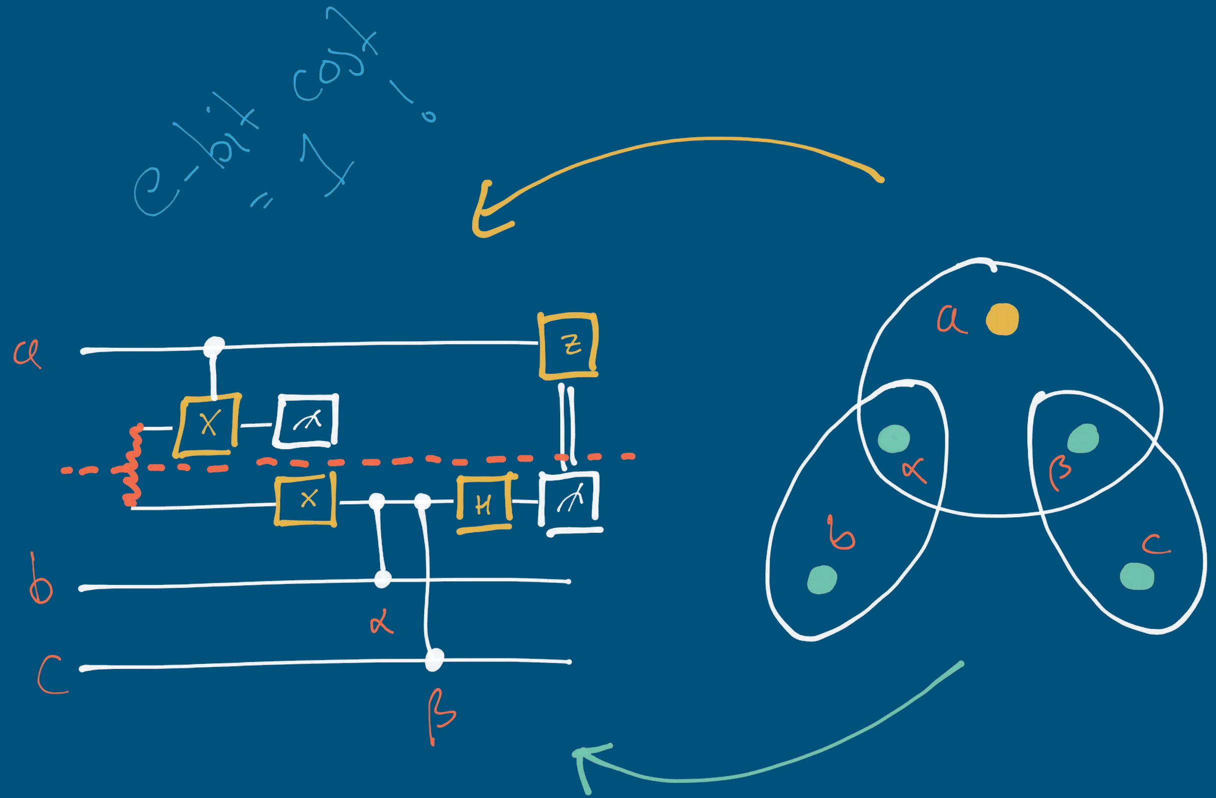




Optimal local implementation of non-local quantum gates


a
b
c
d

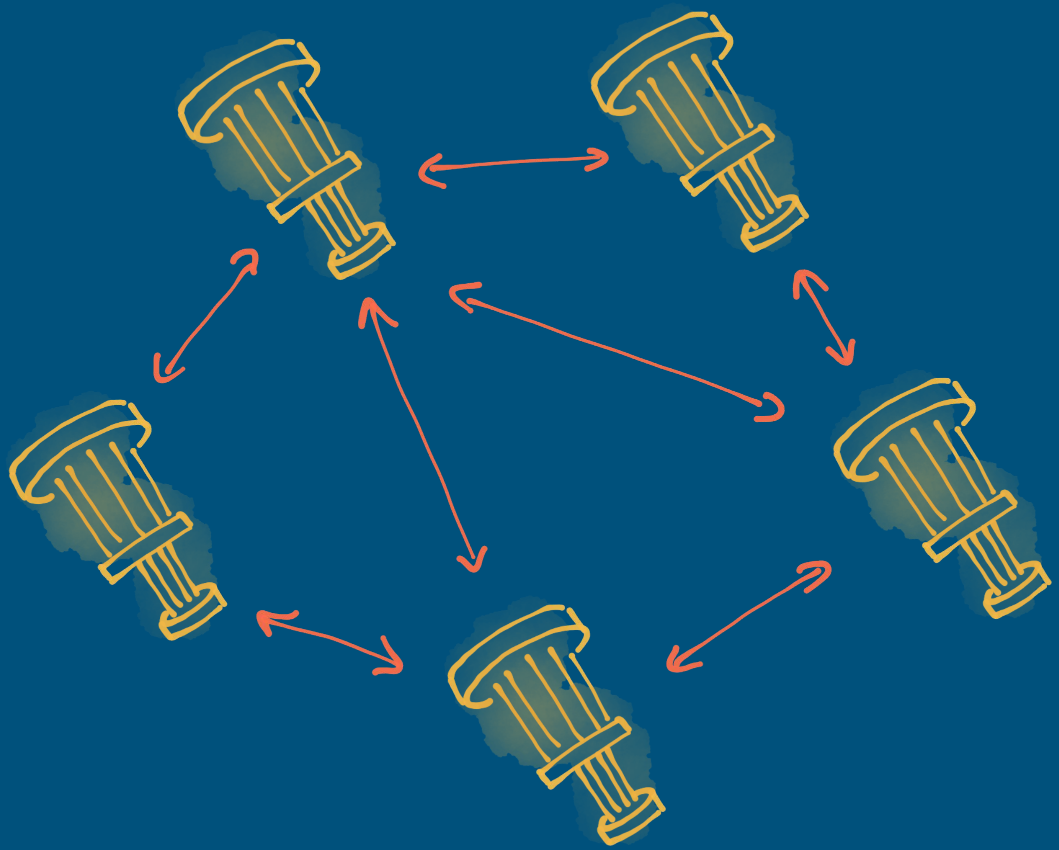

b
c
d
a

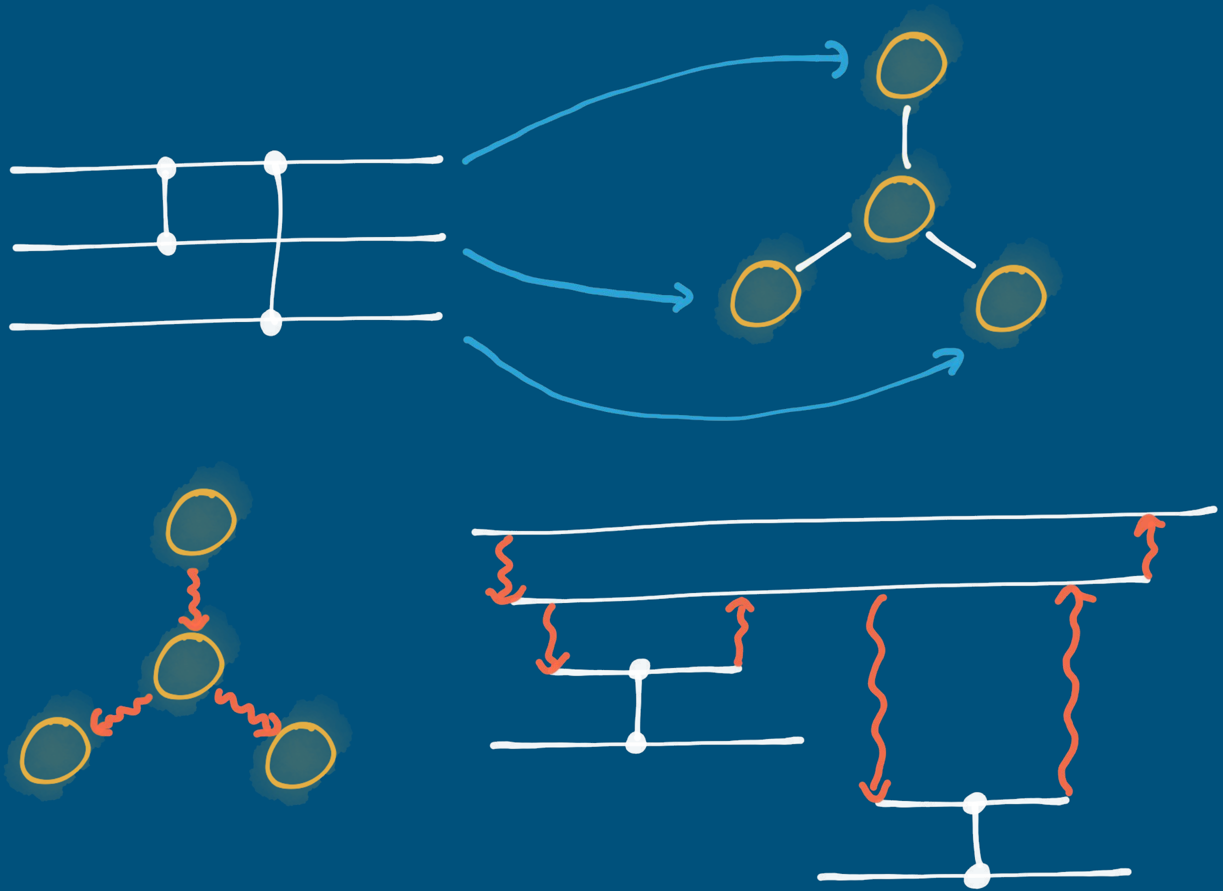


Automated distribution of quantum circuits via hypergraph partitioning

Heterogeneous Networks

- Modules of different sizes.
- Entanglement distribution.
- Qubit allocation and non-local gate distribution.
- Embedding.





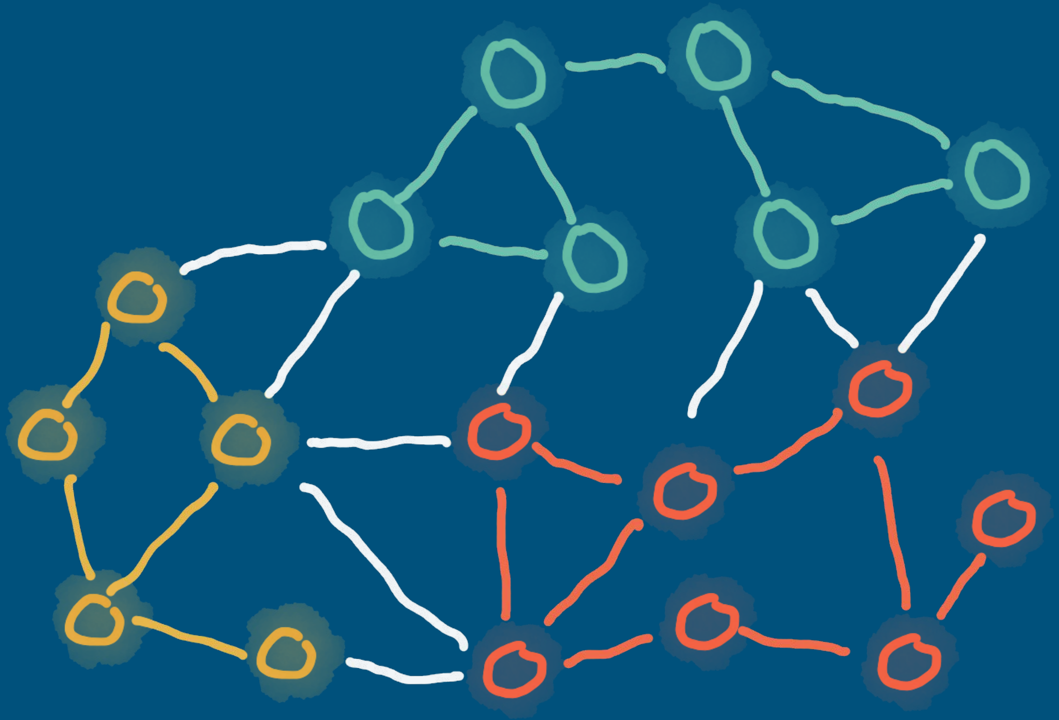
Qubit Allocation and Non-Local Gate Distribution

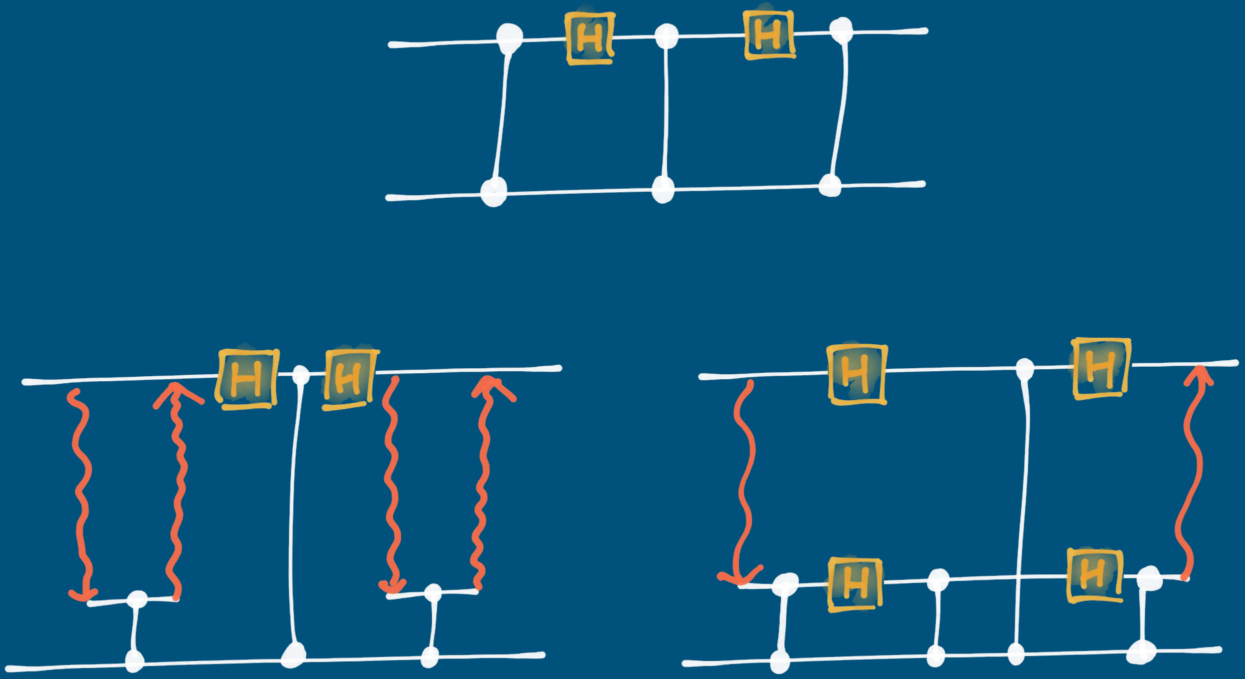
Rounds of updates:

- Move vertices to new module:
  - Gates move freely.
  - Qubits memory bound.
- Calculate cost.
- Rollback or commit.

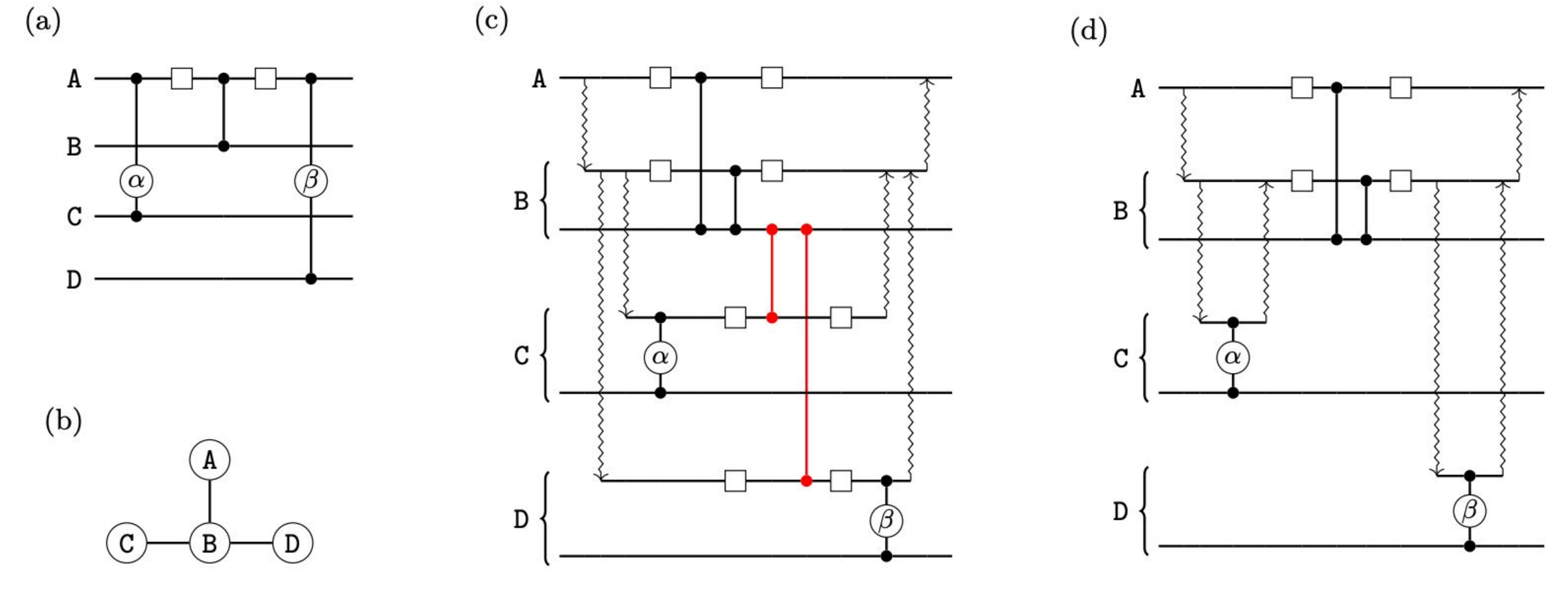
Two techniques:

- General purpose annealing.
- Modified graph partitioning.





Entanglement-efficient bipartite-distributed quantum computing with entanglement-assisted packing processes


(a)
A
B
C
D
α
β
(b)
A
C
B
D
(c)
A
B
C
D
α
β
(d)
A
B
C
D
α
β

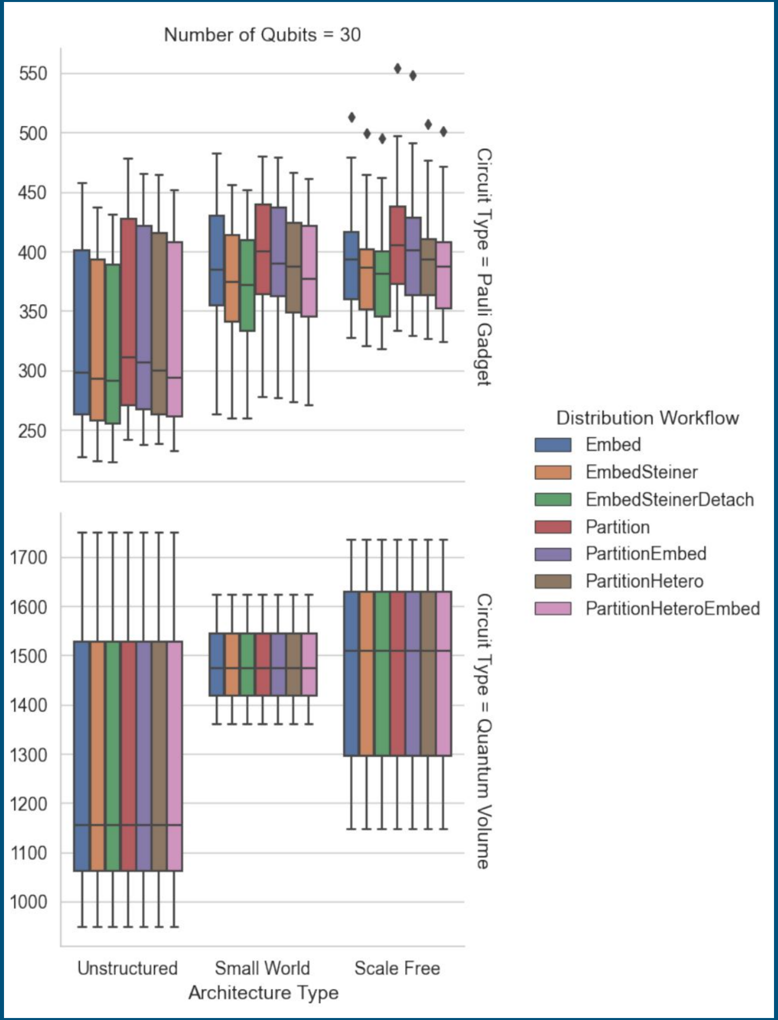
Benchmarks and Implementation

Automated Distribution of Quantum Circuits with pytket-dqc

- Rebase to CRz
- Qubit allocation
- Gate packing
- Non-local gate distribution
- Refinement
- Circuit generation

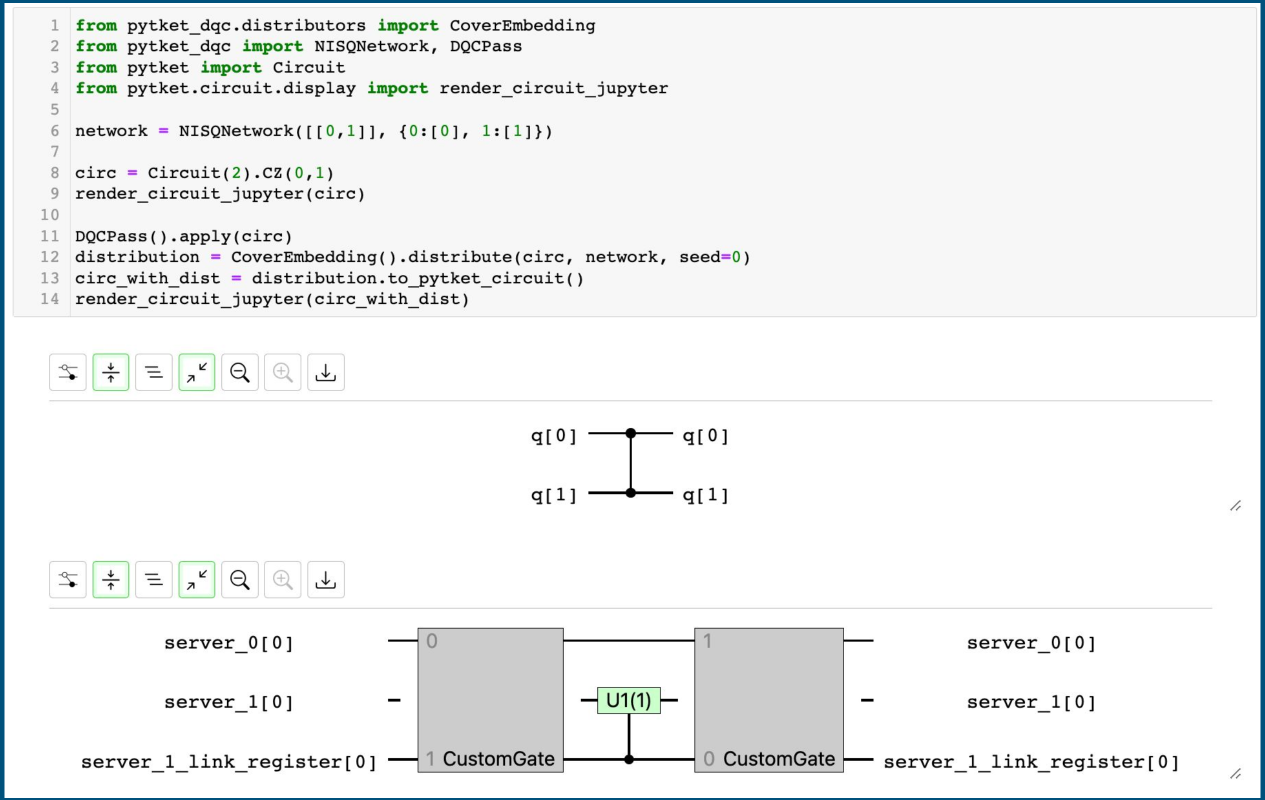
Key Findings

- Each refinement improves the median cost of Pauli Gadget circuits.
- Refinement has little effect on Quantum Volume circuits.
- Techniques combined perform best



Remarks

- Application benchmarks
- Homogeneous networks
- Bound link qubit registers
- Circuit generation
- pytket-dqc



```python
from pytket_dqc.distributors import CoverEmbedding
from pytket_dqc import NISQNetwork, DQCPass
from pytket import Circuit
from pytket.circuit.display import render_circuit_jupyter

network = NISQNetwork([[0,1]], {0:[0], 1:[1]})

circ = Circuit(2).CZ(0,1)
render_circuit_jupyter(circ)

DQCPass().apply(circ)
distribution = CoverEmbedding().distribute(circ, network, seed=0)
circ_with_dist = distribution.to_pytket_circuit()
render_circuit_jupyter(circ_with_dist)
```

Cheers

arxiv.org/abs/2305.14148

Distributing circuits over heterogeneous, modular quantum computing network architectures

–

Entanglement-efficient bipartite-distributed quantum computing with entanglement-assisted packing processes

arxiv.org/abs/2212.12688