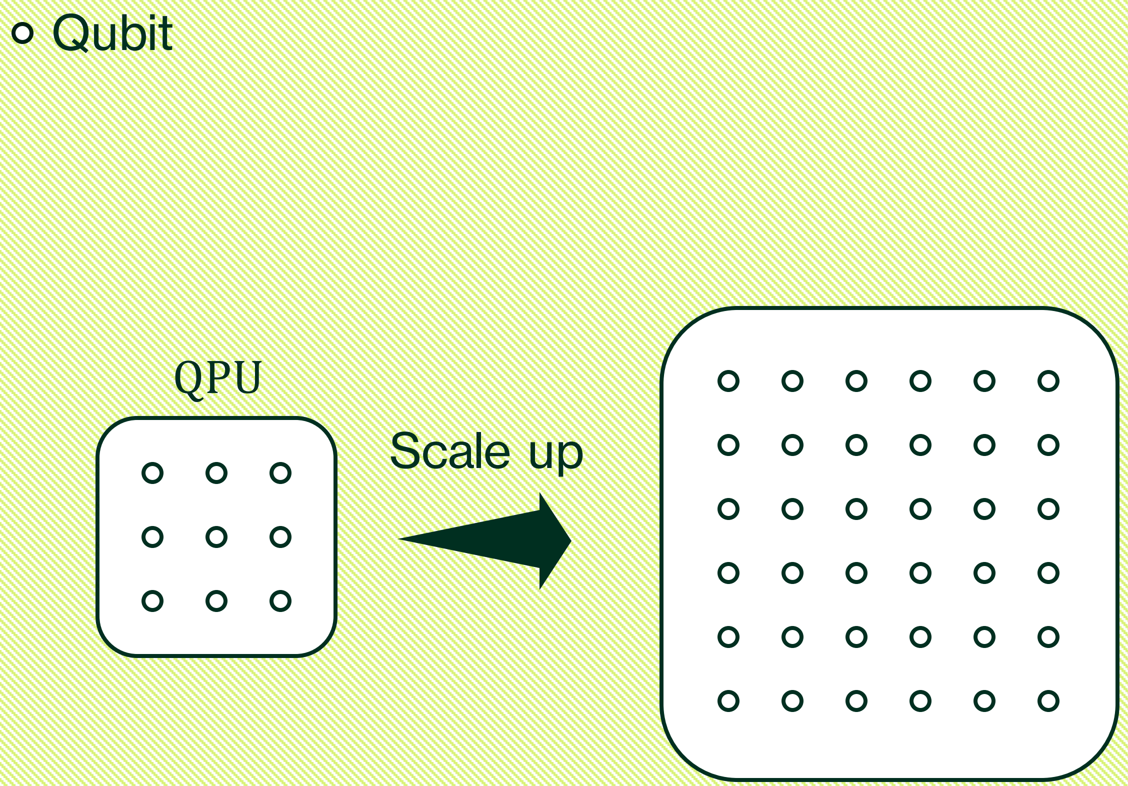
A COMPILER FOR DISTRIBUTED QUANTUM COMPUTING

Walter Nadalin

welinq × SORBONNE UNIVERSITÉ

DISTRIBUTED QUANTUM ARCHITECTURE

Objectives
- **Scalability**
- **Fault tolerance**

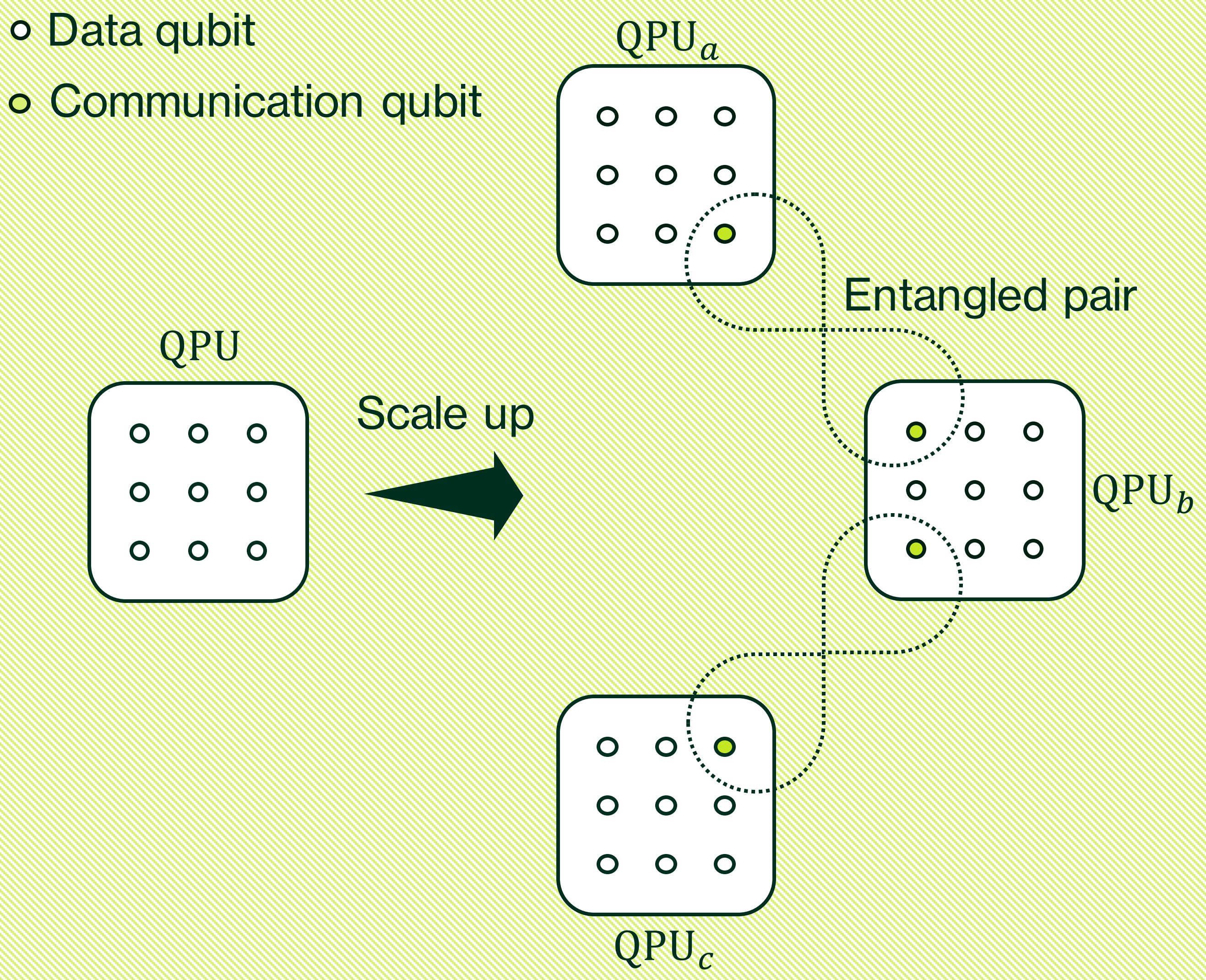


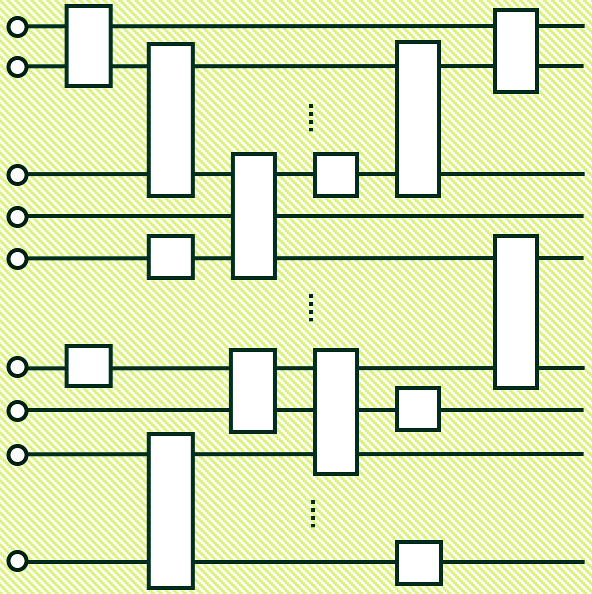
DISTRIBUTED QUANTUM ARCHITECTURE

Objectives

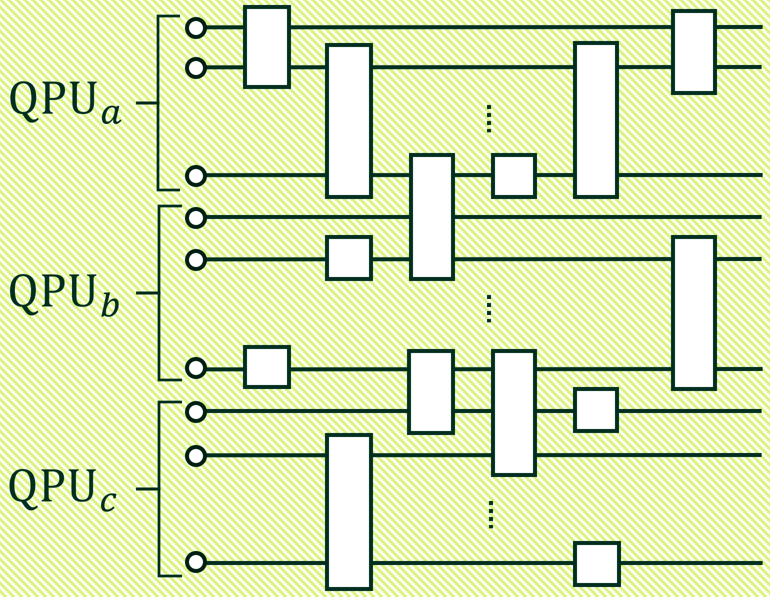
- **Scalability**
- **Fault tolerance**

Use a **network of QPUs** inter-connected with **classical** and **quantum connections** [1]



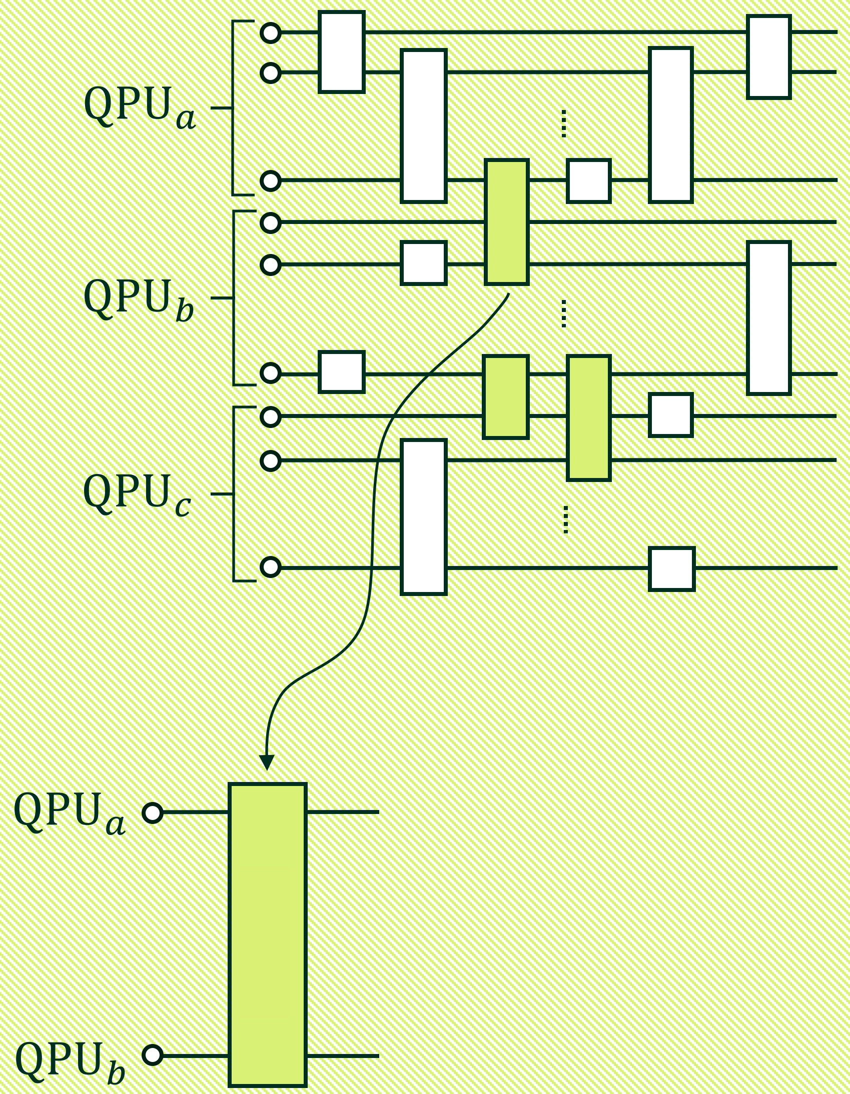


DISTRIBUTED QUANTUM COMPUTING



DISTRIBUTED QUANTUM COMPUTING

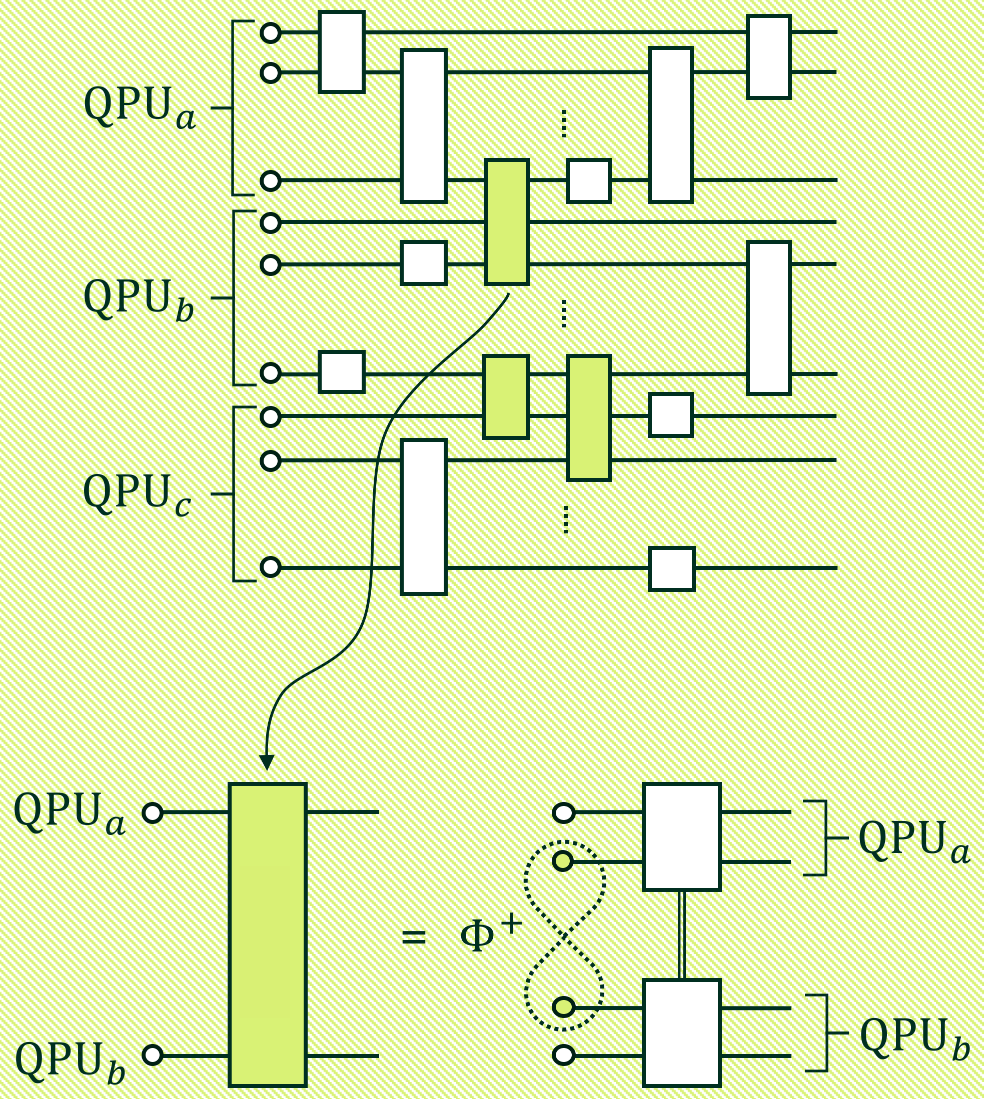
Assign each **data qubit** to a **QPU**



DISTRIBUTED QUANTUM COMPUTING

Assign each **data qubit** to a **QPU**

How to implement **non-local gates**?



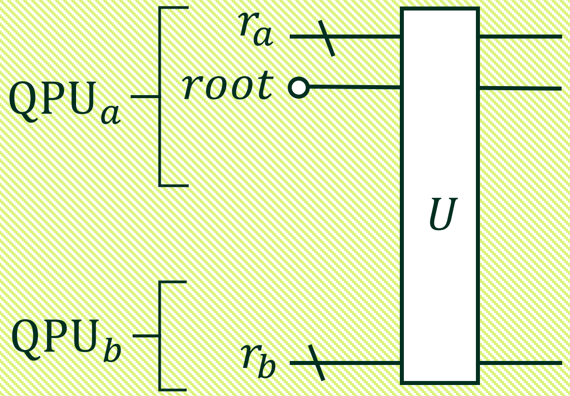
DISTRIBUTED QUANTUM COMPUTING

Assign each **data qubit** to a **QPU**

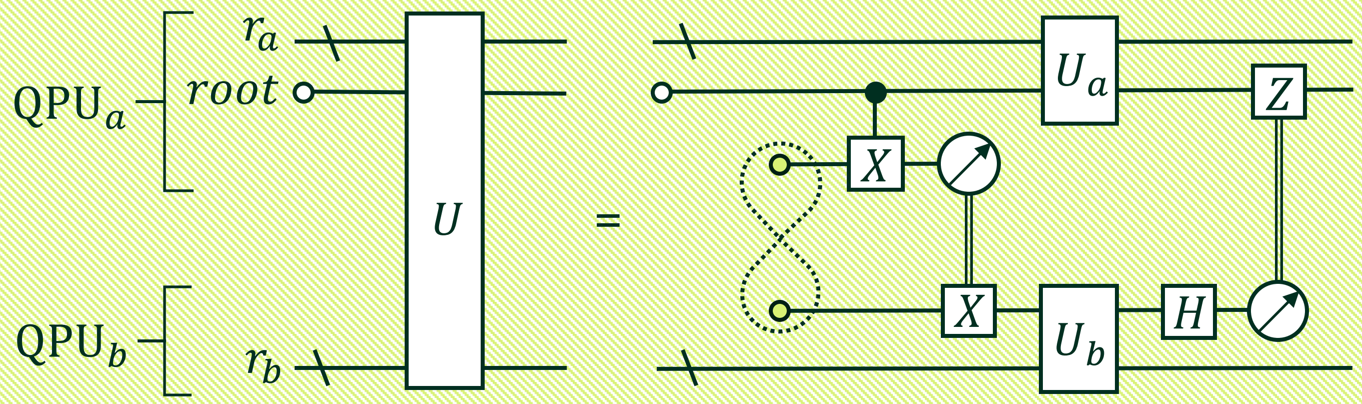
How to implement **non-local gates**?

Leverage **protocols** that utilize
- **Local operations**
- **Classical communication**
- **Pre-distributed entanglement**, such as Bell pairs

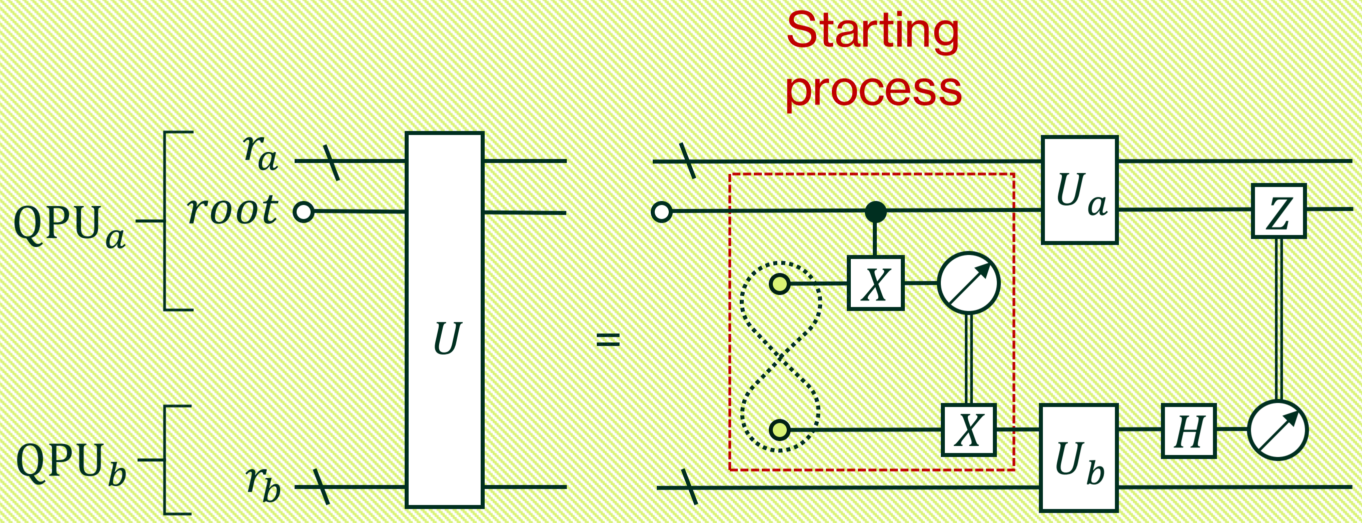
Gate teleportation protocol



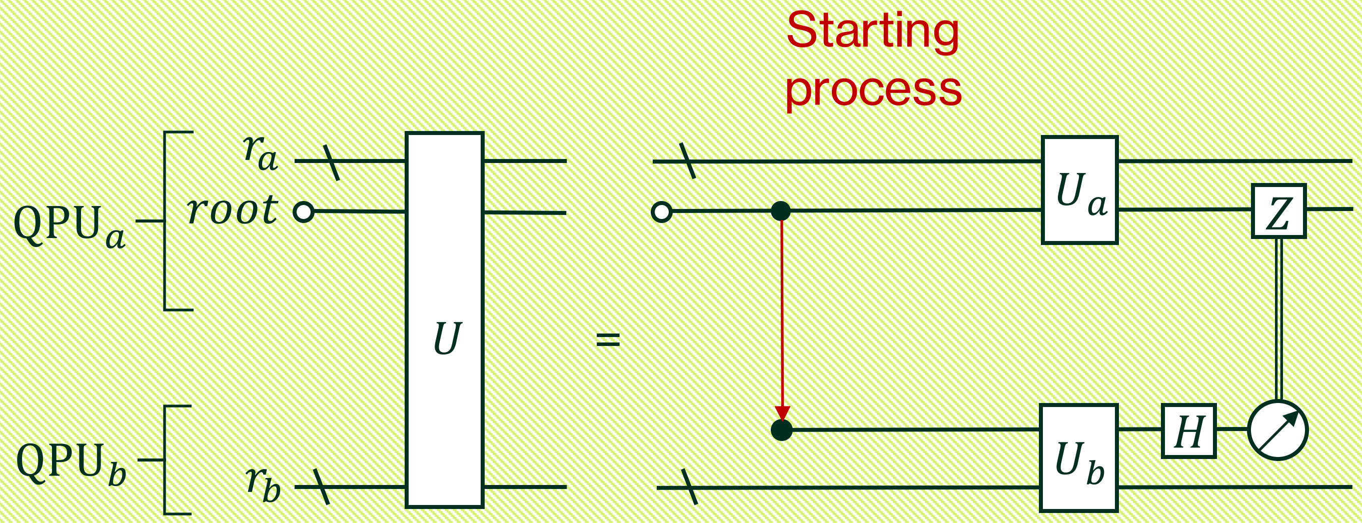
Gate teleportation protocol



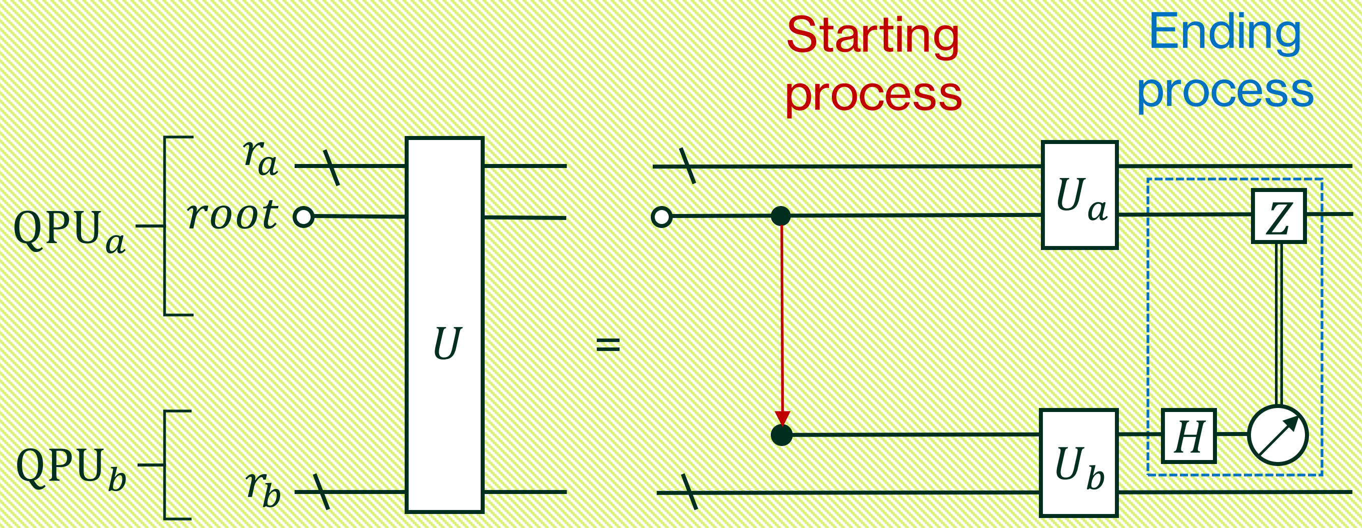
Gate teleportation protocol



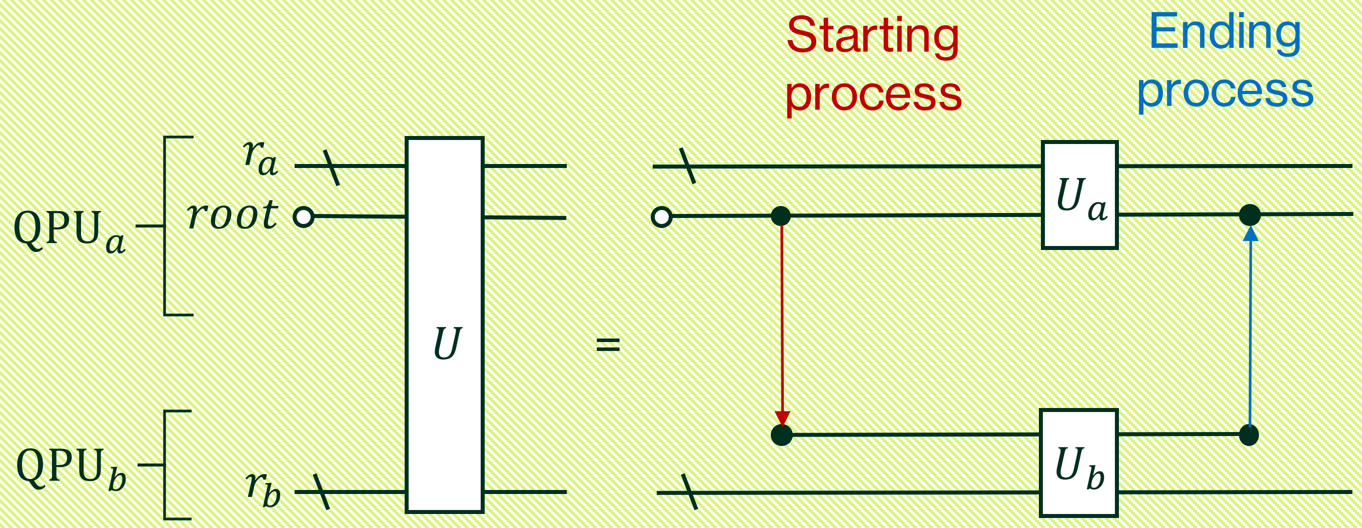
Gate teleportation protocol



Gate teleportation protocol



Gate teleportation protocol



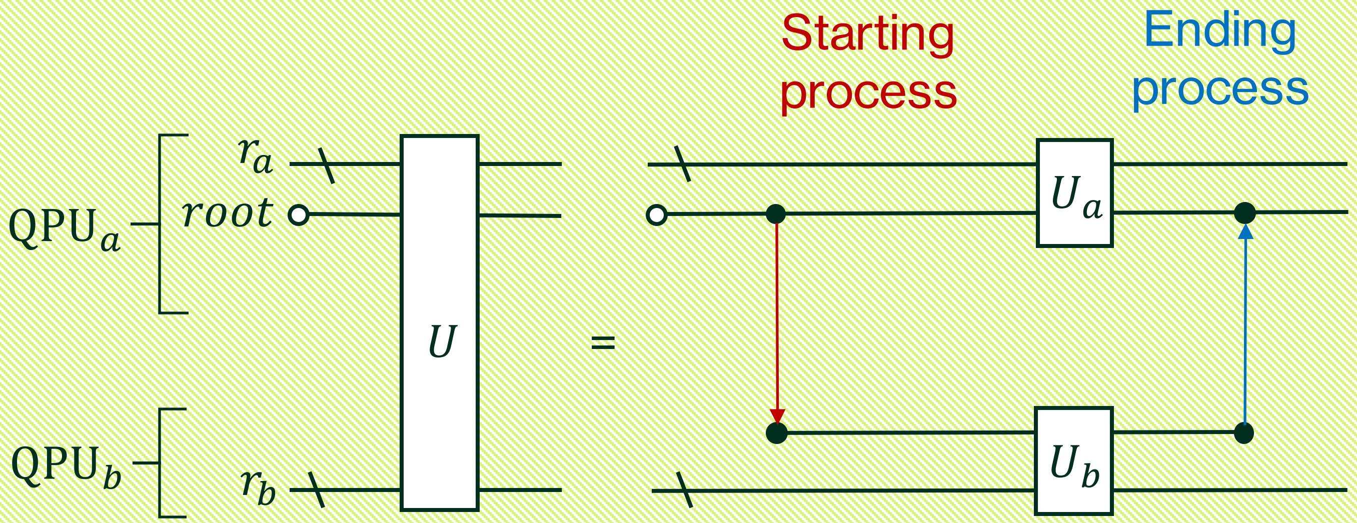
Gate teleportation protocol

Gate teleportation **condition** [2]

$$U = \sum_{\substack{n=0 \\ \ell=0}}^{1} A_\ell \otimes \Delta_{\ell n} |\ell\rangle\langle n| \otimes B_\ell$$

where

- A_ℓ operates on r_a and B_ℓ on r_b
- $\Delta_{\ell n} = \delta_n^\ell$ or $\delta_n^{1-\ell}$



Gate teleportation protocol

Gate teleportation **condition** [2]

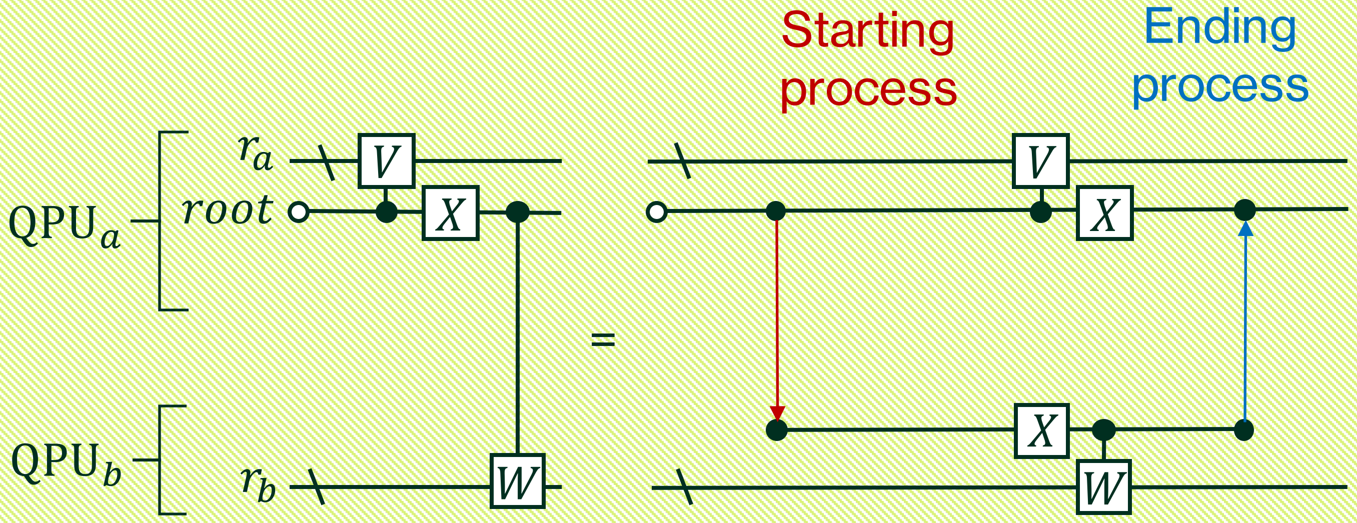
$$U = \sum_{\substack{n=0 \\ \ell=0}}^{1} A_\ell \otimes \Delta_{\ell n} |\ell\rangle\langle n| \otimes B_\ell$$

where

- A_ℓ operates on r_a and B_ℓ on r_b
- $\Delta_{\ell n} = \delta_n^\ell$ or $\delta_n^{1-\ell}$

The condition is satisfied by

- **Diagonal** and **anti-diagonal one-qubit gates** on $root$
- **Gates controlled** by $root$ with local target



Optimization problem

Select the **qubit assignment** to **minimize** the required number of **inter-QPU entangled pairs**

WORKFLOW

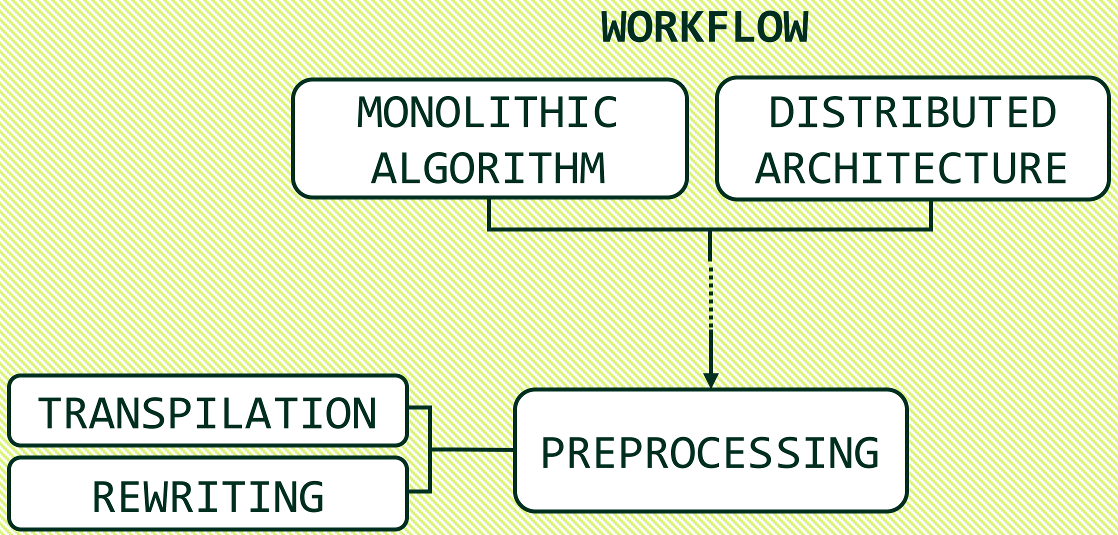
MONOLITHIC ALGORITHM

DISTRIBUTED ARCHITECTURE

Optimization problem

Select the **qubit assignment** to **minimize** the required number of **inter-QPU entangled pairs**

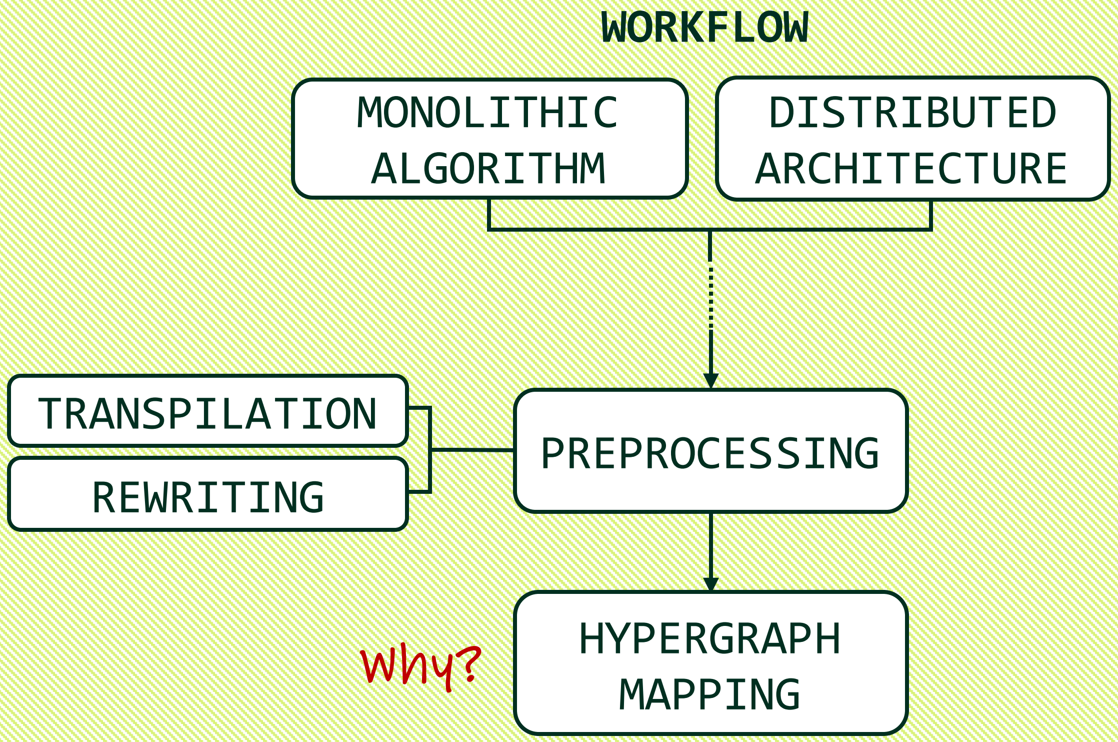
This selection can be **automated** and **embedded** into the **compiler** pipeline



Optimization problem

Select the **qubit assignment** to **minimize** the required number of **inter-QPU entangled pairs**

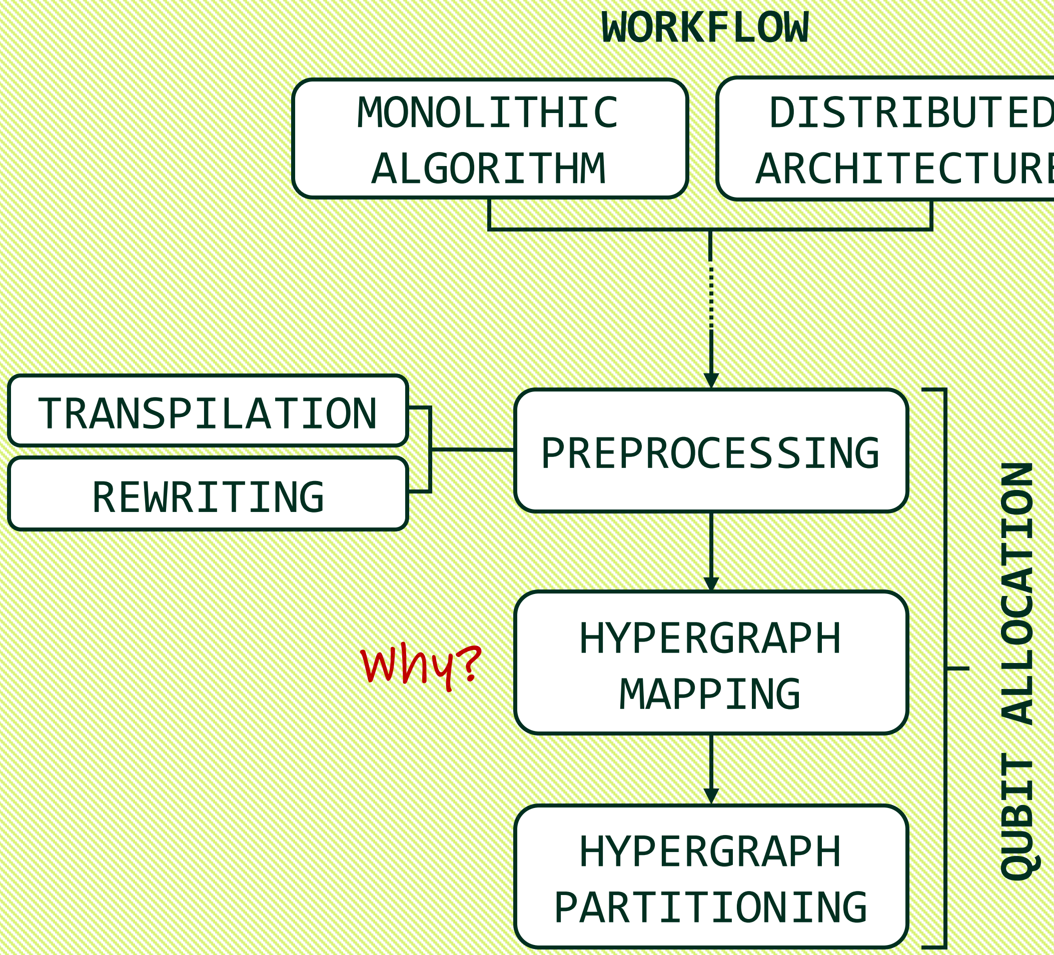
This selection can be **automated** and **embedded** into the **compiler** pipeline



Optimization problem

Select the **qubit assignment** to **minimize** the required number of **inter-QPU entangled pairs**

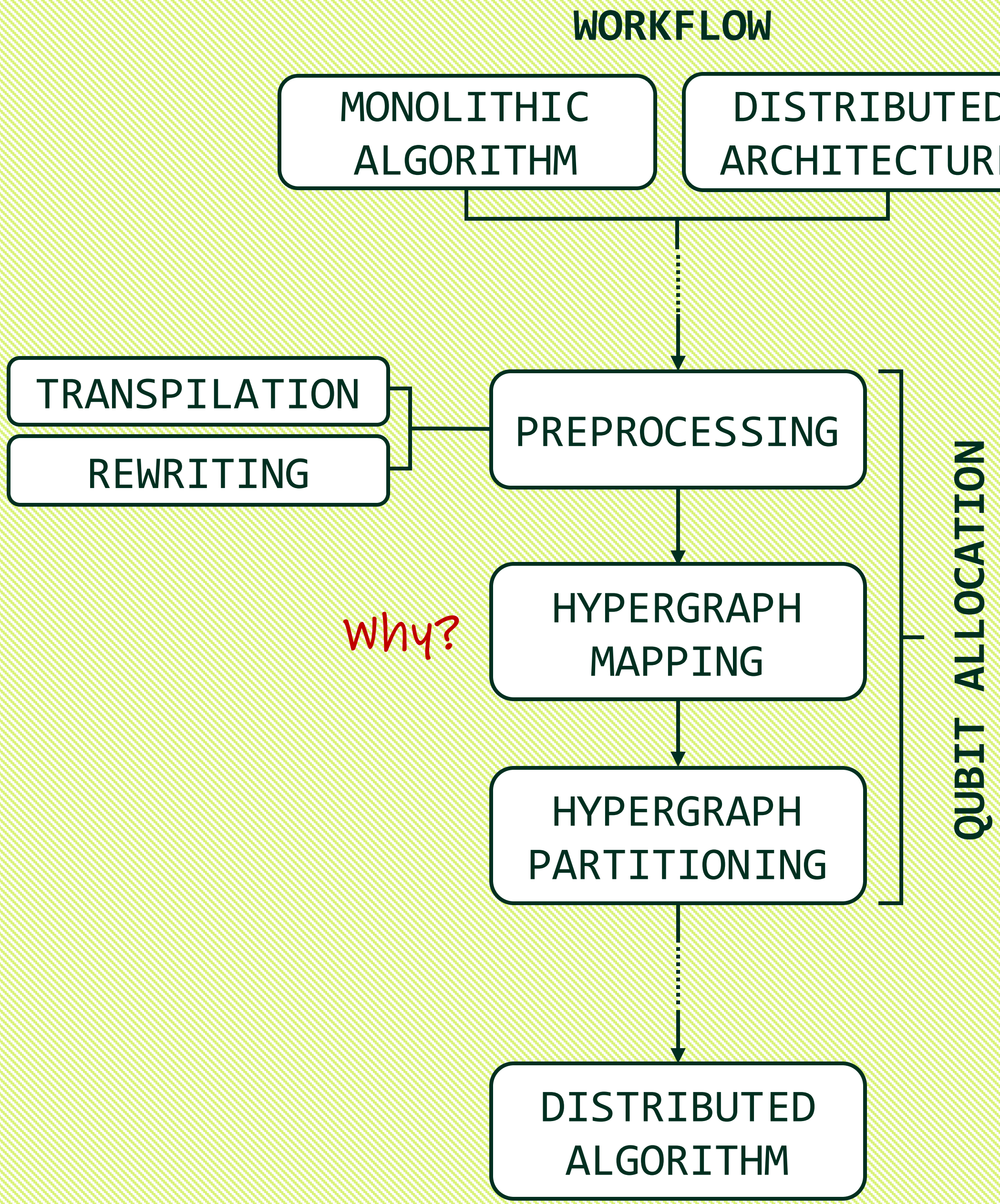
This selection can be **automated** and **embedded** into the **compiler** pipeline



Optimization problem

Select the **qubit assignment** to **minimize** the required number of **inter-QPU entangled pairs**

This selection can be **automated** and **embedded** into the **compiler** pipeline



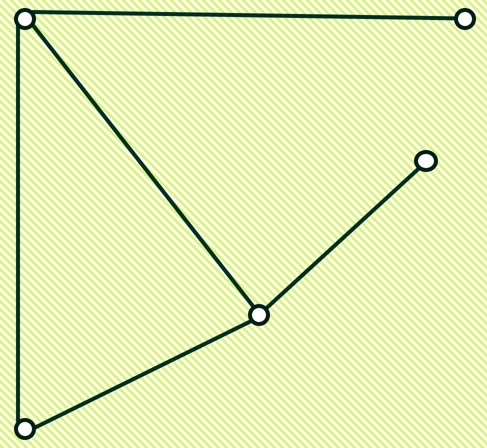
Optimization problem

Select the **qubit assignment** to **minimize** the required number of **inter-QPU entangled pairs**

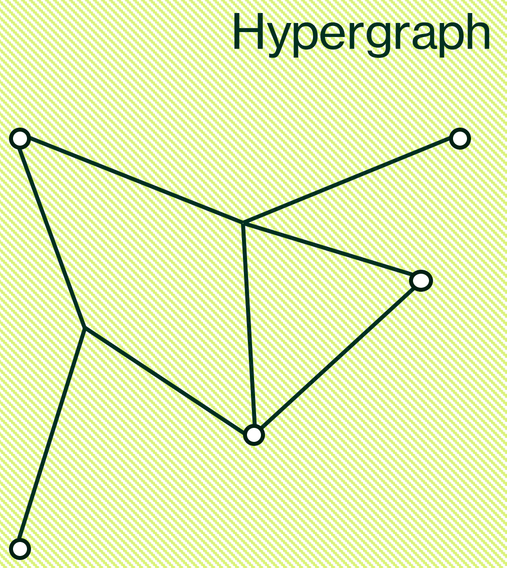
This selection can be **automated** and **embedded** into the **compiler** pipeline

Hypergraph partitioning

Graph



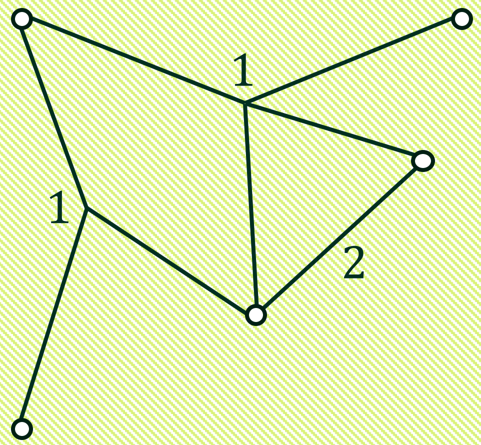
Hypergraph partitioning



An edge can connect **more than two vertices**

Hypergraph partitioning

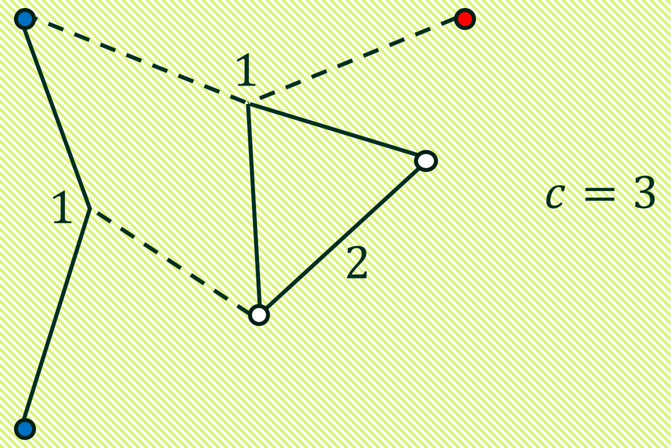
Weighted hypergraph



An edge can connect **more than two vertices**

Hypergraph partitioning

Weighted hypergraph



An edge can connect **more than two vertices**

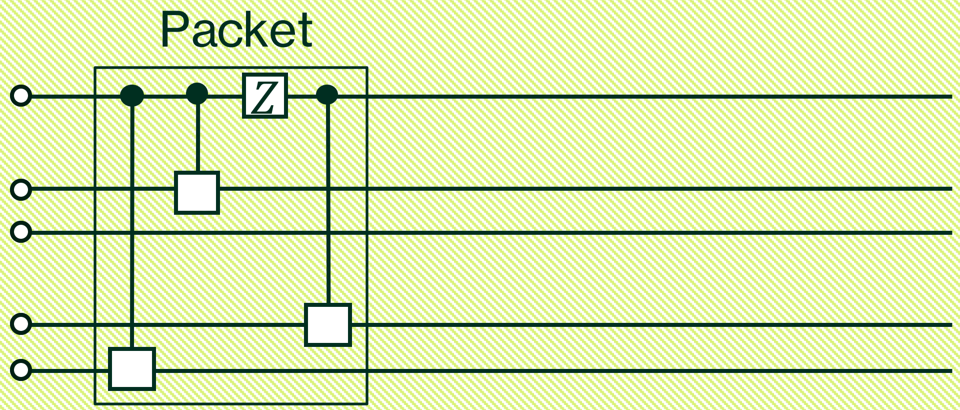
Find a vertex **partition that minimizes** the **cost function**

$$c = \sum_{e \in E} [F(e) - 1]\, w_e$$

with $F(e)$ number of parts connected by e [3]

Hypergraph mapping

A rooted **packet** has two-qubit **controlled gates** and **diagonal** or **anti-diagonal** one-qubit gates on its root

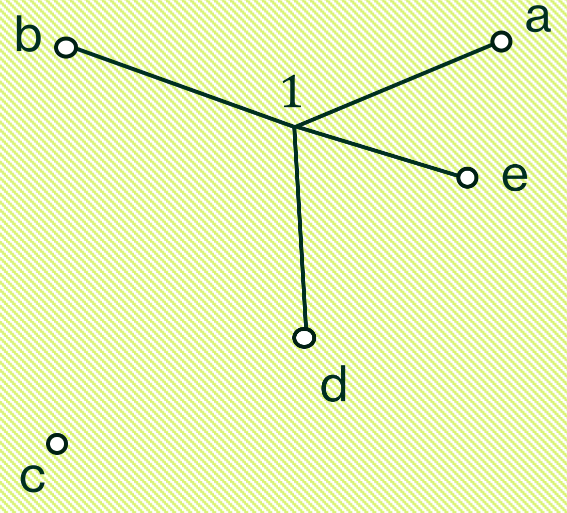


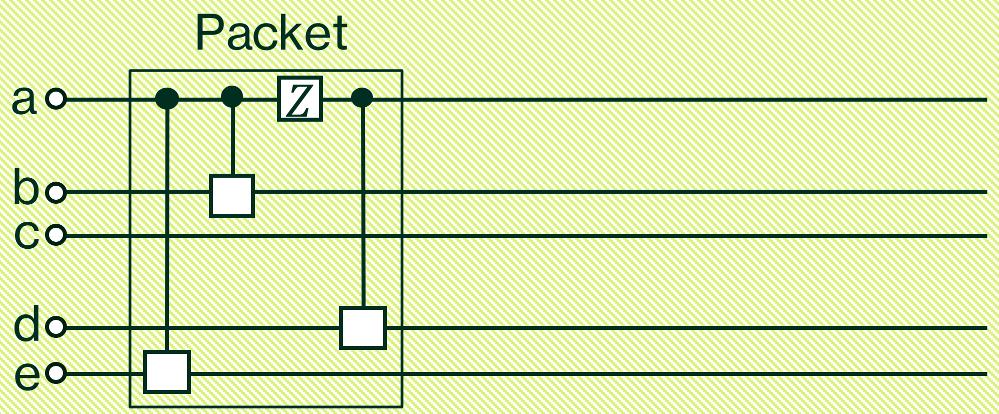
Hypergraph mapping

A rooted **packet** has two-qubit **controlled gates** and **diagonal** or **anti-diagonal** one-qubit gates on its root

Map circuit to hypergraph [4]

- **data qubits** to **vertices**
- **packets** to **edges**

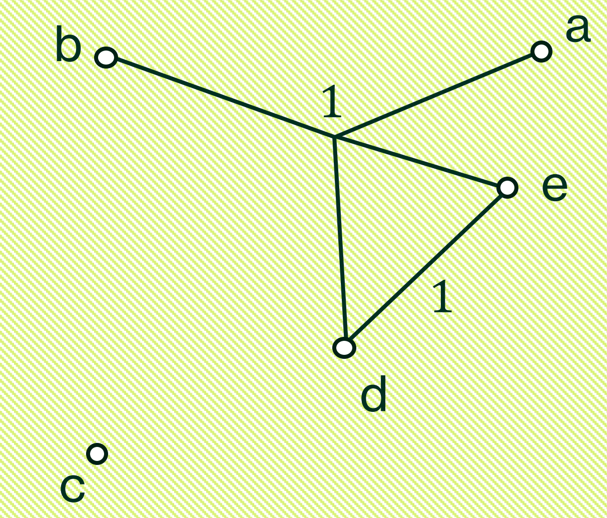


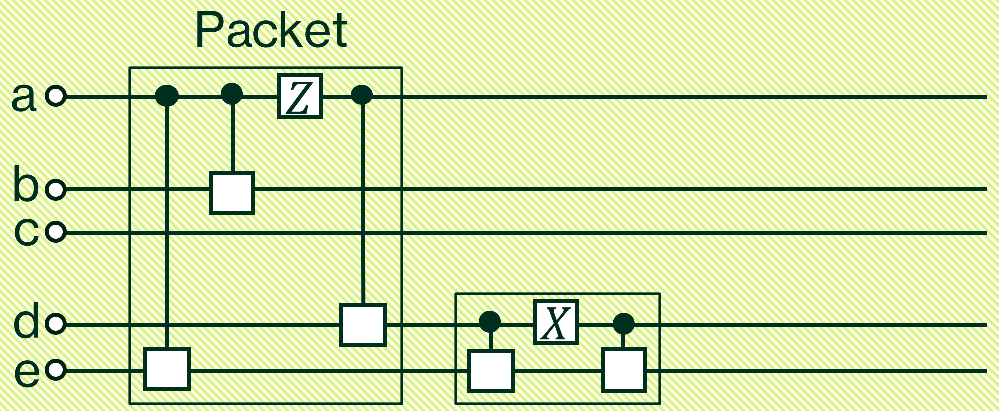


Hypergraph mapping

A rooted **packet** has two-qubit **controlled gates** and **diagonal** or **anti-diagonal** one-qubit gates on its root

Map circuit to hypergraph [4]
- **data qubits** to **vertices**
- **packets** to **edges**

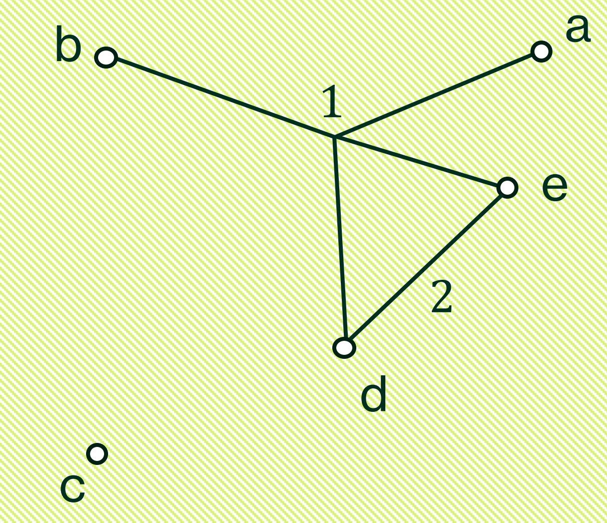


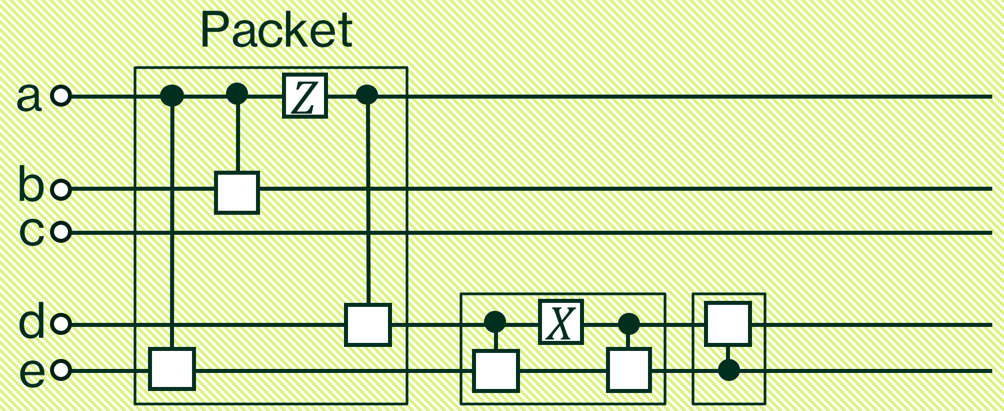


Hypergraph mapping

A rooted **packet** has two-qubit **controlled gates** and **diagonal** or **anti-diagonal** one-qubit gates on its root

Map circuit to hypergraph [4]
- **data qubits** to **vertices**
- **packets** to **edges**



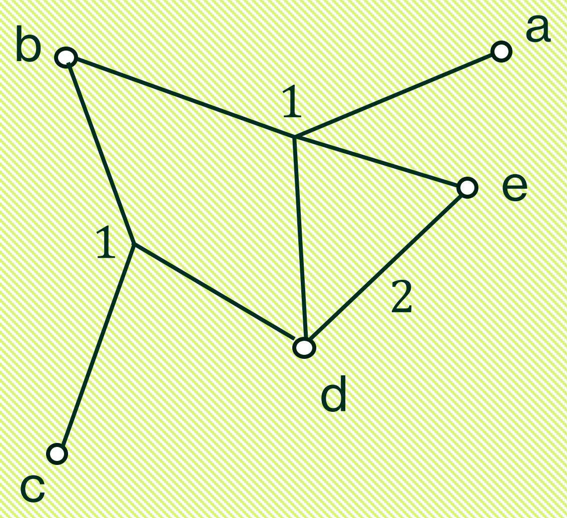


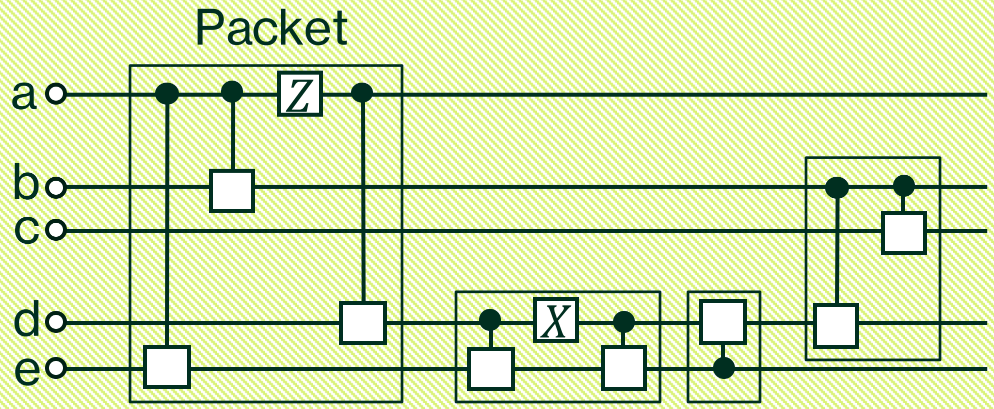
Hypergraph mapping

A rooted **packet** has two-qubit **controlled gates** and **diagonal** or **anti-diagonal** one-qubit gates on its root

Map circuit to hypergraph [4]

- **data qubits** to **vertices**
- **packets** to **edges**



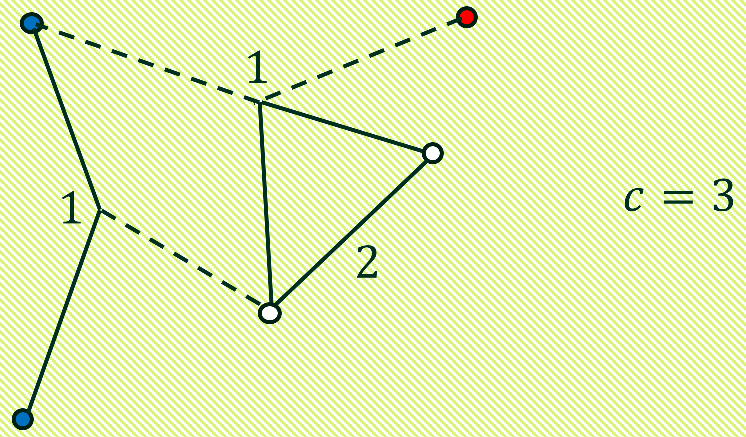


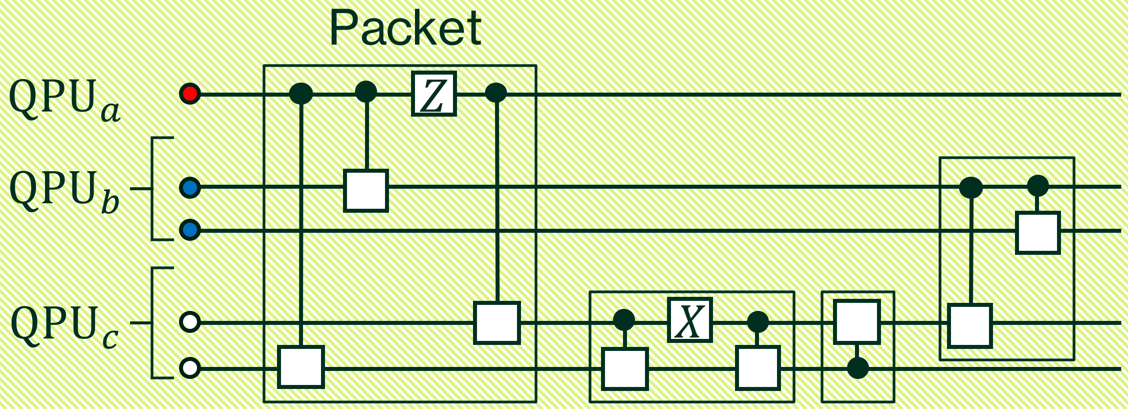
Hypergraph mapping

A rooted **packet** has two-qubit **controlled gates** and **diagonal** or **anti-diagonal** one-qubit gates on its root

Map circuit to hypergraph [4]

- **data qubits** to **vertices**
- **packets** to **edges**





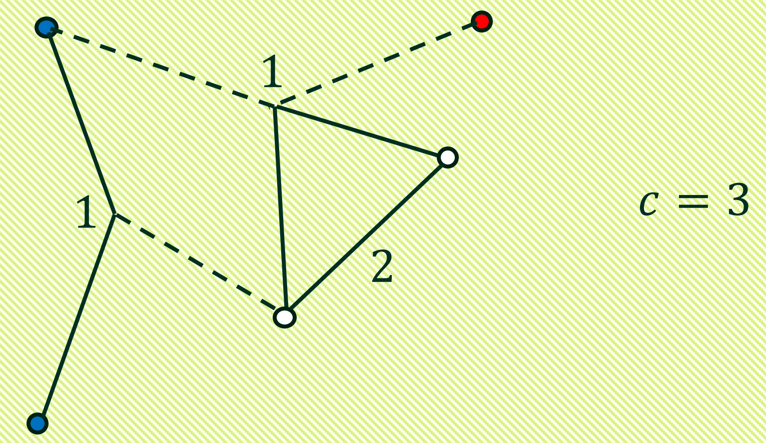
Hypergraph mapping

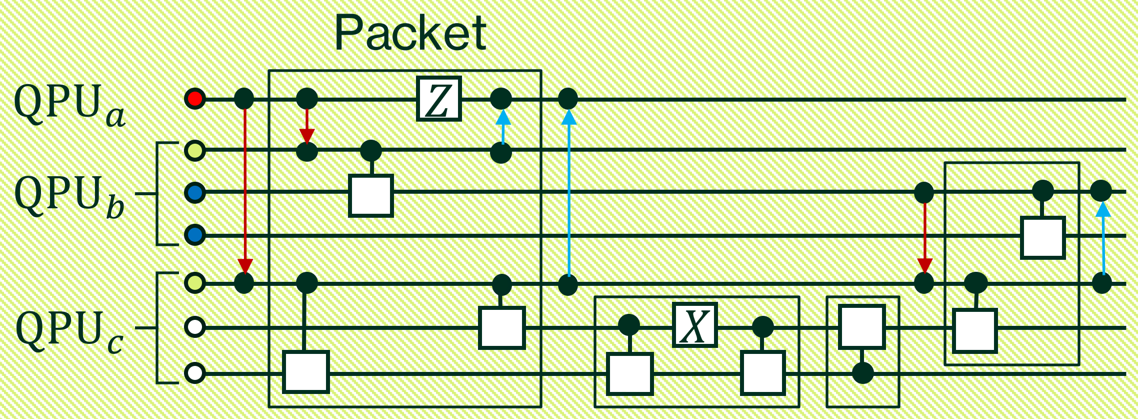
A rooted **packet** has two-qubit **controlled gates** and **diagonal** or **anti-diagonal** one-qubit gates on its root

Map circuit to hypergraph [4]

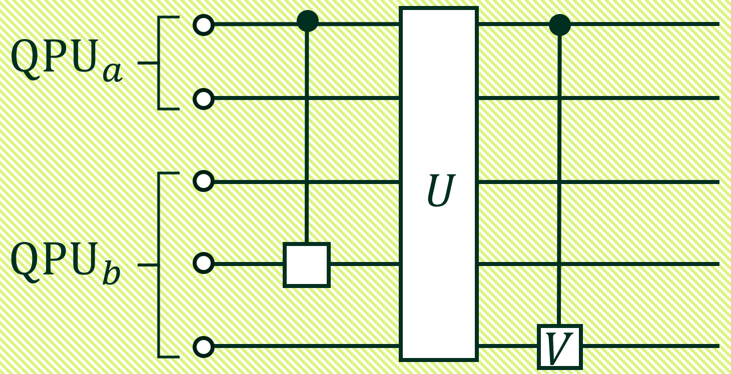
- **data qubits** to **vertices**
- **packets** to **edges**

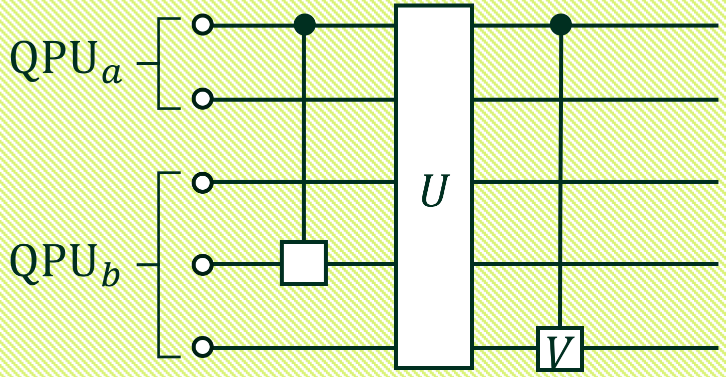
Cost function is equal to **number of Bell pairs**



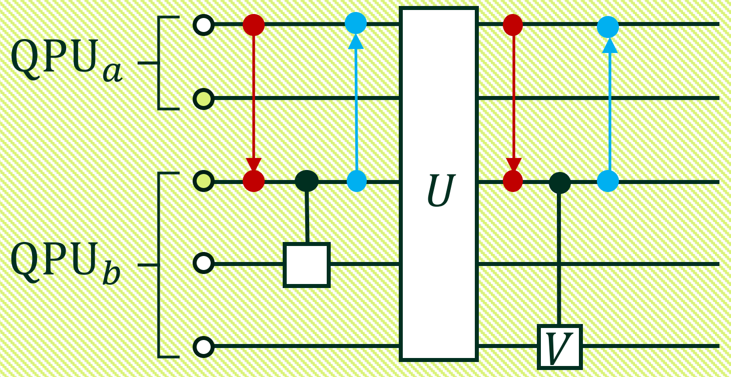


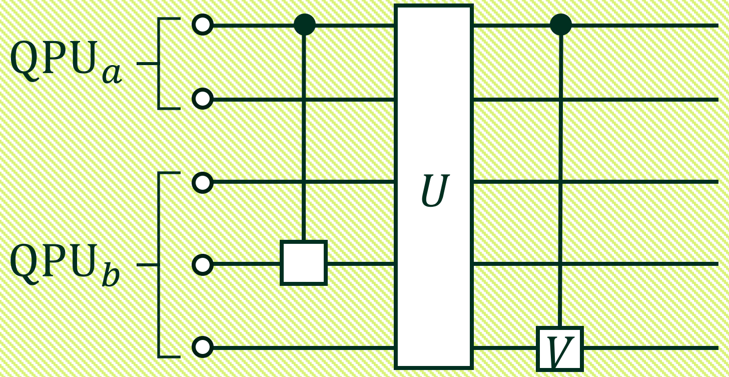
Preprocessing





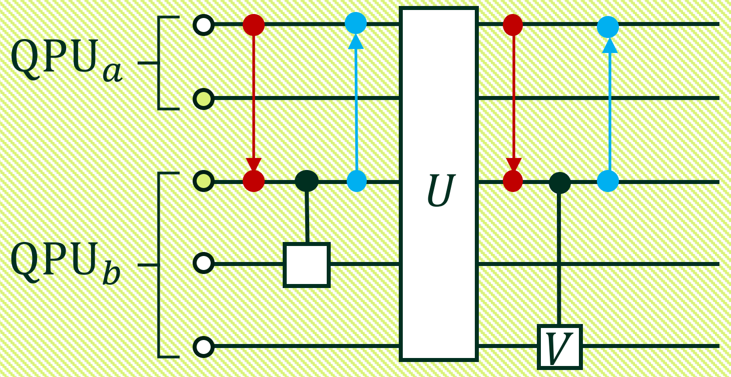
Preprocessing

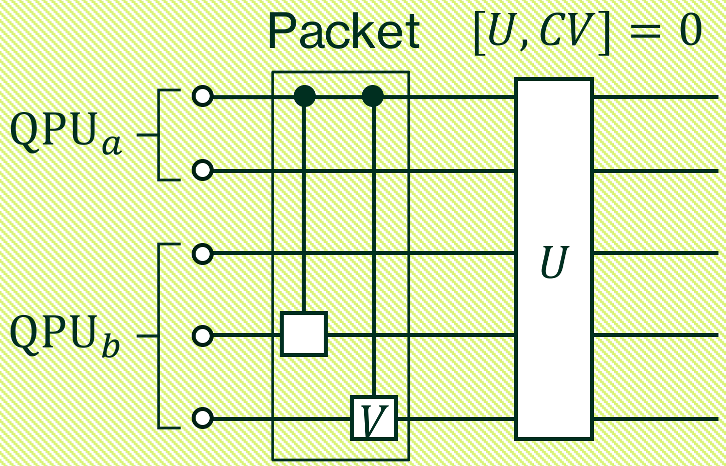




Two Bell pairs are used plus a constant number for U

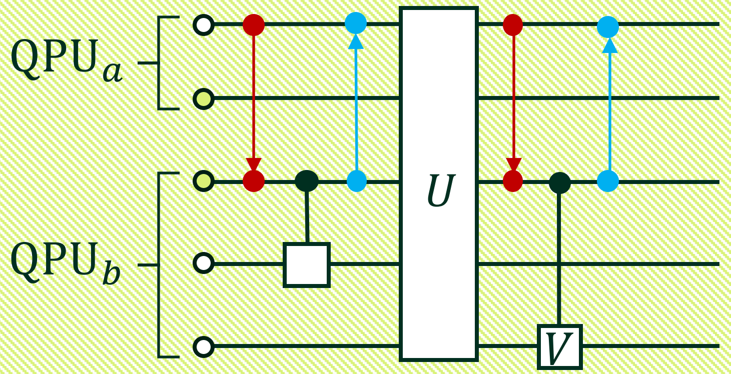
Preprocessing



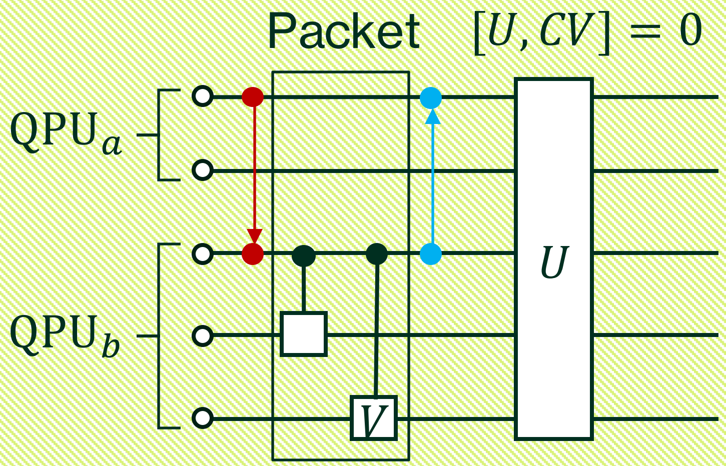


Two Bell pairs are used plus a constant number for U

Preprocessing



Two Bell pairs are used plus a constant number for U



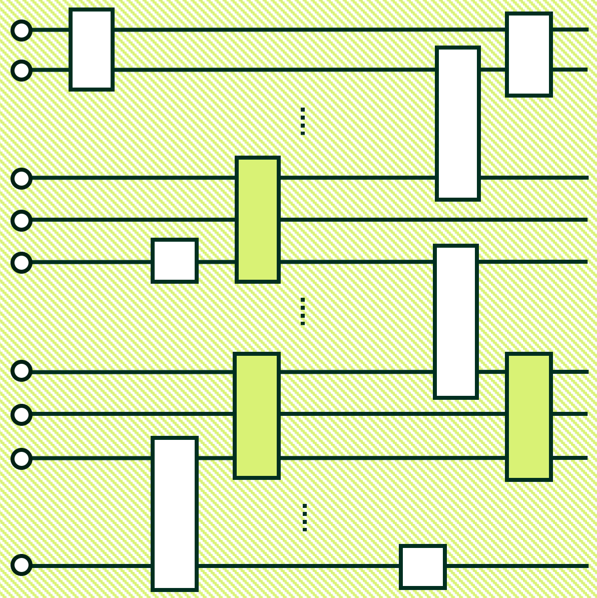
After commuting, the circuit is equivalent and only **one Bell pair** is used, plus a constant number for U

The compiler

In **ARAQNE**

- **preprocessing phase**
- **hypergraph mapping** and **partitioning**

allow to reduce inter-QPU entanglement [5]

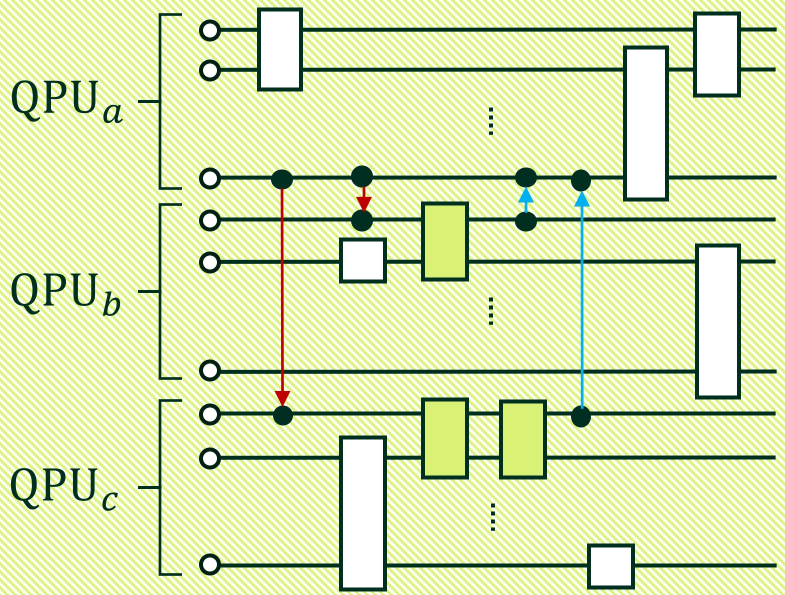


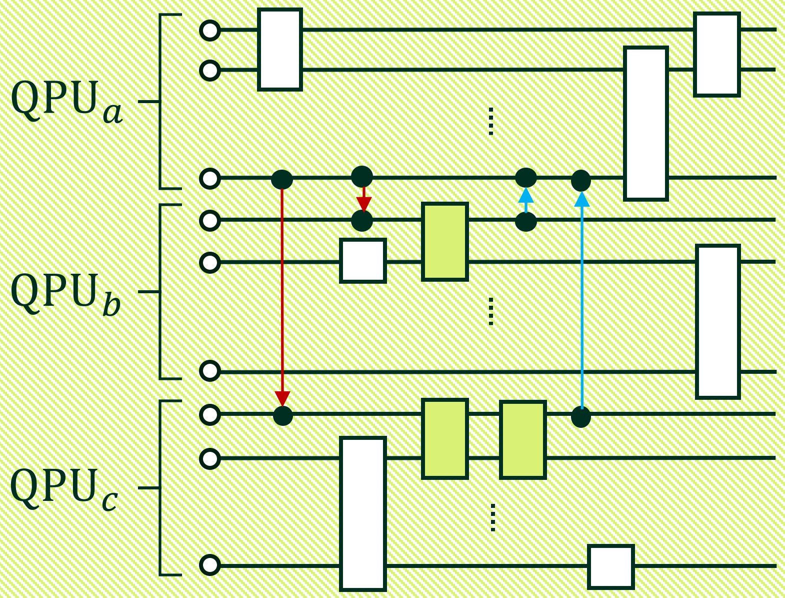
The compiler

In **ARAQNE**

- **preprocessing phase**
- **hypergraph mapping** and **partitioning**

allow to reduce inter-QPU entanglement [5]





The compiler

In **ARAQNE**

- **preprocessing phase**
- **hypergraph mapping** and **partitioning**

allow to reduce inter-QPU entanglement [5]

In particular

- Bipartition of **QFT** requires $O(n)$ **entangled pairs** for $O(n^2)$ **non-local gates**
- Circuits such as **Quantum Volume** require the integration of **other protocols**

The compiler

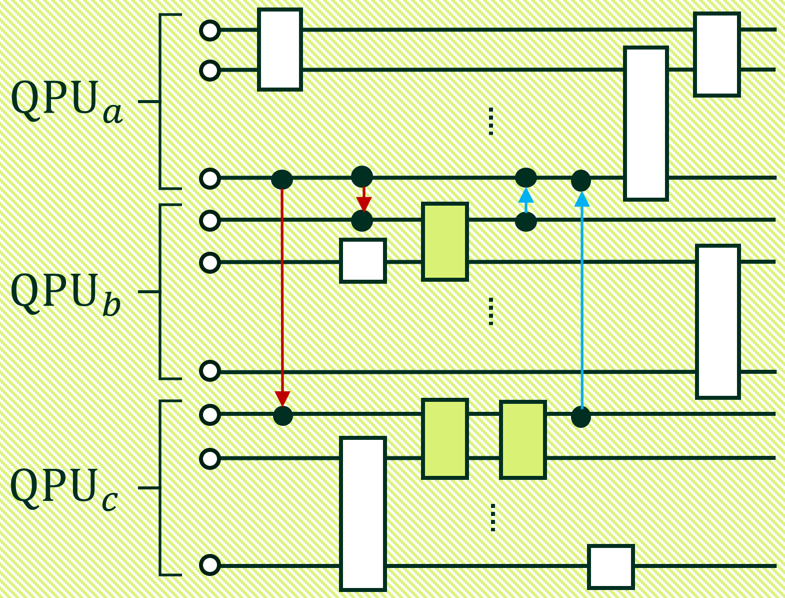
In **ARAQNE**

- **preprocessing phase**
- **hypergraph mapping** and **partitioning**

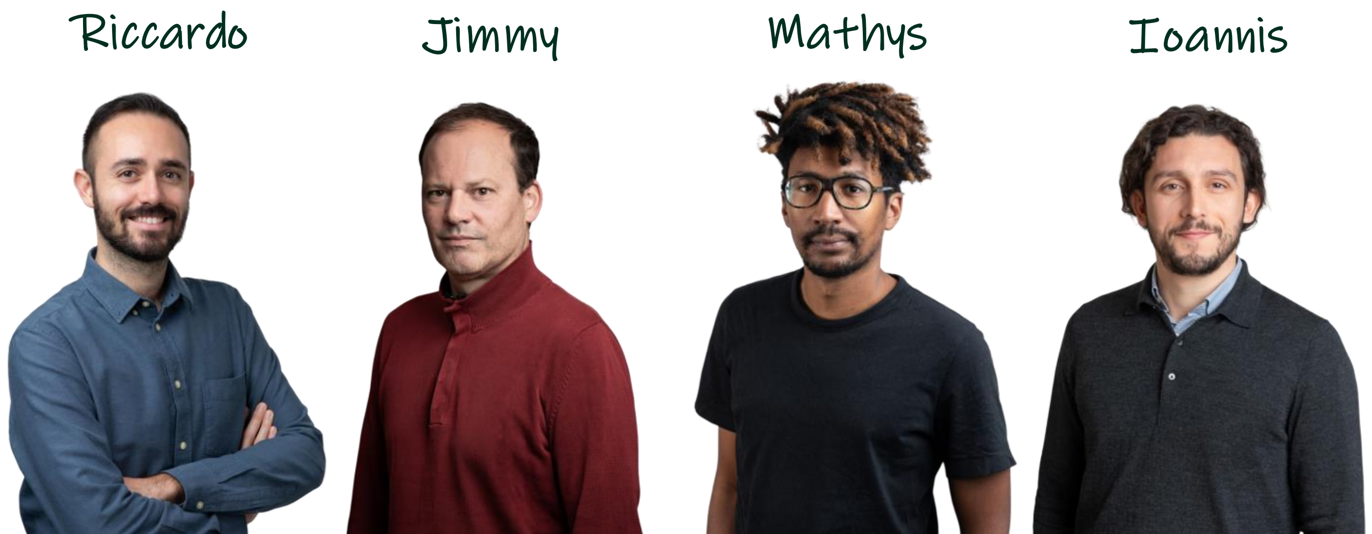
allow to reduce inter-QPU entanglement [5]

In particular

- Bipartition of **QFT** requires $O(n)$ **entangled pairs** for $O(n^2)$ **non-local gates**
- Circuits such as **Quantum Volume** require the integration of **other protocols**



arXiv:2507.01090



References

[1] Caleffi, M. & Amoretti, M. & Ferrari, D. et al. (2024). *Distributed quantum computing: A survey. Computer Networks.*

[2] Wu, J. & Andres-Martinez, P. & Forrer, T. et al. (2024). *Entanglement-efficient distributed quantum computing. Quantum 2.0 Conference and Exhibition.*

[3] Schlag, S. & Heuer, T. & Gottesburen, L. et al. (2022). *High-Quality Hypergraph Partitioning. ACM J. Exp. Algorithmics.*

[4] Andres-Martinez, P. & Forrer, T. & Mills, D. et al. (2024). *Distributing circuits over heterogeneous, modular quantum computing network architectures. IOP Publishing.*

[5] Mengoni, R. & Rennela, M. & Rotureau, J. & Lavdas I. et al. (2025). *Efficient Gate Reordering for Distributed Quantum Compiling in Data Centers. arXiv.*

THANK YOU