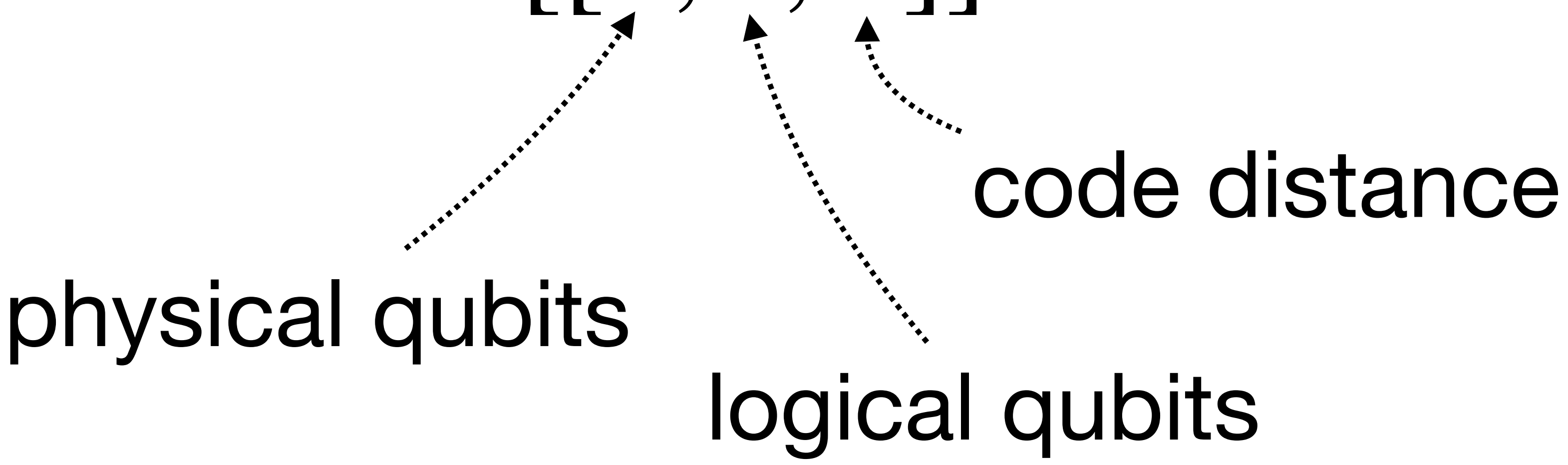
Qiskit FTQC updates

Japan Practitioner Forum

Julien Gacon / jul@zurich.ibm.com

Fault-tolerant quantum computing (FTQC)

- Real-time error correction, given…
  - an $[[n, k, d]]$ error correcting code



physical qubits

logical qubits

code distance

  - gate error $\leq$ (code-specific) threshold
- Access to a discrete, universal gate set, for example: Cliffords + T gate

Surface code

✓ local interactions, achievable error threshold, T factories

x low rate $r = k/n \approx 1/(2d^2)$, leading to large overhead at higher code distances

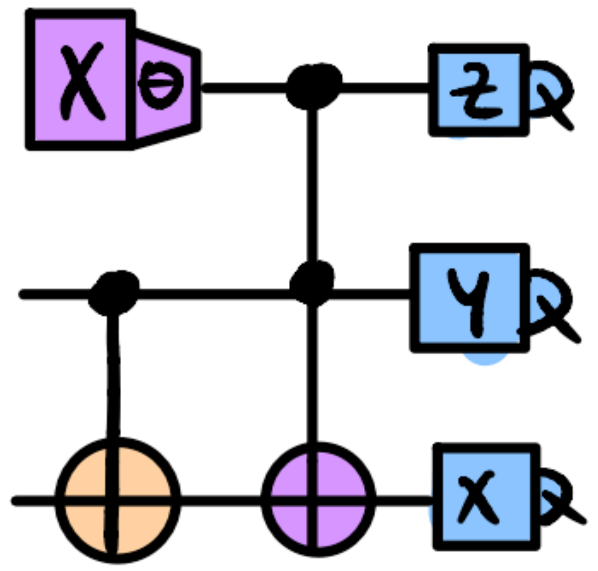
(Good) Quantum LDPC code

✓ constant rate, achievable error threshold, T factories (in prep)

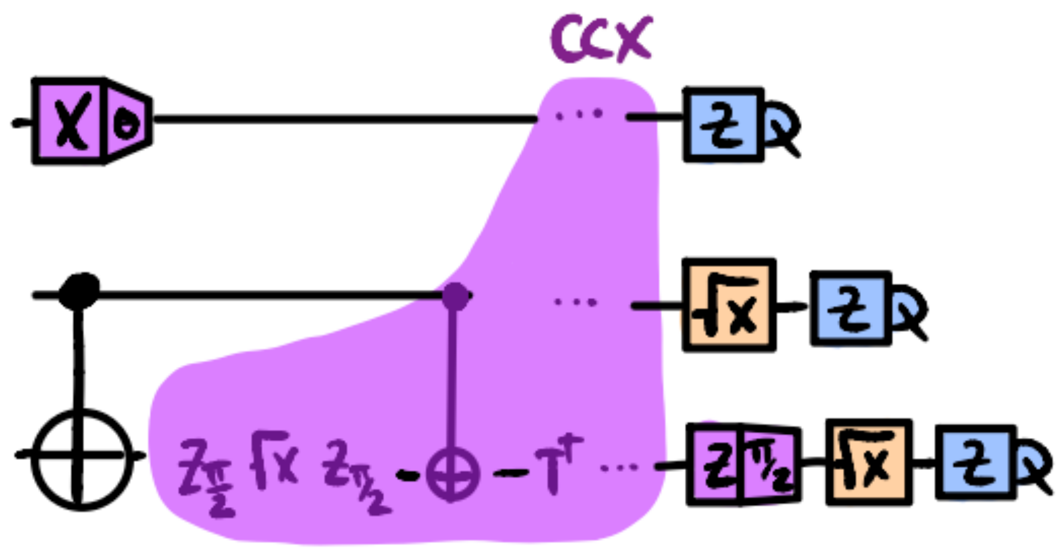
x non-local

New compilation targets

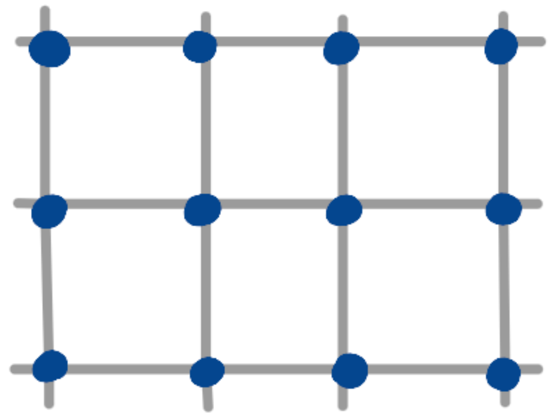
Logical circuit



Near-term basis sets

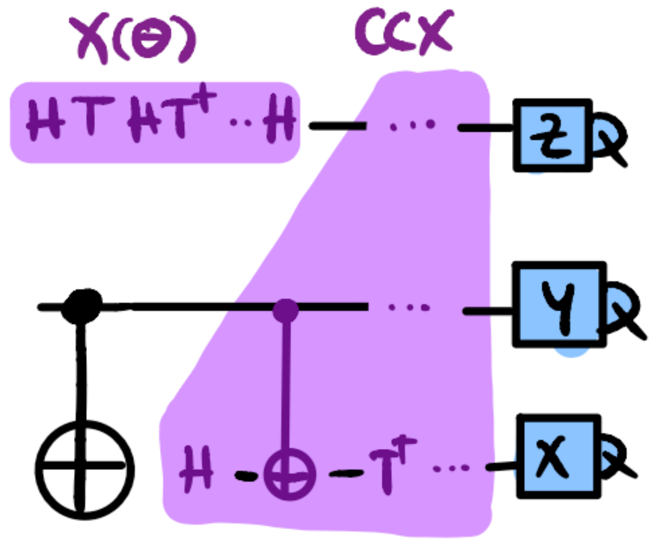


Near-term devices

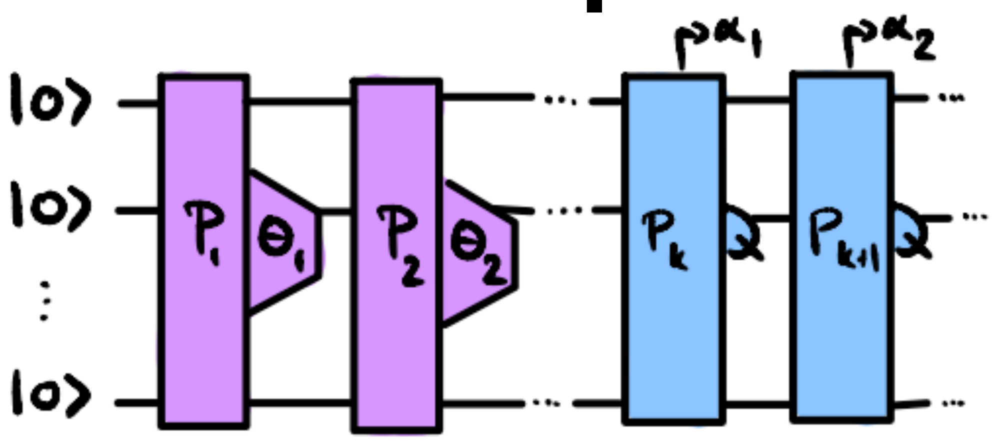


["id", "sx", "x", "cx", "rz"]

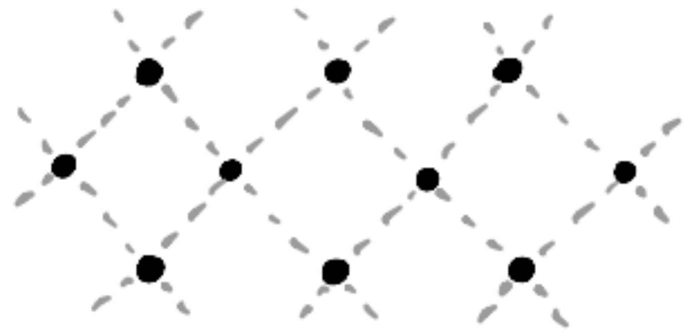
Clifford+T



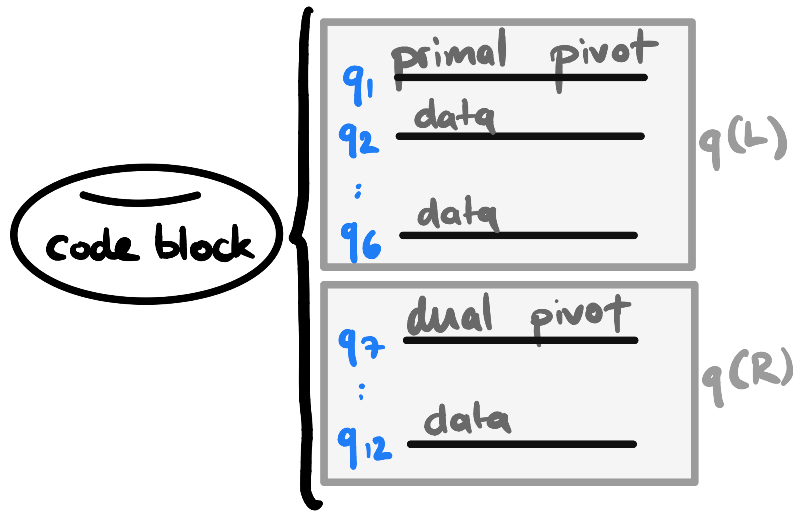
Pauli-based computation (PBC)



Surface code



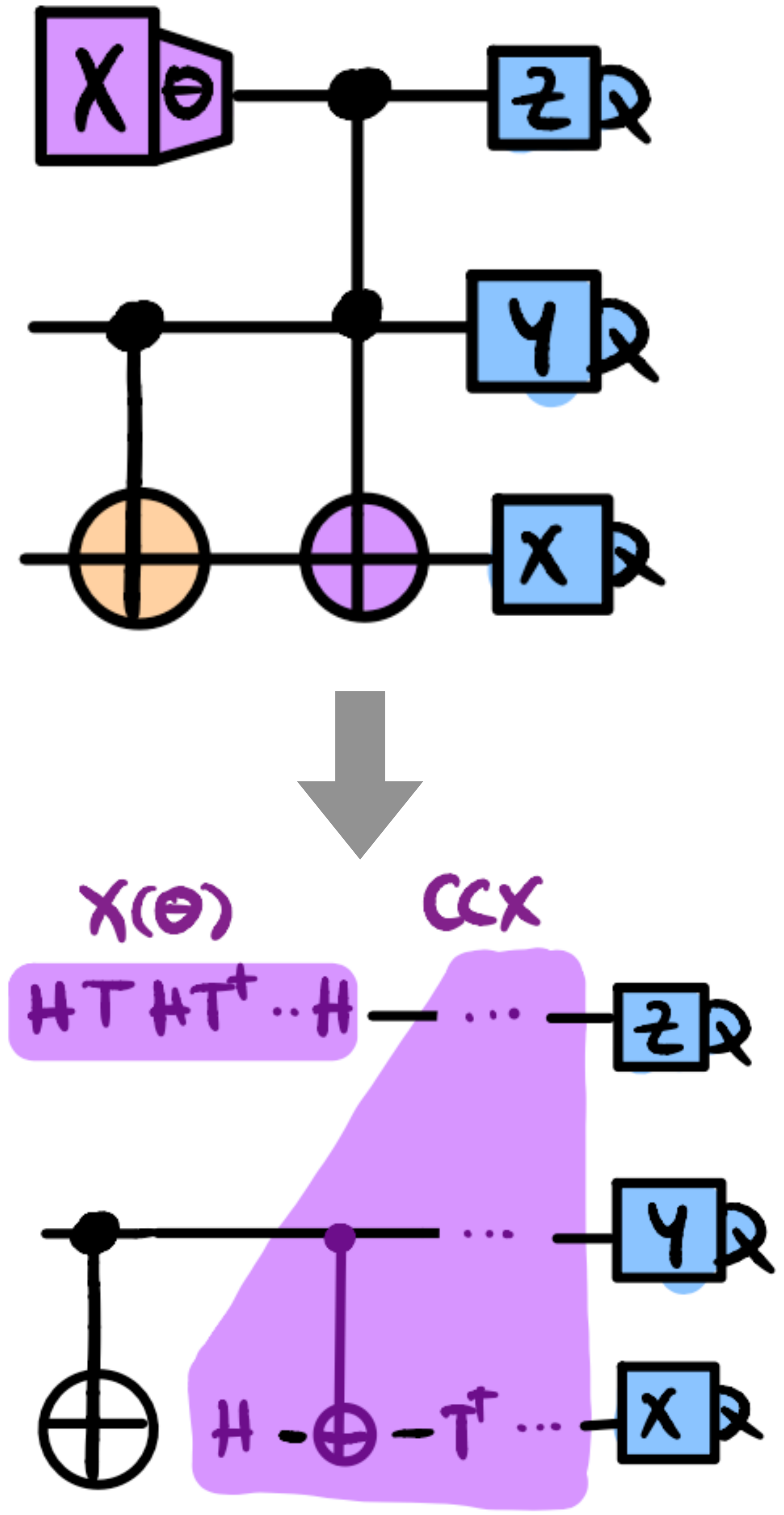
Gross code



["aut", "T", "inter", ..]

Clifford+T

- Universal quantum computation by
  - Clifford gates ($S = R_Z(\pi/2)$, CX, Paulis, …)
  - T gate, $T = R_Z(\pi/4)$
- Any 1-qubit gate can be approximated up to ε by $\mathcal{O}(\log^c(\varepsilon^{-1}))$ Clifford+T
  - Solovay-Kitaev, gridsynth
  - … a very active area of research



Clifford+T

- Qiskit detects a Clifford+T discrete basis gate set
    - use T-count optimization metric
    - use Clifford+T optimization passes
- Plugin-based $R_Z(\theta)$ synthesis algorithms
    - gridsynth: asympotically optimal
    - Solovay-Kitaev: faster runtime

```
from qiskit import (
  QuantumCircuit, generate_preset_pass_manager
)

circuit = QuantumCircuit(2)
circuit.ry(0.123, 0)
circuit.cx(0, 1)
circuit.measure(circuit.qubits, circuit.clbits)

basis = ["t", "h", "s", "cx"]
pm = generate_preset_pass_manager(basis_gates=basis)

discrete = pm.run(circuit)
print(discrete.count_ops())
```

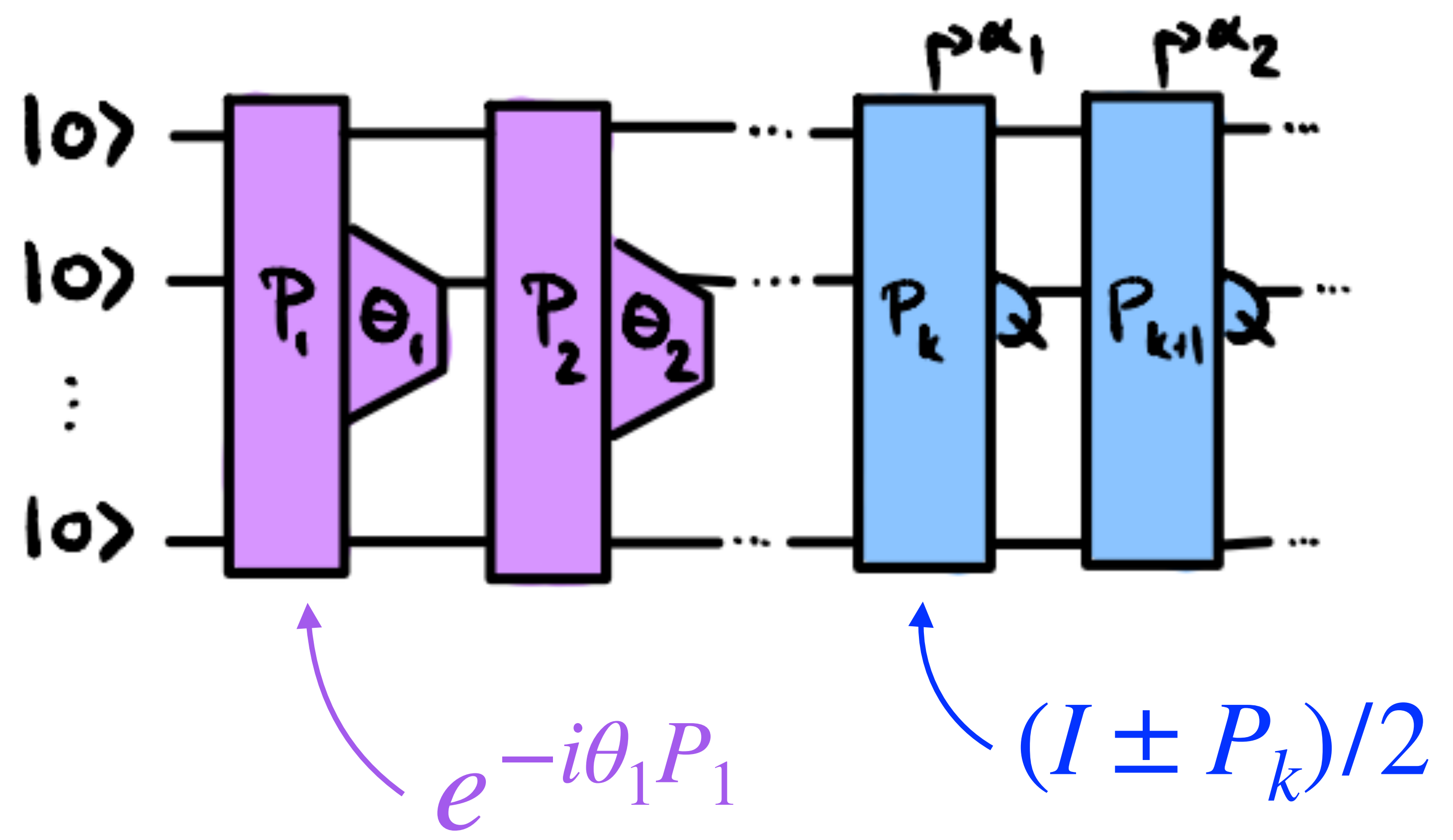
```
{’t’: 35794, ’s': 24210, ‘h': 11921, ‘cx': 1, ‘measure’: 2}
```

```
pm = generate_preset_pass_manager(
  basis_gates=basis,
  unitary_synthesis_method="gridsynth"
)
```

```
{‘h’: 103, ’t': 101, ’s': 58, ‘cx': 1, ‘measure’: 2}
```

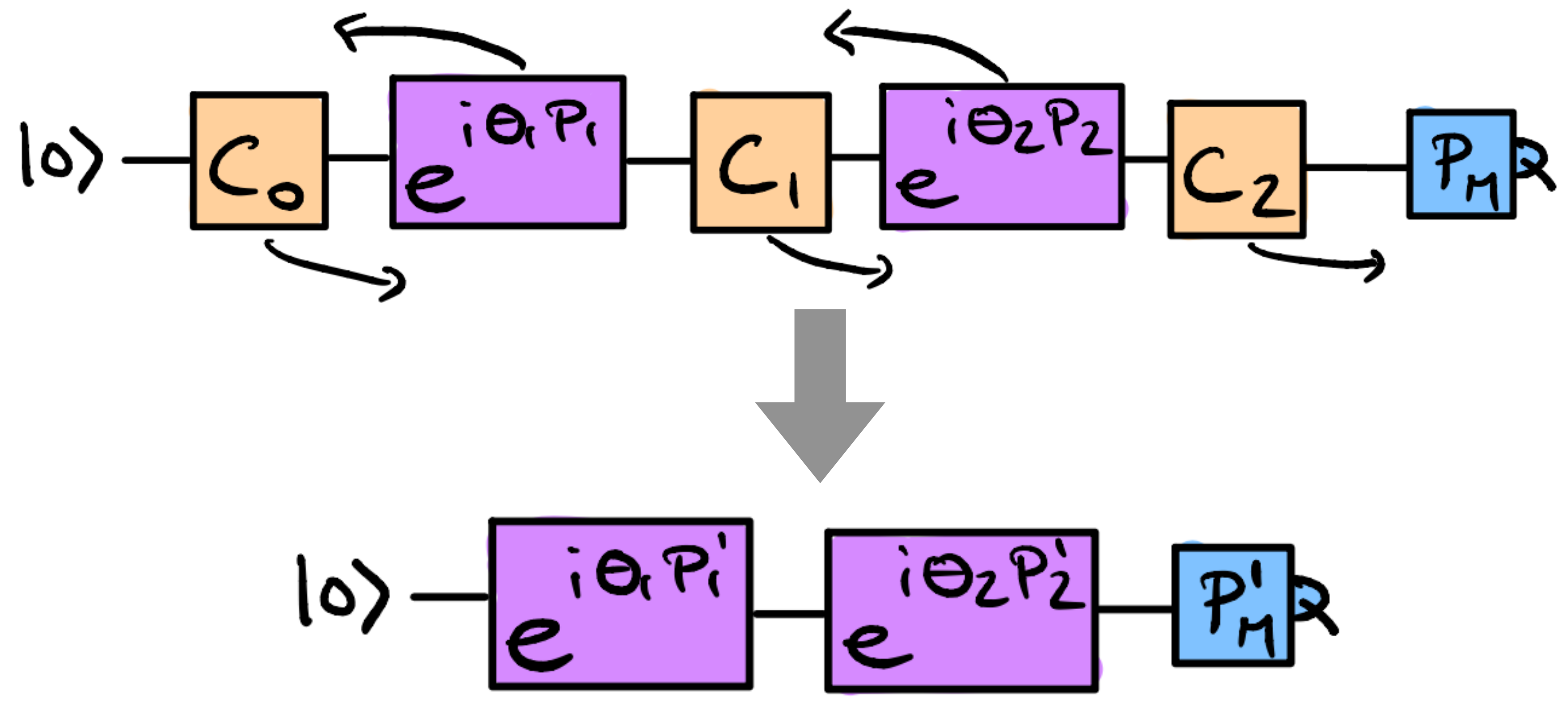
Pauli-based computation (PBC)

- Universal quantum computation by
  - Pauli rotations $\exp(i\theta P)$ for $P \in \{I, X, Y, Z\}^{\otimes n}$
  - nondestructive Pauli measurements $M_P \propto I \pm P$ + classical functions on outcomes $\{\alpha_i\}_i$
- Transformation to PBC is possible e.g. by using the Litinski transformation



$e^{-i\theta_1 P_1}$

$(I \pm P_k)/2$



Bravyi et al. Phys. Rev. X **6**, 021043 (2016)

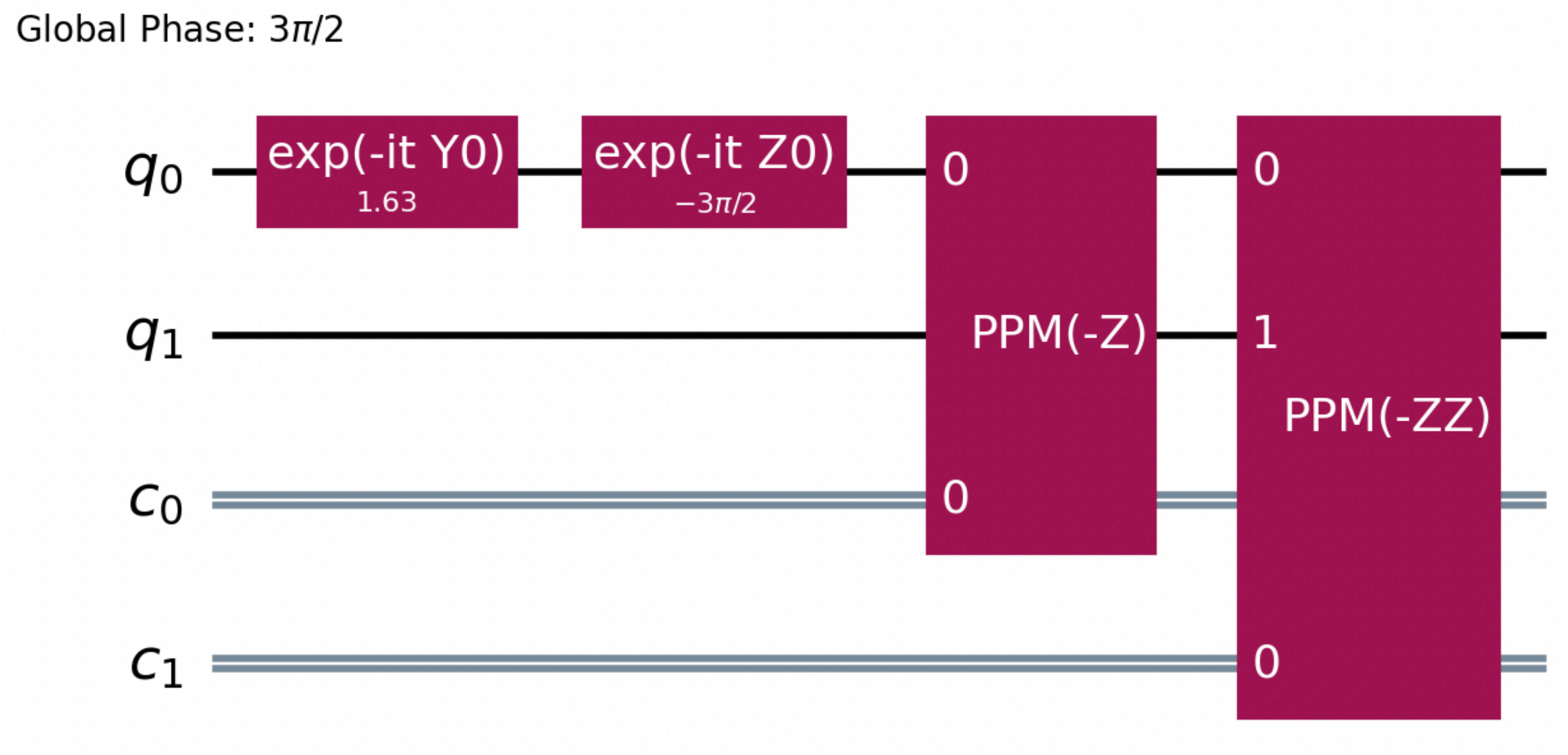
Pauli-based computation (PBC)

Workflow:

- Compile to Clifford+RZ
- Use `LitinskiTransformation` to compile to PBC
- Run Pauli-enabled optimizations, e.g. `CommutativeOptimization`

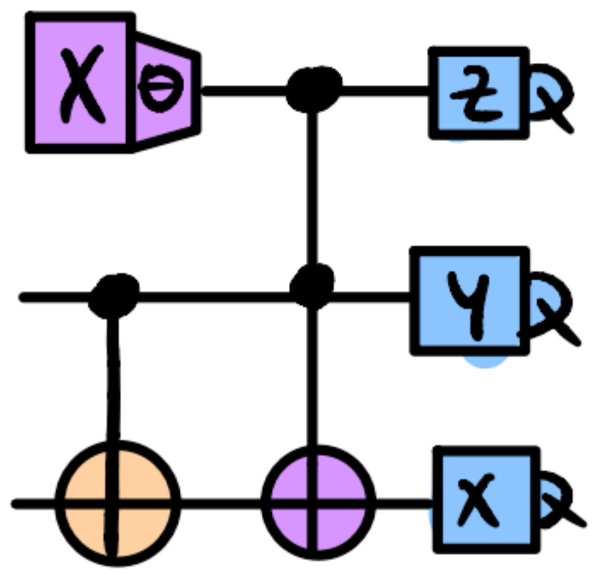
```
from qiskit.transpiler.passes import (
  LitinskiTransformation
)

rz_basis = ["rz", "sx", "x", "cx"]
pm = generate_preset_pass_manager(basis_gates=rz_basis)
pm.optimization.append(
  LitinskiTransformation(fix_clifford=False)
)

pbc = pm.run(circuit)
print(pbc.count_ops())
```

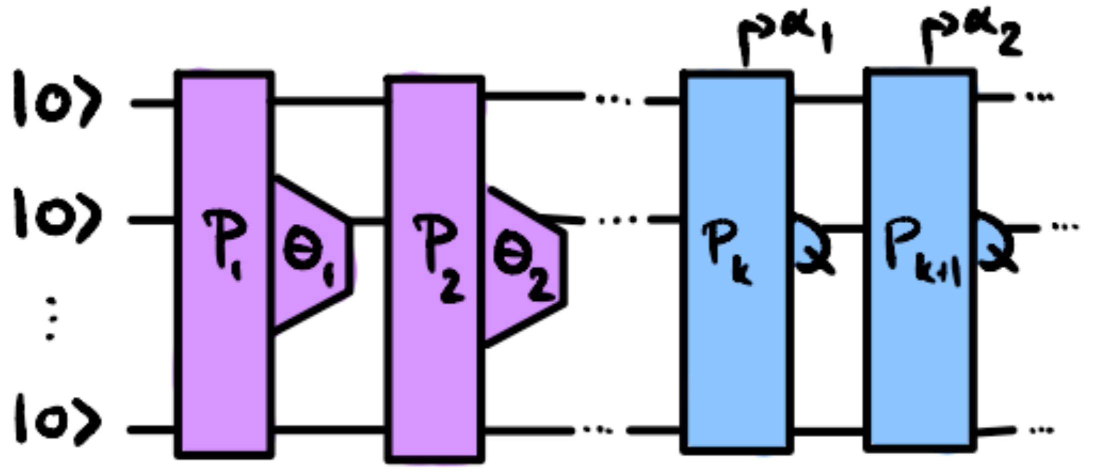
```
{‘PauliEvolution’: 2, ‘pauli_product_measurement': 2}
```

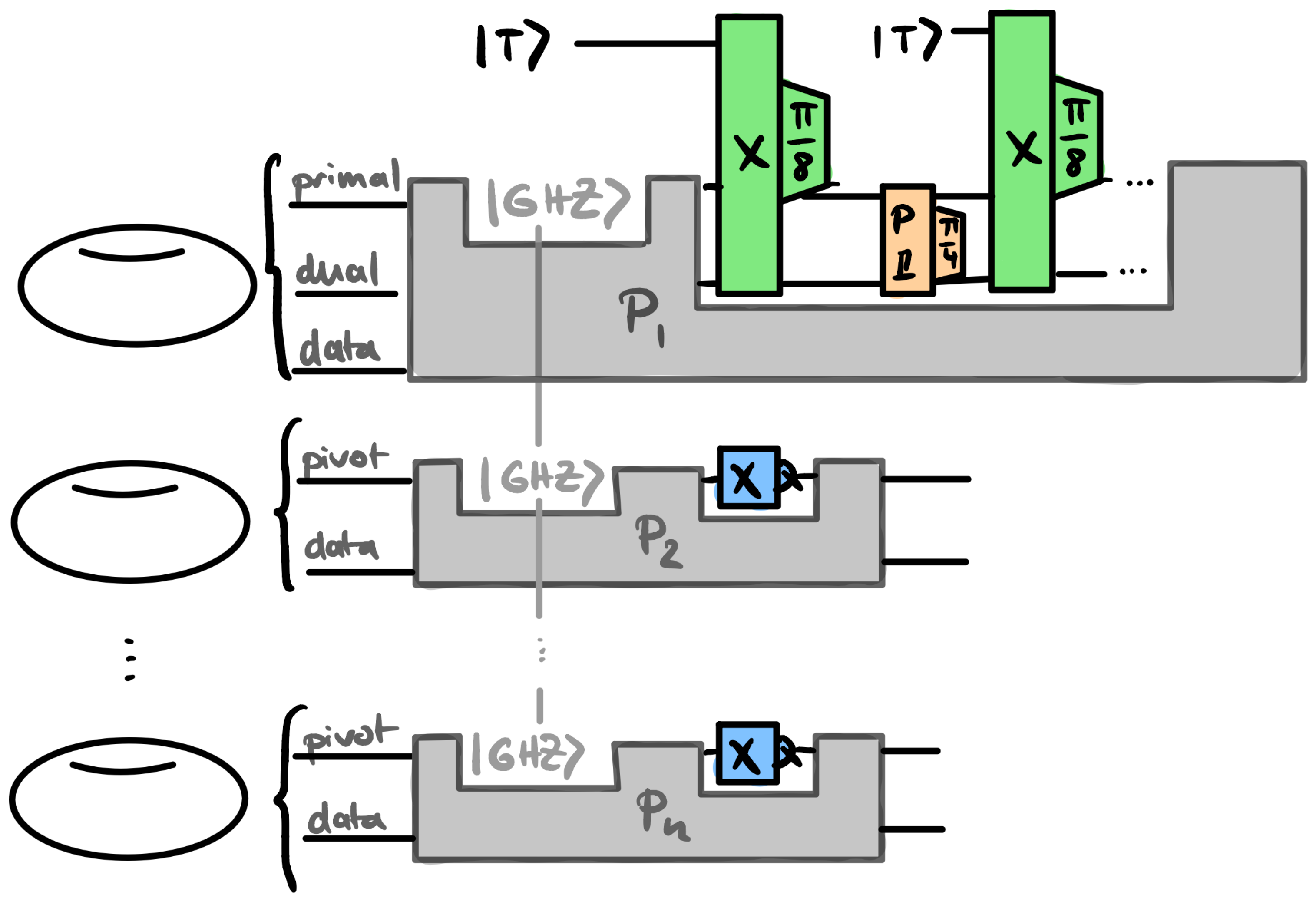


Compiling to bicycle ISA

Logical circuit $\xrightarrow{\text{Qiskit v2.3}}$ **Pauli-based computation** $\xrightarrow{\text{bicycle compiler}}$ **Gross code**



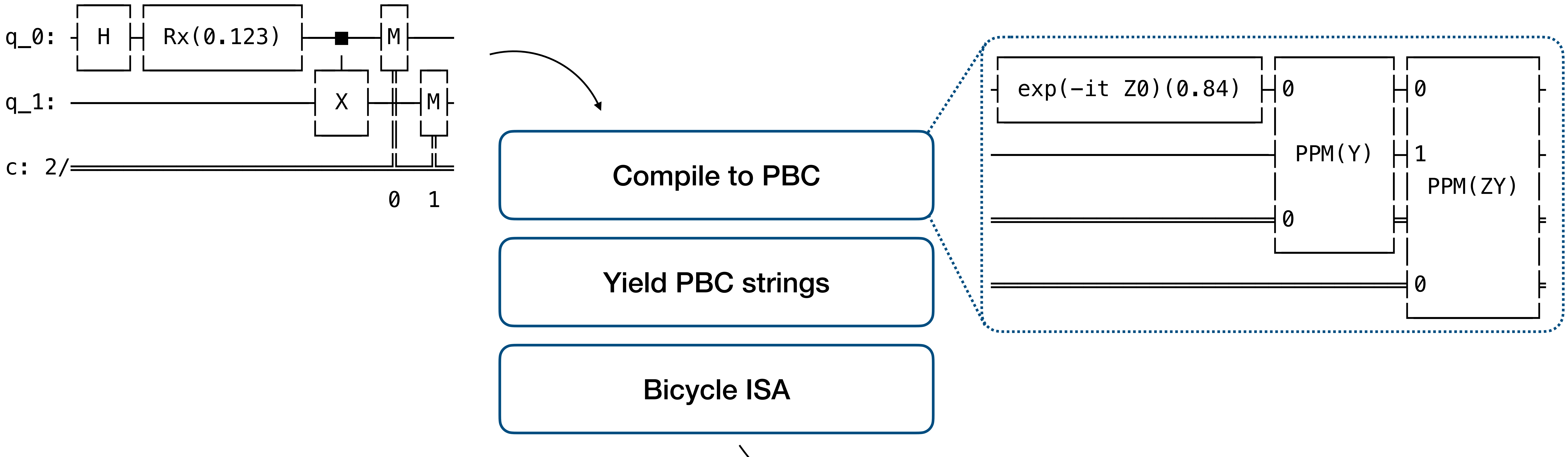




Available online: https://github.com/qiskit-community/bicycle-architecture-compiler

Bell state

A “Hello Gross code” example



Compile to PBC

Yield PBC strings

Bicycle ISA

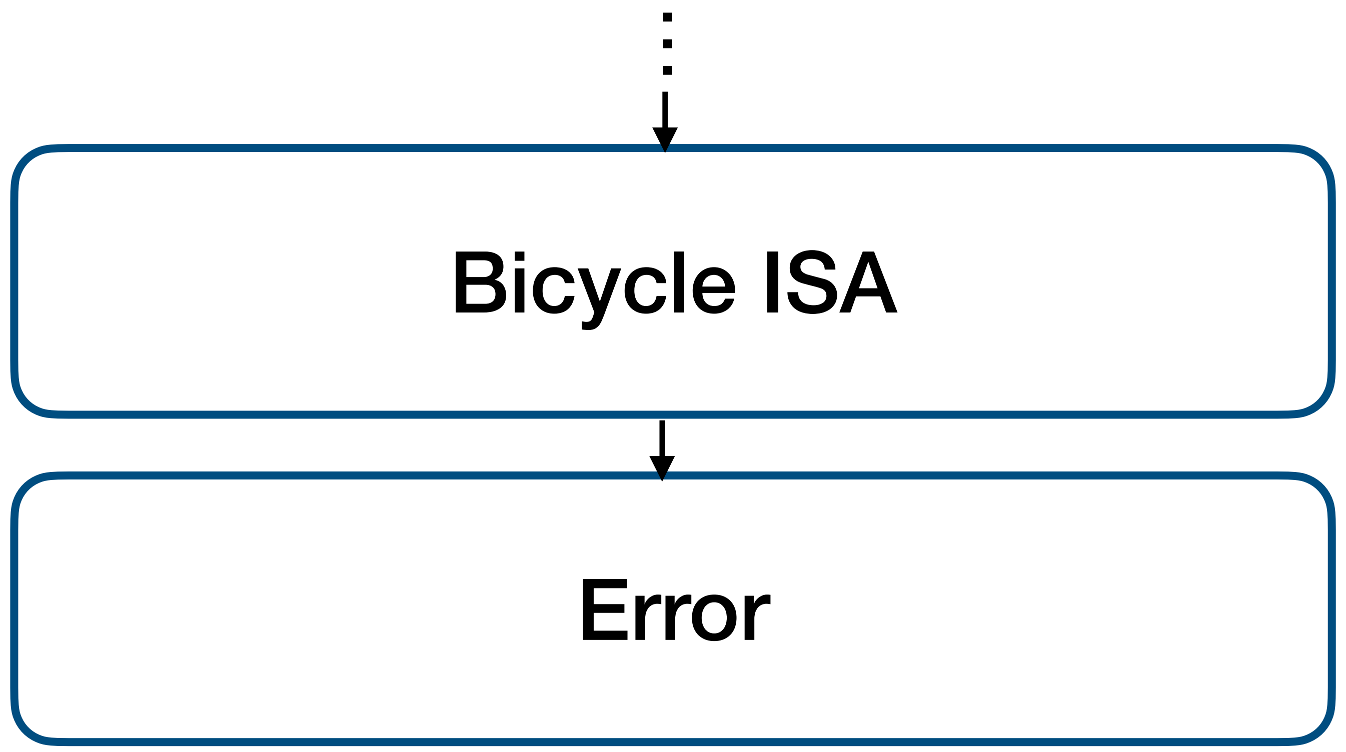
```
87 x T
34 x Automorphism
47 x Measure
```

Compiling a Trotter circuit

Error estimations with the Qiskit pipeline

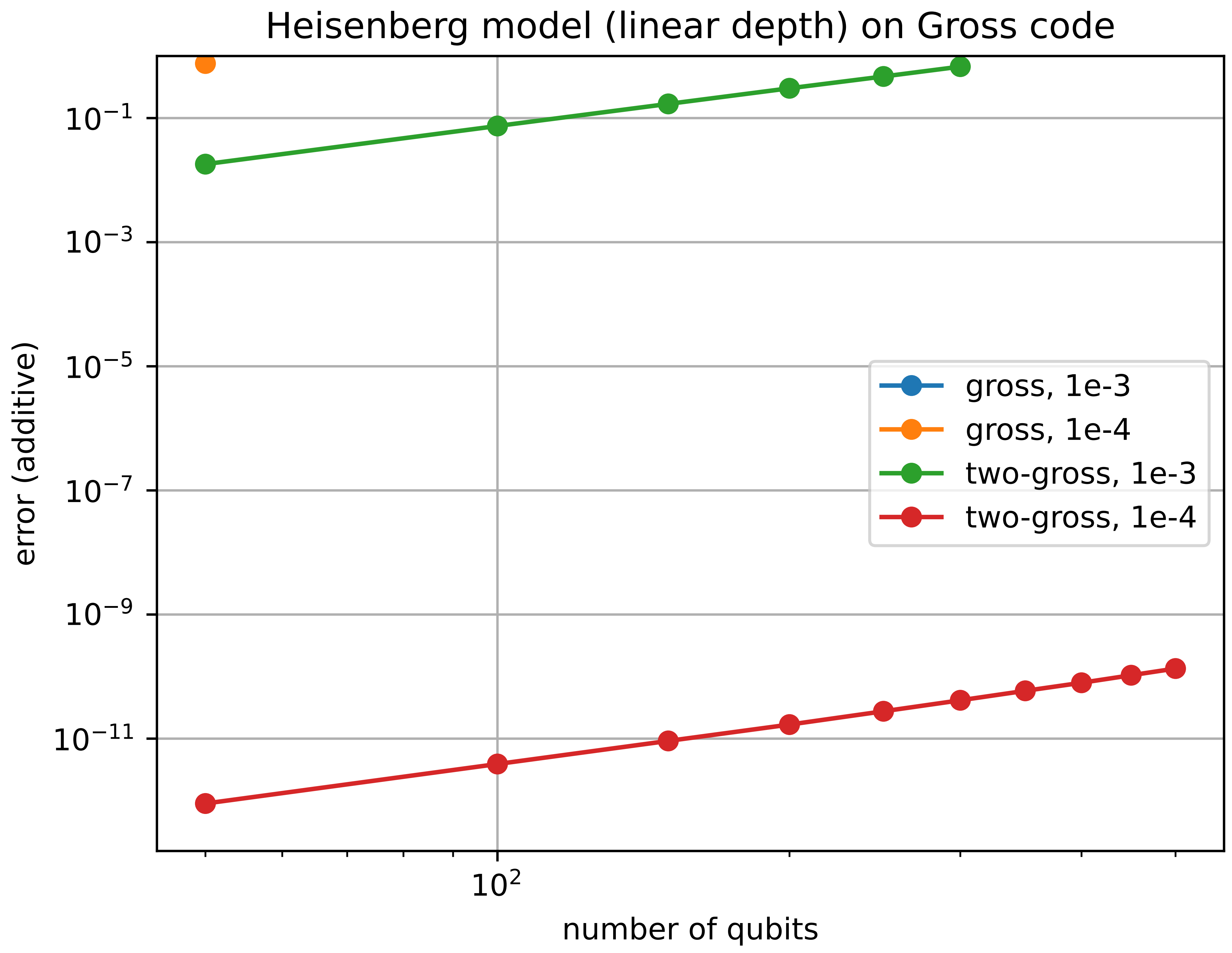
- Use the pipeline for n Trotter steps on an n qubit Heisenberg Hamiltonian

- Feed the bicycle ISA instructions into the `bicycle-numerics` crate



Bicycle ISA

Error



IBM