

Optimisation and verification of quantum circuits with string diagrams

Aleks Kissinger



STRING 3; Birmingham 2019

Quantum software

1. := the code the runs on a quantum computer

factoring



search



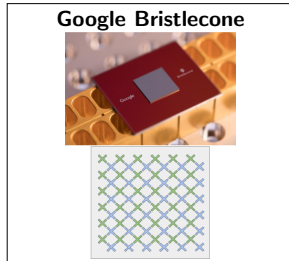
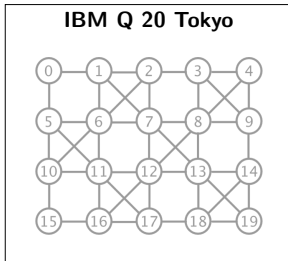
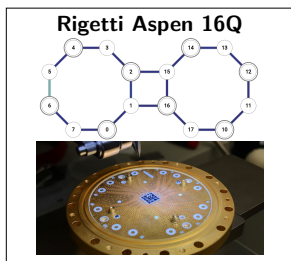
quantum simulation
hidden subgroup
linear systems & codes
pattern matching
collision-finding
network flows
NLP

...

2. := the code that makes that code (better)

- quantum programming languages
- **compilers**
- **optimisation**
- **verification**

Problem: no quantum computers

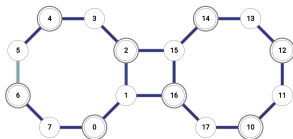


(+ Oxford, Vienna, Delft, Sussex, Grenoble...)

Problem: limited quantum computers

Noisy intermediate-scale quantum (NISQ) devices

- short coherence times (e.g. μs -scale)
- small memory (8-80 qubits)
- noisy primitive operations (gates & measurements)
- limited connectivity:



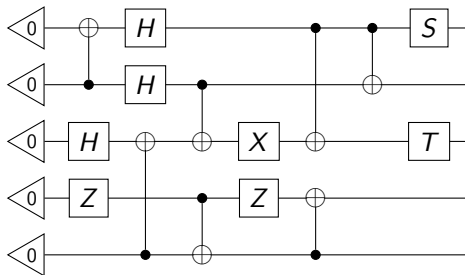
- ...

⇒ **small** advances in software give **big** gains on NISQ hardware!

Quantum circuits

- := the 'assembly language' of quantum computation, e.g.

```
INIT 5
CNOT 1 0
H 2
Z 3
H 0
H 1
CNOT 4 2
...
```

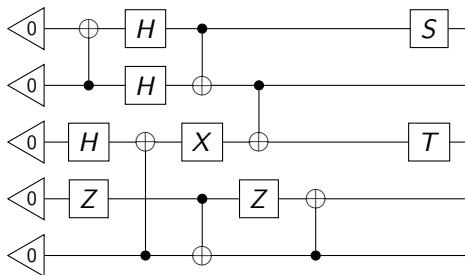


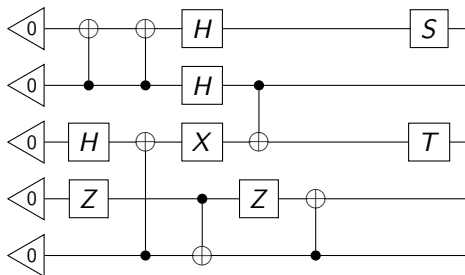
- Built out of fixed set of complex unitary matrices, or *gates*, e.g.

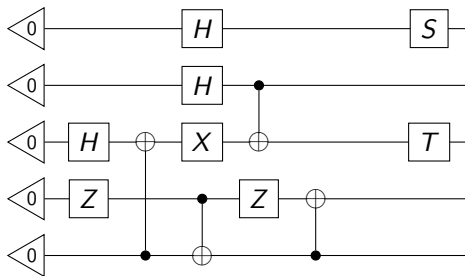
$$\begin{array}{c} \bullet \\ \text{---} \\ | \\ \oplus \\ \text{---} \end{array} := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad \boxed{H} := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \boxed{S} := \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

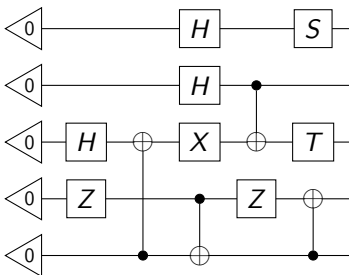
...combined by \otimes and \circ .

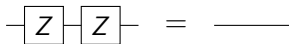
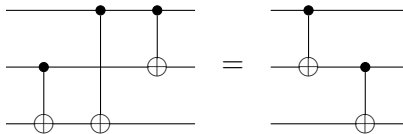
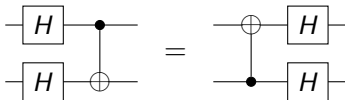
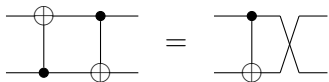
- Universal* sets can gates can be used to construct any $2^n \times 2^n$ unitary matrix
- Gates describe *physical* evolutions of a q. state, which take time and introduce noise
- Lower gate count / gate depth is good!



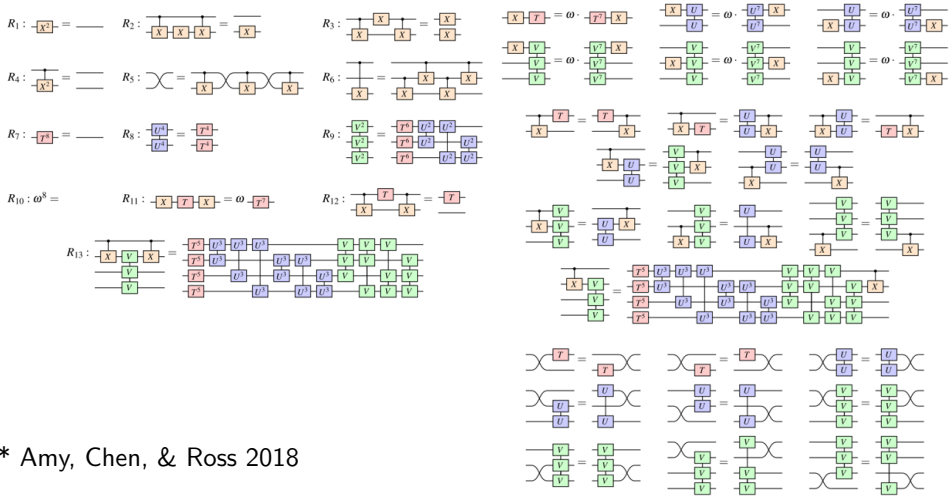




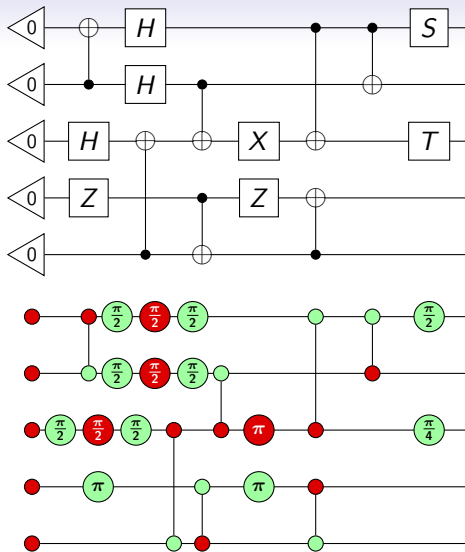


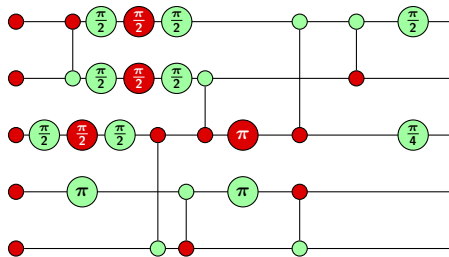
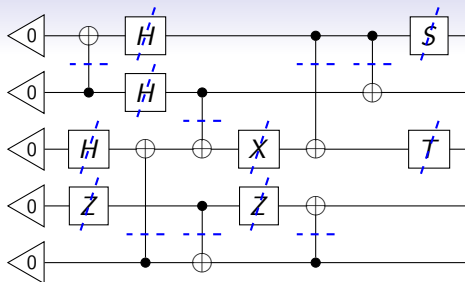


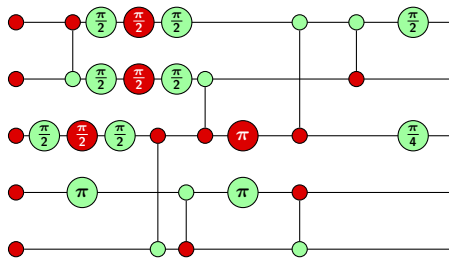
• • •



* Amy, Chen, & Ross 2018



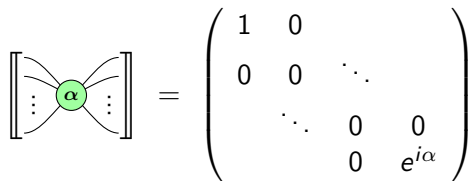




ZX-diagram

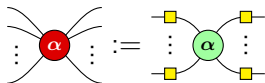
ZX-diagrams

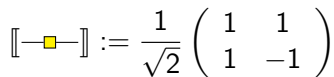
...are made of *spiders*:



A diagram showing a green spider with a central circle labeled α . It has multiple input wires on the left and multiple output wires on the right, with vertical ellipses indicating more wires. This is equated to a matrix:

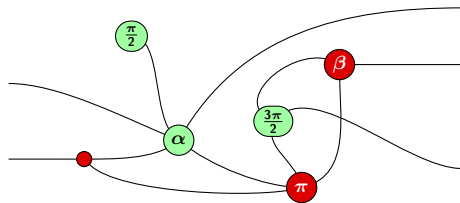
$$= \begin{pmatrix} 1 & 0 & & \\ 0 & 0 & \ddots & \\ & \ddots & 0 & 0 \\ & & 0 & e^{i\alpha} \end{pmatrix}$$

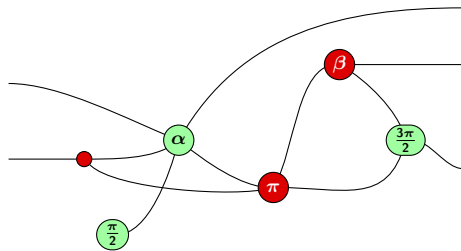


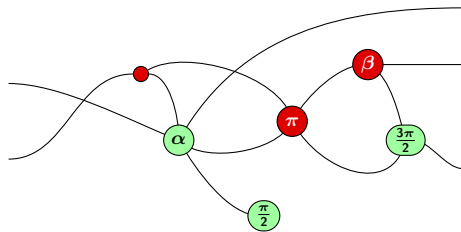


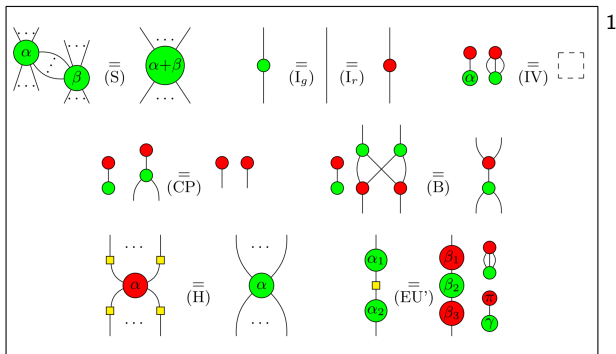
A diagram showing a single yellow square between two horizontal wires. This is equated to a matrix:

$$\llbracket \text{---} \square \text{---} \rrbracket := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$





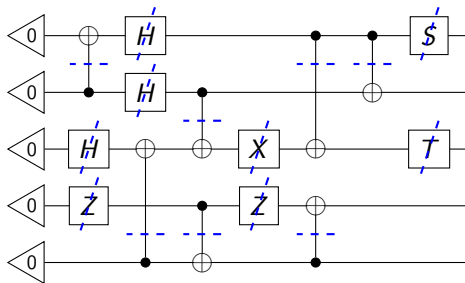


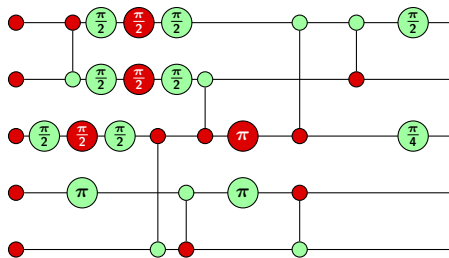


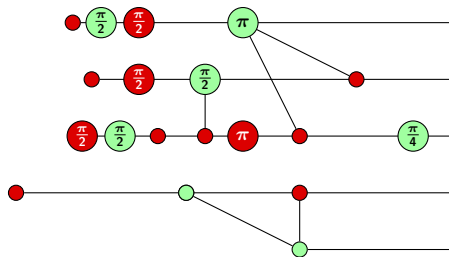
ZX calculus

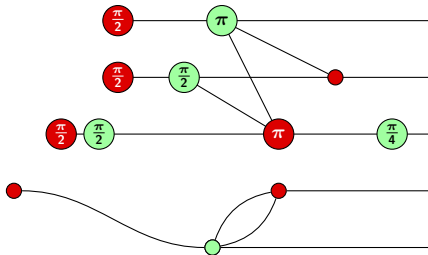
these 8 rules \implies everything before

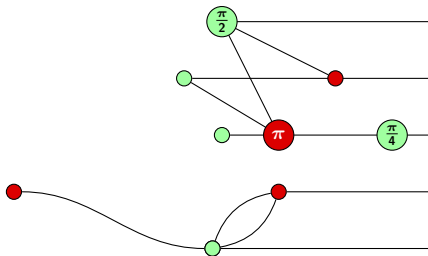
¹Vilmart 2018. arXiv:1812.09114

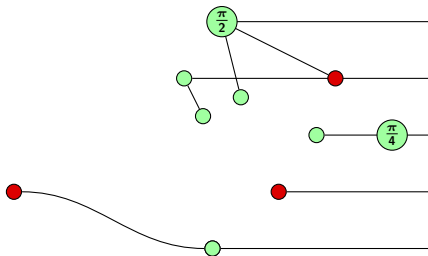


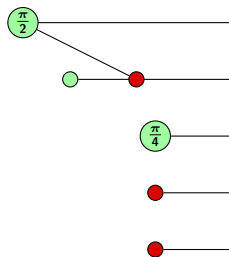












$\frac{\pi}{2}$ —

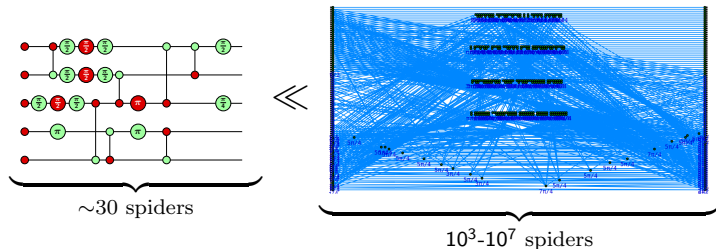
—

$\frac{\pi}{4}$ —

—

—

Q: How do we scale up?



A: Automation.

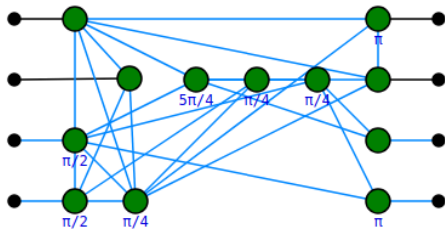


Quantomatic

- General-purpose, **medium scale** tool for teaching and experimenting with graphical calculi

PyZX

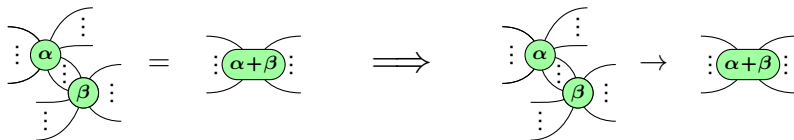
```
In [11]: zx.simplify.clifford_simp(g)
g.normalise()
zx.d3.draw(g)
```



- Special-purpose, **large scale** circuit optimisation tool for ZX-calculus

The idea

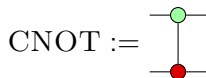
1. Turn *equations* into directed *rewrite rules*



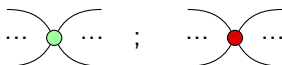
2. Focus on *tractible* fragments of the language.

CNOT circuits

- Built from:



- Efficient to classically simulate (\mathbb{Z}^2 -linear maps)
- Always equal to a circuit of poly depth ($O(n^2)$ -bounded)
- Expressible using *phase-free ZX-diagrams*



Clifford circuits

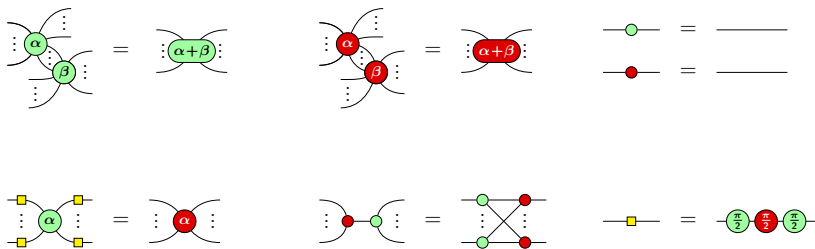
- Built from:

$$\text{CNOT} := \begin{array}{c} \text{---} \circ \text{---} \\ | \\ \text{---} \bullet \text{---} \end{array} \quad H := \text{---} \square \text{---} \quad S := \text{---} \circ \left(\frac{\pi}{2} \right) \text{---}$$

- Efficient to classically simulate (Gottesman-Knill theorem)
- Always equal to a circuit of poly depth ($O(n^2)$ -bounded)
- Expressible by *Clifford ZX-diagrams*

$$\dots \circ \left(\frac{k\pi}{2} \right) \dots \quad ; \quad \dots \bullet \left(\frac{k\pi}{2} \right) \dots \quad k \in \mathbb{Z}$$

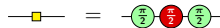
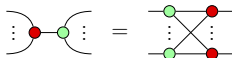
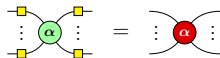
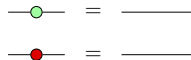
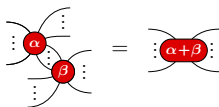
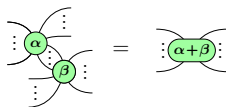
Clifford ZX-calculus



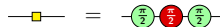
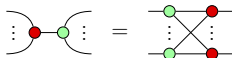
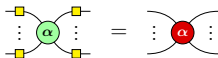
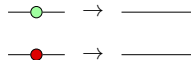
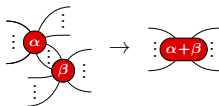
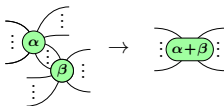
complete for Clifford ZX-diagrams

Equations \rightsquigarrow Rewrite rules

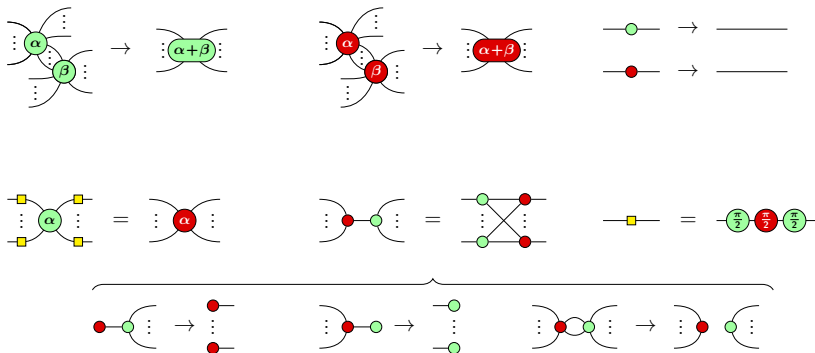
Equations \rightsquigarrow Rewrite rules



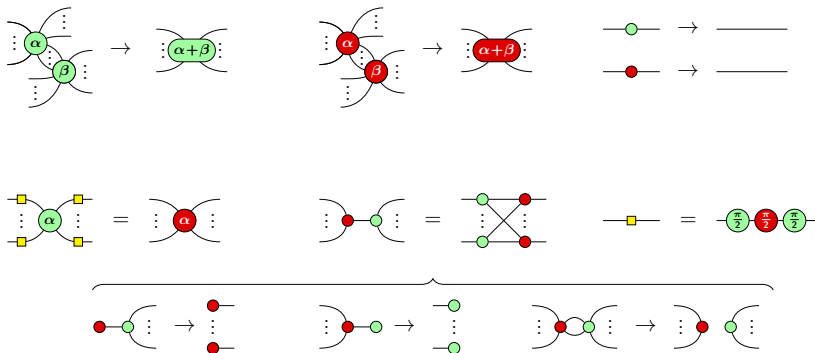
Equations \rightsquigarrow Rewrite rules



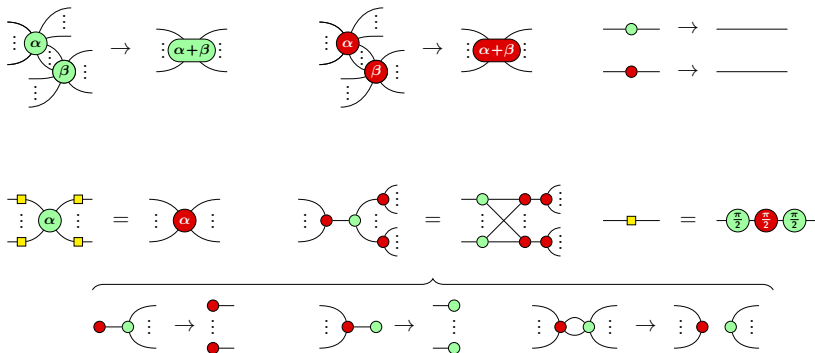
Equations \rightsquigarrow Rewrite rules



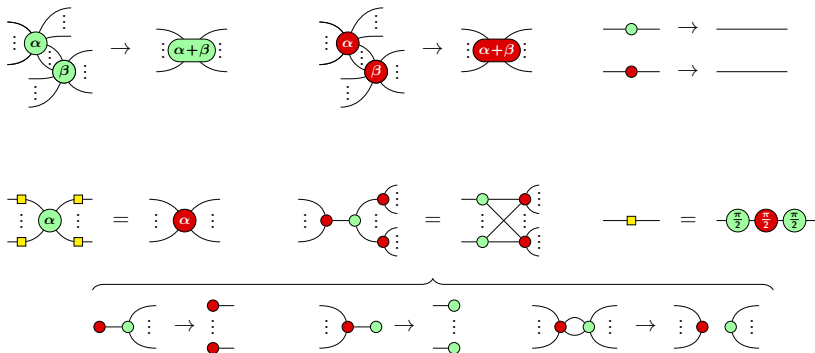
Equations \rightsquigarrow Rewrite rules



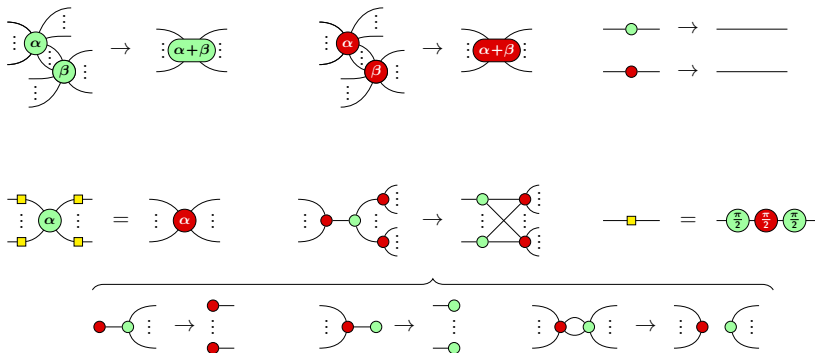
Equations \rightsquigarrow Rewrite rules



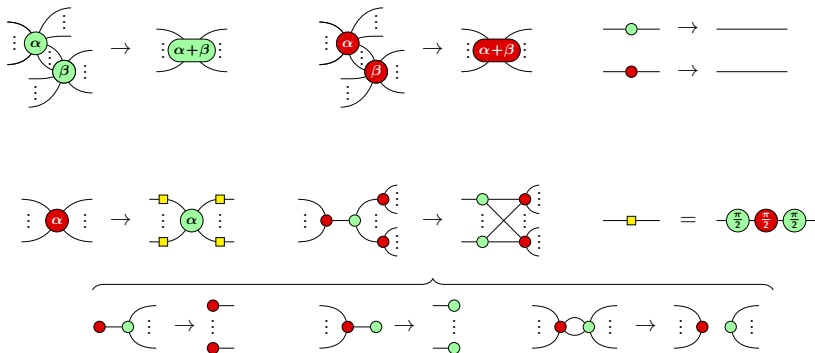
Equations \rightsquigarrow Rewrite rules

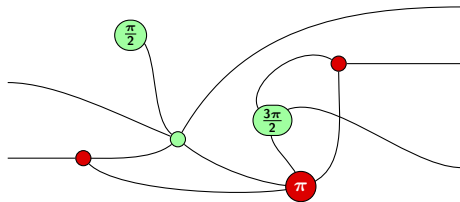


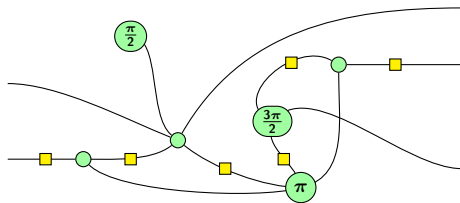
Equations \rightsquigarrow Rewrite rules

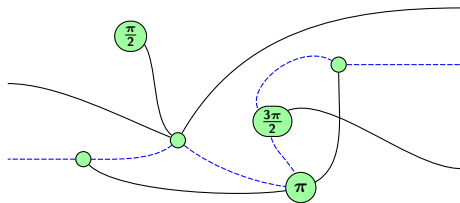


Equations \rightsquigarrow Rewrite rules

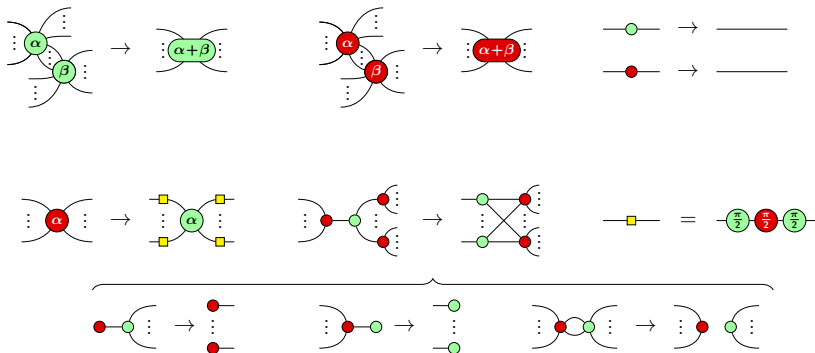




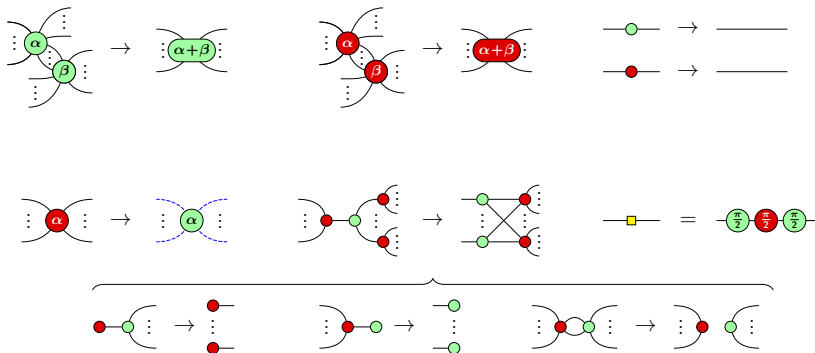




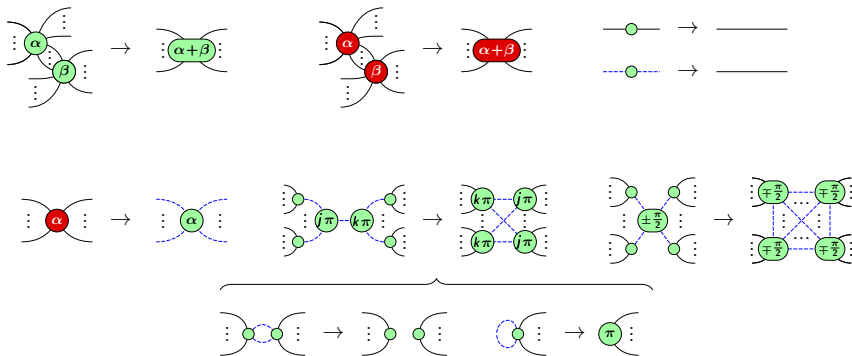
Equations \rightsquigarrow Rewrite rules



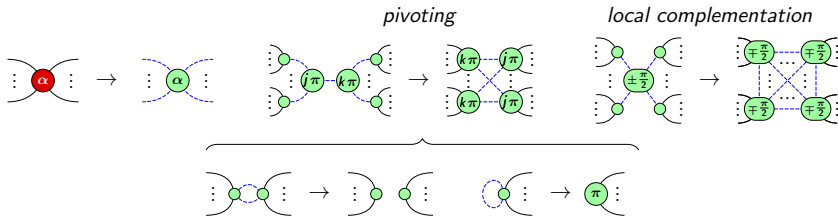
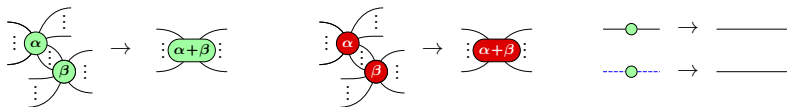
Equations \rightsquigarrow Rewrite rules



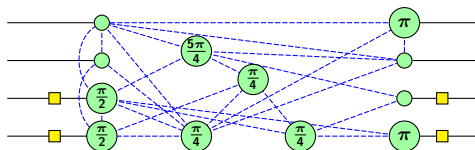
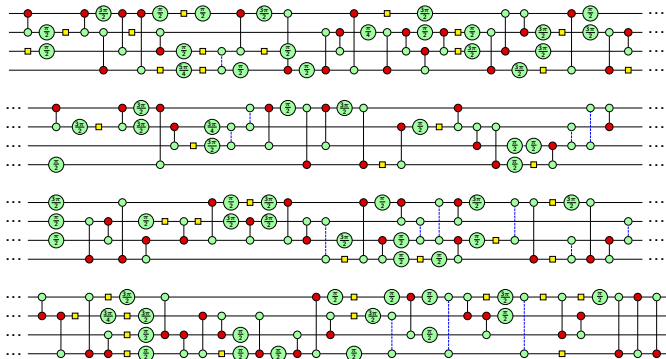
Equations \rightsquigarrow Rewrite rules

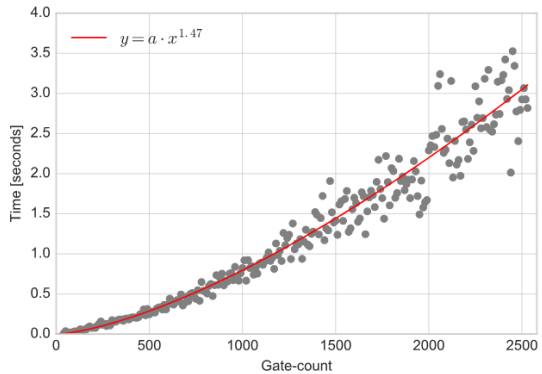


Equations \rightsquigarrow Rewrite rules

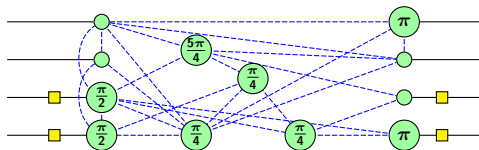


Computing the skeleton





ZX-diagram \rightsquigarrow circuit?



- ZX-diagrams can represent any linear map, but circuits are always unitary
- \implies not all ZX-diagrams are equal to circuits

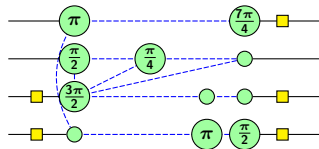
Problem (Circuit extraction problem)

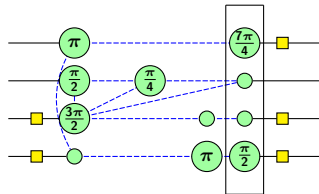
Given a ZX-diagram that represents a unitary linear map, find an equivalent quantum circuit.

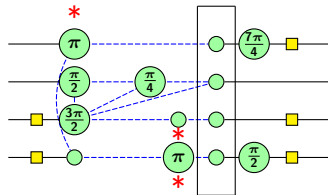
Conjecture

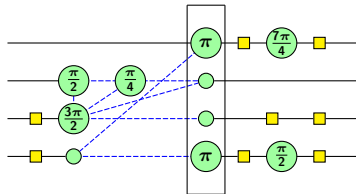
If the circuit extraction problem is solvable with polynomial overhead, then $BQP = PostBQP$.

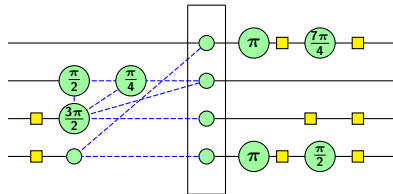
\implies need heuristics or **extra data**

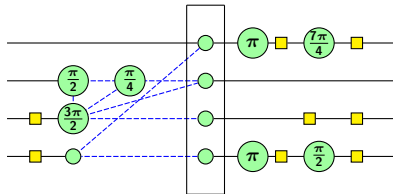




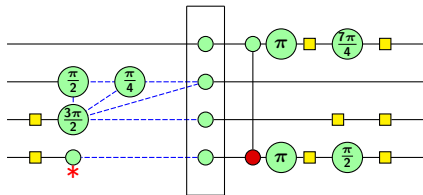




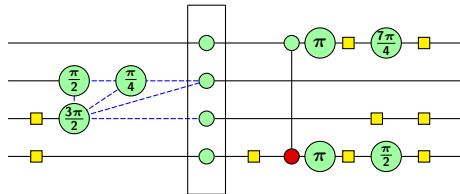


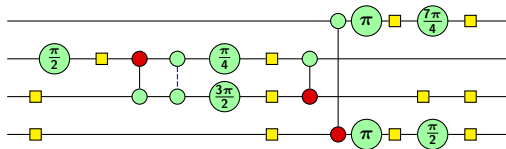


$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \xrightarrow{R_1 := R_1 + R_4} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



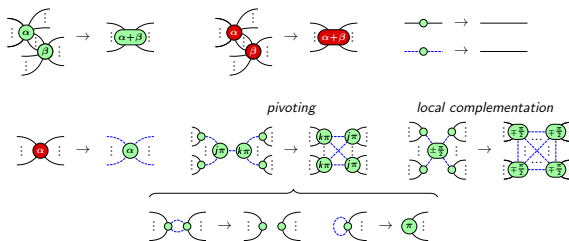


Extraction and gFlow

- Strategy always works iff the graph of a ZX-diagram has *generalised flow* (gFlow)

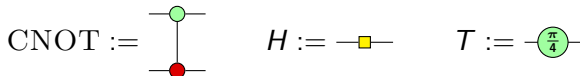
$$\exists \prec, g. \forall v. g(v) \subseteq v^\succ \wedge \mathbf{Odd}(v) \cap \bar{I} \cap v^\prec = \{v\}$$

- **Easy:** quantum circuits have gFlow
- **Harder:** these rules all preserve gFlow:



Clifford+T circuits

- Built from:



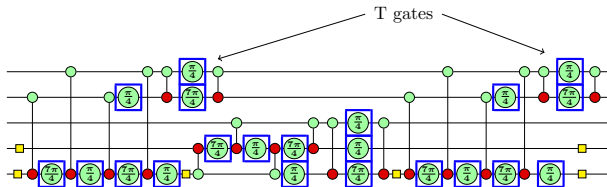
- Contains Clifford circuits ($S = T^2$)
- Universal** for quantum computation

$$\forall U \in \mathcal{C}. \llbracket C \rrbracket \approx U$$

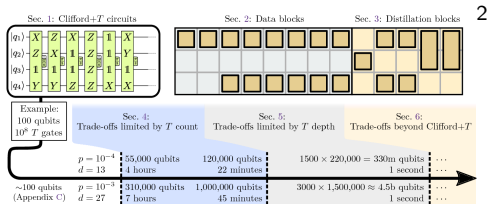
- \implies probably NOT efficient to simulate classically

T-count reduction

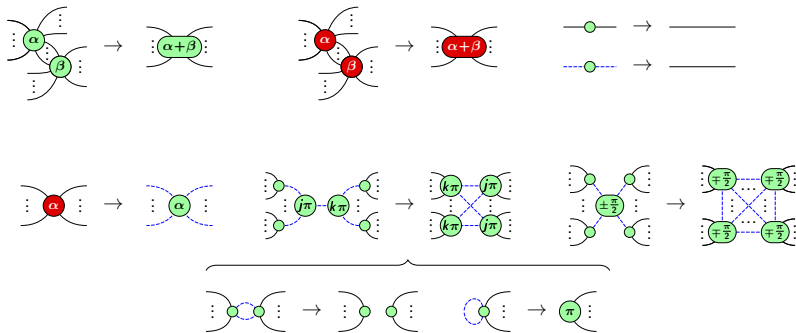
- := reducing the number of T -gates in a circuit

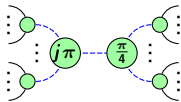


- **Q:** Why T gates?
- **A:** $\sim 100X$ more overhead in **fault-tolerant quantum computing**



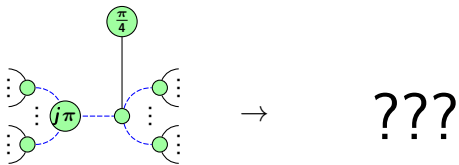
²A Game of Surface Codes. Litinski 2019

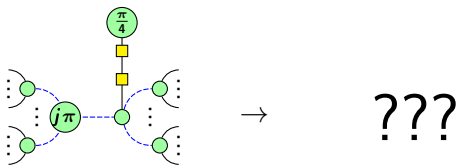


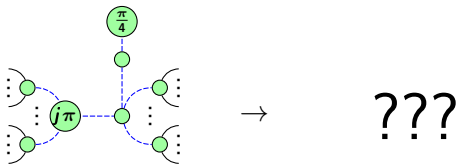


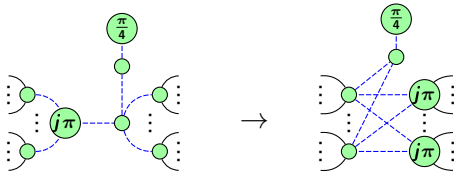
→

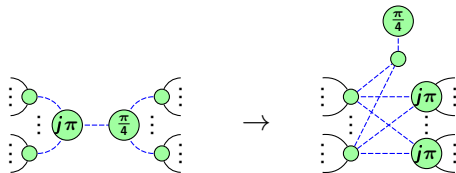
???

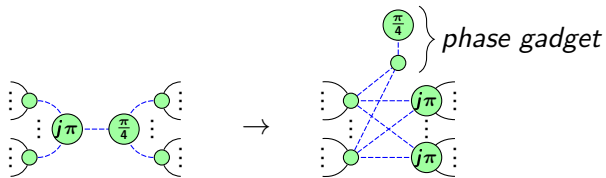


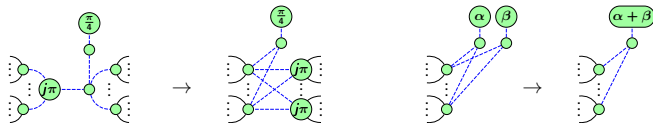
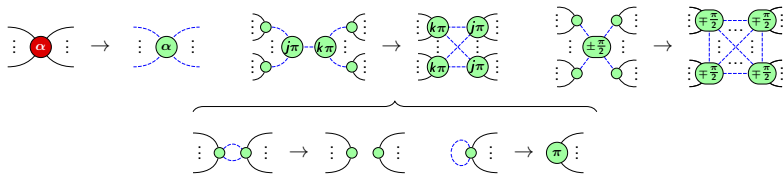
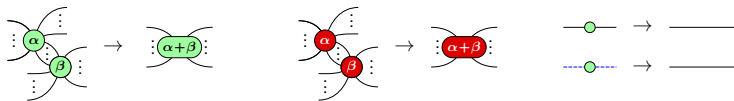




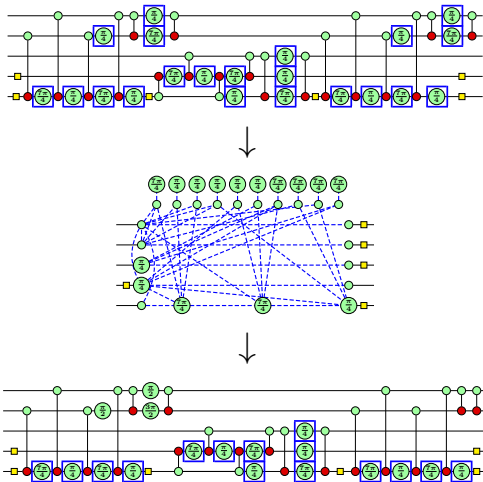








T-count reduction

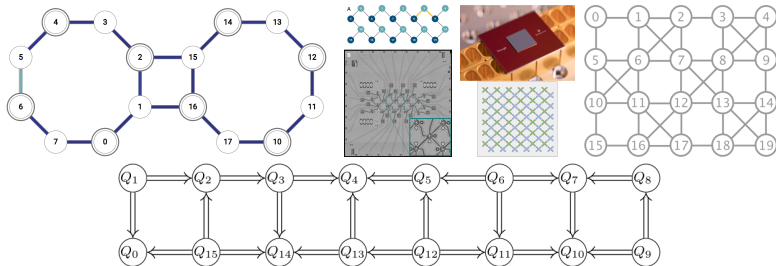


T-count reduction

Circuit	n	T	Best prev.	Method	PyZX
adder₈	24	399	213	RM_m	173
Adder8	23	266	56	NRSCM	56
Adder16	47	602	120	NRSCM	120
Adder32	95	1274	248	NRSCM	248
Adder64	191	2618	504	NRSCM	504
barenco-tof3	5	28	16	Tpar	16
barenco-tof4	7	56	28	Tpar	28
barenco-tof5	9	84	40	Tpar	40
barenco-tof10	19	224	100	Tpar	100
tof ₃	5	21	15	Tpar	15
tof ₄	7	35	23	Tpar	23
tof ₅	9	49	31	Tpar	31
tof ₁₀	19	119	71	Tpar	71
<i>csla-mux₃</i>	15	70	58	RM _r	62
<i>csum-mux₉</i>	30	196	76	RM _r	84
cycle17₃	35	4739	1944	RM_m	1797
<i>gf(2⁴)-mult</i>	12	112	56	TODD	68
<i>gf(2⁵)-mult</i>	15	175	90	TODD	115
<i>gf(2⁶)-mult</i>	18	252	132	TODD	150
<i>gf(2⁷)-mult</i>	21	343	185	TODD	217
<i>gf(2⁸)-mult</i>	24	448	216	TODD	264
ham15-low	17	161	97	Tpar	97
ham15-med	17	574	230	Tpar	212
ham15-high	20	2457	1019	Tpar	1019
hwb ₆	7	105	75	Tpar	75
hwb₈	12	5887	3531	RM_{mkr}	3517
<i>mod-mult-55</i>	9	49	28	TODD	35
mod-red-21	11	119	73	Tpar	73
mod5 ₄	5	28	16	Tpar	8
nth-prime₆	9	567	400	RM_{mkr}	279
<i>nth-prime₈</i>	12	6671	4045	RM _{mkr}	4047
qcla-adder ₁₀	36	589	162	Tpar	162
<i>qcla-com₇</i>	24	203	94	RM _m	95
qcla-mod ₇	26	413	235	NRSCM	237
rc-adder ₆	14	77	47	RM _{mkr}	47
vbe-adder ₃	10	70	24	Tpar	24

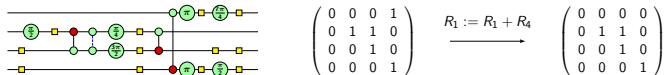
- Compared with state of the art for (ancilla-free) T-count reduction:
 - Amy, Maslov, Mosca. 2014
 - Amy, Mosca. 2016
 - Heyfron, Campbell. 2018
 - Nam *et al.* 2018
- As of March 2019:
 - 26/36 \approx 72% **match** SotA
 - 6/36 \approx 17% **beat** SotA
- In April 2019, Zhang and Chen matched PyZX numbers

Circuit routing

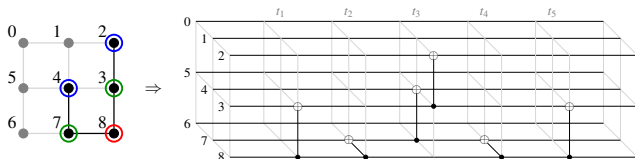


Circuit routing

PyZX uses Gaussian elimination to extract circuits:



By constrained GE, we produce **routed** circuits:



Circuit routing

Architecture	CNOT	QuilC	PyZX	Savings
9q-square	3	3.8	3	21.05%
9q-square	5	10.82	5.2	51.94%
9q-square	10	20.08	11.6	42.23%
9q-square	20	46.24	25.85	44.10%
9q-square	30	72.89	35.55	51.23%
16q-square	4	6.14	4.44	27.69%
16q-square	8	19.68	12.41	36.94%
16q-square	16	48.13	33.08	31.27%
16q-square	32	106.75	82.95	22.30%
16q-square	64	225.69	147.38	34.70%
16q-square	128	457.35	168.12	63.24%
16q-square	256	925.85	169.28	81.72%
rigetti-16q-aspen	4	7.05	4.15	41.13%
rigetti-16q-aspen	8	28.2	11.22	60.21%
rigetti-16q-aspen	16	69.15	33.95	50.90%
rigetti-16q-aspen	32	147.3	101.75	30.92%
rigetti-16q-aspen	64	324.6	189.15	41.73%
rigetti-16q-aspen	128	664.65	220.75	66.79%
rigetti-16q-aspen	256	1367.89	222.15	83.76%
ibm-qx5	4	6.75	4	40.74%
ibm-qx5	8	23.7	8.95	62.24%
ibm-qx5	16	60.5	26.55	56.12%
ibm-qx5	32	140.05	84.4	39.74%
ibm-qx5	64	301.05	152.65	49.29%
ibm-qx5	128	600.9	188.25	68.67%
ibm-qx5	256	1247.8	193.8	84.47%
ibm-q20-tokyo	4	5.5	4	27.27%
ibm-q20-tokyo	8	17.3	7.69	55.55%
ibm-q20-tokyo	16	43.83	20.44	53.37%
ibm-q20-tokyo	32	93.58	66.93	28.48%
ibm-q20-tokyo	64	215.9	165.6	23.30%
ibm-q20-tokyo	128	432.65	237.64	45.07%
ibm-q20-tokyo	256	860.74	245.84	71.44%

- up to 5X more efficient vs. Rigetti, CQC, IBM

Verification

- **Q:** How do we verify correctness of optimised circuits?

$$C \rightsquigarrow D \quad \Longrightarrow \quad \underbrace{[[C]] = [[D]]}_{\text{QMA-complete}}$$

- PyZX optimiser is self-checking

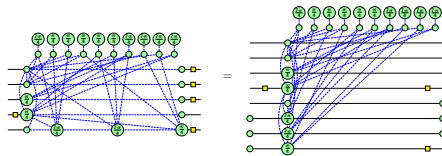
$$C \rightsquigarrow D \quad \Longrightarrow \quad \underbrace{C; D^{-1} \rightsquigarrow 1}_{\sim O(n^{1.5})}$$

- *Sound* approx. of circuit equality helps **find bugs**:

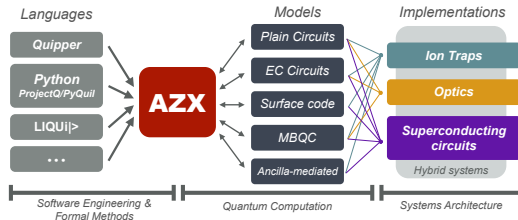
qcla-com ₇	24	203	94	KM _m	95
qcla-mod ₇	26	413	235(X)	NRSCM	237
rc-adder ₆	14	77	17	RM _n	17

Where to?

- More aggressive optimisation \rightsquigarrow more difficult circuit extraction



- Circuit routing for general circuits
- Quantum static analysis
 - PyZX simplifier can prove “=”
 - Alternative models can prove “ \neq ”
 - Functorial semantics \rightsquigarrow ‘abstract interpretation’ for string diagrams
- Smart compilers



Thanks!

- *Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus*. Ross Duncan, Simon Perdrix, AK, John van de Wetering. [arXiv:1902.03178](https://arxiv.org/abs/1902.03178) [quant-ph]
- *Reducing T-count with the ZX-calculus*. AK, John van de Wetering [arXiv:1903.10477](https://arxiv.org/abs/1903.10477) [quant-ph]
- *CNOT circuit extraction for topologically-constrained quantum memories*. **AK**, Arianne Meijer-van de Griend [arXiv:1903.00633](https://arxiv.org/abs/1903.00633) [quant-ph]

Quantomatic: quantomatic.github.io

PyZX: github.com/Quantomatic/pyzx