# An Introduction to the ZX-Calculus
# ...and some applications in quantum software

Aleks Kissinger

UNIVERSITY OF OXFORD

*Full-stack quantum computing 2020*

# Quantum software

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

*factoring*

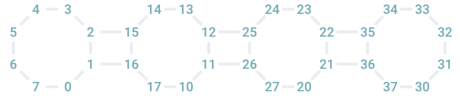

*search*


*physical simulation*
*optimisation problems*
*linear systems & codes*
*network flows*
*natural language processing*
*...*

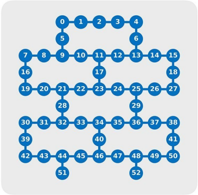
2. := the code that makes that code (better)
   - **compilers**
   - **optimisation**
   - **verification**

# **Problem:** ~~no quantum computers~~

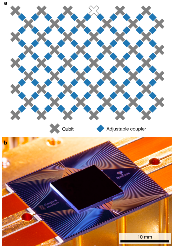
**Rigetti Aspen**




**IBM Q Rochester**




**Google Sycamore**




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

# **Problem:** limited quantum computers

NISQ devices have:

- short coherence times
- low numbers of qubits
- noisy operations
- limited connectivity
- ...

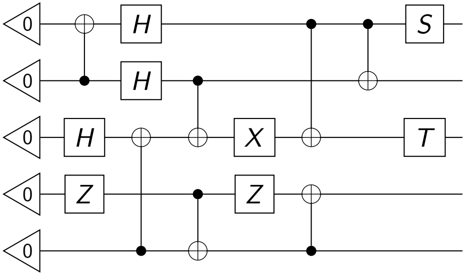
$\Rightarrow$ **small** advances in software give **big** gains on NISQ hardware!

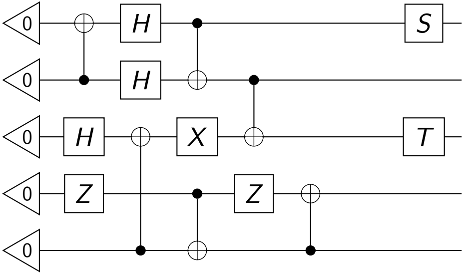
## Quantum circuits

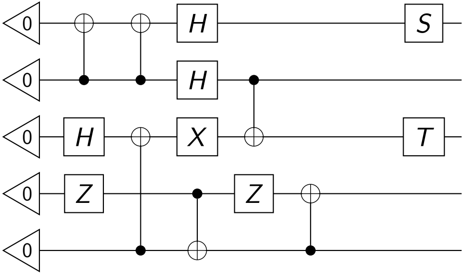
- := the 'assembly language' of quantum computation, e.g.

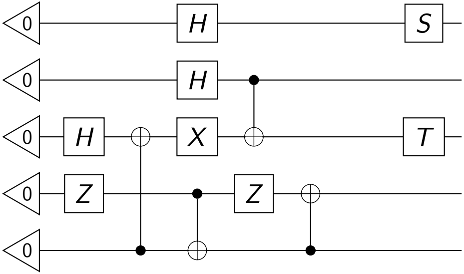
```
INIT 5
CNOT 1 0
H 2
Z 3
H 0
H 1
CNOT 4 2
...
```

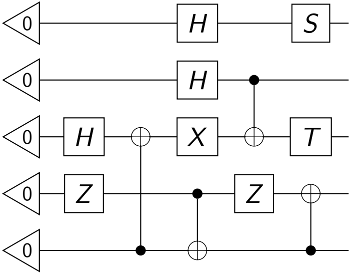
$\Leftrightarrow$

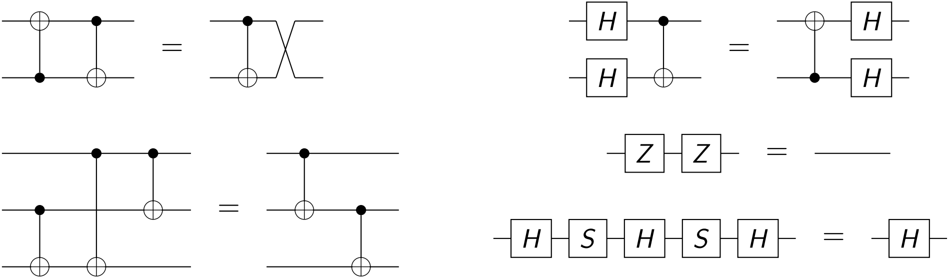



0
0
0
0
0
H
H
H
S
X
T
Z
Z

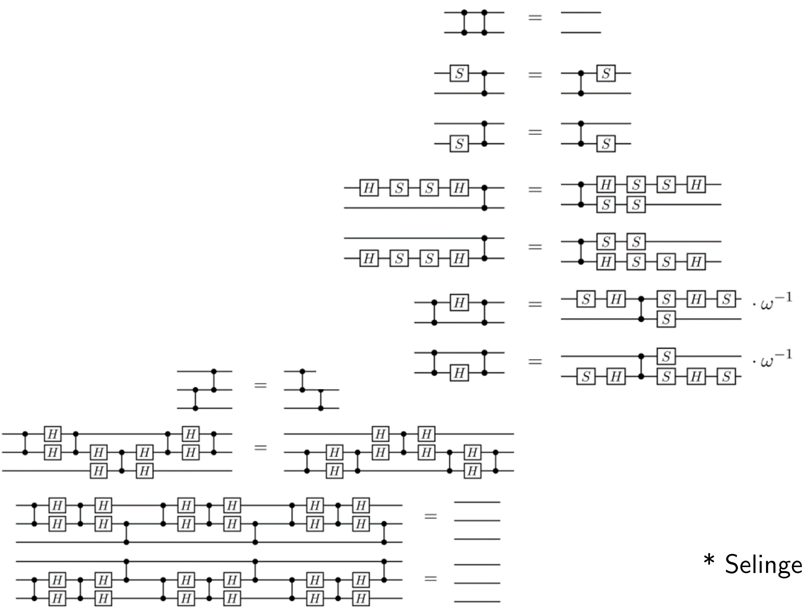

0
0
0
0
0
H
H
S
H
X
T
Z
Z


0
0
0
0
0
H
S
H
H
X
T
Z
Z


0
0
0
0
0
H
S
H
H
X
T
Z
Z

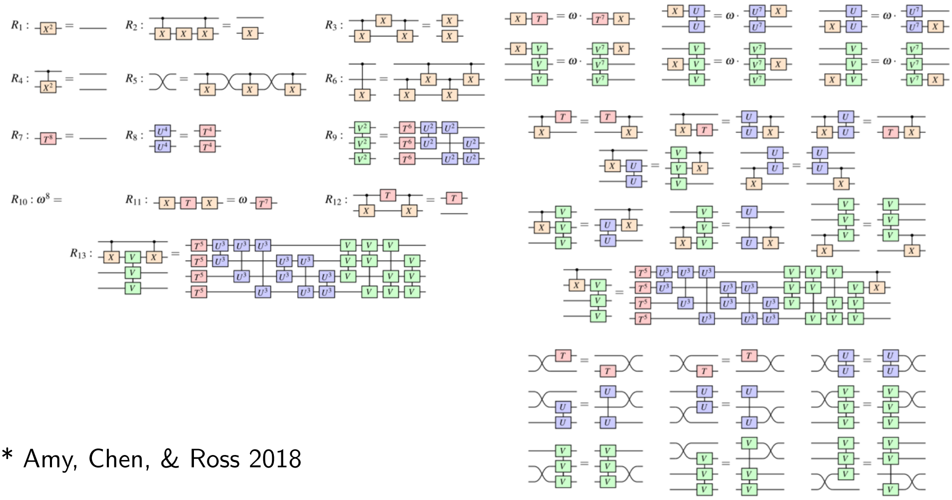

H
H
H
H
Z
Z
H
S
H
S
H
H

...

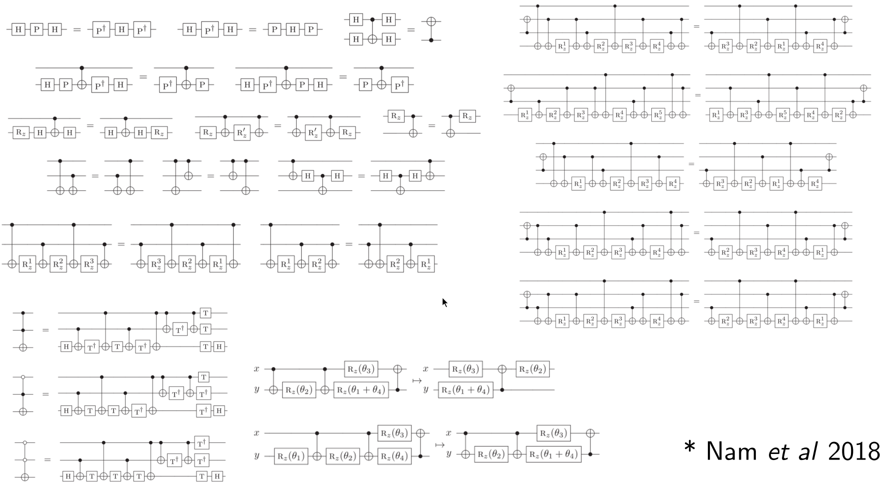



* Selinger 2015

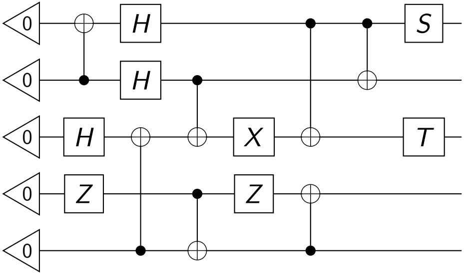
* Selinger 2015

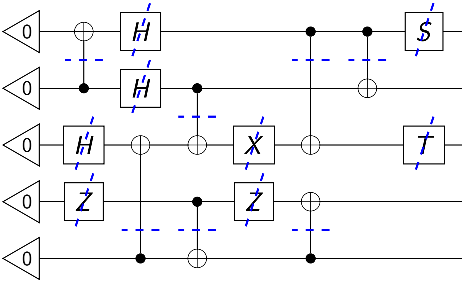


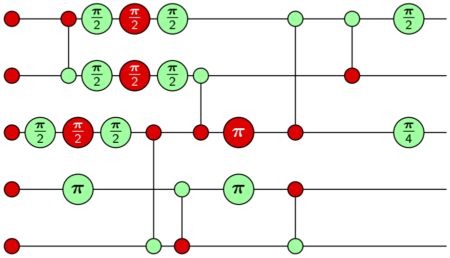
* Amy, Chen, & Ross 2018

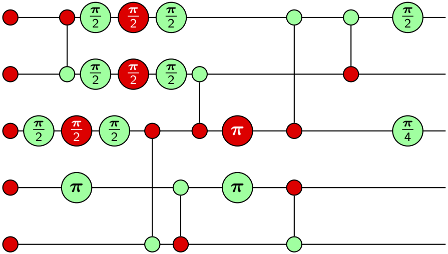


* Nam *et al* 2018


0
0
0
0
0
H
H
H
Z
X
Z
S
T


0
0
0
0
0
H
H
H
Z
X
Z
S
T

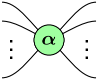

π/2
π/2
π/2
π/2
π/2
π/2
π/2
π/2
π/2
π/2
π
π
π
π/2
π/4



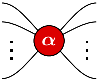

*ZX-diagram*

# ZX-diagrams: perspective 1

...are like circuits, but made from **spiders** instead of unitary gates:

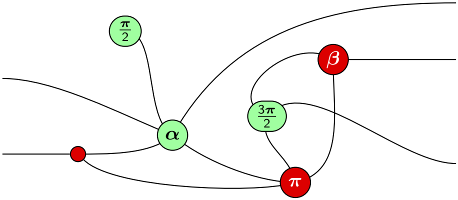
$\mathcal{Z}_\alpha =$ 

$= |0...0\rangle\langle 0...0| + e^{i\alpha}|1...1\rangle\langle 1...1|$

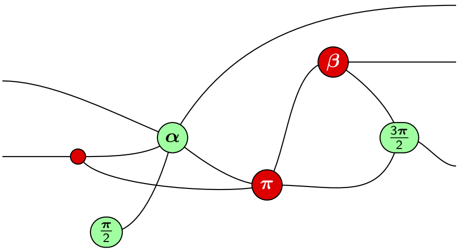
$\mathcal{X}_\alpha =$ 

$= |+...+\rangle\langle +...+| + e^{i\alpha}|-...-\rangle\langle -...-|$

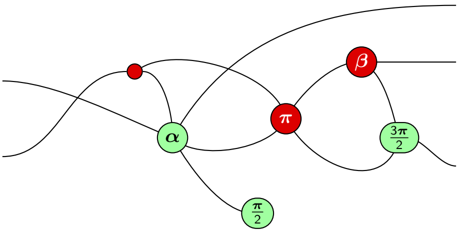
# ZX-diagrams are bendy



# ZX-diagrams are bendy

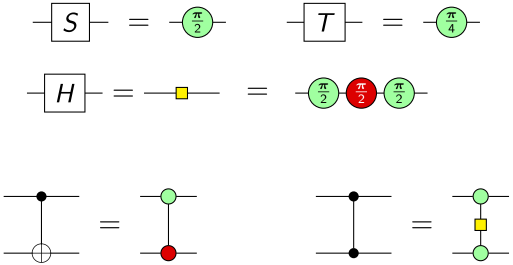


# ZX-diagrams are bendy

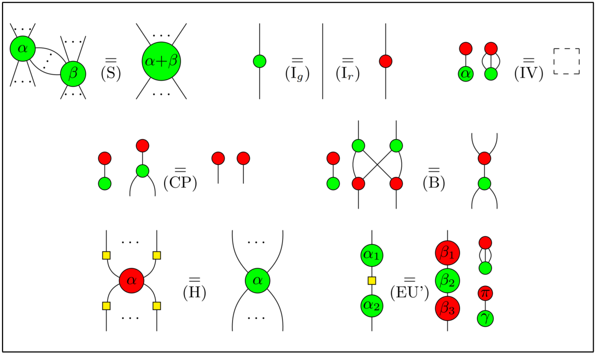


# Why spiders?

- They generate **all** linear maps $\mathbb{C}^{2^m} \to \mathbb{C}^{2^n}$.
- Handy for building most common gates, e.g.




- ....and....

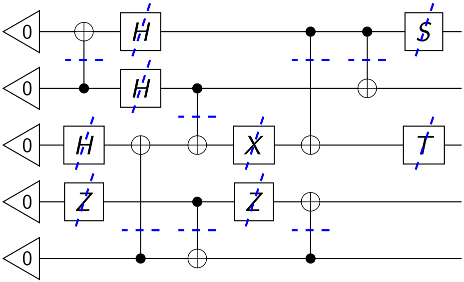


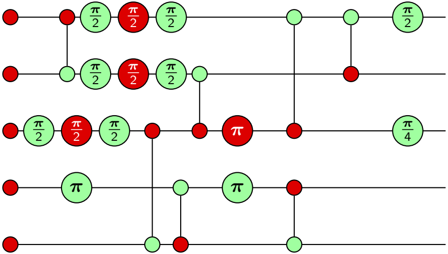
[1]

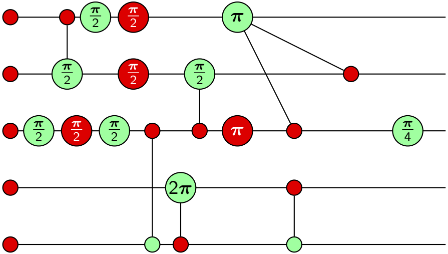
# *ZX calculus*

these 8 rules $\implies$ everything before

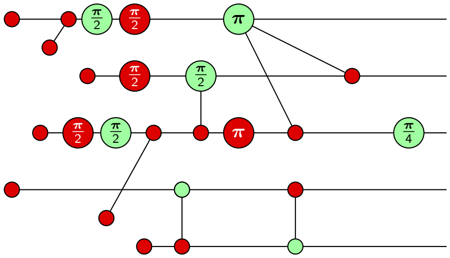
[1] Vilmart 2018. arXiv:1812.09114

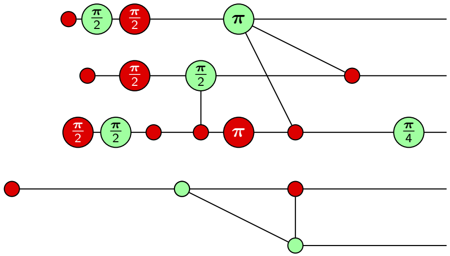

0
0
0
0
0
H
H
H
Z
X
Z
S
T


π/2
π/2
π/2
π/2
π/2
π/2
π/2
π/2
π/2
π/2
π/2
π/2
π
π
π
π/4

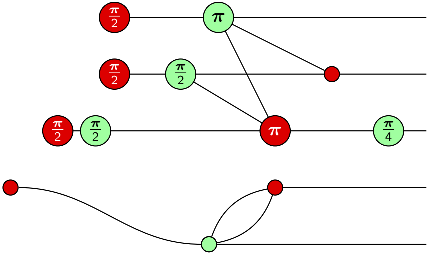


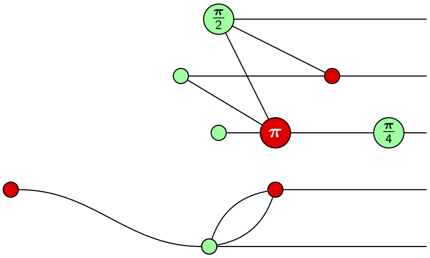
$\frac{\pi}{2}$
$\frac{\pi}{2}$
$\pi$
$\frac{\pi}{2}$
$\frac{\pi}{2}$
$\frac{\pi}{2}$
$\frac{\pi}{2}$
$\frac{\pi}{2}$
$\frac{\pi}{2}$
$\pi$
$\frac{\pi}{4}$
$2\pi$


π/2
π/2
π
π/2
π/2
π/2
π/2
π
π/4

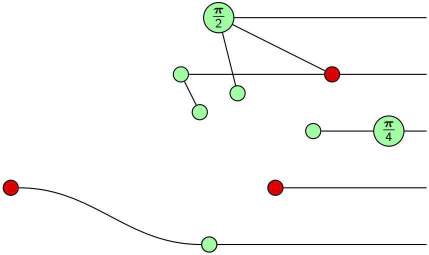


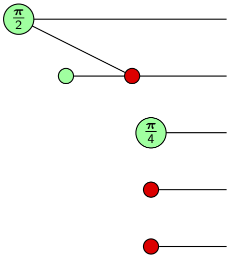
π/2
π/2
π
π/2
π/2
π/2
π/2
π
π/4


$\frac{\pi}{2}$
$\pi$
$\frac{\pi}{2}$
$\frac{\pi}{2}$
$\frac{\pi}{2}$
$\frac{\pi}{2}$
$\pi$
$\frac{\pi}{4}$

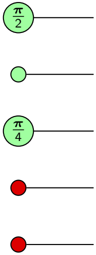


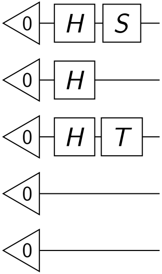
$\frac{\pi}{2}$
$\pi$
$\frac{\pi}{4}$


$\frac{\pi}{2}$
$\frac{\pi}{4}$

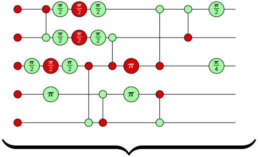


$\frac{\pi}{2}$
$\frac{\pi}{4}$


π
2
π
4

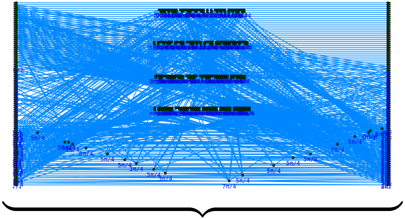

0
H
S
0
H
0
H
T
0
0

**Q:** How do we scale up?



$\sim$30 nodes

$\ll$

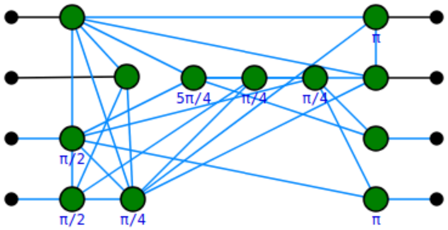


$10^2$-$10^5$ nodes

**A:** Automation.

# PyZX

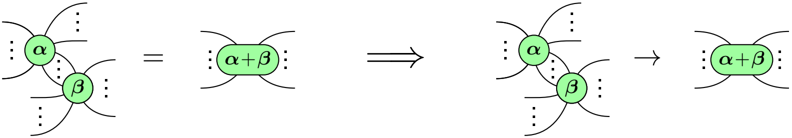
In [11]:
```
zx.simplify.clifford_simp(g)
g.normalise()
zx.d3.draw(g)
```




- Open-source tool for **quantum circuit optimisation, verification, and simulation using the ZX-calculus**

# The idea

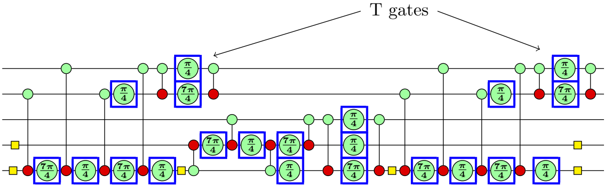
1. Turn *equations* into directed *rewrite rules*



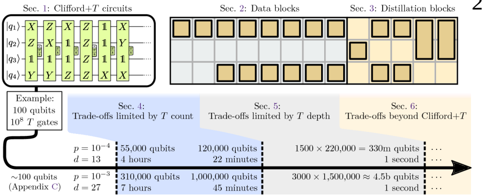

2. Use rewrite rules to *simplify* ZX-diagrams
3. *Extract* meaningful data from simplified diagram
   *(e.g. optimised circuits, amplitudes/probabilities, ...).*

# T-count reduction

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



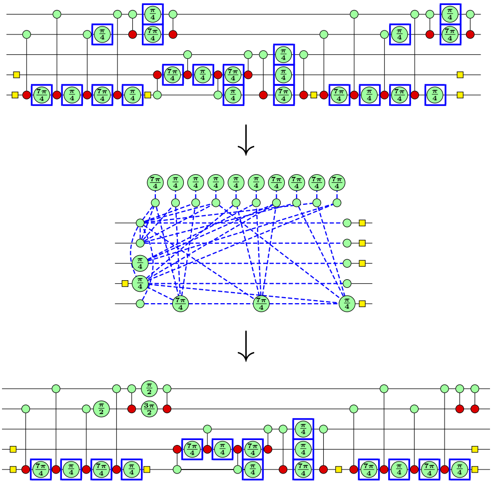

- **Q:** Why T gates?
- **A:** $\sim$100X more overhead in **fault-tolerant quantum computing**



[2]

[2] *A Game of Surface Codes.* Litinski 2019

# T-count reduction



# T-count reduction

| Circuit | n | T | Best prev. | Method | PyZX |
|---|---|---|---|---|---|
| **$\text{adder}_8$** | **24** | **399** | **213** | **$\text{RM}_m$** | **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 |
| $\text{tof}_3$ | 5 | 21 | 15 | Tpar | 15 |
| $\text{tof}_4$ | 7 | 35 | 23 | Tpar | 23 |
| $\text{tof}_5$ | 9 | 49 | 31 | Tpar | 31 |
| $\text{tof}_{10}$ | 19 | 119 | 71 | Tpar | 71 |
| *$\text{csla-mux}_3$* | *15* | *70* | *58* | *$\text{RM}_r$* | *62* |
| *$\text{csum-mux}_9$* | *30* | *196* | *76* | *$\text{RM}_r$* | *84* |
| **$\text{cycle17}_3$** | **35** | **4739** | **1944** | **$\text{RM}_m$** | **1797** |
| *$gf(2^4)$-mult* | *12* | *112* | *56* | *TODD* | *68* |
| *$gf(2^5)$-mult* | *15* | *175* | *90* | *TODD* | *115* |
| *$gf(2^6)$-mult* | *18* | *252* | *132* | *TODD* | *150* |
| *$gf(2^7)$-mult* | *21* | *343* | *185* | *TODD* | *217* |
| *$gf(2^8)$-mult* | *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 |
| $\text{hwb}_6$ | 7 | 105 | 75 | Tpar | 75 |
| **$\text{hwb}_8$** | **12** | **5887** | **3531** | **$\text{RM}_{m\&r}$** | **3517** |
| *mod-mult-55* | *9* | *49* | *28* | *TODD* | *35* |
| mod-red-21 | 11 | 119 | 73 | Tpar | 73 |
| **$\text{mod5}_4$** | **5** | **28** | **16** | **Tpar** | **8** |
| **$\text{nth-prime}_6$** | **9** | **567** | **400** | **$\text{RM}_{m\&r}$** | **279** |
| *$\text{nth-prime}_8$* | *12* | *6671* | *4045* | *$\text{RM}_{m\&r}$* | *4047* |
| $\text{qcla-adder}_{10}$ | 36 | 589 | 162 | Tpar | 162 |
| *$\text{qcla-com}_7$* | *24* | *203* | *94* | *$\text{RM}_m$* | *95* |
| $\text{qcla-mod}_7$ | 26 | 413 | 235 | NRSCM | 237 |
| $\text{rc-adder}_6$ | 14 | 77 | 47 | $\text{RM}_{m\&r}$ | 47 |
| $\text{vbe-adder}_3$ | 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 & Chen matched PyZX numbers with circuit method
- In Nov 2019, de Beaudrap, Bian, & Wang beat SotA again with new ZX method + TODD

# Correctness

- PyZX optimiser is **self-checking**:

$$\underbrace{C \rightsquigarrow D}_{\textit{optimising}} \qquad \Longrightarrow \qquad \underbrace{CD^{\dagger} \rightsquigarrow 1}_{\textit{checking}}$$

| Circuit | n | T | Best prev. | PyZX |
|---|---|---|---|---|
| adder$_8$ | 24 | 399 | 213 | 173 |
| Adder8 | 23 | 266 | 56 | 56 |
| Adder16 | 47 | 602 | 120 | 120 |
| Adder32 | 95 | 1274 | 248 | 248 |
| Adder64 | 191 | 2618 | 504 | 504 |
| barenco-tof3 | 5 | 28 | 16 | 16 |
| barenco-tof4 | 7 | 56 | 28 | 28 |
| barenco-tof5 | 9 | 84 | 40 | 40 |
| barenco-tof10 | 19 | 224 | 100 | 100 |
| tof$_3$ | 5 | 21 | 15 | 15 |
| tof$_4$ | 7 | 35 | 23 | 23 |
| tof$_5$ | 9 | 49 | 31 | 31 |
| tof$_{10}$ | 19 | 119 | 71 | 71 |
| csla-mux$_3$ | 15 | 70 | 58 | 62 |
| csum-mux$_9$ | 30 | 196 | 76 | 84 |
| cycle17$_3$ | 35 | 4739 | 1944 | 1797 |
| gf($2^4$)-mult | 12 | 112 | 56 | 68 |
| gf($2^5$)-mult | 15 | 175 | 90 | 115 |
| gf($2^6$)-mult | 18 | 252 | 132 | 150 |
| gf($2^7$)-mult | 21 | 343 | 185 | 217 |
| gf($2^8$)-mult | 24 | 448 | 216 | 264 |
| ham15-low | 17 | 161 | 97 | 97 |
| ham15-med | 17 | 574 | 230 | 212 |
| ham15-high | 20 | 2457 | 1019 | 1019 |
| hwb$_6$ | 7 | 105 | 75 | 75 |
| hwb$_8$ | 12 | 5887 | 3531 | 3517 |
| mod-mult-55 | 9 | 49 | 28 | 35 |
| mod-red-21 | 11 | 119 | 73 | 73 |
| mod5$_4$ | 5 | 28 | 16 | 8 |
| nth-prime$_6$ | 9 | 567 | 400 | 279 |
| nth-prime$_8$ | 12 | 6671 | 4045 | 4047 |
| qcla-adder$_{10}$ | 36 | 589 | 162 | 162 |
| qcla-com$_7$ | 24 | 203 | 94 | 95 |
| qcla-mod$_7$ | 26 | 413 | 235 | 237 |
| rc-adder$_6$ | 14 | 77 | 47 | 47 |
| vbe-adder$_3$ | 10 | 70 | 24 | 24 |

- This doesn't **prove** our code is correct, but:
  - it's lightweight
  - cost us nothing
  - **builds confidence** in correctness.
- ...and it all happens **before** we fire up the quantum computer

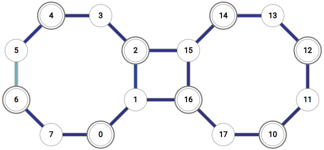
| Circuit | n | T | Best prev. | PyZX | |
|---|---|---|---|---|---|
| $\text{adder}_8$ | 24 | 399 | 213 | 173 | (✓) |
| Adder8 | 23 | 266 | 56 | 56 | (✓) |
| Adder16 | 47 | 602 | 120 | 120 | (✓) |
| Adder32 | 95 | 1274 | 248 | 248 | (✓) |
| Adder64 | 191 | 2618 | 504 | 504 | (✓) |
| barenco-tof3 | 5 | 28 | 16 | 16 | (✓) |
| barenco-tof4 | 7 | 56 | 28 | 28 | (✓) |
| barenco-tof5 | 9 | 84 | 40 | 40 | (✓) |
| barenco-tof10 | 19 | 224 | 100 | 100 | (✓) |
| $\text{tof}_3$ | 5 | 21 | 15 | 15 | (✓) |
| $\text{tof}_4$ | 7 | 35 | 23 | 23 | (✓) |
| $\text{tof}_5$ | 9 | 49 | 31 | 31 | (✓) |
| $\text{tof}_{10}$ | 19 | 119 | 71 | 71 | (✓) |
| $\text{csla-mux}_3$ | 15 | 70 | 58 | 62 | (✓) |
| $\text{csum-mux}_9$ | 30 | 196 | 76 | 84 | (✓) |
| $\text{cycle17}_3$ | 35 | 4739 | 1944 | 1797 | (✓) |
| $\text{gf}(2^4)\text{-mult}$ | 12 | 112 | 56 | 68 | (✓) |
| $\text{gf}(2^5)\text{-mult}$ | 15 | 175 | 90 | 115 | (✓) |
| $\text{gf}(2^6)\text{-mult}$ | 18 | 252 | 132 | 150 | (✓) |
| $\text{gf}(2^7)\text{-mult}$ | 21 | 343 | 185 | 217 | (✓) |
| $\text{gf}(2^8)\text{-mult}$ | 24 | 448 | 216 | 264 | (✓) |
| ham15-low | 17 | 161 | 97 | 97 | (✓) |
| ham15-med | 17 | 574 | 230 | 212 | (✓) |
| ham15-high | 20 | 2457 | 1019 | 1019 | (✓) |
| $\text{hwb}_6$ | 7 | 105 | 75 | 75 | (✓) |
| $\text{hwb}_8$ | 12 | 5887 | 3531 | 3517 | (✓) |
| mod-mult-55 | 9 | 49 | 28 | 35 | (✓) |
| mod-red-21 | 11 | 119 | 73 | 73 | (✓) |
| $\text{mod5}_4$ | 5 | 28 | 16 | 8 | (✓) |
| $\text{nth-prime}_6$ | 9 | 567 | 400 | 279 | (✓) |
| $\text{nth-prime}_8$ | 12 | 6671 | 4045 | 4047 | (✓) |
| $\text{qcla-adder}_{10}$ | 36 | 589 | 162 | 162 | (✓) |
| $\text{qcla-com}_7$ | 24 | 203 | 94 | 95 | (✓) |
| $\text{qcla-mod}_7$ | 26 | 413 | 235 | 237 | (✓) |
| $\text{rc-adder}_6$ | 14 | 77 | 47 | 47 | (✓) |
| $\text{vbe-adder}_3$ | 10 | 70 | 24 | 24 | (✓) |

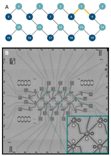
- This doesn't **prove** our code is correct, but:
  - it's lightweight
  - cost us nothing
  - **builds confidence** in correctness.
- ...and it all happens **before** we fire up the quantum computer

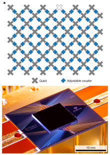
| Circuit | n | T | Best prev. | PyZX | |
|---|---|---|---|---|---|
| $\text{adder}_8$ | 24 | 399 | 213 | 173 | (✓) |
| Adder8 | 23 | 266 | 56 | 56 | (✓) |
| Adder16 | 47 | 602 | 120 | 120 | (✓) |
| Adder32 | 95 | 1274 | 248 | 248 | (✓) |
| Adder64 | 191 | 2618 | 504 | 504 | (✓) |
| barenco-tof3 | 5 | 28 | 16 | 16 | (✓) |
| barenco-tof4 | 7 | 56 | 28 | 28 | (✓) |
| barenco-tof5 | 9 | 84 | 40 | 40 | (✓) |
| barenco-tof10 | 19 | 224 | 100 | 100 | (✓) |
| $\text{tof}_3$ | 5 | 21 | 15 | 15 | (✓) |
| $\text{tof}_4$ | 7 | 35 | 23 | 23 | (✓) |
| $\text{tof}_5$ | 9 | 49 | 31 | 31 | (✓) |
| $\text{tof}_{10}$ | 19 | 119 | 71 | 71 | (✓) |
| $\text{csla-mux}_3$ | 15 | 70 | 58 | 62 | (✓) |
| $\text{csum-mux}_9$ | 30 | 196 | 76 | 84 | (✓) |
| $\text{cycle17}_3$ | 35 | 4739 | 1944 | 1797 | (✓) |
| $\text{gf}(2^4)\text{-mult}$ | 12 | 112 | 56 | 68 | (✓) |
| $\text{gf}(2^5)\text{-mult}$ | 15 | 175 | 90 | 115 | (✓) |
| $\text{gf}(2^6)\text{-mult}$ | 18 | 252 | 132 | 150 | (✓) |
| $\text{gf}(2^7)\text{-mult}$ | 21 | 343 | 185 | 217 | (✓) |
| $\text{gf}(2^8)\text{-mult}$ | 24 | 448 | 216 | 264 | (✓) |
| ham15-low | 17 | 161 | 97 | 97 | (✓) |
| ham15-med | 17 | 574 | 230 | 212 | (✓) |
| ham15-high | 20 | 2457 | 1019 | 1019 | (✓) |
| $\text{hwb}_6$ | 7 | 105 | 75 | 75 | (✓) |
| $\text{hwb}_8$ | 12 | 5887 | 3531 | 3517 | (✓) |
| mod-mult-55 | 9 | 49 | 28 | 35 | (✓) |
| mod-red-21 | 11 | 119 | 73 | 73 | (✓) |
| $\text{mod5}_4$ | 5 | 28 | 16 | 8 | (✓) |
| $\text{nth-prime}_6$ | 9 | 567 | 400 | 279 | (✓) |
| $\text{nth-prime}_8$ | 12 | 6671 | 4045 | 4047 | (✓) |
| $\text{qcla-adder}_{10}$ | 36 | 589 | 162 | 162 | (✓) |
| $\text{qcla-com}_7$ | 24 | 203 | 94 | 95 | (✓) |
| $\text{qcla-mod}_7$ | 26 | 413 | **(X)** 235 | 237 | (✓) |
| $\text{rc-adder}_6$ | 14 | 77 | 47 | 47 | (✓) |
| $\text{vbe-adder}_3$ | 10 | 70 | 24 | 24 | (✓) |

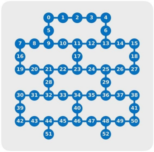
- This doesn't **prove** our code is correct, but:
  - it's lightweight
  - cost us nothing
  - **builds confidence** in correctness.
- ...and it all happens **before** we fire up the quantum computer

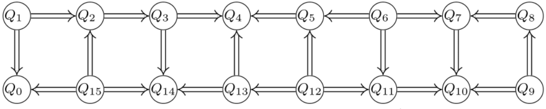
# Circuit routing



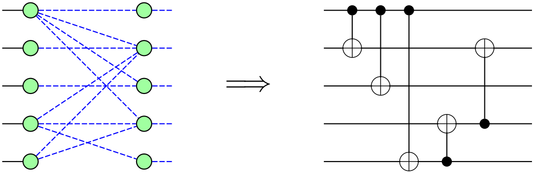




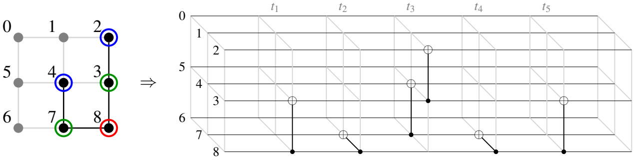



# Circuit routing

- We produce circuits from ZX-diagrams by **circuit extraction**:



- This gives us some freedom…which enables us to do **circuit routing**:



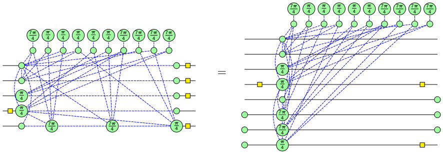
# CNOT Circuit routing

| 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% |

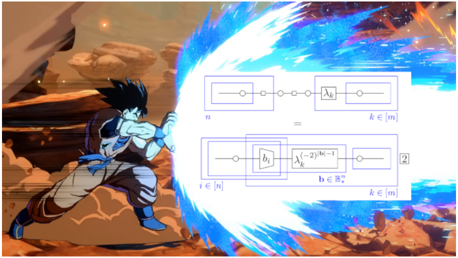
- up to 5X more efficient vs. Rigetti, CQC, IBM (as of April 2019)

# Where to?

- More aggressive optimisation $\rightsquigarrow$ more difficult circuit extraction



- Circuit extraction methods can blow up CNOT count/depth (FIXME)
- More powerful big-step reductions using **ZH-calculus**



# Thanks!

PyZX: github.com/Quantomatic/pyzx


- *PyZX: Large-scale automated diagrammatic reasoning*. AK, John van de Wetering. arXiv:1904.04735 [quant-ph]
- **Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning**. Bob Coecke, AK. Cambridge University Press 2017. cambridge.org/pqp