

# Noise Aware Compilation of AWS Braket Circuits

---

Team iCuHack



# Inspirations

How do we apply ideas from:

- Compiler infrastructure for classical programs
- Programming Language Theory

to compiling quantum circuits?

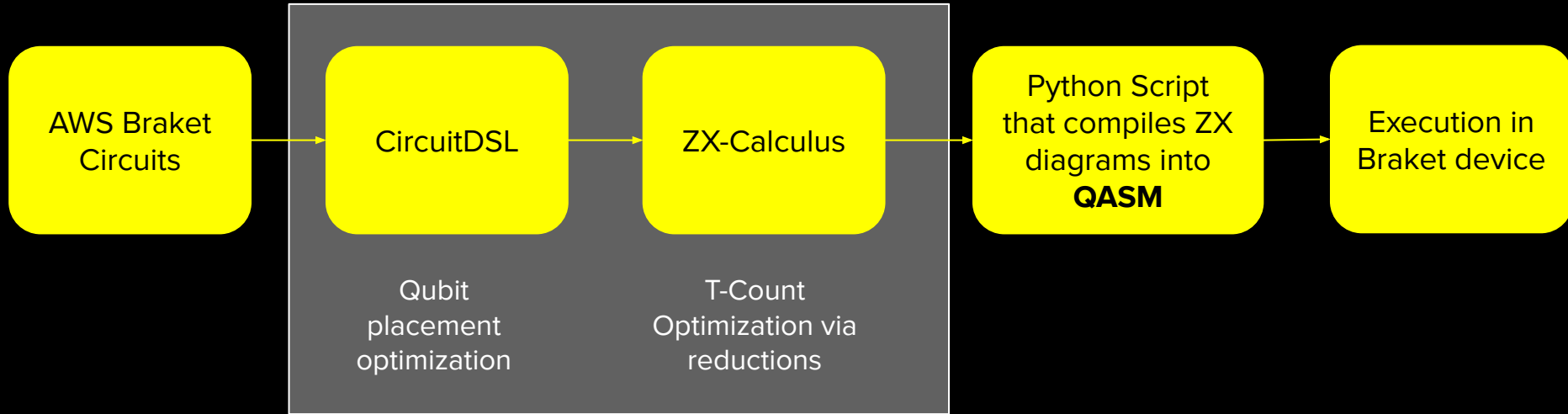
What optimizations can we implement on that for noise-awareness?

Sneak peak:

Greedy Qubit placement optimization on CircuitDSL

Compilation to ZX Calculus for T-Count Reduction

# What We Built



# What we built

```
26
27 def compiler_pipeline(source: str, outfile: str, candidate_pairs):
28     # Parse source into circuitdsl
29     circuit = parse(source)
30
31     # Optimize qubit placement on circuitdsl
32     if candidate_pairs:
33         circuit = qubit_placement_optimization(circuit, candidate_pairs)
34
35     # Compile to ZX calc program that runs circuit qasm with Bracket
36     zxprogram = zxcalc_program(circuit)
37
38     # Emit final compiled program
39     print(f"Compiled to {outfile}...")
40     with open(outfile, 'w') as f:
41         f.write(zxprogram)
42
```

# Source Program to be Compiled:

```
~/Workspace/2024_AWS chris !1 ?12 > cat demo1.py
from braket.circuits import Circuit
from braket.devices import LocalSimulator
from braket.circuits.serialization import IRType

# circuitdsl start
def error_correction_circuit(circuit):
    circuit = circuit.h(0)
    circuit = circuit.h(1)
    circuit = circuit.h(2)
    circuit = circuit.h(3)
    circuit = circuit.cnot(0, 4)
    circuit = circuit.cnot(2, 5)
    circuit = circuit.cnot(1, 4)
    circuit = circuit.cnot(3, 5)
    circuit = circuit.cnot(4, 6)
    circuit = circuit.cnot(5, 6)

    return circuit
# circuitdsl end

def main():
    device = LocalSimulator()
    circuit = Circuit()
    error_correction = error_correction_circuit(circuit)
    qasm_ir = error_correction.to_ir(IRType.OPENQASM)
    print(device.run(error_correction, shots=100).result().measurement_counts)

if __name__ == "__main__":
    main()
```

# Running the compiler

```
~/Workspace/2024_AWS chris !1 ?12 > py311 compiler.py --help
```

```
Usage: compiler.py [OPTIONS] FILENAME [CANDIDATE_PAIRS_JSON]
```

## Arguments:

```
  FILENAME                [required]  
  [CANDIDATE_PAIRS_JSON]
```

## Options:

```
--install-completion  Install completion for the current shell.  
--show-completion      Show completion for the current shell, to copy it or  
                       customize the installation.  
--help                Show this message and exit.
```

```
~/Workspace/2024_AWS chris !1 ?12 > py311 compiler.py demo1.py candidate_pair.json  
Compiled to demo1_compiled.py...
```

# Compiled Code

```
~/Workspace/2024_AWS chris !1 ?12 > cat demo1_compiled.py
import pyzx as zx
from icuhack.qasm_rewrites import qasm_rewrites
from bracket.ir.openqasm import Program
from bracket.devices import LocalSimulator

def error_correction_circuit():
    circuit = zx.Circuit(7)
    circuit.add_gate("H", 0)
    circuit.add_gate("H", 1)
    circuit.add_gate("H", 2)
    circuit.add_gate("H", 3)
    circuit.add_gate("CNOT", 0, 3)
    circuit.add_gate("CNOT", 2, 6)
    circuit.add_gate("CNOT", 1, 3)
    circuit.add_gate("CNOT", 4, 6)
    circuit.add_gate("CNOT", 3, 5)
    circuit.add_gate("CNOT", 6, 5)
    return circuit

def reduce_zx(circuit):
    graph = circuit.to_graph()
    test_graph = graph.copy()
    test_graph = zx.teleport_reduce(test_graph, quiet=False)
    if circuit.verify_equality(zx.Circuit.from_graph(graph)) == True:
        print('verified!')
    c1 = zx.extract_circuit(graph).to_basic_gates()
    c1 = c1.stats()
    c1_parsed = c1.split("\n")
    print('T-count BEFORE reduction: ' + c1_parsed[1][8])
    graph = zx.teleport_reduce(graph, quiet=False)
    c2 = zx.extract_circuit(graph).to_basic_gates()
    c2 = c2.stats()
    c2_parsed = c2.split("\n")
    print('T-count AFTER reduction: ' + c2_parsed[1][8])
    c_opt = zx.extract_circuit(graph.copy())
    return c_opt

def to_qasm(circuit):
    return circuit.to_basic_gates().to_qasm()
```

```
def zxcalc_gen_qasm_postprocess(qasm, num_qubits):
    qasm_lines = qasm.split("\n")
    for i in range(len(qasm_lines)):
        if "qelib" in qasm_lines[i]:
            qasm_lines[i] = ""
    qasm_lines = qasm_rewrites(qasm_lines, num_qubits)
    return "\n".join(qasm_lines)

def main():
    device = LocalSimulator()
    circuit = error_correction_circuit()
    num_qubits = 7
    reduced = reduce_zx(circuit)
    program_qasm = to_qasm(circuit)
    qasm = zxcalc_gen_qasm_postprocess(program_qasm, num_qubits)
    program = Program(source=qasm)
    result = device.run(program, shots=100).result()
    print(result.measurement_counts)

if __name__ == "__main__":
    main()
```

# Running the Compiled Code

```
~/Workspace/2024_AWS chris !1 ?12 > py311 demo1_compiled.py
spider_simp: 4. 2. 2 iterations
id_simp: 1. 1 iterations
verified!
T-count BEFORE reduction: 0
T-count AFTER reduction: 0
Counter({'1111001': 9, '1100000': 8, '0011001': 8, '0110011': 8, '0101010': 7, '1001010': 7, '1011001': 7, '1010011': 7, '1000000': 7, '0000000': 6, '1110011': 6, '1101010': 6, '0001010': 4, '0010011': 4, '0111001': 3, '0100000': 3})
```



# Highlights & Takeaways

## Highlights:

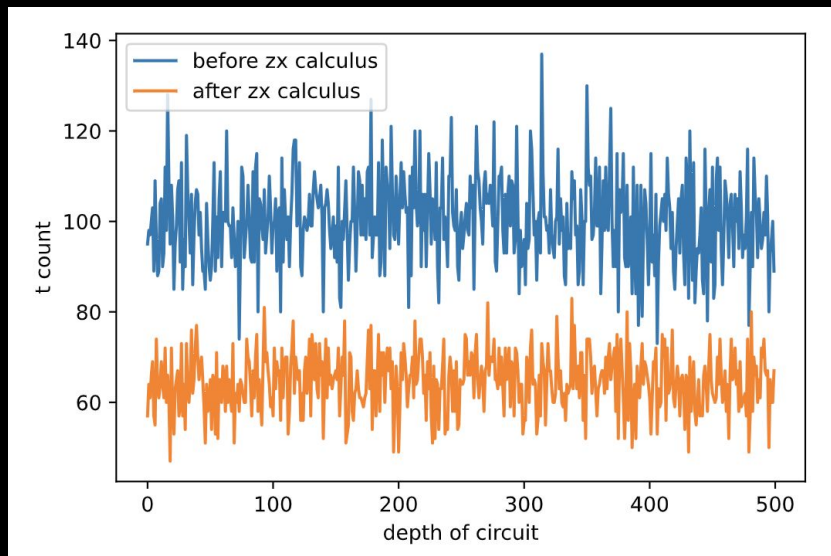
- CircuitDSL as an IR between Braket circuits and ZX calculus
- Greedy Iterative Qubit Placement Optimization on the CircuitDSL
- ZX Calculus as an IR / CircuitDSL-to-ZX compiler
- ZX Calculus Reductions as program optimization and verification

## Takeaways:

- QC needs more mature, standardized IRs (like LLVM, MLIR for classical programs)
- Exciting field for more engineering & development

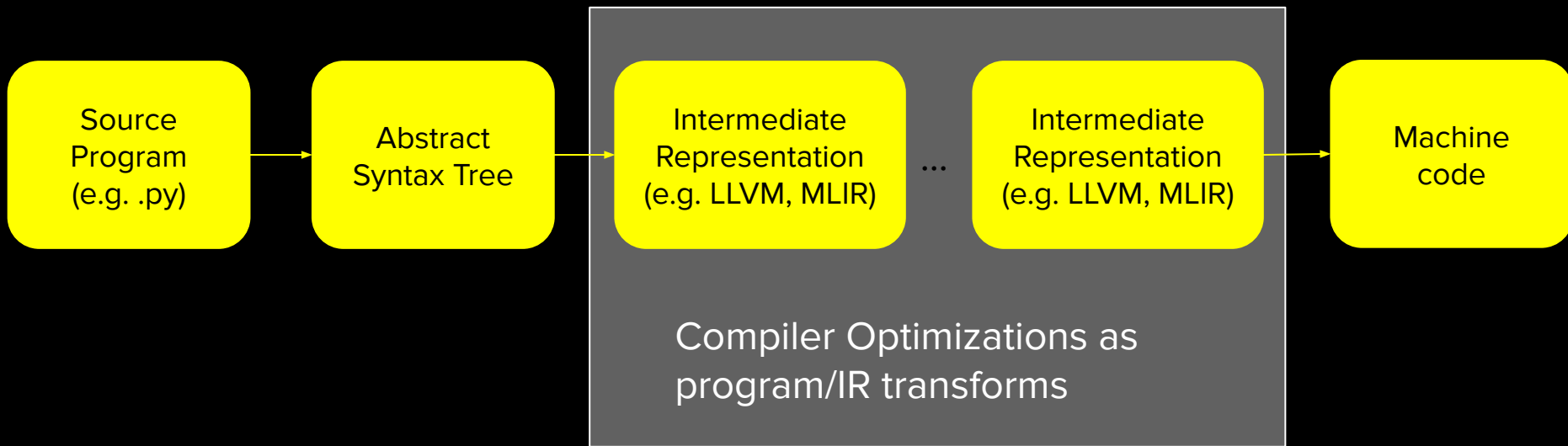
# ZX Calculus Reductions Reduces T-count

ZX Calculus Reductions reduces complexity for randomly generated circuits of any depth

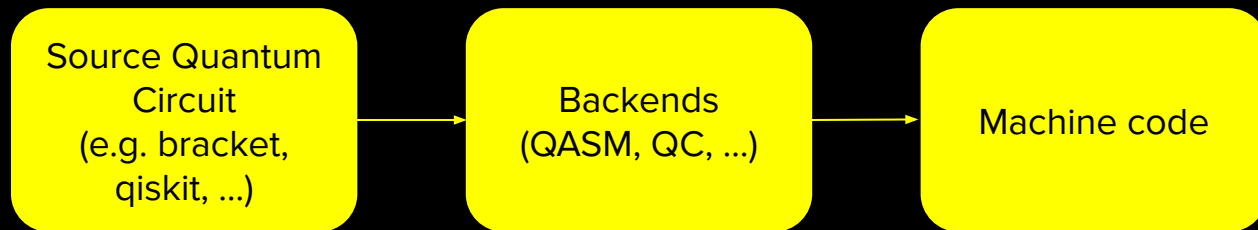


# The Details

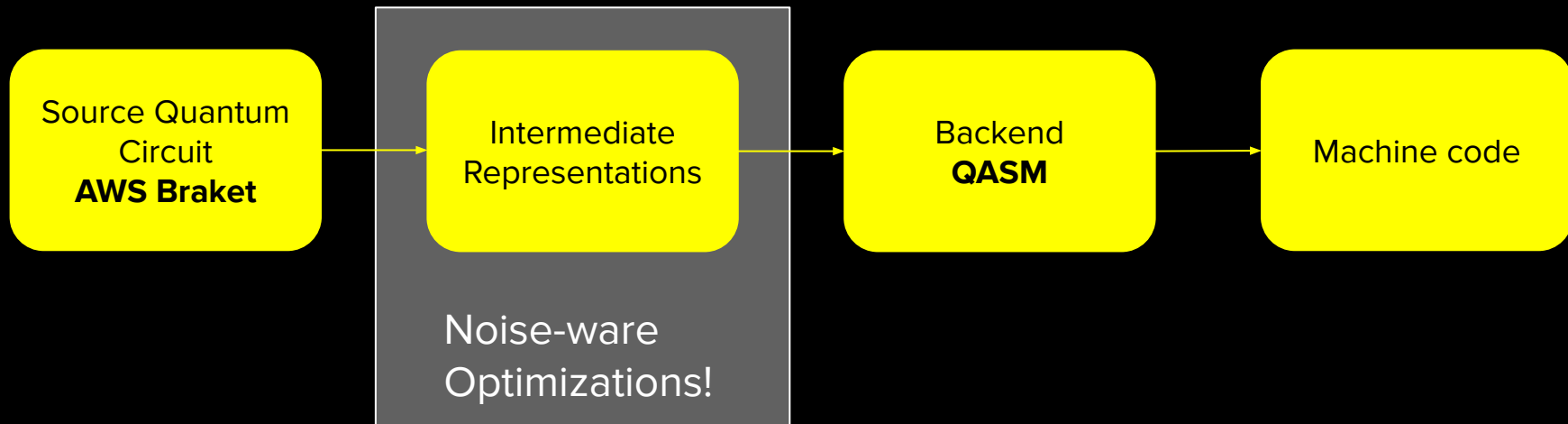
# Classical Compiler Infrastructure



# Quantum Compiler Infrastructure



# Our Approach

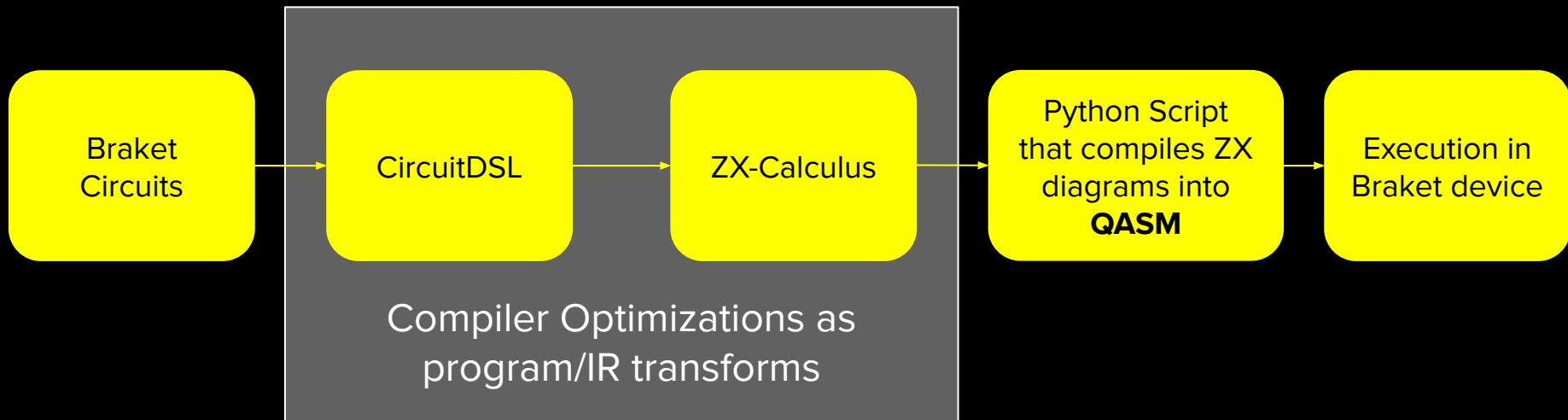


# What Intermediate Representations?

## What optimizations?

- CircuitDSL: A simple grammar to parse Braket circuits into a stack of gates
  - Quantum circuits are mostly defined very sequentially:
    - e.g. `circuit = Circuit.h(0).cnot(0, 1), ...`
    - Easy to lower this to a stack based program representation
    - Easy to then swap/reorder gates and their input qubit arguments for optimization
- ZX-Calculus: A formal language and reduction rule for Quantum Circuits
  - Compile Braket circuits into the ZX-Calculus
  - Perform reductions to find equivalent circuits with lower T-counts
  - Inspired by lambda calculus, combinator calculi for classical programs

# Our Approach (More specified)





# CircuitDSL

```
~/Workspace/2024_AWS chris !3 ?12 > py311 compiler.py demo1.py candidate_pair.json
```

```
CircuitDSL:
```

```
circuit: error_correction_circuit
```

```
program:
```

```
  HADAMARD input: 0
```

```
  HADAMARD input: 1
```

```
  HADAMARD input: 2
```

```
  HADAMARD input: 3
```

```
  CNOT control: 0 target: 4
```

```
  CNOT control: 2 target: 5
```

```
  CNOT control: 1 target: 4
```

```
  CNOT control: 3 target: 5
```

```
  CNOT control: 4 target: 6
```

```
  CNOT control: 5 target: 6
```

# Noise-Aware Optimization on CircuitDSL

- Prioritized correctness of transformation over superoptimization
- Greedy iterative algorithm
  - Given a list of qubit pairs and their fidelities (or noises), we iterate through the 2-qubit gates and reassign qubits, keeping track of the assignments to avoid overlap

# ZX Calculus

```
~/Workspace/2024_AWS chris !3 ?12 > cat demo1_compiled.py
```

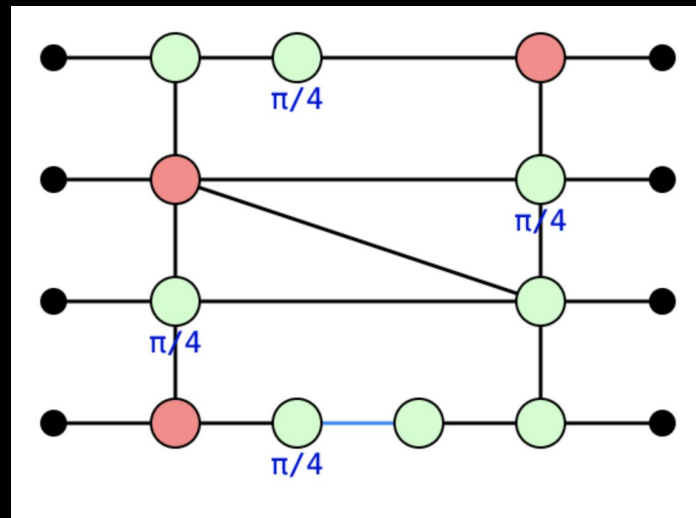
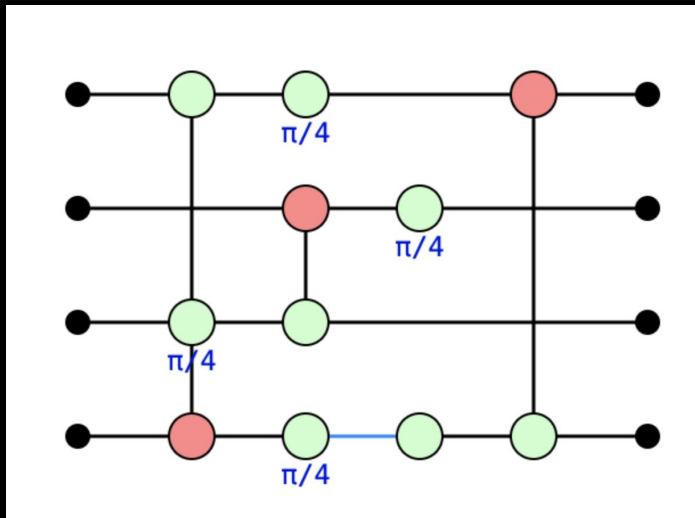
```
import pyzx as zx
from icuhack.qasm_rewrites import qasm_rewrites
from braket.ir.openqasm import Program
from braket.devices import LocalSimulator
```

```
def error_correction_circuit():
    circuit = zx.Circuit(7)
    circuit.add_gate("H", 0)
    circuit.add_gate("H", 1)
    circuit.add_gate("H", 2)
    circuit.add_gate("H", 3)
    circuit.add_gate("CNOT", 0, 3)
    circuit.add_gate("CNOT", 2, 6)
    circuit.add_gate("CNOT", 1, 3)
    circuit.add_gate("CNOT", 4, 6)
    circuit.add_gate("CNOT", 3, 5)
    circuit.add_gate("CNOT", 6, 5)
    return circuit
```

```
def reduce_zx(circuit):
    graph = circuit.to_graph()
    test_graph = graph.copy()
    test_graph = zx.teleport_reduce(test_graph, quiet=False)
    if circuit.verify_equality(zx.Circuit.from_graph(graph))==True:
        print('verified!')
    c1 = zx.extract_circuit(graph).to_basic_gates()
    c1 = c1.stats()
    c1_parsed = c1.split("\n")
    print('T-count BEFORE reduction: ' + c1_parsed[1][8])
    graph = zx.teleport_reduce(graph, quiet=False)
    c2 = zx.extract_circuit(graph).to_basic_gates()
    c2 = c2.stats()
    c2_parsed = c2.split("\n")
    print('T-count AFTER reduction: ' + c2_parsed[1][8])
    c_opt = zx.extract_circuit(graph.copy())
    return c_opt
```

```
def to_qasm(circuit):
    return circuit.to_basic_gates().to_qasm()
```

# ZX Calculus

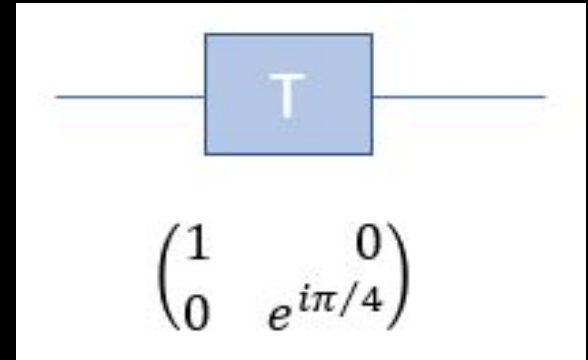


ZX-diagrams before and after applying rewrite rules

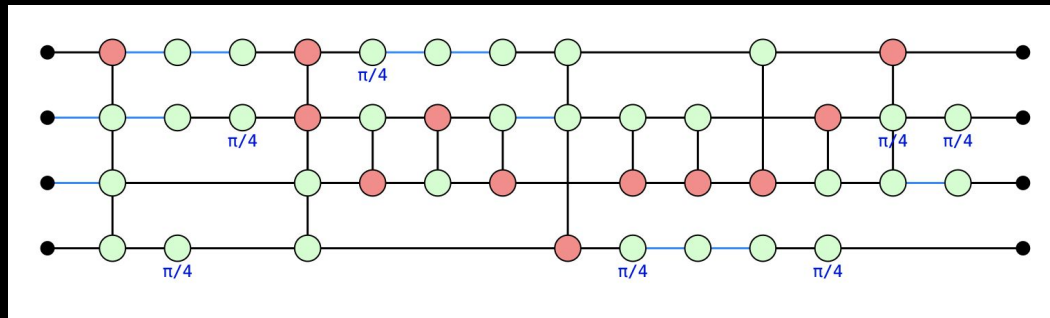
# ZX Calculus for Noise Robustness

1. ZX Calculus **simplifies** quantum circuits to reduce complexity. One measure of circuit complexity is the T-Count.
2. **T-Count**: # of T-gates in a quantum circuit. **Reducing T-gates** reduces **noise** and **complexity** of the circuit.

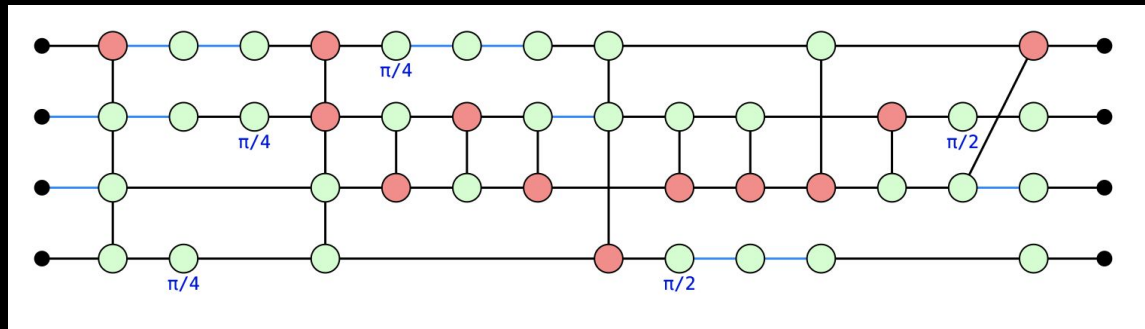
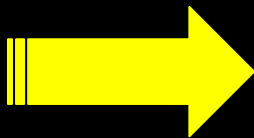
T-Gate:



# ZX Calculus at Work

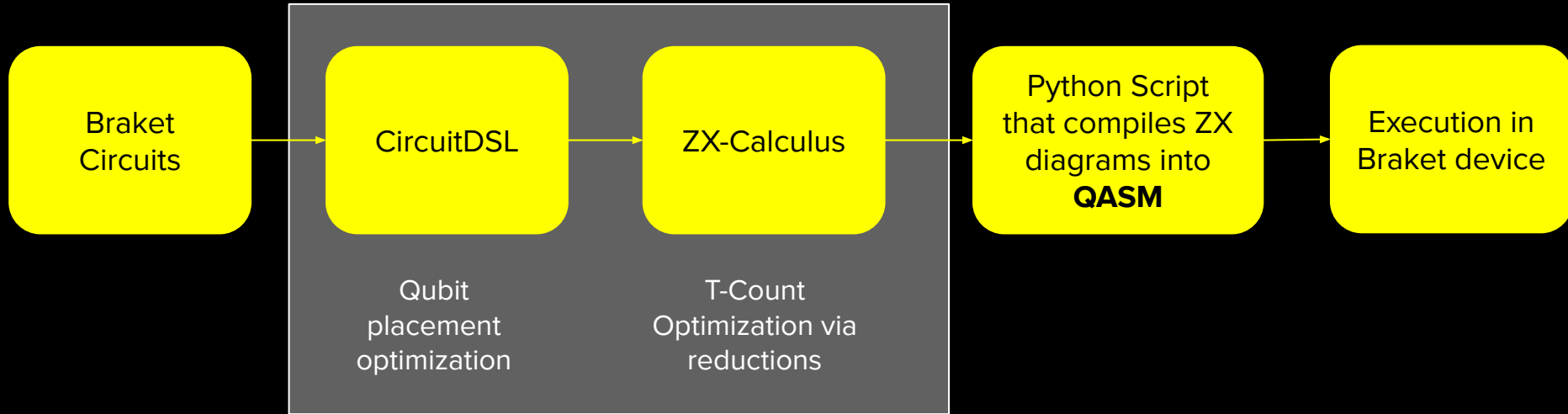


Note the # of T-Gates!



ZX-diagrams before and after applying rewrite rules

# In Summary:



# iCuHack



**Chris Yoon**

Senior @  
Columbia University  
*Computer Science*

Interested in all things  
compilers & programming  
language theory



**Ha Yeon Kim**

Senior @  
Columbia University  
*Computer Science*

Interested in data visualization  
and quantum computing



**Kevin Park**

Senior @  
Columbia University  
*Physics, Math*

Interested in blackholes and  
soccer



**Erica Choi**

Senior @  
Columbia University  
*Math, Computer Science*

Interested in algebraic  
topology, functional analysis,  
and quantum computing



**Joey Koh**

Data Eng./Science  
Product Engineer  
Uni. of Western Australia  
*Electrical Engineering & Finance*

**34 Hackathons.**  
Synthetic Data, Data Pipelines,  
Gen AI Builder.  
Army Lieutenant.

Looking to move to work in US





# Thank you

---

[Link to our repo](#)