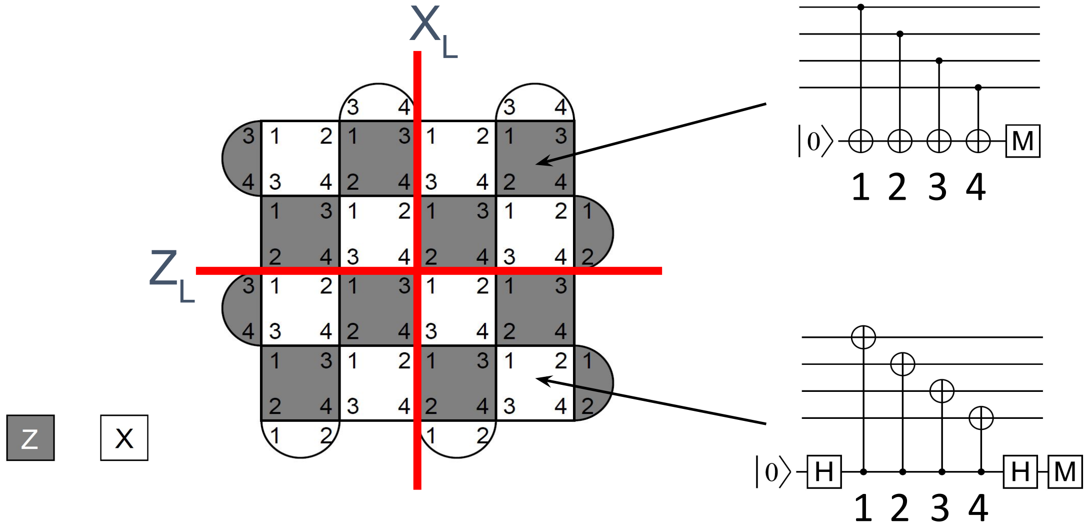
Introduction to Stim

Austin Fowler

Recall: the surface code

Even for distance 5, the surface code is a complex 49 qubit circuit requiring complex error analysis and decoding.



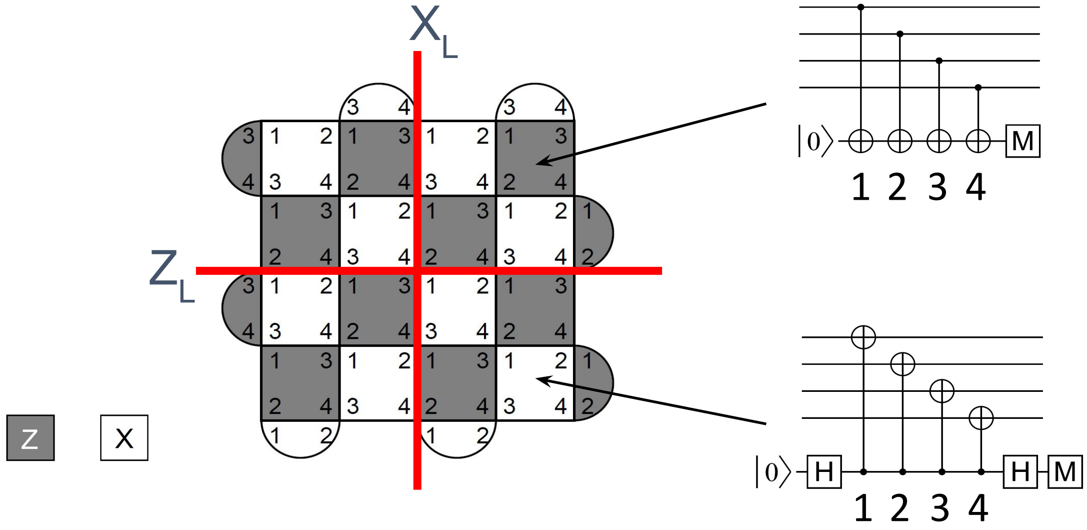
d=5

How are we going to simulate this?

Stim: a fast stabilizer circuit simulator

Written by Craig Gidney

Described in arXiv:2103.02202



d=5

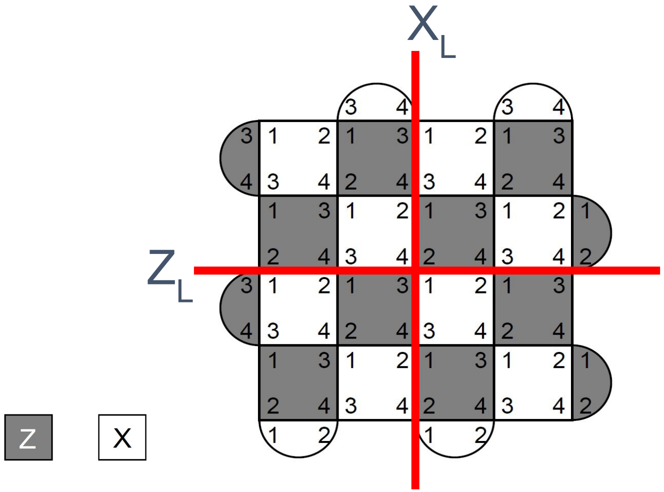
Stim can analyze 100 rounds of a distance 100 surface code in 15 seconds and then begin sampling shots at a rate of 1 kHz.

1000x+ faster than any other simulator.

Stim: overview

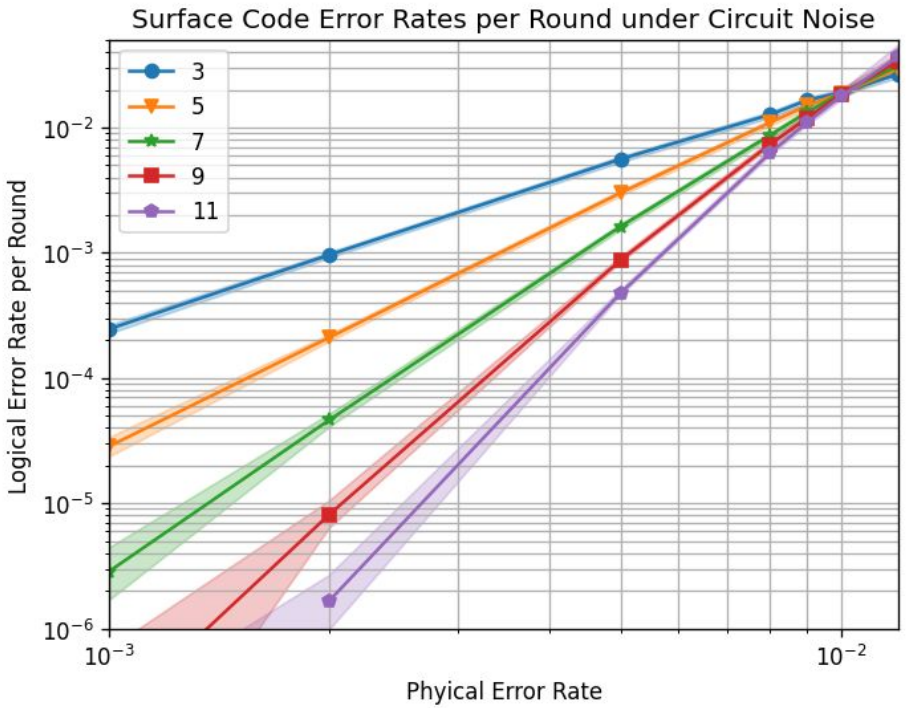
Open source: https://github.com/quantumlib/Stim
Start with: doc/getting_started.ipynb

Download and open in google.colab



d=5

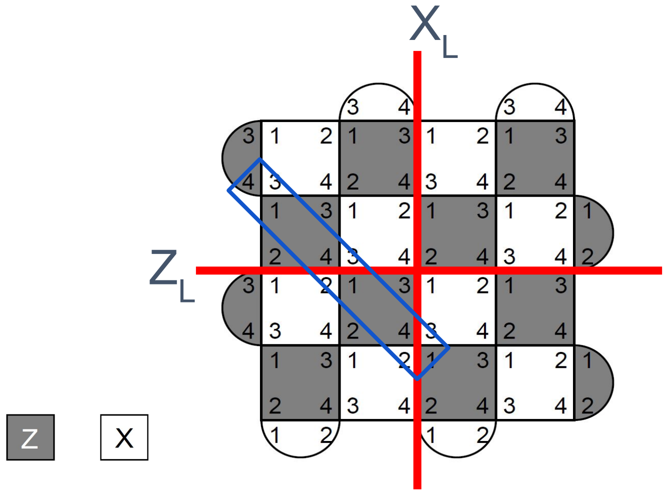
After a few minutes of simulation:



Learn how to generate the above using Stim and what it means.

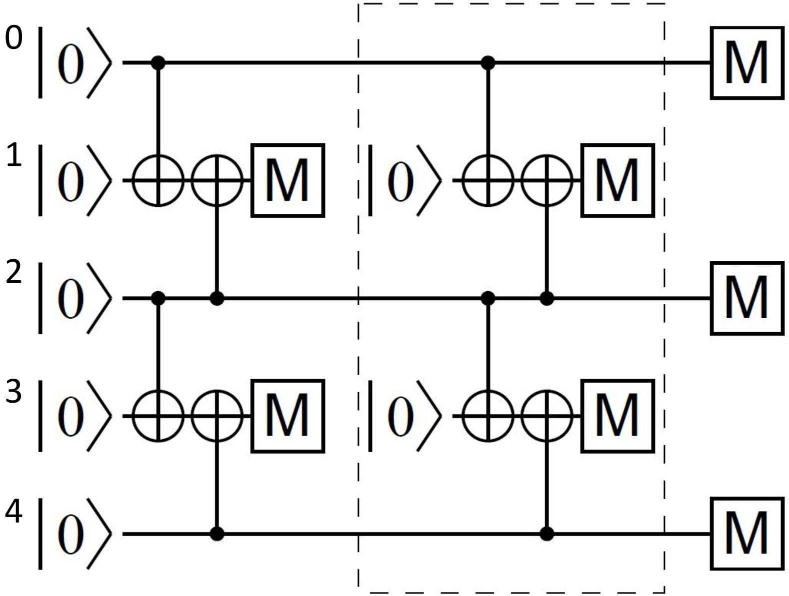
Stim: choose a circuit

Surface code circuits are rather large, so let's focus on just a piece, the repetition code.



d=5

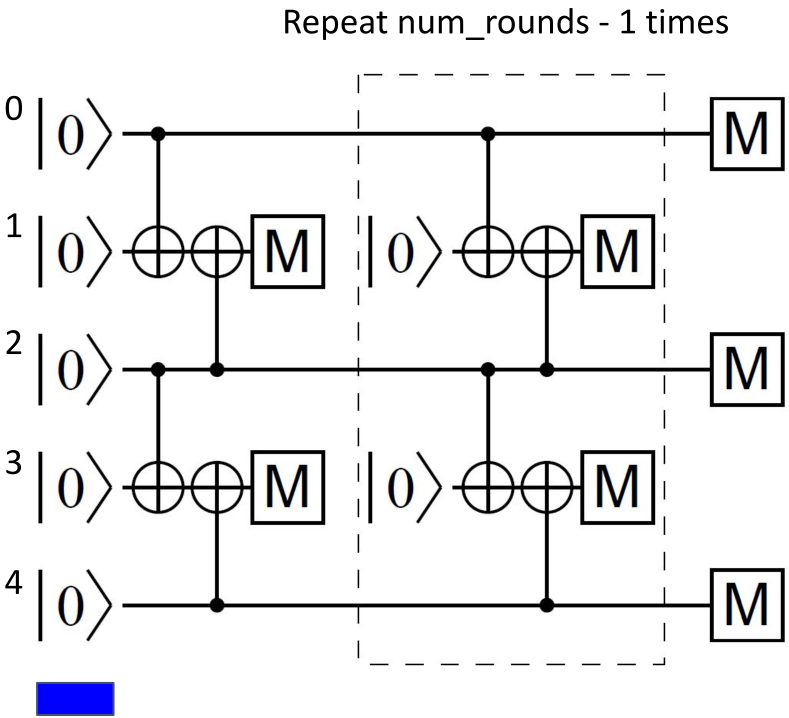
Repeat num_rounds - 1 times



Stim: specify each gate and error processes

```
# Reset all qubits:
R 0 1 2 3 4

X_ERROR(0.001) 0 1 2 3 4
CX 0 1 2 3
DEPOLARIZE2(0.001) 0 1 2 3
DEPOLARIZE1(0.001) 4
CX 2 1 4 3
DEPOLARIZE1(0.001) 0
DEPOLARIZE2(0.001) 2 1 4 3
X_ERROR(0.001) 1 3
M 1 3
DEPOLARIZE1(0.001) 0 2 4
DETECTOR(1, 0) rec[-2]
DETECTOR(3, 0) rec[-1]
```

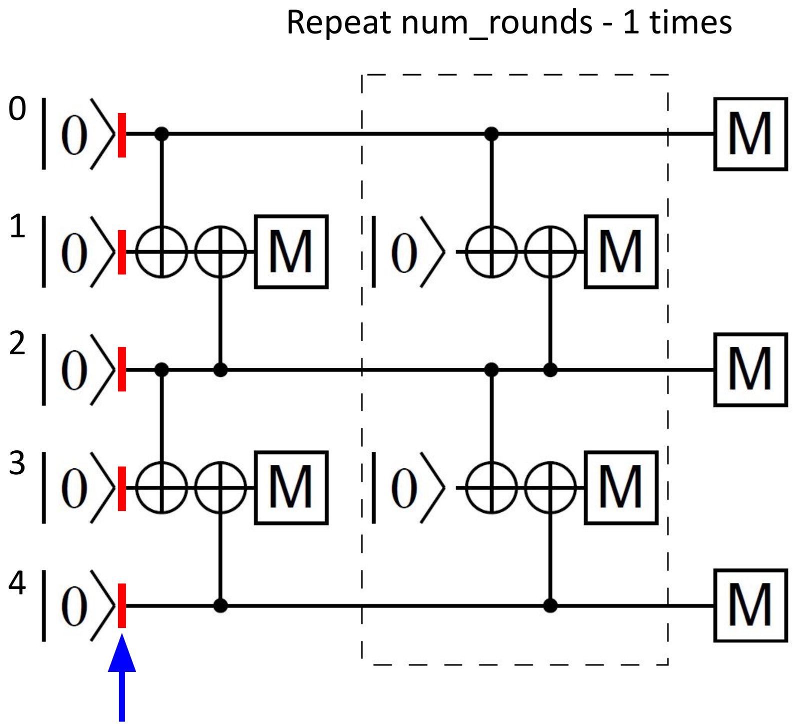


Stim: specify each gate and error processes

```
R 0 1 2 3 4

# Apply p=10^-3 post-reset X noise to each qubit:
X_ERROR(0.001) 0 1 2 3 4

CX 0 1 2 3
DEPOLARIZE2(0.001) 0 1 2 3
DEPOLARIZE1(0.001) 4
CX 2 1 4 3
DEPOLARIZE1(0.001) 0
DEPOLARIZE2(0.001) 2 1 4 3
X_ERROR(0.001) 1 3
M 1 3
DEPOLARIZE1(0.001) 0 2 4
DETECTOR(1, 0) rec[-2]
DETECTOR(3, 0) rec[-1]
```



Stim: specify each gate and error processes

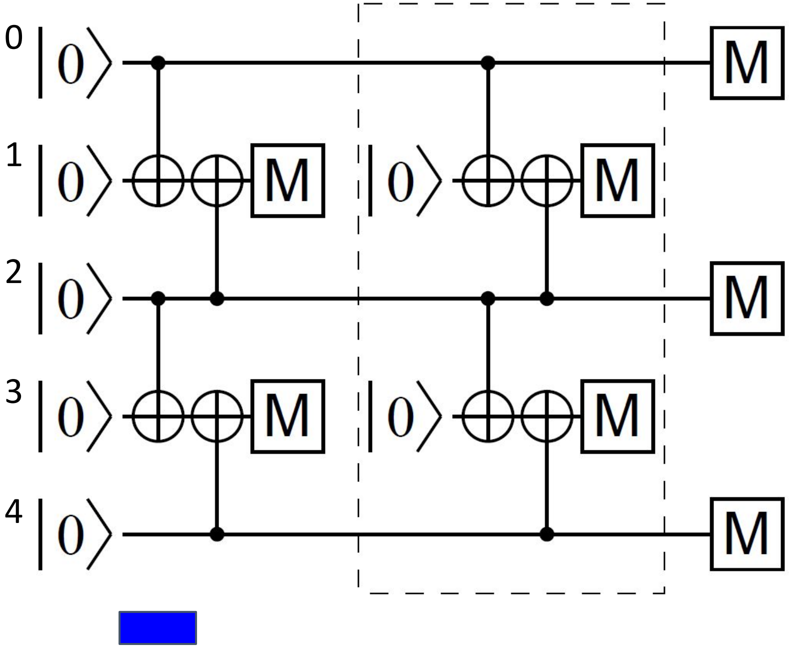
```
R 0 1 2 3 4
X_ERROR(0.001) 0 1 2 3 4
```

 # CNOT gates down:

```
CX 0 1 2 3

DEPOLARIZE2(0.001) 0 1 2 3
DEPOLARIZE1(0.001) 4
CX 2 1 4 3
DEPOLARIZE1(0.001) 0
DEPOLARIZE2(0.001) 2 1 4 3
X_ERROR(0.001) 1 3
M 1 3
DEPOLARIZE1(0.001) 0 2 4
DETECTOR(1, 0) rec[-2]
DETECTOR(3, 0) rec[-1]
```

Repeat num_rounds - 1 times

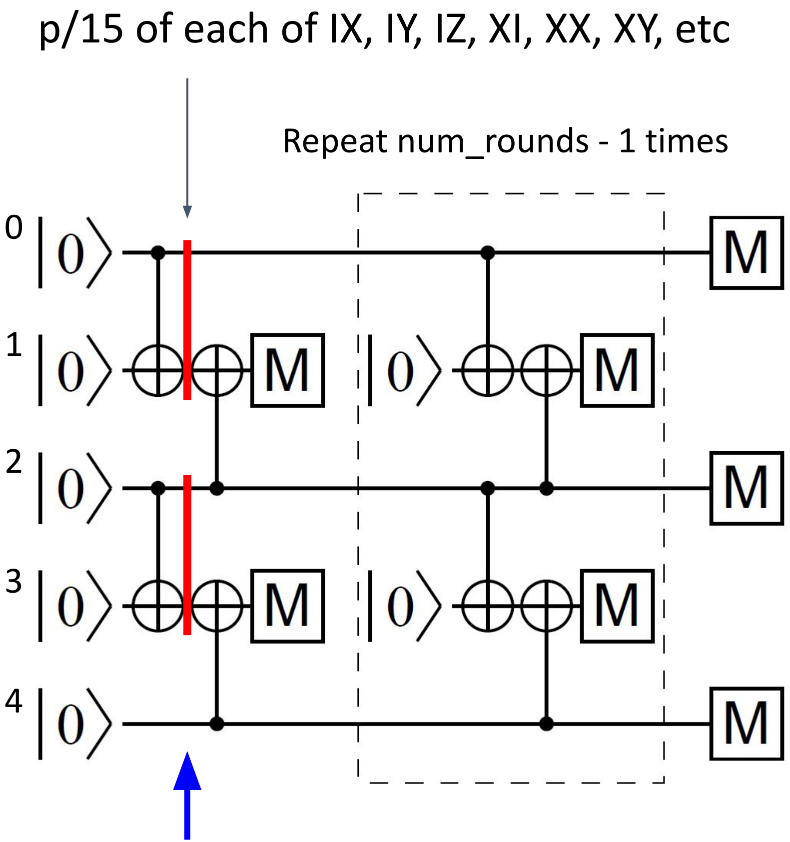


Stim: specify each gate and error processes

```
R 0 1 2 3 4
X_ERROR(0.001) 0 1 2 3 4
CX 0 1 2 3

# Apply 2q noise to each qubit touched by CX:
DEPOLARIZE2(0.001) 0 1 2 3

DEPOLARIZE1(0.001) 4
CX 2 1 4 3
DEPOLARIZE1(0.001) 0
DEPOLARIZE2(0.001) 2 1 4 3
X_ERROR(0.001) 1 3
M 1 3
DEPOLARIZE1(0.001) 0 2 4
DETECTOR(1, 0) rec[-2]
DETECTOR(3, 0) rec[-1]
```

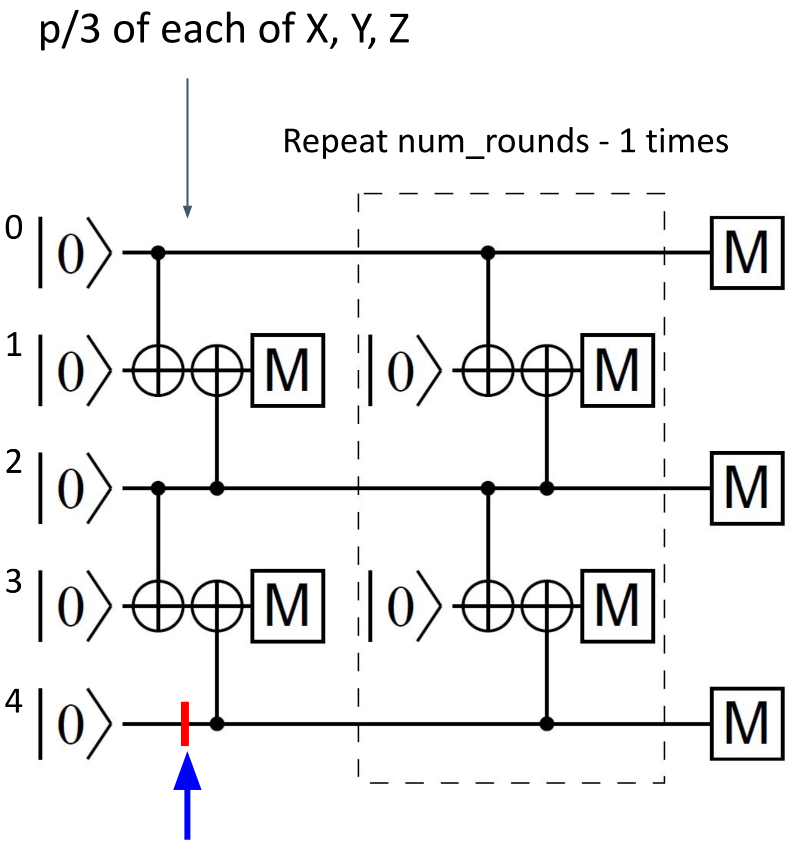


Stim: specify each gate and error processes

```
R 0 1 2 3 4
X_ERROR(0.001) 0 1 2 3 4
CX 0 1 2 3
DEPOLARIZE2(0.001) 0 1 2 3

# Apply 1q noise to the idle qubit:
DEPOLARIZE1(0.001) 4

CX 2 1 4 3
DEPOLARIZE1(0.001) 0
DEPOLARIZE2(0.001) 2 1 4 3
X_ERROR(0.001) 1 3
M 1 3
DEPOLARIZE1(0.001) 0 2 4
DETECTOR(1, 0) rec[-2]
DETECTOR(3, 0) rec[-1]
```

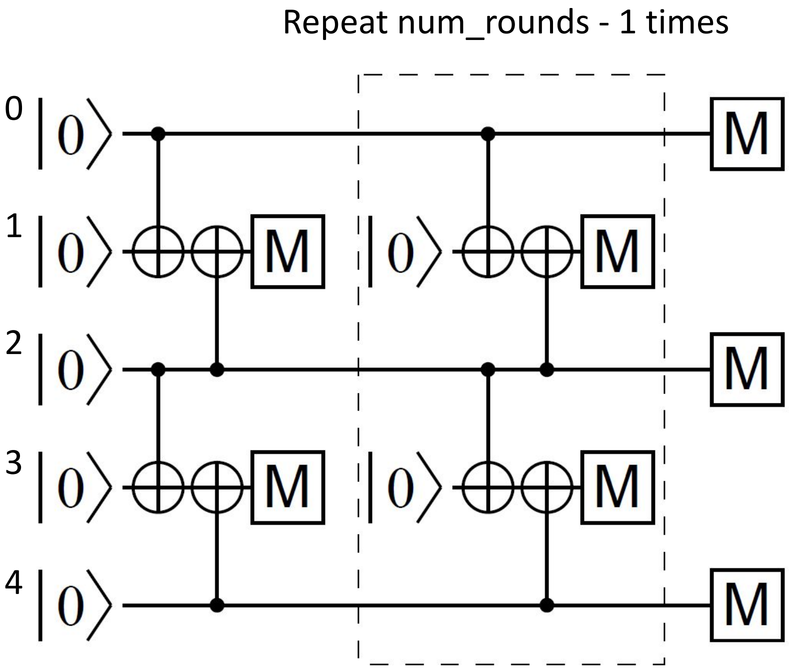


Stim: specify each gate and error processes

```
R 0 1 2 3 4
X_ERROR(0.001) 0 1 2 3 4
CX 0 1 2 3
DEPOLARIZE2(0.001) 0 1 2 3
DEPOLARIZE1(0.001) 4

# CNOT gates up:
CX 2 1 4 3

DEPOLARIZE1(0.001) 0
DEPOLARIZE2(0.001) 2 1 4 3
X_ERROR(0.001) 1 3
M 1 3
DEPOLARIZE1(0.001) 0 2 4
DETECTOR(1, 0) rec[-2]
DETECTOR(3, 0) rec[-1]
```

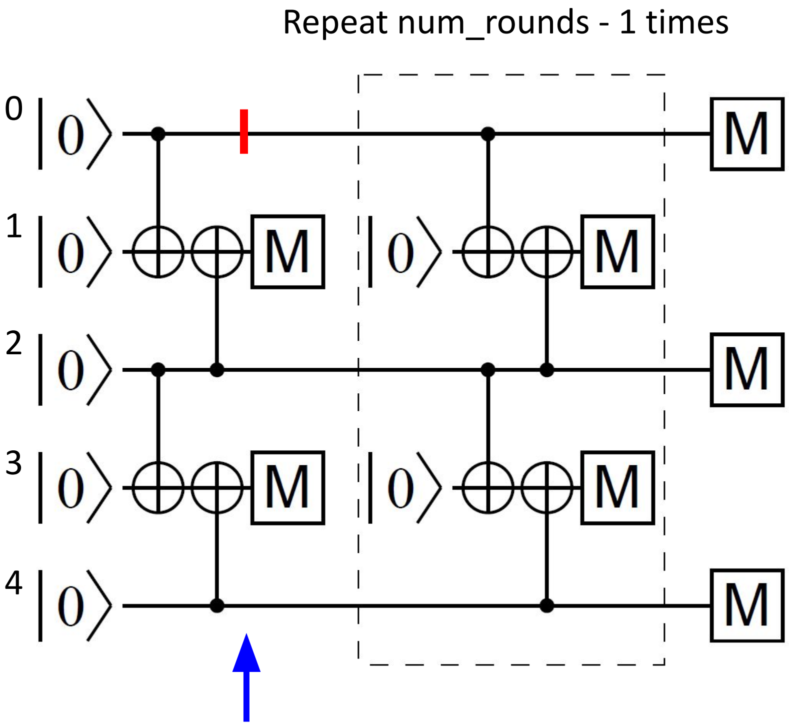


Stim: specify each gate and error processes

```
R 0 1 2 3 4
X_ERROR(0.001) 0 1 2 3 4
CX 0 1 2 3
DEPOLARIZE2(0.001) 0 1 2 3
DEPOLARIZE1(0.001) 4
CX 2 1 4 3

# Apply 1q noise to the idle qubit:
DEPOLARIZE1(0.001) 0

DEPOLARIZE2(0.001) 2 1 4 3
X_ERROR(0.001) 1 3
M 1 3
DEPOLARIZE1(0.001) 0 2 4
DETECTOR(1, 0) rec[-2]
DETECTOR(3, 0) rec[-1]
```

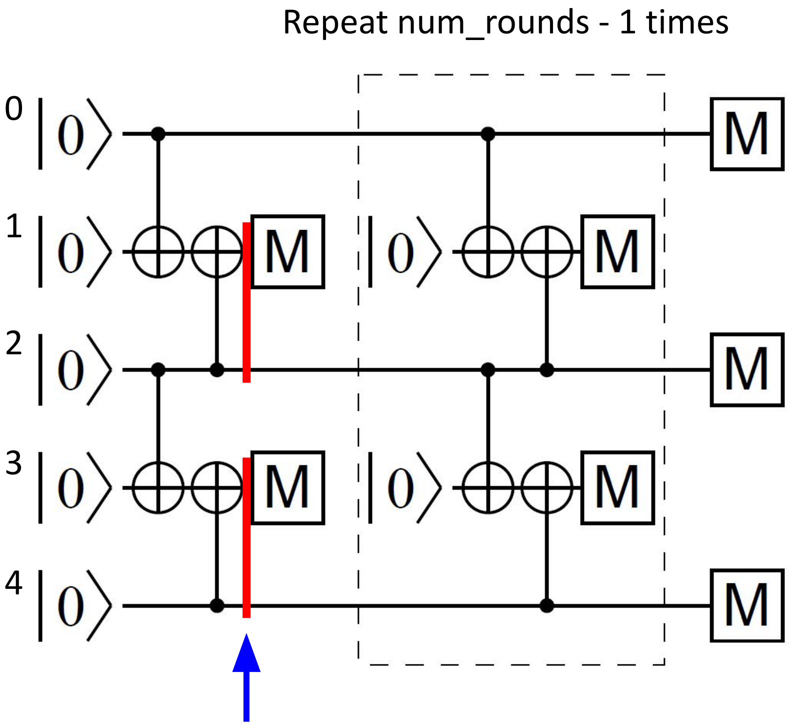


Stim: specify each gate and error processes

```
R 0 1 2 3 4
X_ERROR(0.001) 0 1 2 3 4
CX 0 1 2 3
DEPOLARIZE2(0.001) 0 1 2 3
DEPOLARIZE1(0.001) 4
CX 2 1 4 3
DEPOLARIZE1(0.001) 0

# Apply 2q noise to each qubit touched by CX:
DEPOLARIZE2(0.001) 2 1 4 3

X_ERROR(0.001) 1 3
M 1 3
DEPOLARIZE1(0.001) 0 2 4
DETECTOR(1, 0) rec[-2]
DETECTOR(3, 0) rec[-1]
```

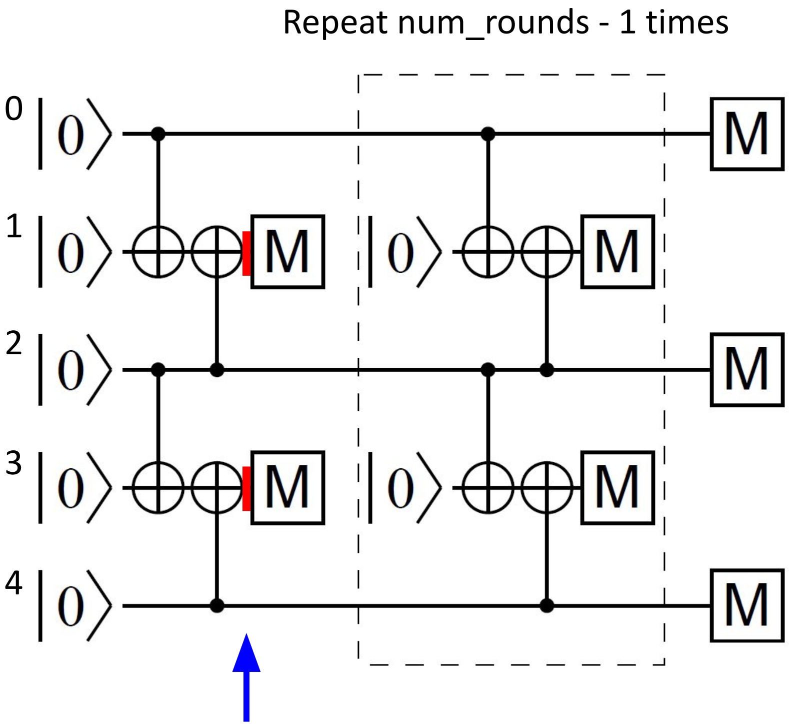


Stim: specify each gate and error processes

```
R 0 1 2 3 4
X_ERROR(0.001) 0 1 2 3 4
CX 0 1 2 3
DEPOLARIZE2(0.001) 0 1 2 3
DEPOLARIZE1(0.001) 4
CX 2 1 4 3
DEPOLARIZE1(0.001) 0
DEPOLARIZE2(0.001) 2 1 4 3

# Apply pre-measurement noise:
X_ERROR(0.001) 1 3

M 1 3
DEPOLARIZE1(0.001) 0 2 4
DETECTOR(1, 0) rec[-2]
DETECTOR(3, 0) rec[-1]
```

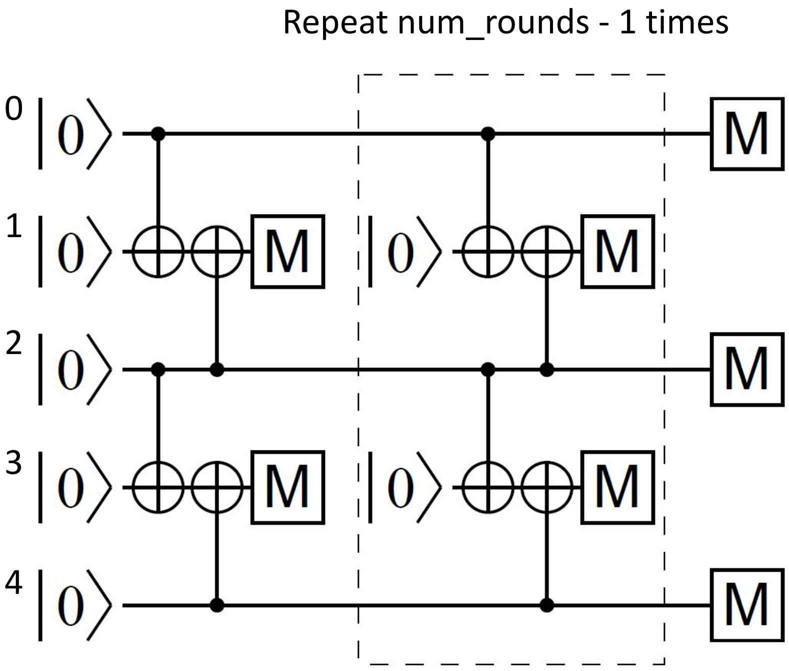


Stim: specify each gate and error processes

```
R 0 1 2 3 4
X_ERROR(0.001) 0 1 2 3 4
CX 0 1 2 3
DEPOLARIZE2(0.001) 0 1 2 3
DEPOLARIZE1(0.001) 4
CX 2 1 4 3
DEPOLARIZE1(0.001) 0
DEPOLARIZE2(0.001) 2 1 4 3
X_ERROR(0.001) 1 3

# Measure qubits 1 and 3:
M 1 3

DEPOLARIZE1(0.001) 0 2 4
DETECTOR(1, 0) rec[-2]
DETECTOR(3, 0) rec[-1]
```

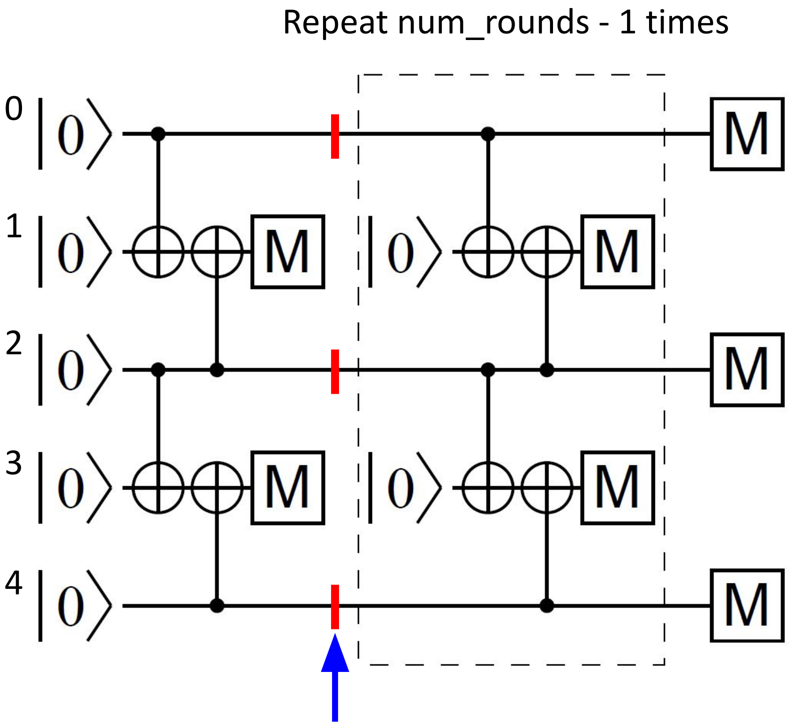


Stim: specify each gate and error processes

```
R 0 1 2 3 4
X_ERROR(0.001) 0 1 2 3 4
CX 0 1 2 3
DEPOLARIZE2(0.001) 0 1 2 3
DEPOLARIZE1(0.001) 4
CX 2 1 4 3
DEPOLARIZE1(0.001) 0
DEPOLARIZE2(0.001) 2 1 4 3
X_ERROR(0.001) 1 3
M 1 3

# Apply 1q noise to each idle qubit:
DEPOLARIZE1(0.001) 0 2 4

DETECTOR(1, 0) rec[-2]
DETECTOR(3, 0) rec[-1]
```

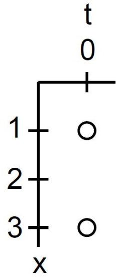


Stim: specify each gate

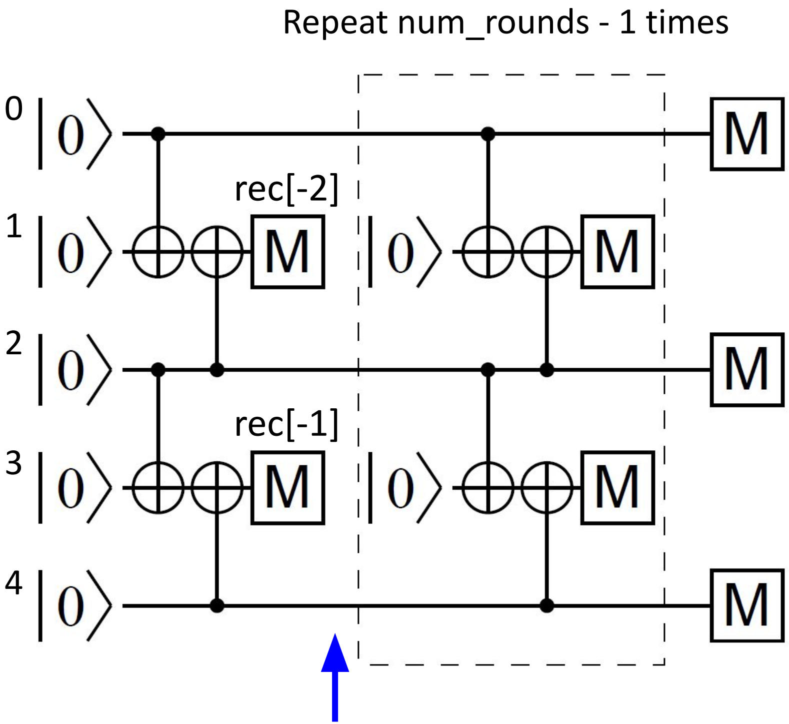
```
R 0 1 2 3 4
X_ERROR(0.001) 0 1 2 3 4
CX 0 1 2 3
DEPOLARIZE2(0.001) 0 1 2 3
DEPOLARIZE1(0.001) 4
CX 2 1 4 3
DEPOLARIZE1(0.001) 0
DEPOLARIZE2(0.001) 2 1 4 3
X_ERROR(0.001) 1 3
M 1 3
DEPOLARIZE1(0.001) 0 2 4
```

 # Set coordinates (x, t) of detectors and the measurements they depend on:

```
DETECTOR(1, 0) rec[-2]
DETECTOR(3, 0) rec[-1]
```



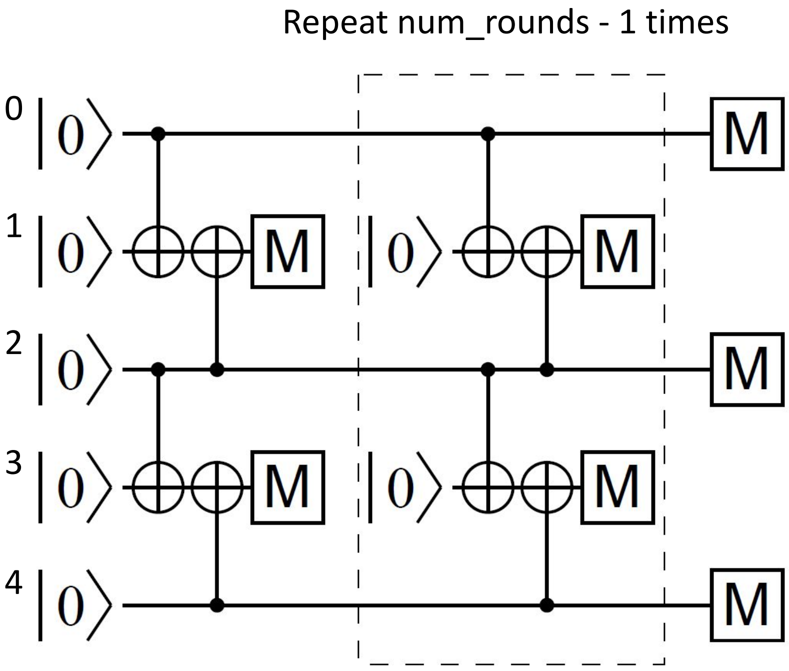
The coordinates of the detectors are user defined and abstract, but typically set to a representation of a coordinate of one of the measurements, eg: (qubit_num, round)



Stim: specify each gate and error processes

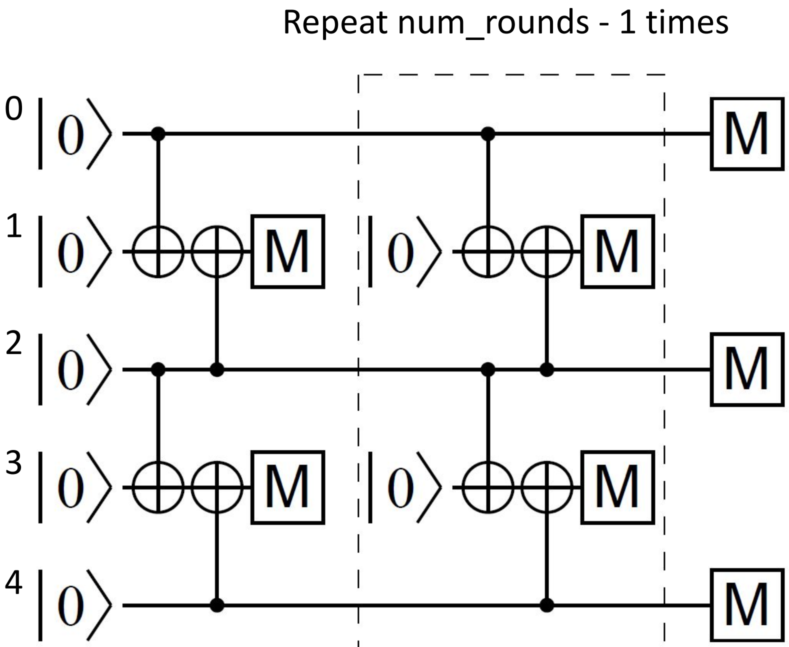
```
R 0 1 2 3 4
X_ERROR(0.001) 0 1 2 3 4
TICK
CX 0 1 2 3
DEPOLARIZE2(0.001) 0 1 2 3
DEPOLARIZE1(0.001) 4
TICK
CX 2 1 4 3
DEPOLARIZE1(0.001) 0
DEPOLARIZE2(0.001) 2 1 4 3
TICK
X_ERROR(0.001) 1 3
M 1 3
DEPOLARIZE1(0.001) 0 2 4
DETECTOR(1, 0) rec[-2]
DETECTOR(3, 0) rec[-1]
```

TICK instructions help Stim display circuits nicely (we'll omit them)



Stim: specify each gate and error processes

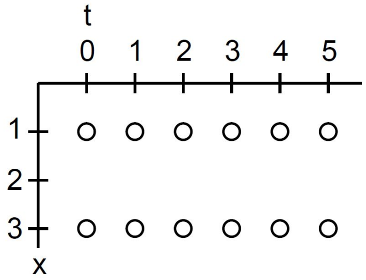
```
R 0 1 2 3 4
X_ERROR(0.001) 0 1 2 3 4
CX 0 1 2 3
DEPOLARIZE2(0.001) 0 1 2 3
DEPOLARIZE1(0.001) 4
CX 2 1 4 3
DEPOLARIZE1(0.001) 0
DEPOLARIZE2(0.001) 2 1 4 3
X_ERROR(0.001) 1 3
M 1 3
DEPOLARIZE1(0.001) 0 2 4
DETECTOR(1, 0) rec[-2]
DETECTOR(3, 0) rec[-1]
```



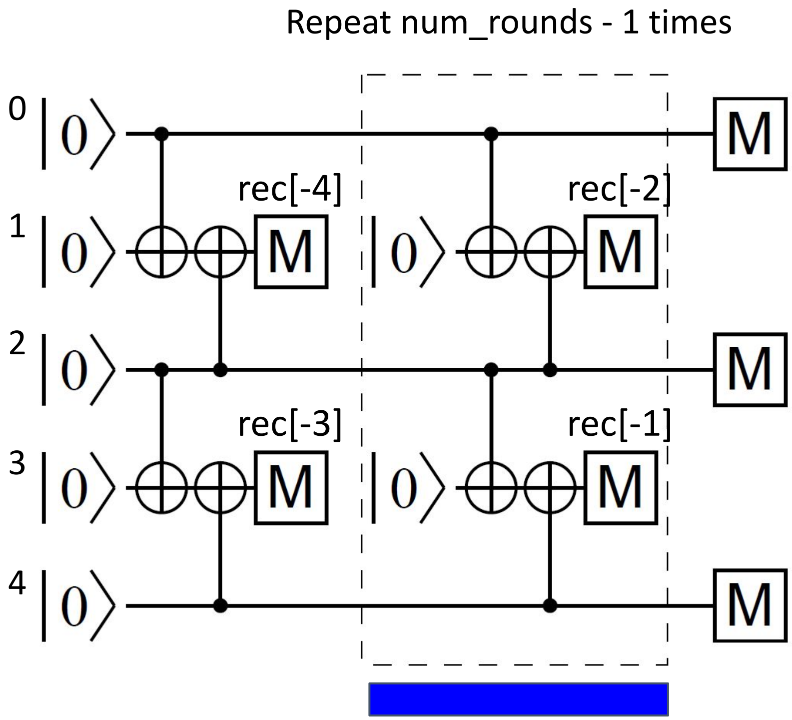
Next block similar, just repeated…

Stim: repeating gates

```
REPEAT 5 {
        R 1 3
        X_ERROR(0.001) 1 3
        DEPOLARIZE1(0.001) 0 2 4
        CX 0 1 2 3
        DEPOLARIZE2(0.001) 0 1 2 3
        DEPOLARIZE1(0.001) 4
        CX 2 1 4 3
        DEPOLARIZE1(0.001) 0
        DEPOLARIZE2(0.001) 2 1 4 3
        X_ERROR(0.001) 1 3
        M 1 3
        DEPOLARIZE1(0.001) 0 2 4
        SHIFT_COORDS(0, 1) # increase t of future detectors
        DETECTOR(1, 0) rec[-2] rec[-4]
        DETECTOR(3, 0) rec[-1] rec[-3]
}
```



5 more rounds of detectors, increasing t by one each time

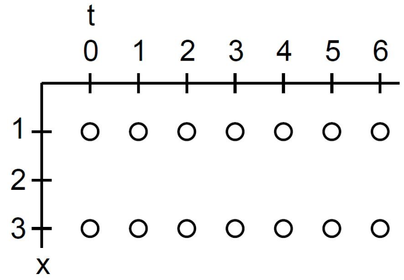


Stim: finishing up

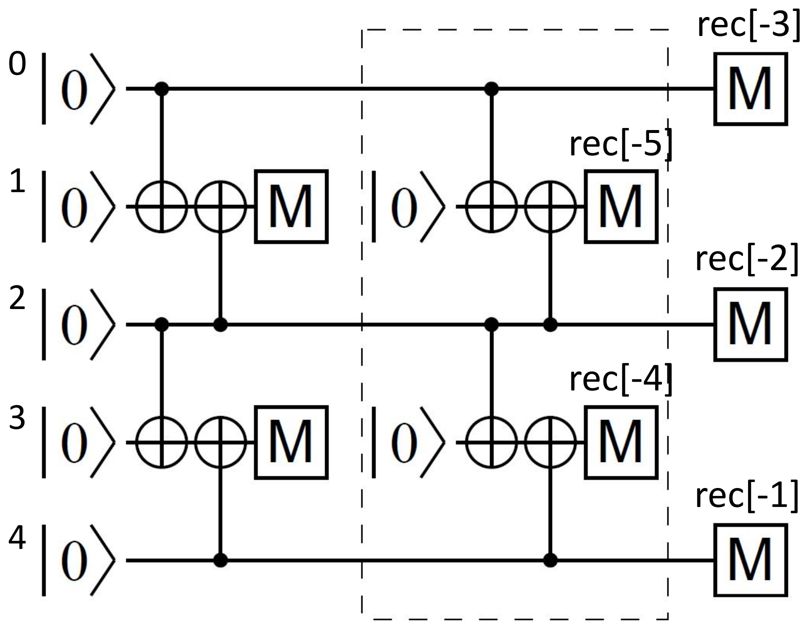
```
X_ERROR(0.001) 0 2 4
M 0 2 4
SHIFT_COORDS(0, 1)
DETECTOR(1, 0) rec[-2] rec[-3] rec[-5]
DETECTOR(3, 0) rec[-1] rec[-2] rec[-4]

# Choose the measurement(s) that make up the logical observables:
OBSERVABLE_INCLUDE(0) rec[-1]
```

Our logical states are 000 and 111, so observing any single qubit tells us which logical state we have measured.

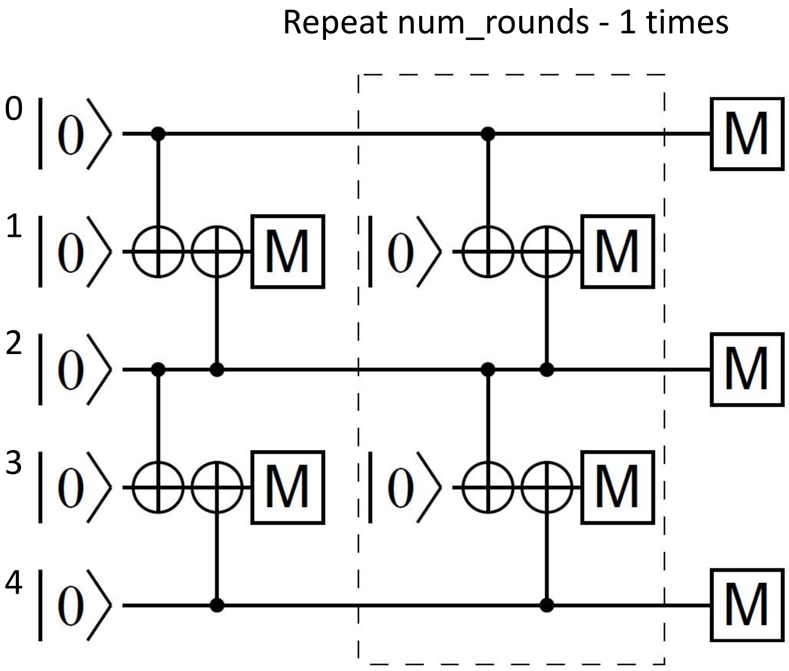


1 more round of detectors, t increased by one



Stim: exercises

1) Assemble the code from the slides into a single file
2) Add TICK instructions
3) Modify the file so that the final round of measurements occurs at the same time as the 2nd last round of measurements (a shorter circuit means less error)
4) Create a Python script to create a text string for an arbitrary code distance d, number of rounds, and error rate p.

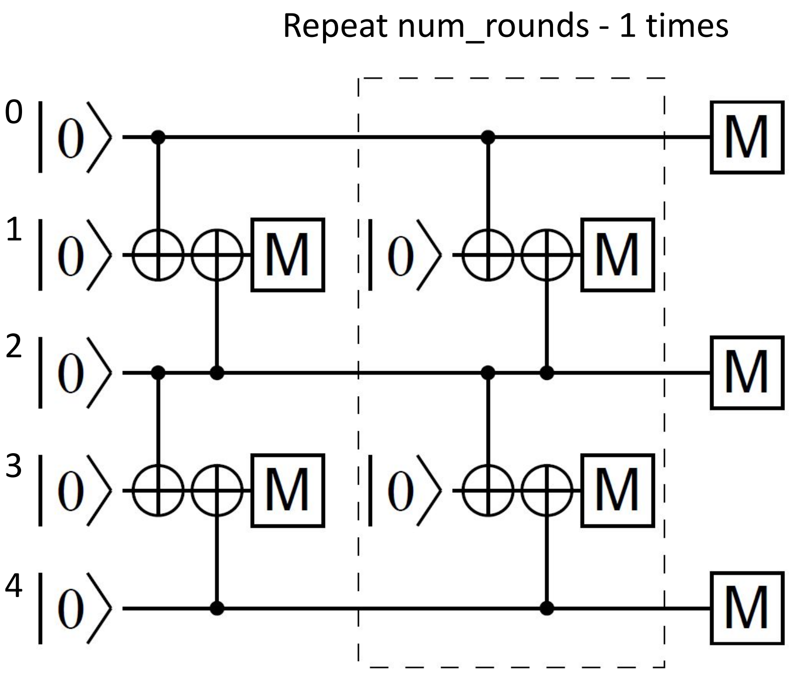


Work is in progress to make this easier:
https://groups.google.com/g/tqec-design-automation

Stim: existing functions

Stim has functions to generate similar circuits:

```
circuit = stim.Circuit.generated(
    "repetition_code:memory",
    rounds=num_rounds,
    distance=d,
    before_round_data_depolarization=p)
```

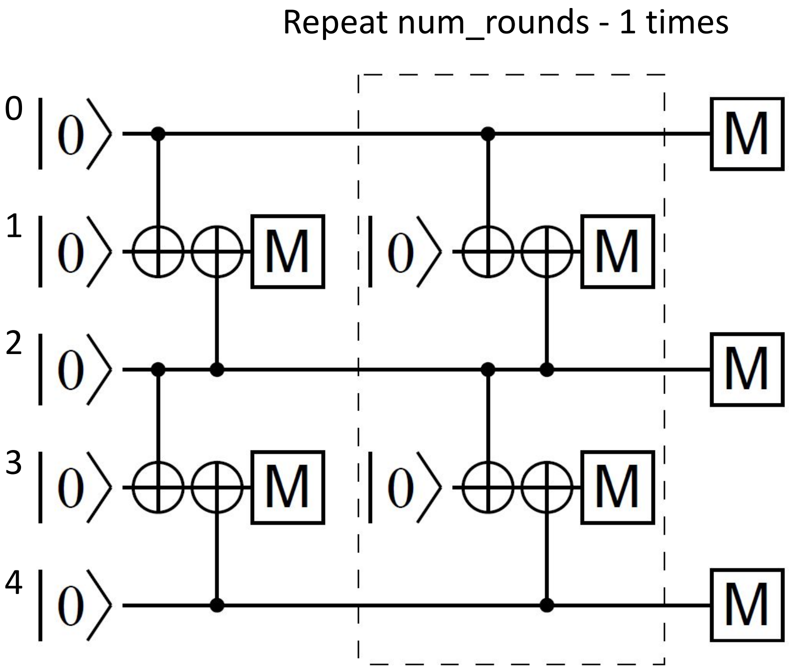


Noise not identical to our example.

Stim: exercise

Find the following function in section 7 of getting_started.ipynb

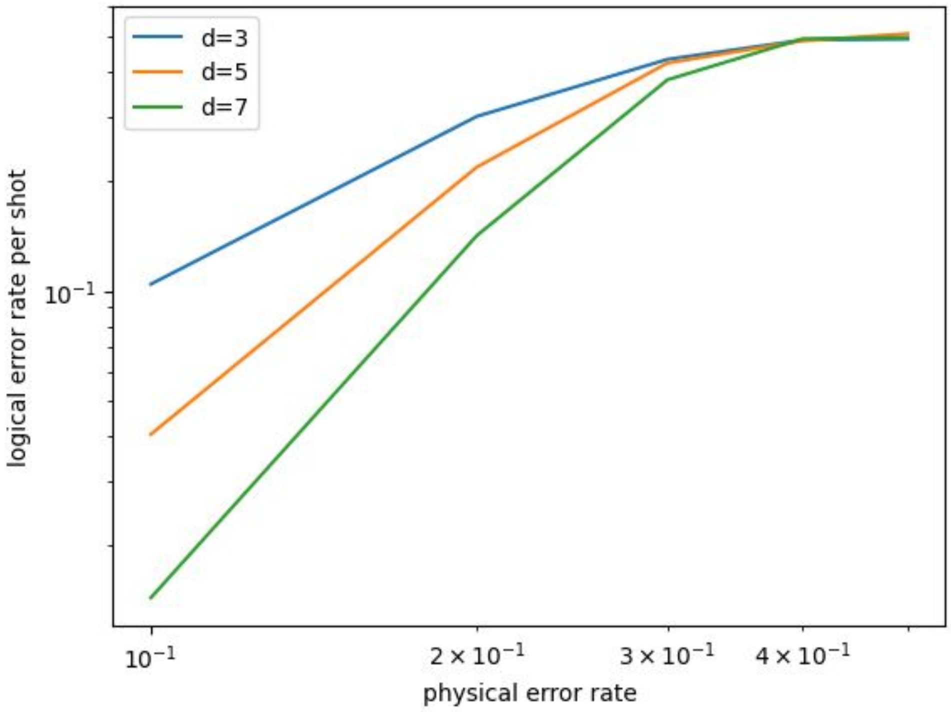
```
circuit = stim.Circuit.generated(
    "repetition_code:memory",
    rounds=num_rounds,
    distance=d,
    before_round_data_depolarization=p)
```

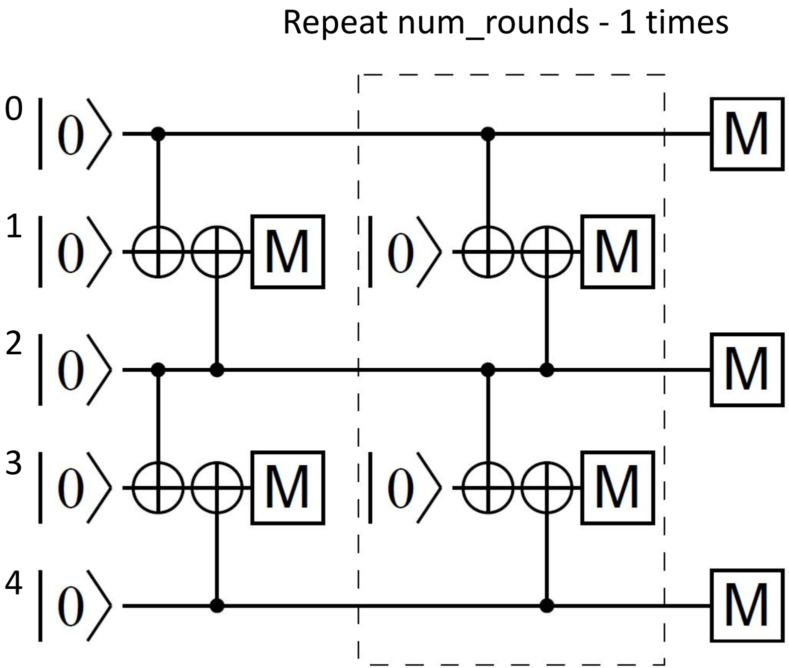


Replace it with our example, similarly parameterized using num_rounds, d, and p.

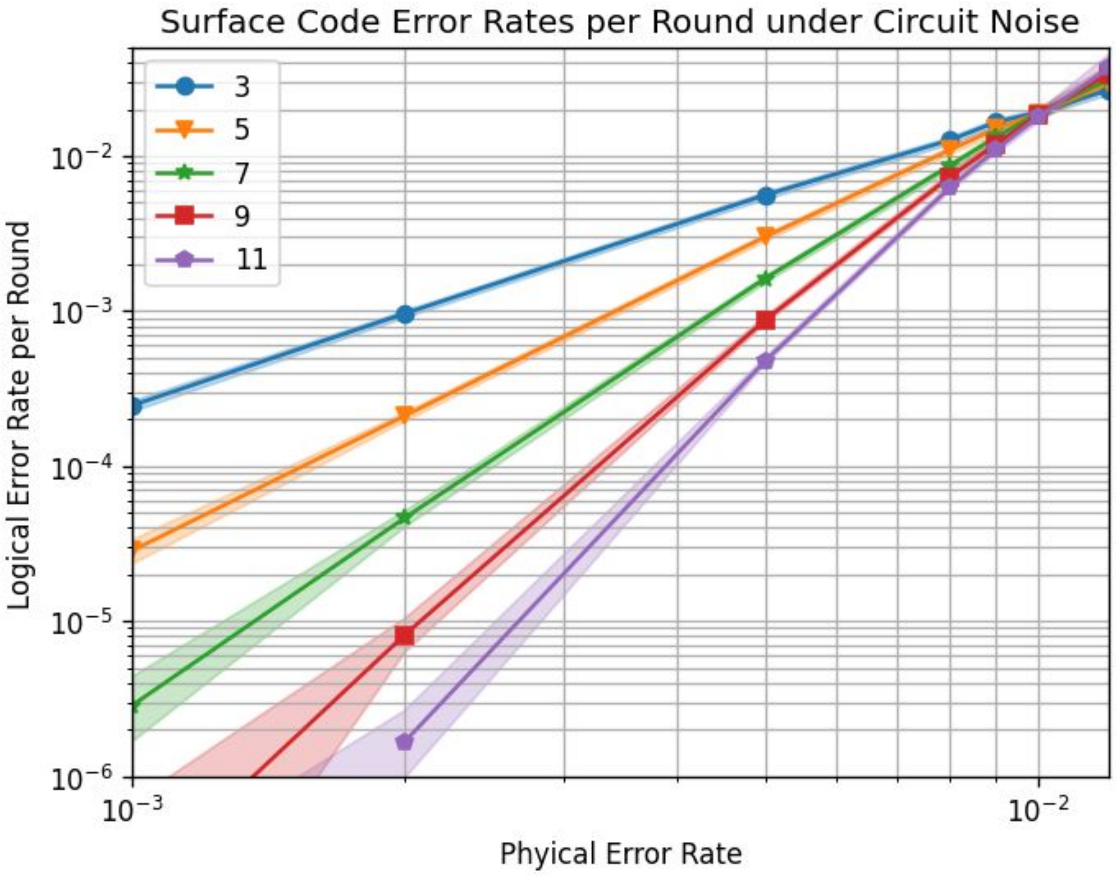
Stim: exercise

You should get a plot similar to:





Stim: exercise



Work through the rest of getting_started.ipynb

Every code has an error rate above which bigger codes lead to worse performance and below which bigger codes lead to better performance. This is the threshold error rate.

You can now build circuits for codes and calculate this threshold error rate using Stim.

Even more important: find the error rate at which increasing the code distance by 2 leads to a factor of 10 suppression and logical error. This is a good experimental target.

Next time: Crumble