



Team Quantum ETS

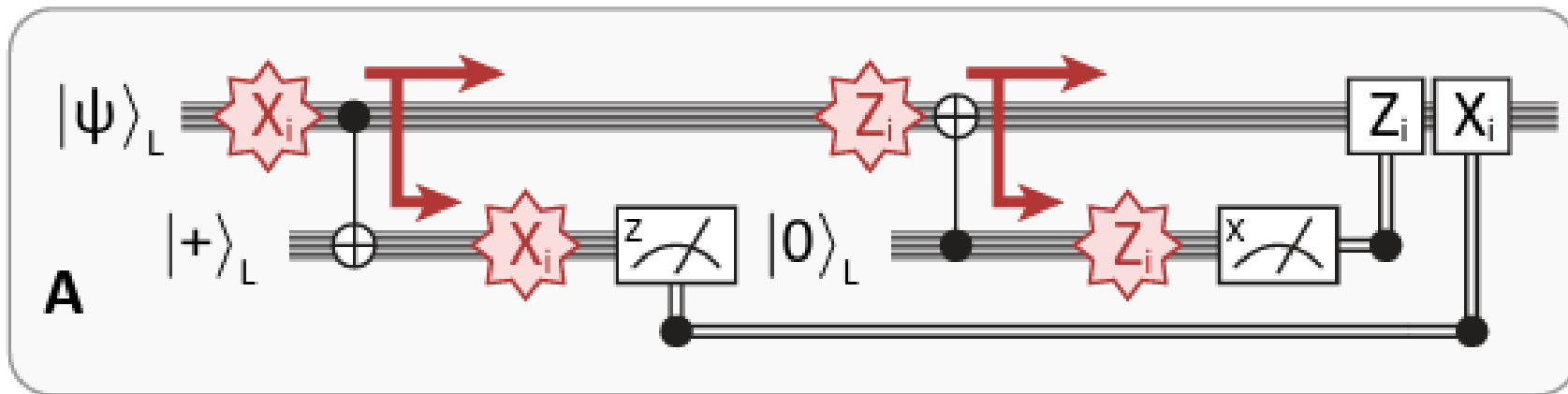
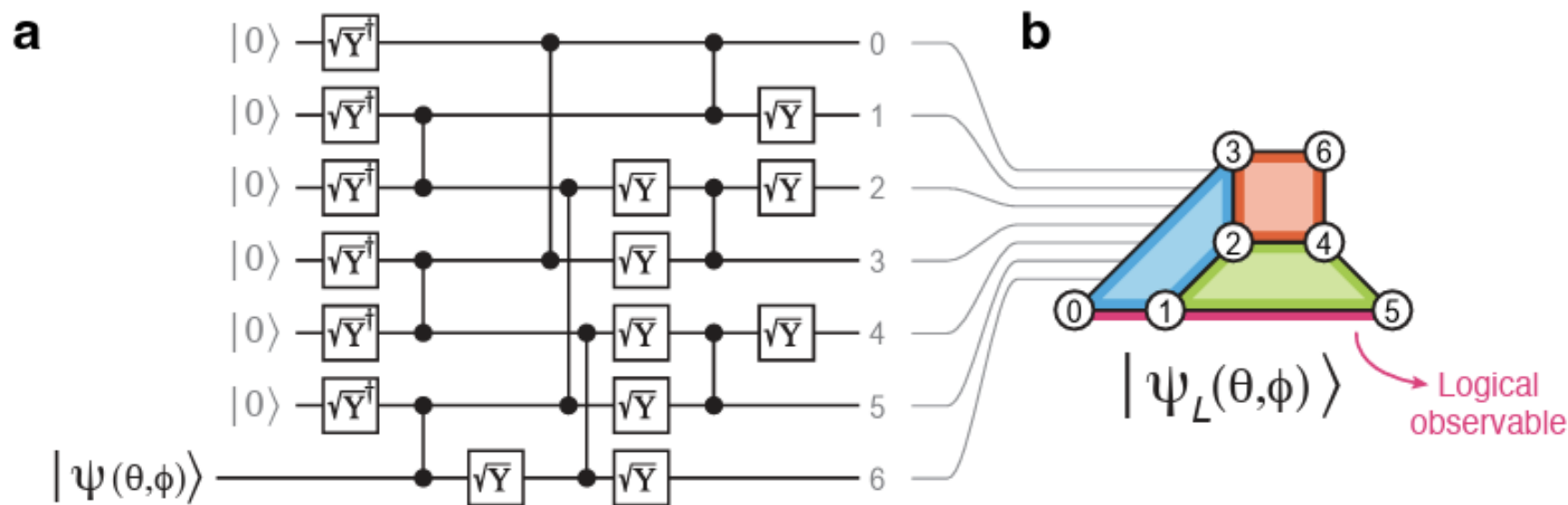
QuEra technical challenge



<https://github.com/QuantumETS/QueraTechnical2026/tree/main>

Samuel Richard
Guy-Philippe Nadon
Jacob Bernier
Félix-Antoine Lavallee

The problem

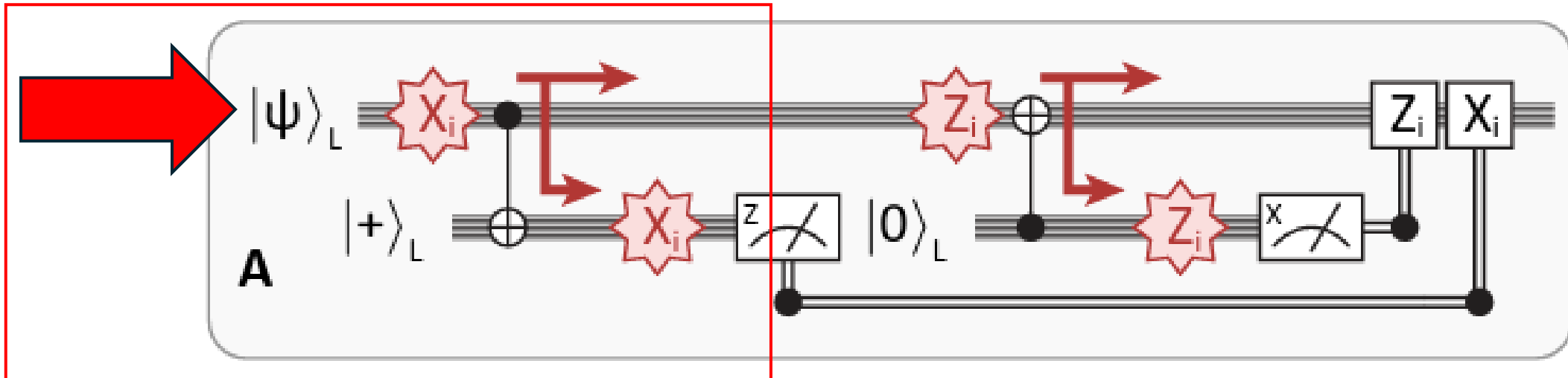


How did we implement Steane code

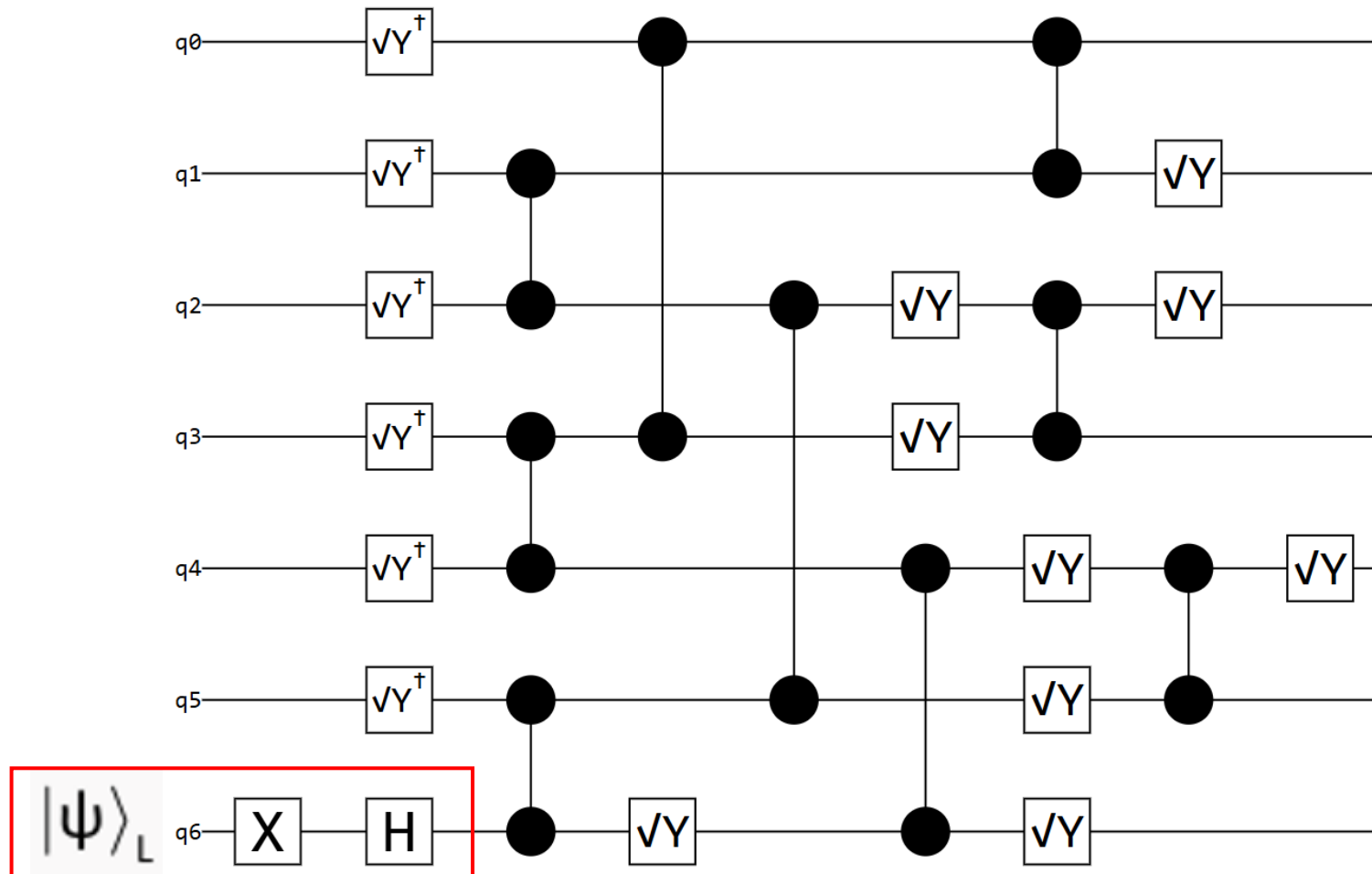
- Step one, encode the state that we want in the steane code

```
qubits_1 = squin.qalloc(n_sean_qubits)
# put arbitrary state
squin.x(qubits_1[-1])
squin.h(qubits_1[-1])
qubits_1 = color_code_factory(qubits_1)
```

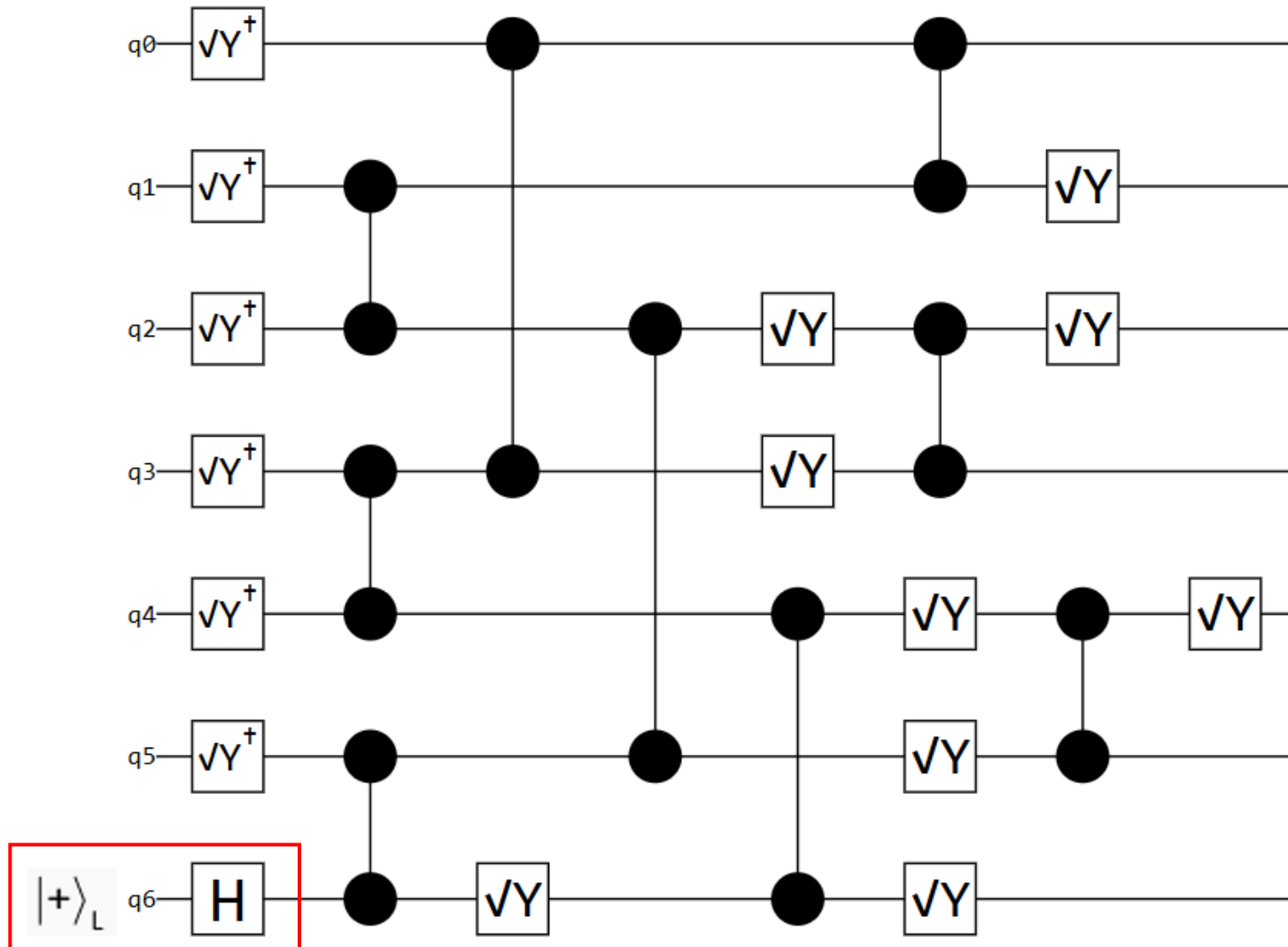
```
# put state + on ancillary registers
qubits_2 = squin.qalloc(n_sean_qubits)
squin.h(qubits_2[-1])
qubits_2 = color_code_factory(qubits_2)
```



Steane code zoomed in for $|\psi\rangle$



Steane code zoomed in for $|+\rangle$



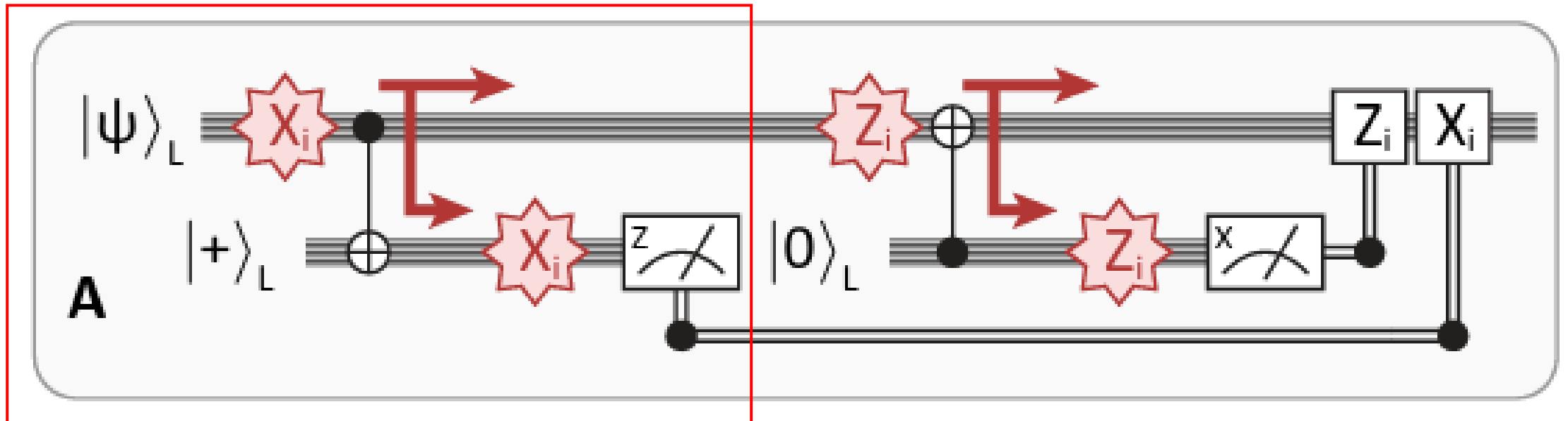
Lets look at post selecting when there's x error

```
qubits_1 = squin.qalloc(n_sean_qubits)
# put arbitrary state
squid.x(qubits_1[-1])
squid.h(qubits_1[-1])
qubits_1 = color_code_factory(qubits_1)

# put state + on ancillary registers
qubits_2 = squin.qalloc(n_sean_qubits)
squid.h(qubits_2[-1])
qubits_2 = color_code_factory(qubits_2)
```

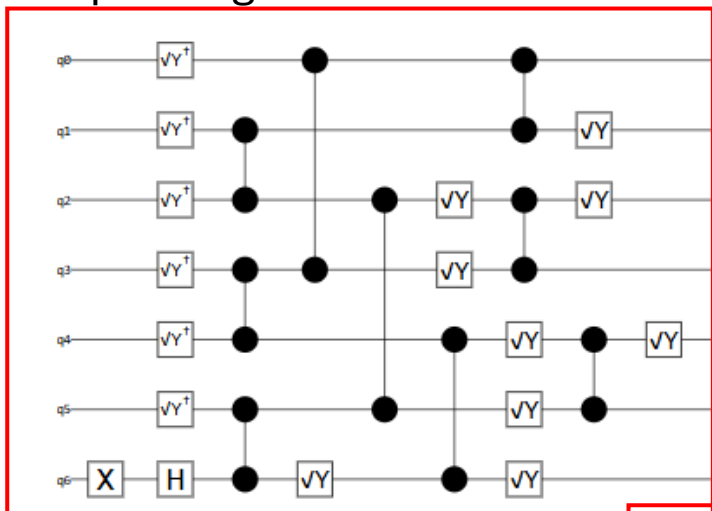
```
# transversal CX
for index in range(len(qubits_1)):
    squid.cx(qubits_1[index], qubits_2[index])

bits = squid.broadcast.measure(qubits_2)
return bits
```

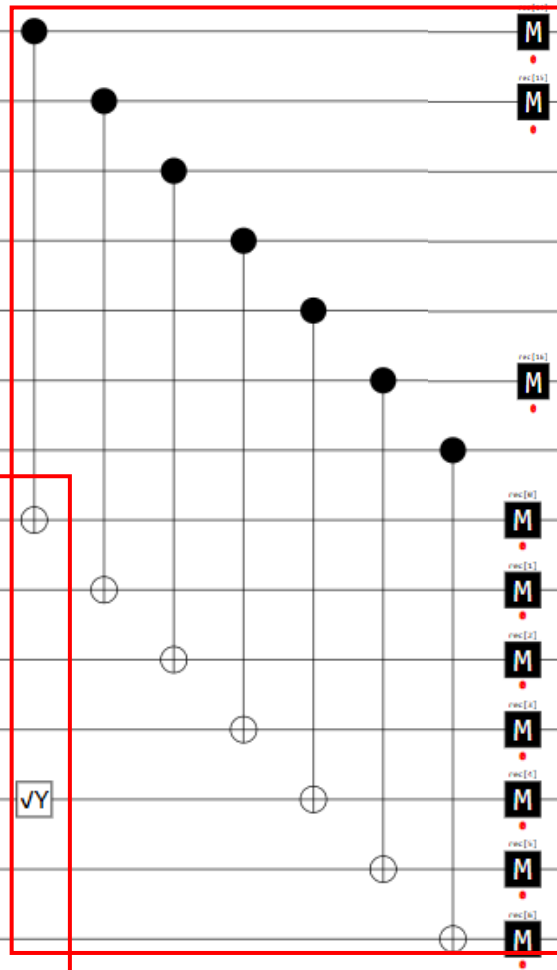


Post selecting for X errors only

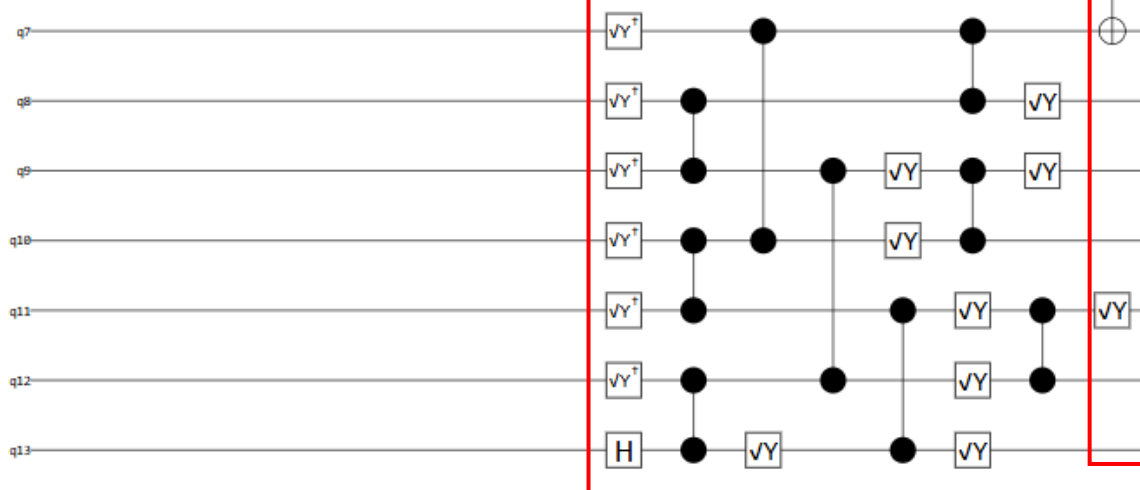
Prepare logical state



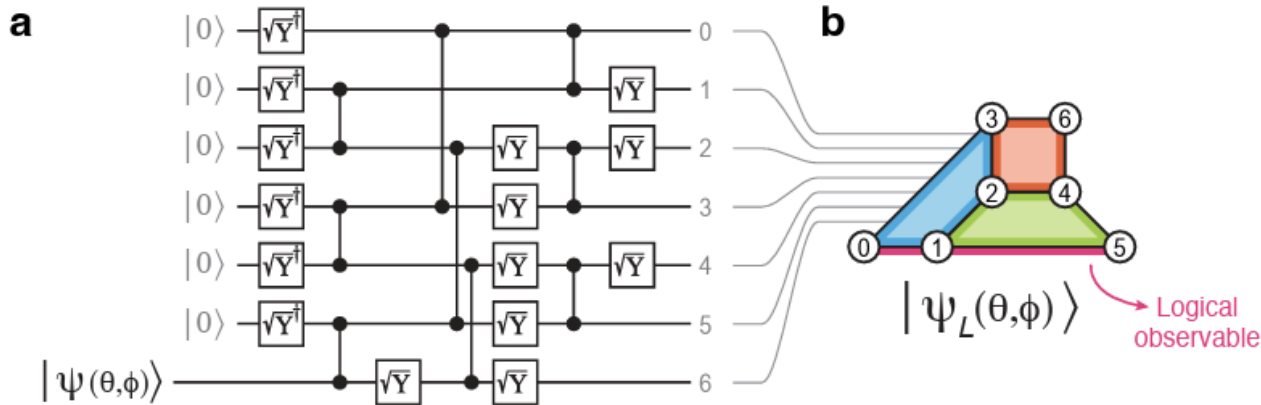
Transversal cnot for x error and “midcircuit” measurement



Prepare logical $|+\rangle$ state



Computing the syndromes



```
def extract_syndrome(group, log_expectation=False) -> int:
    """Compute the parity of measurement results in a plaquette group.

    This function calculates the syndrome for a plaquette by summing all
    measurement results in the group and returning the parity (mod 2).
    A return value of 0 indicates even parity (no error detected), while
    1 indicates odd parity (error detected).

    :param group: List of measurement results (0 or 1) for qubits in a plaquette
    :param log_expectation: If True, logs the -1/+1 mapping of 1/0 results
    :return: Parity of the group (0 for even, 1 for odd)
    :rtype: int

    :Example:

    >>> extract_syndrome([0, 1, 0, 1])
    0
    >>> extract_syndrome([1, 1, 1])
    1
    """
    if log_expectation:
        for i in group:
            logger.debug(f"Measurement result {i} mapped to expectation value {1 if i == 0 else -1}")
    t = 0
    for i in group:
        t = t + i
    return (t % 2)
```

```
def evaluate_syndromes(results: IList[MeasurementResult, Any]) -> tuple[int, int, int]:
    """Evaluate and return syndromes for all three color code plaquettes.

    This function evaluates the syndromes for blue, red, and green plaquettes
    of a 7-qubit color code. Each syndrome is computed as the parity of its
    respective plaquette measurements. A syndrome of 0 indicates even parity
    (no error detected), while 1 indicates odd parity (error detected).

    :param results: Measurement results from all 7 qubits in the color code
    :type results: IList[MeasurementResult, Any]
    :return: Tuple of (blue_syndrome, red_syndrome, green_syndrome)
    :rtype: tuple[int, int, int]

    :Example:

    >>> results = IList([0, 0, 0, 0, 0, 0, 0])
    >>> syndromes = evaluate_syndromes(results)
    >>> print(syndromes)
    (0, 0, 0)
    """
    blue = [results[0], results[1], results[2], results[3]]
    red = [results[2], results[3], results[4], results[6]]
    green = [results[1], results[2], results[4], results[5]]

    blue_plaquette_syndrome = extract_syndrome(blue)
    red_plaquette_syndrome = extract_syndrome(red)
    green_plaquette_syndrome = extract_syndrome(green)

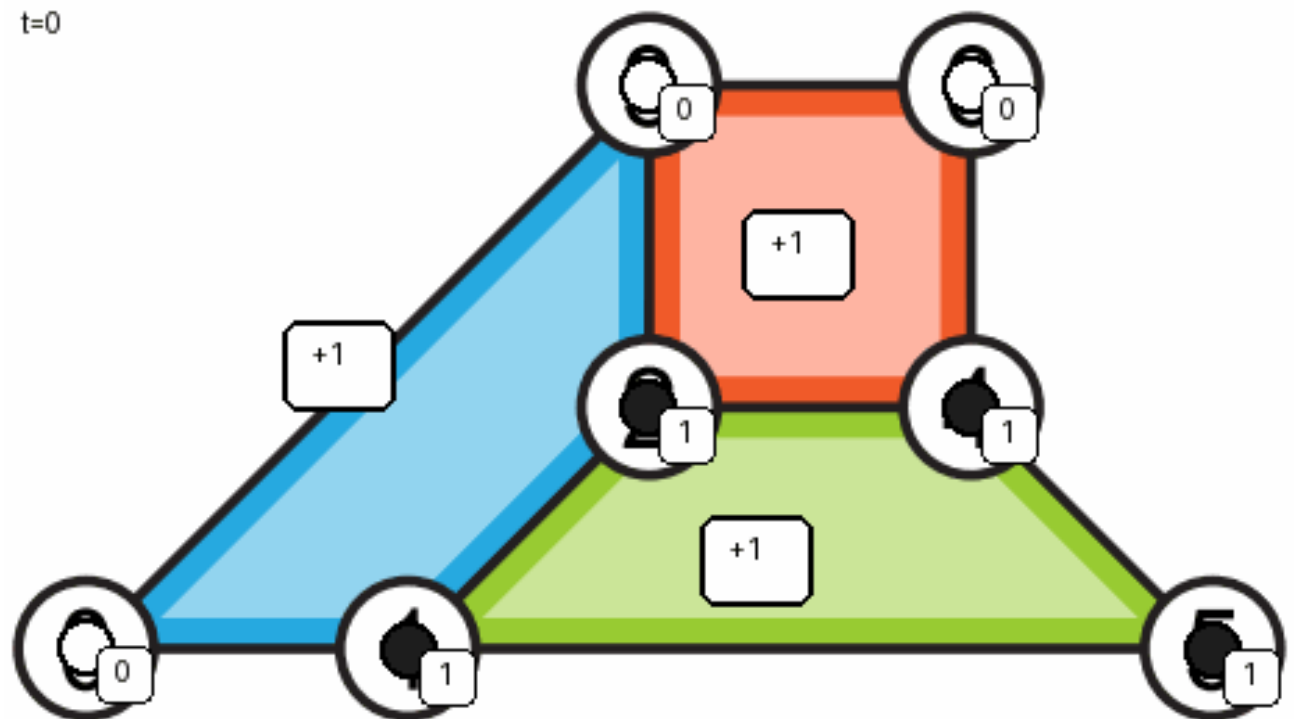
    return (blue_plaquette_syndrome, red_plaquette_syndrome, green_plaquette_syndrome)
```


Parity Checking X error – Noiseless Simulation Data

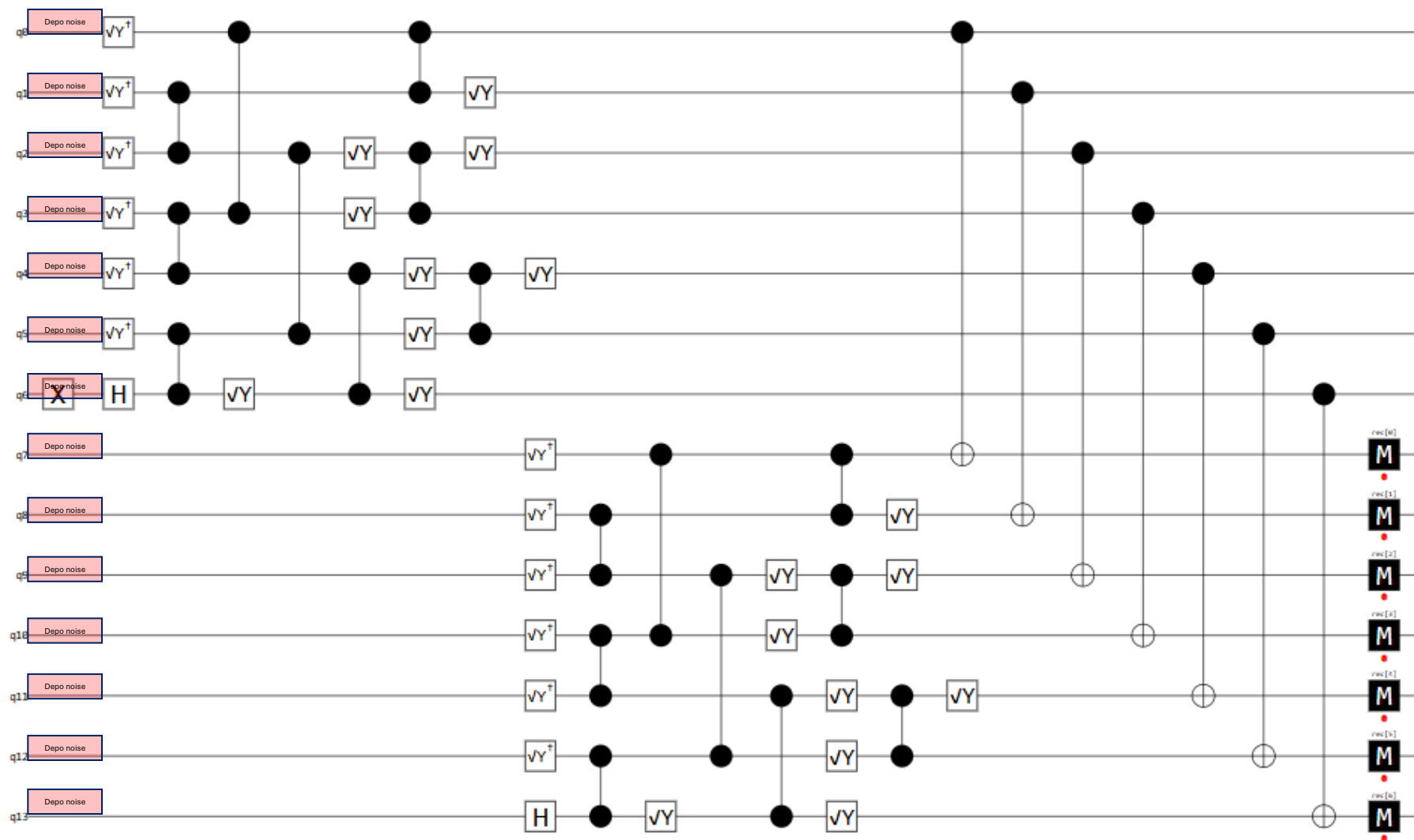
```
[
  {
    "blue": 0,
    "red": 0,
    "green": 0,
    "results": [
      0,
      1,
      1,
      0,
      1,
      1,
      0
    ]
  },
  {
    "blue": 0,
    "red": 0,
    "green": 0,
    "results": [
      1
    ]
  }
]
```

100% success rate!

never a -1 in the colored region = no error on our logical observable



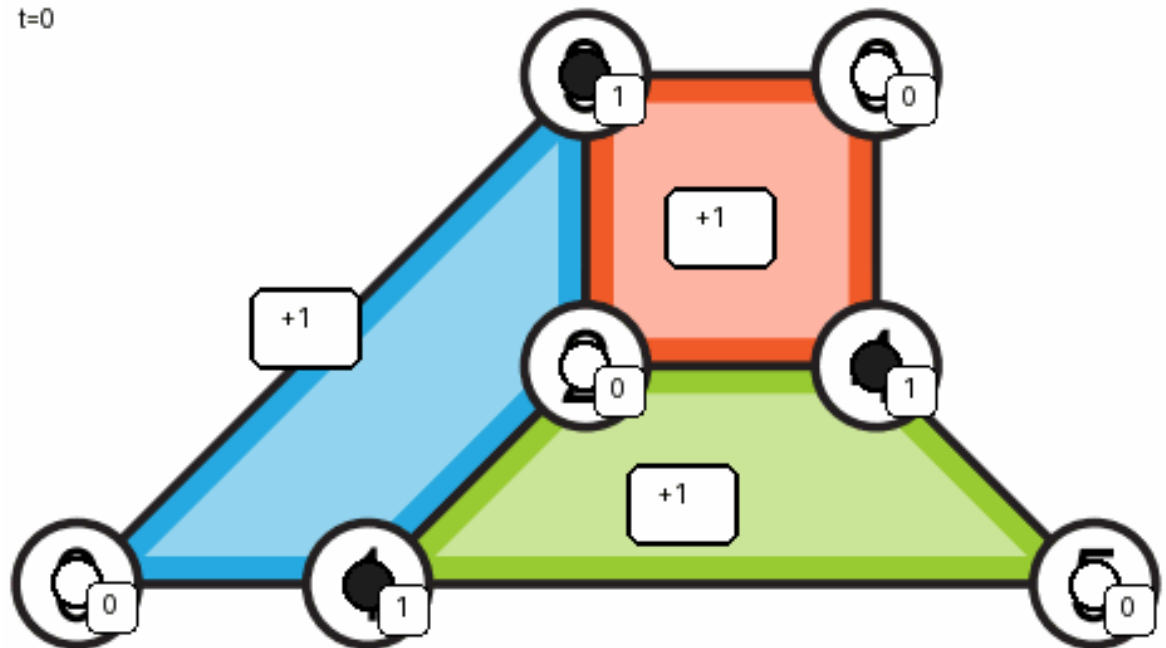
Lets add some noise, location = 0



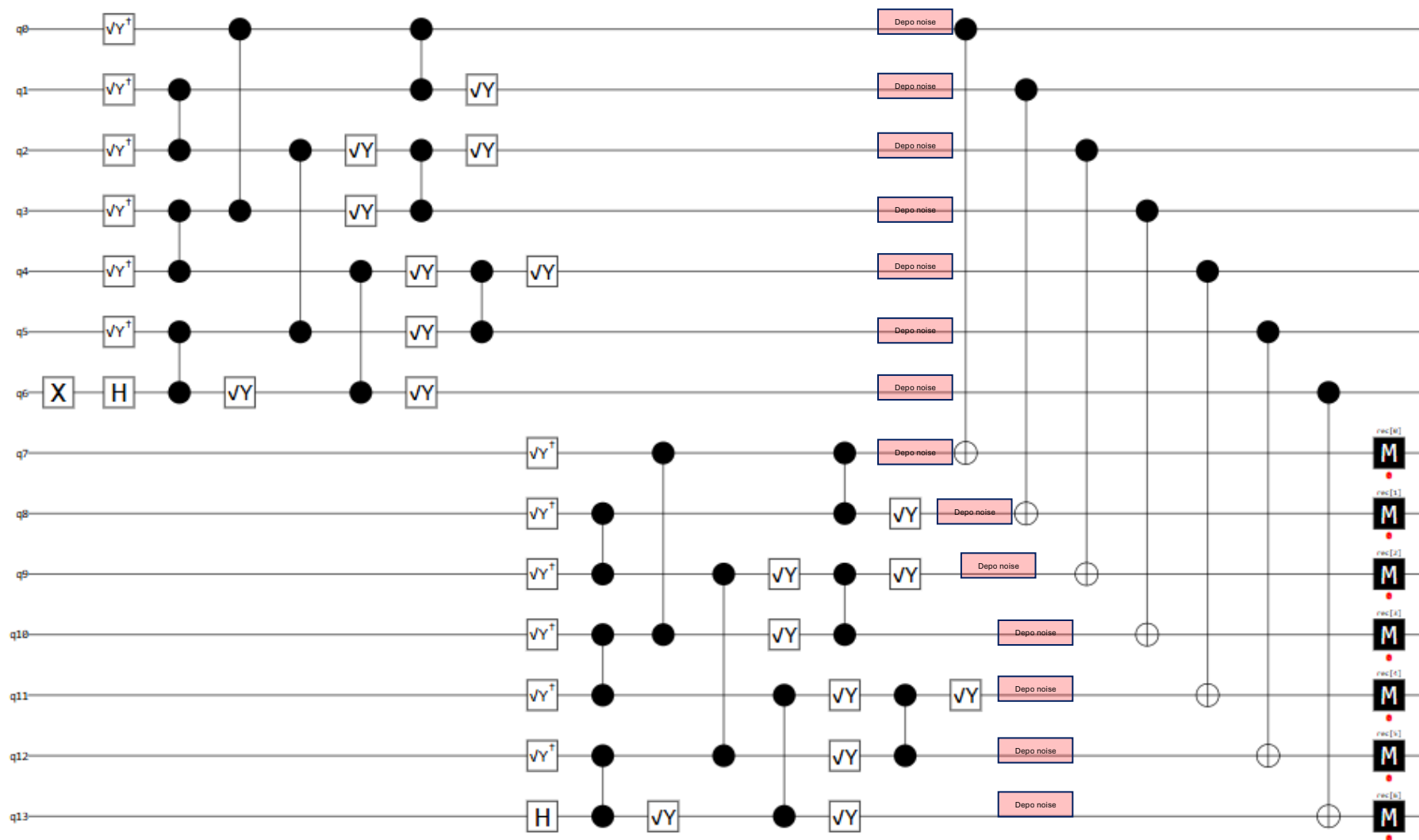
X post selection, $p=0.1$, location=0

```
def noisy_x_error_detection() -> IList[MeasurementResult, Any]:  
    >>> emulator = StackMemorySimulator(min_qubits=14)  
    >>> task = emulator.task(x_error_detection)  
    >>> results = task.run()  
    >>> print(results)  
    ""  
    location = 0      You, now + Uncommitted changes  
    p=0.1  
    qubits_1 = squin.qalloc(n_sean_qubits)  
    qubits_2 = squin.qalloc(n_sean_qubits)  
    if location == 0:  
        for i in range(len(qubits_1)):  
            squin.depolarize(p,qubits_1[i])  
            squin.depolarize(p,qubits_2[i])  
        # put arbitrary state  
        squin.x(qubits_1[-1])  
        squin.h(qubits_1[-1])  
        qubits_1 = color_code_factory(qubits_1)  
  
        # put state + on ancillary registers  
        squin.h(qubits_2[-1])  
        qubits_2 = color_code_factory(qubits_2)  
        if location == 1:  
            for i in range(len(qubits_1)):  
                squin.depolarize(p,qubits_1[i])  
                squin.depolarize(p,qubits_2[i])  
            # transversal CX  
            for index in range(len(qubits_1)):  
                squin.cx(qubits_1[index], qubits_2[index])  
        if location == 2:  
            for i in range(len(qubits_1)):  
                squin.depolarize(p,qubits_1[i])  
                squin.depolarize(p,qubits_2[i])  
            bits = squin.broadcast.measure(qubits_2)  
            return bits
```

~70% success rate with depolarizing noise on first location



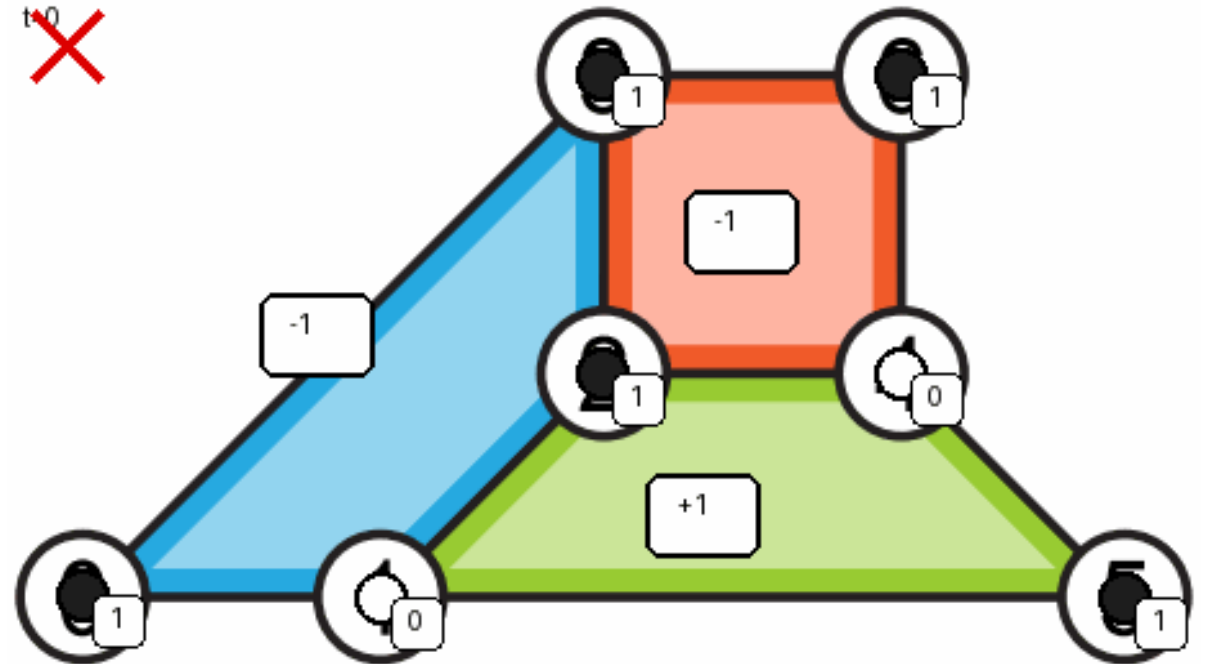
Lets add some noise, location = 1



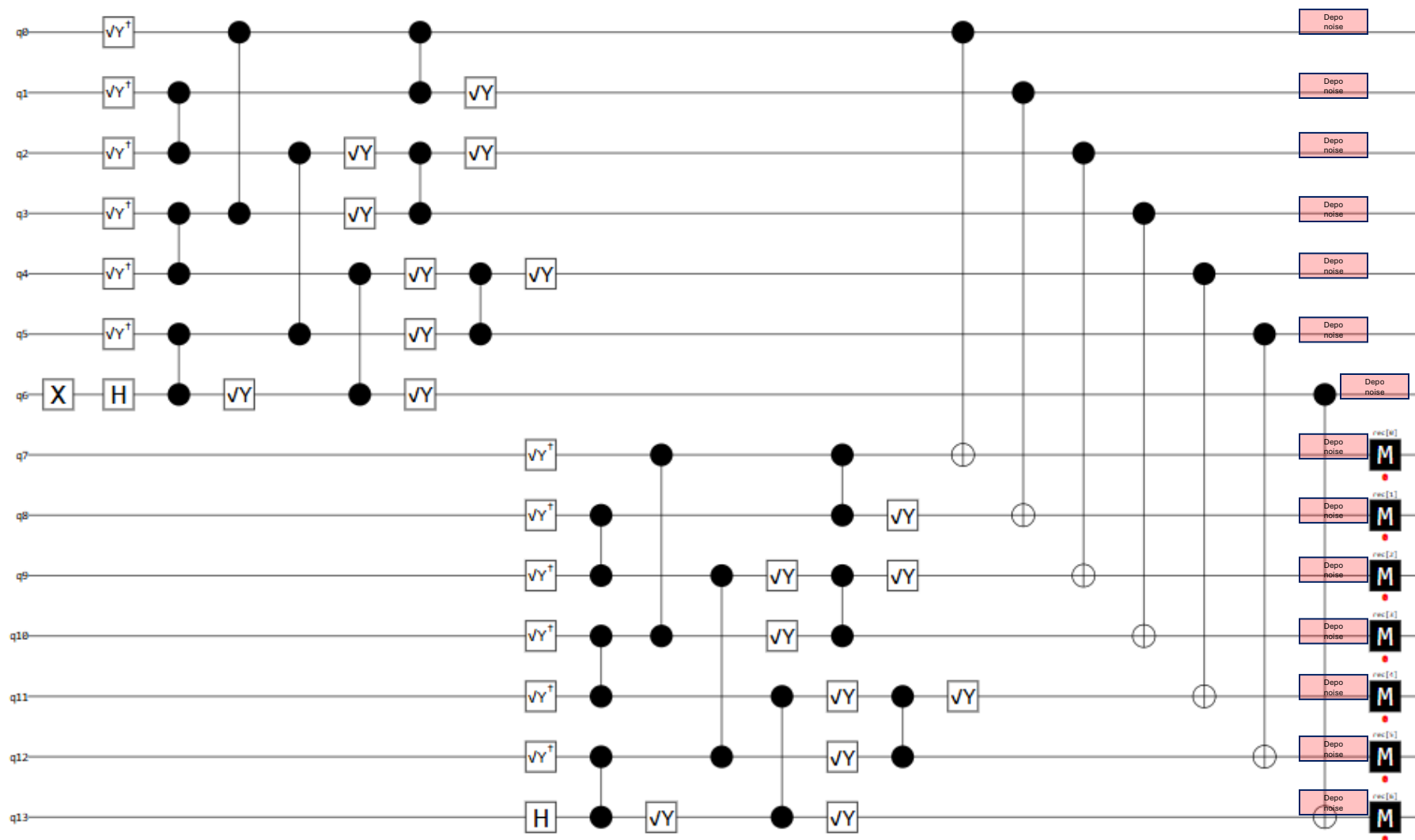
X post selection, $p=0.1$, location=1

```
def noisy_x_error_detection() -> IList[MeasurementResult, Any]:  
    >>> emulator = StackMemorySimulator(min_qubits=14)  
    >>> task = emulator.task(x_error_detection)  
    >>> results = task.run()  
    >>> print(results)  
    """"  
    location = 1      You, now • Uncommitted changes  
    p=0.1  
    qubits_1 = squin.qalloc(n_sean_qubits)  
    qubits_2 = squin.qalloc(n_sean_qubits)  
    if location == 0:  
        for i in range(len(qubits_1)):  
            squin.depolarize(p,qubits_1[i])  
            squin.depolarize(p,qubits_2[i])  
    # put arbitrary state  
    squin.x(qubits_1[-1])  
    squin.h(qubits_1[-1])  
    qubits_1 = color_code_factory(qubits_1)  
  
    # put state + on ancillary registers  
    squin.h(qubits_2[-1])  
    qubits_2 = color_code_factory(qubits_2)  
    if location == 1:  
        for i in range(len(qubits_1)):  
            squin.depolarize(p,qubits_1[i])  
            squin.depolarize(p,qubits_2[i])  
    # transversal CX  
    for index in range(len(qubits_1)):  
        squin.cx(qubits_1[index], qubits_2[index])  
    if location == 2:  
        for i in range(len(qubits_1)):  
            squin.depolarize(p,qubits_1[i])  
            squin.depolarize(p,qubits_2[i])  
    bits = squin.broadcast.measure(qubits_2)  
    return bits
```

~40% success rate with depolarizing noise on first location



Lets add some noise, location = 2

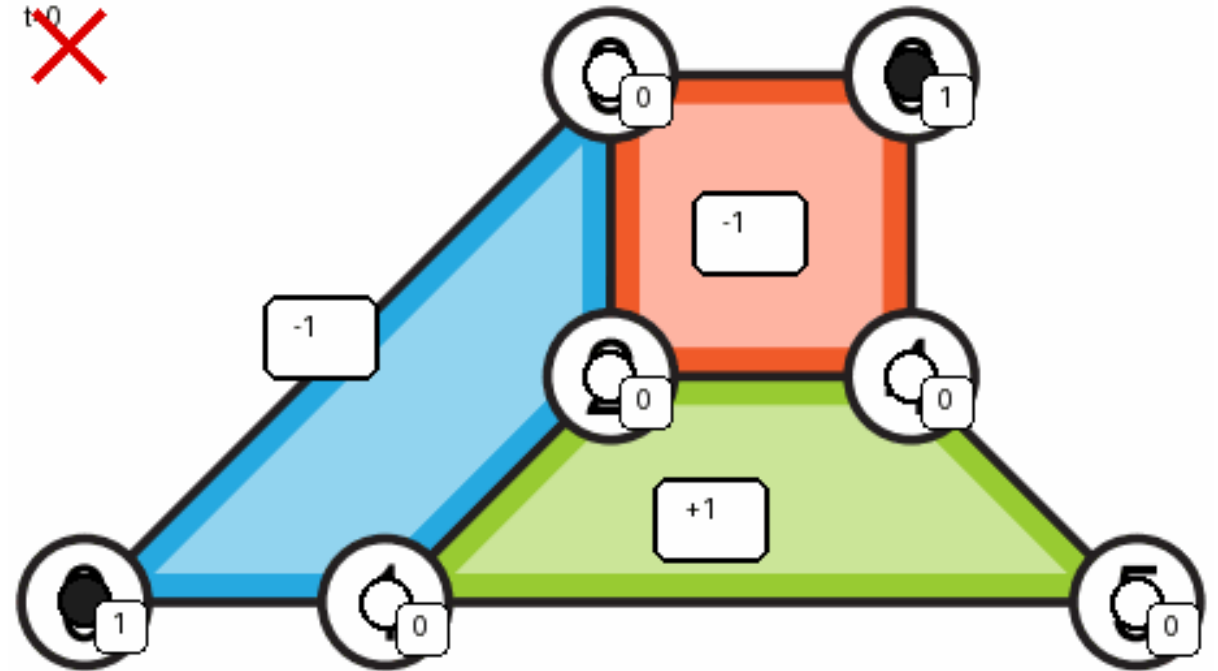


X post selection, $p=0.1$, location=2

```
def noisy_x_error_detection() -> IList[MeasurementResult, Any]:  
    >>> emulator = StackMemorySimulator(min_qubits=14)  
    >>> task = emulator.task(x_error_detection)  
    >>> results = task.run()  
    >>> print(results)  
    ""  
    location = 2      You, now * Uncommitted changes  
    p=0.1  
    qubits_1 = squin.qalloc(n_sean_qubits)  
    qubits_2 = squin.qalloc(n_sean_qubits)  
    if location == 0:  
        for i in range(len(qubits_1)):  
            squin.depolarize(p,qubits_1[i])  
            squin.depolarize(p,qubits_2[i])  
        # put arbitrary state  
        squin.x(qubits_1[-1])  
        squin.h(qubits_1[-1])  
        qubits_1 = color_code_factory(qubits_1)  
  
        # put state + on ancillary registers  
        squin.h(qubits_2[-1])  
        qubits_2 = color_code_factory(qubits_2)  
    if location == 1:  
        for i in range(len(qubits_1)):  
            squin.depolarize(p,qubits_1[i])  
            squin.depolarize(p,qubits_2[i])  
        # transversal CX  
        for index in range(len(qubits_1)):  
            squin.cx(qubits_1[index], qubits_2[index])  
    if location == 2:  
        for i in range(len(qubits_1)):  
            squin.depolarize(p,qubits_1[i])  
            squin.depolarize(p,qubits_2[i])  
        bits = squin.broadcast.measure(qubits_2)  
    return bits
```

~60% success rate with depolarizing noise on first location

~~t=0~~



Why did we choose this model for this part

- Allow to rapidly see where noise affect the results the most
- “Appropriately describe average noise for large circuits” (see <https://arxiv.org/pdf/2103.08591>)
- Simple and comprehensible; lots errors happen at the same time on real quantum systems
- Overall well studied and abstract specifics

$$\Delta_{\lambda}(\rho) = (1 - \lambda)\rho + \frac{\lambda}{d}(\text{tr}(\rho))I = (1 - \lambda)\rho + \frac{\lambda}{d}I$$

Now lets look at almost the whole thing

```
qubits_1 = squin.qalloc(n_sean_qubits)
# put arbitrary state
squid.x(qubits_1[-1])
squid.h(qubits_1[-1])
qubits_1 = color_code_factory(qubits_1)

# put state + on ancillary registers
qubits_2 = squin.qalloc(n_sean_qubits)
squid.h(qubits_2[-1])
qubits_2 = color_code_factory(qubits_2)

# transversal CX
for index in range(len(qubits_1)):
    squid.cx(qubits_1[index], qubits_2[index])
```

```
bits_x = squin.broadcast.measure(qubits_2)

qubits_3 = squin.qalloc(n_sean_qubits)

qubits_3 = color_code_factory(qubits_3)

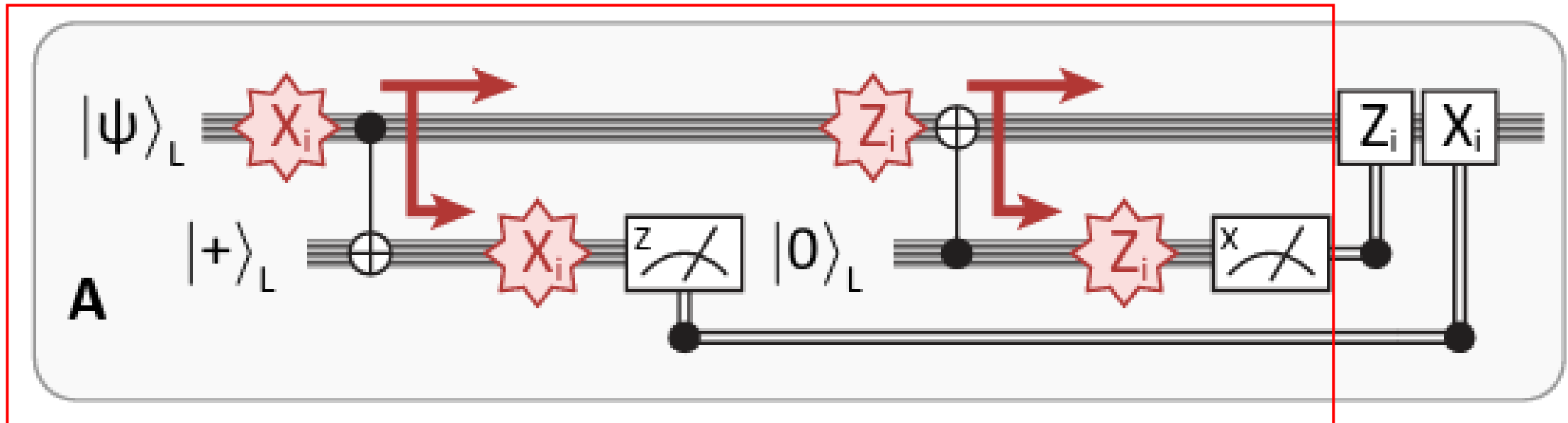
squin.z(qubits_3[0])

# transversal CX
for index in range(len(qubits_1)):
    squin.cx(qubits_3[index], qubits_1[index])
```

```
# Apply Hadamard gates for X-basis measurement
squid.broadcast.h(qubits_3)

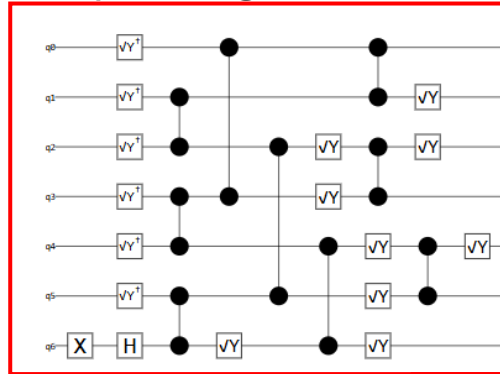
bits_z = squid.broadcast.measure(qubits_3)
bits_psi = squid.broadcast.measure(IList([qubits_1[0], qubits_2[0], qubits_3[0]]))

return (bits_x, bits_z, bits_psi)
```

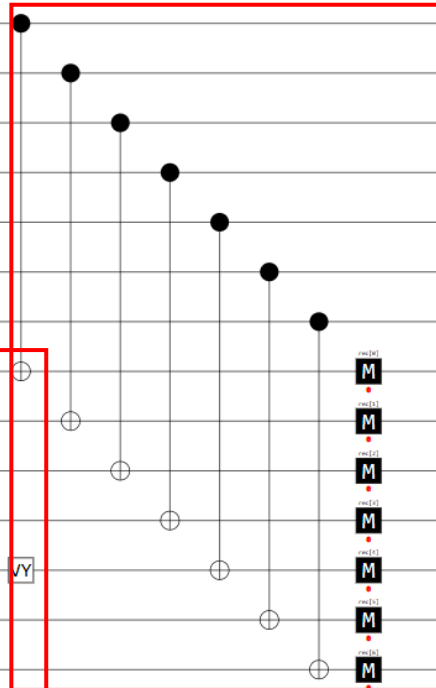


First iteration

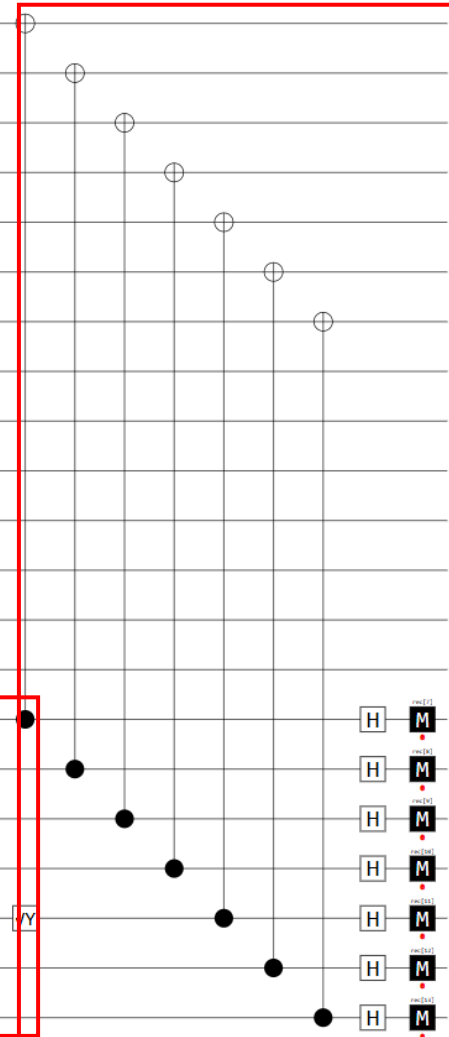
Prepare logical state



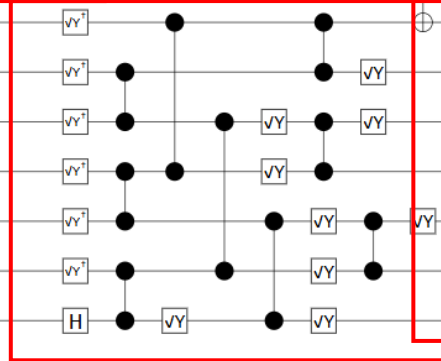
Transversal cnot for x error and midcircuit measurement



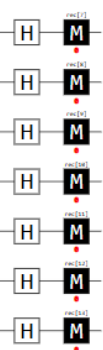
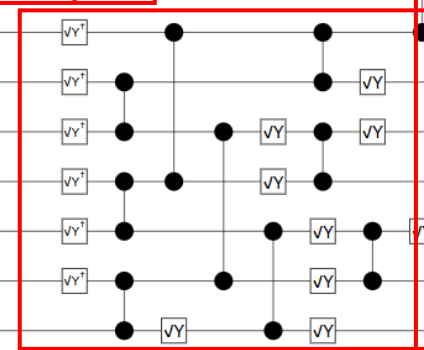
Reversed transversal cnot for z error detection and midcircuit measurement



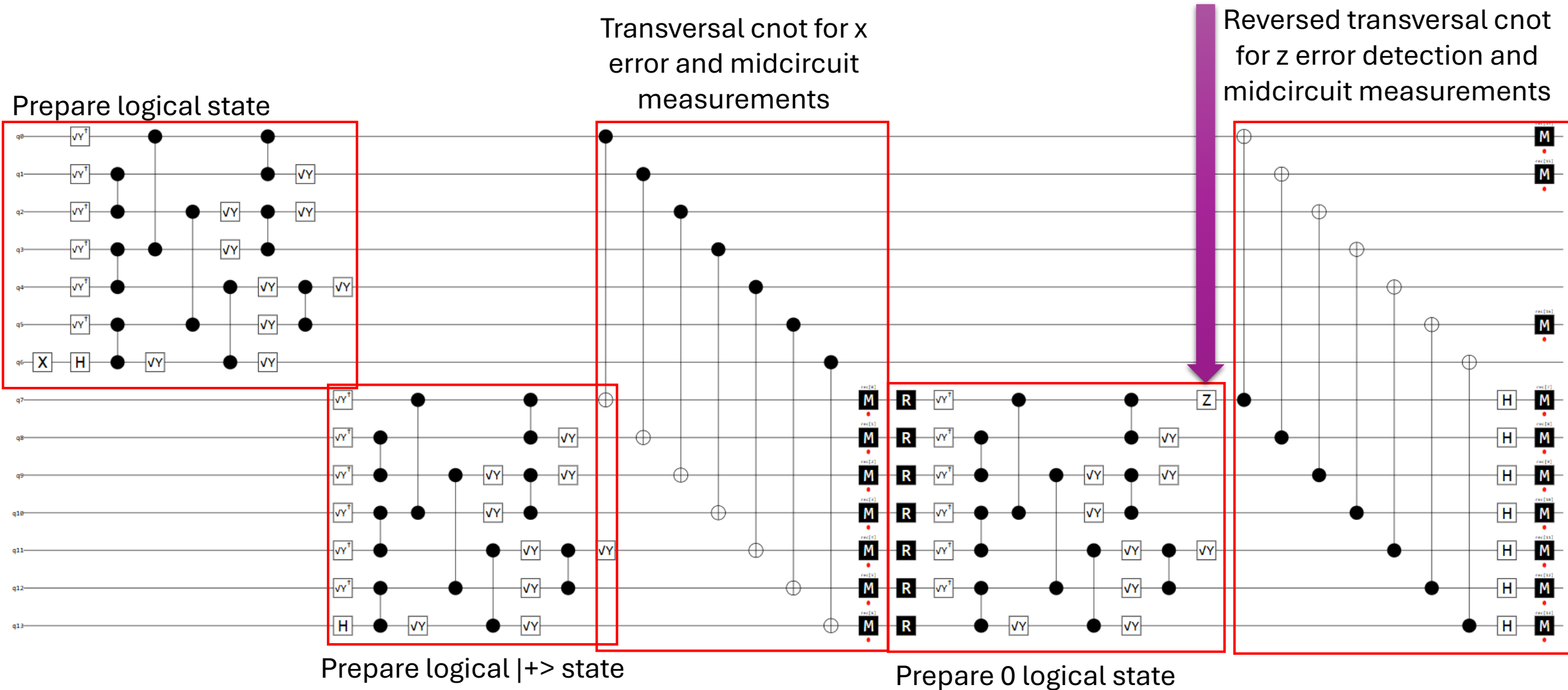
Prepare logical $|+\rangle$ state



Prepare 0 logical state



Then figured we could reuse the qubits...



Mid circuit measurement and error correction

With depolarizing on 1 channel on q0

```
valid : 331, invalid : 669
```

With no noise (implementation works!)

```
valid : 1000, invalid : 0
```

```
# transversal CX
for index in range(len(qubits_1)):
    squin.cx(qubits_2[index], qubits_1[index])

# Apply Hadamard gates for X-basis measurement
squin.broadcast.h(qubits_2)

bits_z = squin.broadcast.measure(qubits_2)

~
blue = bits_x[0] ^ bits_x[1] ^ bits_x[2] ^ bits_x[3]
red = bits_x[2] ^ bits_x[3] ^ bits_x[4] ^ bits_x[6]
green = bits_x[1] ^ bits_x[2] ^ bits_x[4] ^ bits_x[5]

if blue == 1 and red == 0 and green == 0:
    squin.x(qubits_1[0])
elif blue == 1 and red == 0 and green == 1:
    squin.x(qubits_1[1])
elif blue == 1 and red == 1 and green == 1:
    squin.x(qubits_1[2])
elif blue == 1 and red == 1 and green == 0:
    squin.x(qubits_1[3])
elif blue == 0 and red == 1 and green == 1:
    squin.x(qubits_1[4])
elif blue == 0 and red == 0 and green == 1:
    squin.x(qubits_1[5])
elif blue == 0 and red == 1 and green == 0:
    squin.x(qubits_1[6])

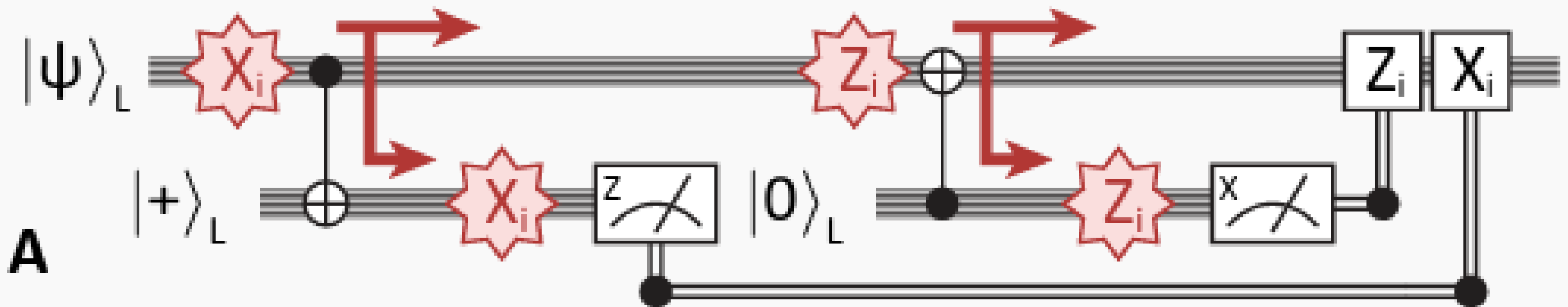
~
blue = bits_z[0] ^ bits_z[1] ^ bits_z[2] ^ bits_z[3]
red = bits_z[2] ^ bits_z[3] ^ bits_z[4] ^ bits_z[6]
green = bits_z[1] ^ bits_z[2] ^ bits_z[4] ^ bits_z[5]

if blue == 1 and red == 0 and green == 0:
    squin.z(qubits_1[0])
elif blue == 1 and red == 0 and green == 1:
    squin.z(qubits_1[1])
```

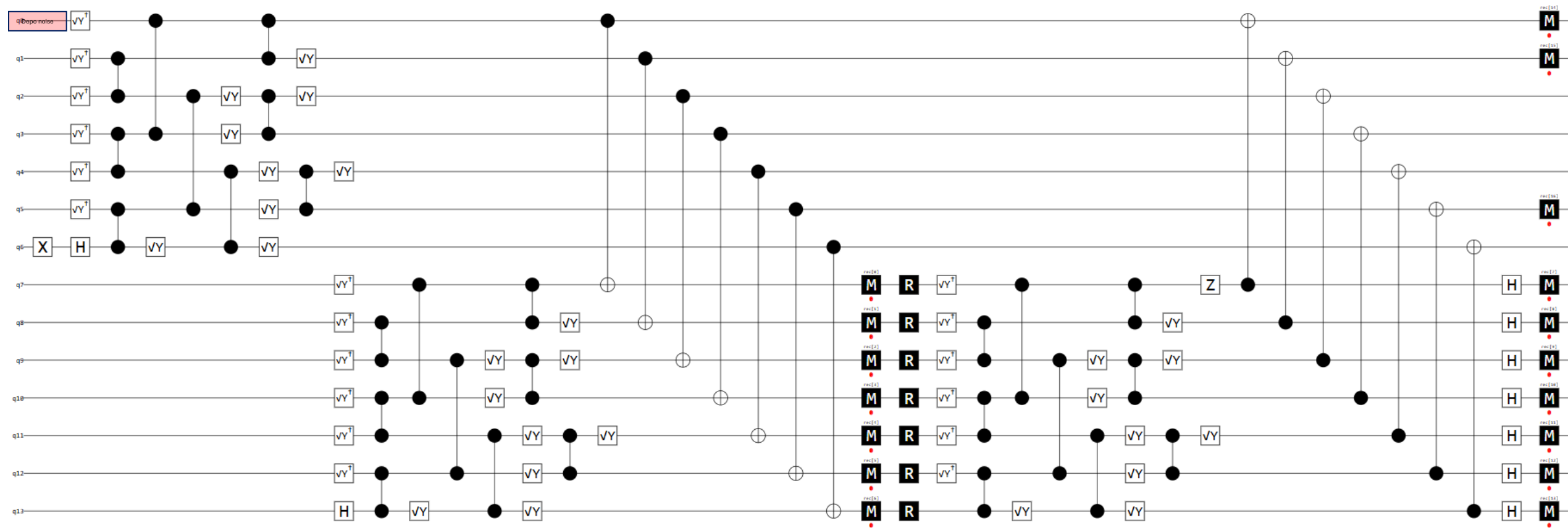
Final circuit*

Comment on noise property on this circuit, BONUS 2

*We do mid circuit measurement to figure out which gates to put and do put a Z_i or a X_i gate depending on the results of the mid circuit measurement. As to whether or not it improved the results is to be verified.



Final example



Next steps

- Figure out better heuristics to select the correct qubits to correct
- Try steane code with greater distance
- Experiment with magic state

