Polar codes decoding

EE 276 (Information Theory), Winter 2020-21

Stanford University

Acknowledgement

- Parts of this are based on:
  - Slides from Prof. Pilanci's lecture:
    - https://web.stanford.edu/class/ee276/files/polarcodes_EE276_2021_annotated.pdf
  - Tutorial by Erdal Arıkan himself:
    - https://simons.berkeley.edu/sites/default/files/docs/2689/slidesarikan.pdf
  - EE388 Modern Coding Course notes at Stanford
    - https://web.stanford.edu/class/ee388/HOMEWORK2018/lecture-9-10-11.pdf
    - Definitely recommended course if you want to learn all about the recent advances in coding theory including LDPC codes, polar codes, and several cool applications

Polar codes

- Polar codes involve a recursive circuit construction which forms an invertible map from the input bits ($U_1,\ldots,U_N$) to the channel input bits ($X_1,\ldots,X_N$)
- The actual channels from ($X_1,\ldots,X_N$) to channel outputs ($Y_1,\ldots,Y_N$) are independent of each other with fixed capacity, say C
- But the new "bit-channels" from U_i to (Y^N,U^{i-1}) (assuming previous bits already decoded) polarize as N grows larger
  - This means that for large N, NC channels have capacity 1 while N(1-C) channels have capacity 0
  - The specific values of i that give good/bad bit-channels depends on the original channel type and parameters (e.g., BEC, BSC, etc.)
- What does this achieve? Channels with capacity 0 or 1 are trivial to work with!
  - Capacity 0 means no information can be transmitted, so we simply freeze the bits
  - Capacity 1 means noiseless communication, so we can just send the message bits over these channels
- Thus, we can freeze N(1-C) bits and use NC bits for the actual message bits
  - Achieved rate is NC message bits/N channel transmission = C (in the limit)

Polar codes successive cancellation decoding

- As you saw in class, polar codes can be decoded using successive cancellation decoding
- Here we'll try to understand the procedure in slightly more detail
- We will focus on BEC for ease of explanation
  - For other channels, we work with log likelihood ratios instead of bits at the intermediate nodes, but the general principles are still the same
- This should hopefully help you with the homework question and inspire you to read more on polar code!
- It's important to understand that polar codes were the first deterministic and efficient schemes for achieving capacity – phenomena such as polarization also occur for random transforms but they're not efficient

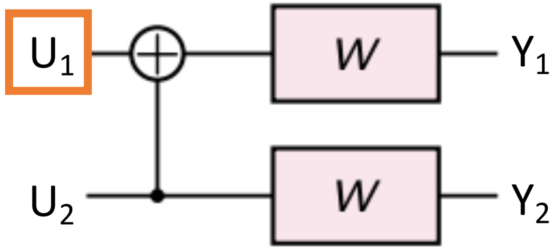
SC decoding: basic algorithm

- For input bits U^N (some of which are frozen) and channel output Y^N
  - We first decode U_1 based on Y^N assuming that $(U_2,\ldots,U_N)$ are random
  - Then we decode U_2 based on Y^N,U_1 assuming that $(U_3,\ldots,U_N)$ are random
  - …
  - Finally we decode U_N based on $Y^N,U_1,\ldots,U_{N-1}$
- At any stage if you get an erasure of U_i, you abort and declare failure
- Where do the frozen bits come in the picture? When you fail to decode a frozen bit U_i, you don't abort – instead, you set U_i to the known frozen value
  - We still go through the process of the decoding U_i even though it is frozen – this allows us to compute a bunch of intermediate values to be used later

SC decoding: suboptimality and complexity

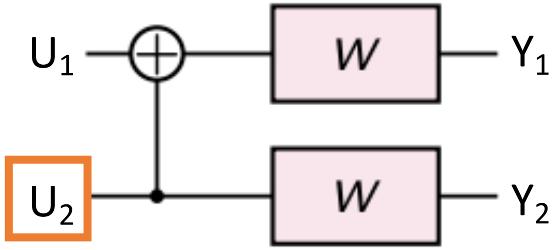
- Why assume (U_{i+1},…,U_N) are random when decoding U_i, even though of some these are frozen and hence known?
  - We might declare a failure at U_i which possibly could be decoded if we had used all the information about the upcoming frozen bits
  - SC decoding is not optimal
- SC decoding can be implemented efficiently, whereas we don't know if the optimal maximum likelihood decoding is efficient
  - SC decoding uses the recursive structure of polar codes: O(NlogN) complexity
  - And it still achieves capacity!
- For short block lengths, we can do better, e.g., using list decoding and CRCs (see work by Tal and Vardy at https://ieeexplore.ieee.org/document/7055304). These are one of the best codes at short block lengths and are part of 5G standards.
- Here we won't see how the NlogN complexity can be achieved
  - The naive algorithm is N^2 (still polynomial time)
  - Getting to NlogN uses the recursive structure, similar to how FFT works

2x2 decoding (W = BEC, ? = erasure)



Decoding U_1:

?,	if Y_1 or Y_2 is ?
$Y_1 \oplus Y_2$,	otherwise

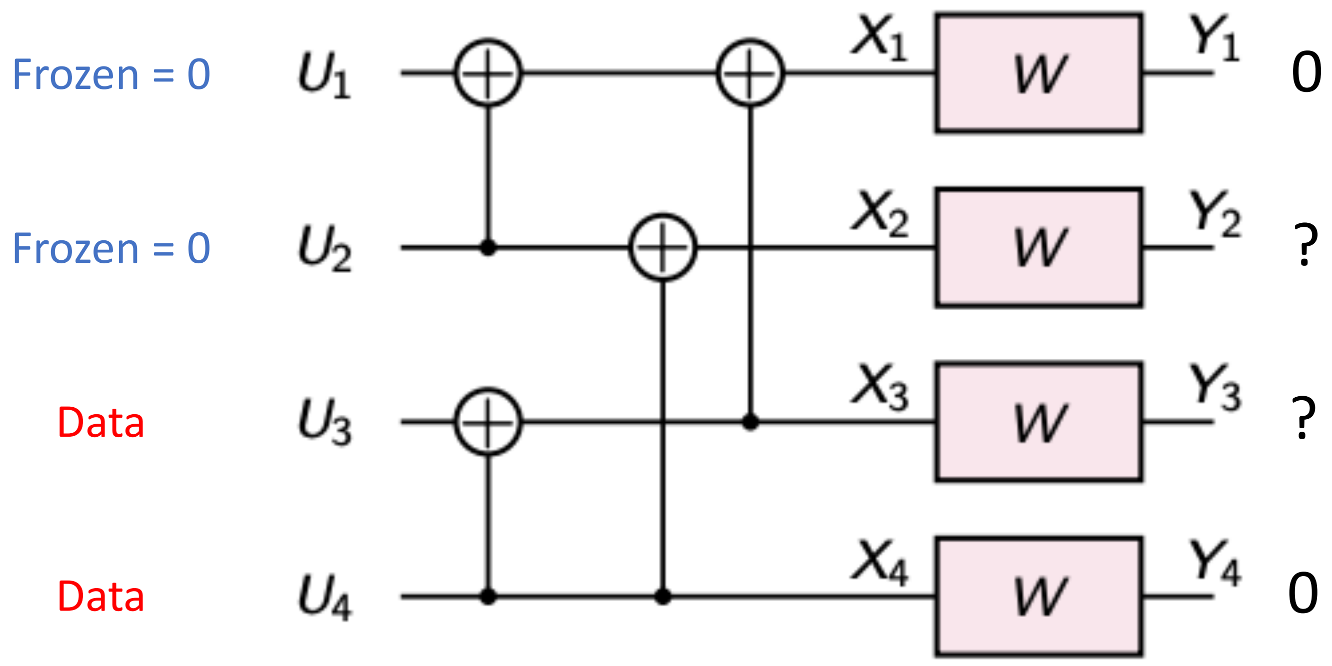


Decoding U_2:

?,	if Y_1 and Y_2 are ?
Y_2,	if Y_2 not erased
$Y_1 \oplus U_1$,	if Y_1 not erased

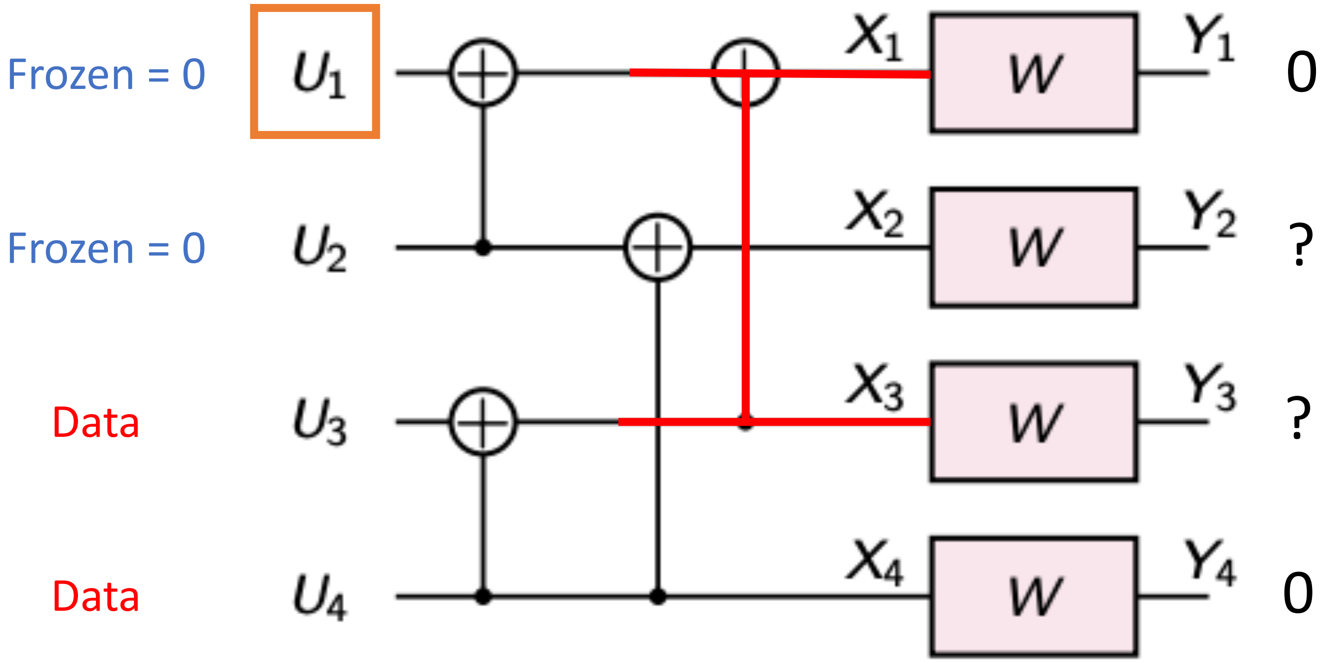
Decoding for larger N consists of multiple 2x2 decoding steps

4x4 example

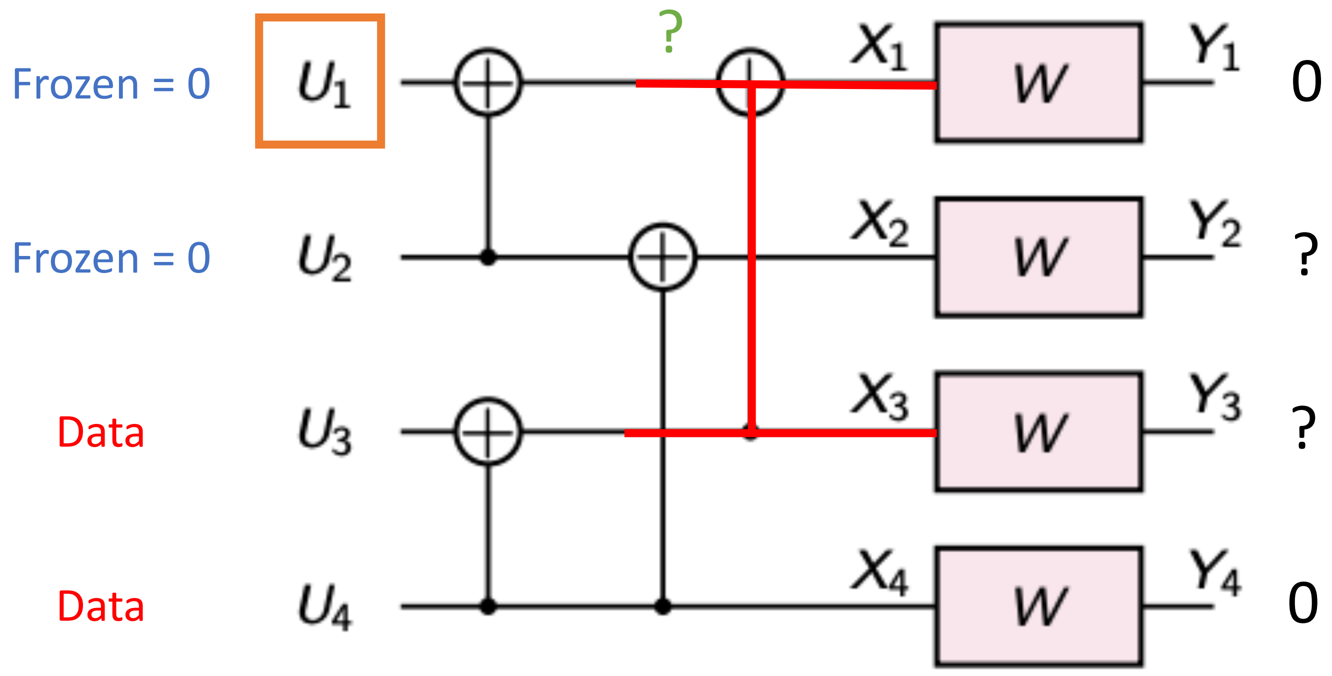


First, we attempt to decode U_1 by decoding the intermediate 2x2 blocks from right to left

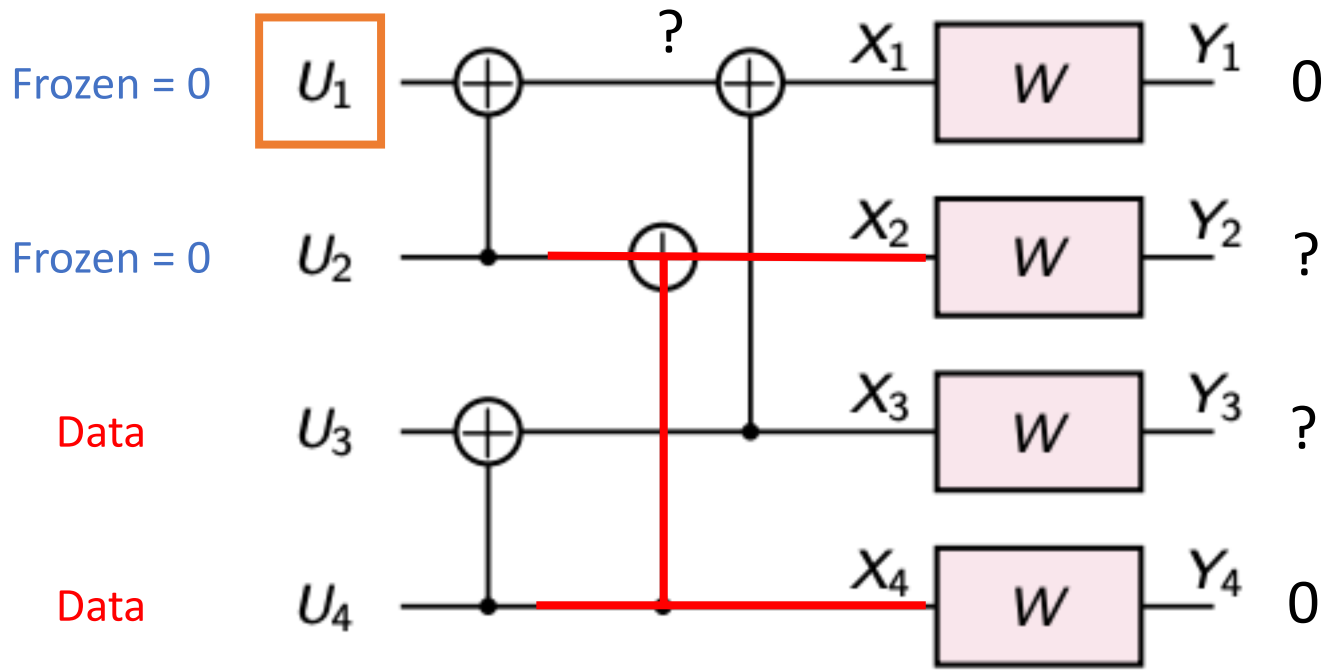
4x4 example: decoding U_1



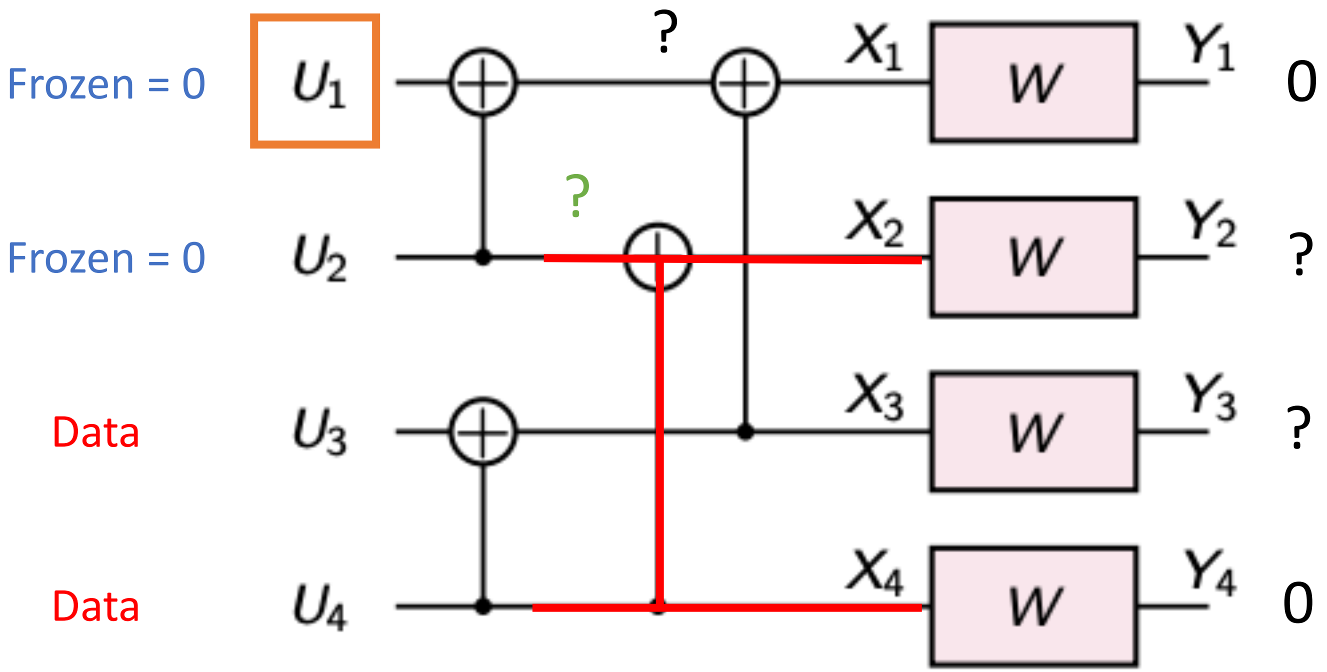
4x4 example: decoding U_1



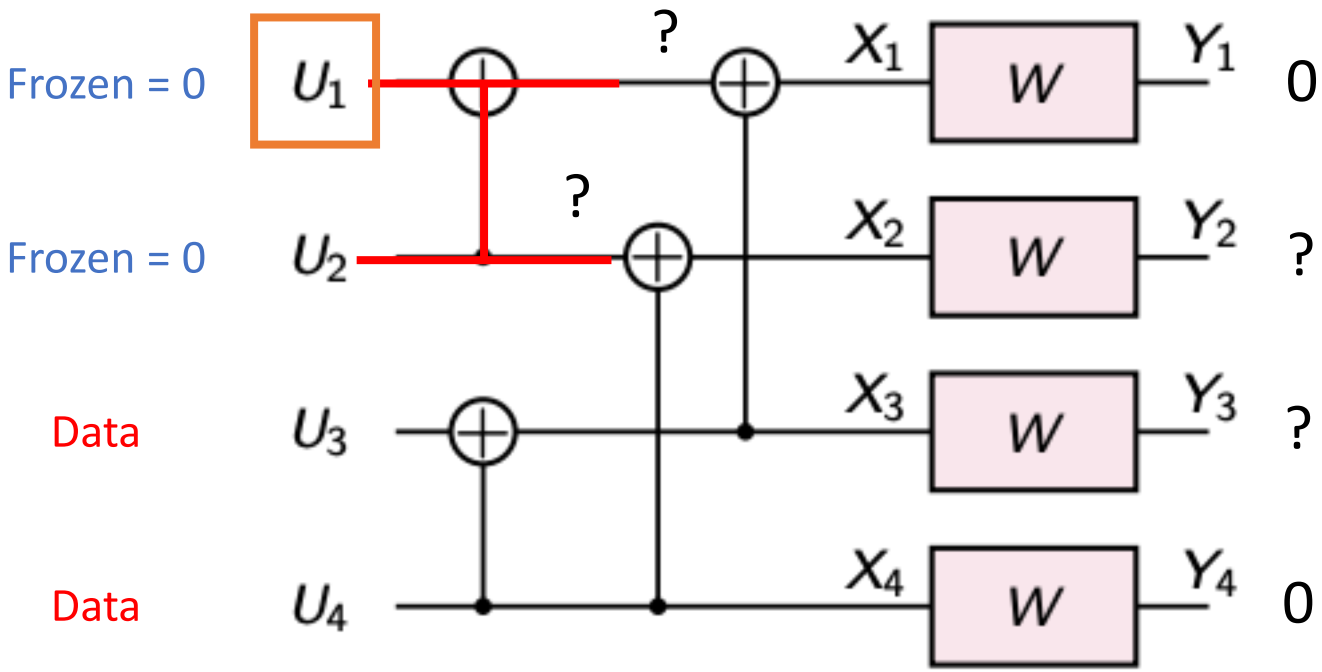
4x4 example: decoding U_1



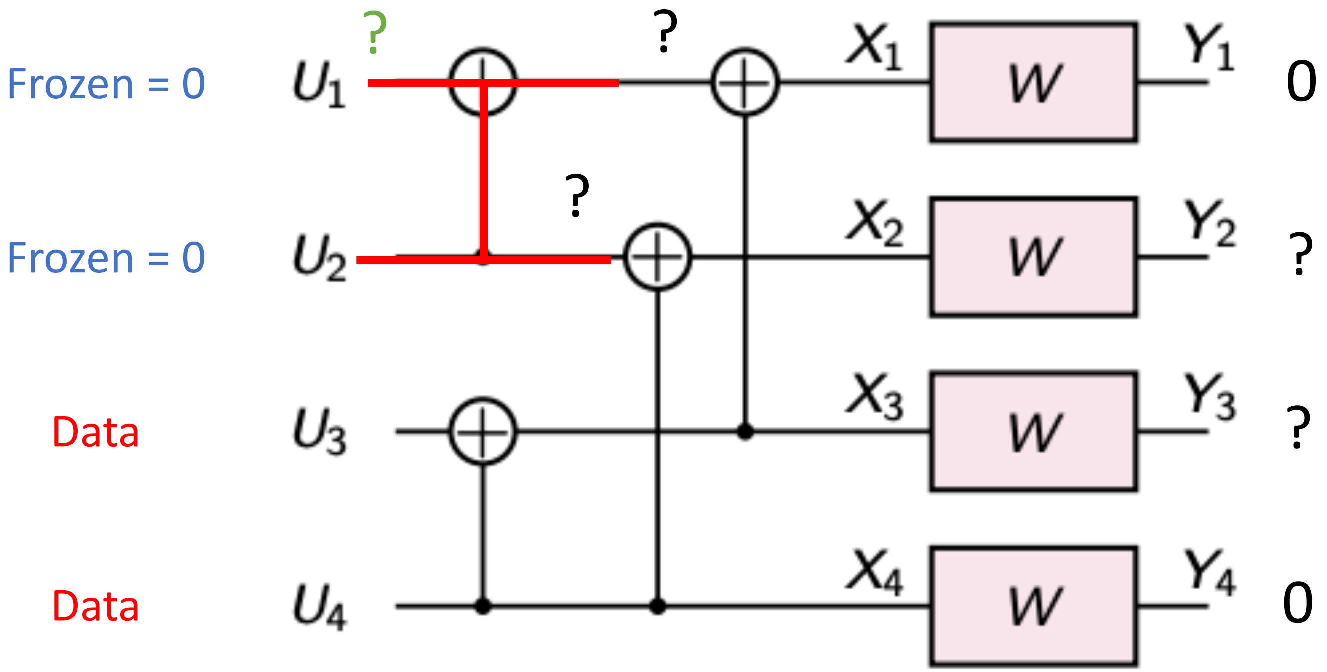
4x4 example: decoding U_1



4x4 example: decoding U_1

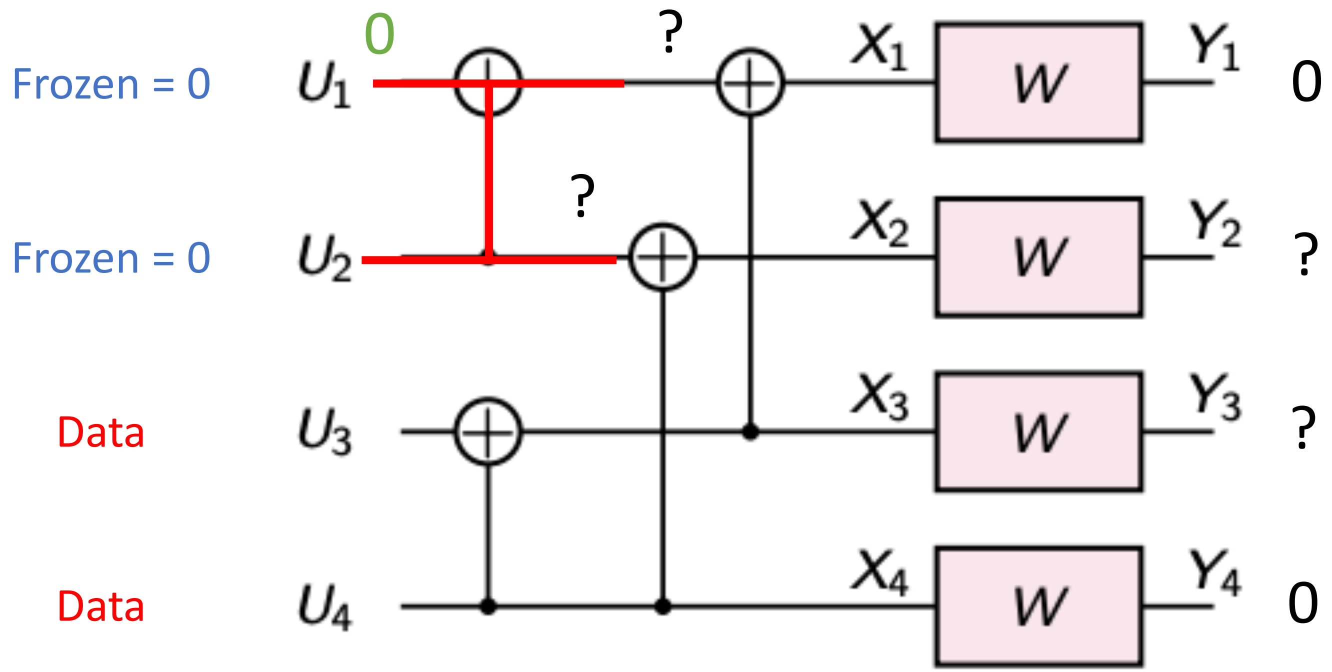


4x4 example: decoding U_1



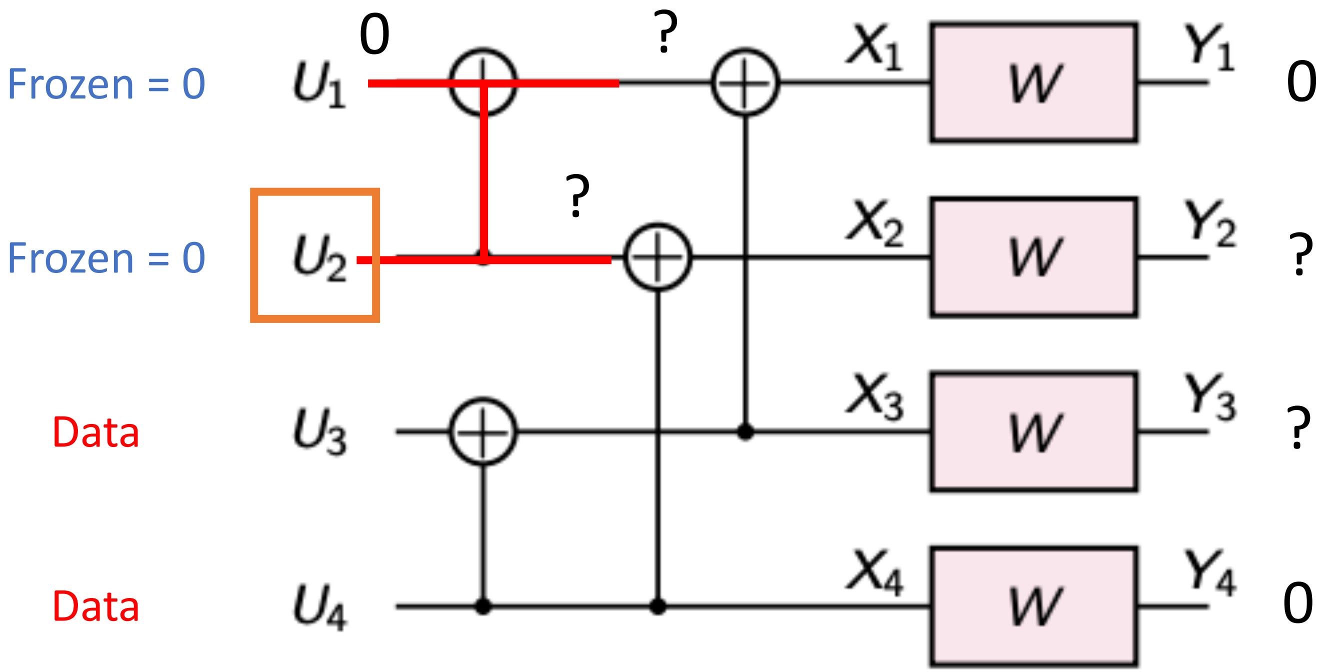
So we failed to decode U_1 ☹ - but it's frozen so we just set it to 0!

4x4 example: decoding U_1

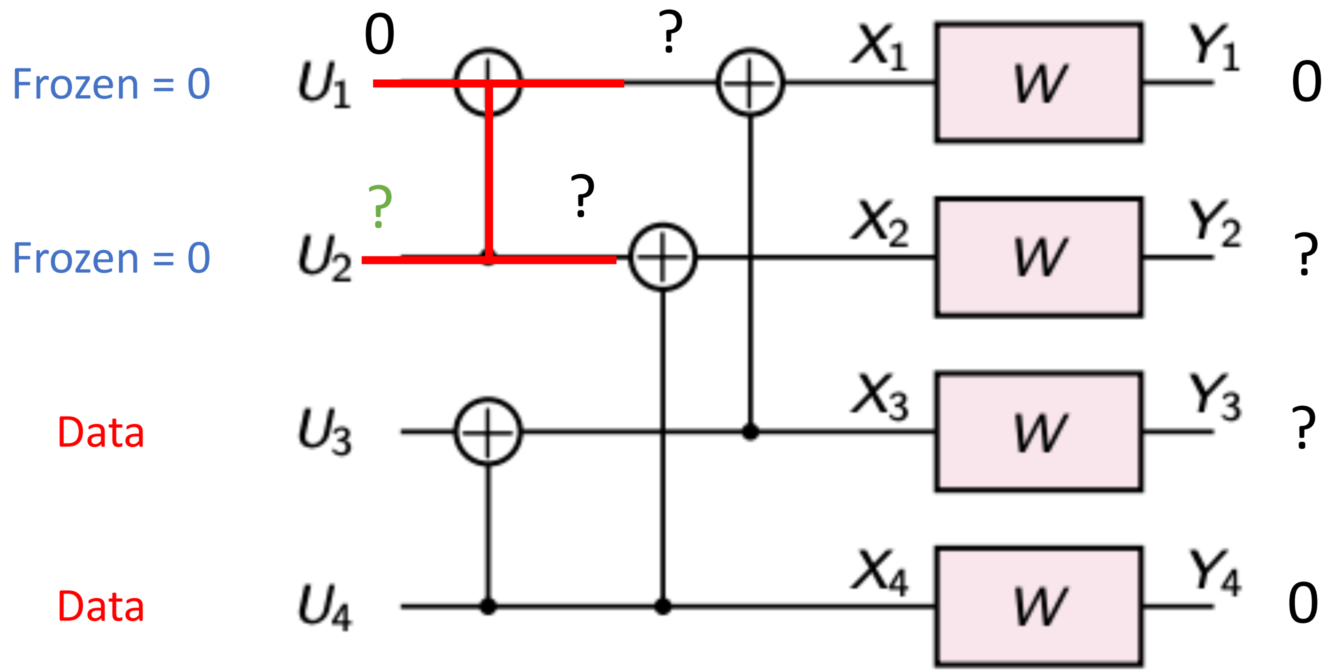


So we failed to decode U_1 ☹ - but it's frozen so we just set it to 0!

4x4 example: decoding U_2

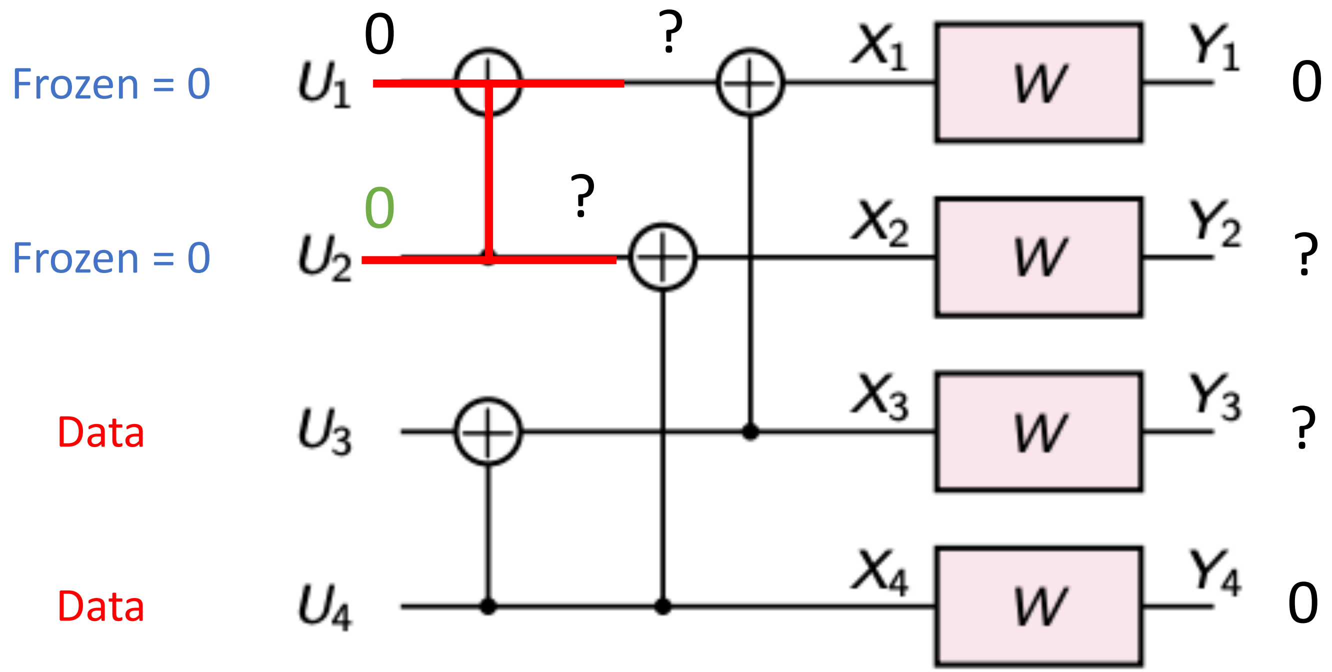


4x4 example: decoding U_2



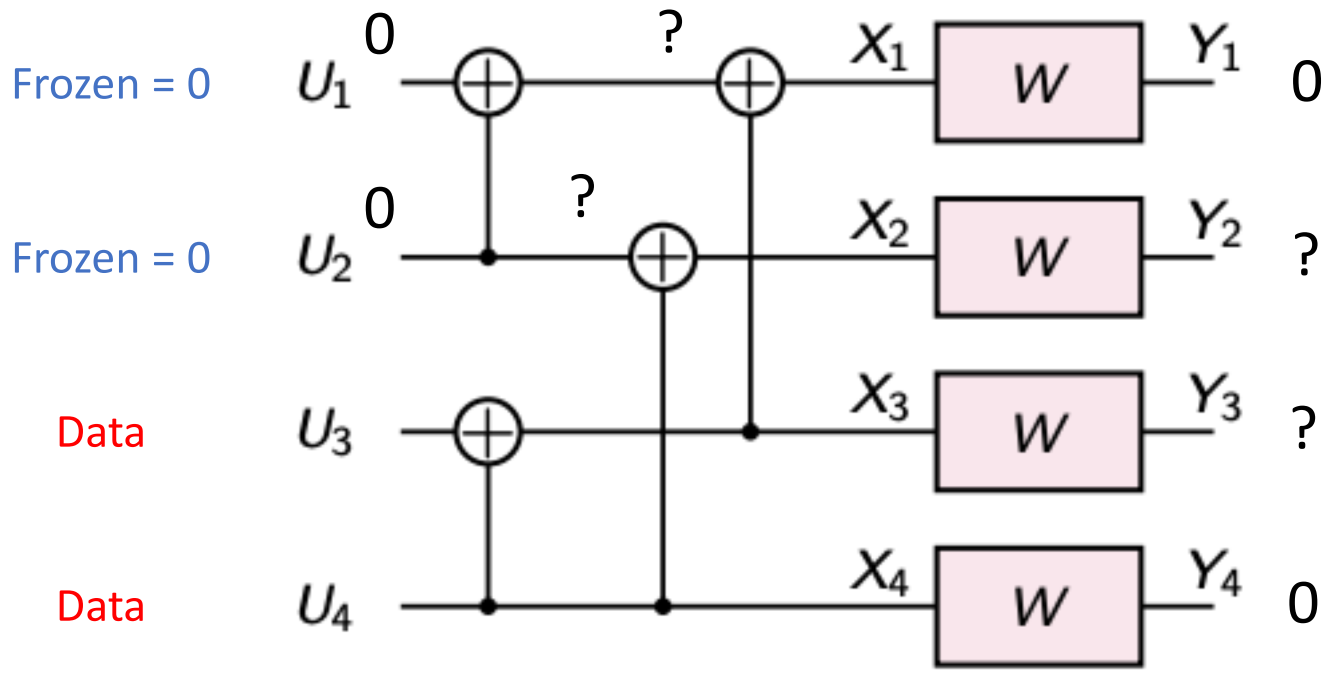
Even knowing U_1, we can't decode U_2 since we have two erasures.
But U_2 is also frozen, so we keep moving on.

4x4 example: decoding U_2



Even knowing U_1, we can't decode U_2 since we have two erasures.
But U_2 is also frozen, so we keep moving on.

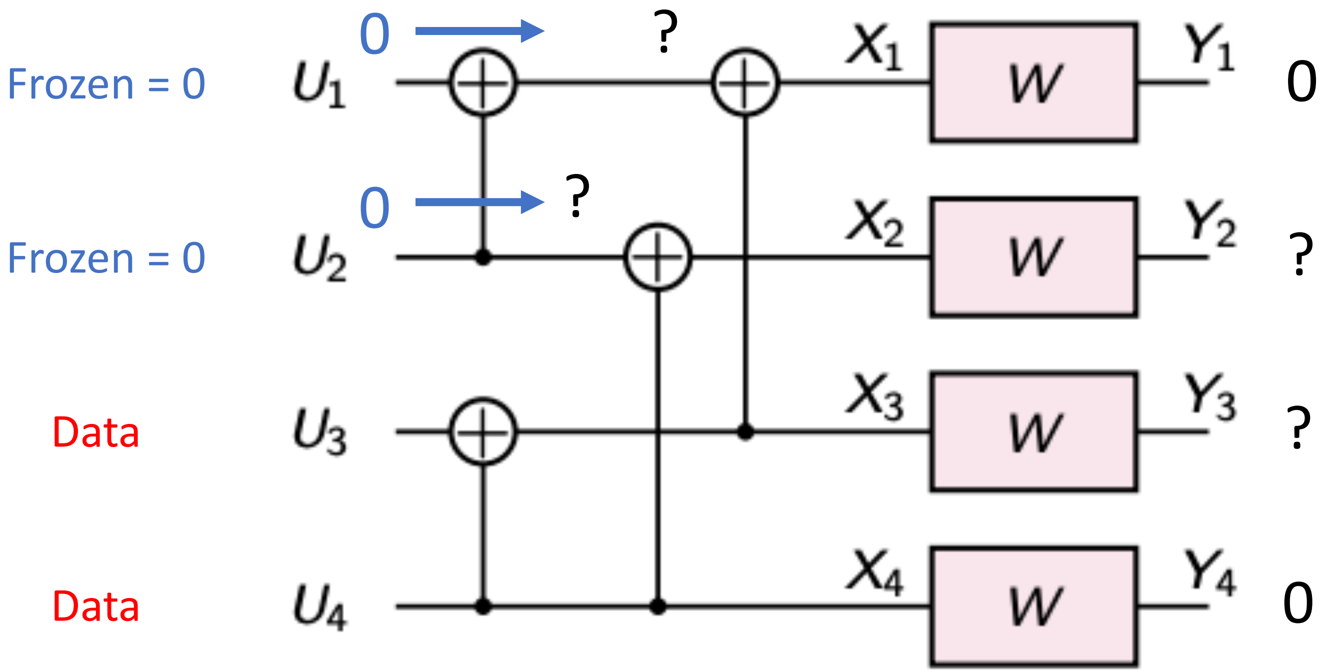
4x4 example: U_1 and U_2 decoded



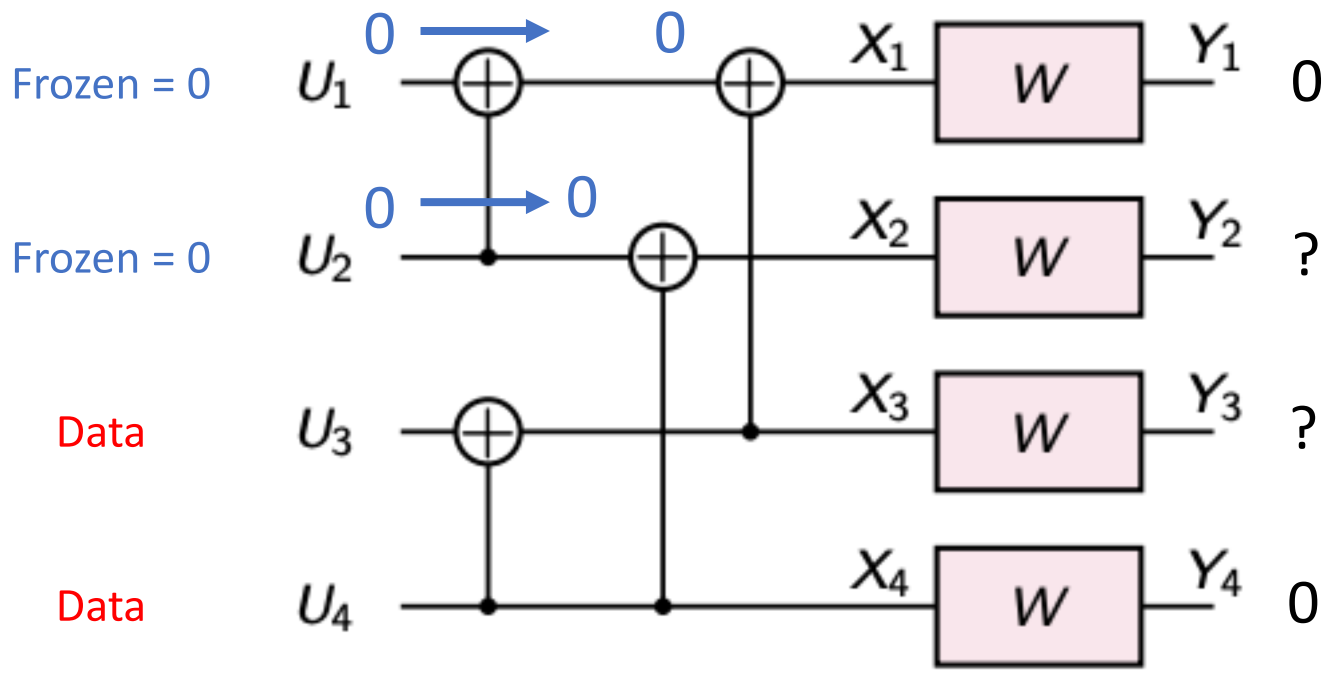
4x4 example

- Now that U_1 and U_2 are decoded, the upcoming bits will be decoded assuming these are known – so we update the rest of circuit using left-to-right circuit evaluations as shown next
- In general, you will have an alternation of
  - Decoding based on previously decoded bits
  - Update intermediate LLR (log-likelihood ratios)/bit values based on currently decoded bits

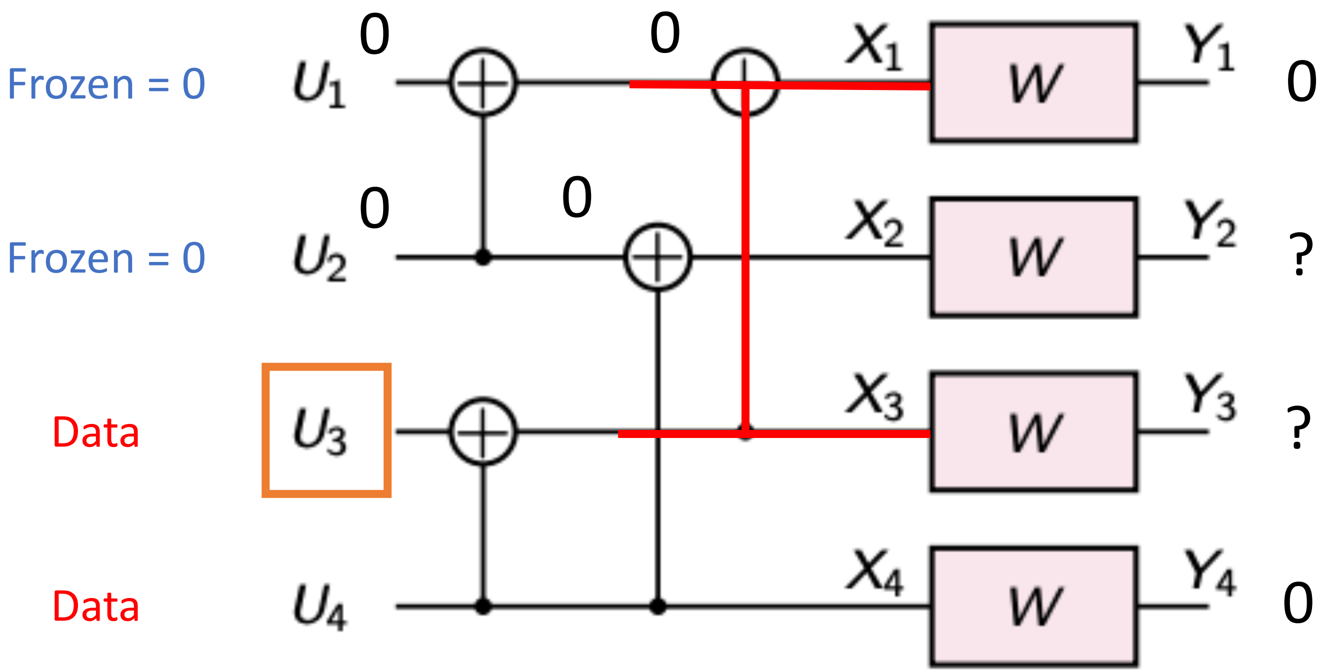
4x4 example: update intermediate bits



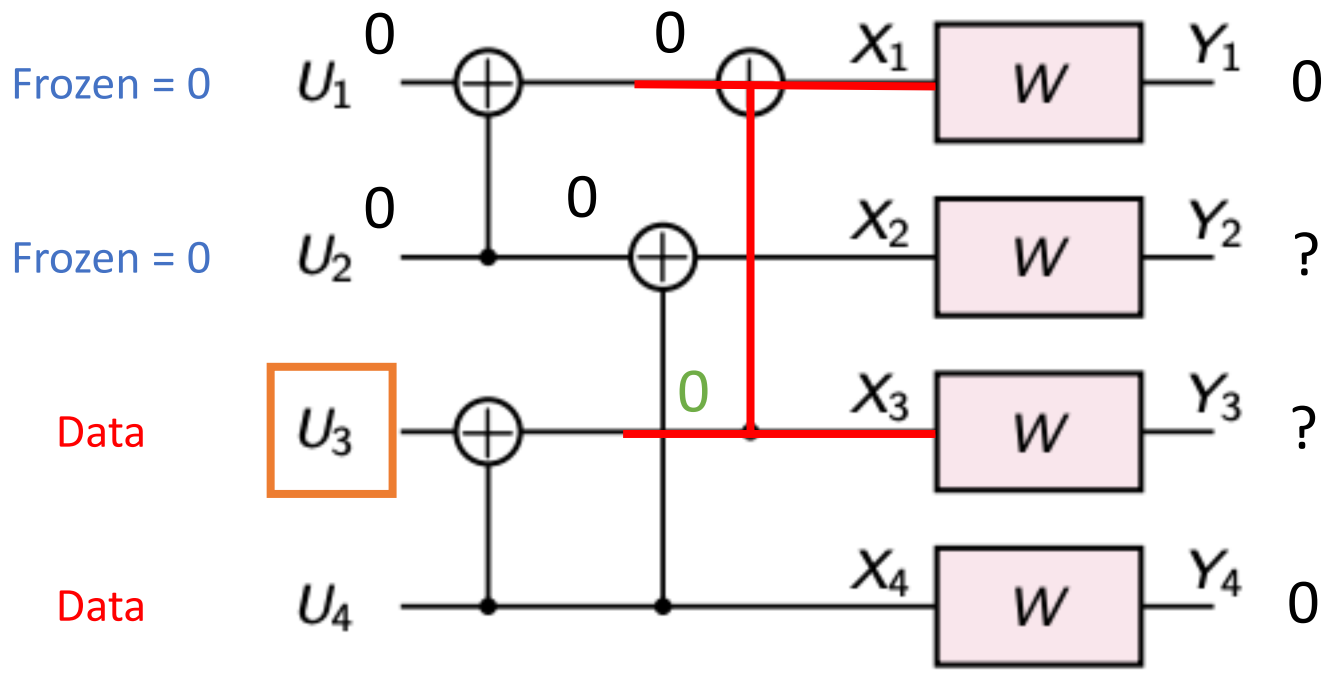
4x4 example: update intermediate bits



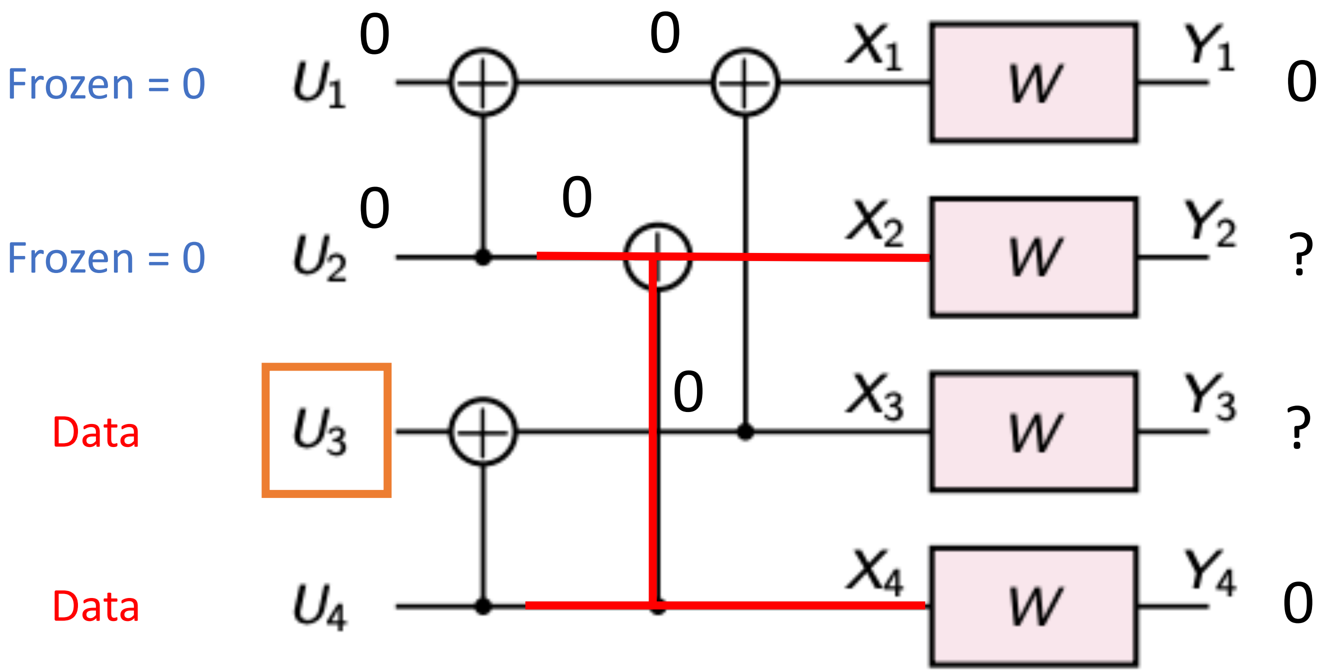
4x4 example: decoding U_3



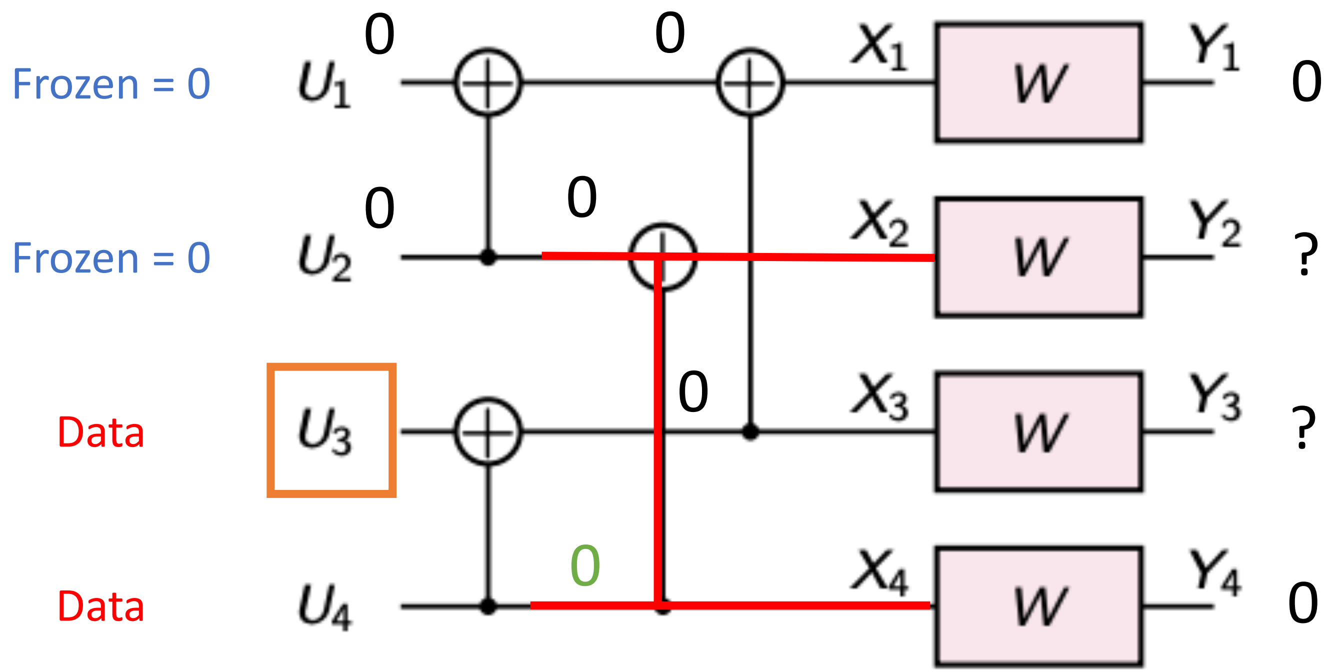
4x4 example: decoding U_3



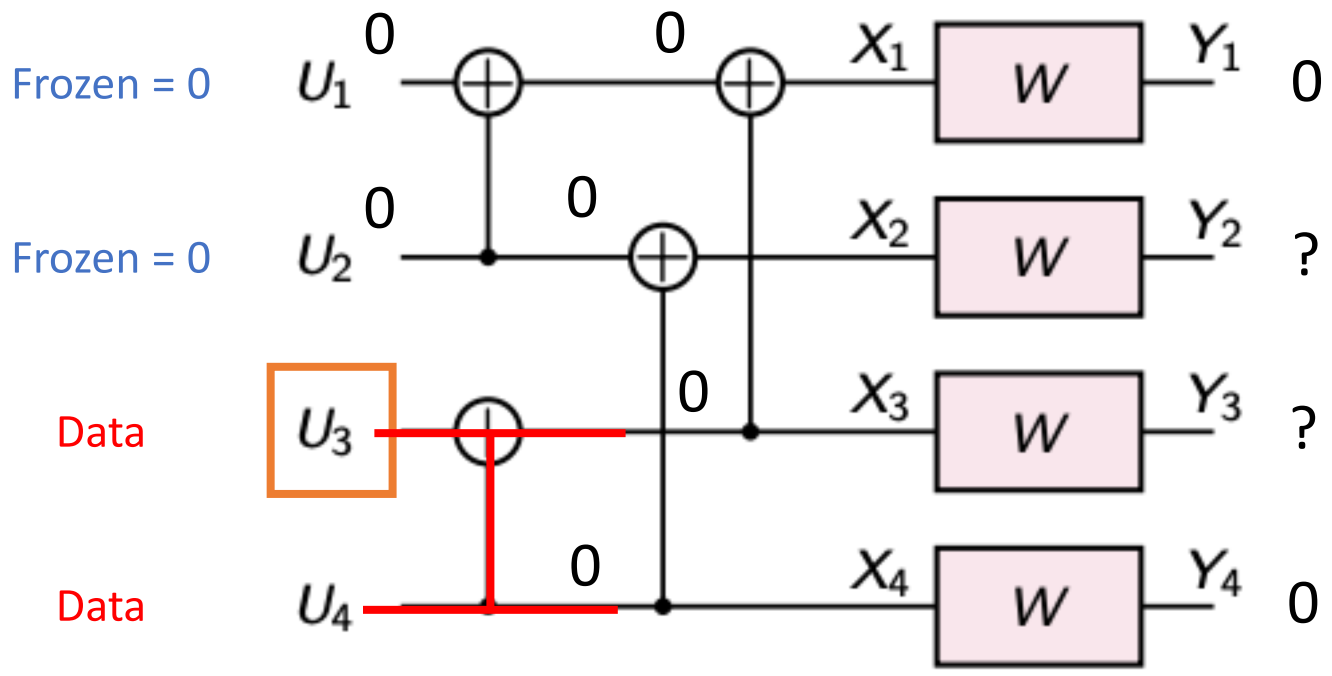
4x4 example: decoding U_3



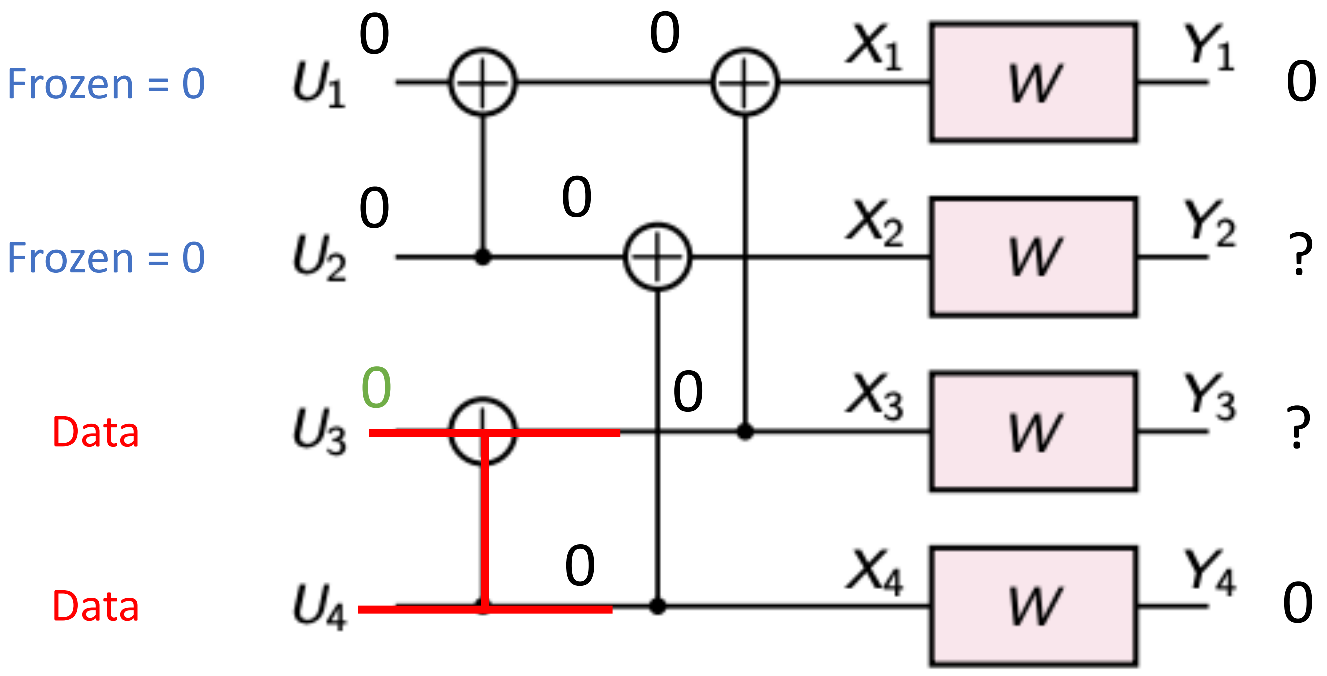
4x4 example: decoding U_3



4x4 example: decoding U_3

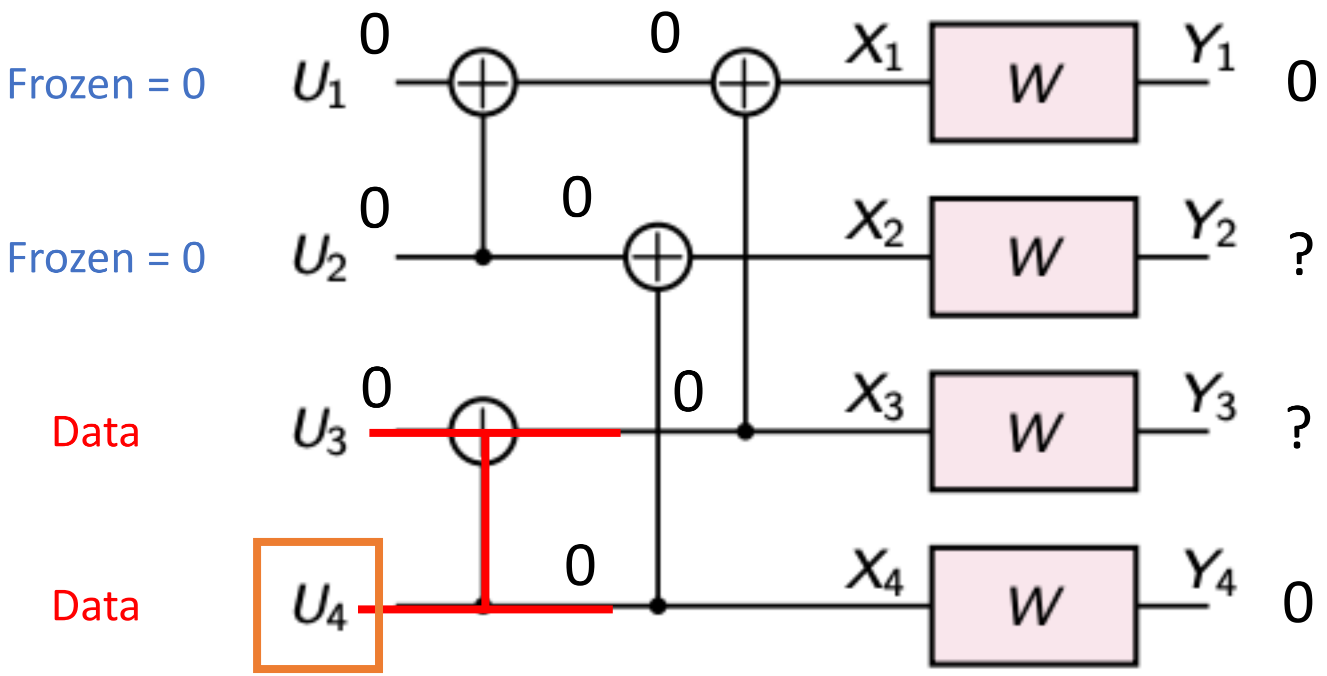


4x4 example: decoding U_3

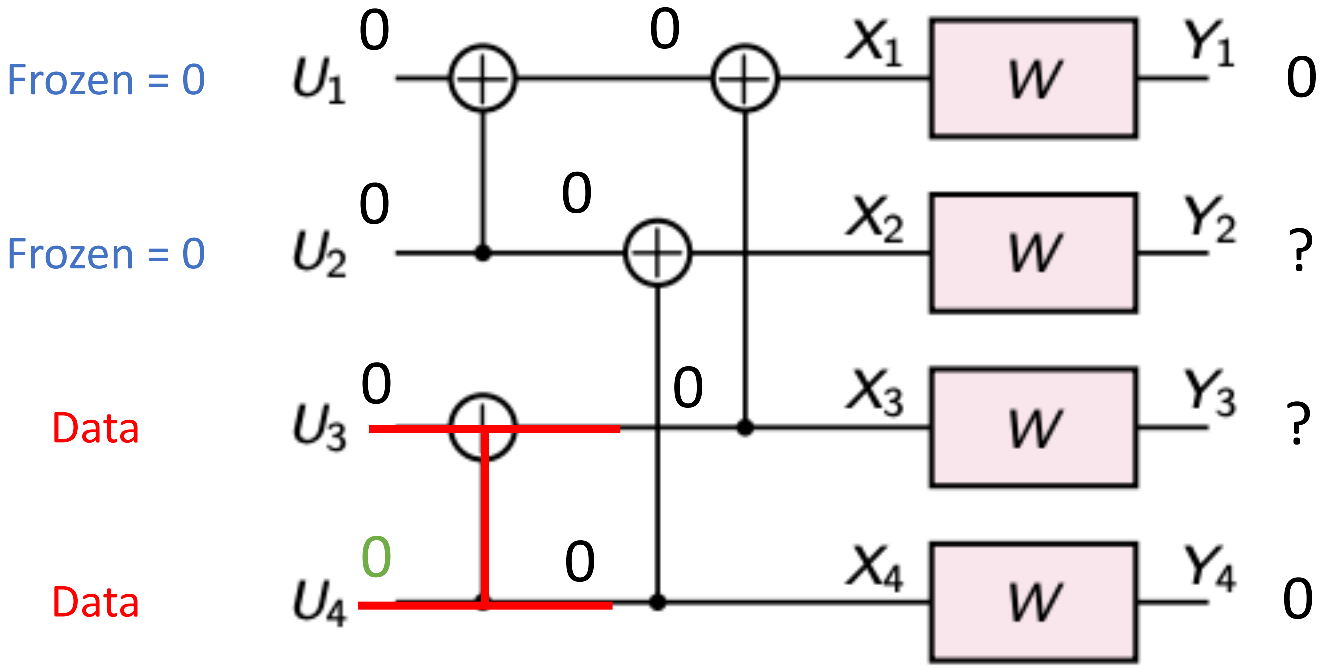


U_3 decoded successfully!

4x4 example: decoding U_4

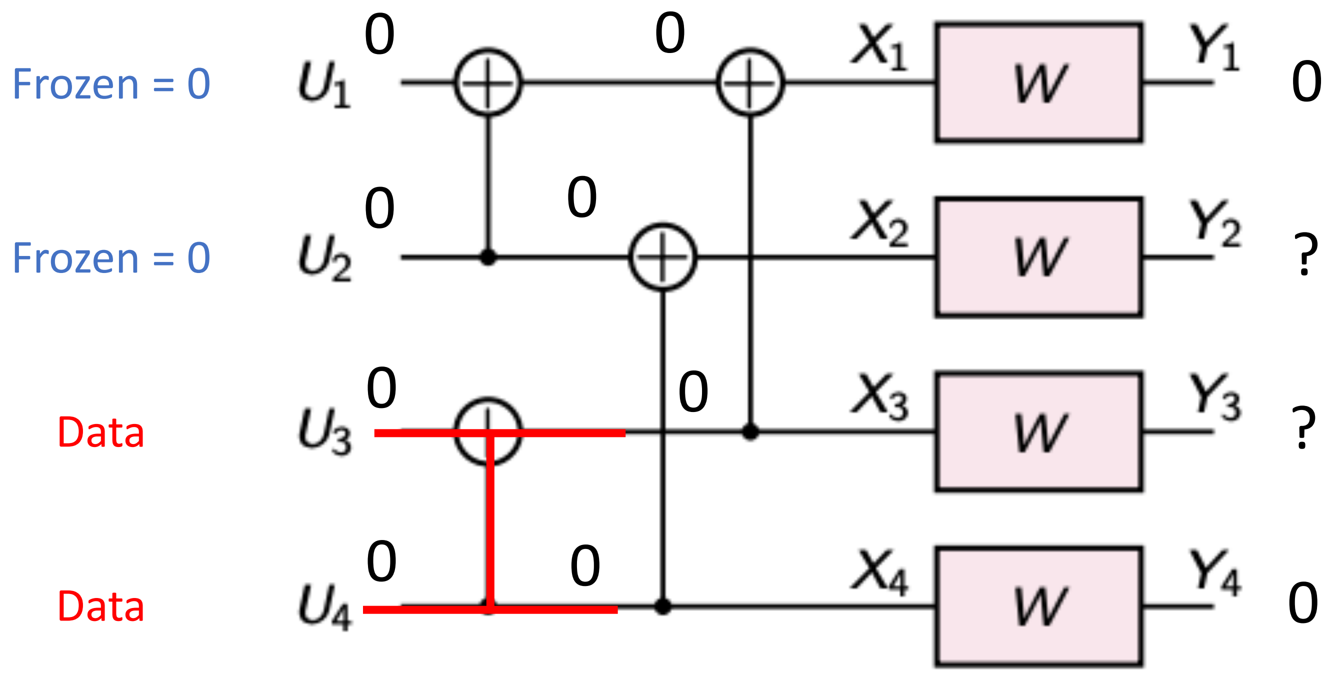


4x4 example: decoding U_4



U_4 decoded successfully!

4x4 example



Decoding successful!

We’re almost done

- For a bigger 8x8 example, see another set of slides created in 2019
  - https://web.stanford.edu/class/ee276/files/SC_decoding_8x8.pdf
- BEC is very special because every intermediate value is either
  - An erasure (?): i.e., you have no idea about its value
  - Or 0/1: i.e., you know the value exactly
- But for other channels such as BSC/BI-AWGN, you have some uncertainty
  - Expressed in terms of log-likelihood ratios (LLRs)
  - The intermediate node values are LLRs given the output and the currently decoded bits
  - After you compute the LLR of the input bit, you perform a hard decoding into 0 or 1 based on the value of the LLR

Thank You!