List-Decoding of Polar Codes

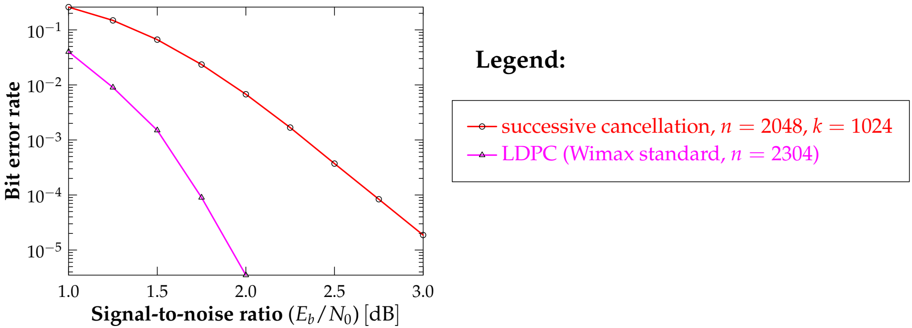
Ido Tal and Alexander Vardy

University of California San Diego

9500 Gilman Drive, La Jolla, CA 92093, USA

Problem and goal

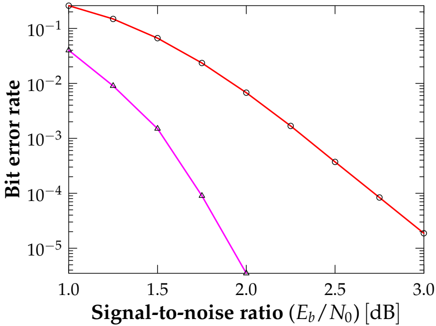
- Channel polarization is slow. For short to moderate code lengths, polar codes have disappointing performance.



- In this talk, we present a generalization of the SC decoder which greatly improves performance at short code lengths.

Avenues for improvement

From here onward, consider a polar code of length $n = 2048$ and rate $R = 0.5$, optimized for a BPSK-AWGN channel with $E_b/N_0 = 2.0$ dB.



Legend:

- Why is our polar code under-performing?
    - Is the SC decoder under-performing?
    - Are the polar codes themselves weak at this length?

A critical look at successive cancellation

Successive Cancellation Decoding

---

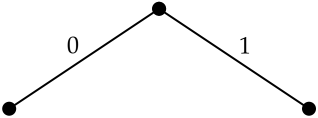
for $i = 0, 1, \ldots, n-1$ **do**
- **if** $\widehat{u}_i$ *is frozen* **then** set $\widehat{u}_i$ accordingly;
- **else**
  - **if** $W_i(\mathbf{y}_0^{n-1}, \widehat{u}_0^{i-1}|0) > W_i(\mathbf{y}_0^{n-1}, \widehat{u}_0^{i-1}|1)$ **then**
    - set $\widehat{u}_i \leftarrow 0$;
  - **else**
    - set $\widehat{u}_i \leftarrow 1$;

Potential weaknesses (interplay):

- Once an unfrozen bit is set, there is "no going back". A bit that was set at step i can not be changed at step $j > i$.
- Knowledge of the value of future frozen bits is not taken into account.

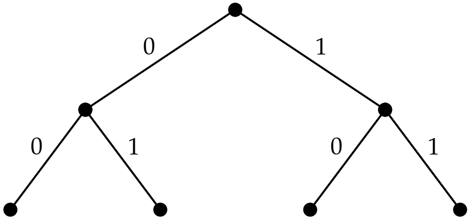
List decoding of polar codes

Key idea: Each time a decision on $\widehat{u}_i$ is needed, split the current decoding path into two paths: **try both $\widehat{u}_i = 0$ and $\widehat{u}_i = 1$.**



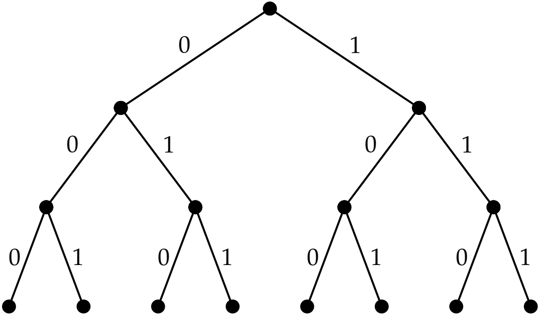
List decoding of polar codes

Key idea: Each time a decision on $\widehat{u}_i$ is needed, split the current decoding path into two paths: **try both $\widehat{u}_i = 0$ and $\widehat{u}_i = 1$.**



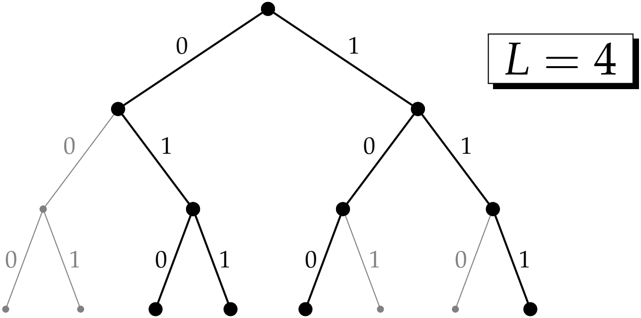
List decoding of polar codes

Key idea: Each time a decision on $\widehat{u}_i$ is needed, split the current decoding path into two paths: **try both $\widehat{u}_i = 0$ and $\widehat{u}_i = 1$.**



List decoding of polar codes

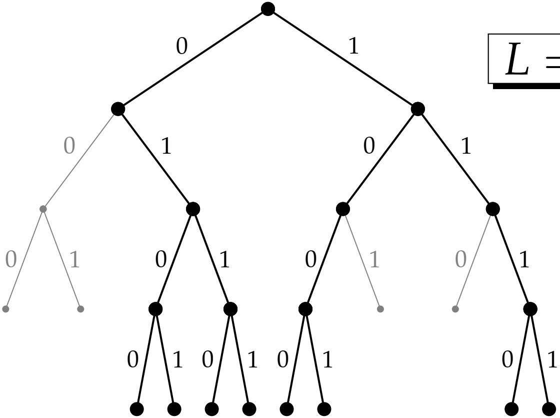
Key idea: Each time a decision on $\widehat{u}_i$ is needed, split the current decoding path into two paths: **try both $\widehat{u}_i = 0$ and $\widehat{u}_i = 1$.**



When the number of paths grows beyond a prescribed threshold L, discard the worst (least probable) paths, and keep only the L best paths.

List decoding of polar codes

Key idea: Each time a decision on $\widehat{u}_i$ is needed, split the current decoding path into two paths: **try both $\widehat{u}_i = 0$ and $\widehat{u}_i = 1$.**

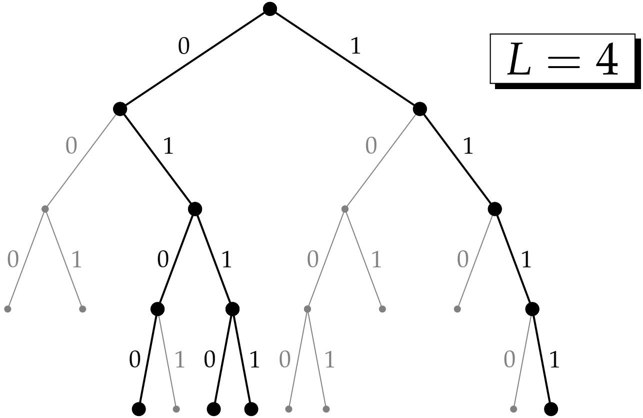


$$\boxed{L = 4}$$

When the number of paths grows beyond a prescribed threshold L, discard the worst (least probable) paths, and keep only the L best paths.

List decoding of polar codes

Key idea: Each time a decision on $\widehat{u}_i$ is needed, split the current decoding path into two paths: **try both $\widehat{u}_i = 0$ and $\widehat{u}_i = 1$.**



When the number of paths grows beyond a prescribed threshold L, discard the worst (least probable) paths, and keep only the L best paths.

$\boxed{\text{At the end, select the single most likely path.}}$

List-decoding: complexity issues

The idea of branching while decoding is not new. In fact a very similar idea was applied for Reed-Muller codes.

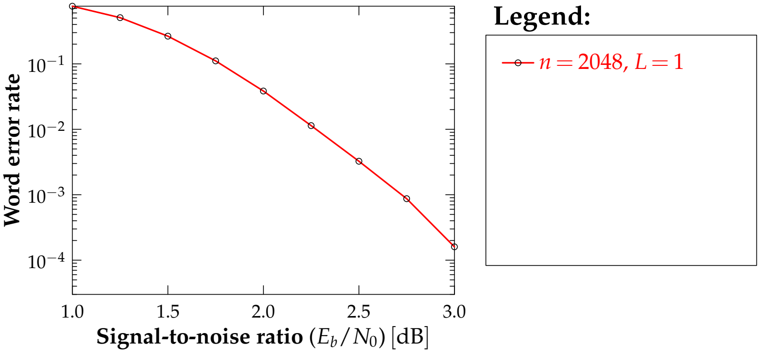
I. Dumer, K. Shabunov, Soft-decision decoding of Reed-Muller codes: recursive lists, *IEEE Trans. on Information Theory*, **52**, pp. 1260–1266, 2006.

Our contribution

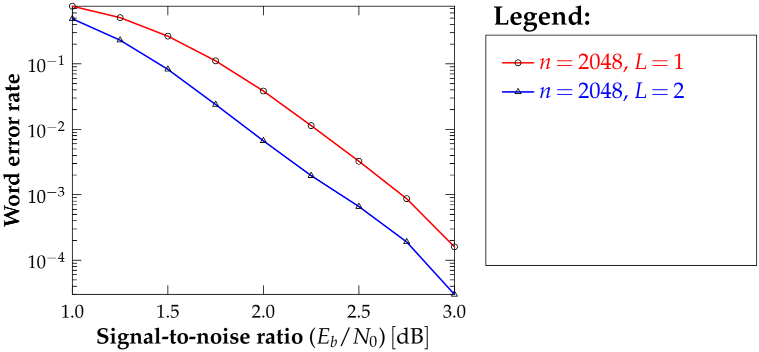
- We consider list decoding of polar codes.
- However, in a naive implementation, the time would be $O(L \cdot n^2)$.
- We show that this can be done in $O(L \cdot n \log n)$ time and $O(L \cdot n)$ space.

We will return to the complexity issue later. For now, let's see how decoding performance is affected.

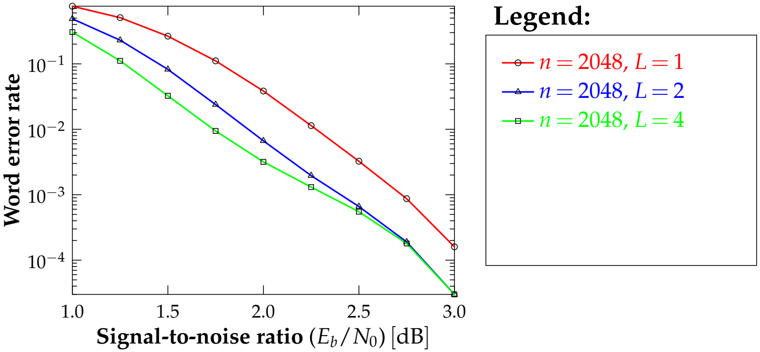
Approaching ML performance



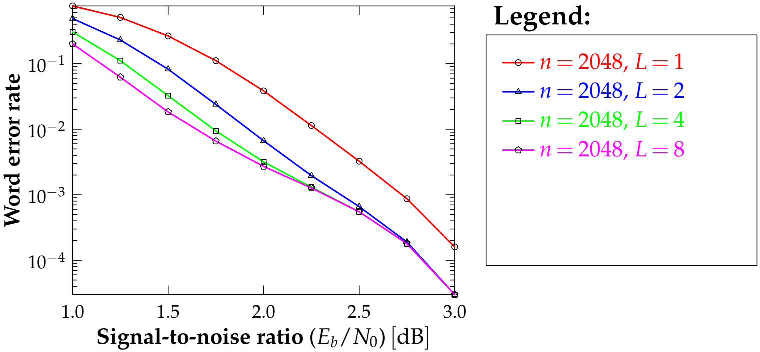
Approaching ML performance



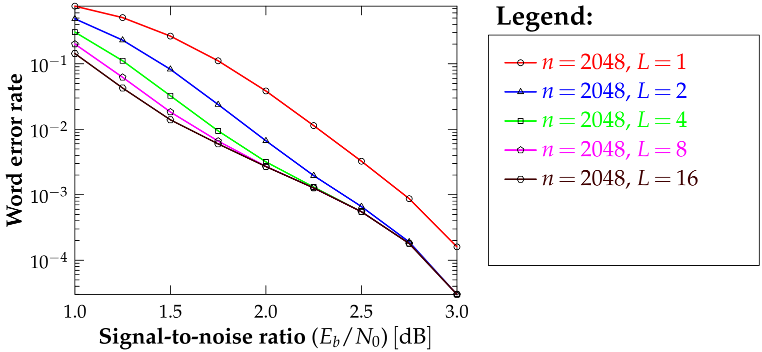
Approaching ML performance



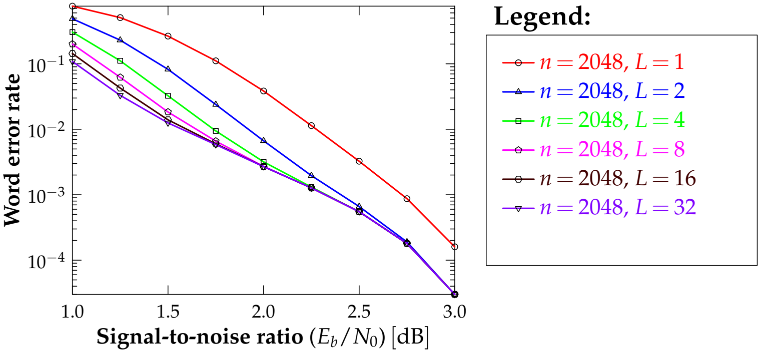
Approaching ML performance



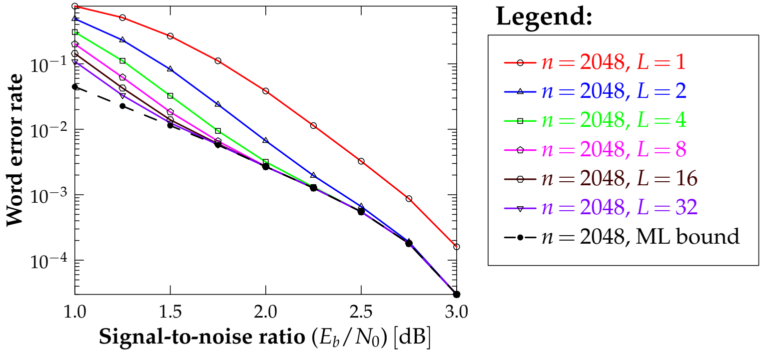
Approaching ML performance



Approaching ML performance

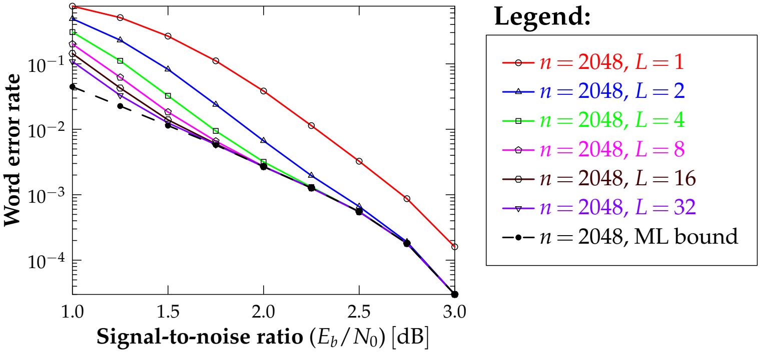


Approaching ML performance



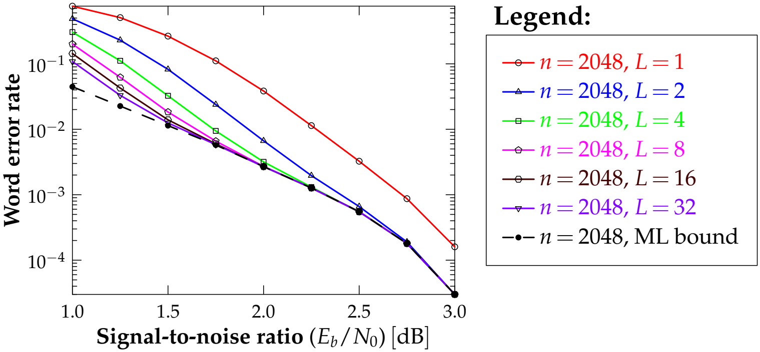
- List-decoding performance quickly approaches that of maximum-likelihood decoding as a function of list-size.

Approaching ML performance



- List-decoding performance quickly approaches that of maximum-likelihood decoding as a function of list-size.
- Good: our decoder is essentially optimal.
- Bad: Still not competitive with LDPC…

Approaching ML performance



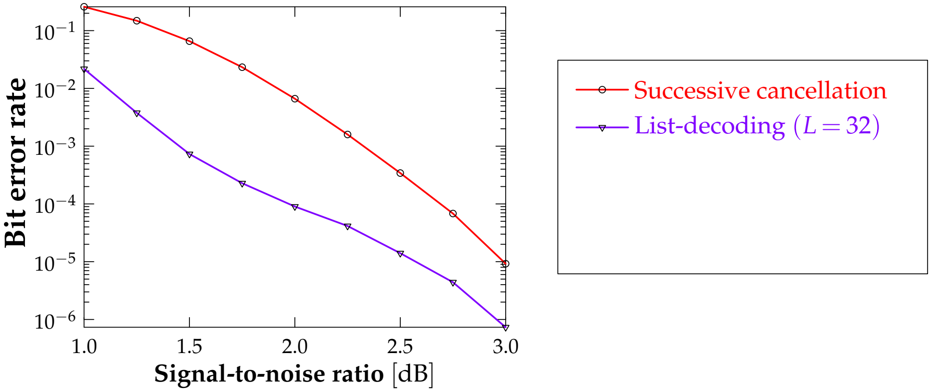
- List-decoding performance quickly approaches that of maximum-likelihood decoding as a function of list-size.
- Good: our decoder is essentially optimal.
- Bad: Still not competitive with LDPC…
- Conclusions: Must somehow "fix" the polar code.

A simple concatenation scheme

- Recall that the last step of decoding was "pick the most likely codeword from the list".
- An error: the transmitted codeword is not the most likely codeword in the list.
- However, very often, the transmitted codeword is still a member of the list.
- We need a "genie" to single-out the transmitted codeword.
- Idea: Let there be $k + r$ unfrozen bits. Of these,
  - Use the first k bits to encode information.
  - Use the last r unfrozen bits to encode the CRC value of the first k bits.
  - Pick the most probable codeword on the list with correct CRC.

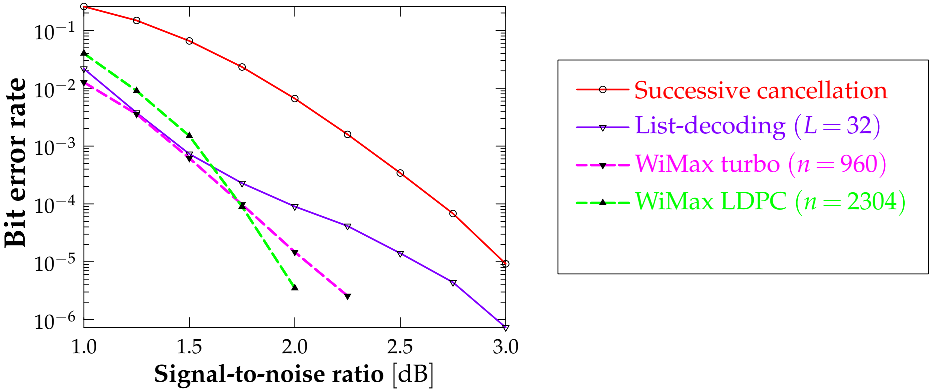
Approaching LDPC performance

Simulation results for a polar code of length $n = 2048$ and rate $R = 0.5$, optimized for a BPSK-AWGN channel with $E_b/N_0 = 2.0\,$dB.



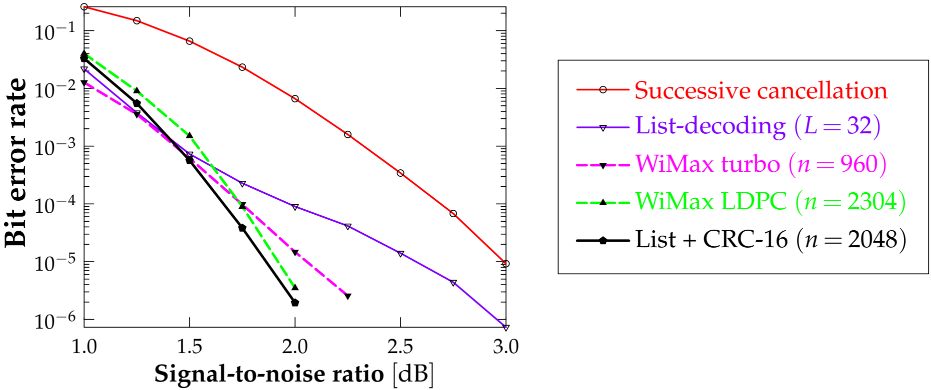
Approaching LDPC performance

Simulation results for a polar code of length $n = 2048$ and rate $R = 0.5$, optimized for a BPSK-AWGN channel with $E_b/N_0 = 2.0\,\text{dB}$.



Approaching LDPC performance

Simulation results for a polar code of length $n = 2048$ and rate $R = 0.5$, optimized for a BPSK-AWGN channel with $E_b/N_0 = 2.0\,\text{dB}$.



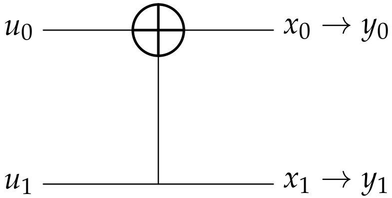
Polar codes (+CRC) under list decoding are competitive with the best LDPC codes at lengths as short as $n = 2048$.

Quadratic complexity of list decoding

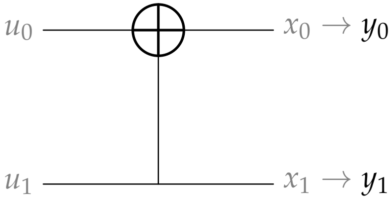
Naive implementation recap

- In a naive implementation, the decoding paths are independent. They don't share information.
- Each decoding path has a set of variables associated with it. For example, at stage i, each decoding path must remember the values of the bits $\widehat{u}_0, \widehat{u}_1, \ldots, \widehat{u}_{i-1}$.
- It turns out (as we shall see) that each decoding path has $\Theta(n)$ memory associated with it.
- When a path is split in two, one decoding path is left with the original variables while the other must be handed a copy of them.
- Each copy operation takes $O(n)$ time.
- Thus, the overall time complexity is $O(L \cdot n^2)$.

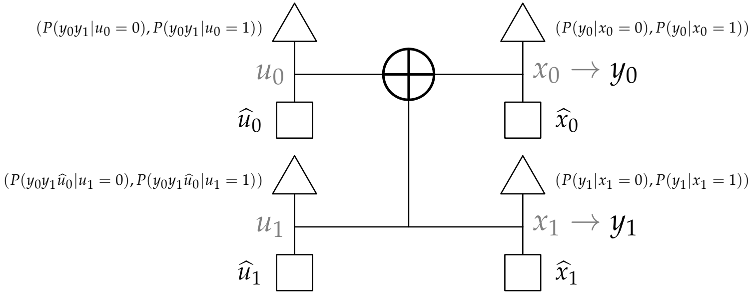
A closer look at successive cancellation



A closer look at successive cancellation



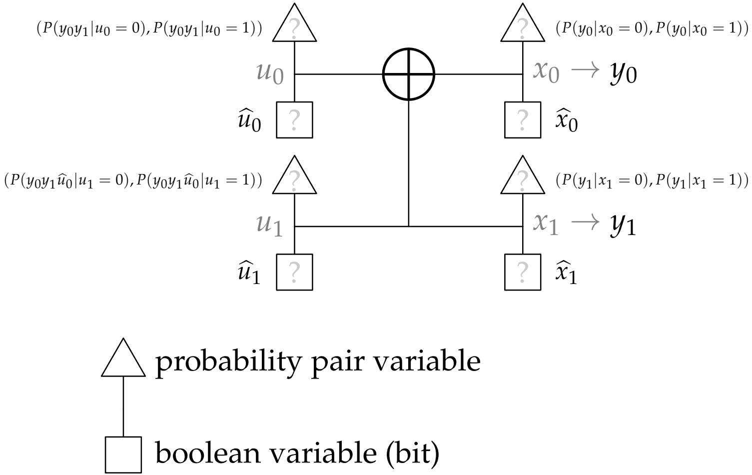
A closer look at successive cancellation



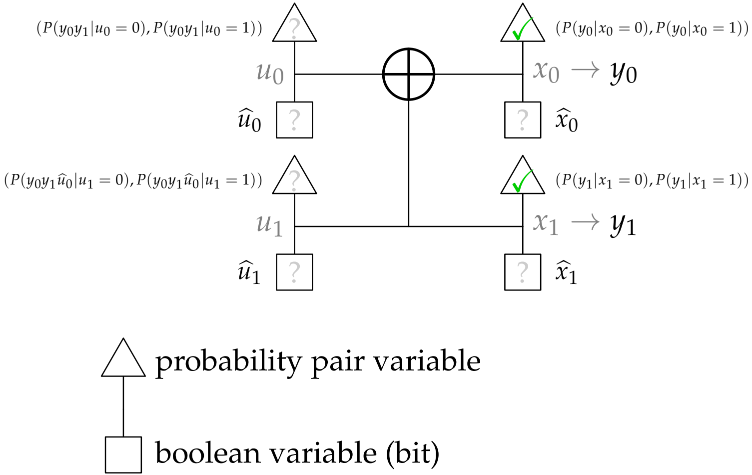
probability pair variable

boolean variable (bit)

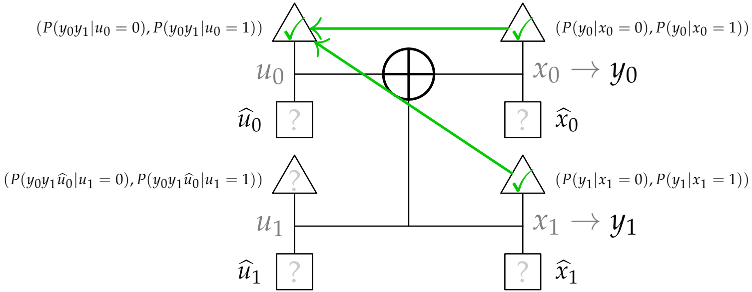
A closer look at successive cancellation

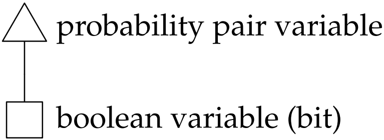


A closer look at successive cancellation

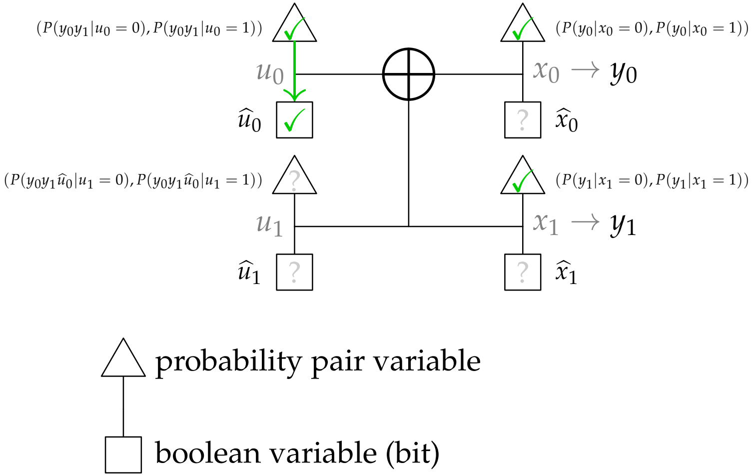


A closer look at successive cancellation

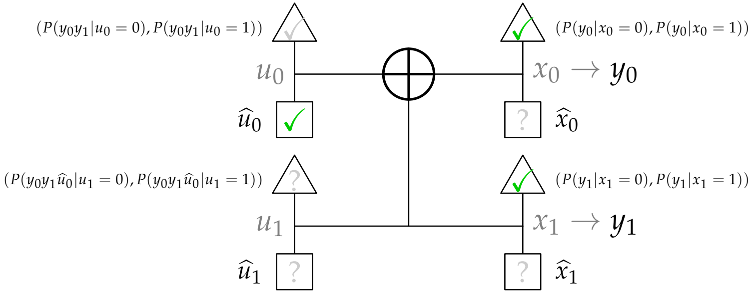




A closer look at successive cancellation



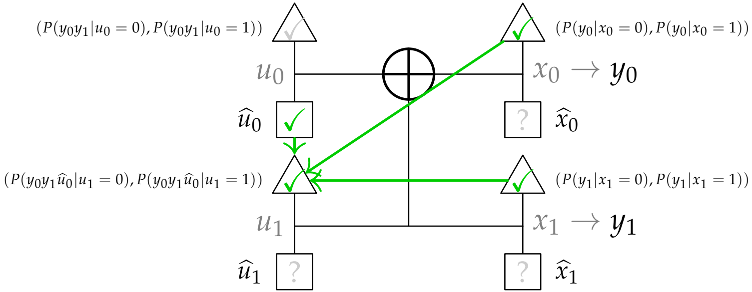
A closer look at successive cancellation

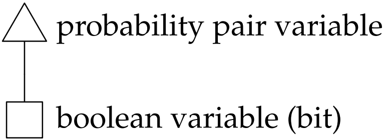


probability pair variable

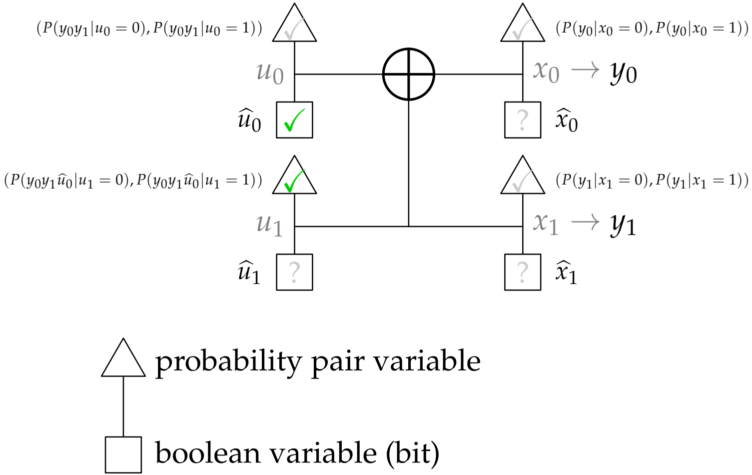
boolean variable (bit)

A closer look at successive cancellation

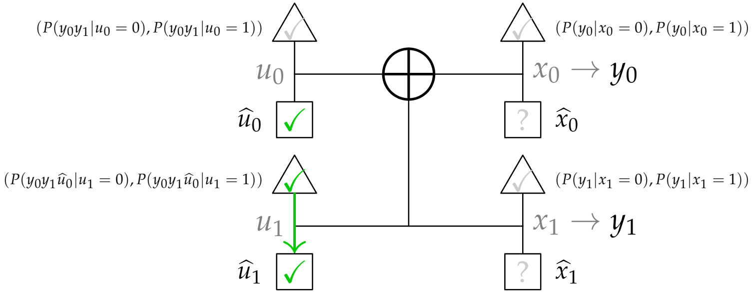




A closer look at successive cancellation



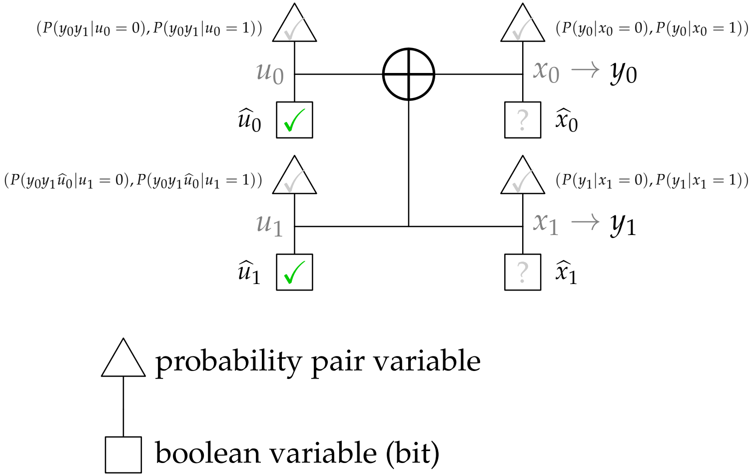
A closer look at successive cancellation



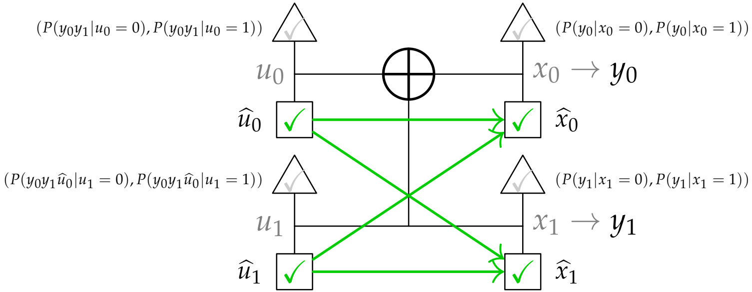
probability pair variable

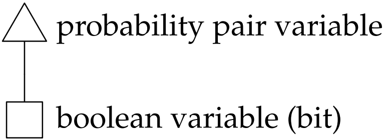
boolean variable (bit)

A closer look at successive cancellation

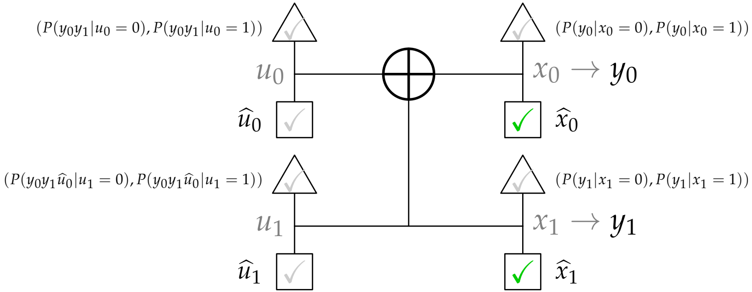


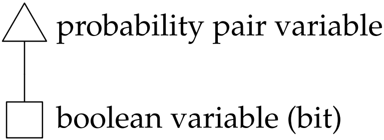
A closer look at successive cancellation





A closer look at successive cancellation

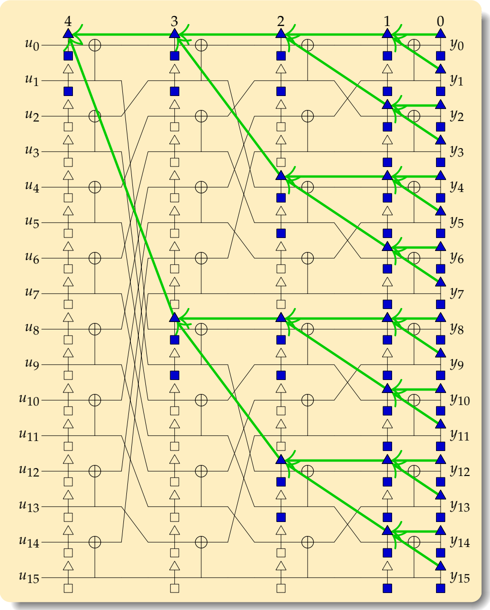




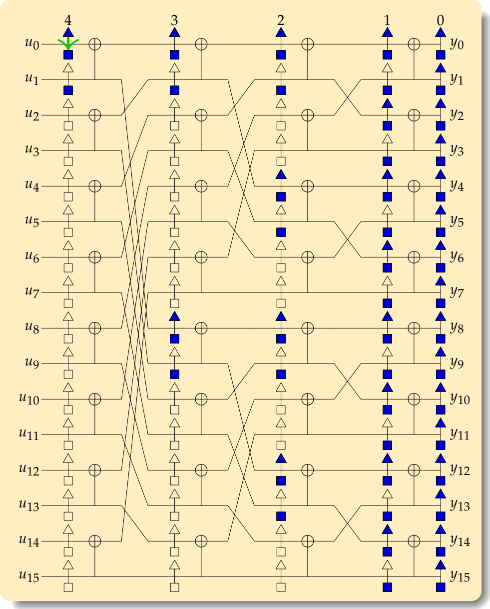
A larger example

Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.



A larger example



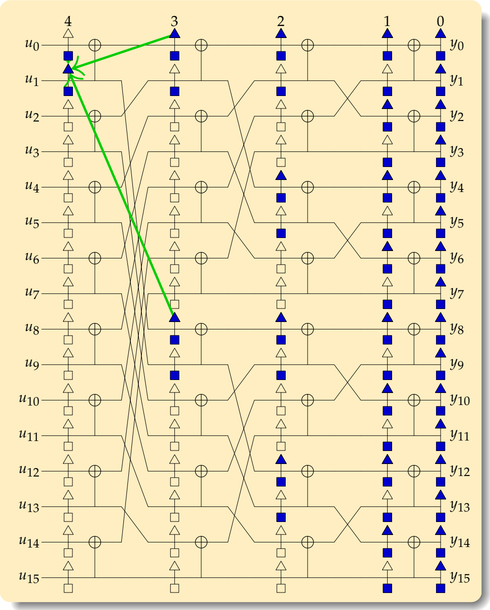
Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.

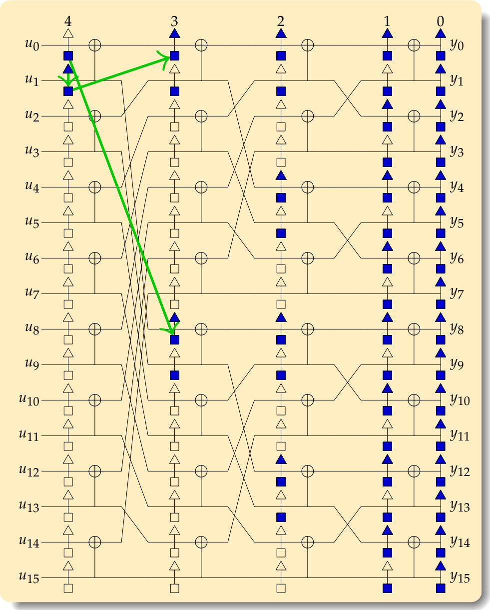
A larger example

Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.



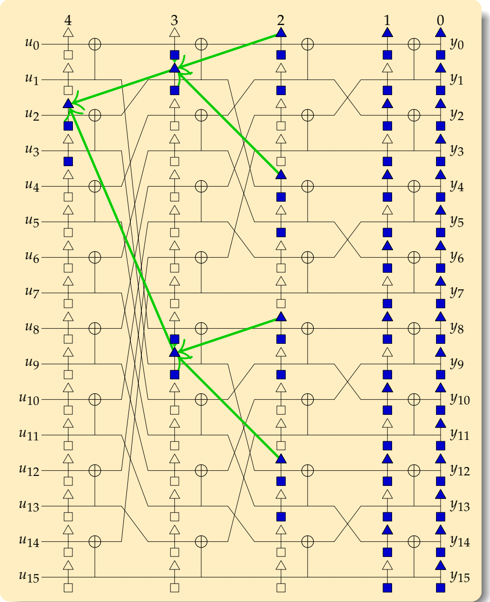
A larger example



Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.

A larger example



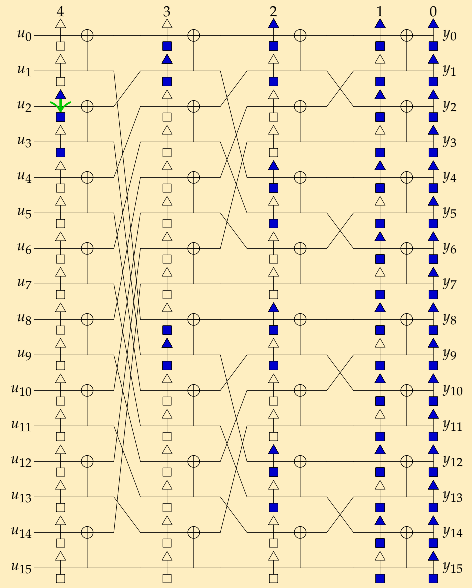
Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.

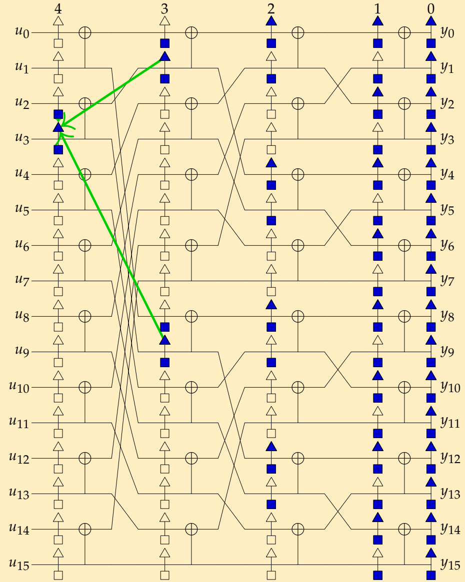
A larger example

Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.



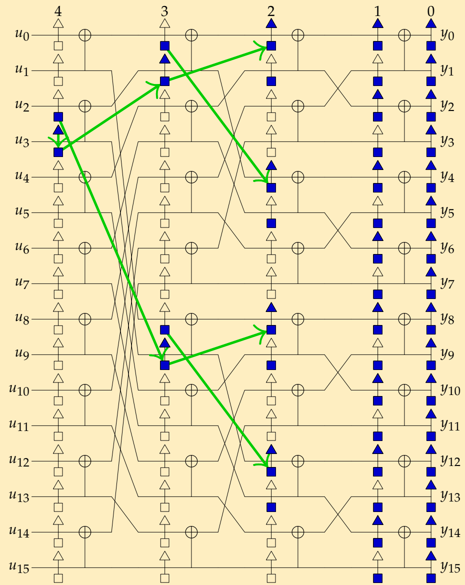
A larger example



Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.

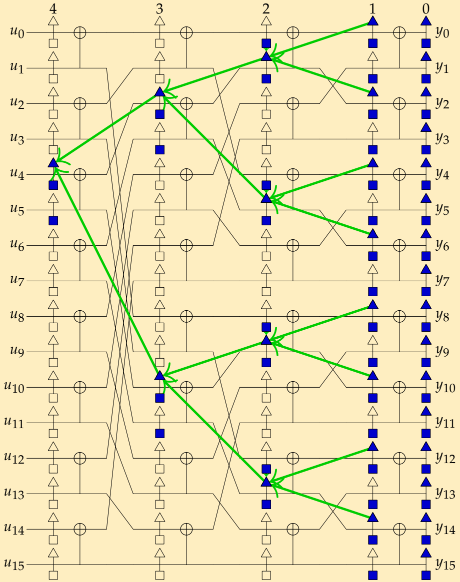
A larger example



Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.

A larger example



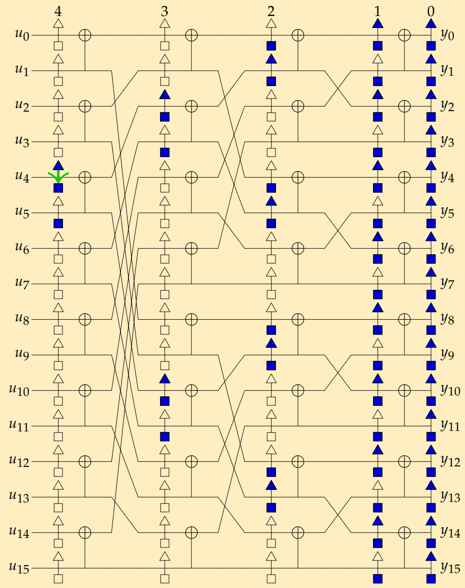
Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.

A larger example

Key point

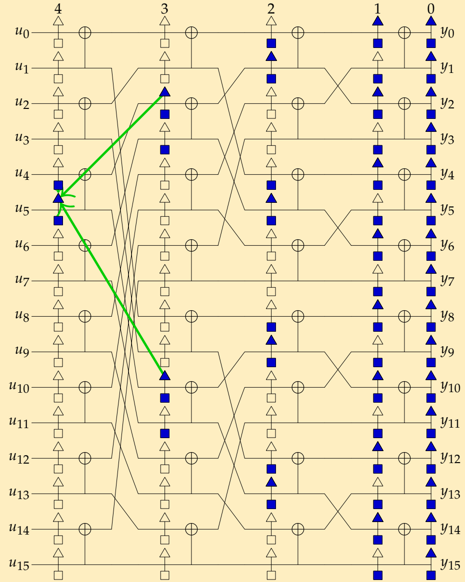
The memory needed to hold the variables at level t is $O(n/2^t)$.



A larger example

Key point

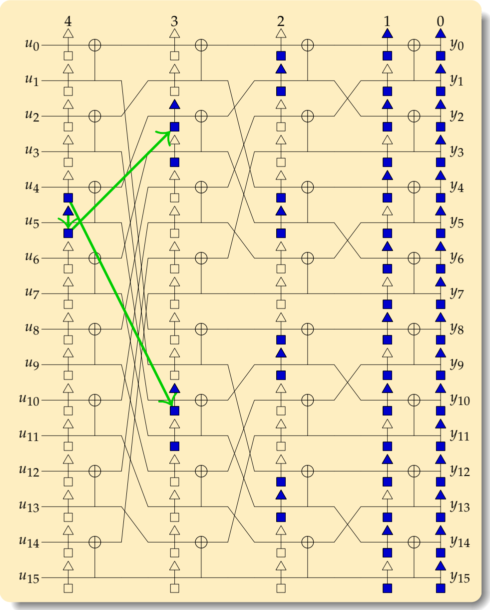
The memory needed to hold the variables at level t is $O(n/2^t)$.



A larger example

Key point

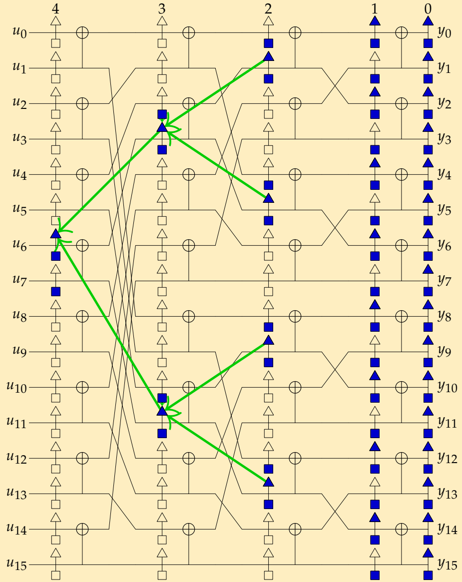
The memory needed to hold the variables at level t is $O(n/2^t)$.



A larger example

Key point

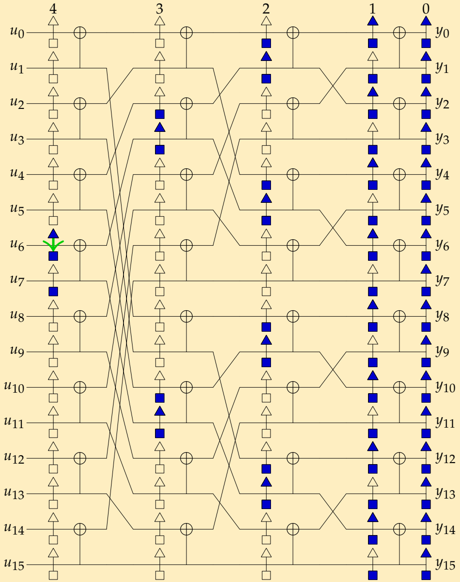
The memory needed to hold the variables at level t is $O(n/2^t)$.



A larger example

Key point

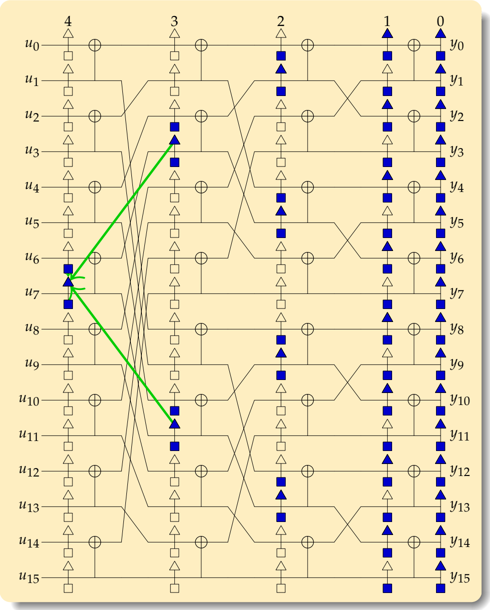
The memory needed to hold the variables at level t is $O(n/2^t)$.



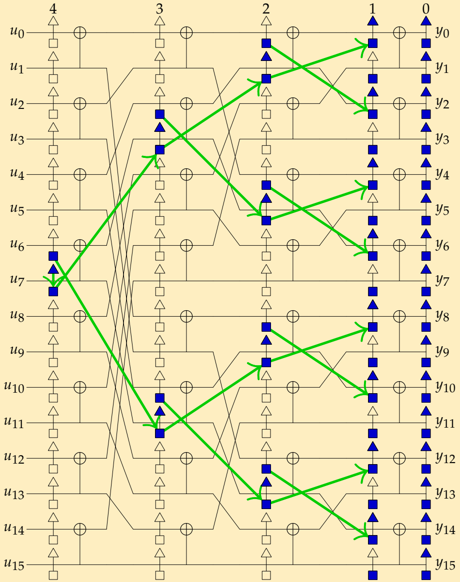
A larger example

Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.



A larger example



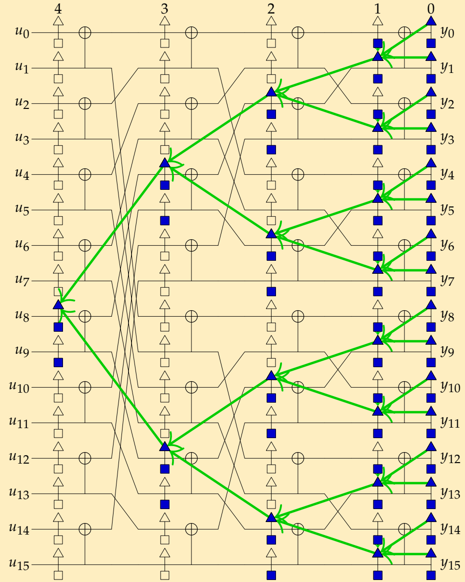
Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.

A larger example

Key point

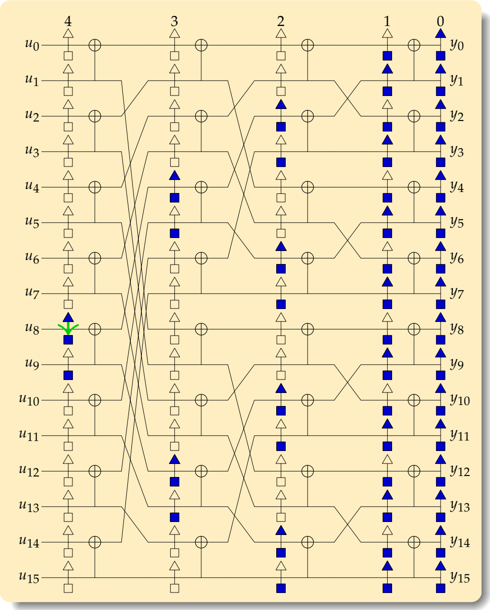
The memory needed to hold the variables at level t is $O(n/2^t)$.



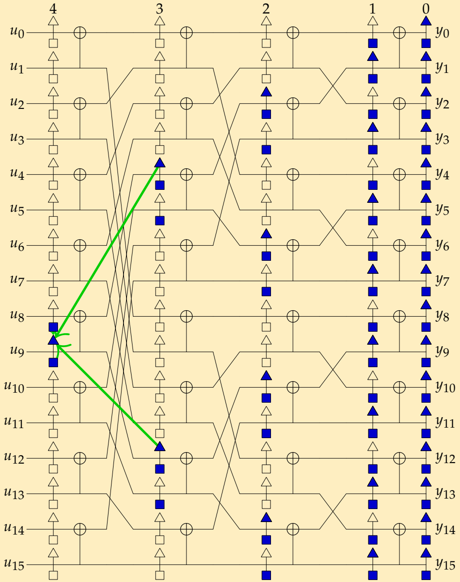
A larger example

Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.



A larger example



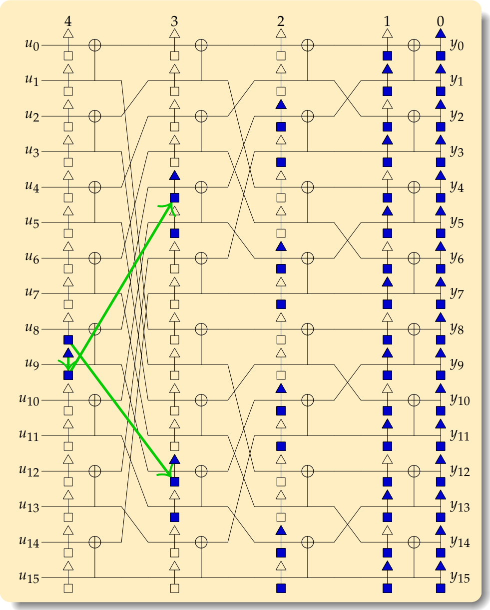
Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.

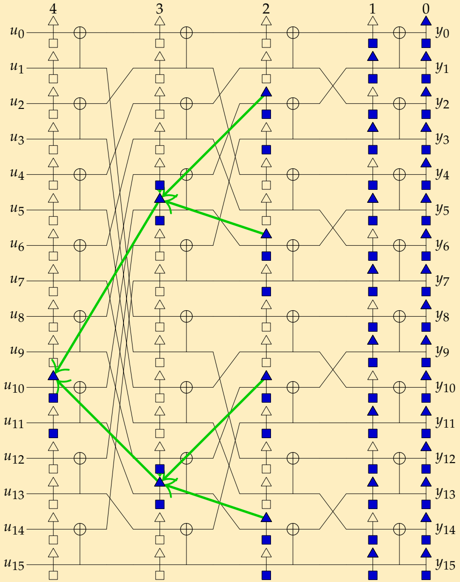
A larger example

Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.



A larger example



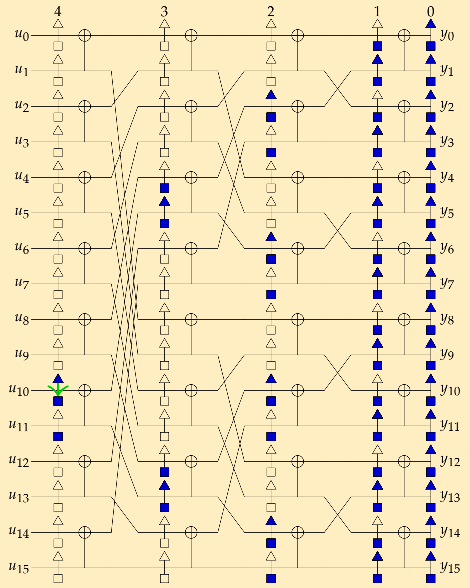
Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.

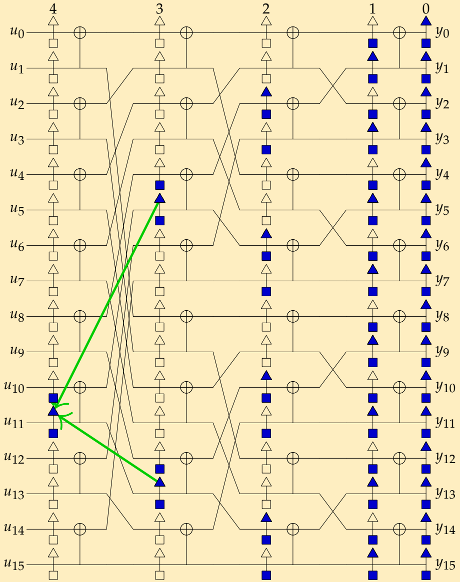
A larger example

Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.



A larger example



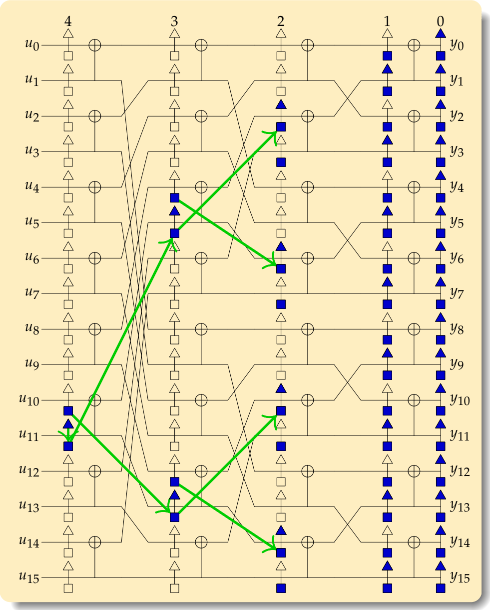
Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.

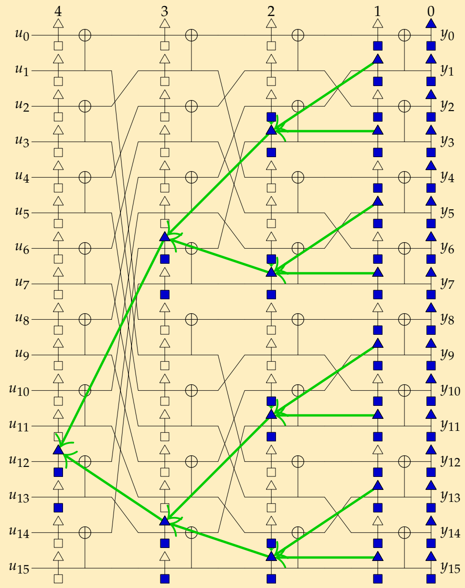
A larger example

Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.



A larger example



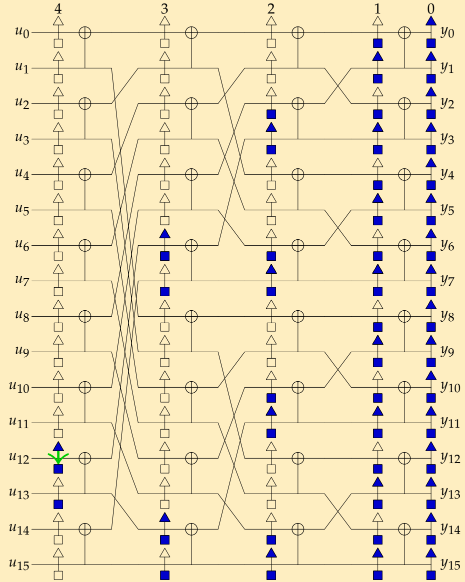
Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.

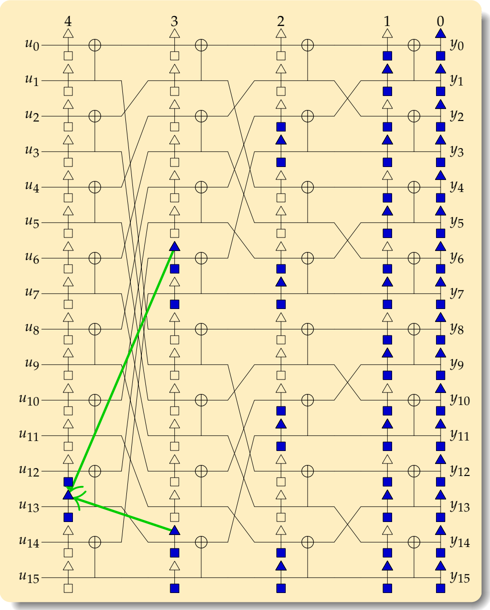
A larger example

Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.



A larger example



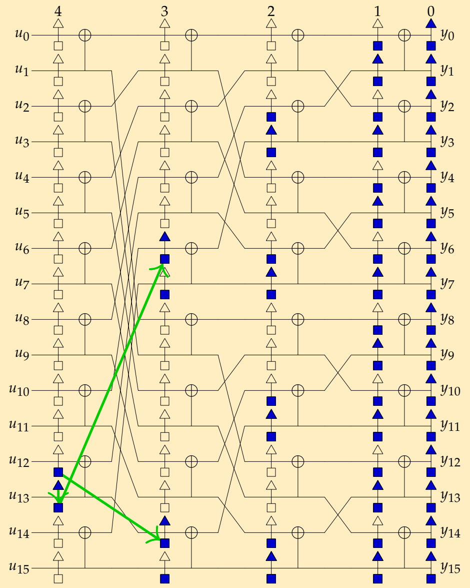
Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.

A larger example

Key point

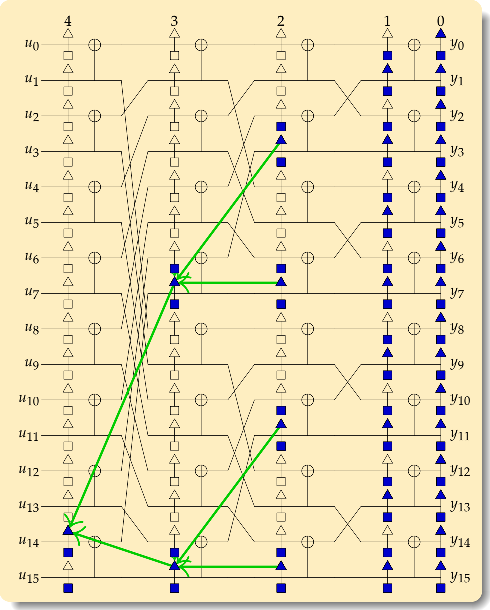
The memory needed to hold the variables at level t is $O(n/2^t)$.



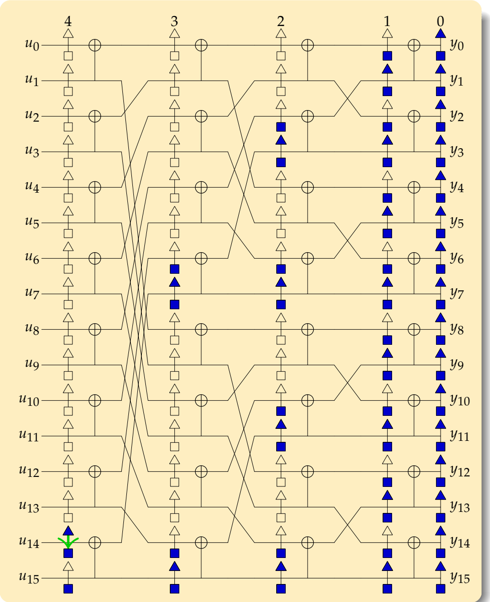
A larger example

Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.



A larger example



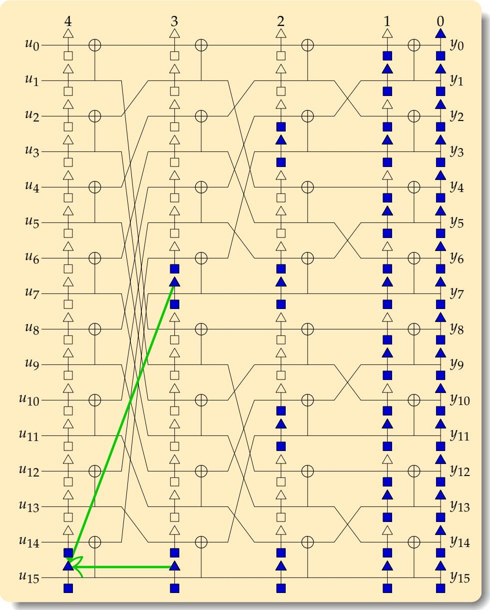
Key point

The memory needed to hold the variables at level t is $O(n/2^t)$.

A larger example

Key point

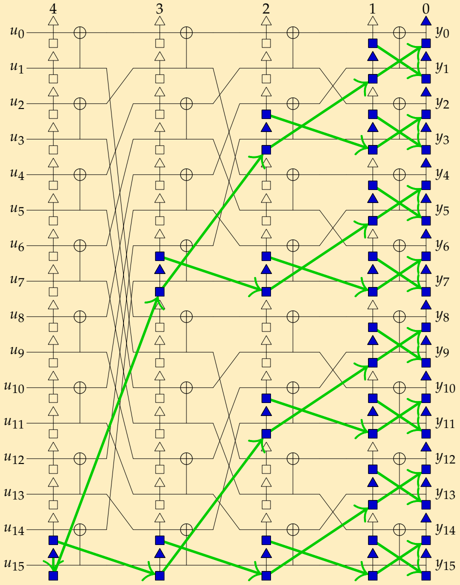
The memory needed to hold the variables at level t is $O(n/2^t)$.



A larger example

Key point

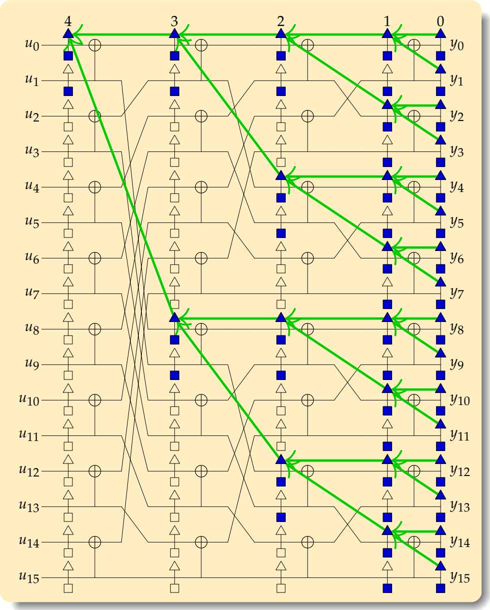
The memory needed to hold the variables at level t is $O(n/2^t)$.



A larger example

Key point

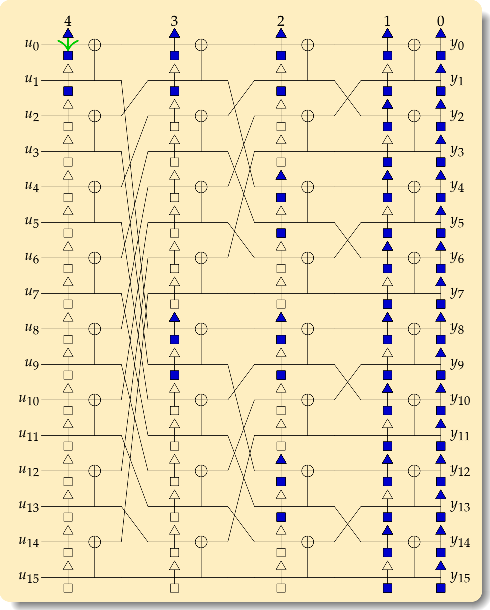
Level t is written to once every $O(2^{m-t})$ stages.



A larger example

Key point

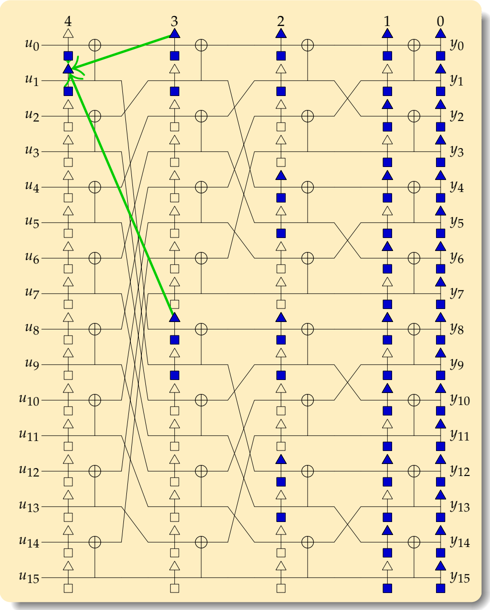
Level t is written to once every $O(2^{m-t})$ stages.



A larger example

Key point

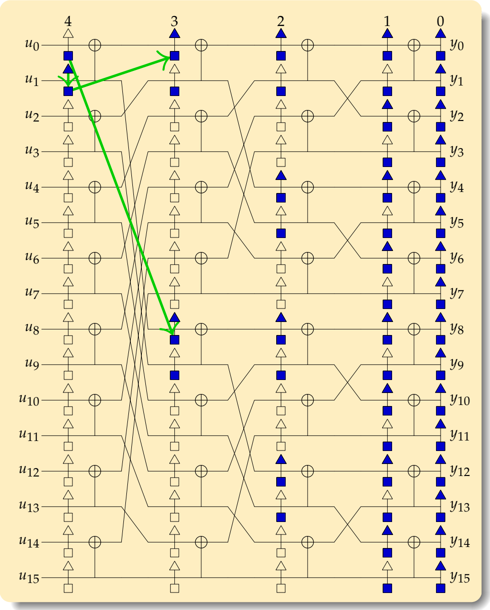
Level t is written to once every $O(2^{m-t})$ stages.



A larger example

Key point

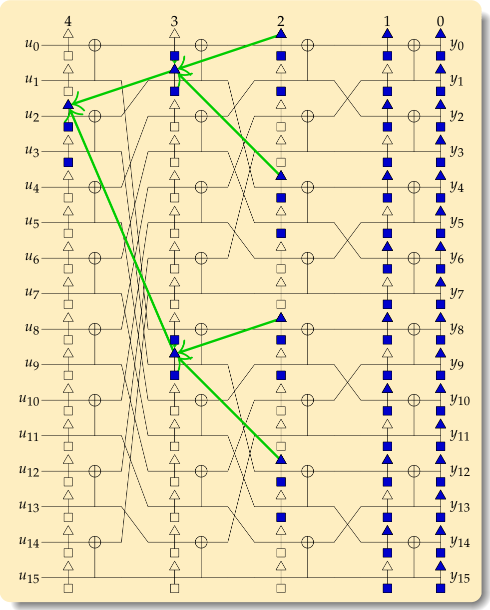
Level t is written to once every $O(2^{m-t})$ stages.



A larger example

Key point

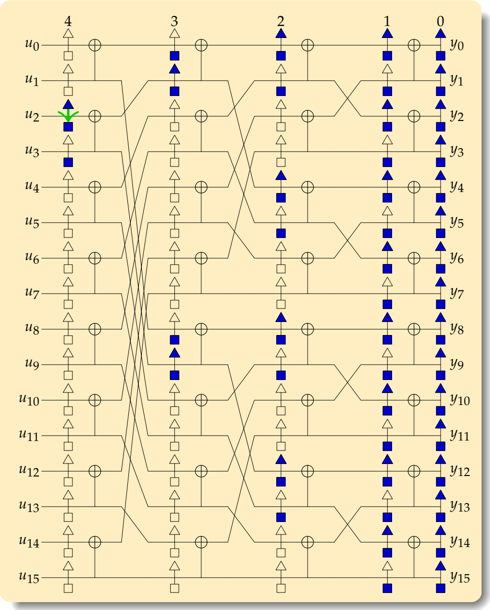
Level t is written to once every $O(2^{m-t})$ stages.



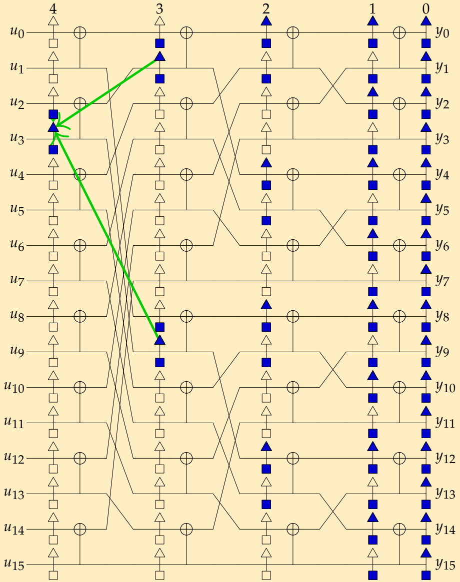
A larger example

Key point

Level t is written to once every $O(2^{m-t})$ stages.



A larger example



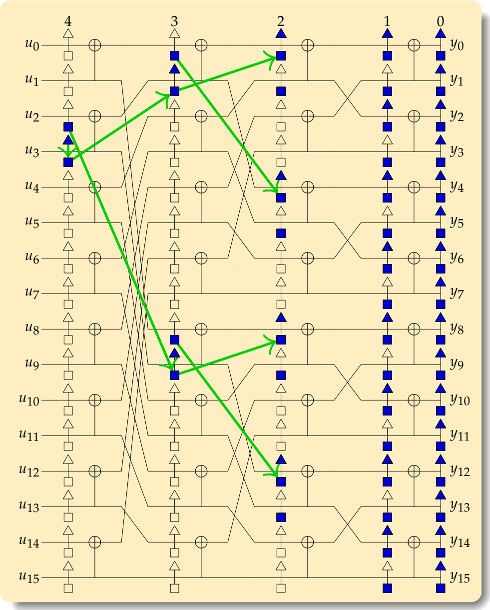
Key point

Level t is written to once every $O(2^{m-t})$ stages.

A larger example

Key point

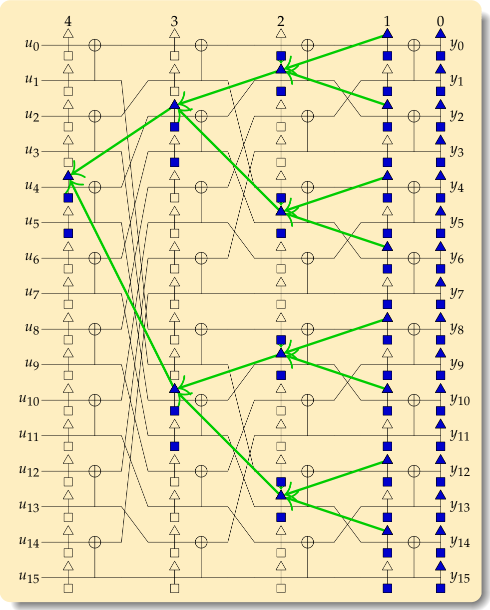
Level t is written to once every $O(2^{m-t})$ stages.



A larger example

Key point

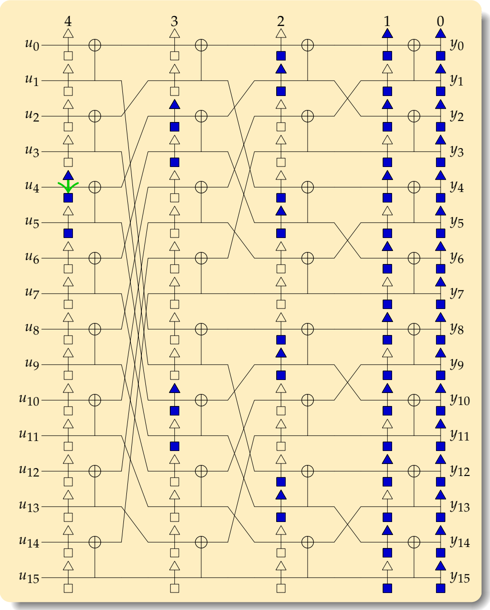
Level t is written to once every $O(2^{m-t})$ stages.



A larger example

Key point

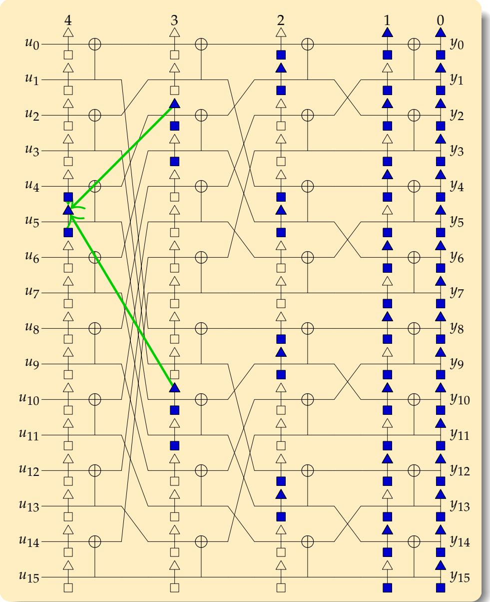
Level t is written to once every $O(2^{m-t})$ stages.



A larger example

Key point

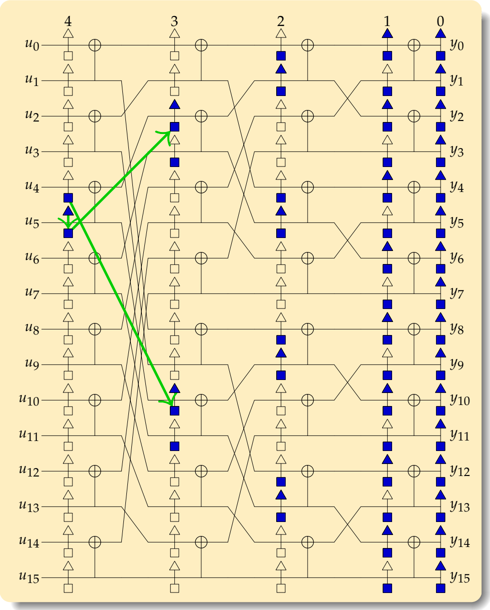
Level t is written to once every $O(2^{m-t})$ stages.



A larger example

Key point

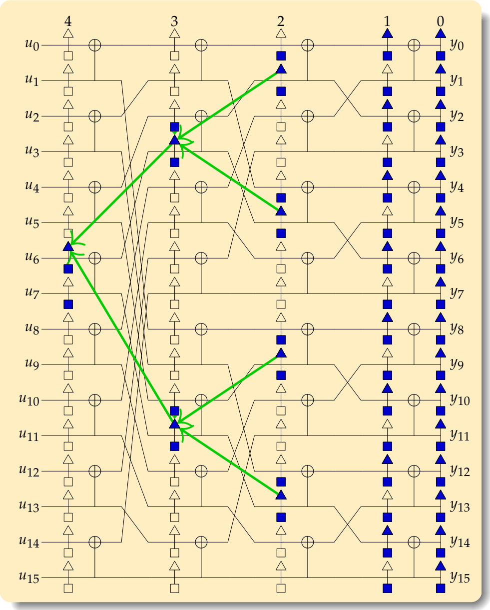
Level t is written to once every $O(2^{m-t})$ stages.



A larger example

Key point

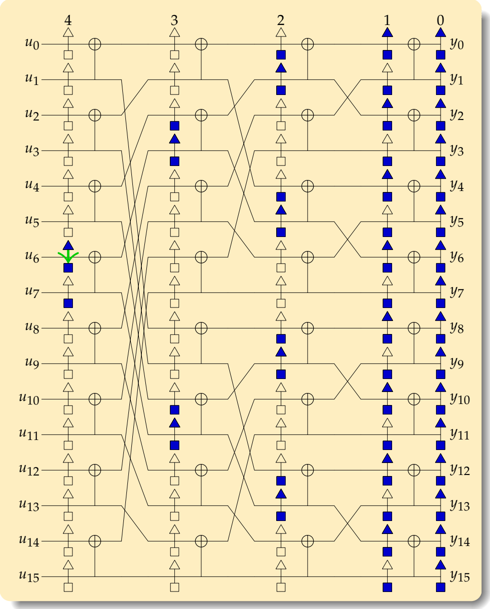
Level t is written to once every $O(2^{m-t})$ stages.



A larger example

Key point

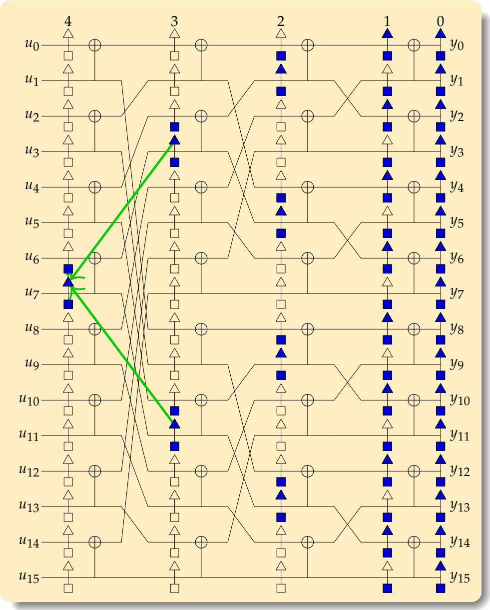
Level t is written to once every $O(2^{m-t})$ stages.



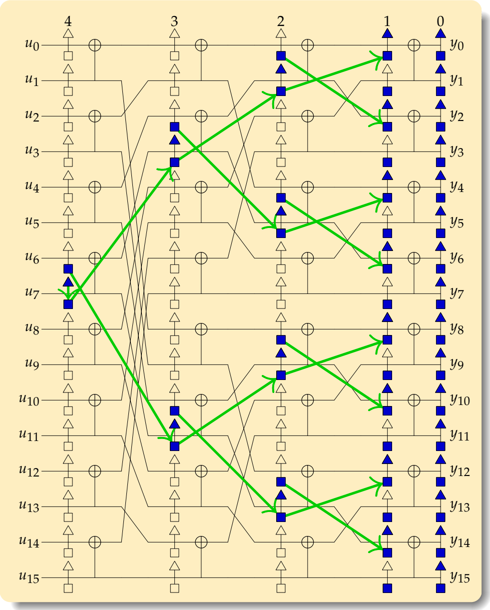
A larger example

Key point

Level t is written to once every $O(2^{m-t})$ stages.



A larger example



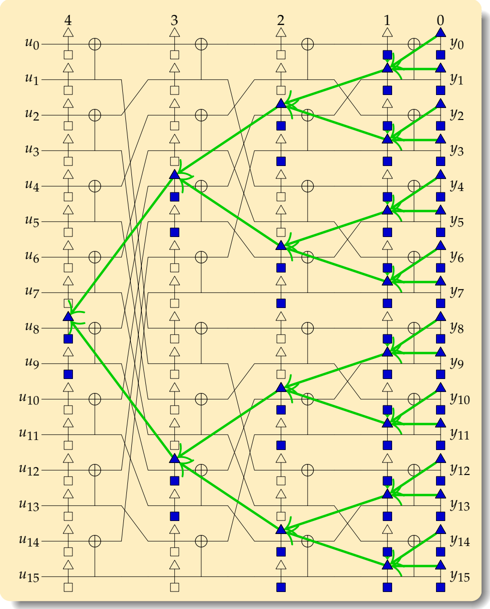
Key point

Level t is written to once every $O(2^{m-t})$ stages.

A larger example

Key point

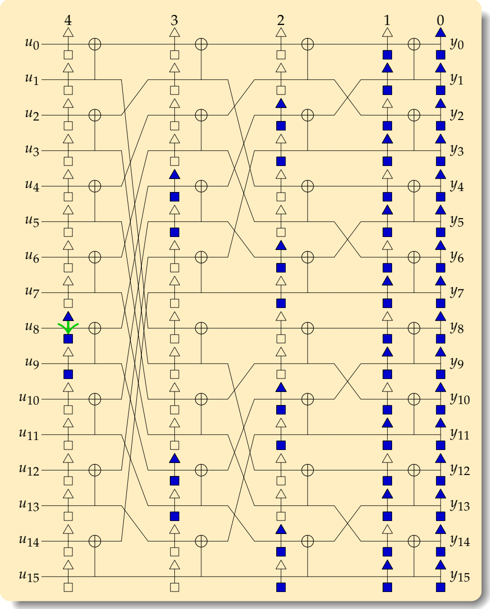
Level t is written to once every $O(2^{m-t})$ stages.



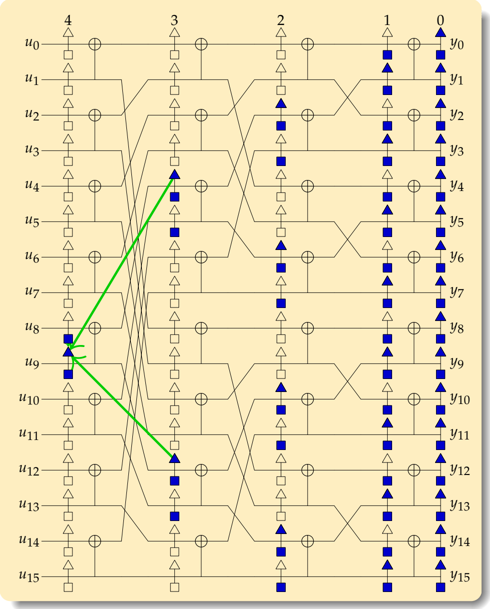
A larger example

Key point

Level t is written to once every $O(2^{m-t})$ stages.



A larger example



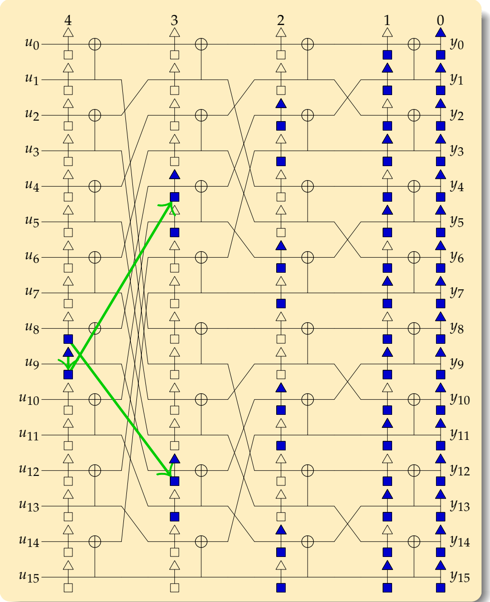
Key point

Level t is written to once every $O(2^{m-t})$ stages.

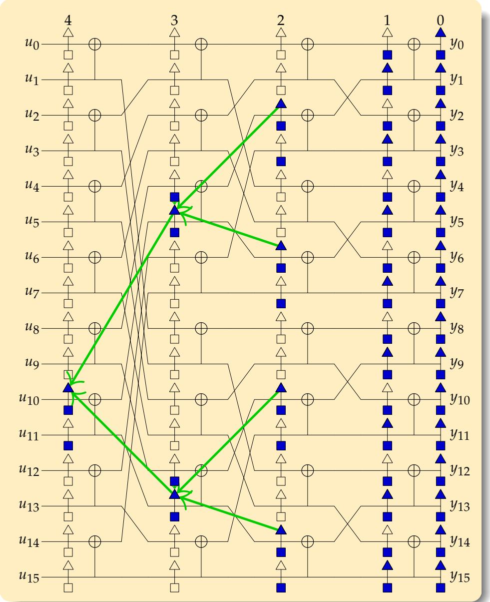
A larger example

Key point

Level t is written to once every $O(2^{m-t})$ stages.



A larger example



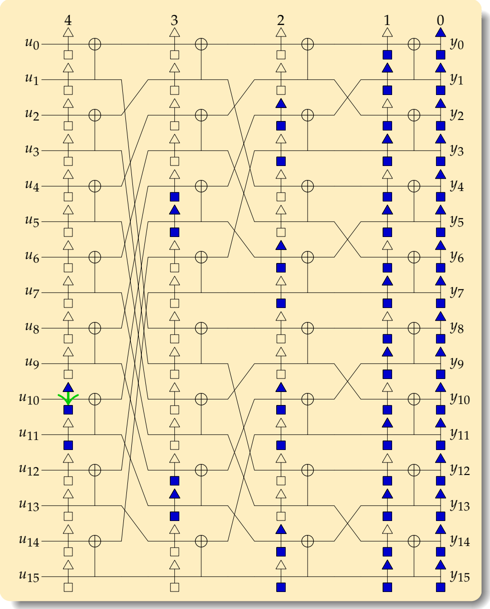
Key point

Level t is written to once every $O(2^{m-t})$ stages.

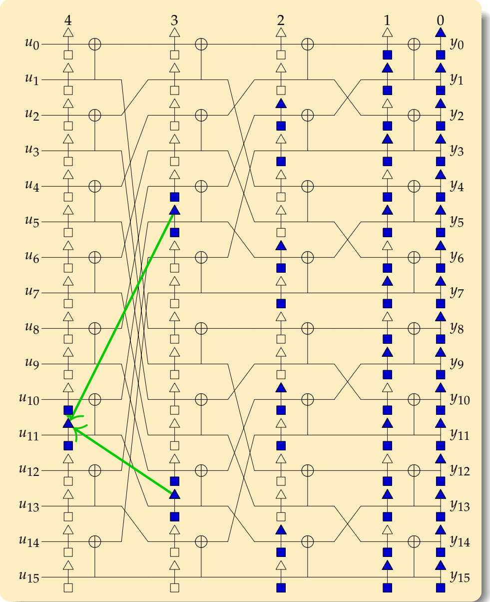
A larger example

Key point

Level t is written to once every $O(2^{m-t})$ stages.



A larger example



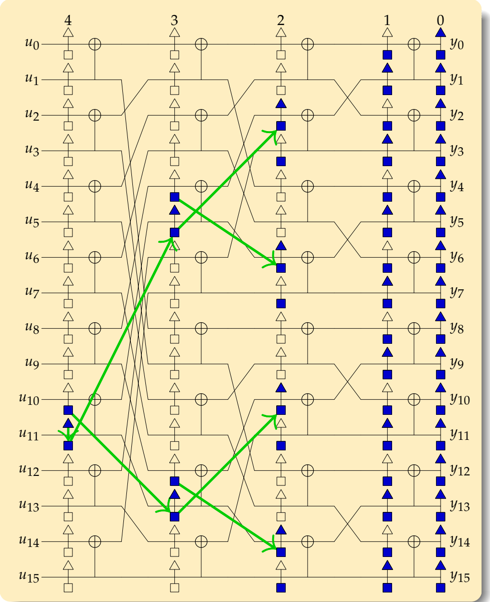
Key point

Level t is written to once every $O(2^{m-t})$ stages.

A larger example

Key point

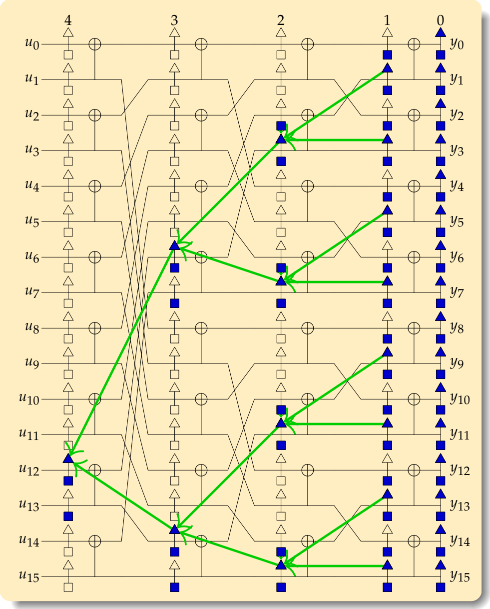
Level t is written to once every $O(2^{m-t})$ stages.



A larger example

Key point

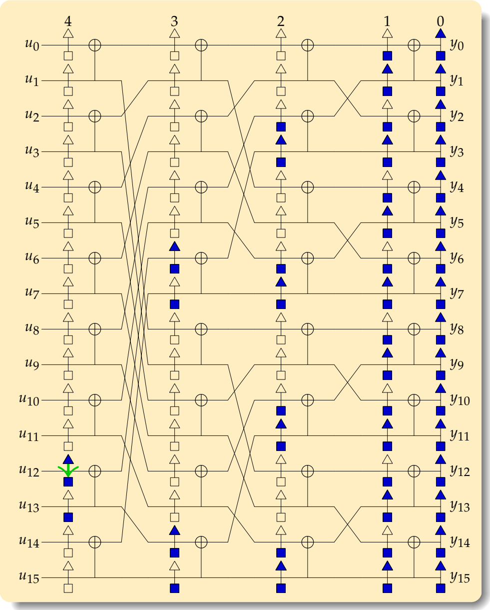
Level t is written to once every $O(2^{m-t})$ stages.



A larger example

Key point

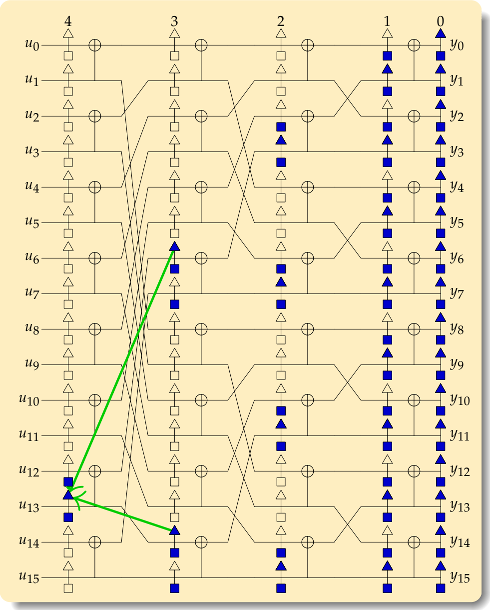
Level t is written to once every $O(2^{m-t})$ stages.



A larger example

Key point

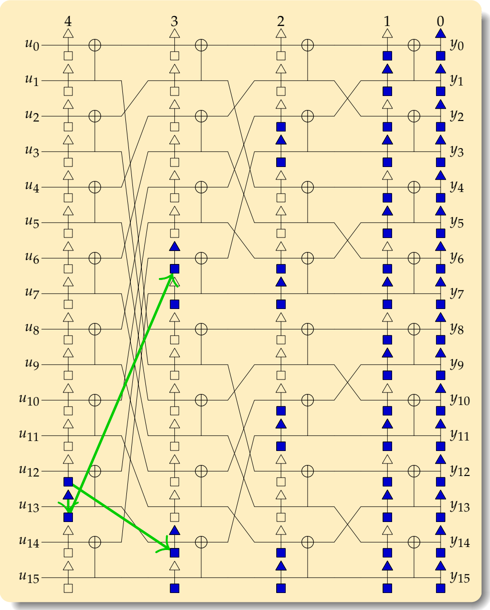
Level t is written to once every $O(2^{m-t})$ stages.



A larger example

Key point

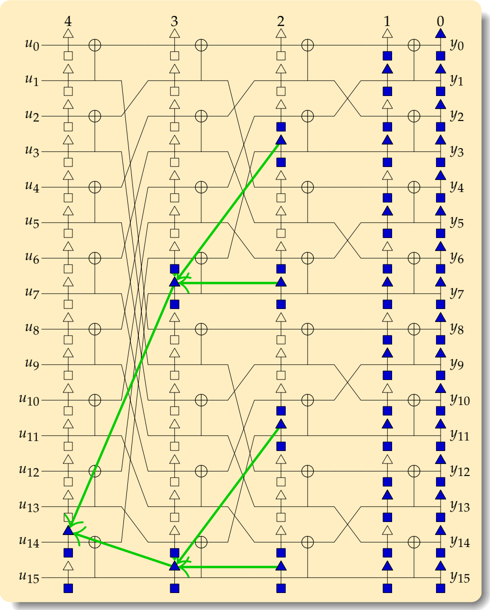
Level t is written to once every $O(2^{m-t})$ stages.



A larger example

Key point

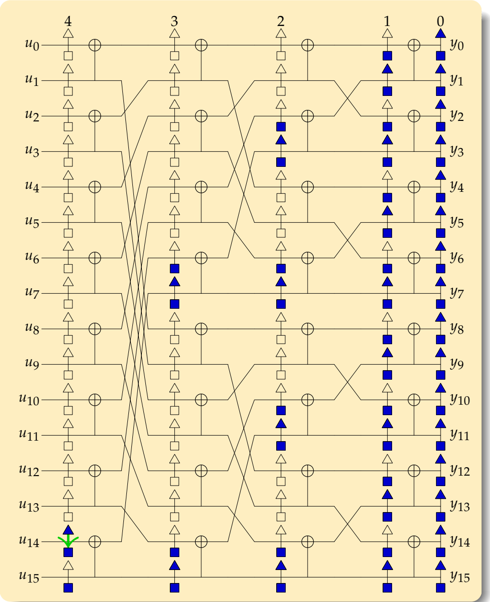
Level t is written to once every $O(2^{m-t})$ stages.



A larger example

Key point

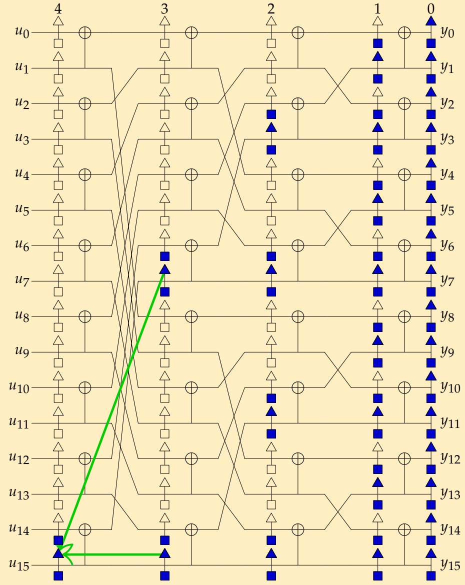
Level t is written to once every $O(2^{m-t})$ stages.



A larger example

Key point

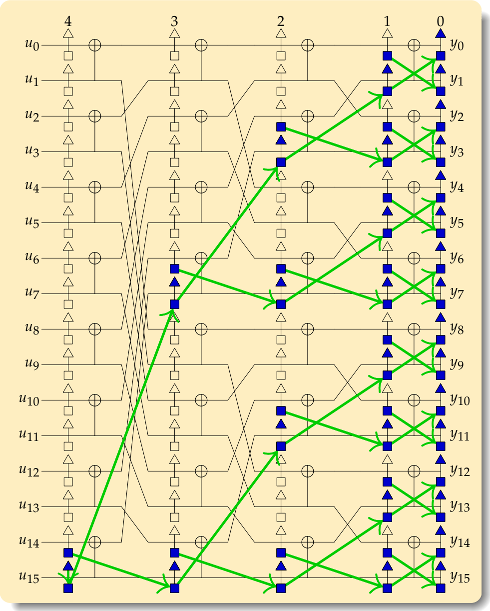
Level t is written to once every $O(2^{m-t})$ stages.



A larger example

Key point

Level t is written to once every $O(2^{m-t})$ stages.



Application to list decoding

- In a naive implementation, at each split we make a copy of the variables.
- We can do better:
  - At each split, flag the corresponding variables as belonging to both paths.
  - Give each path a unique variable (make a copy) only before that variable will be written to.
  - If a path is killed, deflag its corresponding variables.
- Thus, instead of wasting a lot of time on copy operations at each stage, we typically perform only a small number of copy operations.

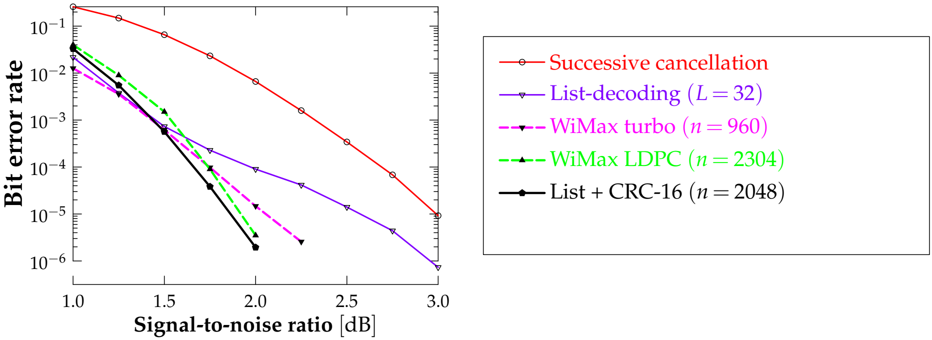
This was a mile high view, there are many details to be filled (book-keeping, data structures), but the end result is a running time of $O(L \cdot n \log n)$ with $O(L \cdot n)$ memory requirements.

Very recent results

Gross and Sarkis (MacGill University) have recently attained the following results.

- Full independent verification of our simulation data.
- Further improvement of performance using systematic polar codes.

E. Arıkan, Systematic polar codes, *IEEE Comm. Letters*, accepted for publication.



Very recent results

Gross and Sarkis (MacGill University) have recently attained the following results.

- Full independent verification of our simulation data.
- Further improvement of performance using systematic polar codes.

E. Arıkan, Systematic polar codes, *IEEE Comm. Letters*, accepted for publication.