List Decoding of Universal Polar Codes

Boaz Shuval and Ido Tal

Overview

- Universal polarization:
  - memoryless setting: Şaşoğlu & Wang, ISIT 2011
  - settings with memory: Shuval & Tal, ISIT 2019
- Error probability analysis based on successive cancellation (SC) decoding
- Potential for better results using successive cancellation list (SCL) decoding

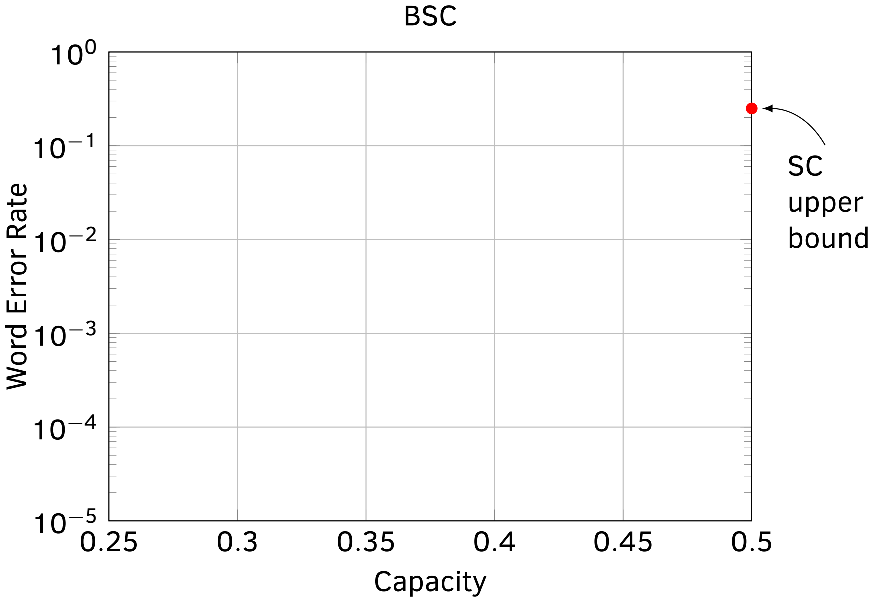
This talk:

An efficient implementation of SCL for universal polarization

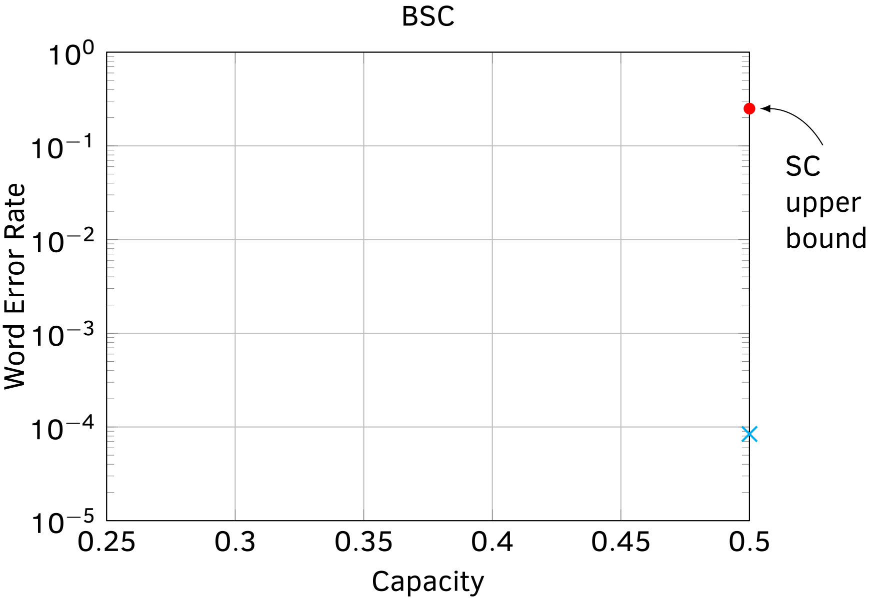
Setting

- Communication with uncertainty:
  - Encoder: Knows channel belongs to a set of channels
  - Decoder: Knows channel statistics (e.g., via estimation)
- Memory:
  - In channels
  - In input distribution
- Universal polar code achieves:
  - Vanishing error probability over set
  - Best rate (infimal information rate over set)

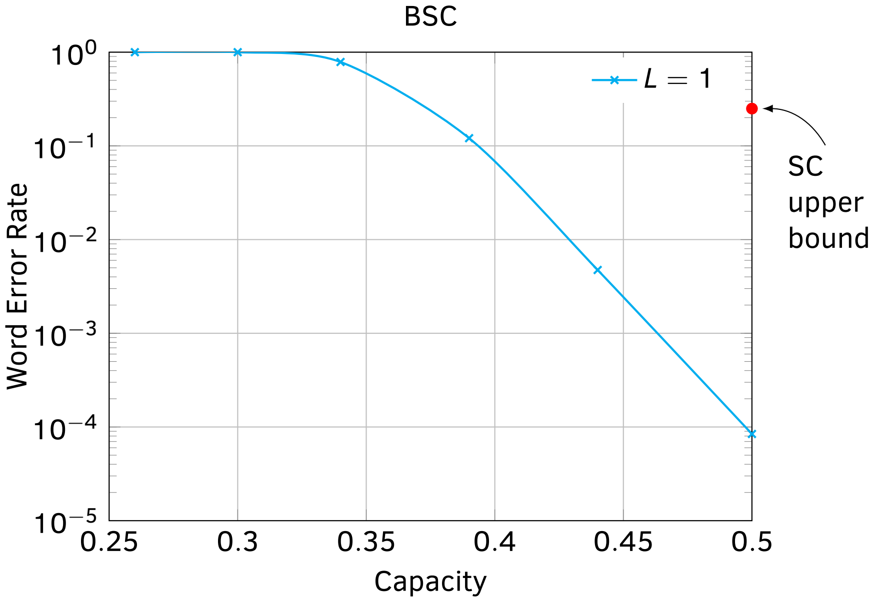
List decoding improves probability of error!



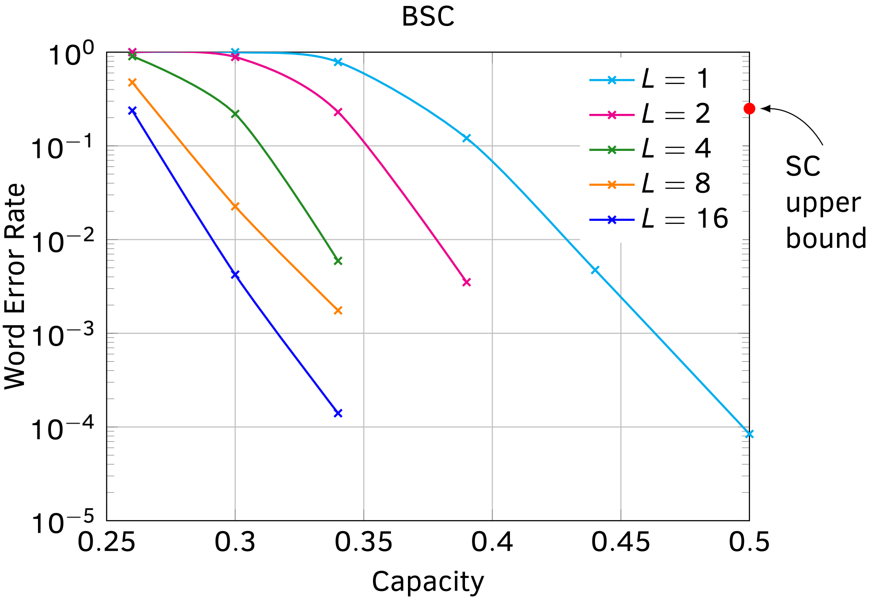
List decoding improves probability of error!



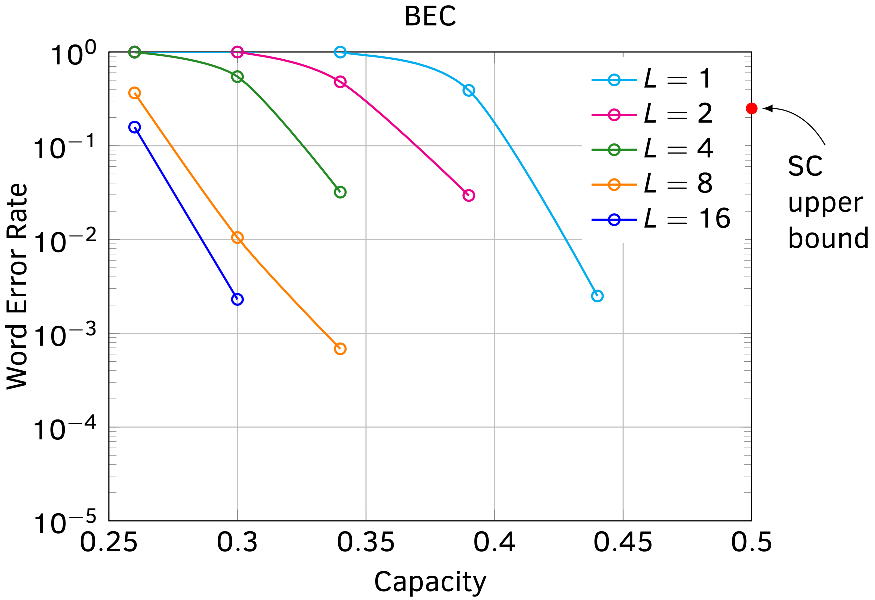
List decoding improves probability of error!



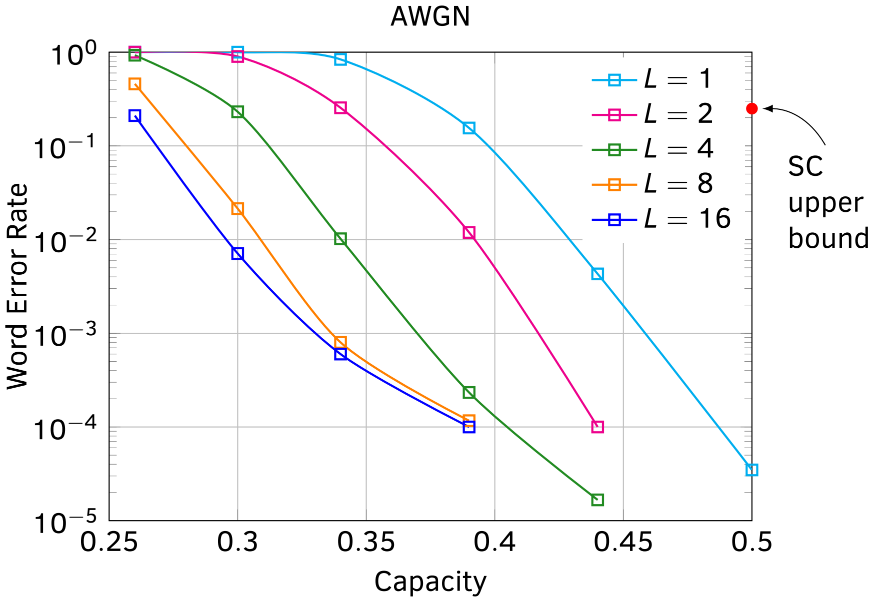
List decoding improves probability of error!



List decoding improves probability of error!



List decoding improves probability of error!



SCL in a nutshell

- Parameter: list size L
- Successively decode $u_0, u_1, \ldots, u_{N-1}$
- Decoding path: sequence of decoding decisions

$$\hat{\mathbf{u}}_{i-1} = \begin{bmatrix} \hat{u}_0 & \hat{u}_1 & \ldots & \hat{u}_{i-1} \end{bmatrix}$$

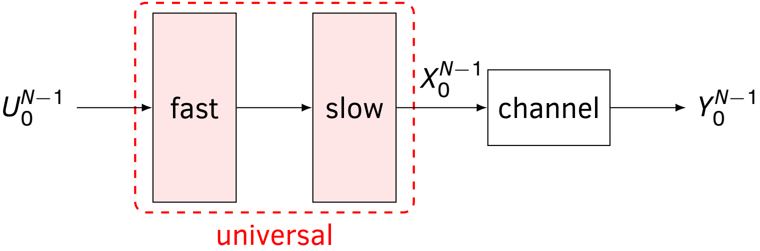
- If u_i is not frozen, split each decoding path into two paths,

$$\begin{tikzcd} & \begin{bmatrix} \hat{\mathbf{u}}_{i-1} & 0 \end{bmatrix} \\ \hat{\mathbf{u}}_{i-1} \arrow[ur] \arrow[dr] & \\ & \begin{bmatrix} \hat{\mathbf{u}}_{i-1} & 1 \end{bmatrix} \end{tikzcd}$$

- For each i, keep L best decoding paths
- Final decoding: choose best path

The Universal Construction

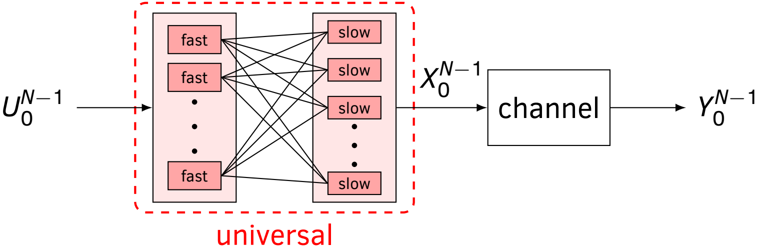
- Concatenation of two transform blocks



  - Slow — for universality
  - Fast (Arıkan) — for vanishing error probability

The Universal Construction

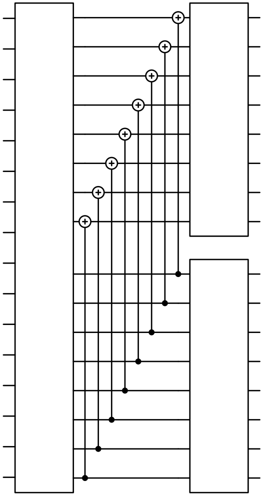
- Concatenation of two transform blocks



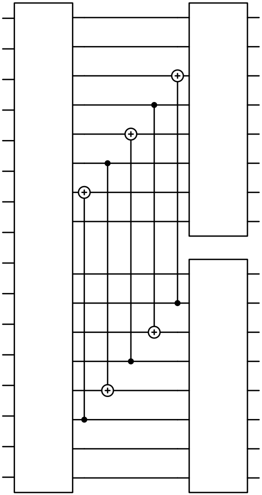
  - Slow — for universality
  - Fast (Arıkan) — for vanishing error probability
- Each block contains multiple copies of basic transforms
- Polar: certain indices of U_0^{N-1} are data bits,
  other indices "frozen" (Honda & Yamamoto 2013)
- Universal: data bit locations regardless of channel

Fast Transform vs. Slow Transform

Fast (Arıkan) transform

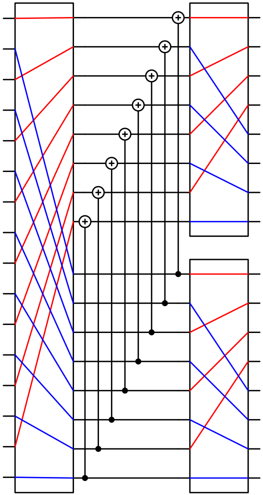


Slow transform

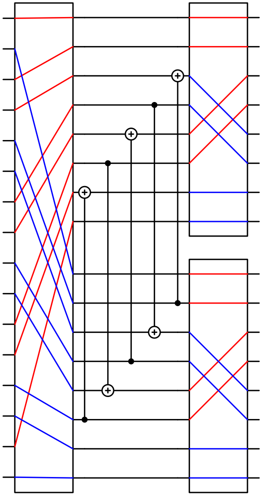


Fast Transform vs. Slow Transform

Fast (Arıkan) transform

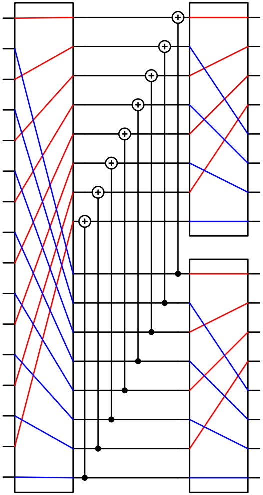


Slow transform

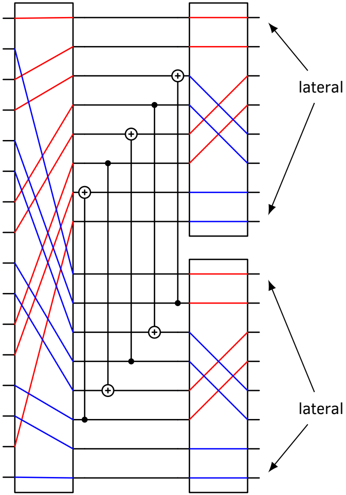


Fast Transform vs. Slow Transform

Fast (Arıkan) transform

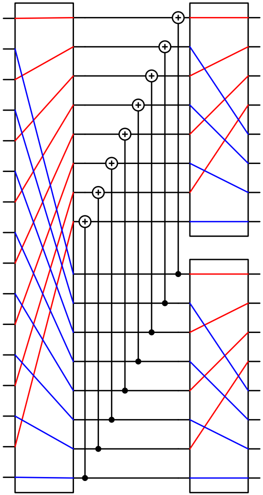


Slow transform

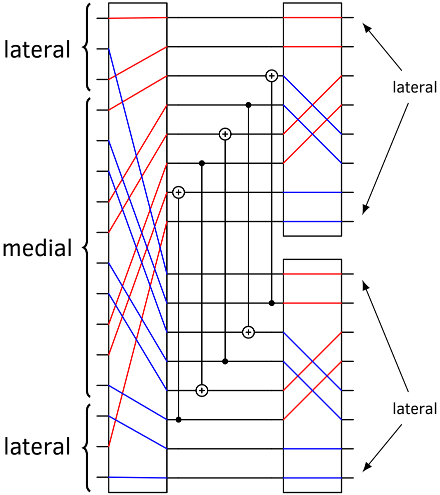


Fast Transform vs. Slow Transform

Fast (Arıkan) transform

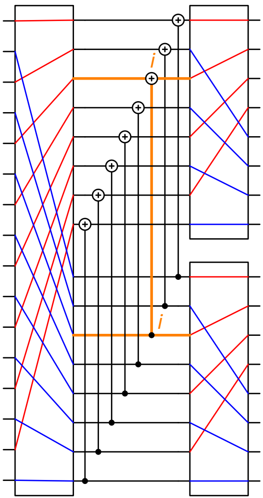


Slow transform

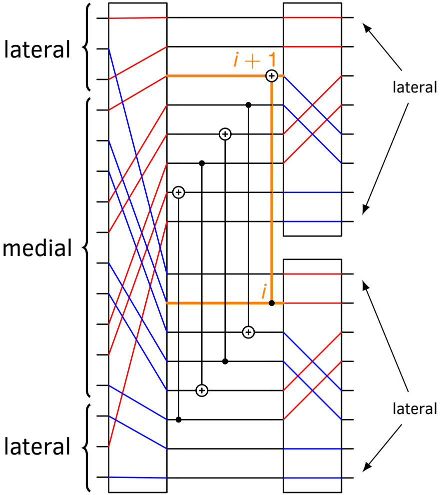


Fast Transform vs. Slow Transform

Fast (Arıkan) transform

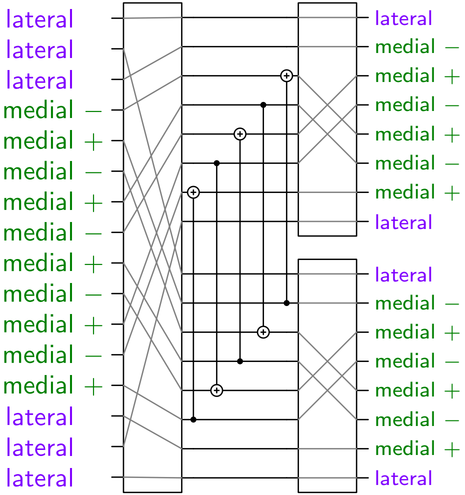


Slow transform



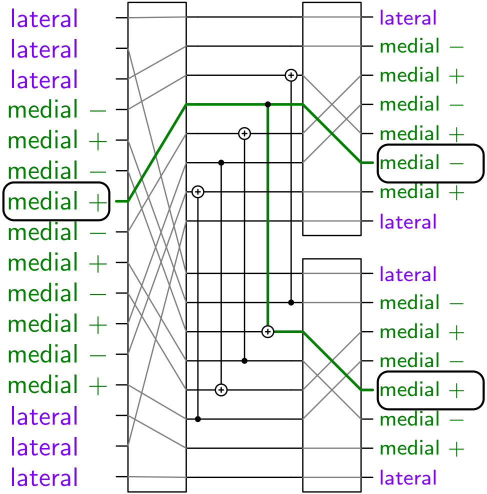
The Rules of Slow

- Any medial on left is a child of medial − and medial + on right



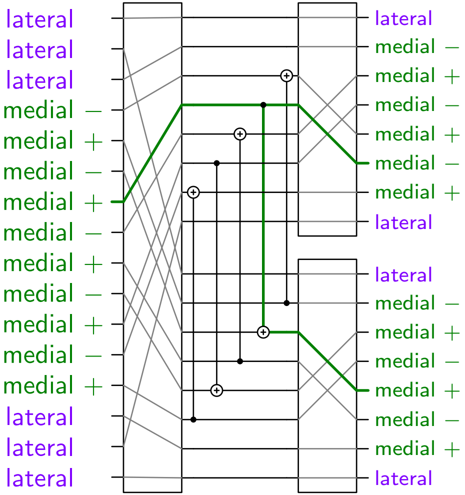
The Rules of Slow

- Any medial on left is a child of medial − and medial + on right



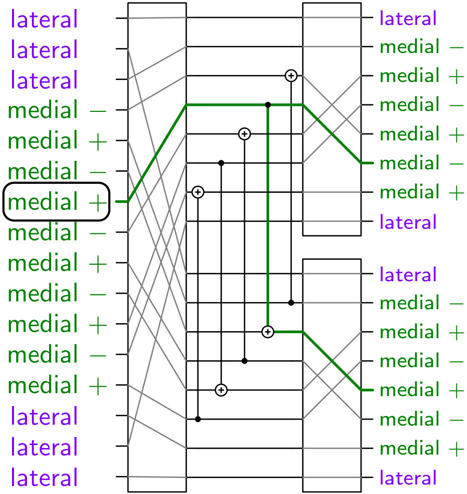
The Rules of Slow

- Any medial on left is a child of medial $-$ and medial $+$ on right
- Update of a medial $+$ on left updates both medial $-$ and medial $+$ on right



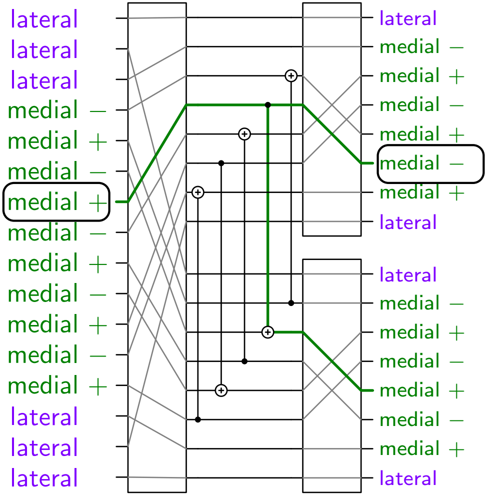
The Rules of Slow

- Any medial on left is a child of medial $-$ and medial $+$ on right
- Update of a medial $+$ on left updates both medial $-$ and medial $+$ on right



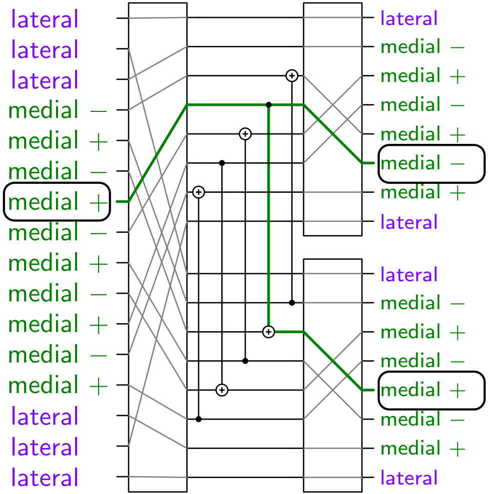
The Rules of Slow

- Any medial on left is a child of medial $-$ and medial $+$ on right
- Update of a medial $+$ on left updates both medial $-$ and medial $+$ on right



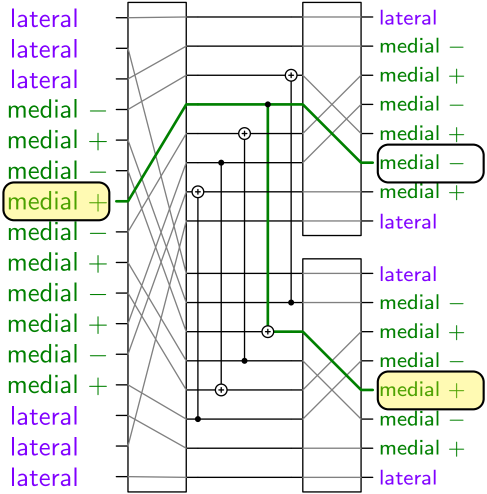
The Rules of Slow

- Any medial on left is a child of medial − and medial + on right
- Update of a medial + on left updates both medial − and medial + on right



The Rules of Slow

- Any medial on left is a child of medial − and medial + on right
- Update of a medial + on left updates both medial − and medial + on right



Corollary

Update of a medial + on left results in a cascade of updates, all the way to the channel on the far right

List Decoding — Fast Transform vs. Slow Transform

Fast Transform

The layer at depth λ gets updated once every 2^{λ} writes to the U vector

Deep layers get updated infrequently

Slow Transform

The layer at depth λ gets updated every other write to the U vector

All layers get updated frequently

List Decoding — Fast Transform vs. Slow Transform

Fast Transform

The layer at depth λ gets updated once every 2^{λ} writes to the U vector

Deep layers get updated infrequently

Good news for sharing data between list decoding paths

Slow Transform

The layer at depth λ gets updated every other write to the U vector

All layers get updated frequently

Bad news for sharing data between list decoding paths

Is All Hope Lost?

Fast Transform

- Layer at depth λ is updated once every 2^λ writes
- At layer of depth λ, 2^λ indices are updated at once

Slow Transform

- Layer at depth λ is updated at every other write
- At layer of depth λ, at most two indices are updated

Is All Hope Lost?

Fast Transform

- Layer at depth λ is updated once every 2^λ writes
- At layer of depth λ, 2^λ indices are updated at once

Slow Transform

- Layer at depth λ is updated at every other write
- At layer of depth λ, at most two indices are updated

Our Saving Grace

The updates are cyclic!

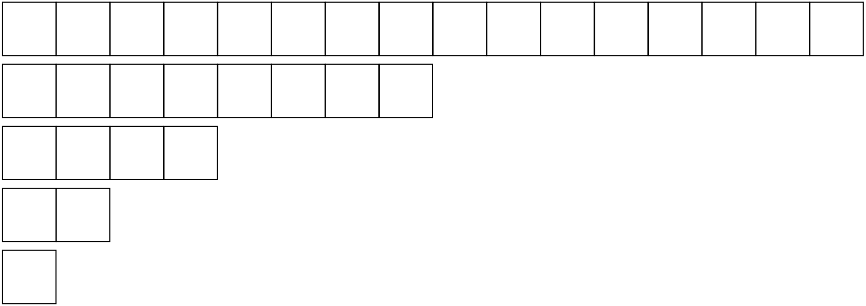
A Bespoke Data Structure

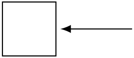
- Aim: mimic the fast transform's update regime
  - Frequent small updates
  - Infrequent large updates
- Bespoke data structure: Cyclic Exponential Array
- Main idea: an array comprised of sub-arrays, growing exponentially in length

Example

entire array:

cyclic exponential array:



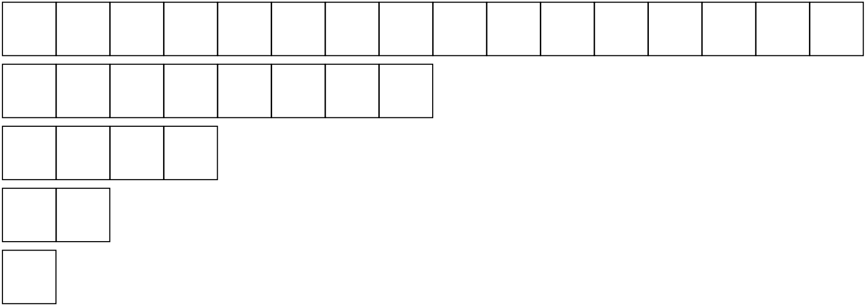


last written value

Example

entire array:

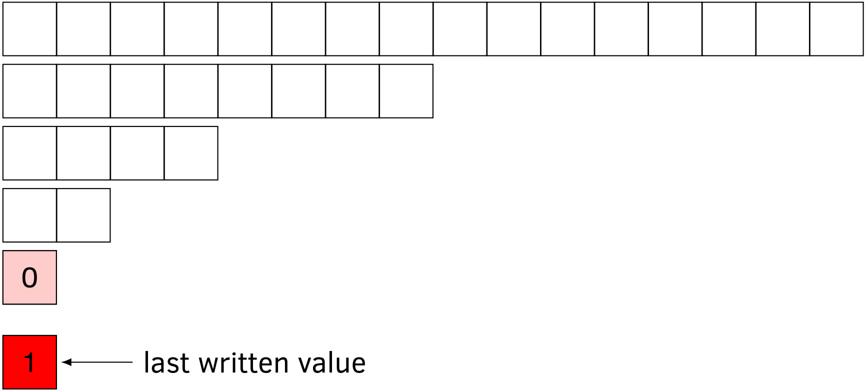
cyclic exponential array:



Example

entire array:

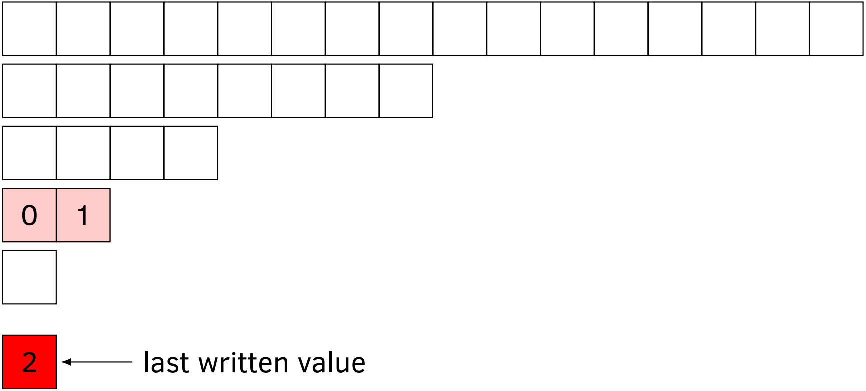
cyclic exponential array:



Example

entire array:

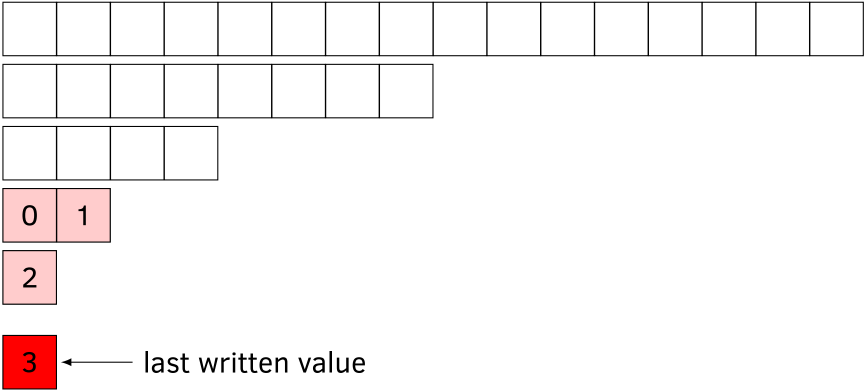
cyclic exponential array:



Example

entire array:

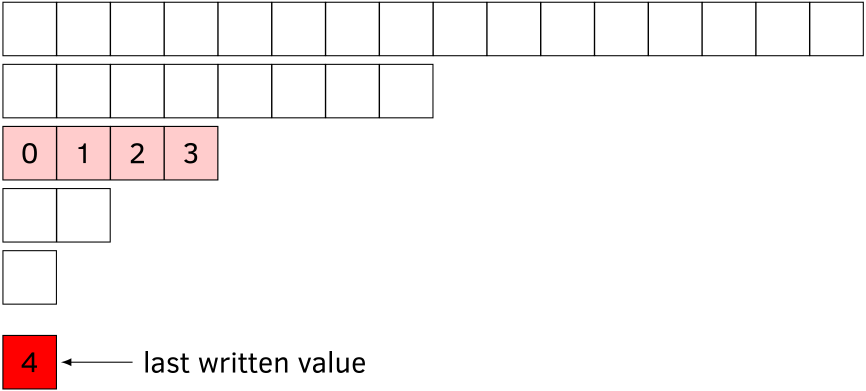
cyclic exponential array:



Example

entire array:

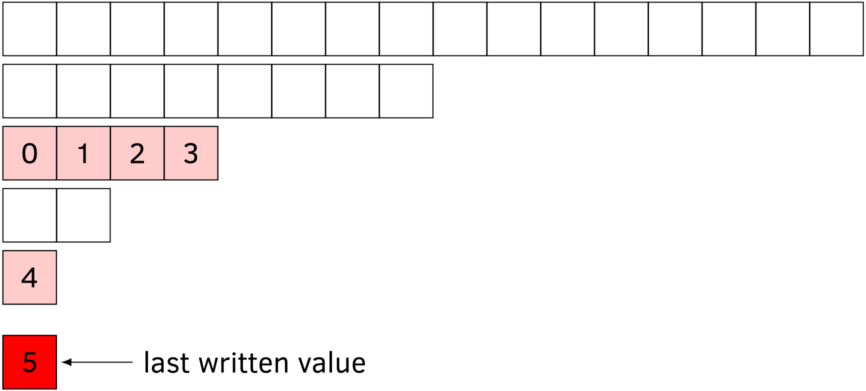
cyclic exponential array:



Example

entire array:

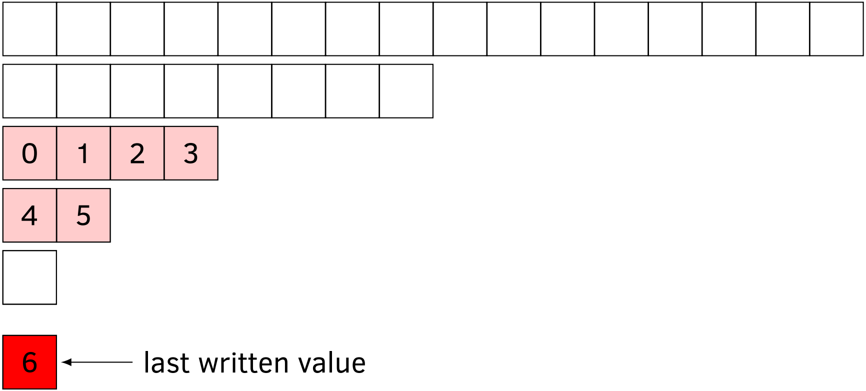
cyclic exponential array:



Example

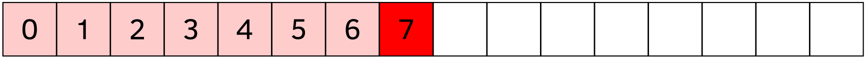
entire array:

cyclic exponential array:

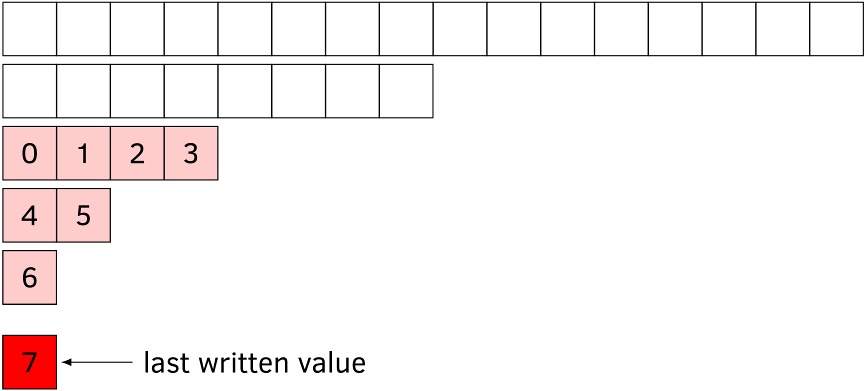


Example

entire array:

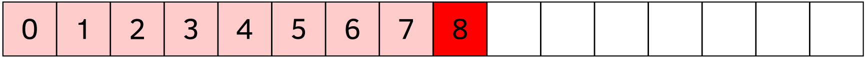


cyclic exponential array:

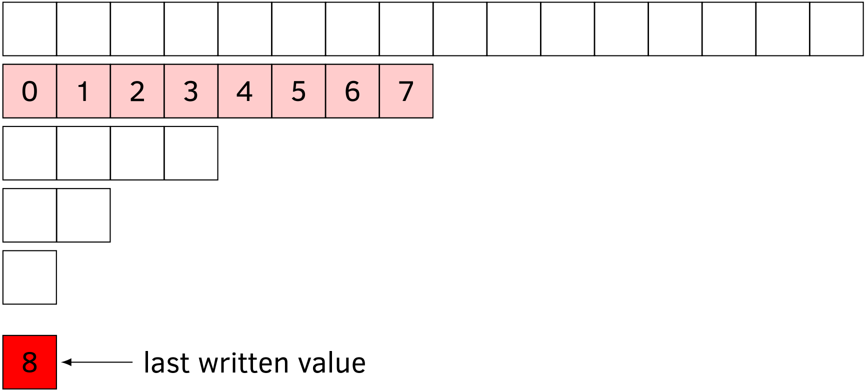


Example

entire array:

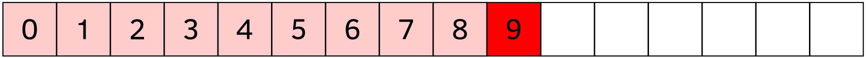


cyclic exponential array:

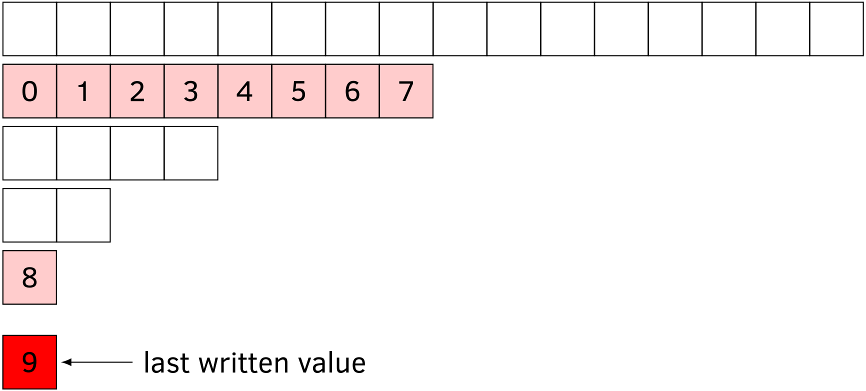


Example

entire array:

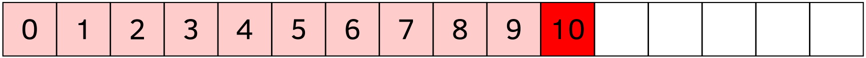


cyclic exponential array:

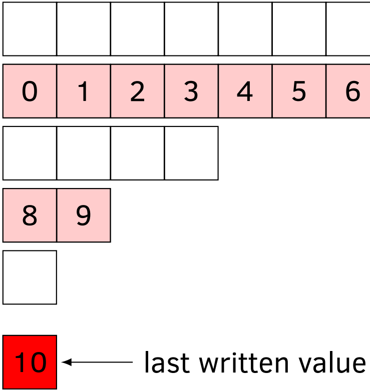


Example

entire array:

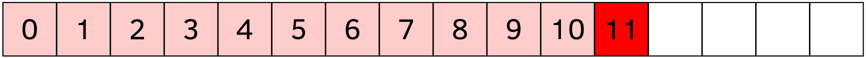


cyclic exponential array:

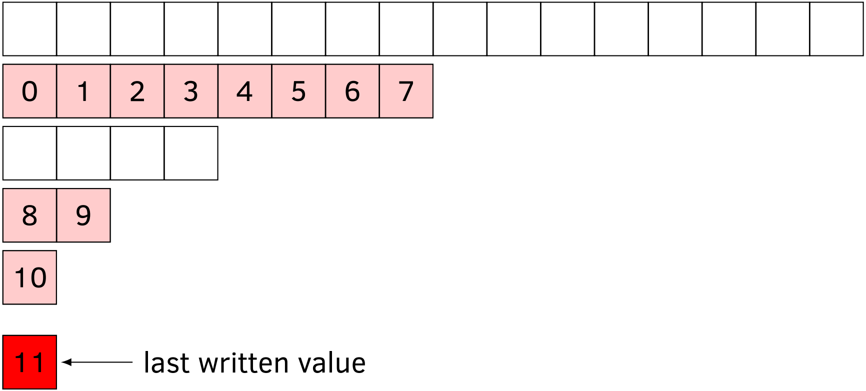


Example

entire array:

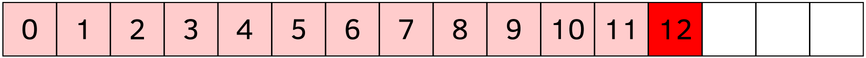


cyclic exponential array:

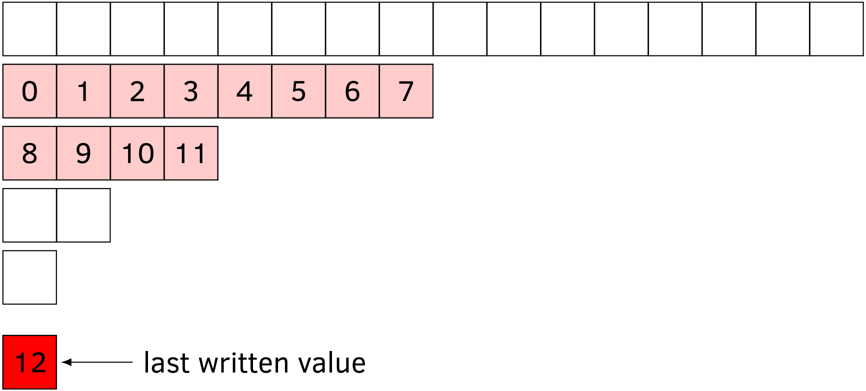


Example

entire array:

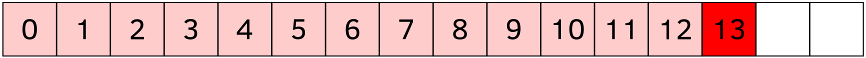


cyclic exponential array:

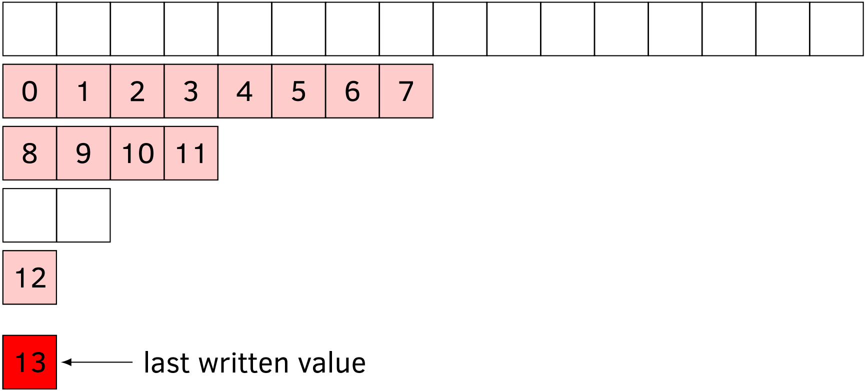


Example

entire array:

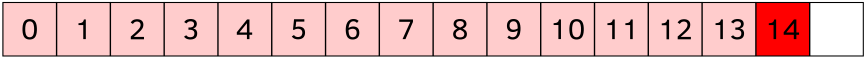


cyclic exponential array:

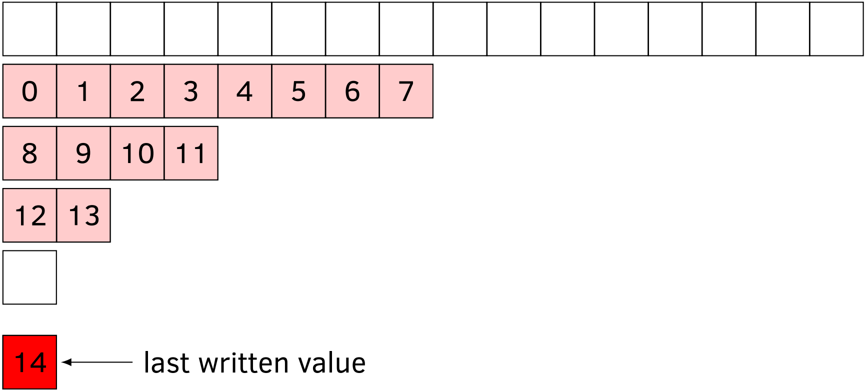


Example

entire array:

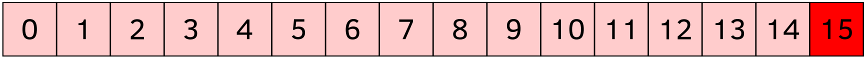


cyclic exponential array:

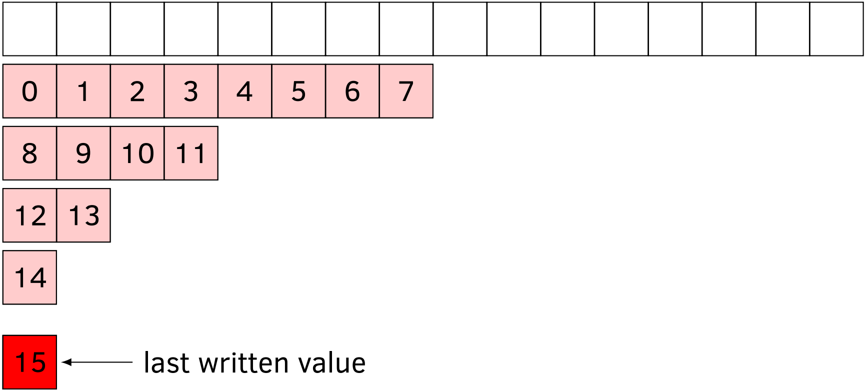


Example

entire array:

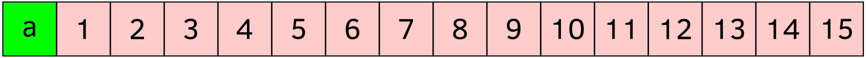


cyclic exponential array:

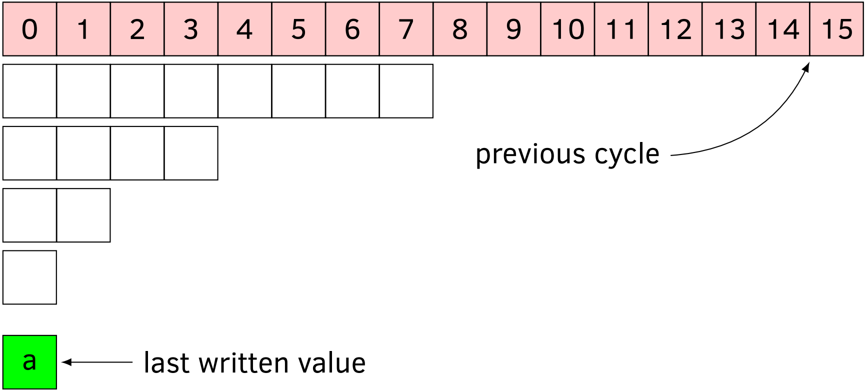


Example

entire array:



cyclic exponential array:

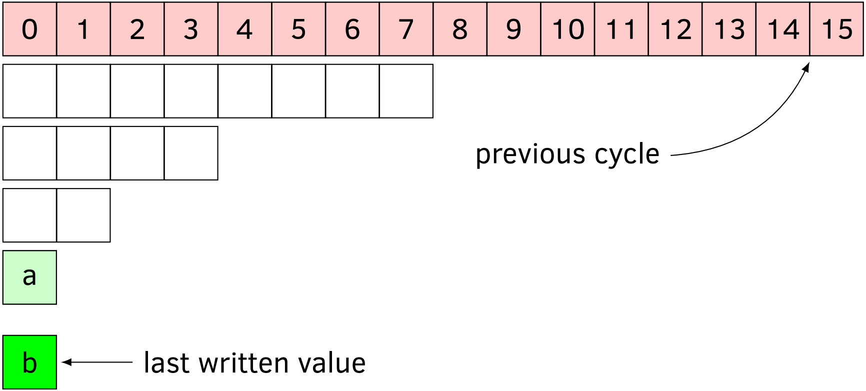


Example

entire array:

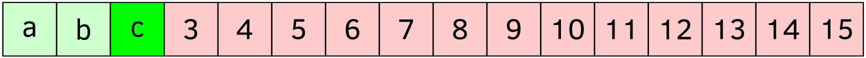
a	b	2	3	4	5	6	7	8	9	10	11	12	13	14	15

cyclic exponential array:

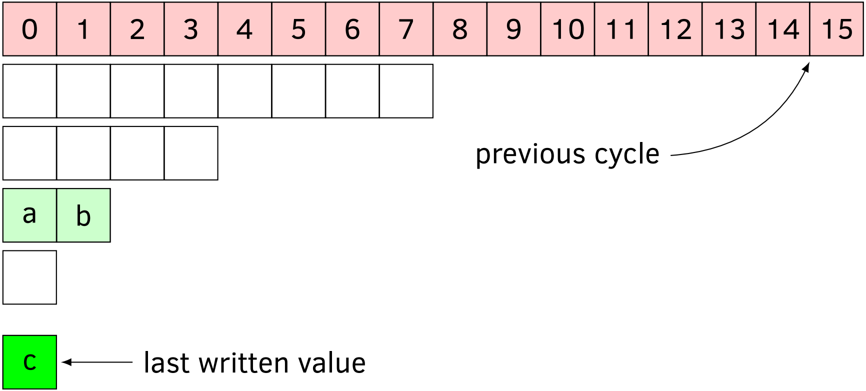


Example

entire array:

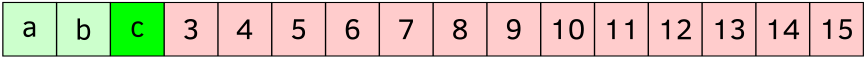


cyclic exponential array:

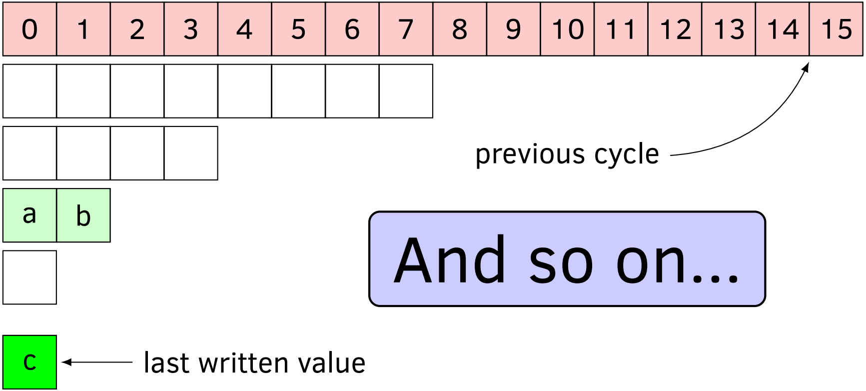


Example

entire array:



cyclic exponential array:



Theorem

Let:

- Blocklength N
- Channel with or without memory (S states)
- List size L

Theorem

*Our SCL decoder for **universal polar codes** has:*

- *running time:* $O(L \cdot S^3 \cdot N \log N)$
- *space complexity:* $O(L \cdot S^2 \cdot N)$

We made the deadline, thanks to

- Computing resources (cores galore)
  - Amir Baer
  - Goel Samuel
- And we parallelized using the excellent
  - GNU Parallel by Ole Tange[1]
- Origami artwork by Bernie Peyton

[1] O. Tange (2018): GNU Parallel 2018, March 2018, https://doi.org/10.5281/zenodo.1146014

Something to look forward to

Look out for our code on github!

For the impatient: email us at polarbear@technion.ac.il