Linear time list recovery via expander codes

Brett Hemenway and Mary Wootters

June 17 2016

Outline

Our Results

One slide version

- Inner code + expander graph $\Rightarrow$ Expander code

Our Results

One slide version

- Inner code + expander graph $\Rightarrow$ Expander code
- [SS96, Spi96, Zem01, BZ02, BZ05, BZ06]

Inner code has decent distance $\Rightarrow$ So does the expander code and it's decodable in linear time!

Our Results

One slide version

- Inner code + expander graph $\Rightarrow$ Expander code
- [SS96, Spi96, Zem01, BZ02, BZ05, BZ06]

Inner code has decent distance	$\Rightarrow$	So does the expander code and it's decodable in linear time!

- [HOW13]

Inner code has decent locality	$\Rightarrow$	So does the expander code and it's locally decodable in sub-linear time!

Our Results

One slide version

- Inner code + expander graph $\Rightarrow$ Expander code
- [SS96, Spi96, Zem01, BZ02, BZ05, BZ06]

Inner code has decent distance	$\Rightarrow$	So does the expander code and it's decodable in linear time!

- [HOW13]

Inner code has decent locality	$\Rightarrow$	So does the expander code and it's locally decodable in sub-linear time!

- Moral?

Inner code has decent _____	$\Rightarrow$	So does the expander code and there's an efficient algorithm!

Our Results

One slide version

- Inner code + expander graph $\Rightarrow$ Expander code
- [SS96, Spi96, Zem01, BZ02, BZ05, BZ06]

Inner code has decent distance	$\Rightarrow$	So does the expander code and it's decodable in linear time!

- [HOW13]

Inner code has decent locality	$\Rightarrow$	So does the expander code and it's locally decodable in sub-linear time!

- Moral?

Inner code has decent _____	$\Rightarrow$	So does the expander code and there's an efficient algorithm!

- This work:

Inner code has decent list-recoverability	$\Rightarrow$	So does the expander code and it's list-recoverable in linear time!

Our Results

~~One~~ Two slide version

Inner code has decent list-recoverability $\Rightarrow$ So does the expander code and it's list-recoverable in linear time!

- List-recoverable: decodable from *uncertainty* (instead of errors).

Our Results

~~One~~ Two slide version

Inner code has decent list-recoverability $\Rightarrow$ So does the expander code and it's list-recoverable in linear time!

- List-recoverable: decodable from *uncertainty* (instead of errors).
- Known linear-time list-recoverable codes have rate $< 1/2$.

Our Results

~~One~~ Two slide version

Inner code has decent list-recoverability $\Rightarrow$ So does the expander code and it's list-recoverable in linear time!

- List-recoverable: decodable from *uncertainty* (instead of errors).
- Known linear-time list-recoverable codes have rate $< 1/2$.
- Expander codes can have rate $1 - \varepsilon$!

Our Results

~~One~~ Two slide version

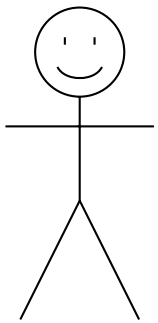
Inner code has decent list-recoverability $\Rightarrow$ So does the expander code and it's list-recoverable in linear time!

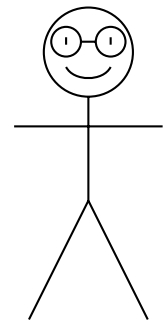
- List-recoverable: decodable from *uncertainty* (instead of errors).
- Known linear-time list-recoverable codes have rate $< 1/2$.
- Expander codes can have rate $1 - \varepsilon$!

- We can plug our codes into [Meir'14] and get the optimal* rate/error trade-off, for any rate.

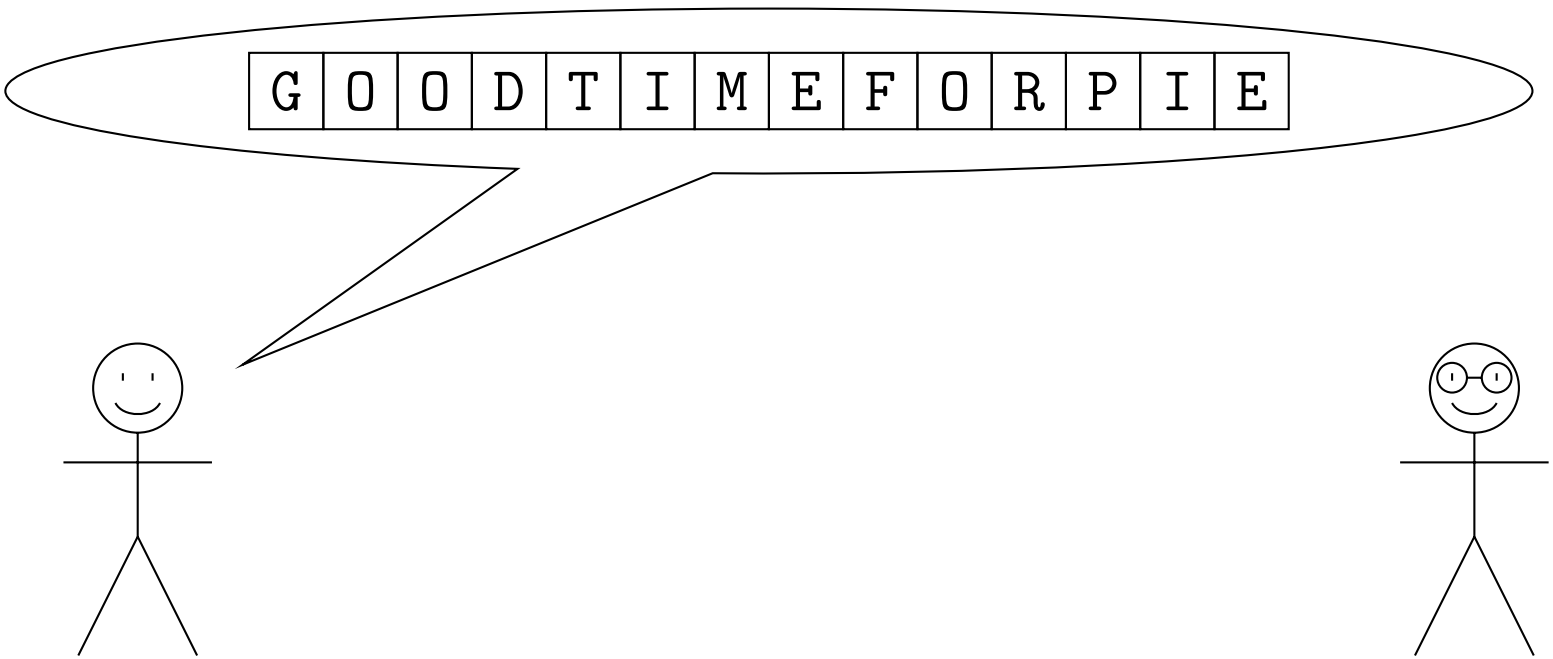
Outline

List Decoding

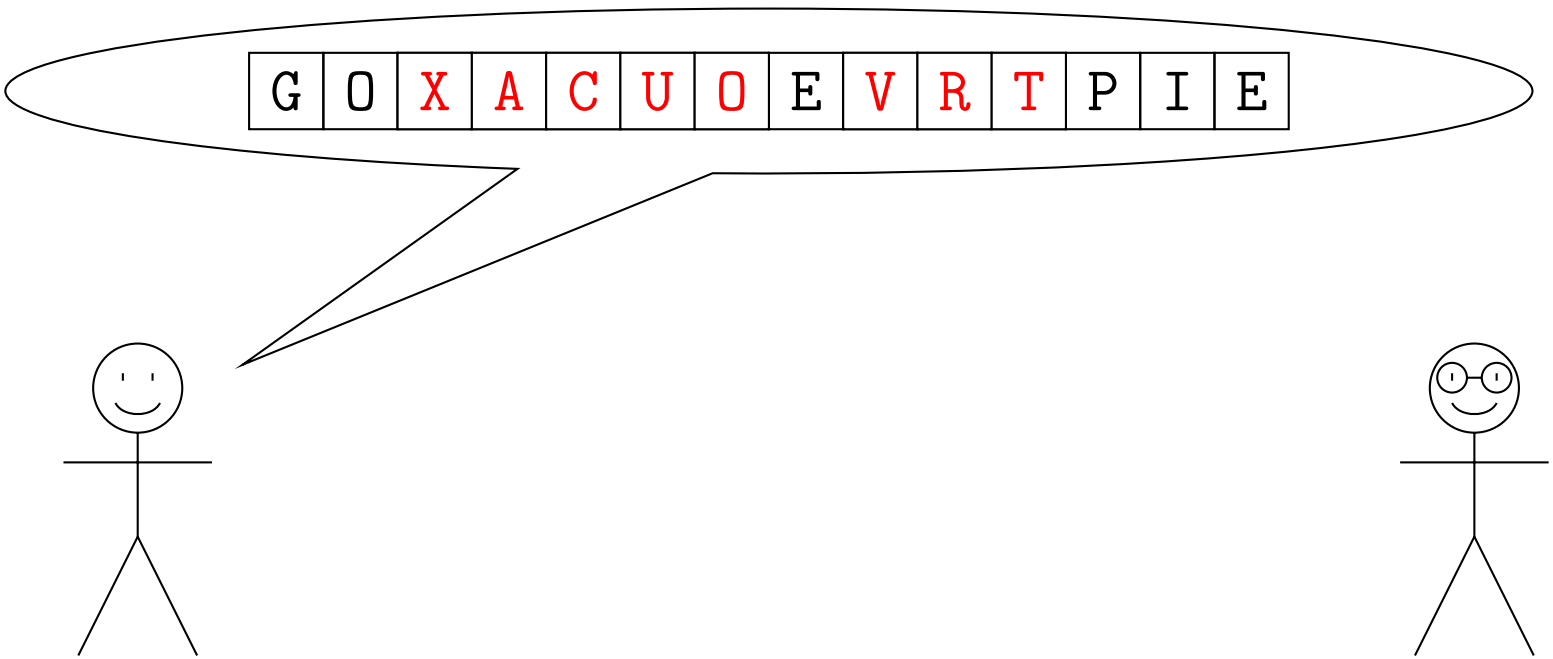




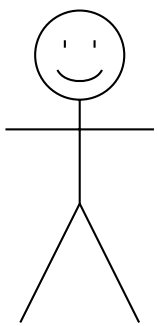
List Decoding

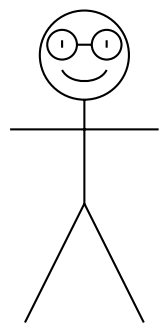


List Decoding

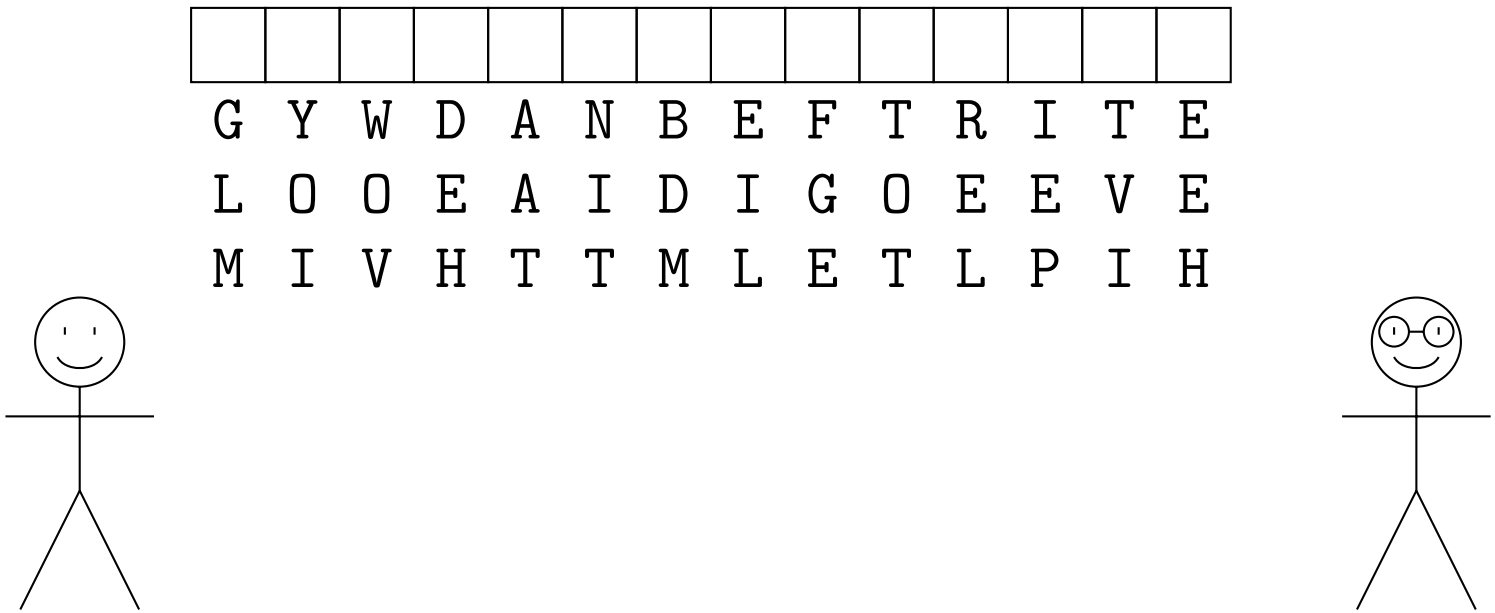


List Decoding

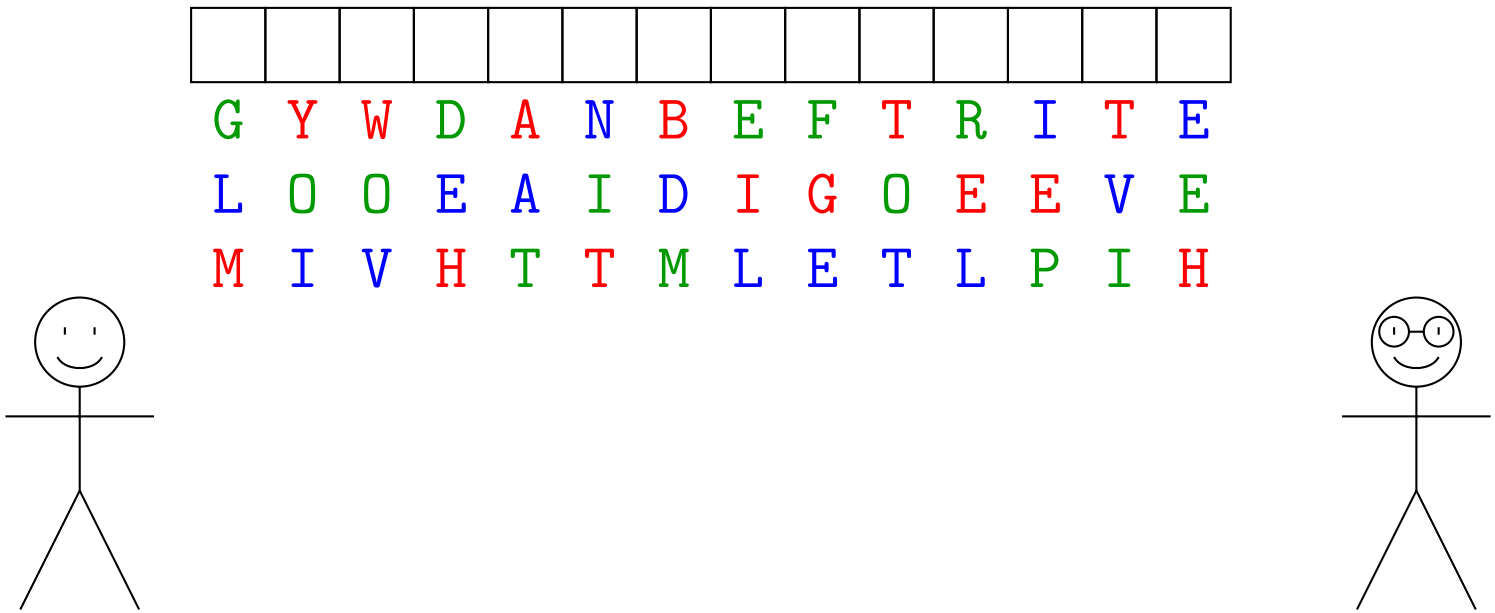




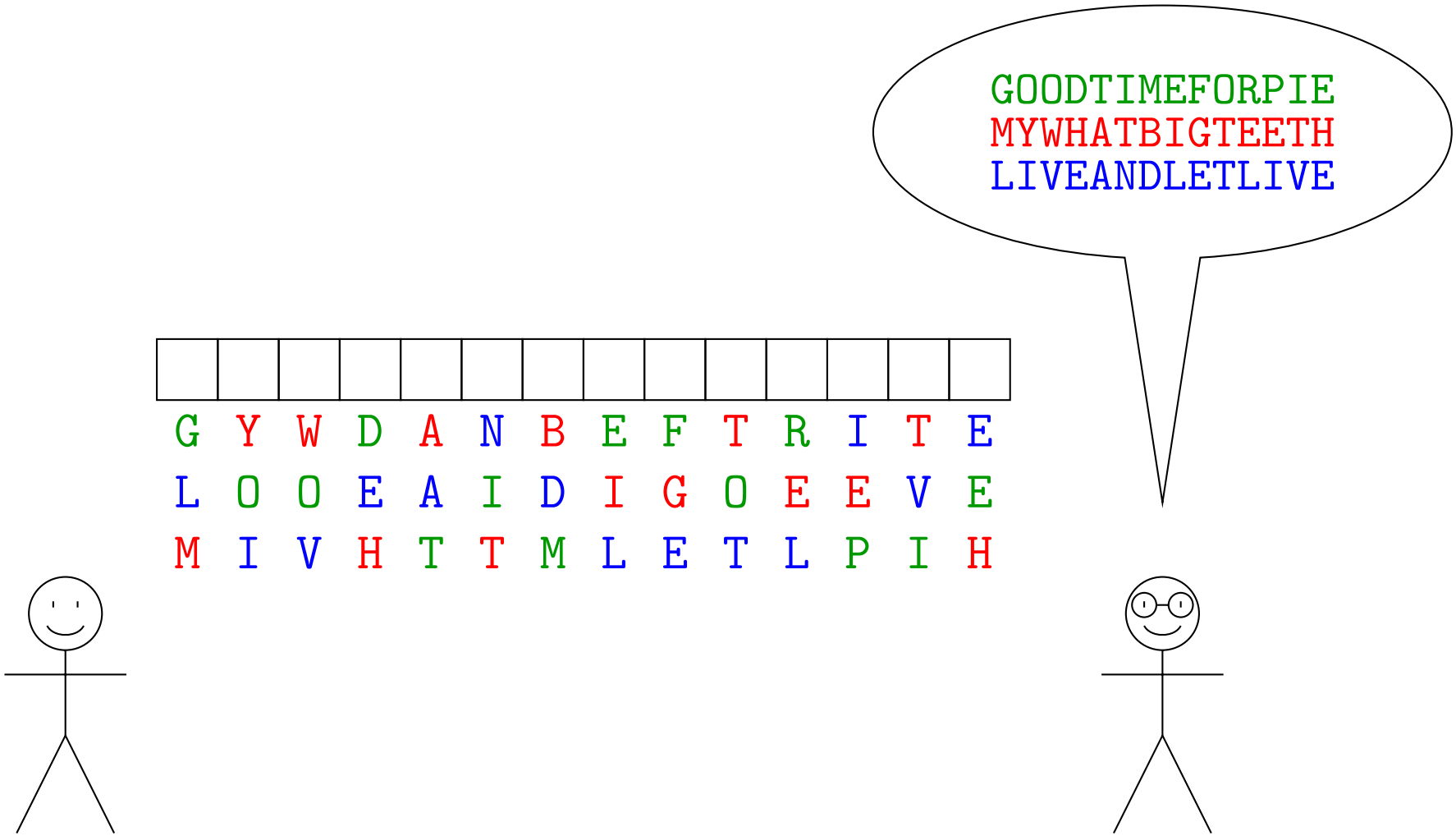
List Recovery



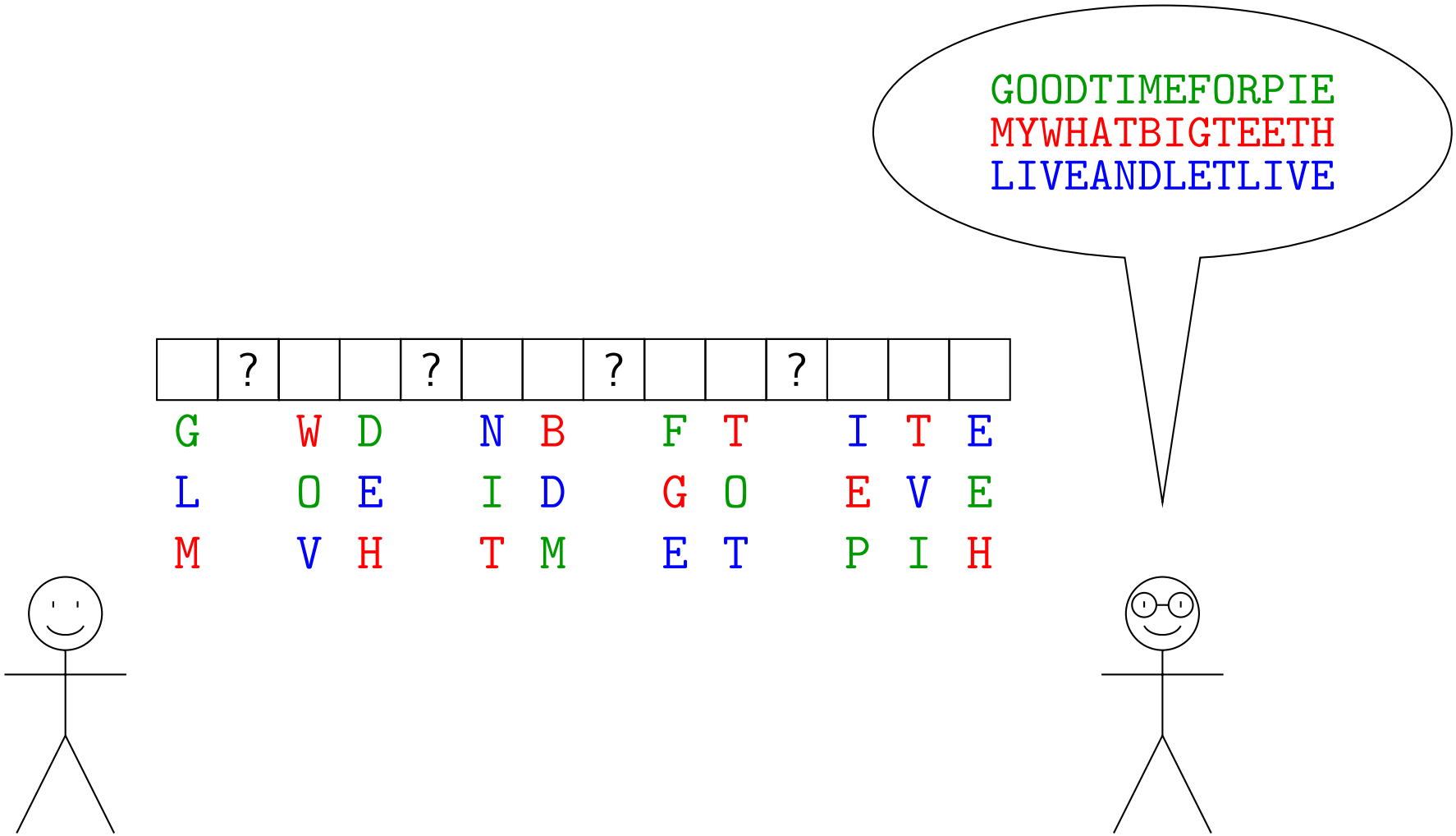
List Recovery



List Recovery



List Recovery From Erasures



List Recovery

Definition

Definition

$\mathcal{C} \in \Sigma^N$ is (α, ℓ, L)-list-recoverable (from erasures) if:

- for any set of lists $S_1, \ldots, S_N$
  so that at least αN of them have size $\leq \ell$,
- there are at most L codewords $c \in \mathcal{C}$ so that $c_i \in S_i$ for all i.

List Recovery

Definition

Definition

$\mathcal{C} \in \Sigma^N$ is (α, ℓ, L)-list-recoverable (from erasures) if:

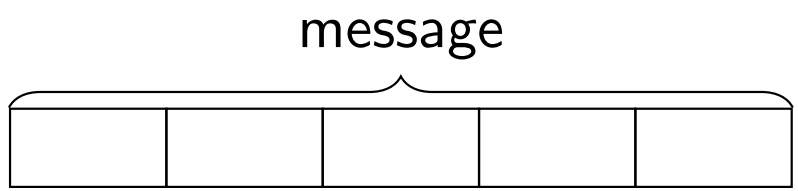
- for any set of lists $S_1, \ldots, S_N$
  so that at least αN of them have size $\leq \ell$,
- there are at most L codewords $c \in \mathcal{C}$ so that $c_i \in S_i$ for all i.

- $\alpha \longleftarrow$ fraction of codeword not erased
- $\ell \longleftarrow$ number of symbols in each slot
- $L \longleftarrow$ number of codewords that match

Note: $L \geq \ell$.

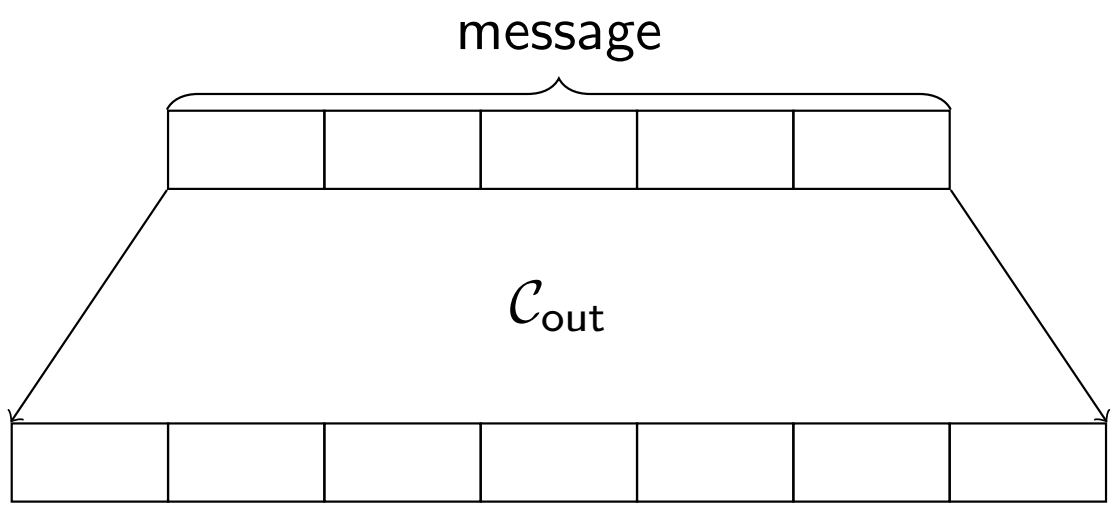
List decoding concatenated codes

An application of list recovery



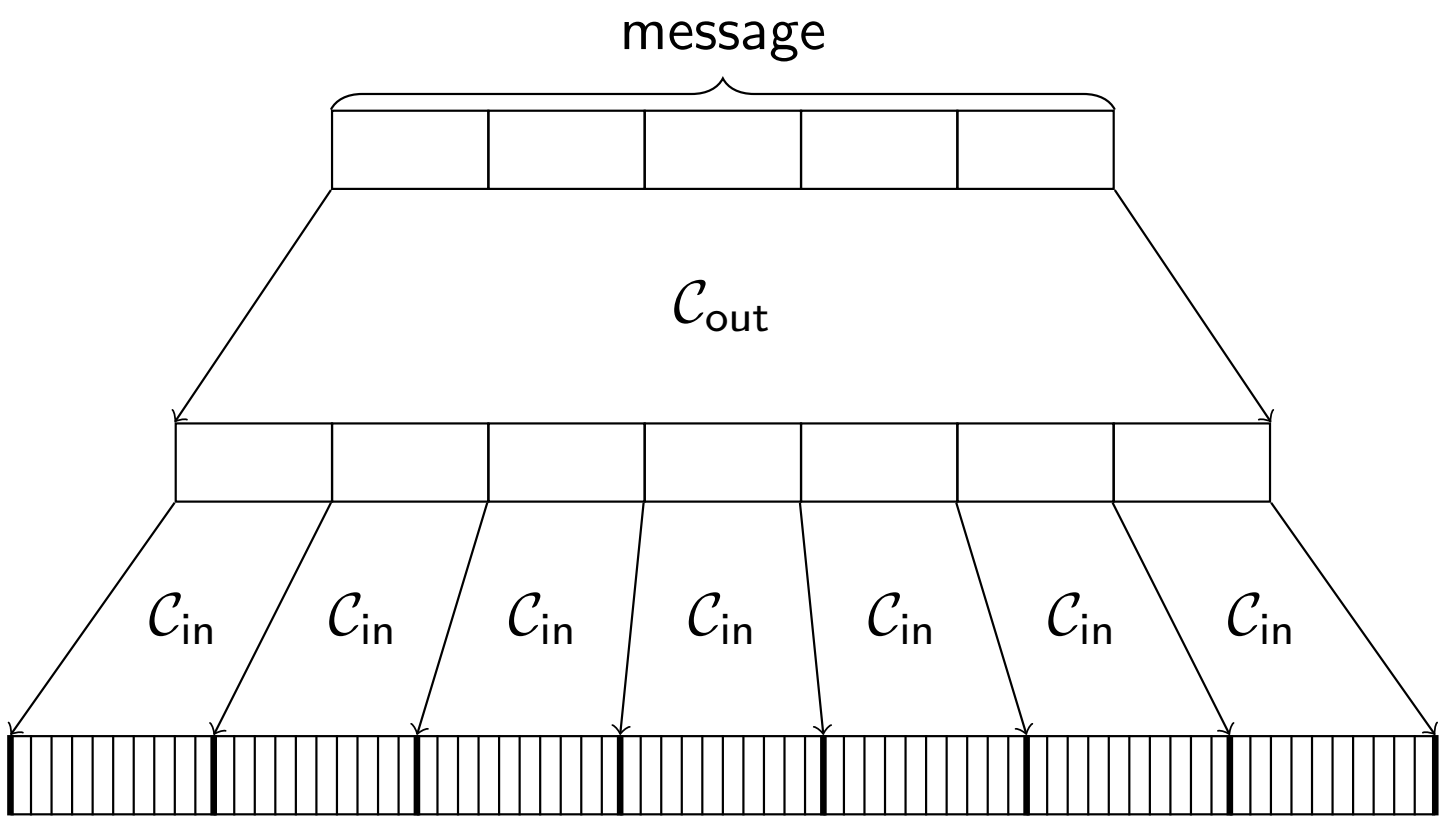
List decoding concatenated codes

An application of list recovery



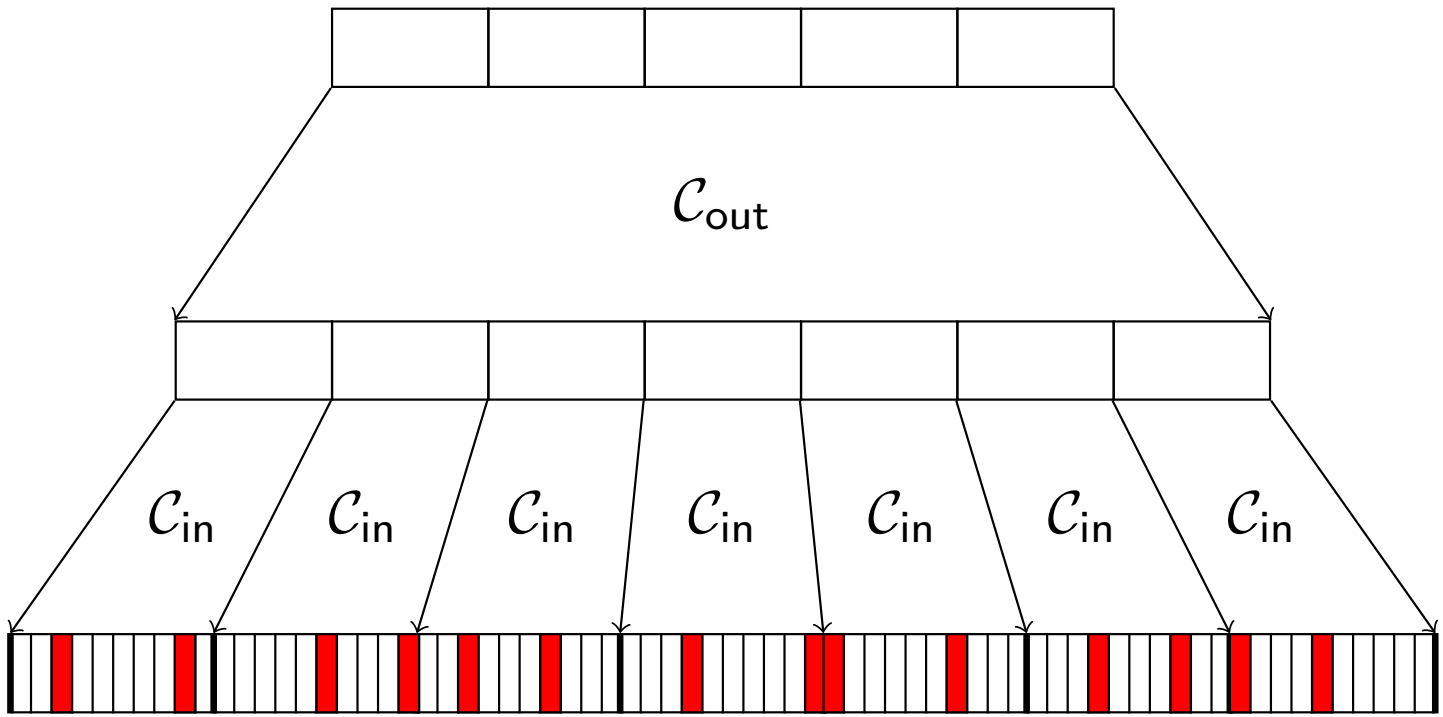
List decoding concatenated codes

An application of list recovery



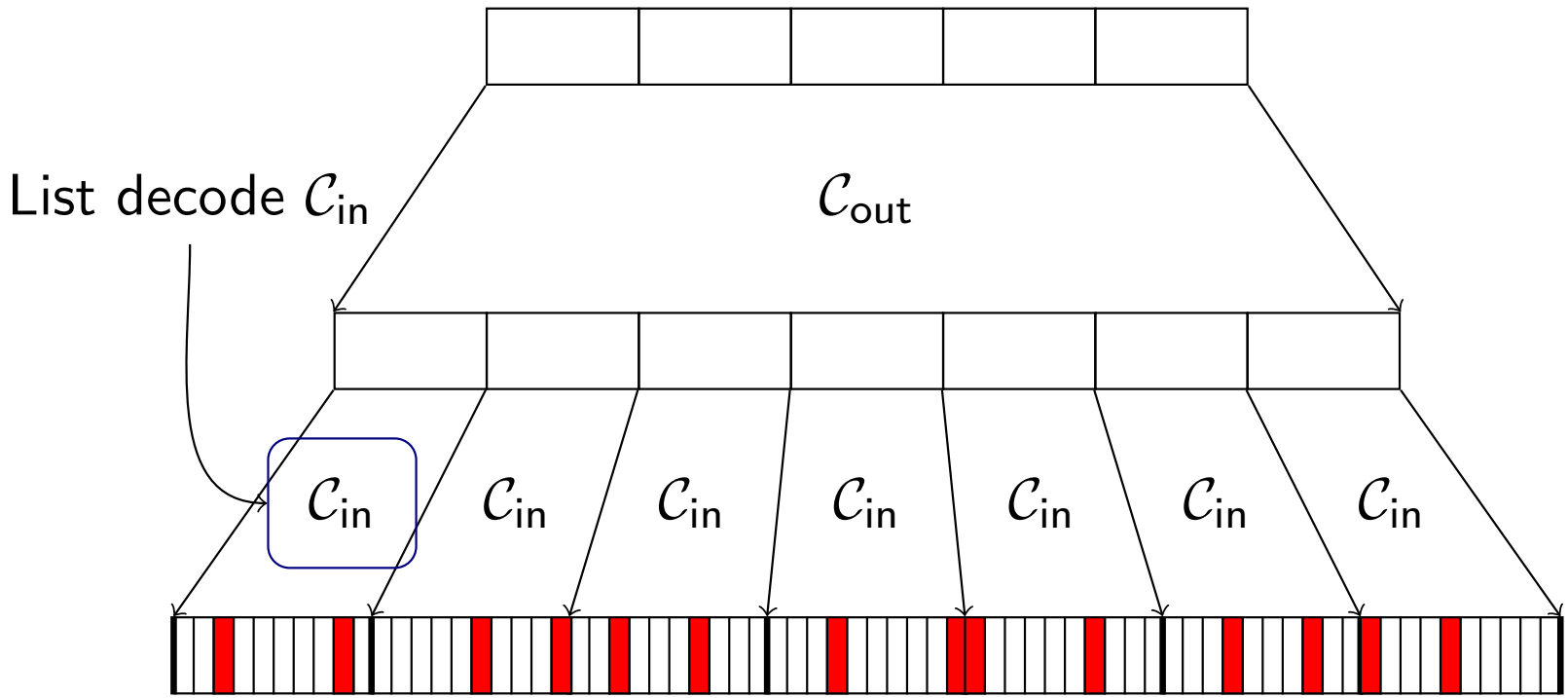
List decoding concatenated codes

An application of list recovery



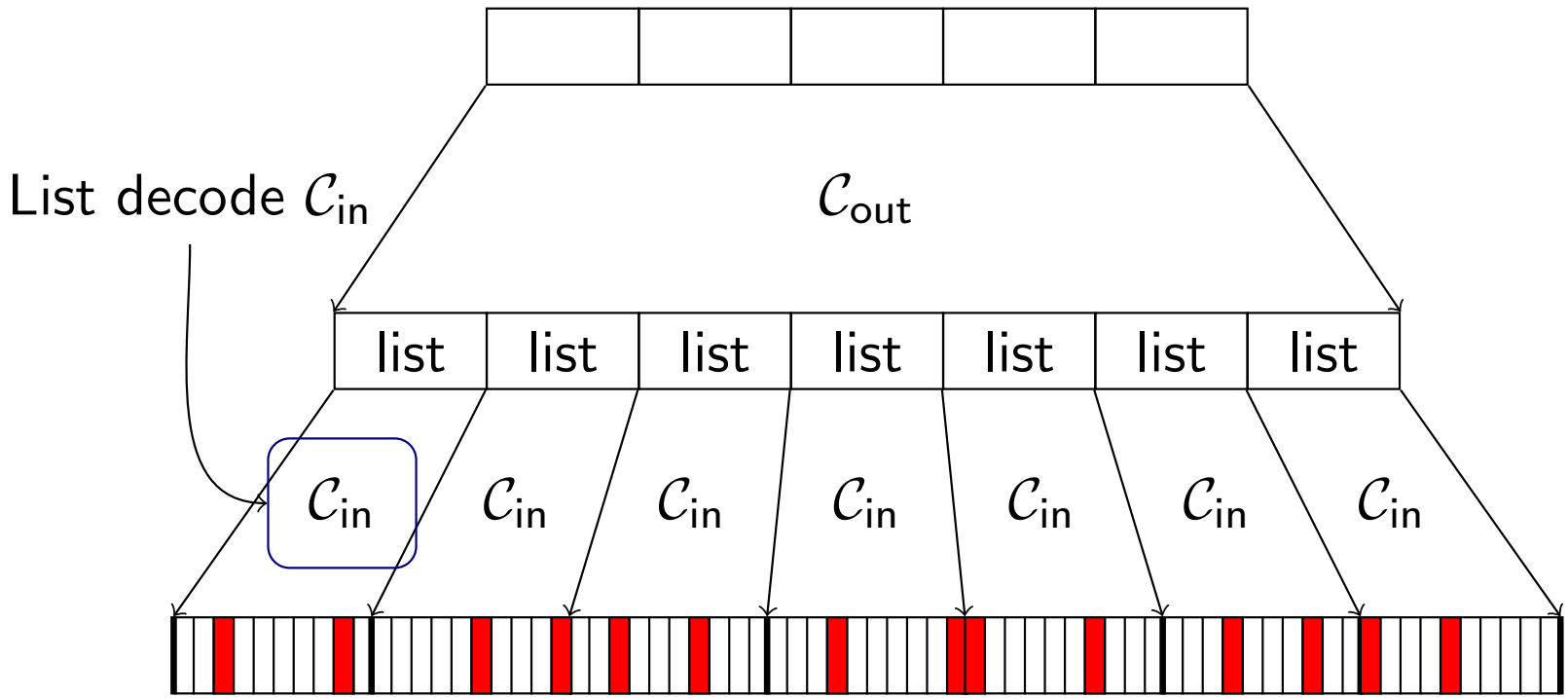
List decoding concatenated codes

An application of list recovery



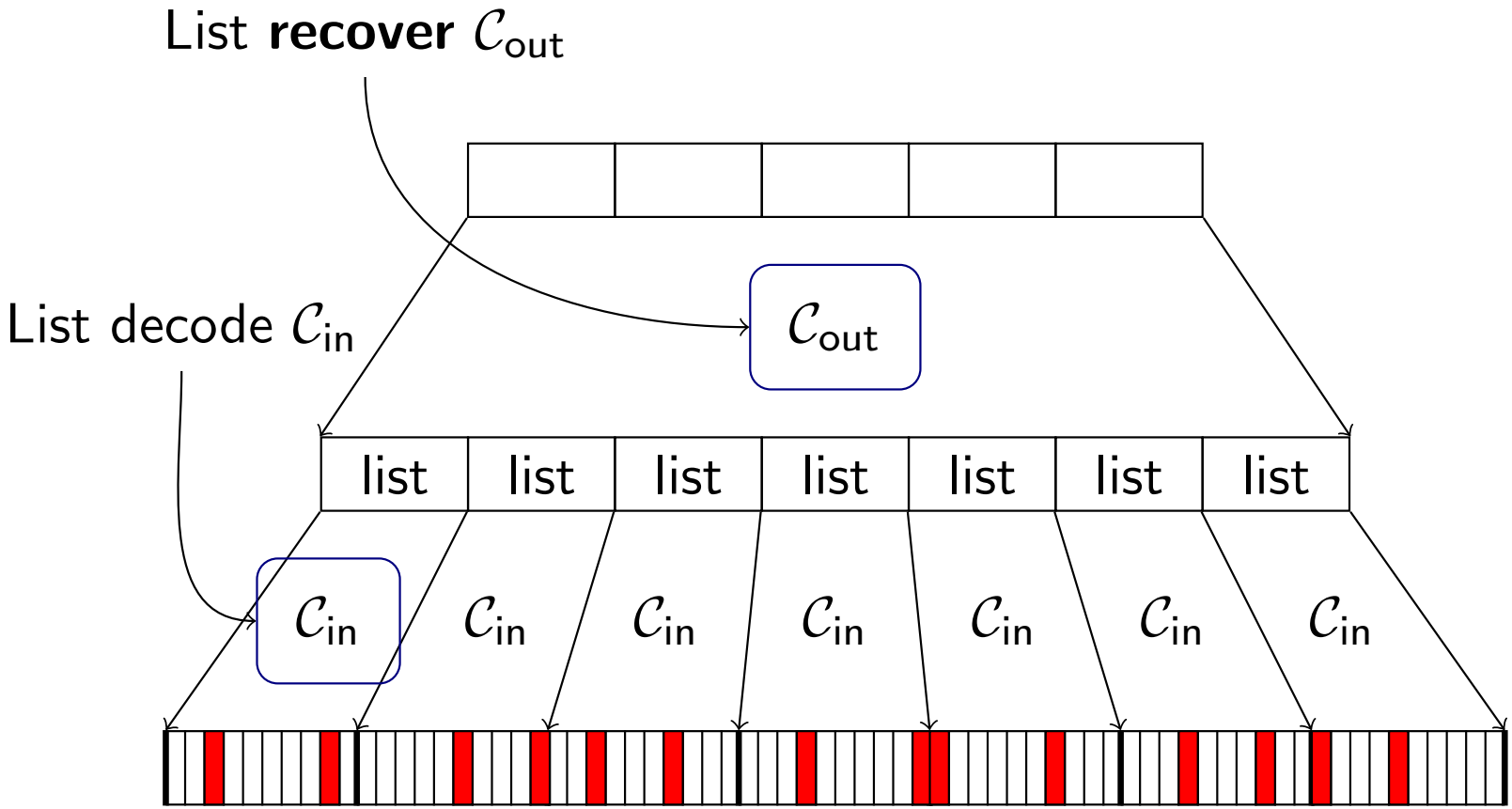
List decoding concatenated codes

An application of list recovery



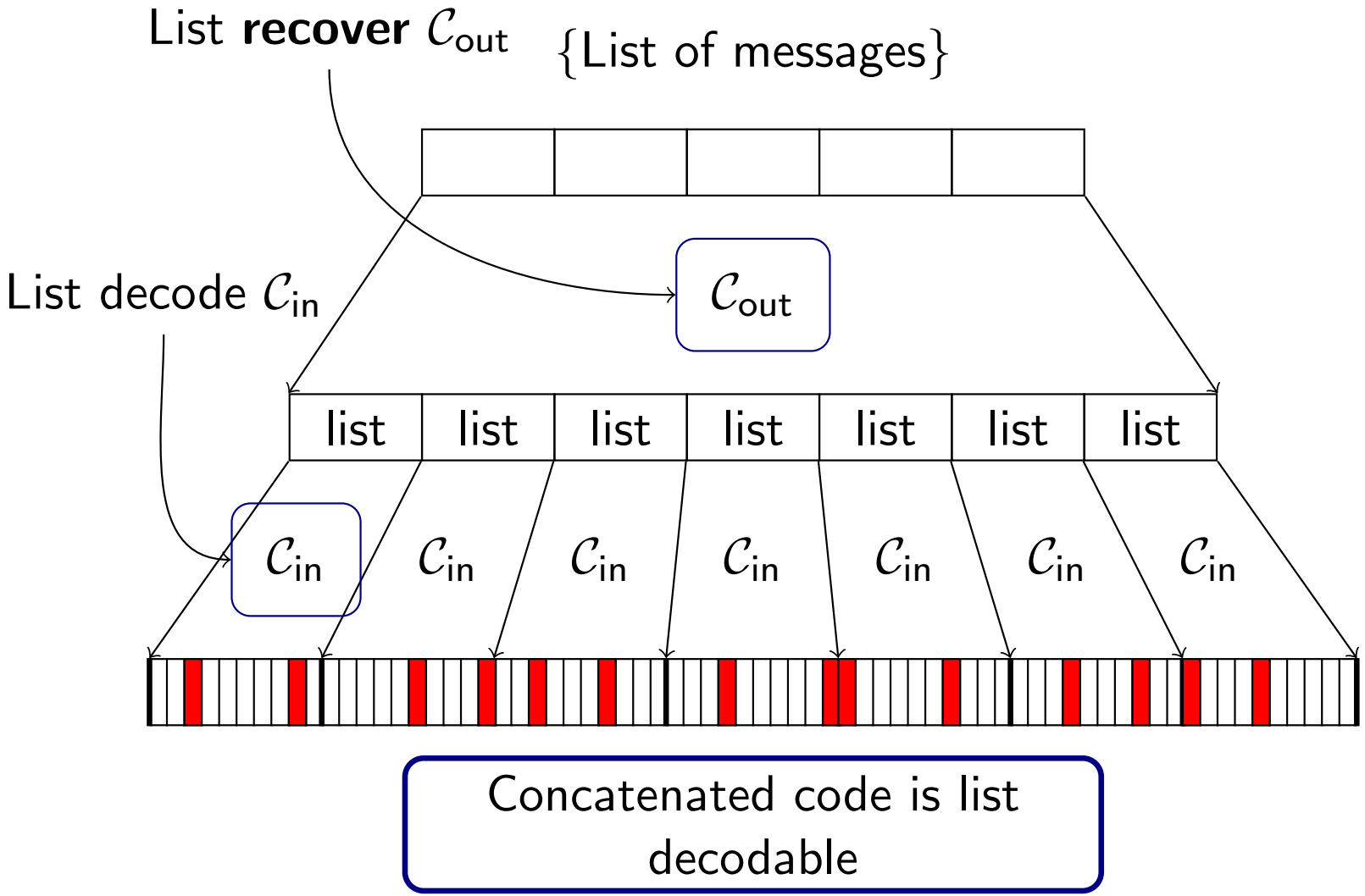
List decoding concatenated codes

An application of list recovery



List decoding concatenated codes

An application of list recovery



List Recovery

Applications

- Related to list decoding [GI02, GI03, GI04]
- Compressed sensing [NPR12, GNP$^+$13]
- Group testing [INR10]
- Erasure model is weaker than error model (erasure model was studied before in [GI04])

Outline

Tanner Codes [Tanner'81]

Given:

- A d-regular graph G with n vertices and $N = \frac{nd}{2}$ edges
- An *inner code* $\mathcal{C}_0$ with block length d over Σ

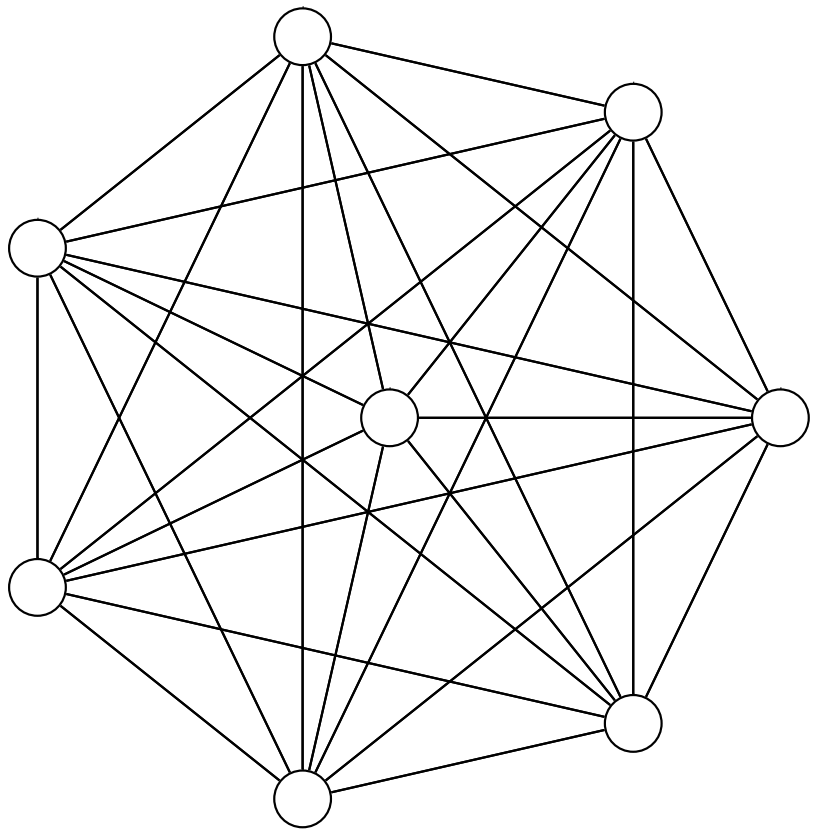
We get a *Tanner code* $\mathcal{C}$.

- $\mathcal{C}$ has block length N and alphabet Σ.
- Codewords are labelings of edges of G.
- A labeling is in $\mathcal{C}$ if the labels on each vertex form a codeword of $\mathcal{C}_0$.
- (We fix an arbitrary ordering of edges at each vertex)

Example [Tanner'81]

G is K_8, and $\mathcal{C}_0$ is the $[7, 4, 3]$-Hamming code.

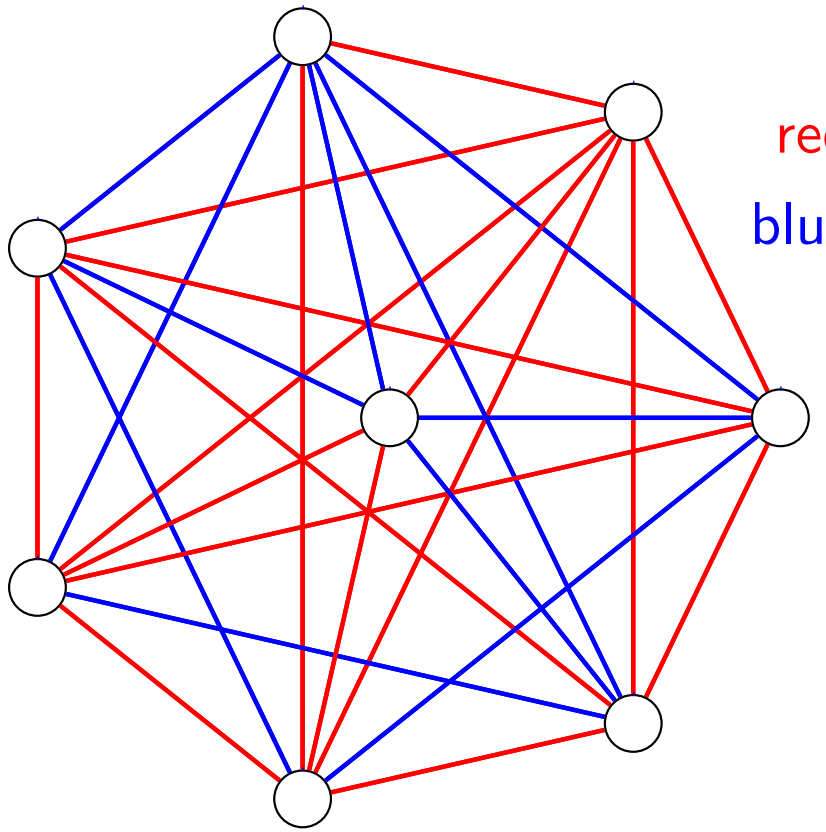
$$N = \binom{8}{2} = 28 \text{ and } \Sigma = \{0, 1\}$$



Example [Tanner'81]

G is K_8, and $\mathcal{C}_0$ is the $[7, 4, 3]$-Hamming code.

A codeword of $\mathcal{C}$ is a labeling of edges of G.



red $\mapsto 0$

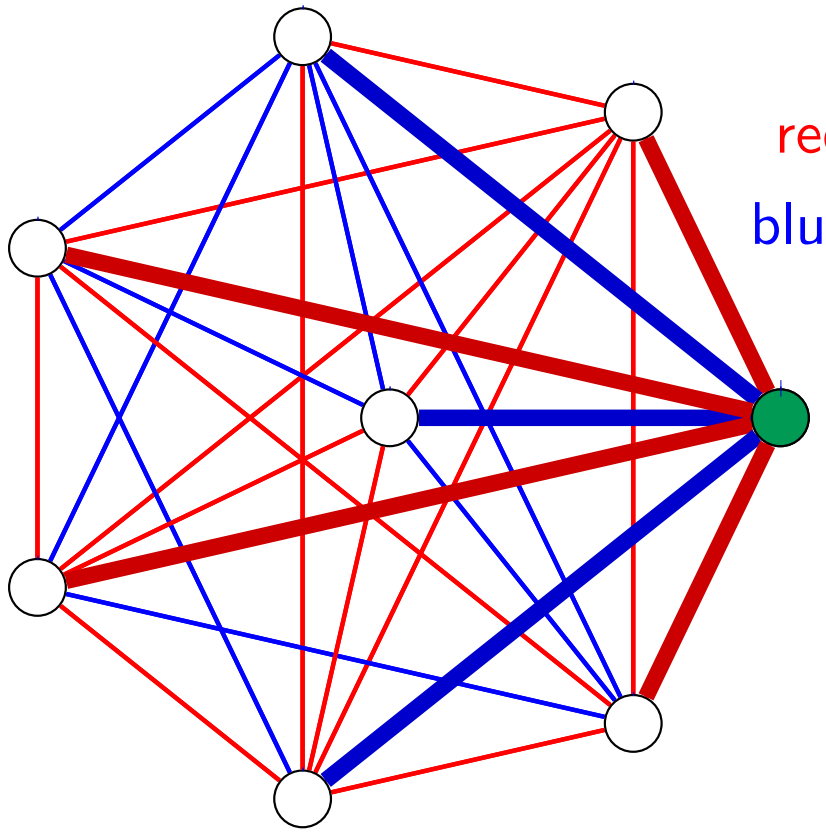
blue $\mapsto 1$

$$(0,0,0,0,0,0,0,1,1,0,1,1,1,0,1,0,0,1,0,1,0,0,0,1,0,0,1,1) \in \mathcal{C} \subset \{0,1\}^{28}$$

Example [Tanner'81]

G is K_8, and $\mathcal{C}_0$ is the $[7, 4, 3]$-Hamming code.

These edges form a codeword in the Hamming code



red $\mapsto 0$

blue $\mapsto 1$

$$(0,0,0,0,0,0,0,1,1,0,1,1,1,0,1,0,0,1,0,1,0,0,0,1,0,0,1,1) \in \mathcal{C} \subset \{0,1\}^{28}$$

Encoding Tanner Codes

Encoding is Easy!

1. Generate parity-check matrix
   Requires:
   - Edge-vertex incidence matrix of graph
   - Parity-check matrix of inner code
2. Calculate a basis for the kernel of the parity-check matrix
3. This basis defines a generator matrix for the linear Tanner Code
4. Encoding is just multiplication by this generator matrix

Linearity

If the inner code $\mathcal{C}_0$ is linear, so is the Tanner code $\mathcal{C}$

- $\mathcal{C}_0 = \mathrm{Ker}(H_0)$ for some *parity check* matrix H_0.

$$x \in \mathcal{C}_0 \iff \boxed{H_0} \quad \boxed{x} \quad = 0$$

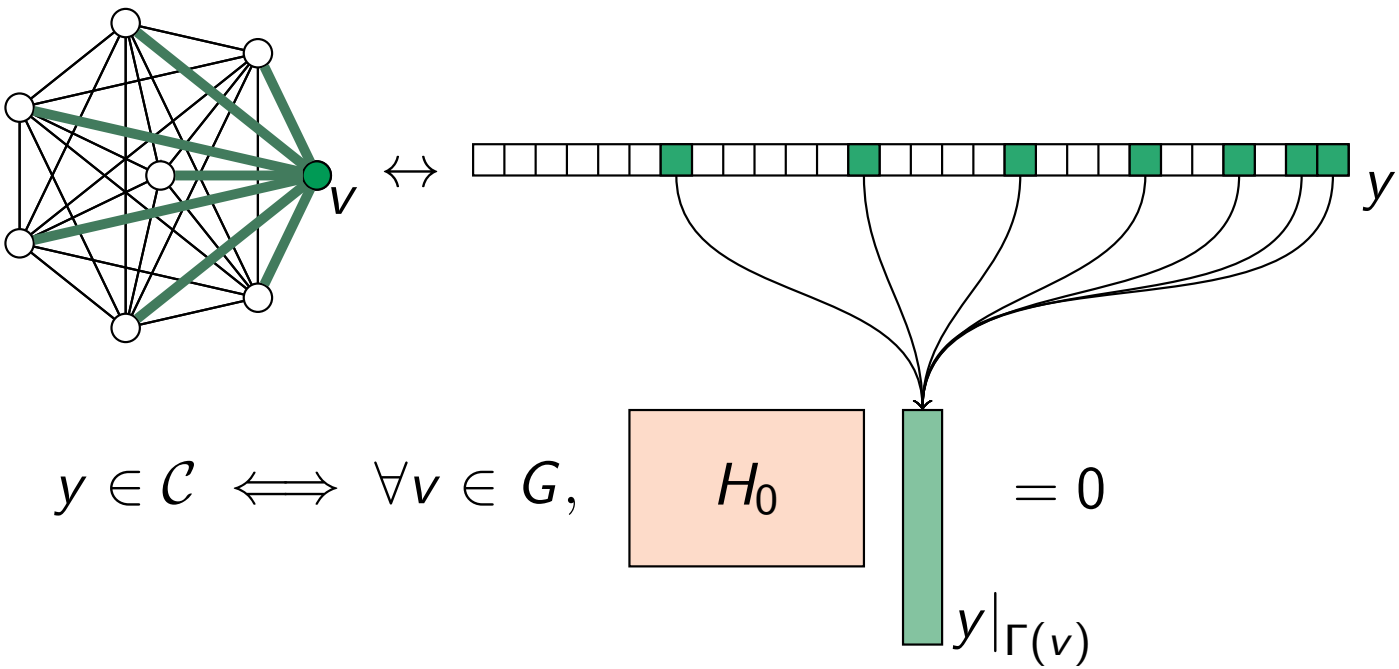
Linearity

If the inner code $\mathcal{C}_0$ is linear, so is the Tanner code $\mathcal{C}$

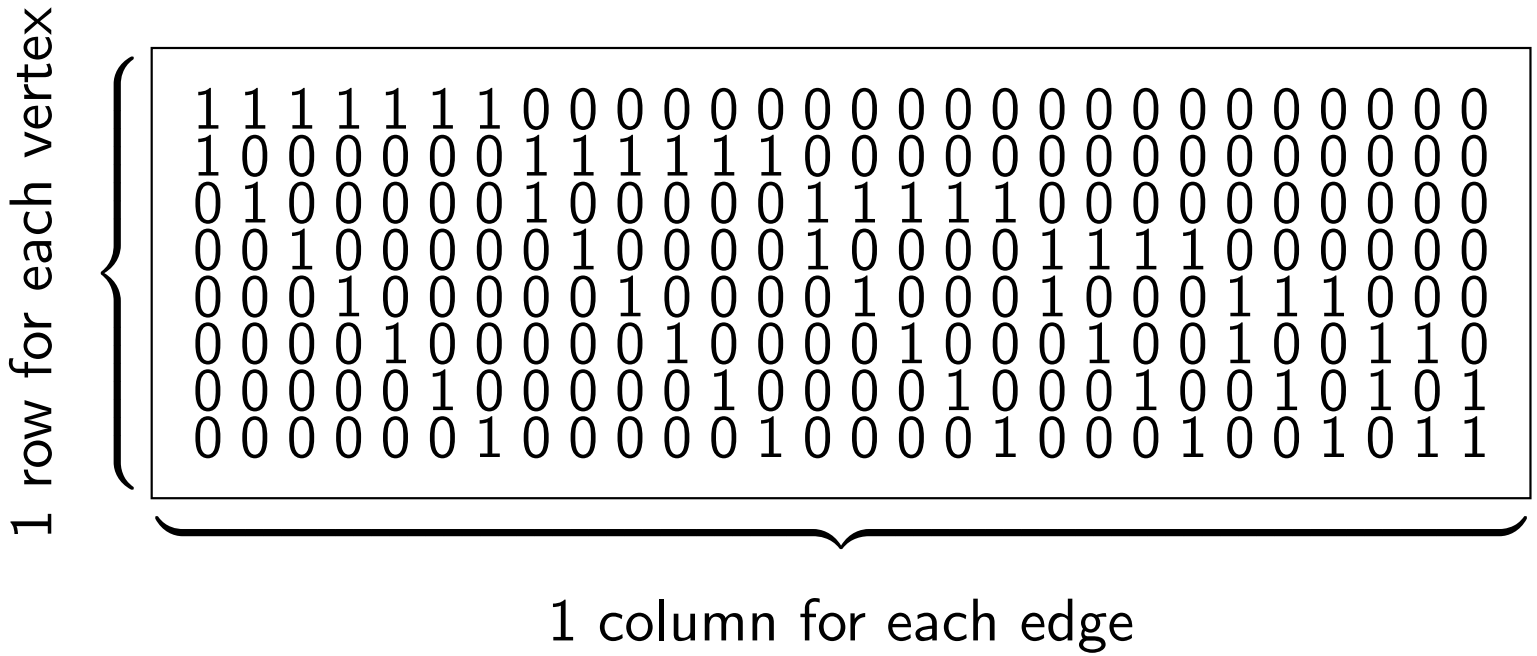
- $\mathcal{C}_0 = \mathrm{Ker}(H_0)$ for some *parity check* matrix H_0.

$$x \in \mathcal{C}_0 \iff \boxed{H_0} \; \boxed{x} \; = 0$$

- So codewords of the Tanner code $\mathcal{C}$ also are defined by linear constraints:



Example: vertex edge incidence matrix of K_8



- Columns have weight 2
  (Each edge hits two vertices)
- Rows have weight 7
  (Each vertex has degree seven)

Example: parity-check matrix of a Tanner code

K_8 and the $[7, 4, 3]$-Hamming code

Parity-check
of Hamming code

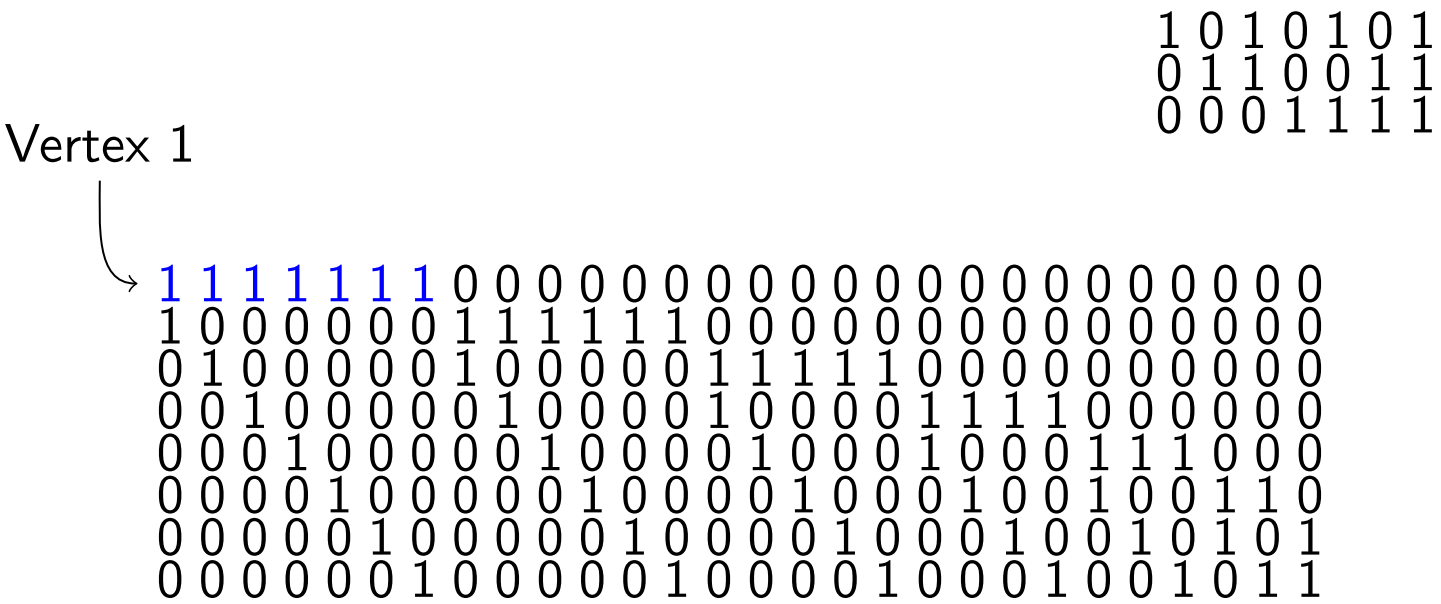
$$\begin{matrix} 1&0&1&0&1&0&1 \\ 0&1&1&0&0&1&1 \\ 0&0&0&1&1&1&1 \end{matrix}$$

$$\begin{matrix} 1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&1&0&0&0&0&0&1&1&1&1&1&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&1&1&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&1&1&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&1&1&0 \\ 0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&1&1 \\ 0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&1 \end{matrix}$$

Edge-vertex incidence matrix of K_8

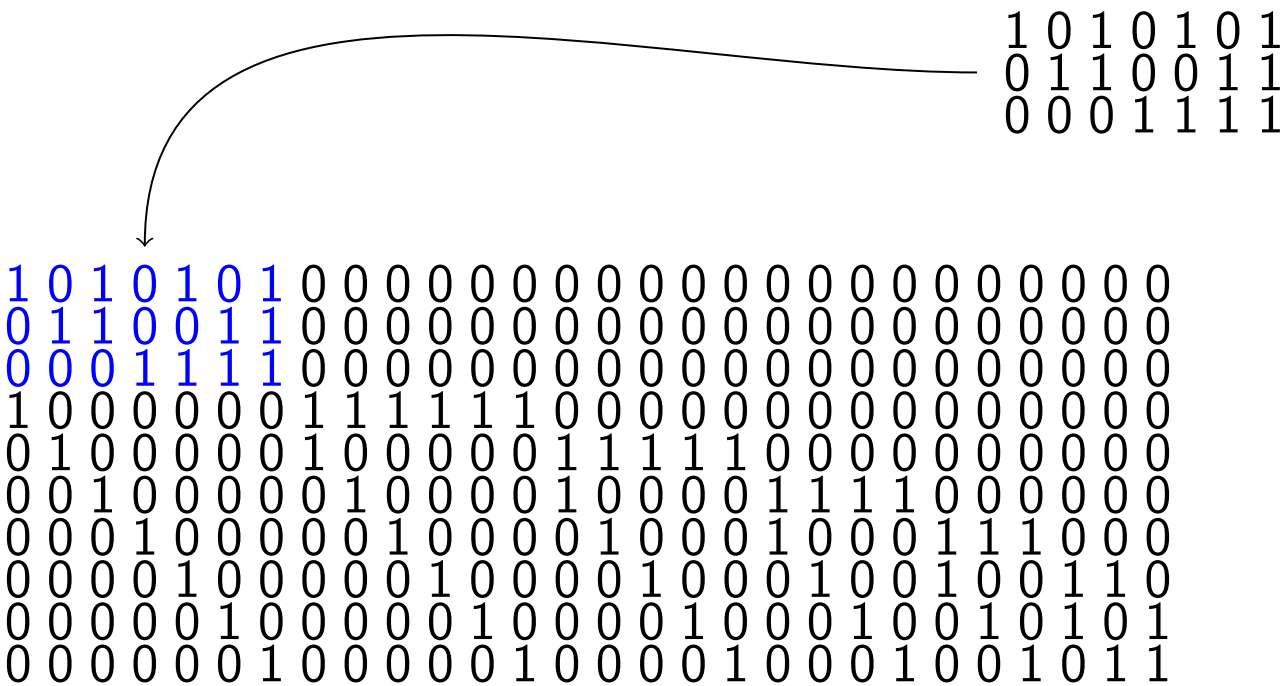
Example: parity-check matrix of a Tanner code

K_8 and the $[7, 4, 3]$-Hamming code



Example: parity-check matrix of a Tanner code

K_8 and the $[7, 4, 3]$-Hamming code



Example: parity-check matrix of a Tanner code

K_8 and the $[7, 4, 3]$-Hamming code

$$\begin{array}{ccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1
\end{array}$$

$$\begin{array}{cccccccccccccccccccccccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1
\end{array}$$

Example: parity-check matrix of a Tanner code

K_8 and the $[7, 4, 3]$-Hamming code

$$\begin{array}{ccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1
\end{array}$$

$$\begin{array}{cccccccccccccccccccccccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1
\end{array}$$

Example: parity-check matrix of a Tanner code

K_8 and the $[7, 4, 3]$-Hamming code

$$
\begin{pmatrix}
1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&1&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&1&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&1&0&0\\
0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&1&0&1&0&1\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&1\\
0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0
\end{pmatrix}
$$

If the inner code has good rate, so does the outer code

Say that $\mathcal{C}_0$ is linear

- If $\mathcal{C}_0$ has rate r_0, it satisfies $(1 - r_0)d$ linear constraints.
- Each of the n vertices of G must satisfy these constraints.

If the inner code has good rate, so does the outer code

Say that $\mathcal{C}_0$ is linear

- If $\mathcal{C}_0$ has rate r_0, it satisfies $(1 - r_0)d$ linear constraints.
- Each of the n vertices of G must satisfy these constraints.

$$\Downarrow$$

- $\mathcal{C}$ is defined by at most $n \cdot (1 - r_0)d$ constraints.

If the inner code has good rate, so does the outer code

Say that $\mathcal{C}_0$ is linear

- If $\mathcal{C}_0$ has rate r_0, it satisfies $(1 - r_0)d$ linear constraints.
- Each of the n vertices of G must satisfy these constraints.

$$\Downarrow$$

- $\mathcal{C}$ is defined by at most $n \cdot (1 - r_0)d$ constraints.
- Length of $\mathcal{C} = N = \#$ edges $= nd/2$

If the inner code has good rate, so does the outer code

Say that $\mathcal{C}_0$ is linear

- If $\mathcal{C}_0$ has rate r_0, it satisfies $(1 - r_0)d$ linear constraints.
- Each of the n vertices of G must satisfy these constraints.

$$\Downarrow$$

- $\mathcal{C}$ is defined by at most $n \cdot (1 - r_0)d$ constraints.
- Length of $\mathcal{C} = N = \#$ edges $= nd/2$
- The rate of $\mathcal{C}$ is

$$R = \frac{k}{N} \geq \frac{N - n \cdot (1 - r_0)d}{N} = 2r_0 - 1.$$

Better rate bounds?

- The lower bound $R > 2r_0 - 1$ is *independent* of the ordering of edges around a vertex
- Tanner already noticed that order matters.
  Let G be the complete bipartite graph with 7 vertices per side
  Let $\mathcal{C}_0$ be the $[7, 4, 3]$ hamming code
  Then different "natural" orderings achieve a Tanner code with
  - $[49, 16, 9]$ $(\frac{16}{49} \approx .327)$
  - $[49, 12, 16]$ $(\frac{12}{49} \approx .245)$
  - $[49, 7, 17]$ $(\frac{7}{49} \approx .142)$ Meets lower bound of $2 \cdot \frac{4}{7} - 1$

Expander codes

When the underlying graph is an *expander graph*, the Tanner code is a *expander code.*

- Expander codes admit very fast decoding algorithms [Sipser and Spielman 1996]
- Further improvements in [Sipser'96, Zemor'01, Barg and Zemor'02,'05,'06]

Outline

Linear-time list-recoverable codes

Theorem (Main Theorem)

- *Suppose that $\mathcal{C}_0$ is (α_0, ℓ, L_0)-list recoverable from erasures.*
- *Suppose G is an expander graph of degree d.*
  - *d needs to be big enough compared to ℓ, L_0.*

- *Then the expander code $\mathcal{C}$ is (α, ℓ, L)-list recoverable from erasures.*
  - *It can be list-recovered in linear time.*
  - *Above, α, L are constants (independent of n).*

...

Comparison with Previous Work

Code	Rate	Decoding time
Random code	$1 - \varepsilon$	lots
Folded RS (+friends) [GR08, Gur11, DL12]	$1 - \varepsilon$	$\mathrm{poly}(n)$
[GI03, GI04]	$1/\mathrm{poly}(\ell)$	$O(n)$
This work (Random inner code)	$1 - \varepsilon$	$O(n)$

- All list sizes L are constants
  - but some are *huge* constants

Improving the rate/error tradeoff

Theorem ("Dream theorem" of Meir'14)

Constructing codes of rate $\to 1$ *with property* $\mathcal{P}$ *and decent distance*

$$\Rightarrow$$

Constructing codes with $\mathcal{P}$ *that approach the Singleton bound.*

Improving the rate/error tradeoff

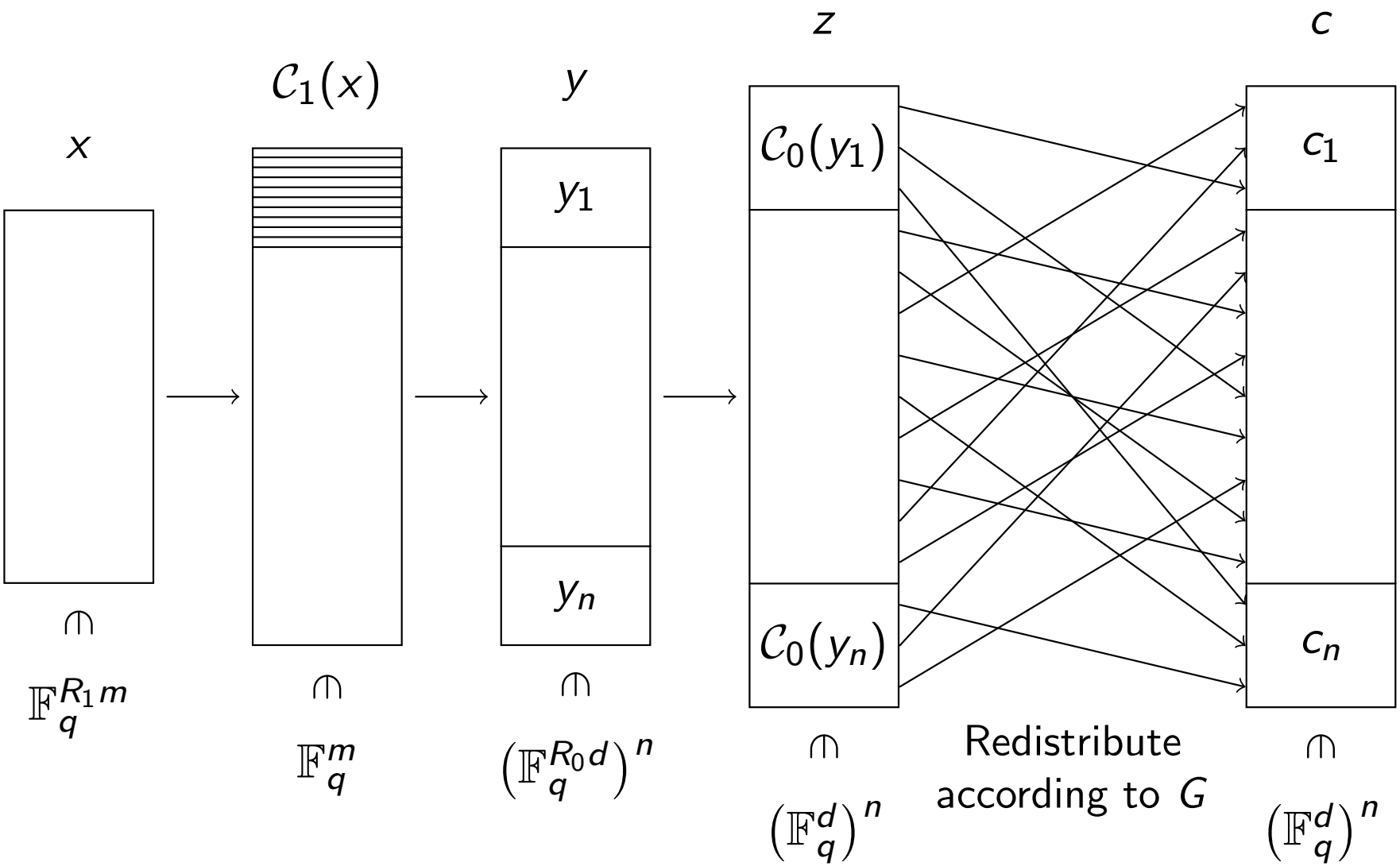
Theorem

For any $R > 0$, $\ell > 0$, *and* $\varepsilon > 0$,
and for any large enough L, d, q *(depending only on* ℓ, ε*),*
there is a family of codes with:

- *rate at least* R
- $(R + \varepsilon, \ell, L)$*-list-recoverable in linear time.*

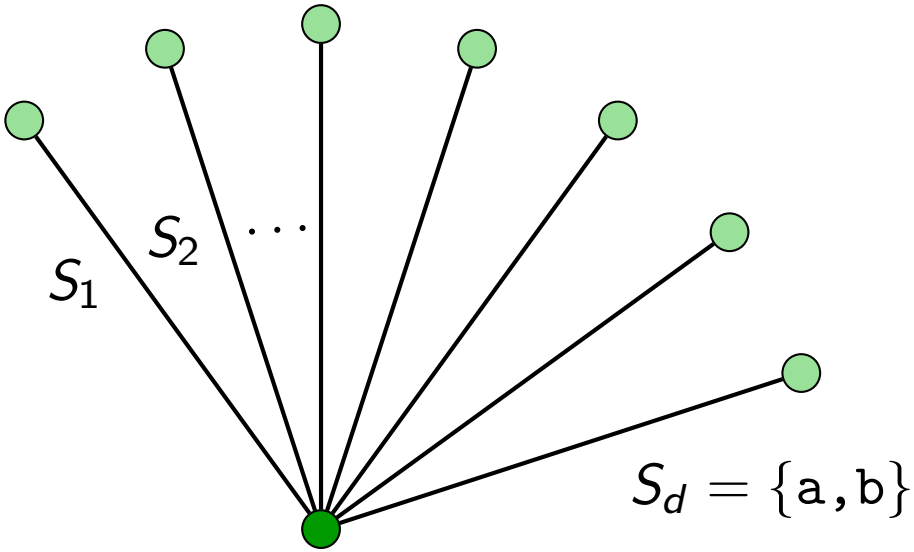
Improving the rate/error tradeoff

The Or Meir Construction [Mei14]



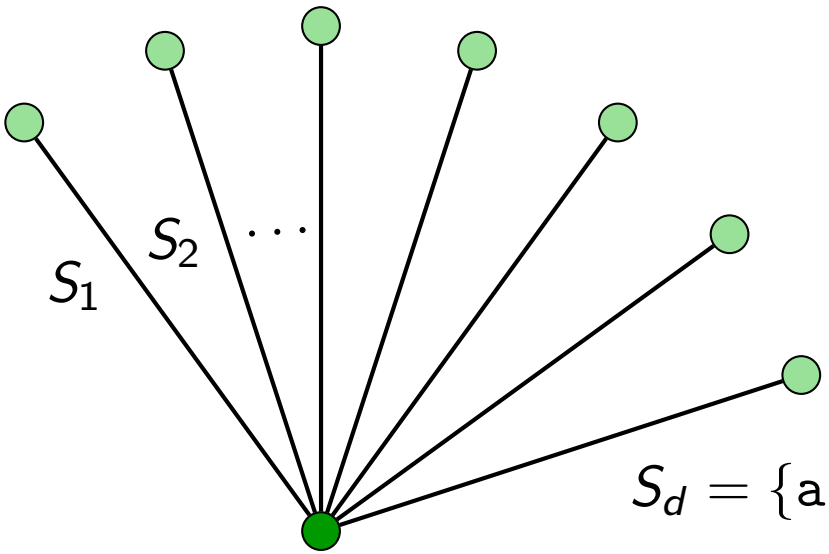
The algorithm

Suppose that $\mathcal{C}_0$ is list-recoverable.



The algorithm

Suppose that $\mathcal{C}_0$ is list-recoverable.

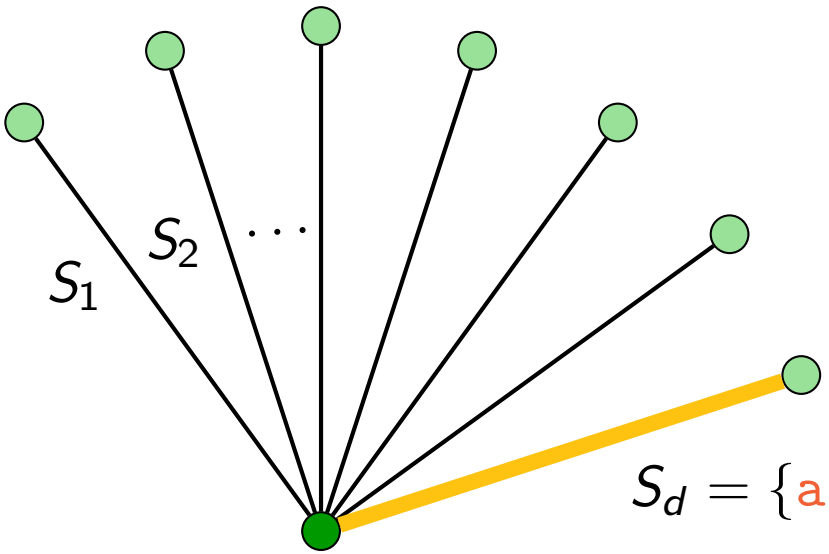


c_1	a d c d c f b f c d c f a
$\vdots$	d b c r e t a b c b c d b
c_L	a b x d e f a b z d e d a

1. List recover locally at this vertex: get $\{c_1, \ldots, c_L\}$

The algorithm

Suppose that $\mathcal{C}_0$ is list-recoverable.



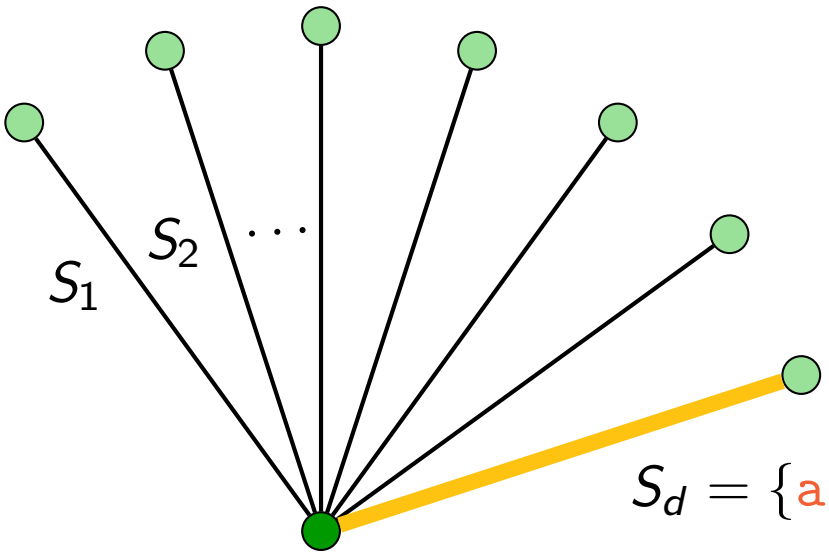
1. List recover locally at this vertex: get $\{c_1, \ldots, c_L\}$

c_1	a d c d c f b f c d c f a
$\vdots$	d b c r e t a b c b c d b
c_L	a b x d e f a b z d e d a

2. What if we know this symbol is "a"?

The algorithm

Suppose that $\mathcal{C}_0$ is list-recoverable.



1. List recover locally at this vertex: get $\{c_1, \ldots, c_L\}$

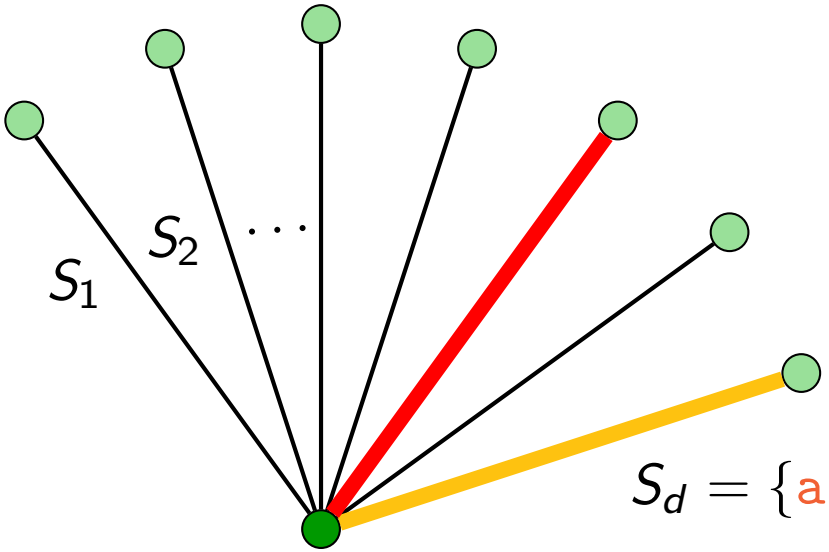
c_1	a	d	c	d	c	f	b	f	c	d	c	f	a
$\vdots$	d	b	c	r	e	t	a	b	c	b	c	d	b
c_L	a	b	x	d	e	f	a	b	z	d	e	d	a

$S_d = \{\mathtt{a}, \mathtt{b}\}$

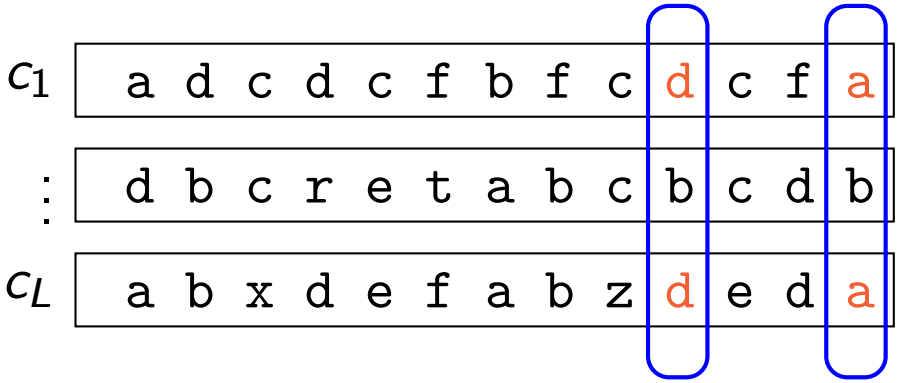
2. What if we know this symbol is "a"?

The algorithm

Suppose that $\mathcal{C}_0$ is list-recoverable.



1. List recover locally at this vertex: get $\{c_1, \ldots, c_L\}$

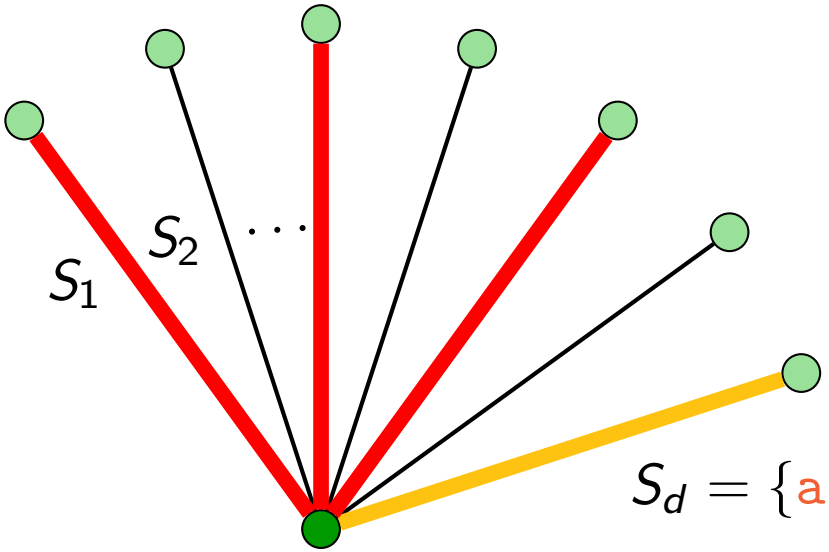


$S_d = \{\mathtt{a}, \mathtt{b}\}$

2. What if we know this symbol is "a"?

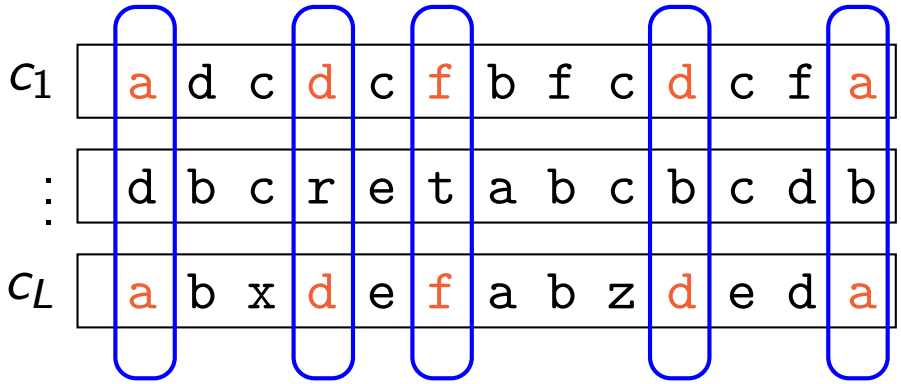
The algorithm

Suppose that $\mathcal{C}_0$ is list-recoverable.



1. List recover locally at this vertex: get $\{c_1, \ldots, c_L\}$

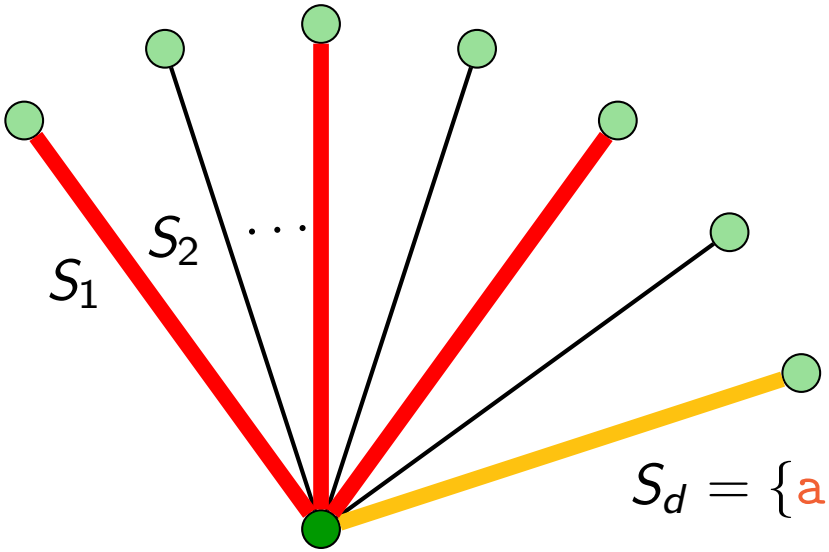
2. What if we know this symbol is "a"?



3. Then we know a bunch of other symbols too.

The algorithm

Suppose that $\mathcal{C}_0$ is list-recoverable.



1. List recover locally at this vertex: get $\{c_1, \ldots, c_L\}$

2. What if we know this symbol is "a"?

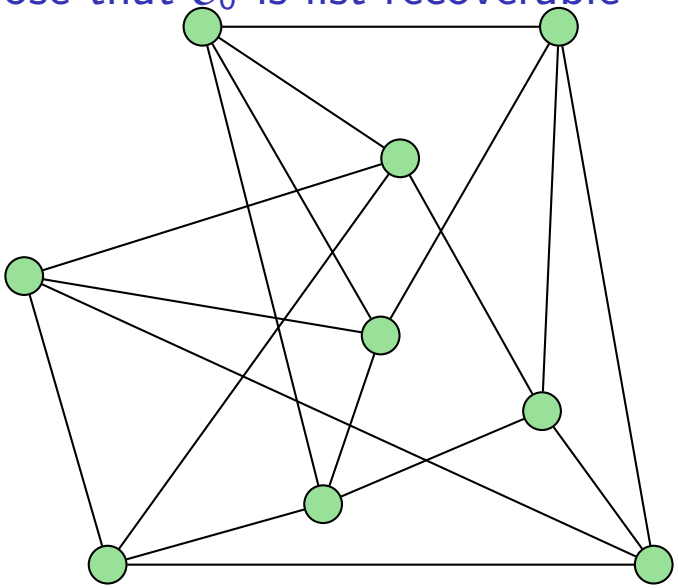
c_1	a	d	c	d	c	f	b	f	c	d	c	f	a
$\vdots$	d	b	c	r	e	t	a	b	c	b	c	d	b
c_L	a	b	x	d	e	f	a	b	z	d	e	d	a

3. Then we know a bunch of other symbols too.

(L^ℓ possible columns. Each choice fixes about d/L^ℓ others)

The algorithm

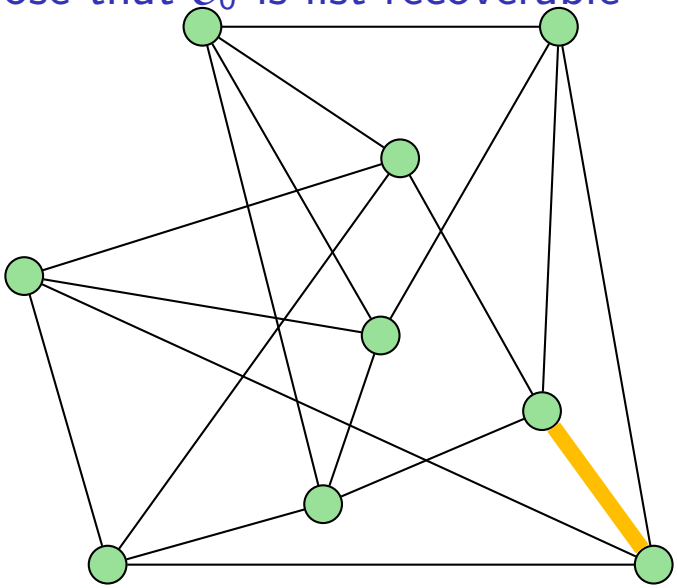
Suppose that $\mathcal{C}_0$ is list-recoverable



- Make one decision.

The algorithm

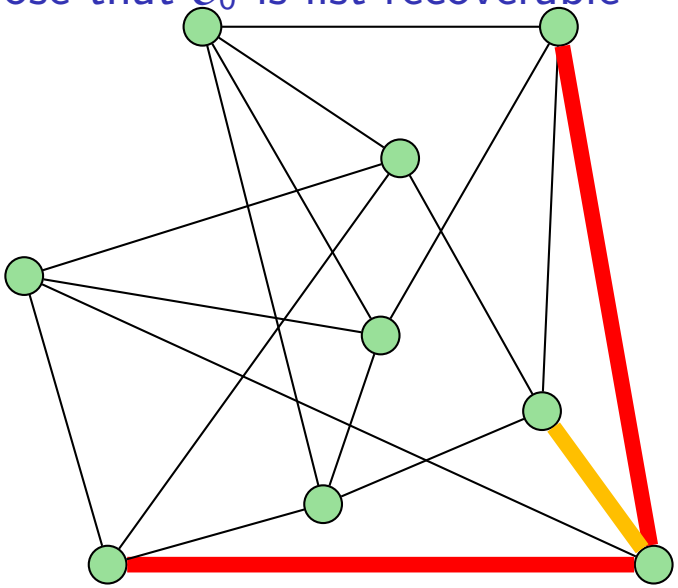
Suppose that $\mathcal{C}_0$ is list-recoverable



- Make one decision.

The algorithm

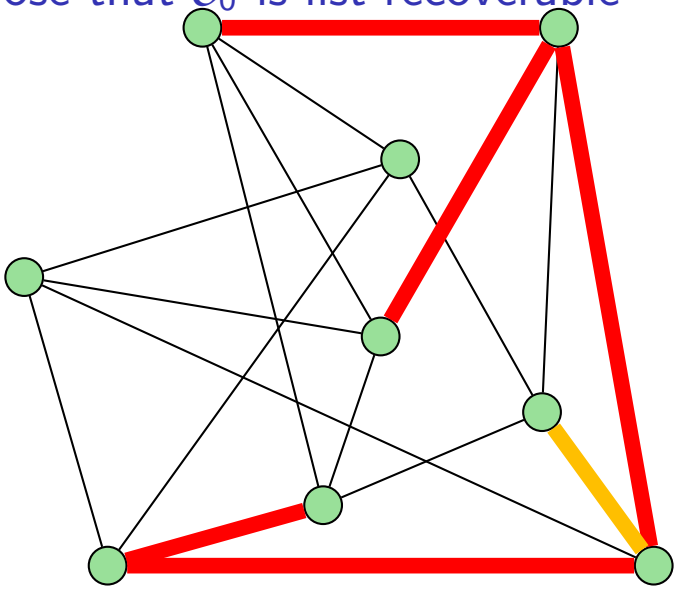
Suppose that $\mathcal{C}_0$ is list-recoverable



- Make one decision.

The algorithm

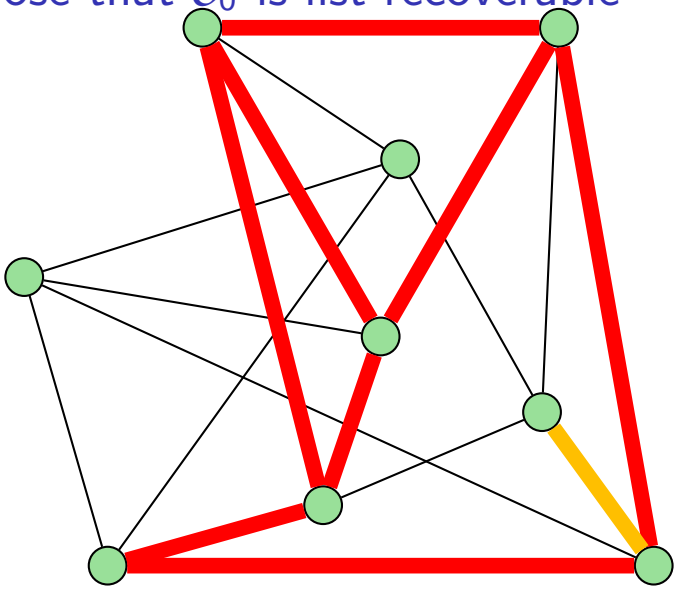
Suppose that $\mathcal{C}_0$ is list-recoverable



- Make one decision.

The algorithm

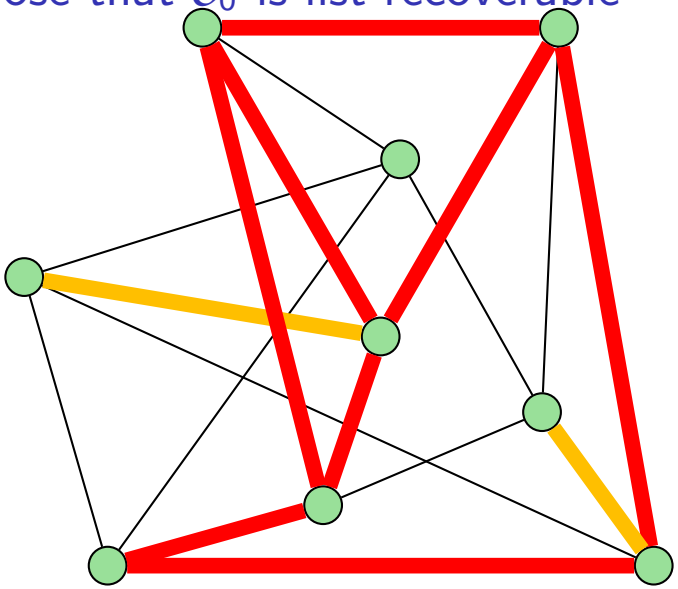
Suppose that $\mathcal{C}_0$ is list-recoverable



- Make one decision.
- Determine a bunch of other edges.

The algorithm

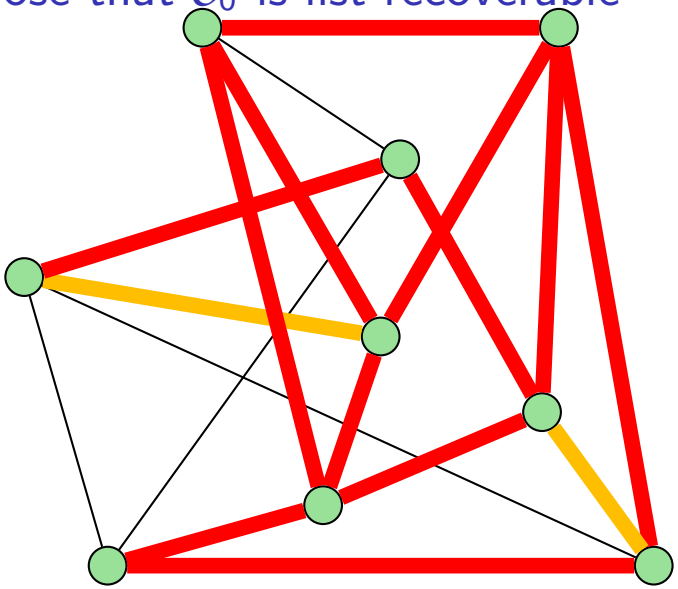
Suppose that $\mathcal{C}_0$ is list-recoverable



- Make one decision.
- Determine a bunch of other edges.
- Make another decision.

The algorithm

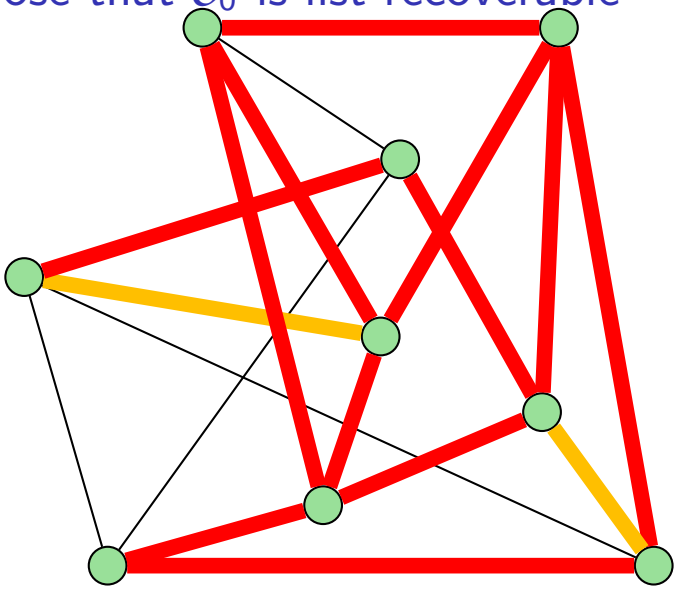
Suppose that $\mathcal{C}_0$ is list-recoverable



- Make one decision.
- Determine a bunch of other edges.
- Make another decision.
- Determine some more edges.

The algorithm

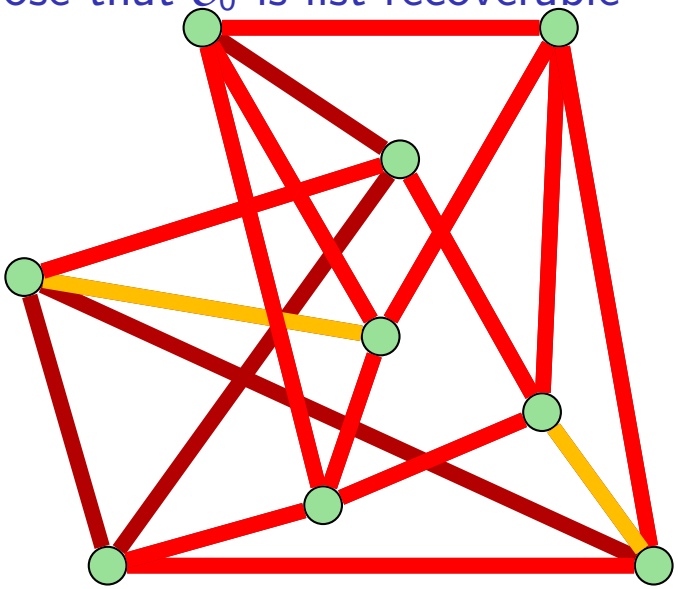
Suppose that $\mathcal{C}_0$ is list-recoverable



- Make one decision.
- Determine a bunch of other edges.
- Make another decision.
- Determine some more edges.
- ...

The algorithm

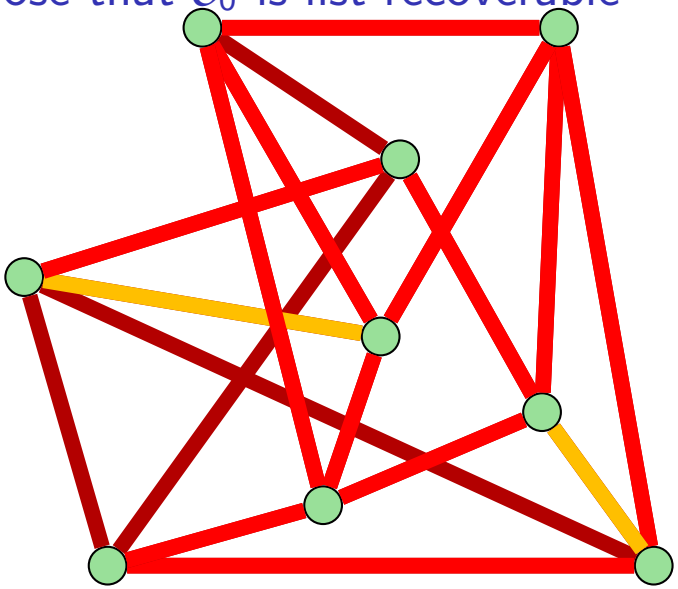
Suppose that $\mathcal{C}_0$ is list-recoverable



- Make one decision.
- Determine a bunch of other edges.
- Make another decision.
- Determine some more edges.
- ...
- Correct the rest using regular decoding alg.

The algorithm

Suppose that $\mathcal{C}_0$ is list-recoverable



- Make one decision.
- Determine a bunch of other edges.
- Make another decision.
- Determine some more edges.
- ...
- Correct the rest using regular decoding alg.

$$\text{Number of possibilities} = \ell^{\text{number of } \textbf{decision edges}} = \ell^{O(1)}.$$

Why does propagation work?

Equivalence classes of edges

- $L > \ell$ so choosing single symbol won't determine CW
- L CWs, ℓ choices per index
- L^ℓ possible "columns"
  (For each index i, L^ℓ maps from CW to symbol)
  (L^ℓ maps from $[L] \to \ell$)
- Edges are in same equivalance class (wrt vertex) if knowing one determines all the others
- Each vertex has d edges
  At most L^ℓ equivalence classes
  Average equivalence class is of size d/L^ℓ

Why does propagation work?

Large Equivalance Classes

- Average size of an equivalence class is d/L^{ℓ}
- What if we get unlucky and pick an edge in a small equivalence class?
- By the expansion property of the graph there is a large (constant fraction) subset of edges that all have large equivalence class
- Probability a uniformly chosen edge covers an $\varepsilon(\varepsilon - \lambda)$ fraction of the graph is at least $1 - 3\varepsilon\ell^{L}$

Results

- $(R + \eta, \ell, L)$-list recoverable codes
  - Any $R > 0$
  - Any $\ell > 0$
  - Any $\eta > 0$
  - L depends only on ℓ, η
- Linear-time recovery algorithm

Outline

Moral of the story

Inner code has decent _____

$\Rightarrow$

So does the expander code and there's an efficient algorithm!

Moral of the story

Inner code has decent list-recoverability

$\Rightarrow$

So does the expander code and it's list-recoverable in linear time!

Moral of the story

Inner code has decent list-recoverability $\Rightarrow$ So does the expander code and it's list-recoverable in linear time!

Open Questions:

Moral of the story

Inner code has decent list-recoverability $\Rightarrow$ So does the expander code and it's list-recoverable in linear time!

Open Questions:

- What other properties can expander codes improve?

Moral of the story

Inner code has decent list-recoverability $\Rightarrow$ So does the expander code and it's list-recoverable in linear time!

Open Questions:

- What other properties can expander codes improve?
- Handle errors?

Moral of the story

Inner code has decent list-recoverability $\Rightarrow$ So does the expander code and it's list-recoverable in linear time!

Open Questions:

- What other properties can expander codes improve?
- Handle errors?
- Other applications with erasures?

Moral of the story

Inner code has decent list-recoverability $\Rightarrow$ So does the expander code and it's list-recoverable in linear time!

Open Questions:

- What other properties can expander codes improve?
- Handle errors?
- Other applications with erasures?
- Better inner code? Better constants?

The end

Thanks!

References I


Alexander Barg and Gilles Zemor.
Error exponents of expander codes.
Information Theory, IEEE Transactions on, 48(6):1725–1729, June 2002.


Alexander Barg and Gilles Zemor.
Concatenated codes: serial and parallel.
Information Theory, IEEE Transactions on, 51(5):1625–1634, May 2005.


Alexander Barg and Gilles Zemor.
Distance properties of expander codes.
Information Theory, IEEE Transactions on, 52(1):78–90, January 2006.


Zeev Dvir and Shachar Lovett.
Subspace Evasive Sets.
In *STOC '12*, pages 351–358, October 2012.

References II

Venkatesan Guruswami and Piotr Indyk.
Near-optimal Linear-time Codes for Unique Decoding and New List-decodable Codes over Smaller Alphabets.
In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 812–821, New York, NY, USA, 2002. ACM.

Venkatesan Guruswami and Piotr Indyk.
Linear time encodable and list decodable codes.
In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, STOC '03, pages 126–135, New York, NY, USA, 2003. ACM.

References III

Venkatesan Guruswami and Piotr Indyk.
Efficiently decodable codes meeting Gilbert-Varshamov bound for low rates.
In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 756–757, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

Anna C. Gilbert, Hung Q. Ngo, Ely Porat, Atri Rudra, and Martin J. Strauss.
2/2-Foreach Sparse Recovery with Low Risk.
In FedorV Fomin, Rsiņš Freivalds, Marta Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming*, volume 7965 of *Lecture Notes in Computer Science*, pages 461–472. Springer Berlin Heidelberg, 2013.

References IV


Venkatesan Guruswami and Atri Rudra.
Concatenated Codes Can Achieve List-decoding Capacity.
In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, pages 258–267, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.


Venkatesan Guruswami.
Linear-algebraic list decoding of folded Reed-Solomon codes.
In *IEEE Conference on Computational Complexity*, pages 77–85, 2011.


Brett Hemenway, Rafail Ostrovsky, and Mary Wootters.
Local Correctability of Expander Codes.
In *ICALP*, LNCS. Springer, April 2013.

References V


Piotr Indyk, Hung Q. Ngo, and Atri Rudra.
Efficiently decodable non-adaptive group testing.
In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 1126–1142, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.


Or Meir.
Locally Correctable and Testable Codes Approaching the Singleton Bound.
ECCC Report 2014-107, 2014.


Hung Q. Ngo, Ely Porat, and Atri Rudra.
Efficiently Decodable Compressed Sensing by List-Recoverable Codes and Recursion.
In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, volume 14 of *Leibniz International*

References VI

Proceedings in Informatics (LIPIcs), pages 230–241, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.


Daniel A. Spielman.
Linear-time encodable and decodable error-correcting codes.
Information Theory, IEEE Transactions on, 42(6):1723–1731, November 1996.


Michael Sipser and Daniel A. Spielman.
Expander codes.
Information Theory, IEEE Transactions on, 42(6):1710–1722, November 1996.


Gilles Zemor.
On expander codes.
Information Theory, IEEE Transactions on, 47(2):835–837, February 2001.