Tutorial review
on

Fountain Codes
and
Locally Repairable Codes

Hong-Yeop Song

Yonsei University

Seoul, Korea

Feb. 2019

http://coding.yonsei.ac.kr

Contents

- Fountain Codes
  - Introduction to digital fountain
  - Approximations of digital fountain
  - Break
  - LT codes and Raptor codes
- Break time
- Locally Repairable Codes

YONSEI UNIVERSITY

Tutorial review

on

Fountain Codes

Introduction to Digital Fountain

- Binary Erasure Channels
- Data Transmission
  - Data Carousel Approach
- Fountain Code Paradigm
- Ideal Digital Fountain
- Examples of Binary Fountain codes

Wiki says…

- In coding theory, **fountain codes** (also known as **rateless erasure codes**) are a class of **erasure codes** with the property that
  - a potentially **limitless** sequence of encoding symbols can be generated from a given set of source symbols such that
  - **the original source symbols can ideally be recovered from any subset of the encoding symbols of size equal to or only slightly larger than the number of source symbols.**
- The term *fountain* or *rateless* refers to the fact that **these codes do not exhibit a fixed code rate.**

Wiki says…

- A fountain code is **optimal** if the original k source symbols can be recovered from any k encoding symbols.
- Fountain codes are known that have efficient encoding and decoding algorithms and that allow the recovery of the original k source symbols from any k' of the encoding symbols with **high** probability, where k' is just slightly larger than k.

Wiki says…

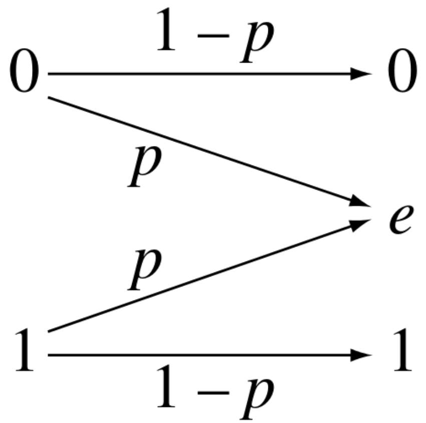
- **LT codes** were the first practical realization of **fountain codes**.
- **Raptor codes and online codes** were subsequently introduced, and achieve **linear time encoding and decoding complexity** through a pre-coding stage of the input symbols.

Discrete coding channel model

- A discrete coding channel is a model of communication channel including **digital modem**, rx(tx) **antenna** and **analog physical** (RF/CABLE) **channel**.
- It is characterized by **(1) input alphabet (2) output alphabet, and (3) transition probabilities between these two**.
- Famous examples are BSC, BEC, etc.

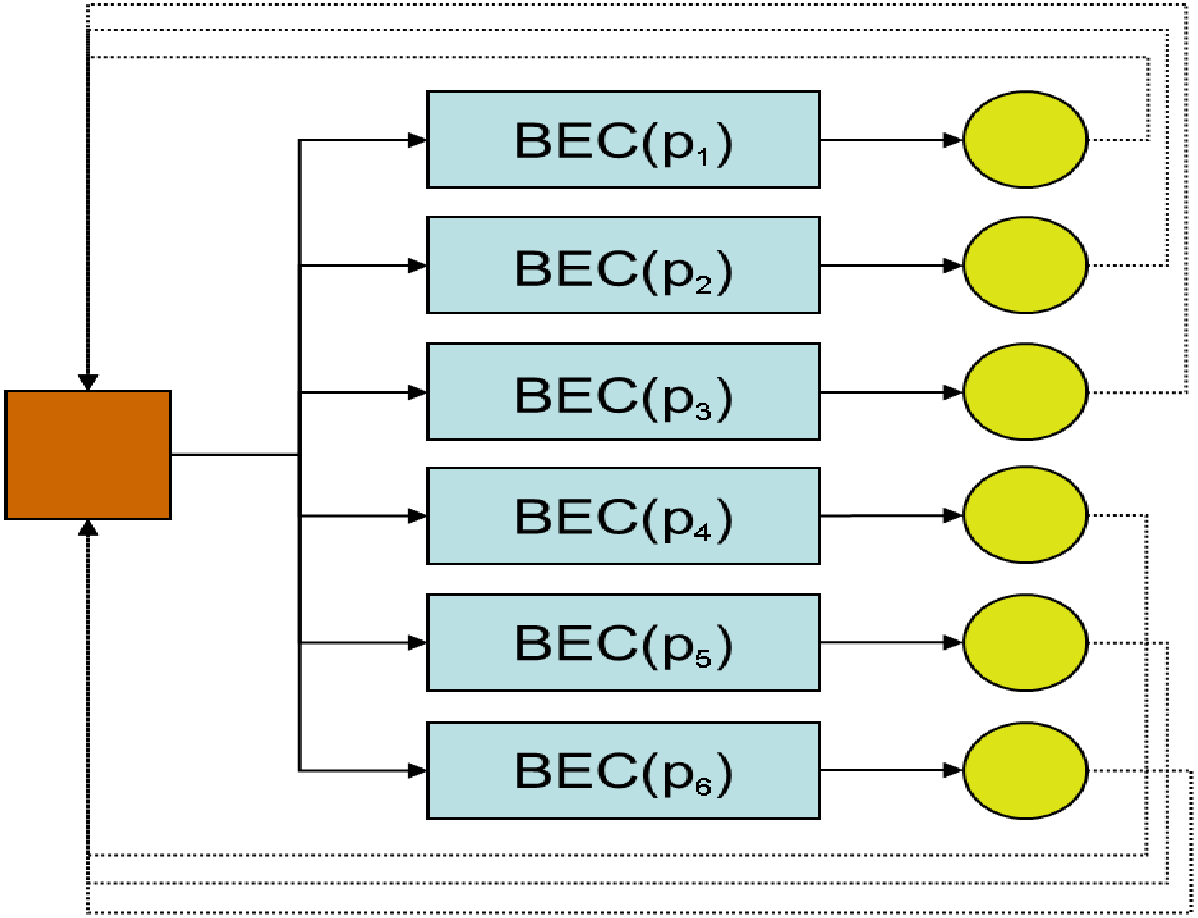
Binary Erasure Channel

- Many applications for data transmissions are based on the packet transmissions **(over the wired or wireless internet with relatively "high" reliability)**
- Packet losses on **the packet based data transmission systems** can be well modeled by the binary erasure channel (BEC), where received 0 or 1 is error-free with probability $1-p$.

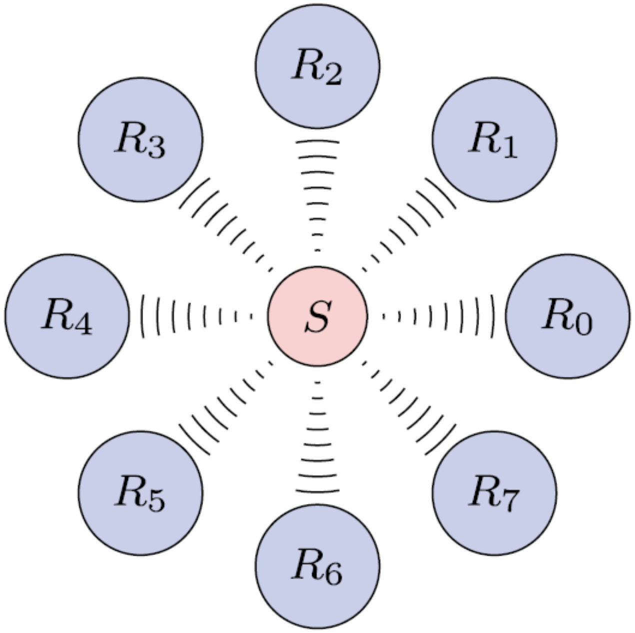


BEC(p)

Communication over Multiple Unknown BECs



Fountain codes is designed for…



Dissemination of bulk data to many users, where

(1) each user may **choose when** to receive,

(2) **no retransmission** is required, hence,

(3) **no back channel for retransmission request** is needed, and finally,

(4) reception of **the same amount of data** as the source data is enough to recover the original bulk data

Data Carousel Approach (Merry-go-round)

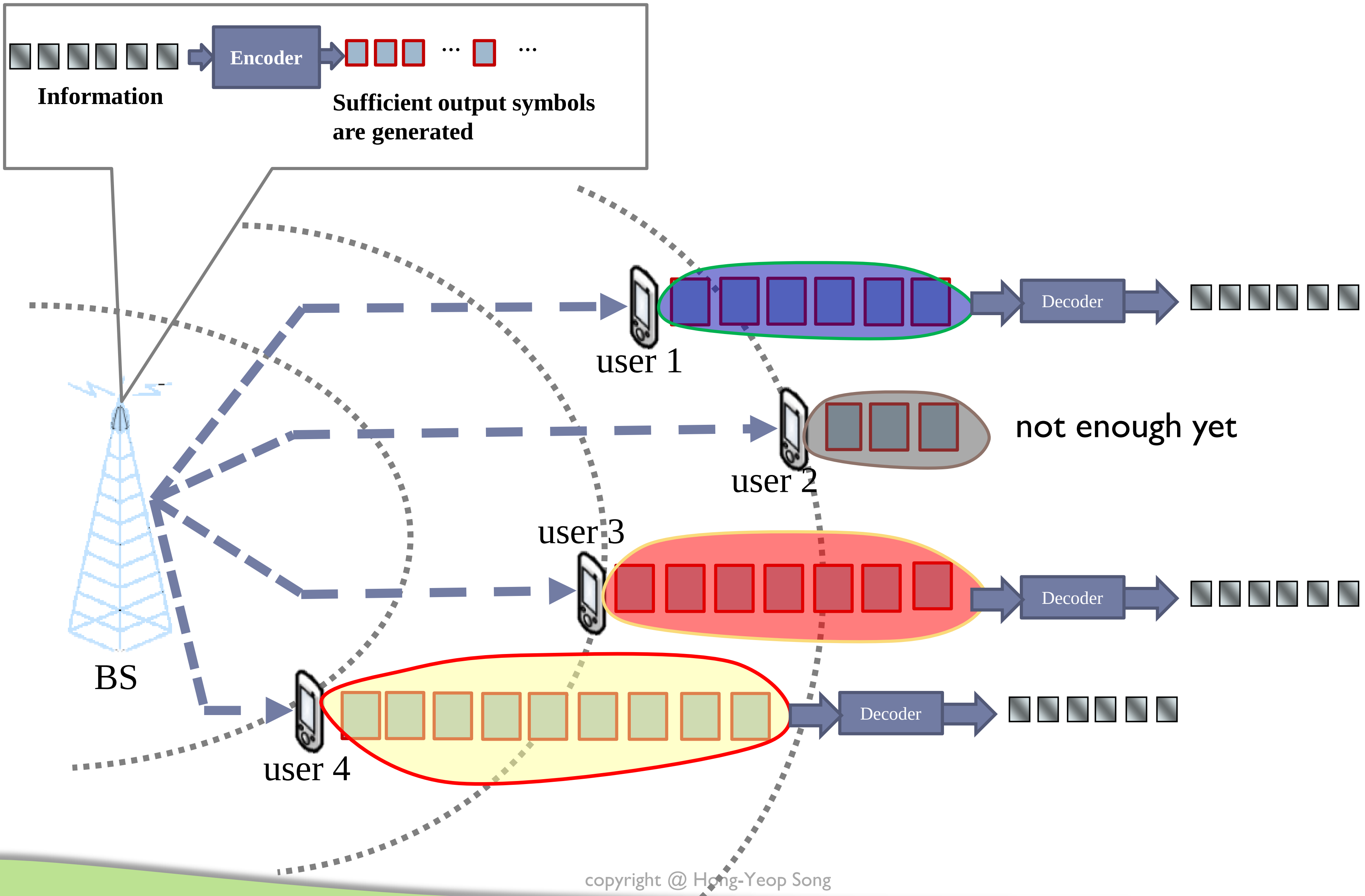
- **The source repeatedly loops through transmission of all data packets**
  - Receivers may join the stream at any time
  - Extremely high reception overhead
- **Adding redundant codewords to the carousel**
  - Reduce reception overhead
  - The source repeatedly loops through the set of coded blocks
- **These approaches eliminate the need for retransmission requests**
  - can be thought as weak approximations of an ideal solution, *digital fountain*

Why the Name Fountain Code?

- Think of the symbols as drops of water
- Fill a bucket with these drops

- As soon as you have enough drops, the bucket is full, and you can drink your water
- **It does not matter (1) which particular drops fill your bucket, or (2) when to receive the drops in which order, only the total amount matters**

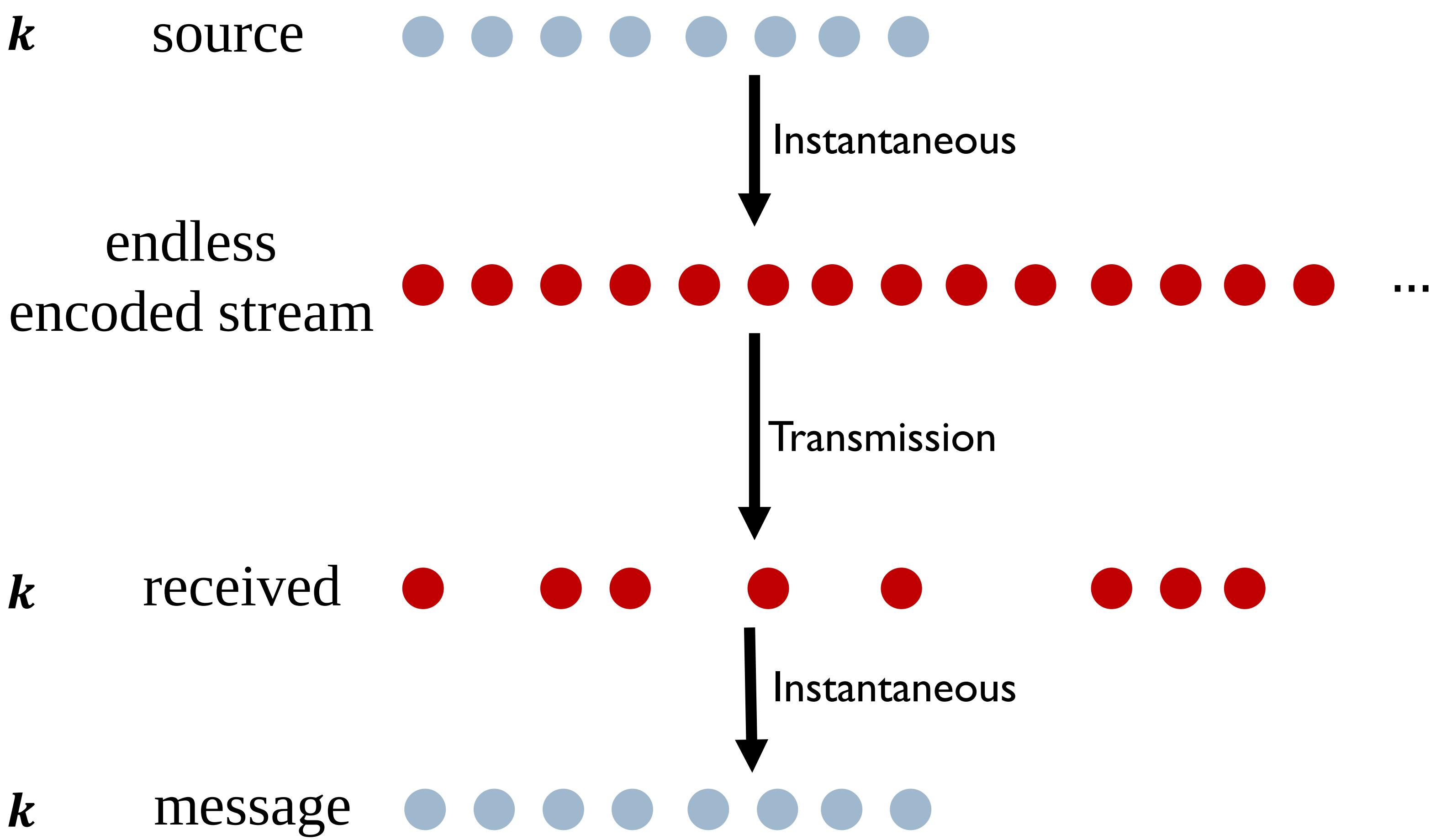
Point-to-Multipoint Communication



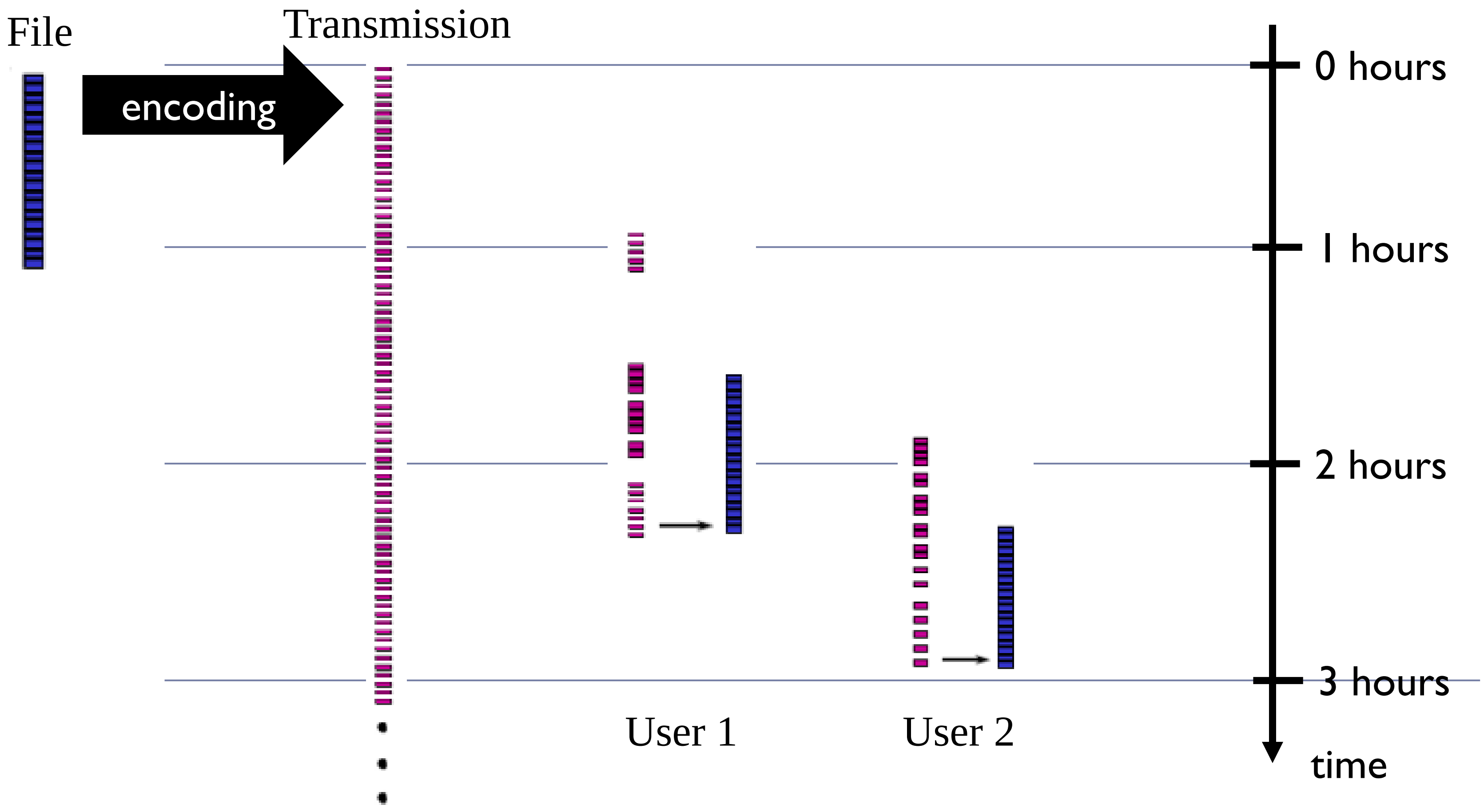
Digital Fountain Ideal

- An ideal digital fountain that transmits a file **(consisting of *k* symbols)** should have the following properties:
  1. It can generate an **endless supply** of encoding packets **with constant encoding cost per symbol** in terms of time or arithmetic operations
  2. A user can reconstruct the file using **any *k* symbols with constant decoding cost per symbol**, meaning that the decoding is linear in ***k***.

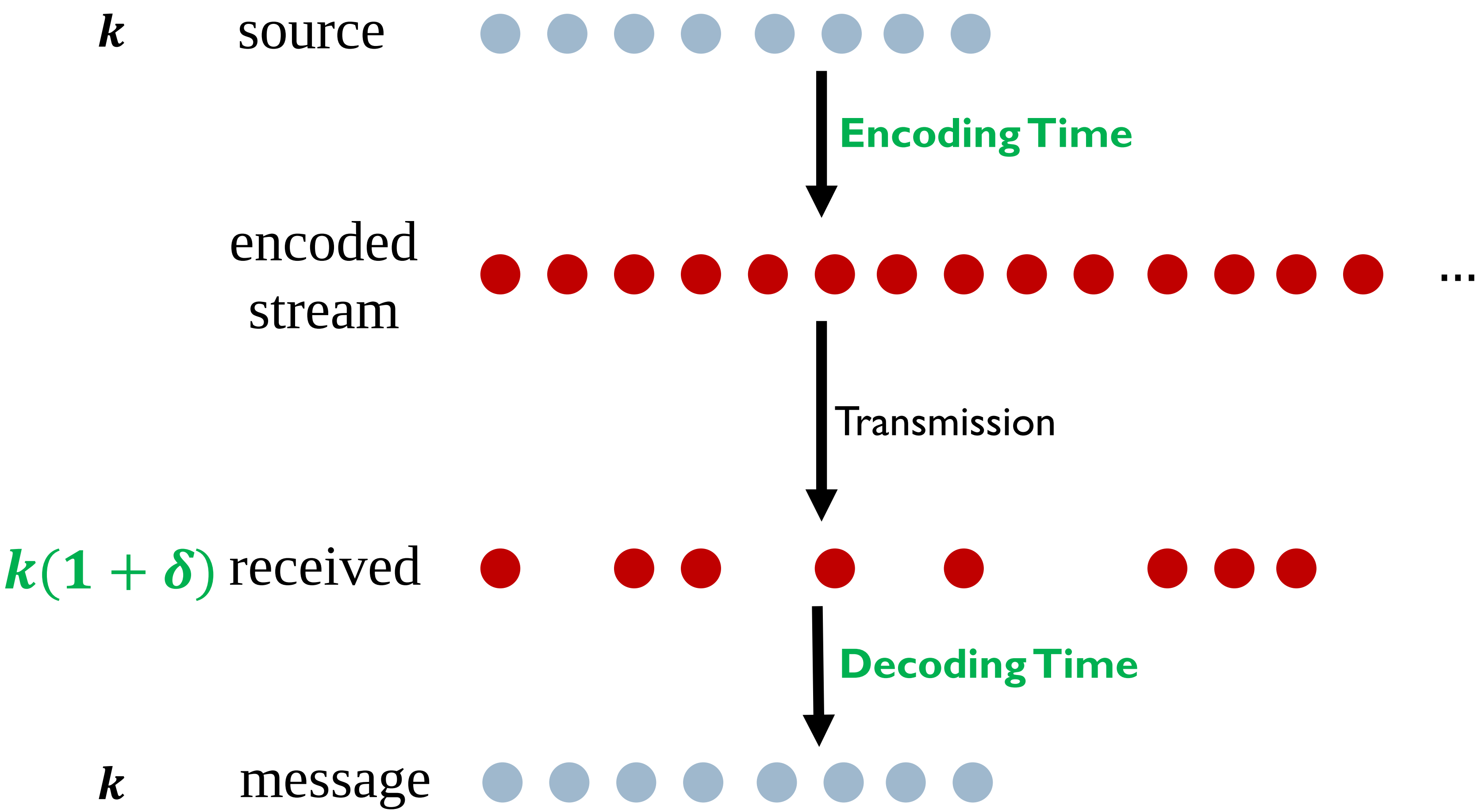
Digital Fountain (ideal case)



Digital Fountain



Approximating A Digital Fountain (in reality)



Binary Fountain Codes

- Notations
  - $V =$ the k-tuple vector space over $\mathbb{F}_2$
  - $V^* = V \setminus \{\mathbf{0}_{k\times 1}\}$
  - **Fix a distribution $\mathcal{D}$ on V^***
- A binary fountain code is **defined by** parameters $(\mathcal{D}, k)$, where k is the number of input symbols

EXAMPLE:

When $k = 2$, for example, we have a distribution on

{(01), (10), (11)},

as {1/4, 1/4, 1/2},

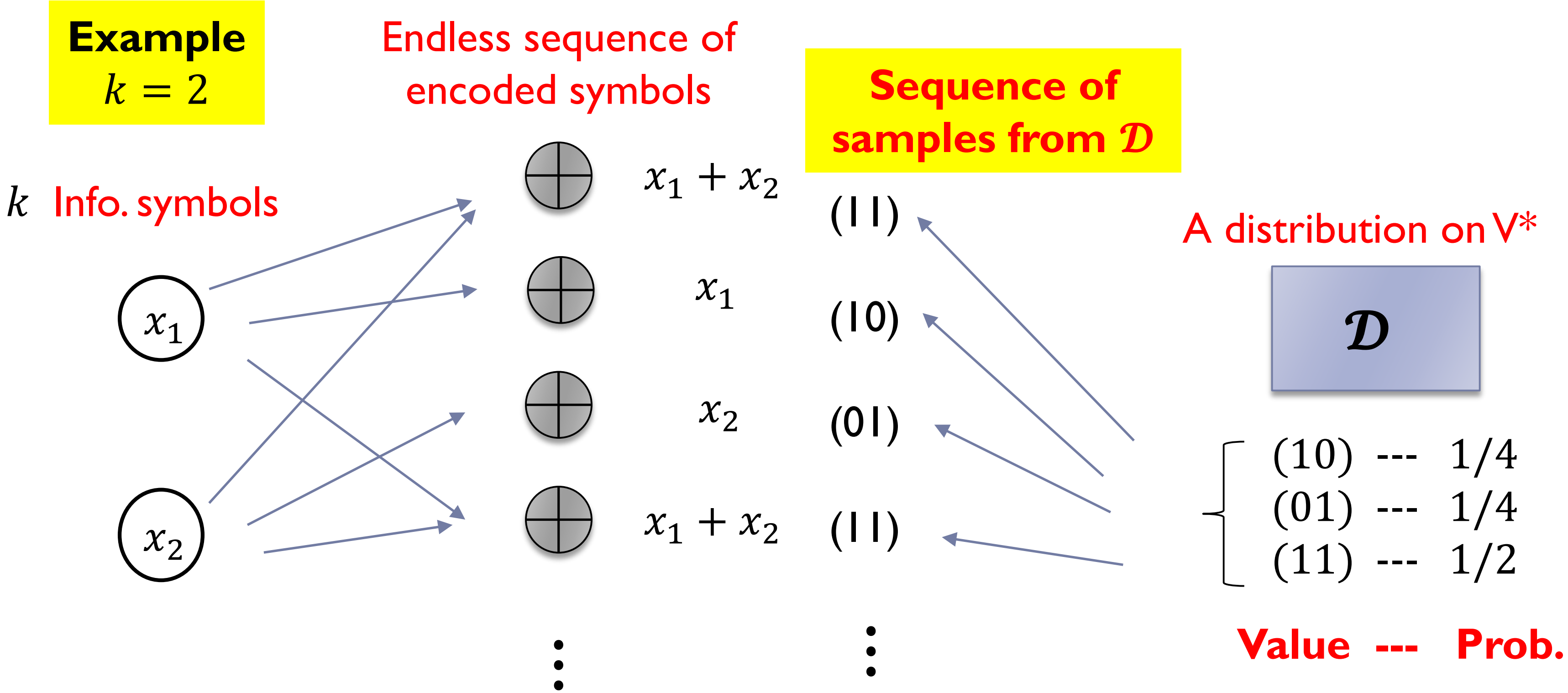
meaning that

Prob(01)=Prob(10)=1/4 and

Prob(11)=1/2

Encoding

- To construct a coded output symbol,
  1. sample independently from $\mathcal{D}$ and
  2. add input symbols corresponding to the sampled output
- Repeat this endlessly (rateless)



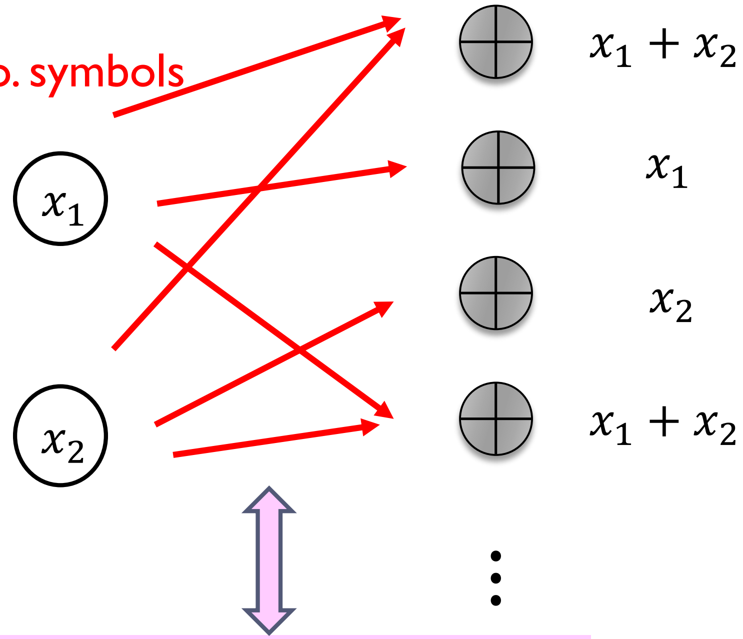
Encoding (alternative implementation)

Example
$k = 2$

Endless
encoded. symbols

Sequence of samples from the distribution

k Info. symbols



2
1
1
2
⋮

A distribution on $\{1,2,\ldots,k = 2\}$

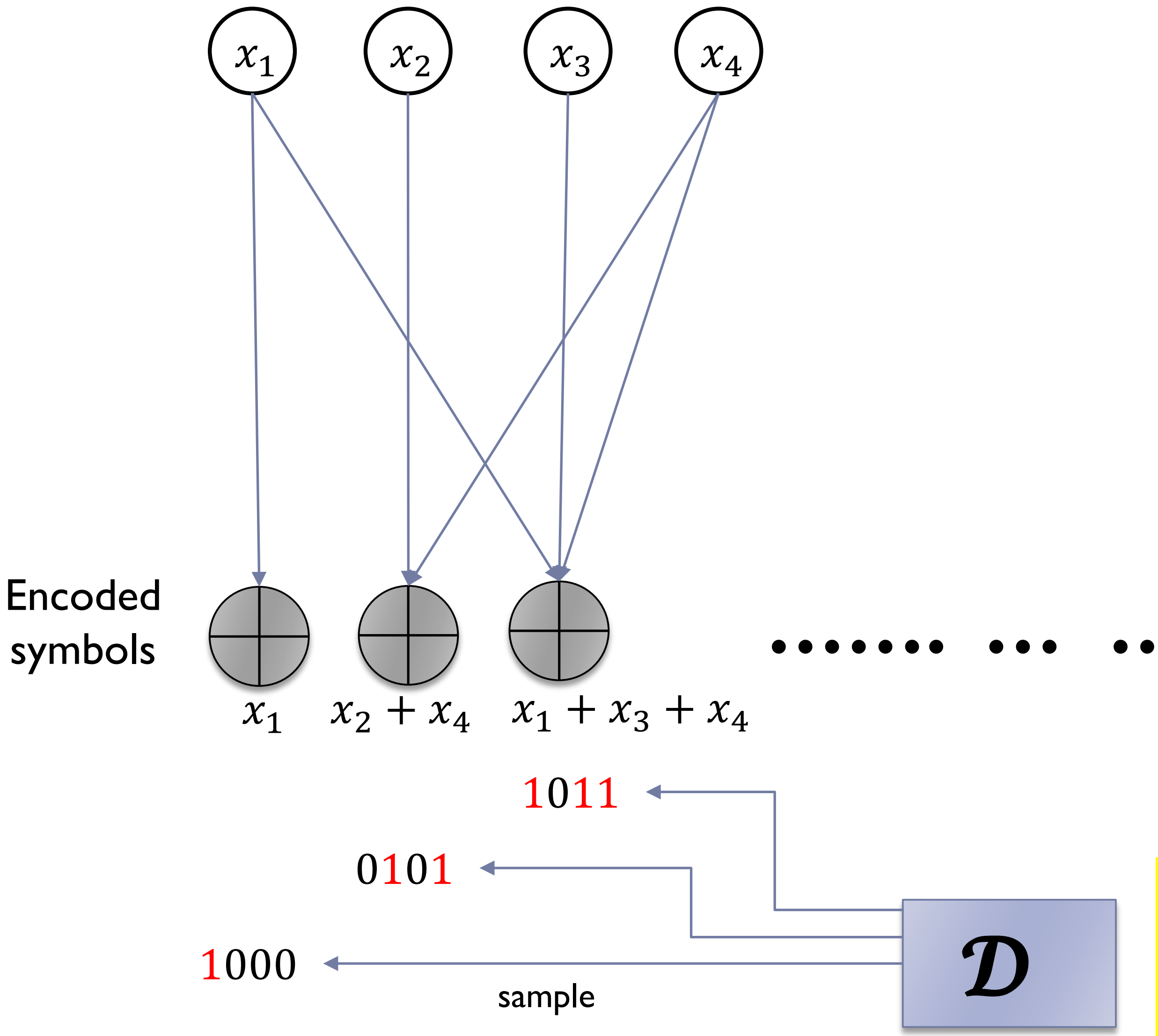
$$\begin{cases} 1 \text{ --- } 1/2 \\ 2 \text{ --- } 1/2 \end{cases}$$

Value --- Prob.

Choose uniform-randomly from the symbols according to the sample

The randomness is moved from the distribution $\mathcal{D}$ to here

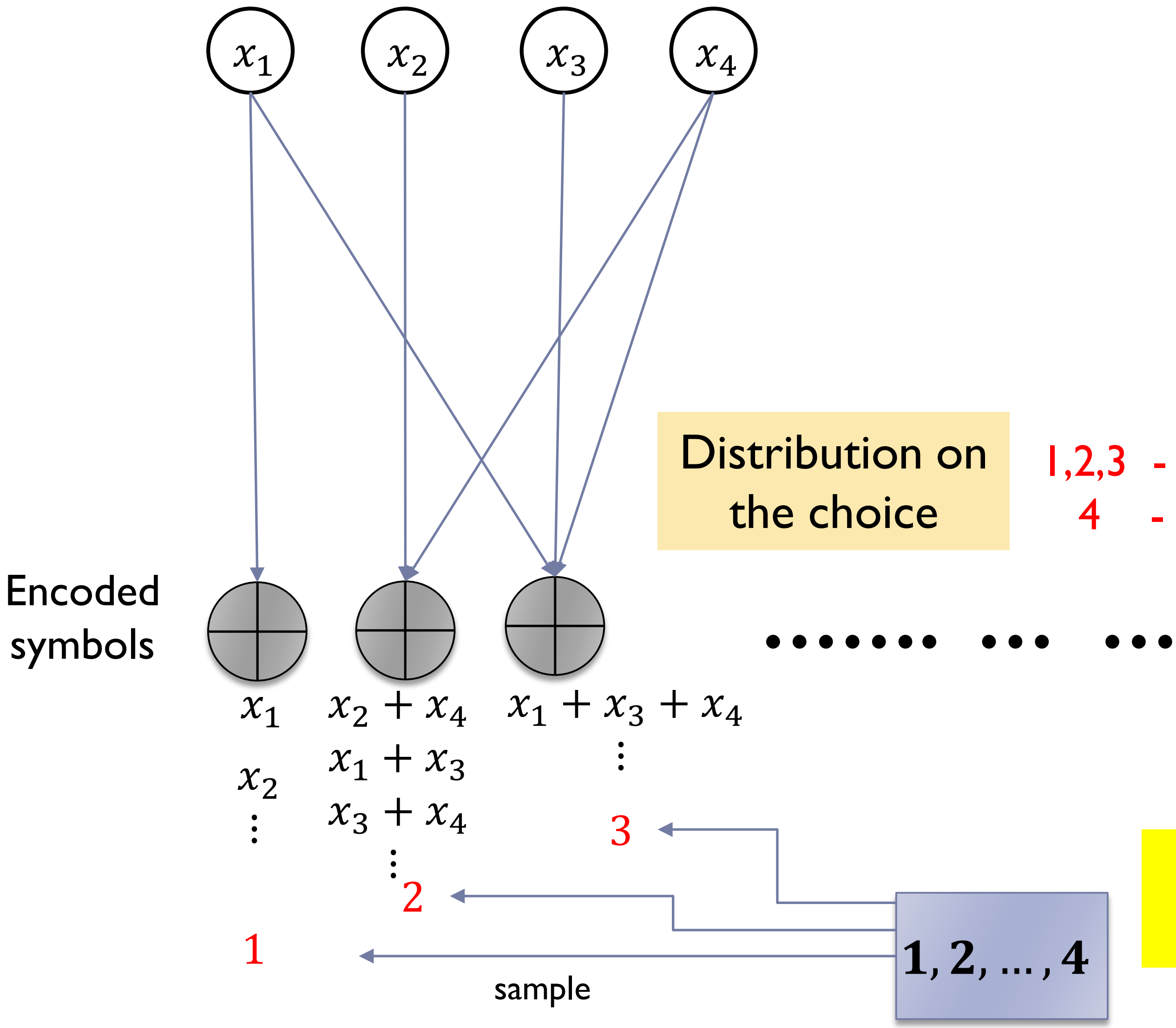
Example for $k = 4$



1000	1/32
0100	1/32
0010	1/32
0001	1/32
1100	1/16
0110	1/16
0011	1/16
1010	1/16
0101	1/16
1001	1/16
1110	1/16
1101	1/16
1011	1/16
0111	1/16
1111	**4/16**

A given distribution on binary 4-tuples

Example for $k = 4$ (alternative way)



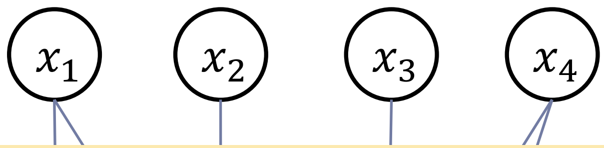
Distribution on the choice

1,2,3 - uniform random for 4,6,4
4 - only one choice

1	2/16
2	6/16
3	4/16
4	**4/16**

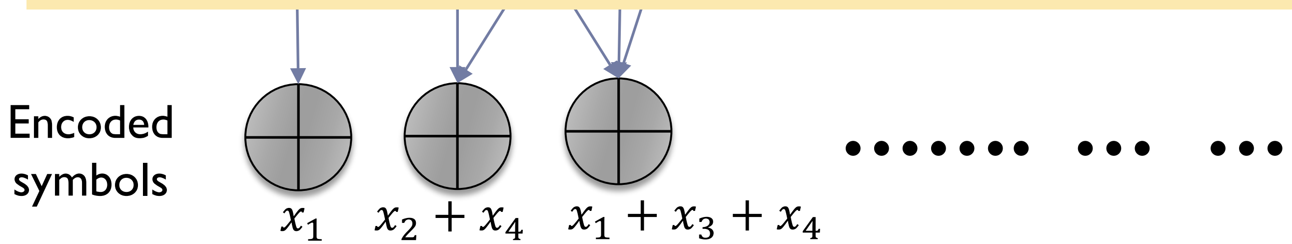
A distribution on {1,2,3,4}

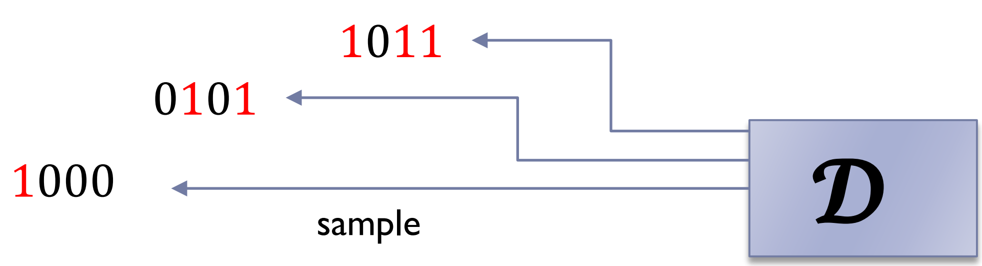
Key issues on design and implementation



Key 2 (implementation issue – tricky)

- Each tx-and-rx coded symbol must contain an indication of how it was generated, otherwise the received symbol is **useless**
- **In practice, this can be done by indicating a seed for the random number generators shared by tx and rx**





Key 1 (design issue)

how to design $\mathcal{D}$?

Decoding is to solve some **simultaneous linear equations** over $\mathbb{F}_2$

$$x_1 + x_2 = c_1$$

$$x_1 = c_2$$

$$x_2 = c_3$$

$$x_1 + x_2 = c_4$$

$$\vdots$$

Determine

$x_1 = ?$

$x_2 = ?$

- Best known algorithm?
  - **Gauss Elimination Algorithm (200 years old) in quadratic time**
- Here, we have to do much faster.

Performance Measures

- Encoding cost
  - The expected encoding cost of a fountain code with parameters $(\mathcal{D}, k)$ is **the expectation of the weight function under $\mathcal{D}$**
    $$E_{\mathcal{D}}[weight(x)]$$
  - This corresponds to the expected **per-symbol cos**t of encoding
  - **The best encoding cost** is constant $O(1)$
- Decoding cost
  - The expected decoding cost of a fountain code with parameters $(\mathcal{D}, k)$ **using a specific decoding algorithm** is **the expected number of arithmetic operations** (i.e., additions over $\mathbb{F}_2$) that the algorithm uses to decode the source symbols
  - **The best decoding cost** is linear in k, i.e., $O(k)$
- Overhead (reception overhead)
  - ε is the overhead if decoding can be done from any set of $k(1+\varepsilon)$ output symbols with high probability

Some History

- Fountain codes were stipulated by **Byers et al in 1998**, and their applications discussed. A construction was, however, not given.

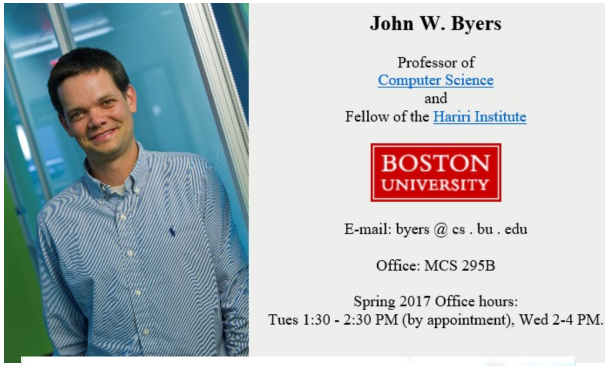
Byers, J. W., Luby, M., Mitzenmacher, M., & Rege, A. (1998). A digital fountain approach to reliable distribution of bulk data. *ACM SIGCOMM Computer Communication Review*, *28*, 56–67.

- First construction of efficient Fountain codes was given by **Luby in 1998** (published in 2002).

Luby, M. (2002). Lt codes. In *Annual symposium on foundations of computer science, 2002* (pp. 271–280).

- **Raptor codes** were invented motivated by the objective of improving the encoding and decoding complexity (published in 2006).

Shokrollahi, A. (2006). Raptor codes. *IEEE Transactions on Information Theory*, *52*(6), 2551–2567.



APPROXIMATIONS OF DIGITAL FOUNTAIN

REED-SOLOMON CODES
TORNADO CODES

Not rateless = With "rate"

LT CODES
RAPTOR CODES

Rateless approach in theory, but there is a "rate" in practice (overhead)

Reed-Solomon Codes

RS codes (over BEC) is the first example of "**fountain-like**" codes

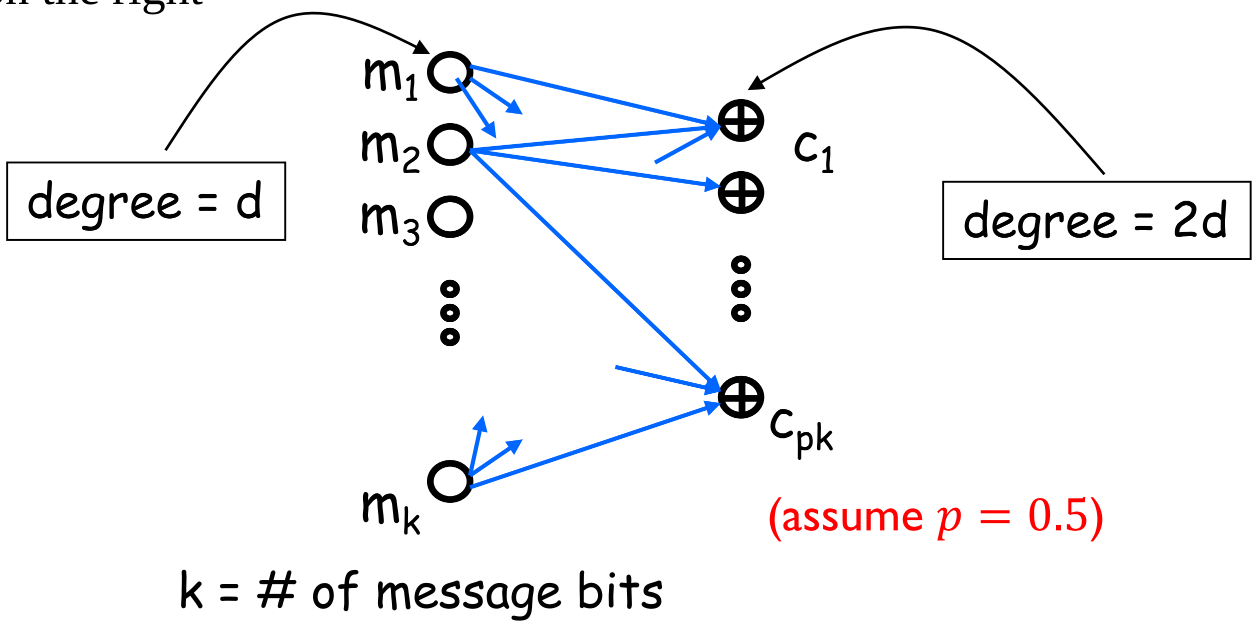
- Optimal overhead
  - A message of k symbols can be recovered from any $n = k + r$ encoding symbols
- Dense systems of linear equations
  - Poor encoding cost (linear in k)
  - Poor decoding cost (quadratic in k)
- Limitation on the number of distinct encoding symbols
  - The field size gives a limitation on the number of distinct encoding symbols that can be generated
  - Larger fields introduce nontrivial overhead for the resulting field arithmetic operations

Tornado Codes [1997]

- Sparse systems of equations
  - Large encoding/decoding cost for RS codes arises from the dense system of linear equations
  - **Fast encoding and decoding (linear in *k*)**
- Suboptimal overhead
  - The price we pay for much faster encoding and decoding is that k packets no longer suffice to reconstruct the source data
- Challenges for Tornado Codes
  - Designing the proper structure for the system of equations so that
    - The number of additional packets is small
    - The encoding/decoding costs are small

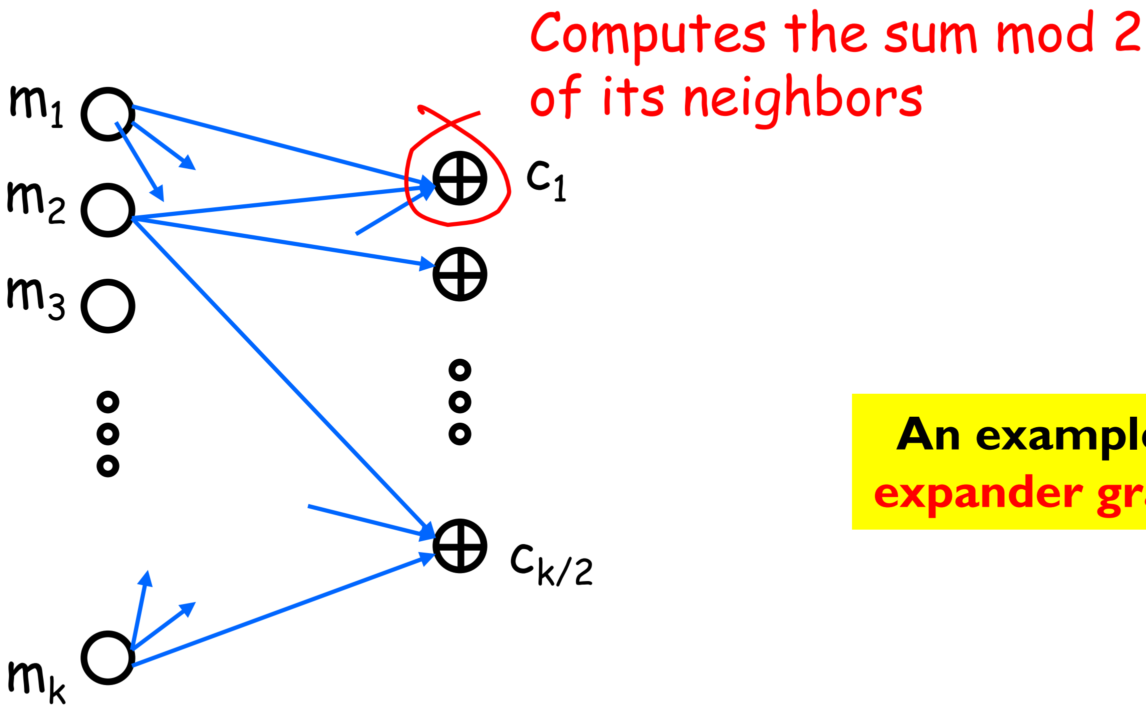
Tornado Codes

- Random erasure channel model (BEC)
  - Each bit is lost independently with a fixed (given) probability μ
  - We know the positions of the lost bits
- We want a binary linear block code with **rate $(1-p)$** that can correct $(1-\varepsilon)p$ **fraction** of the **erasures**
  - This is a binary $[n, (1-p)n, (1-\varepsilon)pn+1]$ code
- We will use ***d*-regular bipartite graphs** with k nodes on the left and pk on the right



Tornado Codes: Encoding

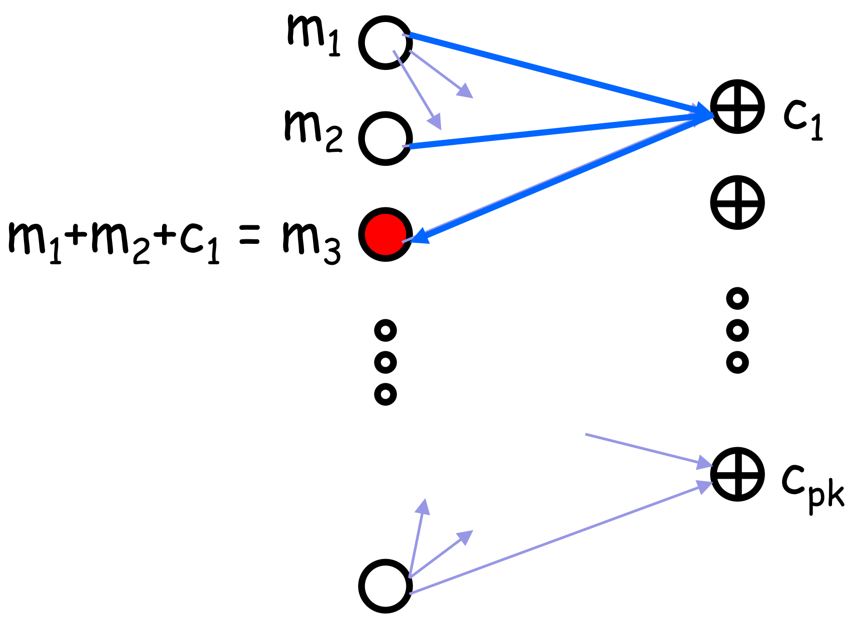
- Linear time encoding



An example of
expander graphs

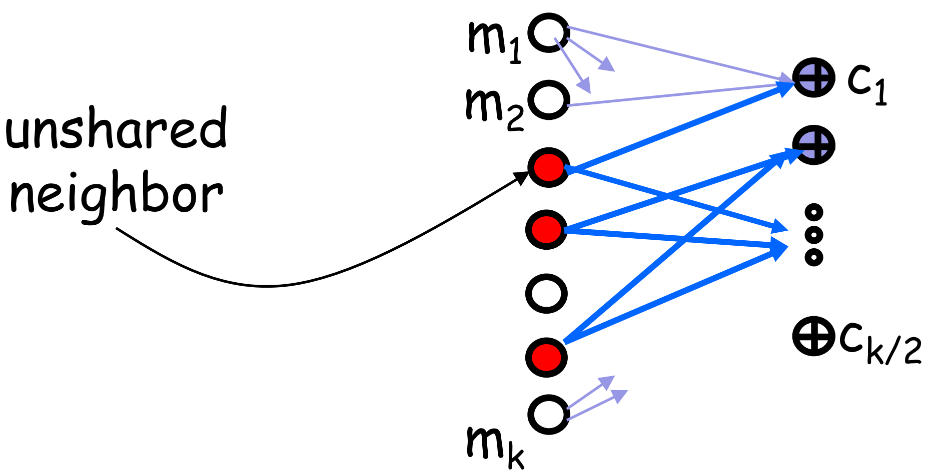
Tornado Codes: Decoding

- Assume that **all the check bits are intact** **(NOT in ERROR)**
- Find a check bit c_i such that only one of its neighbors is erased
  - an unshared neighbor
- Fix the erased symbol, and repeat



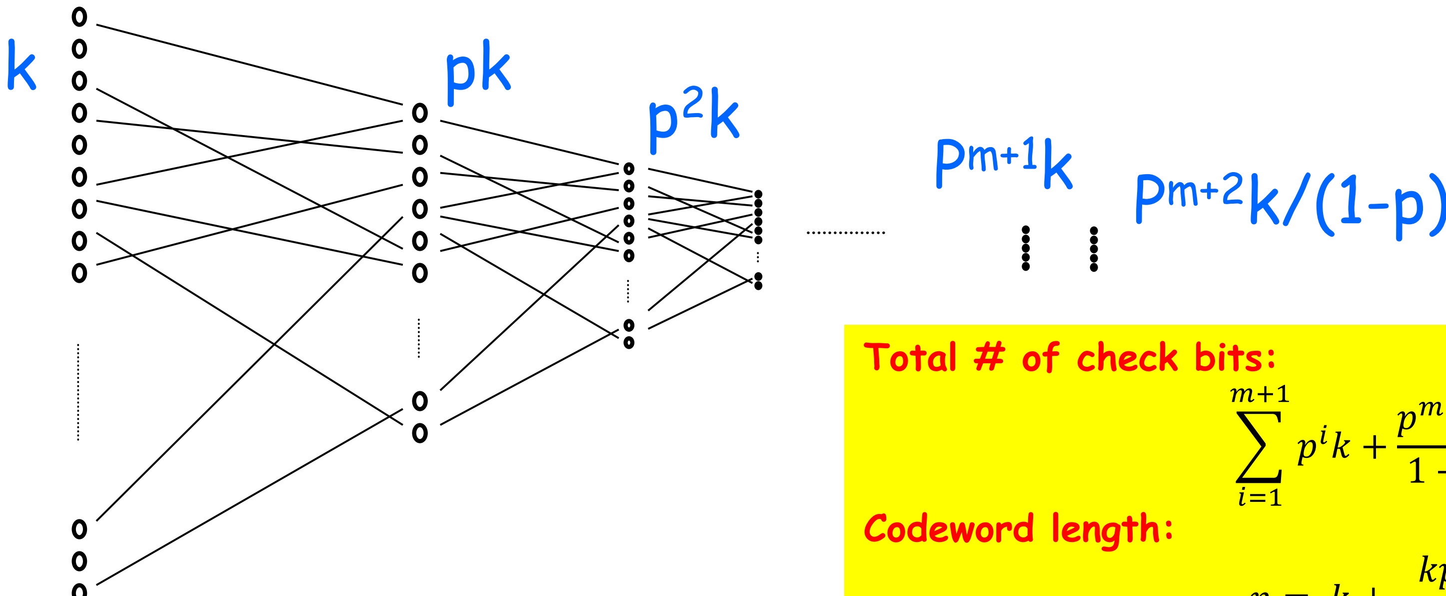
Tornado Codes: Decoding

- We need to always find such a check bit with the unshared-neighbors property
- Consider the set of corrupted message bits and their neighbors
  - At least one check bit has an unshared neighbor
- Can we always find unshared neighbors?



Tornado Codes: Decoding

- What if check bits are erased?
  - Use another bipartite graph to construct another level of [check bits for the check bits]()
  - Final level is encoded using RS or some other (linear block) code



Total # of check bits:

$$\sum_{i=1}^{m+1} p^i k + \frac{p^{m+2}k}{1-p} = \frac{kp}{1-p}$$

Codeword length:

$$n = k + \frac{kp}{1-p} = \frac{k}{1-p}$$

Rate:

$$\frac{k}{n} = (1-p)$$

Tornado Codes - summary

- Optimal degree sequences
  - The decoding algorithm is equivalent to finding a node of degree one on the right, and then removing it, its neighbor, and all edges adjacent to the neighbor from the sub-graph
  - Repeat this until **no nodes of degree one are available at every step of decoding**
  - The optimal degree distributions are designed in such a way that there are a small number of degree one right nodes available at every time step
- Better approximation to digital fountains than Reed-Solomon codes
  - Linear time encoding/decoding
- Still suffer from a powerful drawback in that the code is not rateless

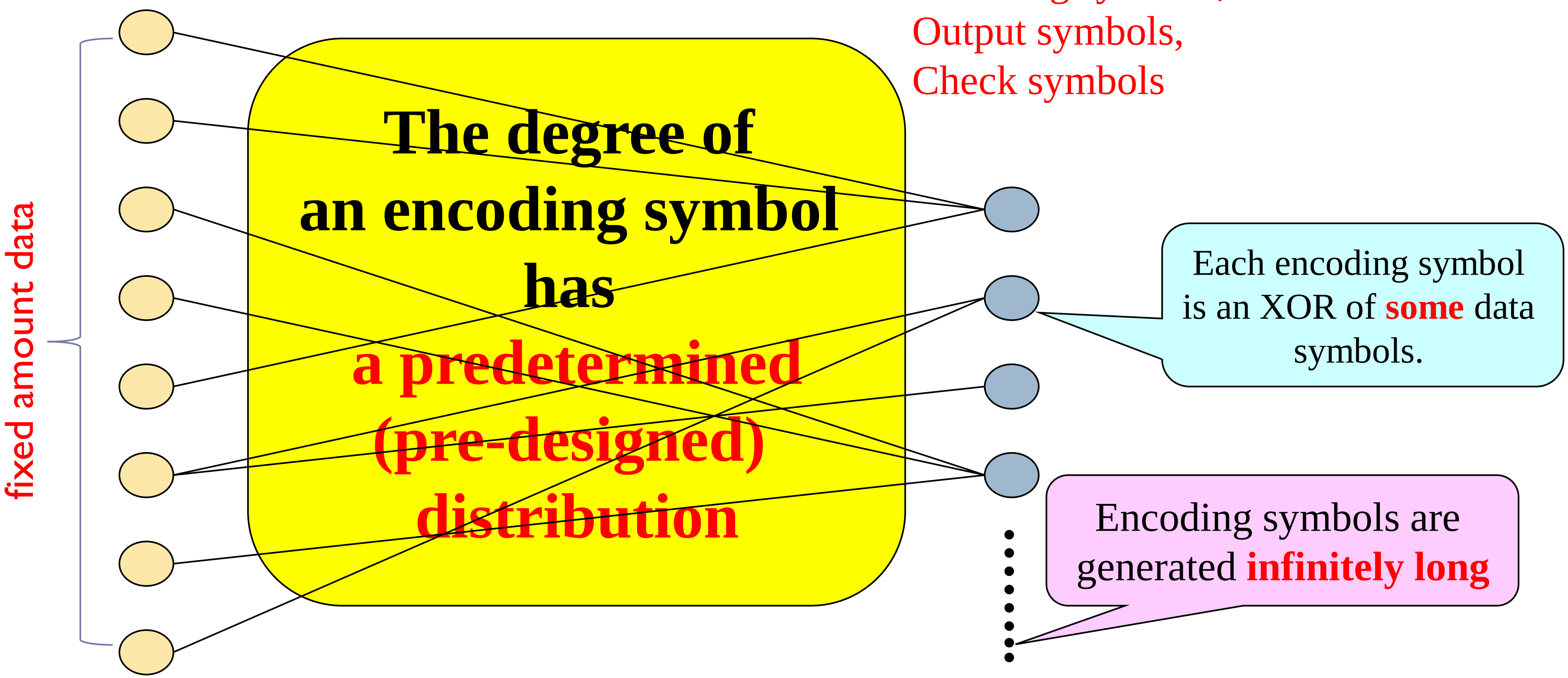
LT Codes [2002]

- The **first practical realization of rateless codes**
- Advantage over Tornado codes
  - With Tornado codes, even after designing the degree distribution, some care must be taken to design the actual graph used as well
  - With LT codes, there is no explicit graph to optimize
- **Near-ideal digital fountain**
  - **Encoding** can be done **on the fly** in time proportional to $\ln k$
  - **Decoding** can be done in time proportional to $k \ln k$

LT Codes – encoding/decoding

Data symbols,
Input symbols,
Information symbols,
Source symbols

Encoding symbols,
Output symbols,
Check symbols



LT Codes – the best one could hope for

- Reduce **the average degree** to a constant, and **the decoding time** to $O(k)$

This cannot be done in the strict LT framework

  - Simply for every message node to have at least one neighbor when there are $O(k)$ encoding symbols, the average degree must be at least $\Omega(\ln k)$
- However, **using pre-coding**, the average degree can be reduced to a constant
  - This is the “Raptor code”

Raptor Codes [2006]

- Extend the idea of LT codes one important step farther

- First pre-code the message M by encoding it with a fixed erasure code
  - Now treat **the encoded version M′** as the message, so that encoding symbols are the XOR of packets of M′, in a manner similar to LT codes
  - Now, we just need to recover a constant fraction of the packets of M′
  - The bound on the average degree $\Omega(\ln k)$ no longer applies

- For any constant $\varepsilon > 0$ and sufficiently large k, the message M can be decoded **after receiving only $(1+\varepsilon)k$ packets with high probability**, with the degree of each encoding symbol being $O\left(\ln\left(\frac{1}{\varepsilon}\right)\right)$ and the total decoding time being $O\left(k\ ln\left(\frac{1}{\varepsilon}\right)\right)$

LT CODES

ENCODING

DECODING

DEGREE DISTRIBUTIONS

- **PRELIMINARY (BALLS-IN-BINS EXERCISE)**
- **IDEAL SOLITON DISTRIBUTION**
- **ROBUST SOLITON DISTRIBUTION**

M. Luby, "LT codes," presented at the Proc. IEEE Foundations of Computer Science (FOCS), 2002.

Encoding

- An $(\boldsymbol{k}, \boldsymbol{\rho}(\boldsymbol{x}))$ LT code
  - The number of input symbols is $\boldsymbol{k}$
  - The degree sequence is $(\rho_1, \rho_2, \ldots, \rho_k)$ $, where$ $\boldsymbol{\rho}(\boldsymbol{x}) = \sum_{i=1}^{k} \boldsymbol{\rho_i x^i}$

- **Any number of encoding symbols can be independently generated from $\boldsymbol{k}$ information symbols by the following encoding process:**
  1. Determine the degree $\boldsymbol{d}$ of an encoding symbol. The degree is chosen at random from a given degree distribution $\boldsymbol{\rho}(\boldsymbol{x})$.
  2. **Choose** d distinct information symbols uniformly at random. They will be neighbors of the encoding symbol.
  3. Assign the XOR of the chosen d information symbols to the encoding symbol

- The degree distribution $\boldsymbol{\rho}(\boldsymbol{x})$ determines the **performance** of LT codes, such as the number of encoding symbols for successful decoding (overhead).

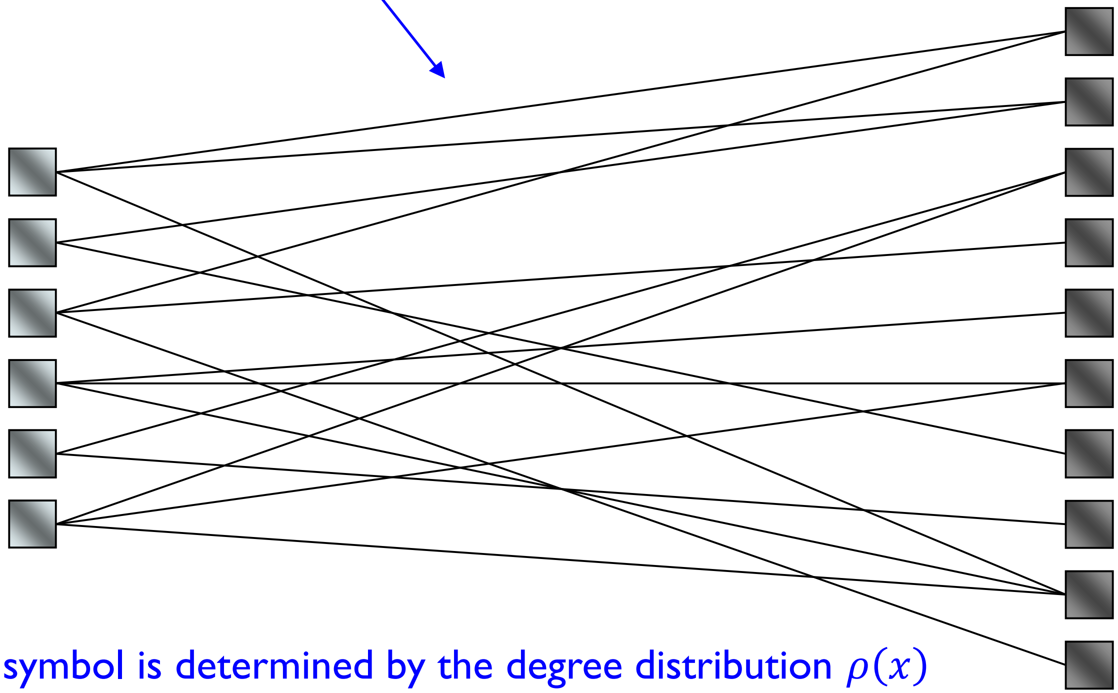
Encoding

※ The receiver must know **the encoding graph G** that is used by the sender

- Encoding graph G

degree distribution $\rho(x)$

d	ρ_d
1	0.5
2	0.4
3	0.1
4	0
5	0
$6 = k$	0



The degree d of an encoding symbol is determined by the degree distribution $\rho(x)$

d	2	2	2	1	1	2	1	1	3	1
v	(101000)	(110000)	(000011)	(001000)	(000100)	(000101)	(010000)	(000010)	(100101)	(001000)

Every vector of weight d is chosen with probability $\frac{\rho_d}{\binom{k}{d}}$

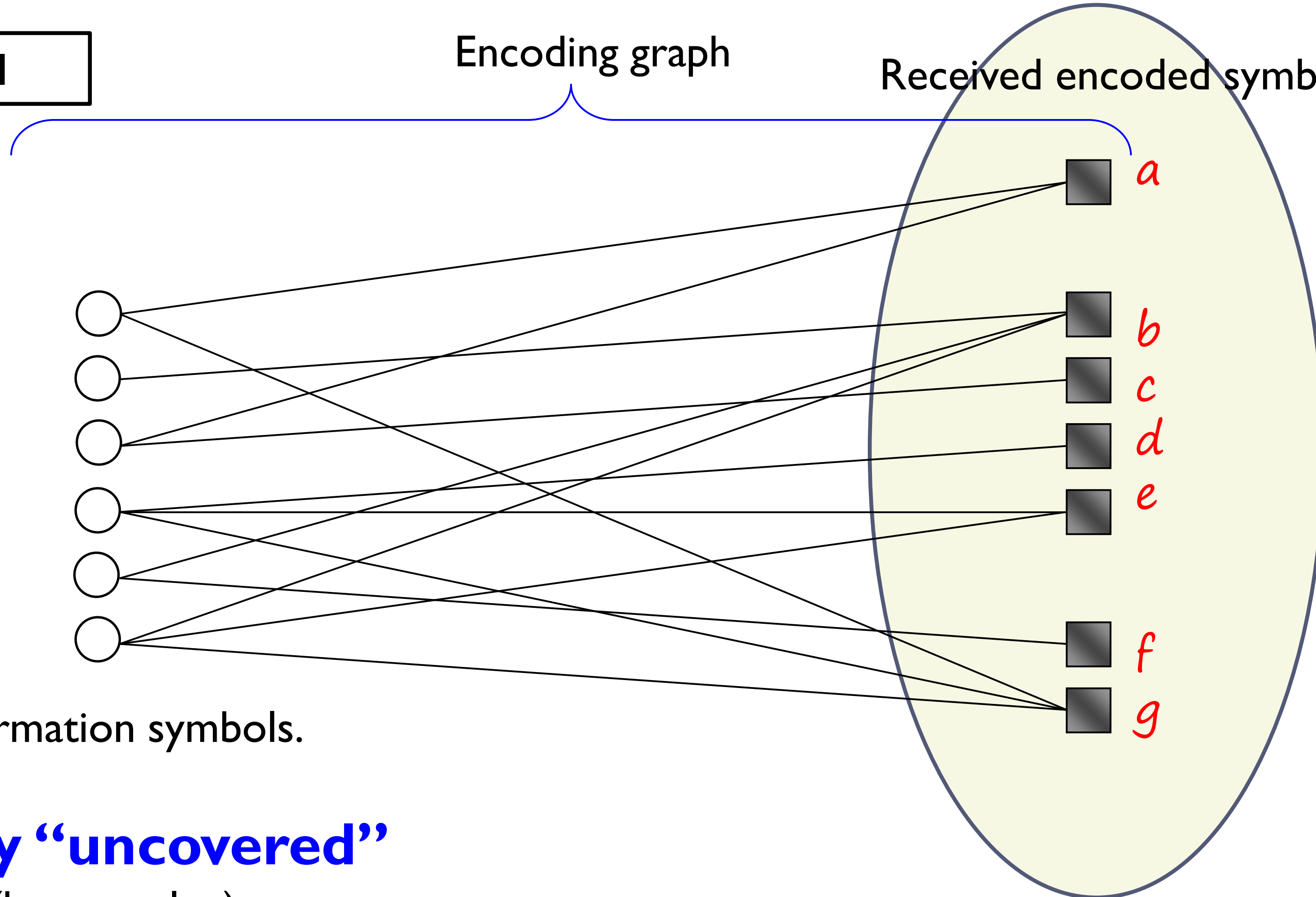
Decoding

I will go through some examples first,

and then, summarize the process

Decoding

Example 1



Information symbols.

initially "uncovered"

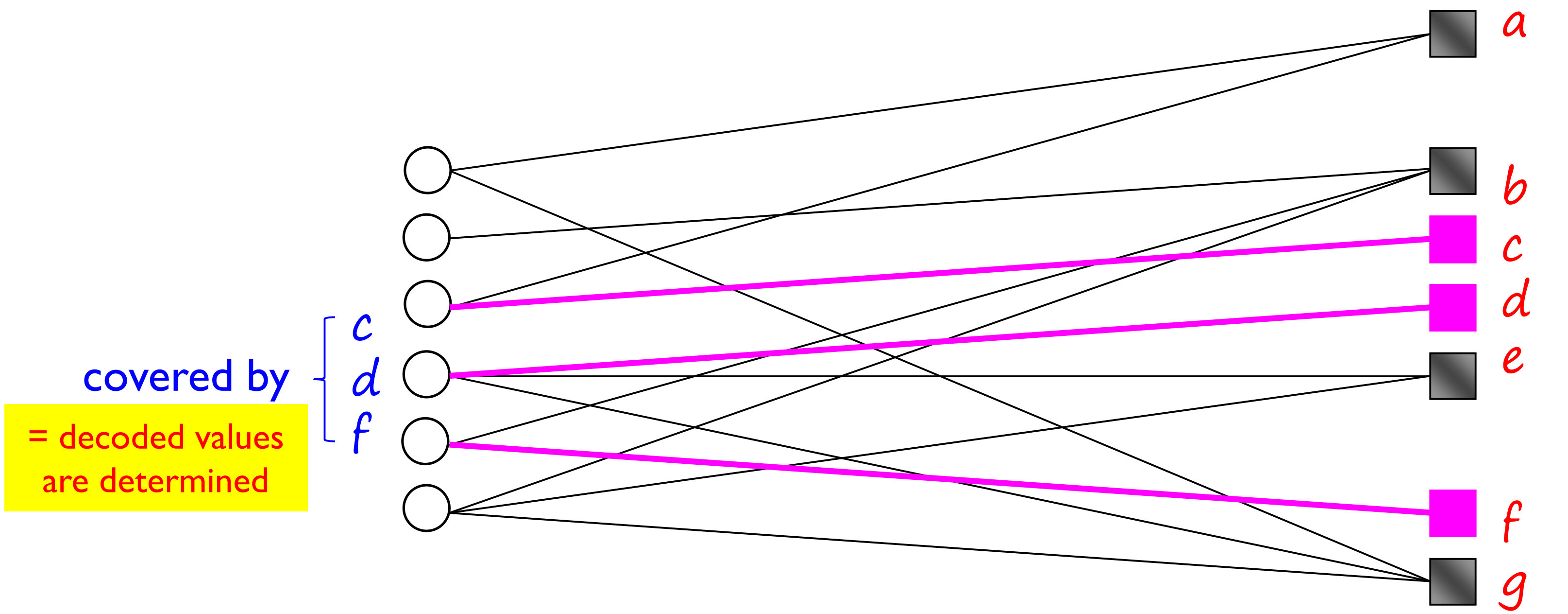
(by any value)

= no values are assigned

Decoding

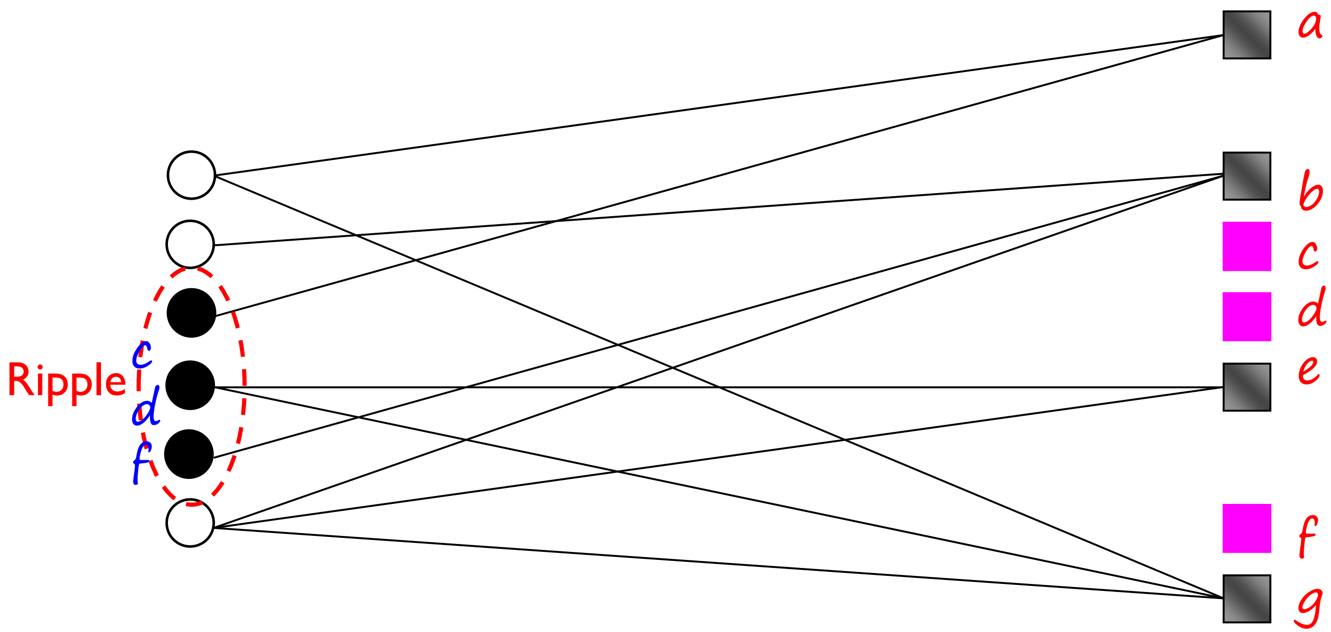
Example 1

Find **all** the encoding symbols of **degree 1**.
Here, there are 3 such symbols.
"Release" them to **"cover"** the info symbols.



Decoding

Example 1



These three information symbols are now **"covered"**

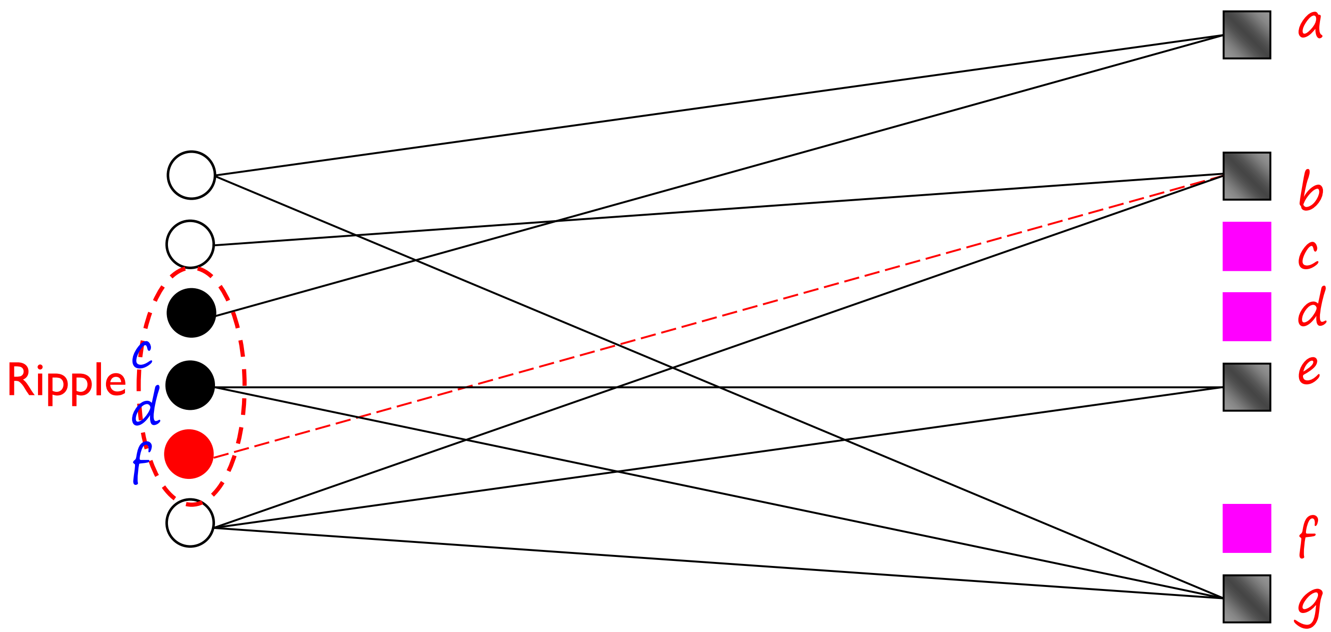
(= the decoded values are determined)

Then, these three information symbols (covered but **not yet processed**)

are put into the set, called **"the Ripple"**

Decoding

Example 1

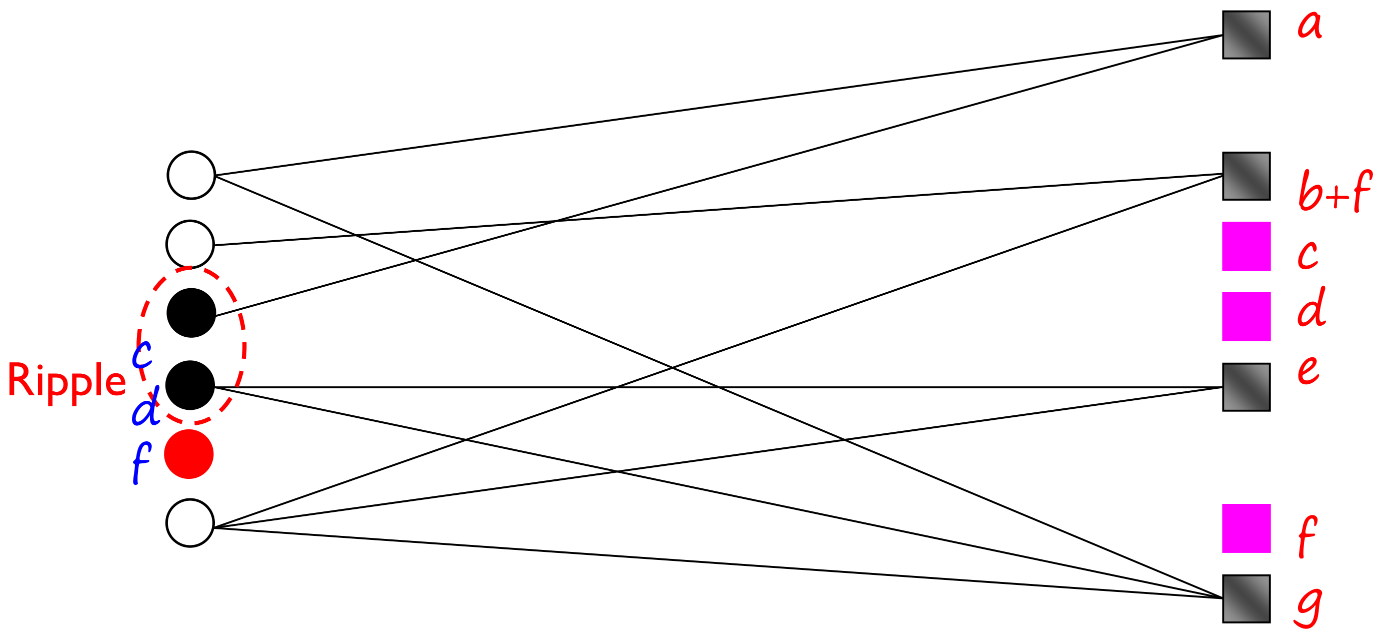


Any one information symbol in a Ripple
is **chosen** to be **"processed"**

All the neighbor encoding symbols are updated
and the corresponding edges are removed.
The processed info symbol is removed from the Ripple.

Decoding

Example 1

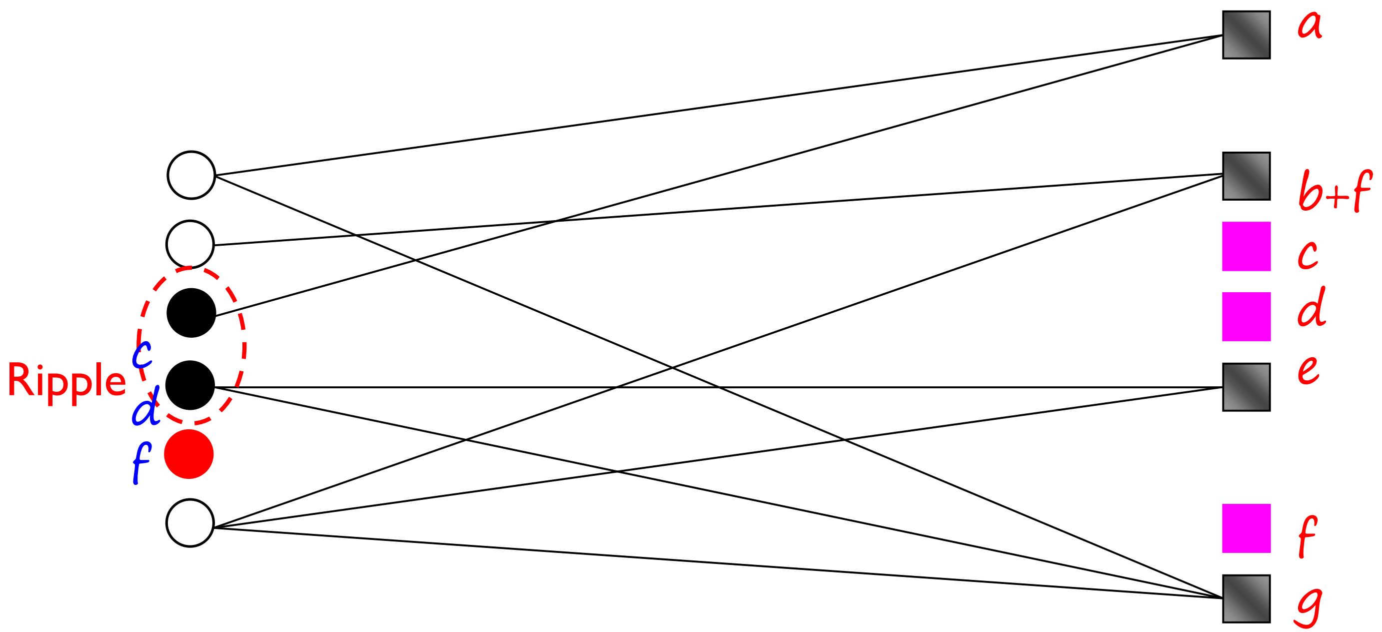


1 information symbol in the Ripple is processed.

In the Ripple, 2 information symbols are remained

Decoding

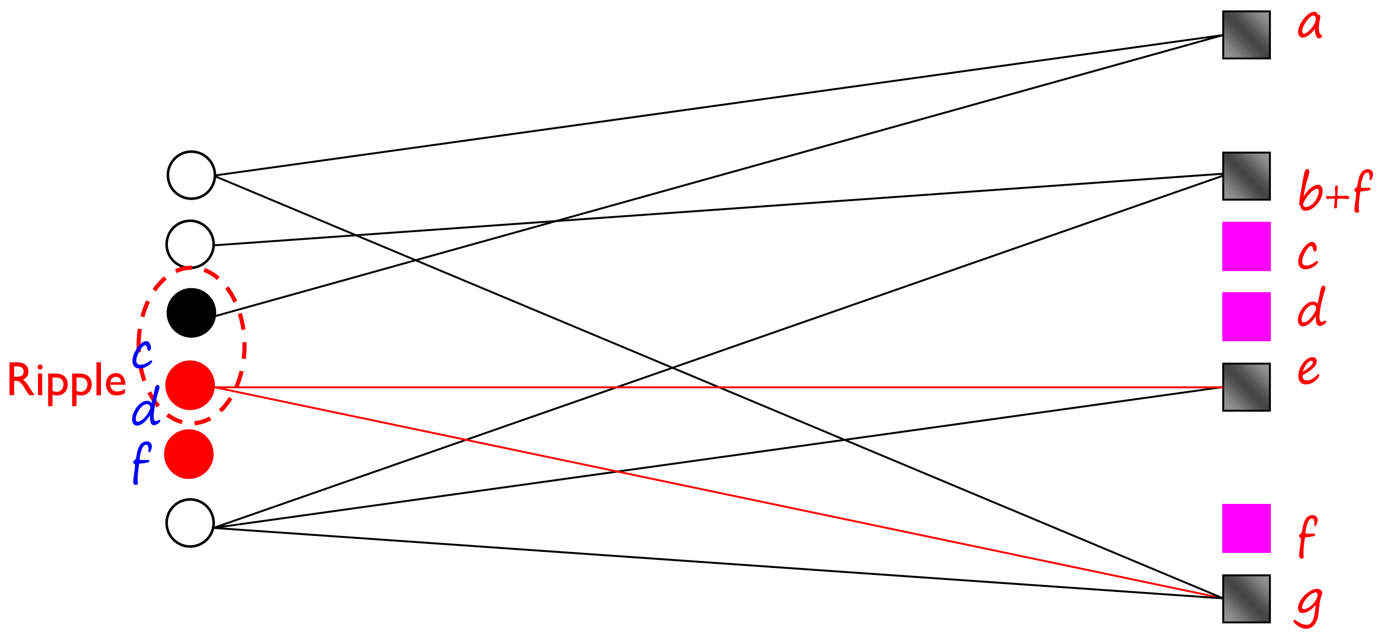
Example 1



No encoding symbol has degree one

Decoding

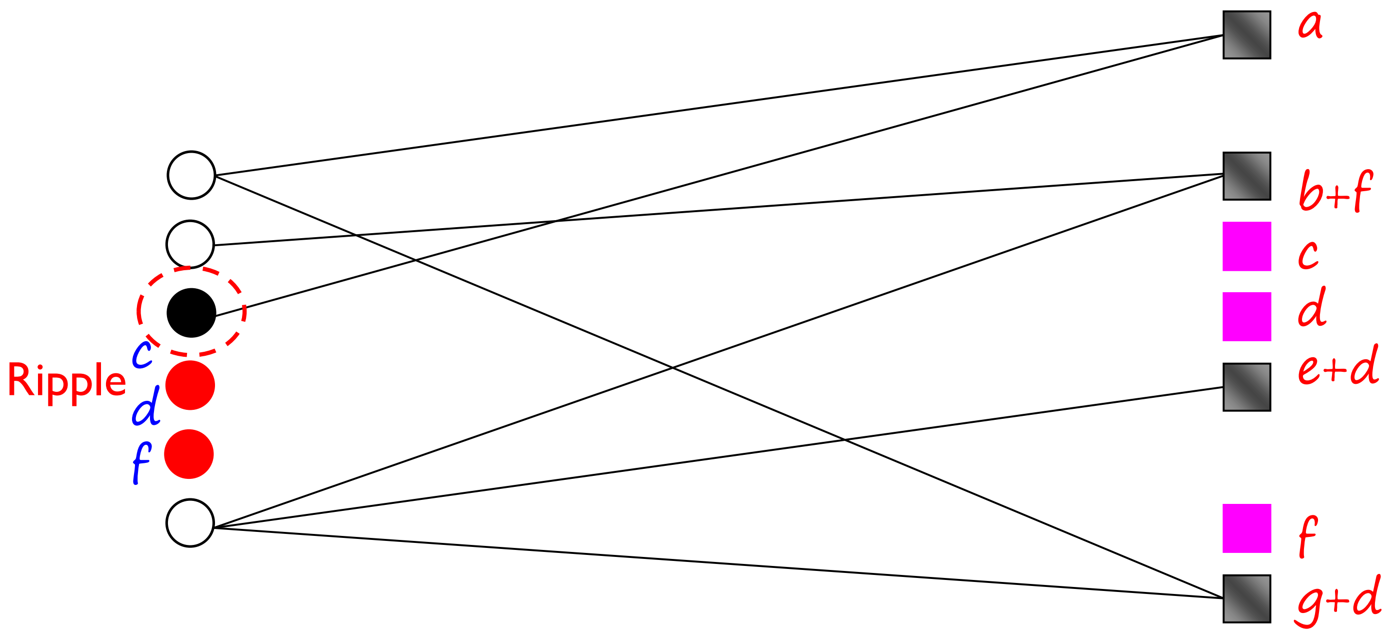
Example 1



Any one information symbol in the Ripple is AGAIN chosen to be **processed**

Decoding

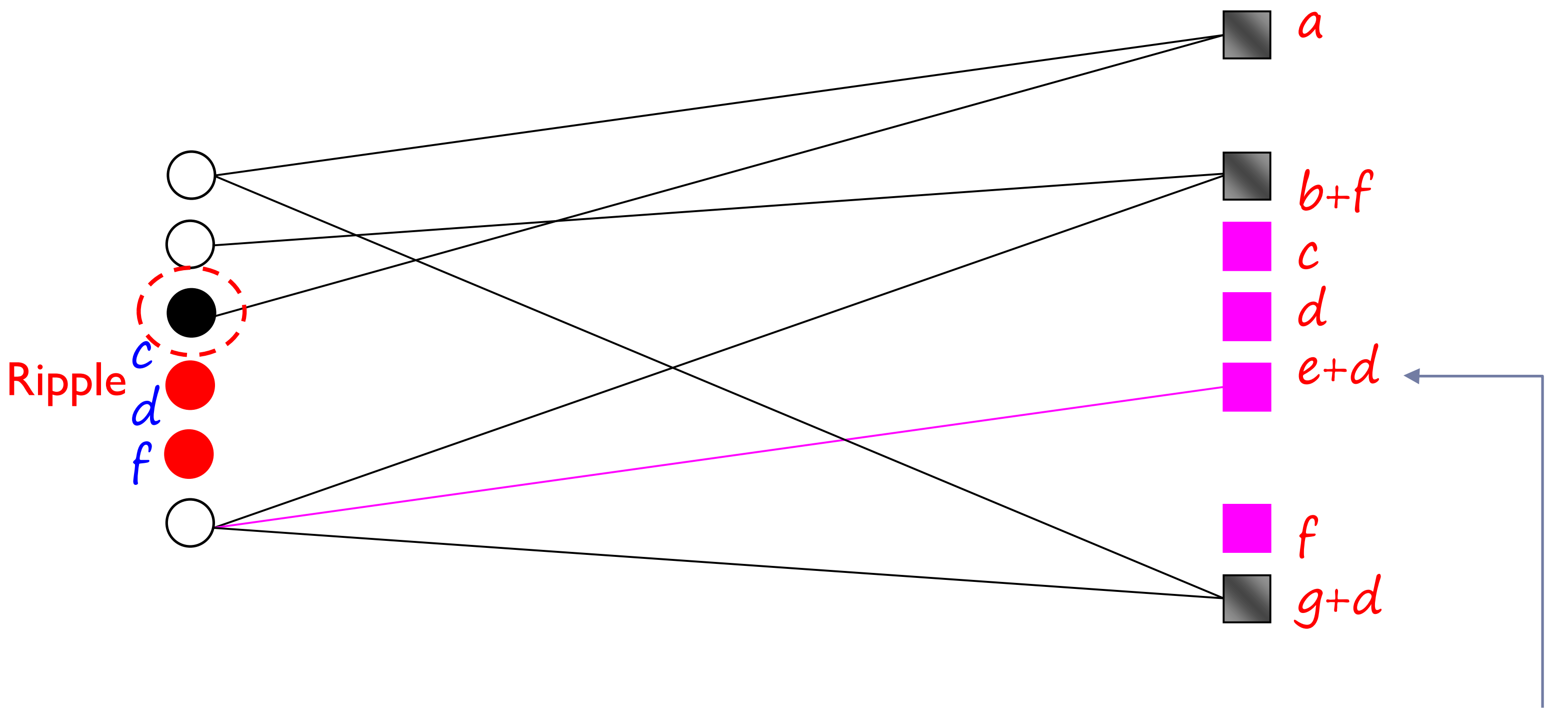
Example 1



The only one remaining information symbol in the Ripple is **processed**

Decoding

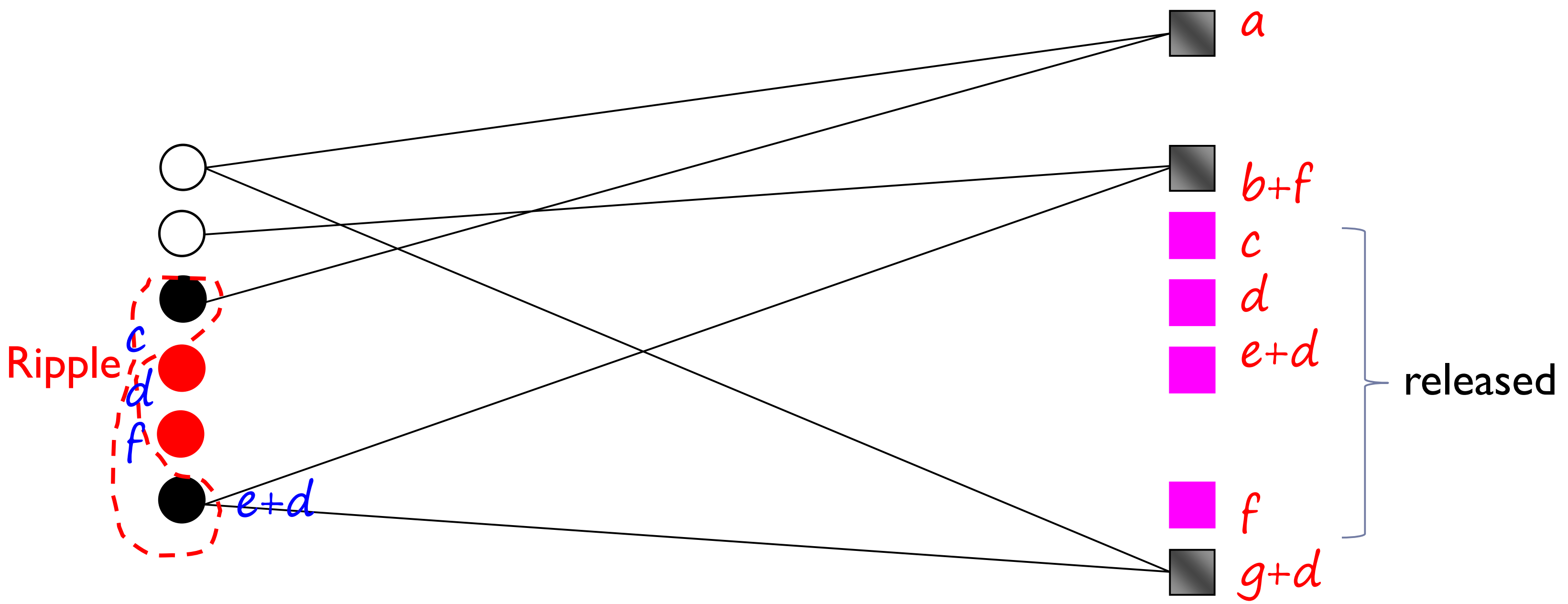
Example 1



Release the encoding symbol of degree 1 to **cover** the neighbor symbol

Decoding

Example 1

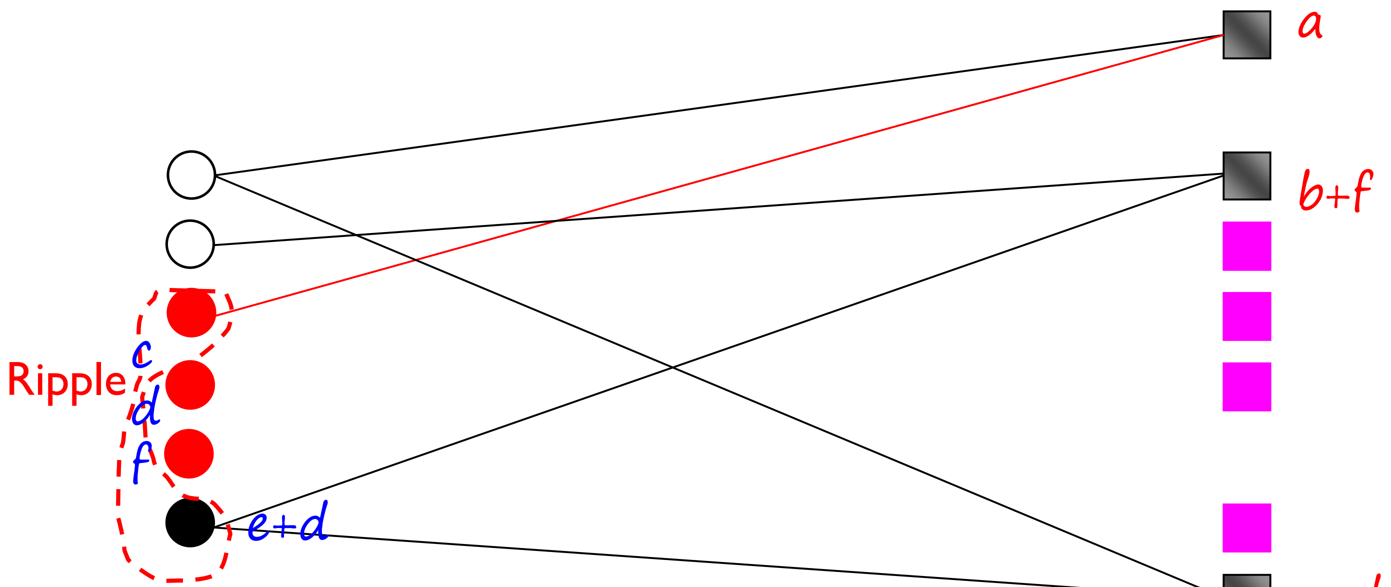


One more information symbol is **covered** (by the value $e+d$)

Then, this information symbol (covered but not yet processed) is put into **the Ripple**

Decoding

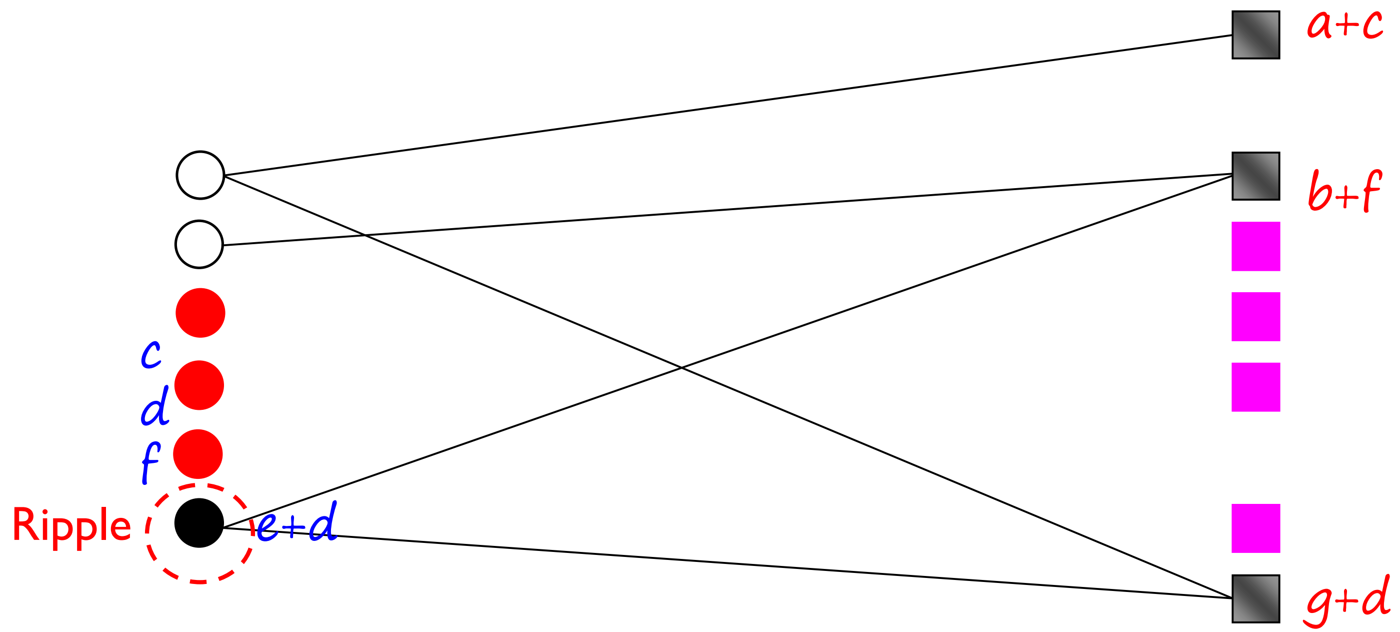
Example 1



1 information symbol in a Ripple is chosen to be **processed**

Decoding

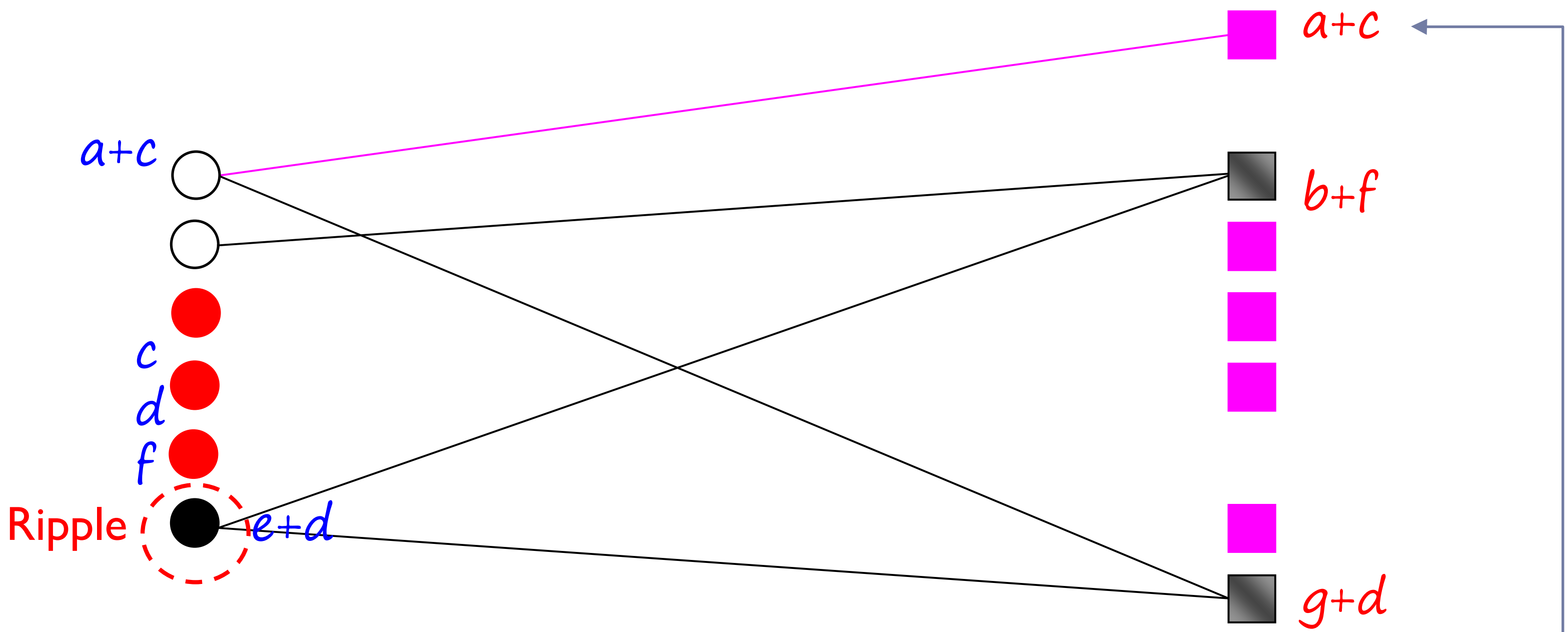
Example I



1 information symbol in a Ripple is processed
In the Ripple, 1 information symbol is remained

Decoding

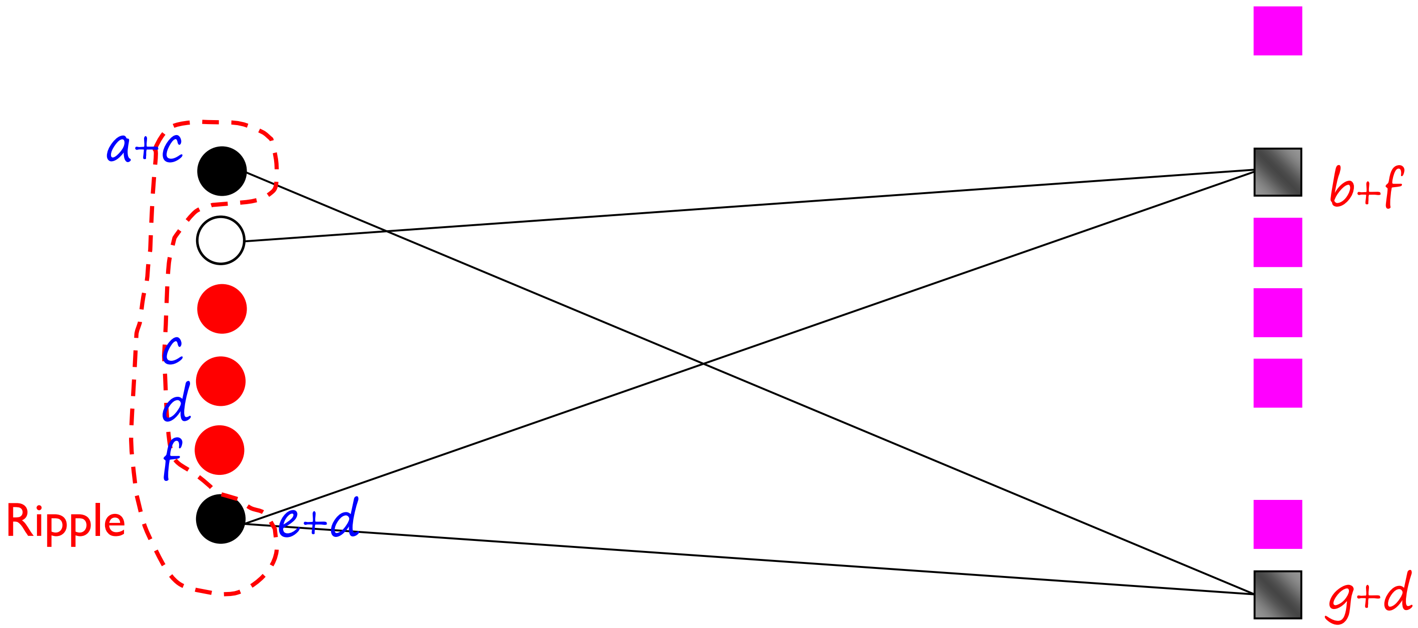
Example 1



Release the encoding symbol of degree 1

Decoding

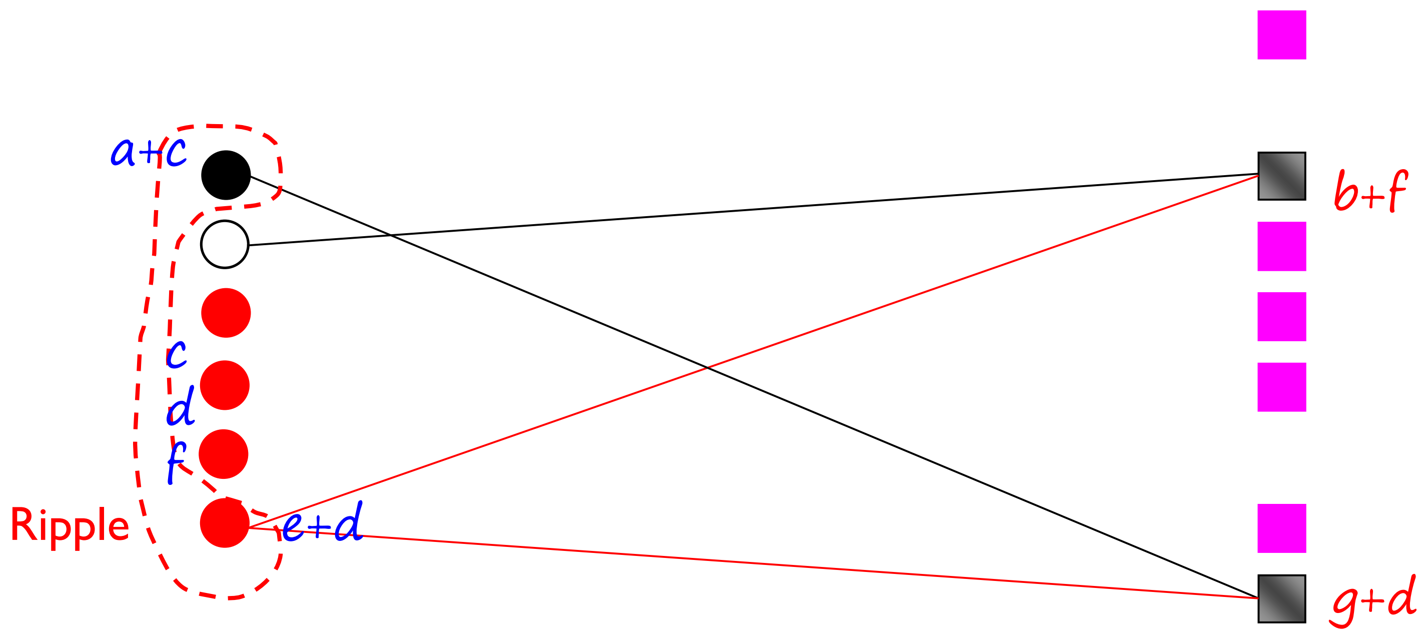
Example 1



One more information symbol is covered
Then, this information symbol is put into the Ripple

Decoding

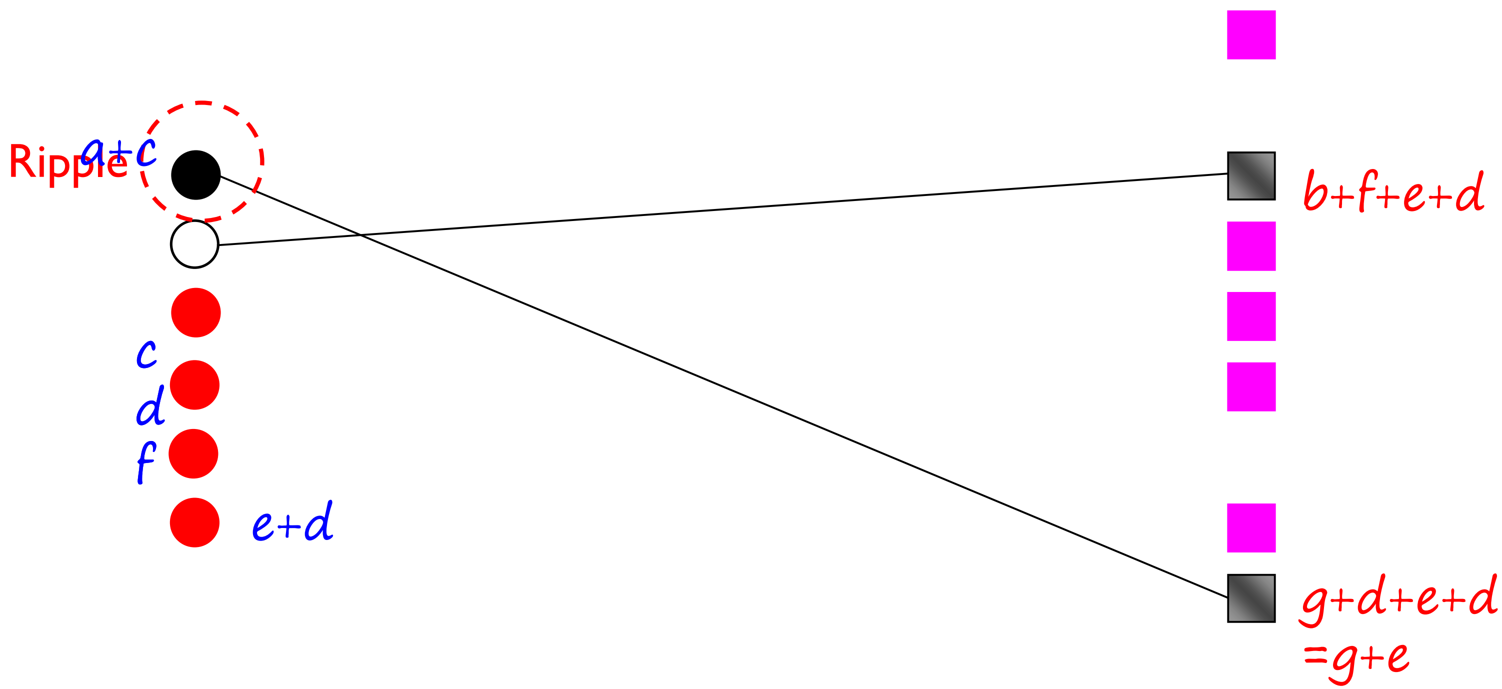
Example 1



1 information symbol in a Ripple is chosen to be processed

Decoding

Example I



1 information symbol in a Ripple is processed
In the Ripple, 1 information symbol is remained

We must have that

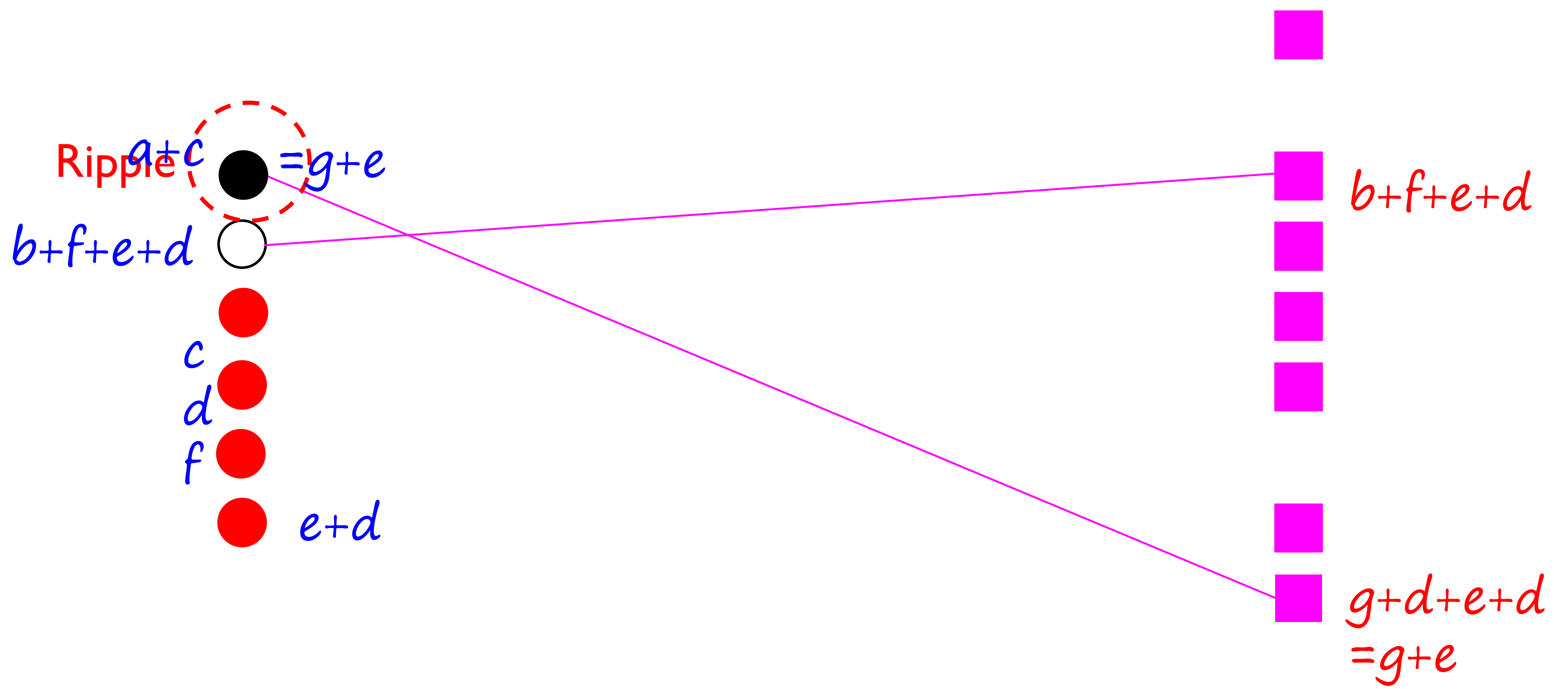
$g+e=a+c$

Otherwise, decoding fails!!

This must always be true since we assume BEC

Decoding

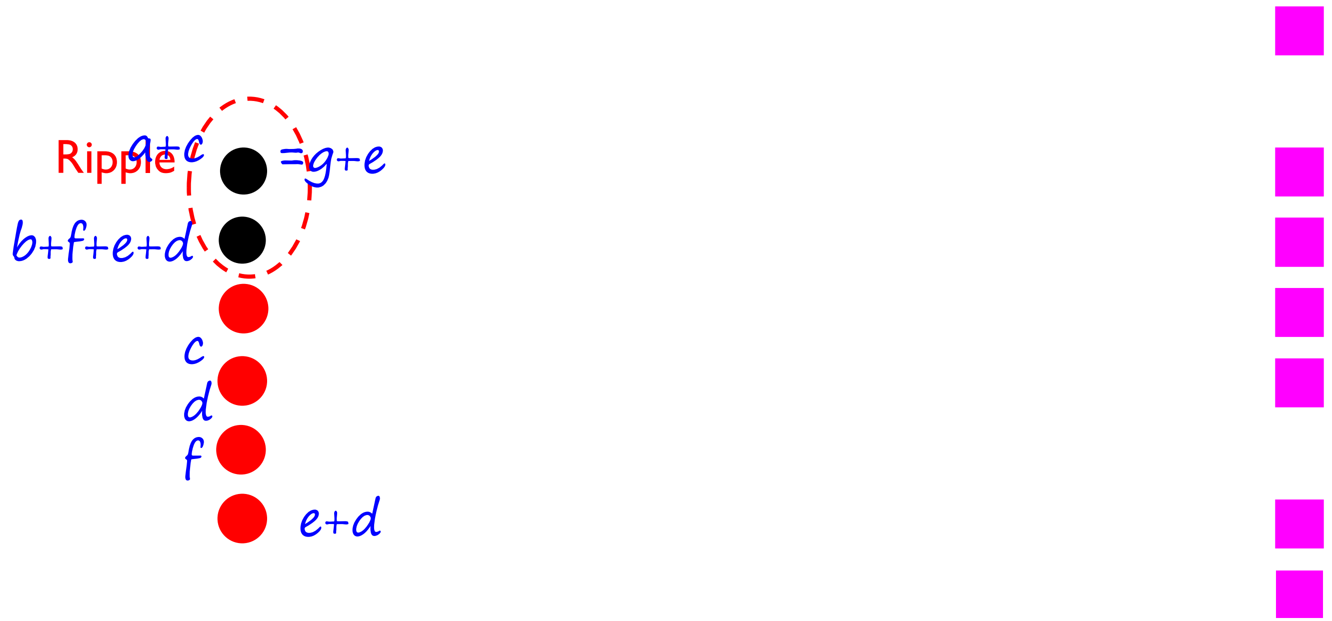
Example I



Release encoding symbols of degree 1

Decoding

Example 1



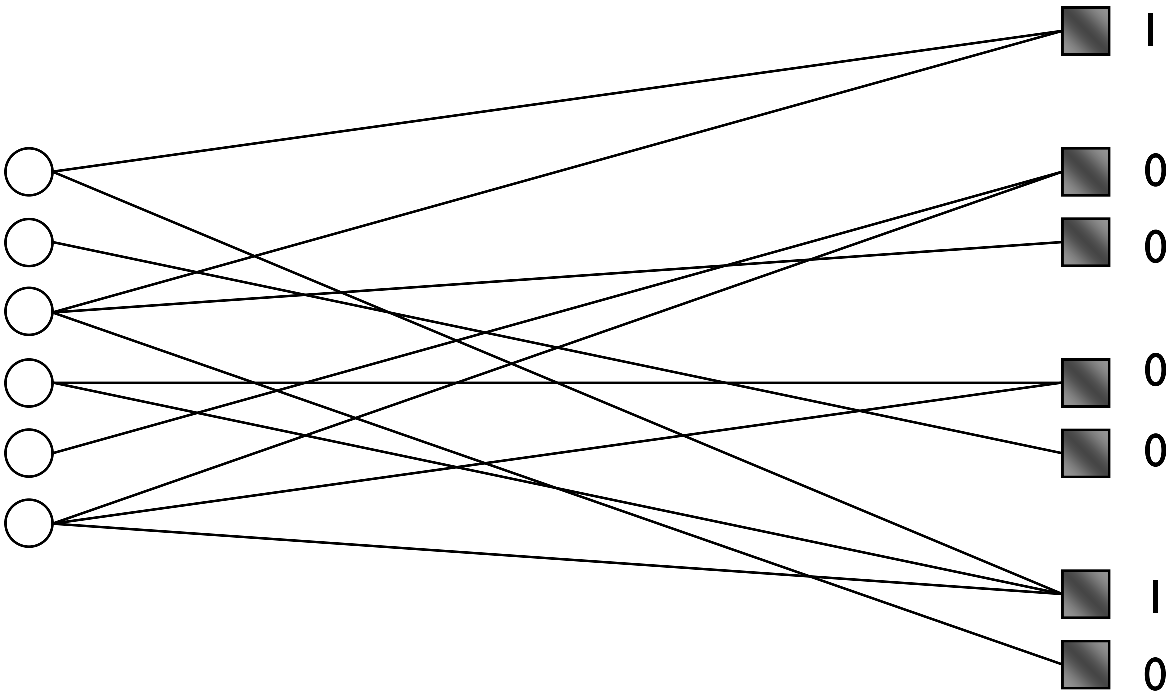
All information symbols are covered!

One condition for the end of decoding

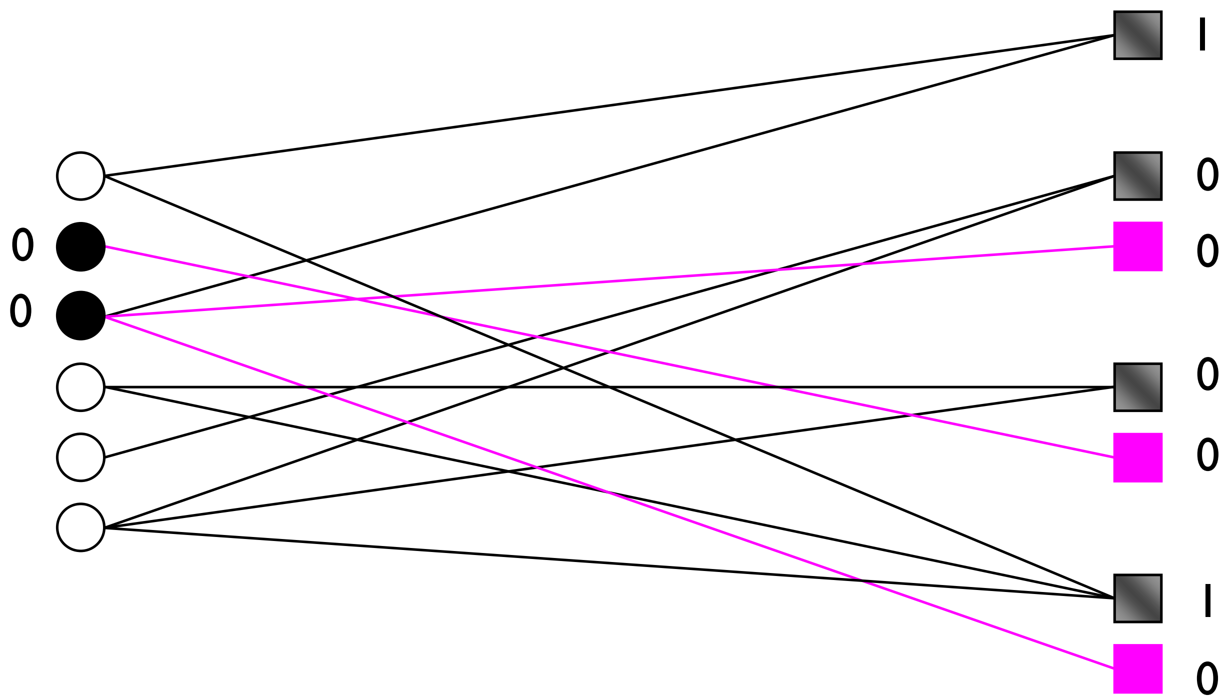
Success!!

Decoding Example of Failure

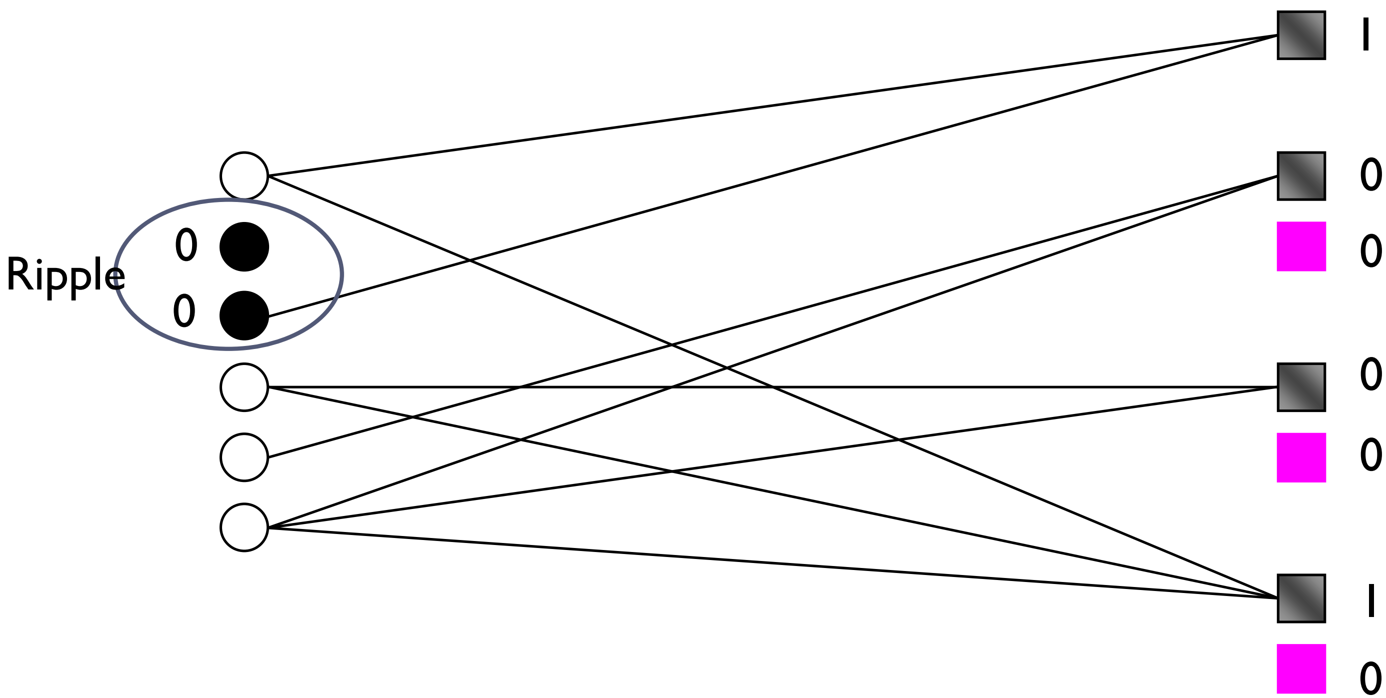
Example 2



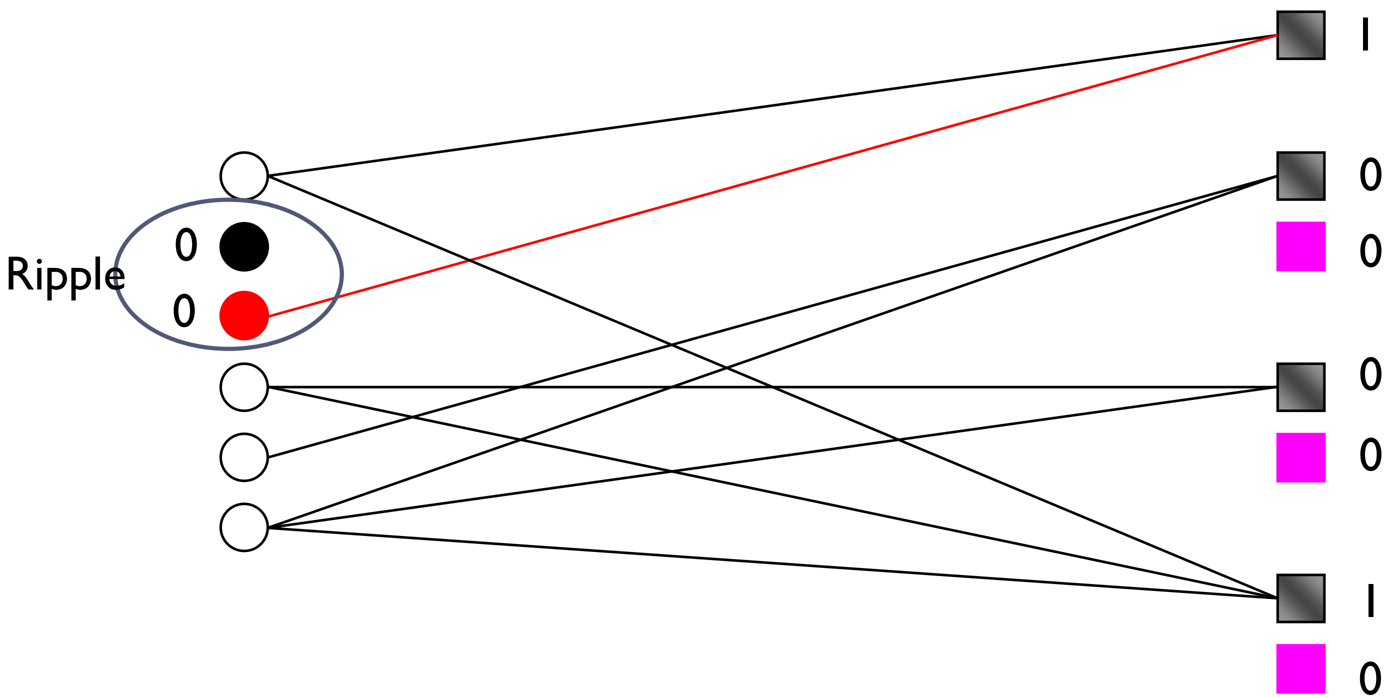
Decoding



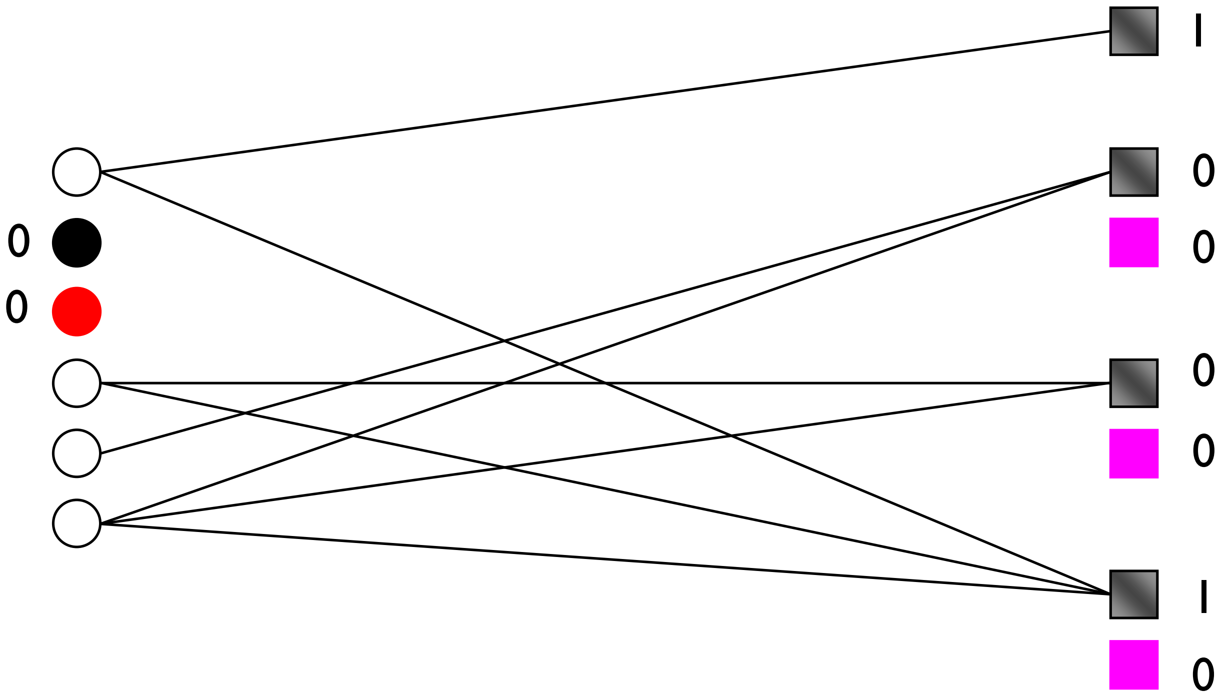
Decoding



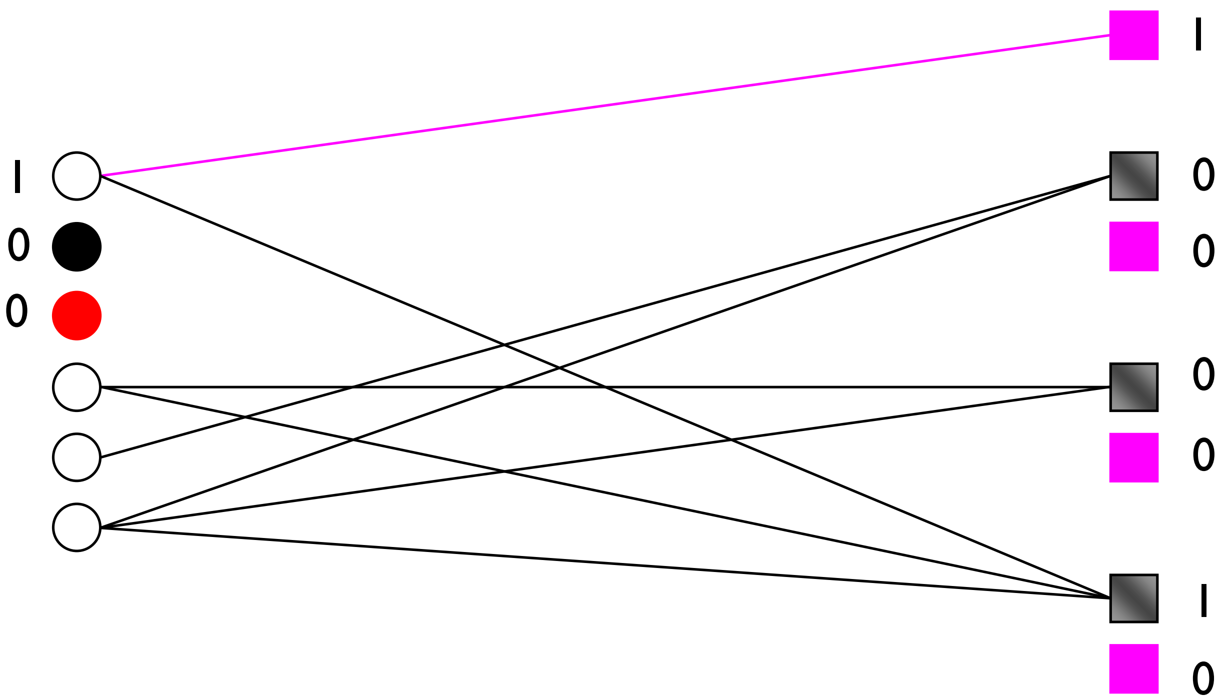
Decoding



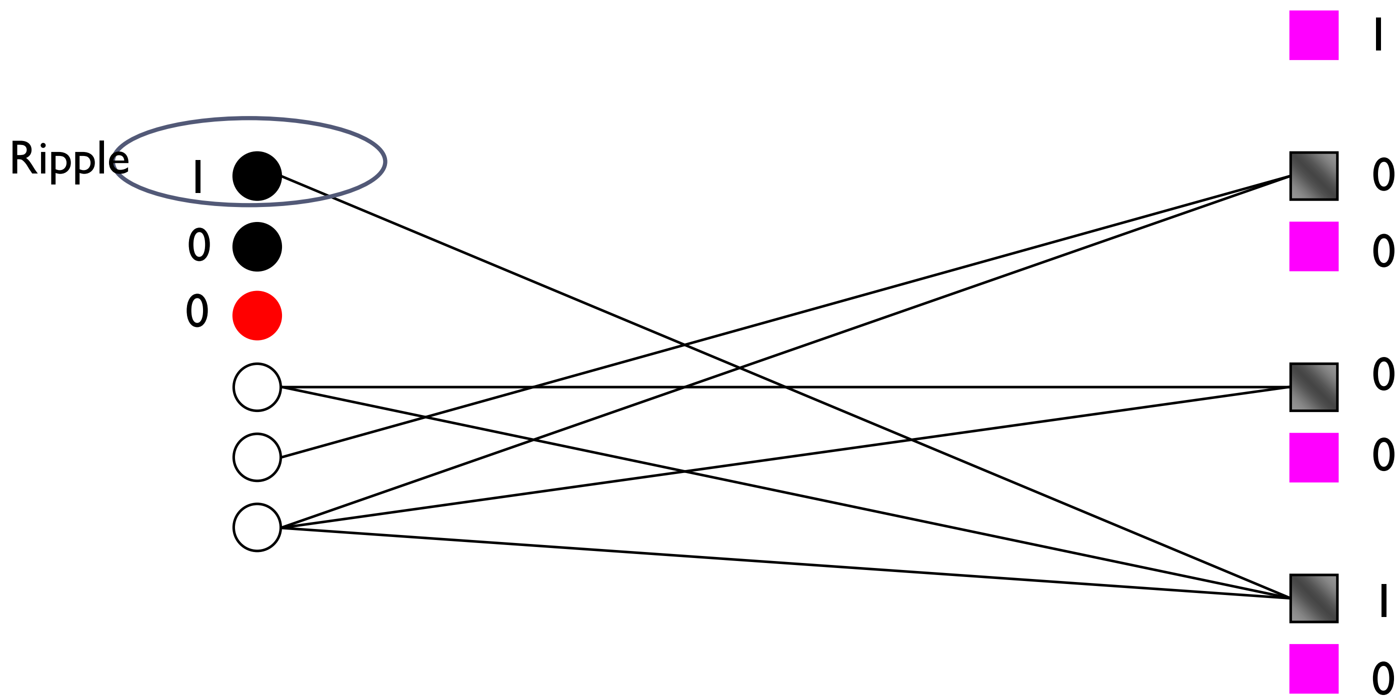
Decoding



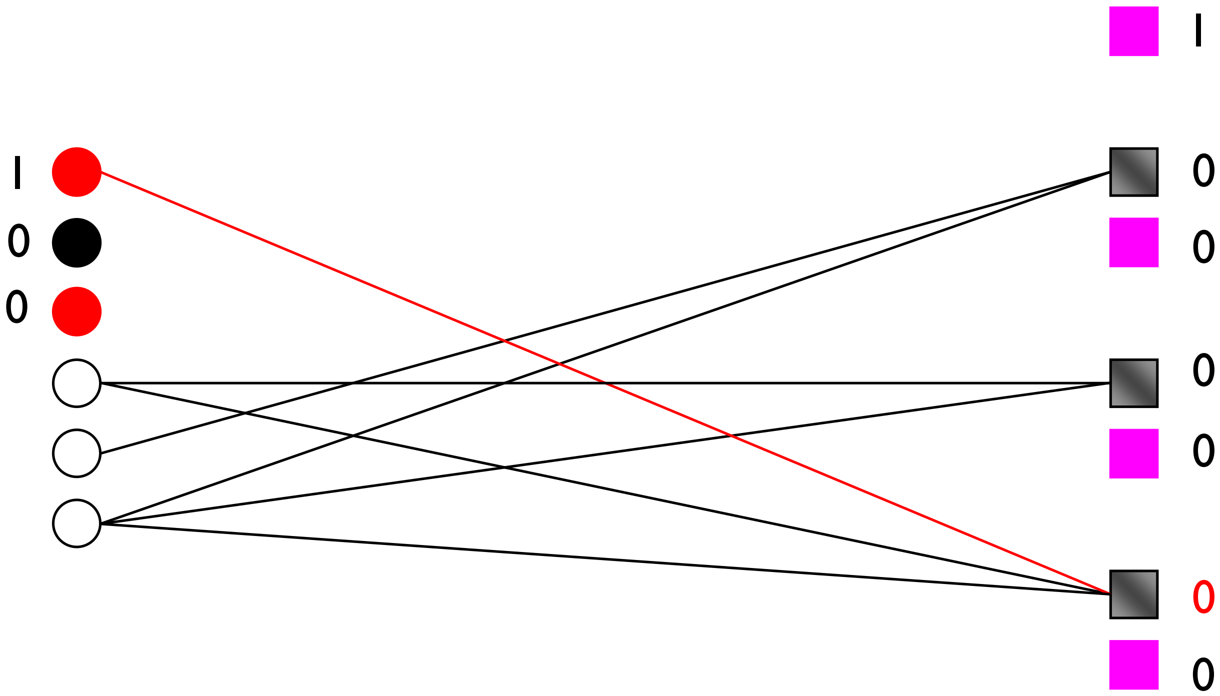
Decoding



Decoding



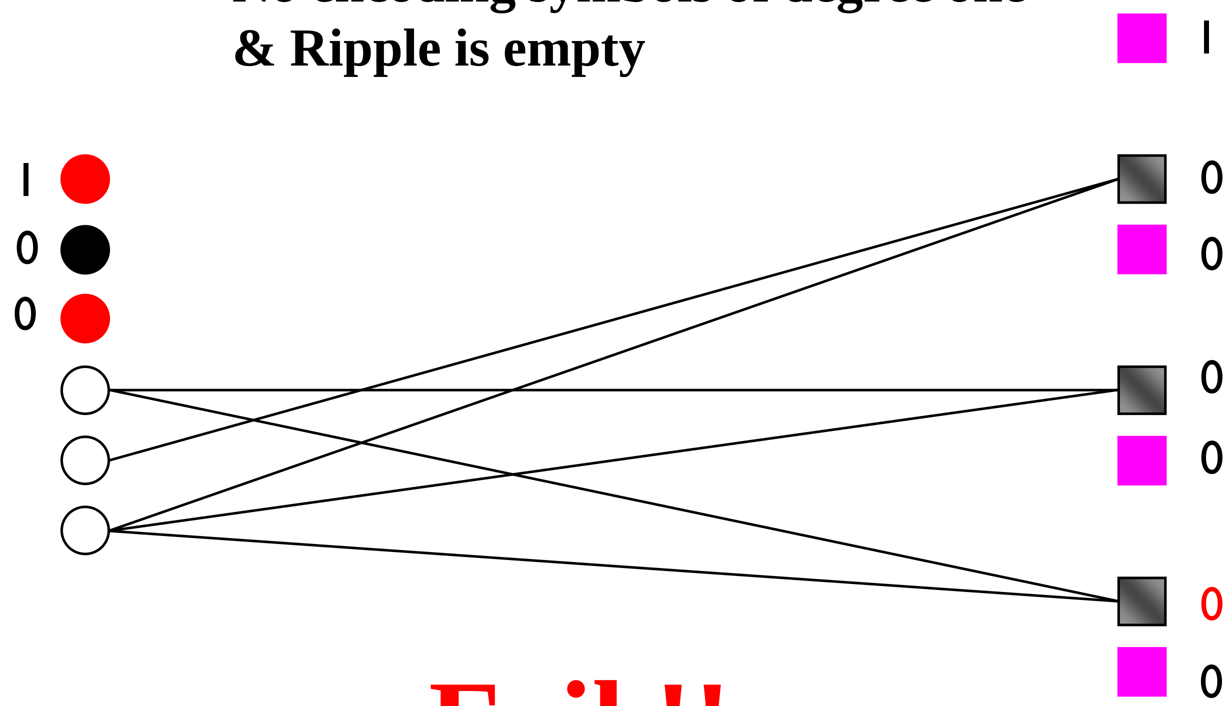
Decoding



Decoding

Another condition for the end of decoding

No encoding symbols of degree one
& Ripple is empty

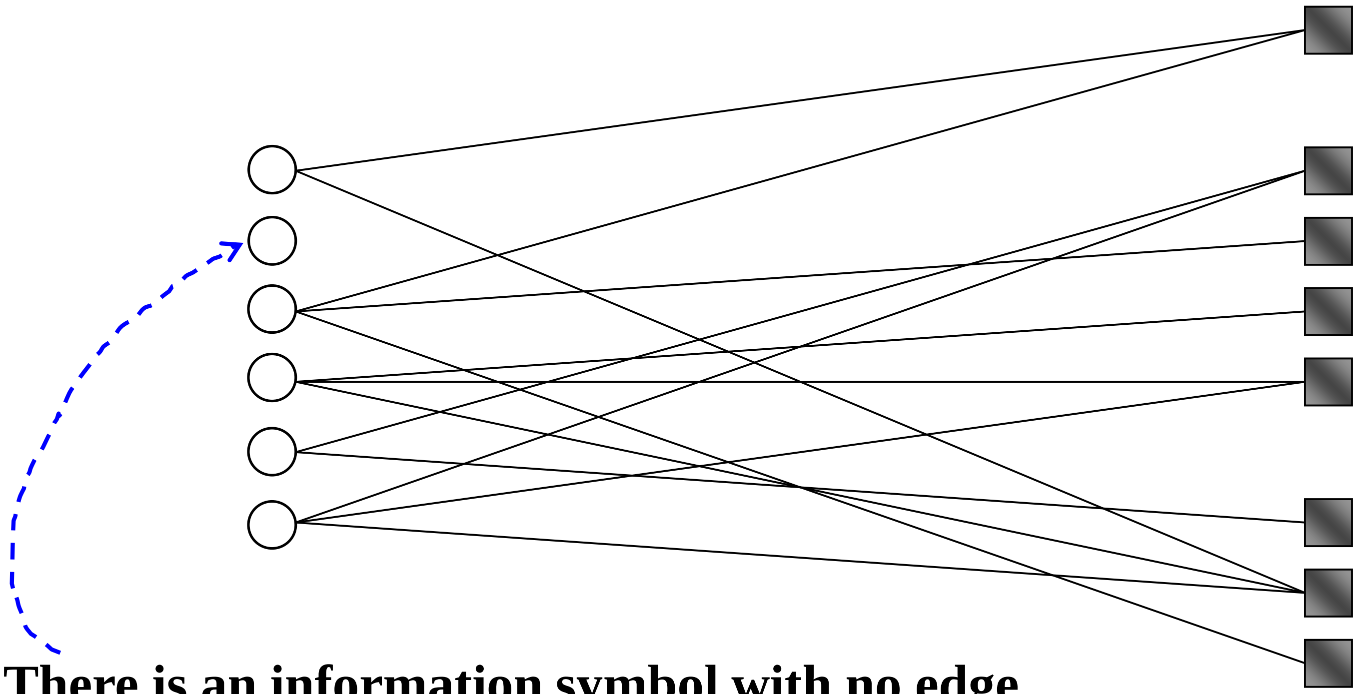


Fail !!

since some of the information symbols
are still uncovered.

Decoding

Example 3



There is an information symbol with no edge

Fail !!

even without beginning

Decoding - LT Process

(All information symbols are initially **uncovered** and a **ripple** is empty)

1. **(Release)**
   - **Release** all the degree-1 encoding symbols.
   - If no encoding symbol has degree-1, then go to Step 3.
2. **(Cover)**
   - The released encoding symbols **cover** their unique neighbor information symbols, and these **covered symbols** are put into a set called "**ripple**."
   - If all the information symbols are covered, decoding succeeds.
3. **(Process)**
   - If ripple is empty, (with some uncovered symbols) the decoding fails.
   - Otherwise, any one information symbol in the ripple is chosen (at random) to be processed: **the edges connecting the information symbol to its neighbor encoding symbols are removed and the value of each encoding symbol is updated.**

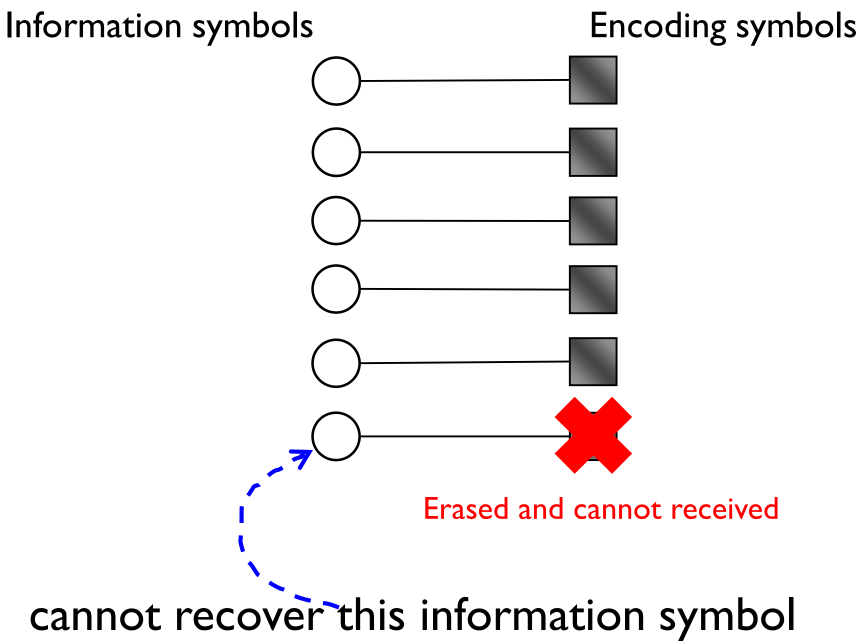
- The decoding process continues by iterating the above three steps

Designing The Degree Distribution - code design problem

- The desired property
  - The probability of success recovery is **as high as possible**
  - The number of required encoding symbols for successful decoding is kept **small**
- The degree distribution is a critical part of the design
  - Many packets must have **low degree**
    - The decoding process can get started and keep going
    - The total number of operations involved in the encoding and decoding is kept small
  - Some packets must have **high degree**
    - In order to ensure that there are not some source packets connected to no one
- **Therefore, the degree distribution of encoding symbols needs to be elaborately designed so as to balance between the above trade-off**

Designing The Degree Distribution

- The ***All-At-Once*** distribution (Example)
  - $\rho_1 = 1$ and $\rho_d = 0 \; for \; d = 2,3,\ldots$
  - Encoding symbols have one neighbor each
  - Any received encoding symbol can immediately recover the associated information symbol
  - Same as the **Data Carousel**



Balls-In-Bins Exercise

Imagine that we throw n balls **independently at random** into k bins, where $\boldsymbol{k}$ ($\cong \boldsymbol{n}$) is a large number, say, 1,000 or 10,000 or more. **What is the fraction of bins without any ball ?**

- The probability that any bin is empty is $\left(1-\frac{1}{k}\right)^n \cong e^{-n/k} \cong e^{-1}$
- The number of empty bins will be k/e
- So, **the fraction of bins without any ball** becomes again $1/e$

Define a random variable $X_i = 1$ if i-th bin is empty and $X_i = 0$ otherwise. Then,

$$P[X_i = 1]{=}1/e \text{ and } P[X_i = 0]{=}1 - 1/e$$

Therefore,

$$E[X_i] = 1/e$$

The number Y of empty bins is now given by

$$Y = \sum X_i$$

Its expectation becomes

$$E[Y] = \sum E[X_i] = \sum 1/e = k/e$$

Balls-In-Bins Exercise

If we throw **three times as many balls as there are bins**, is it likely that any bins will be empty?

- If $n = 3k$, the empty fraction drops to $1/e^3$, about 5%

In order for the probability that all $\boldsymbol{k}$ bins have at least one ball to be $\mathbf{1-\delta}$, we require at least $\boldsymbol{n = k\, ln\left(\frac{k}{\delta}\right)}$ balls

- For general n, the expected number of empty bins is $ke^{-\frac{n}{k}}$
- $ke^{-\frac{n}{k}} < \delta$ only if $n > k \ln(k/\delta)$

Conclusion:

In order for **almost every** $\boldsymbol{k}$ bins to be filled with **at least one ball** we have to throw at least $\boldsymbol{k\, ln(k)}$ balls

Balls-In-Bins Exercise ➡ Degree Distribution

- The classical process of throwing balls into bins can be viewed as the LT process
  - Balls: edges
  - Bins: source packets **(information symbols)**
  - In order for decoding to be successful, **every bin(source symbol) must have (necessarily) at least one ball(edge in the graph) in it. Otherwise, decoding will not be started.**

- <u>The special case where all the encoding symbols have degree one:</u>
  - This has the 'all-at-once' distribution: $\rho(1) = 1$ (every symbol has degree 1)

If every encoding symbol has degree 1, then
the receiver must have at least $k \ln k$ encoding symbols
(necessary condition for the start of the decoding possible)
Less than this much receptions will definitely fail to recover k source symbols

Balls-In-Bins Exercise ➡ Degree Distribution

- The classical process of throwing balls into bins can be viewed as the LT process
  - Balls: edges
  - Bins: source packets **(information symbols)**
  - In order for decoding to be successful, **every bin(source symbol) must have (necessarily) at least one ball(edge in the graph) in it. Otherwise, decoding will not be started.**

Conversely, (one can prove that)

For the successful recovery using **exactly *k* encoding symbols**, it is required that

every encoding symbol must have degree **at least ln *k***.

Ideal Soliton Distribution

- Ideally, to avoid redundancy, we would like **the received graph** to have the property that **just one output symbol has degree one at each iteration**
  - At each iteration, when this output node is released, the degrees in the graph are reduced in such a way **that one new degree-one output node appears**
  - This ***Ideal Soliton distribution*** displays ideal behavior in terms of the expected number of encoding symbols needed to recover the data, in contrast to the *All-At-Once distribution*

- In expectation, **this ideal behavior** is achieved by the ideal soliton distribution

$$\begin{cases} \rho(1) = \dfrac{1}{k} \\ \rho(i) = \dfrac{1}{i(i-1)}, \quad i = 2, \ldots, k \end{cases}$$

Ideal Soliton Distribution

- In practice, the Ideal Soliton Distribution shows poor performance
- The main problem with Ideal Soliton Distribution is that the expected **ripple size** is **too small** (=1)
  - Any variation in the ripple size is likely to make the ripple disappear and then the overall process fails

- The **Robust Soliton Distribution** ensures that the expected ripple size is **large enough** at each point in the process so that it never disappears completely in high probability
  - On the other hand, in order to minimize the overall number of encoding symbols used, it is important to **minimize the expected ripple size** so that not too many released encoding symbols redundantly cover input symbols already in the ripple

Robust Soliton Distribution

Ideal soliton

- The Robust Soliton distribution

perturbed

$$\mu_d = \frac{(\rho_d + \tau_d)}{\beta}$$

where β is the normalization constant chosen to ensure that μ is a probability distribution, and

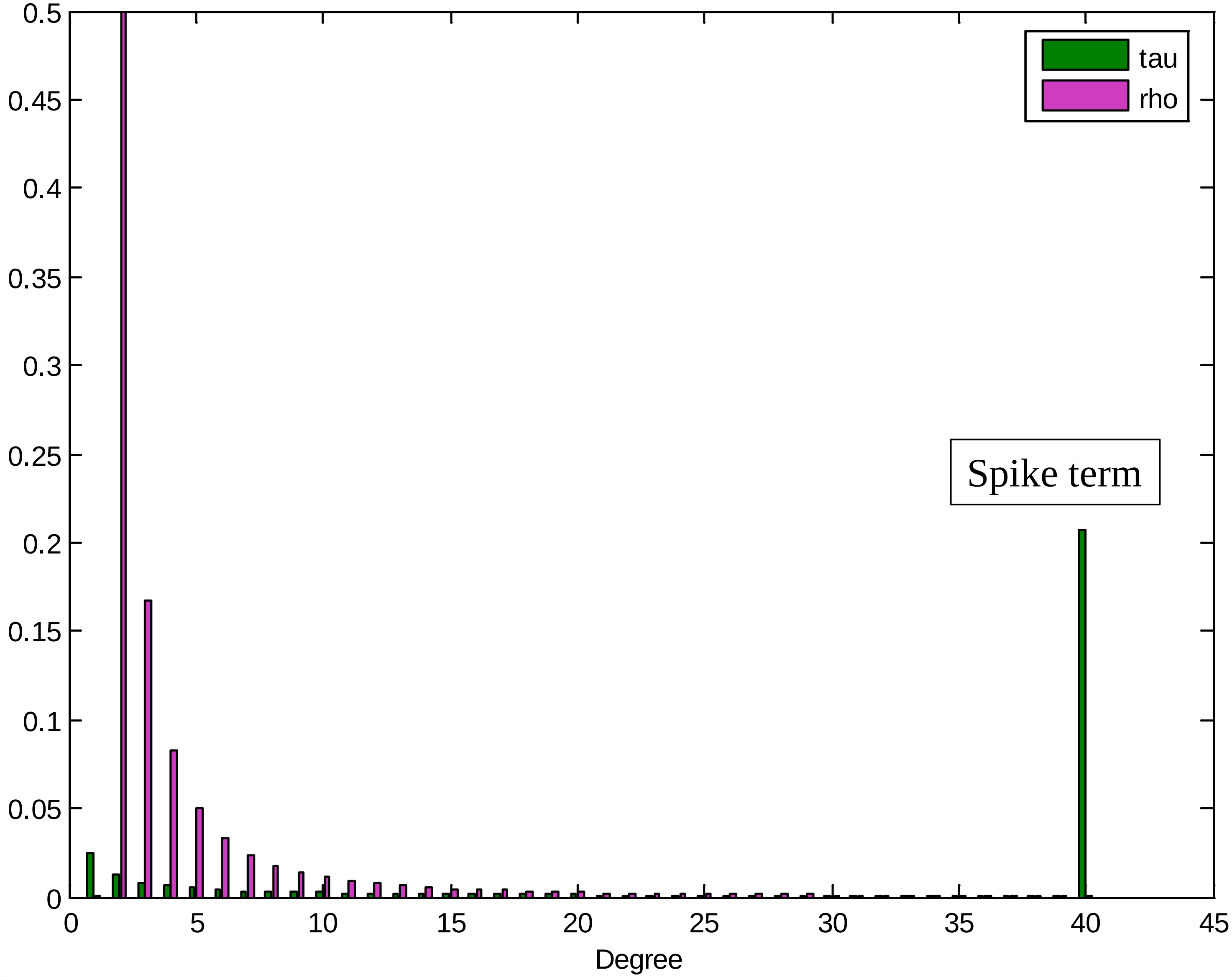
$$\tau_d = \begin{cases} \frac{R/k}{d} & \text{for } d = 1, \dots, \frac{k}{R} - 1 \\ \left(\frac{R}{k}\right) \ln\left(\frac{R}{\delta}\right) & \text{for } d = \frac{k}{R} \\ 0 & \text{for } d = \frac{k}{R} + 1, \dots, k \end{cases}$$

Diminishing marginals

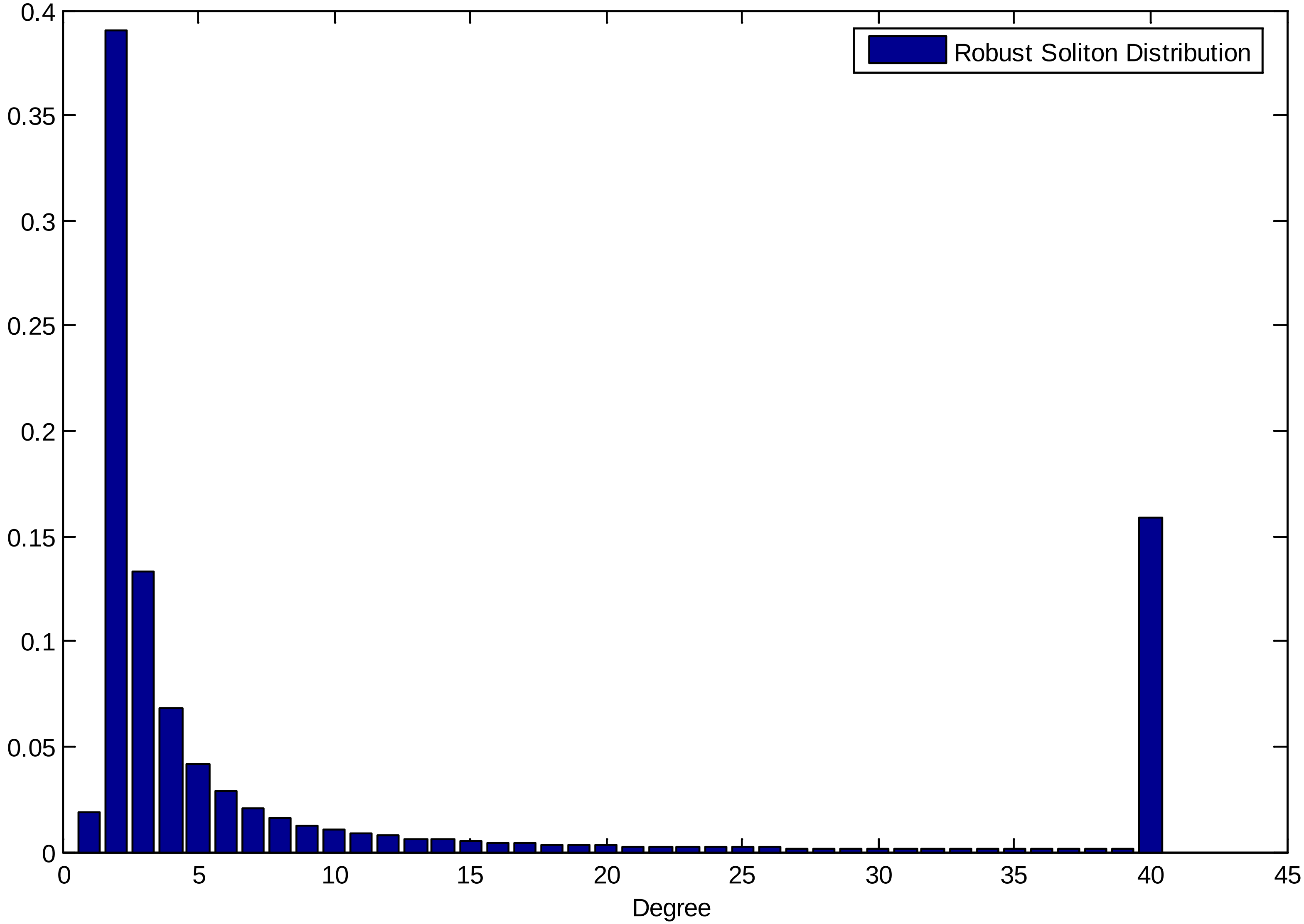
(a spike!!)

- It is designed to ensure that the expected number of degree-one output symbols is about $R \equiv c \ln\left(\frac{k}{\delta}\right)\sqrt{k} \gg 1$ (rather than 1 in ideal soliton), throughout the decoding process
  - The parameter δ is a bound on the probability that the decoding fails to run to completion
  - The parameter c is a constant of order 1

Ideal Soliton Distribution and $\tau(d)$ for the case $k = 10000, c = 0.2, \delta = 0.05$



Robust Soliton Distribution for the case $k = 10000, c = 0.2, \delta = 0.05$



Average Degree of an Encoding Symbol (for RSD)

- The average degree of an encoding symbol is

$$D = \frac{\sum_i \; i\big(\rho(i) + \tau(i)\big)}{\beta}$$

$$\leq \sum_i i\big(\rho(i) + \tau(i)\big)$$

$$= \underline{\sum_{i=2}^{k+1} \frac{1}{i-1}} + \sum_{i=1}^{\frac{k}{R}-1} \frac{R}{k} + \ln \frac{R}{\delta}$$

Sum of harmonic series

$\approx \ln(k)$

Heavy burden on encoding/decoding complexity!!!
(though it guarantees good performance)

RAPTOR CODES

A. Shokrollahi, “Raptor codes,” IEEE Trans. Inf. Theory, vol. 52, no. 6, pp. 2551-2567, Jun. 2006.

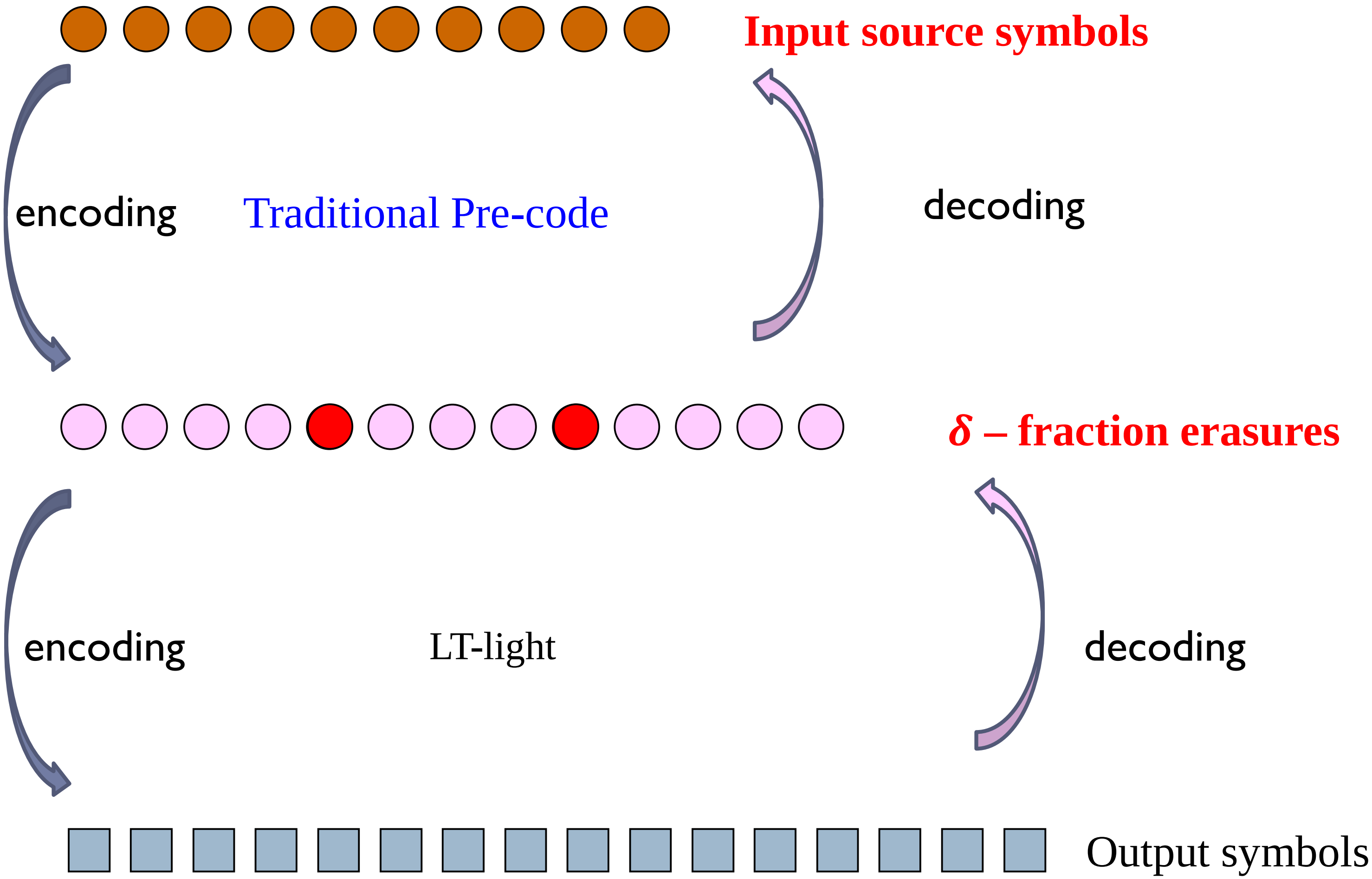
Raptor Codes

- For LT codes, the encoding and decoding costs scale as ***k ln k***
- We want linear time encoding and decoding!
  - The code graph should have a lower average degree

- **(Example)** For the use of an LT code with average degree $\bar{d} \cong 3$
  - A fraction of the input symbols will not be connected to the graph and so will not be recovered
  - From balls-in-bins exercise, the expected fraction of not recovered is $\delta \equiv e^{-\bar{d}}$, which for $\bar{d} = 3$ is 5% (too much!!!)

- Shokrollahi's trick in **Raptor codes**
  - **LT code** can recover a $(1-\delta)$-fraction
  - **Pre-code** can recover **all the original symbols** from any $(1-\delta)$-fraction

Raptor Codes

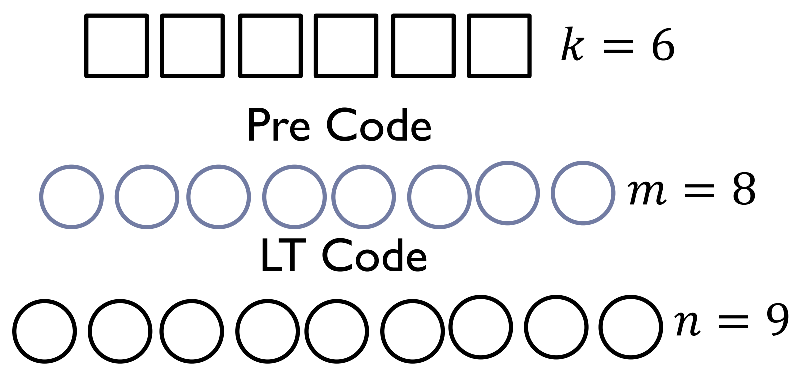


Raptor Codes

- In general, Raptor code can be defined as **a two-stage process**
  - An (m, k) linear block code C, called pre-code, as the outer code
  - An LT code specified by a node degree distribution $\Omega_D(x)$ as the inner code

  - **Encoding cost** is the sum of the encoding costs of the individual codes
  - **Decoding cost** is the sum of the individual decoding costs

- Two additional performance measures
  - **Space** $\quad$ Space $= \frac{m}{k} = \frac{8}{6}$
  - **Overhead** $\quad$ Overhead $\epsilon = \frac{n-k}{k} = \frac{3}{6}$



Raptor Codes

- Two extreme example
  - LT codes (without pre-code)
    - No pre-codes: space is close to 1
    - Overhead is close to 0
    - Time: logarithmic encoding and decoding cost
  - Pre-Code Only codes (without LT code)
    - Pre-codes with $\Omega(x) = x$: space is away from 1
    - Overhead is away from 0

- **Design Raptor codes between these two extremes**
  - **Constant encoding and decoding cost**
  - **Space is close to 1**
  - **Overhead is close to 0**
  - **These codes can be designed by choosing an appropriate output distribution $\Omega(x)$ and an appropriate pre-code C**

Raptor Codes

- A Raptor code that will asymptotically have **constant** encoding and decoding costs, and **minimum** overhead and space **when**
  - The pre-code C has rate $R = \frac{1+\epsilon/2}{1+\epsilon}$ and is able to decode up to $(1-R)/2$ fraction of erasures
    $\epsilon = 0.1 \quad \rightarrow R = \frac{105}{110}$ (1-105/110)/2 = 5/220
    - This is significantly less powerful than a capacity-achieving code, which can decode up to $(1-R)$ fraction of erasures
  - $\Omega_D(x)$ is close to an ideal soliton distribution but with some weight for degree one and capped to a maximum degree D

$$\Omega_D(x) = \frac{1}{\mu+1}\left(\mu x + \sum_{i=2}^{D} \frac{x^i}{i(i-1)} + \frac{x^{D+1}}{D}\right)$$

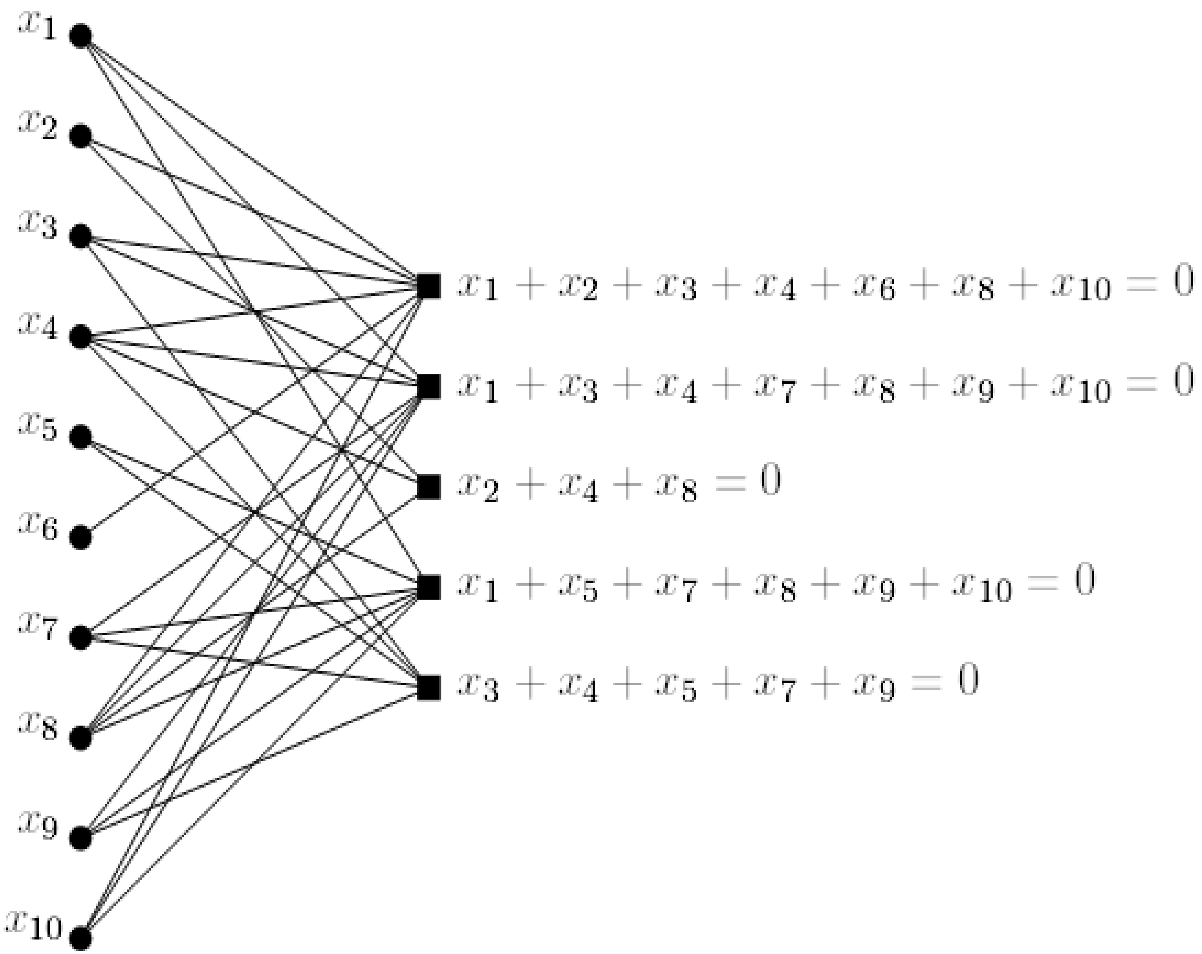
D = 4.4/0.1 = 44

A spike at D+1

    - Setting $D = \lceil 4(1+\epsilon)/\epsilon \rceil$ and $\mu = \left(\frac{\epsilon}{2}\right) + \left(\frac{\epsilon}{2}\right)^2$
- This code has
  - space consumption $1/R$, = 110/105
  - overhead ϵ and $\epsilon = 0.1$
  - encoding/decoding cost of $O\left(ln\left(\frac{1}{\epsilon}\right)\right)$ $O(ln(10))$

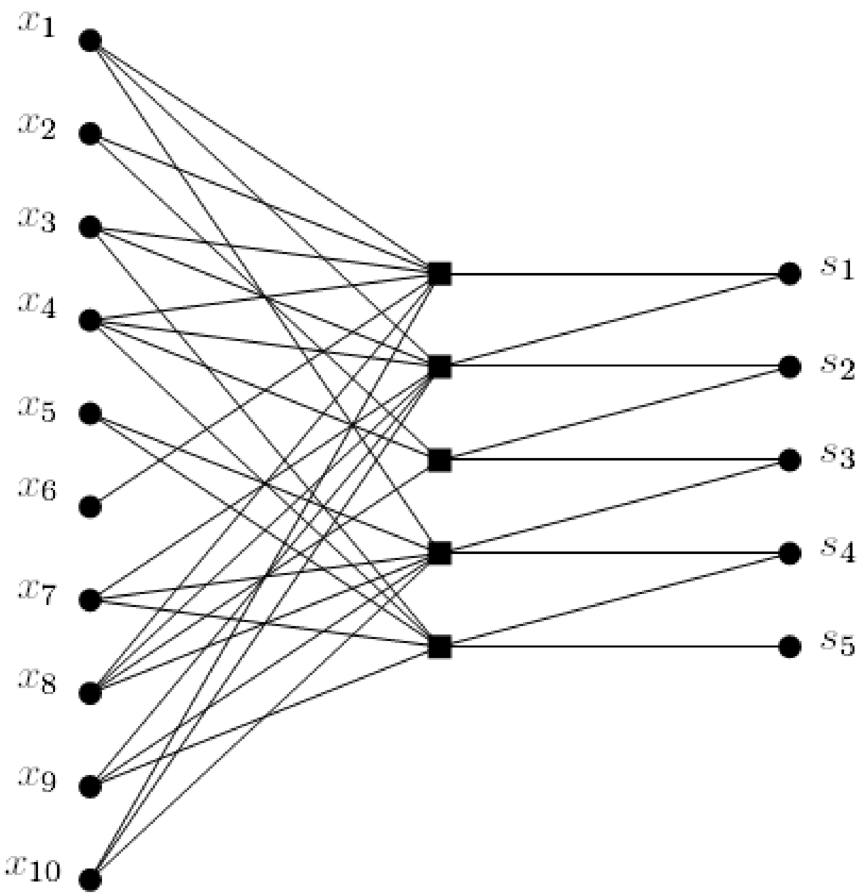
A variety of codes can be used as the pre-code

- LDPC codes - Low density gives a low complexity



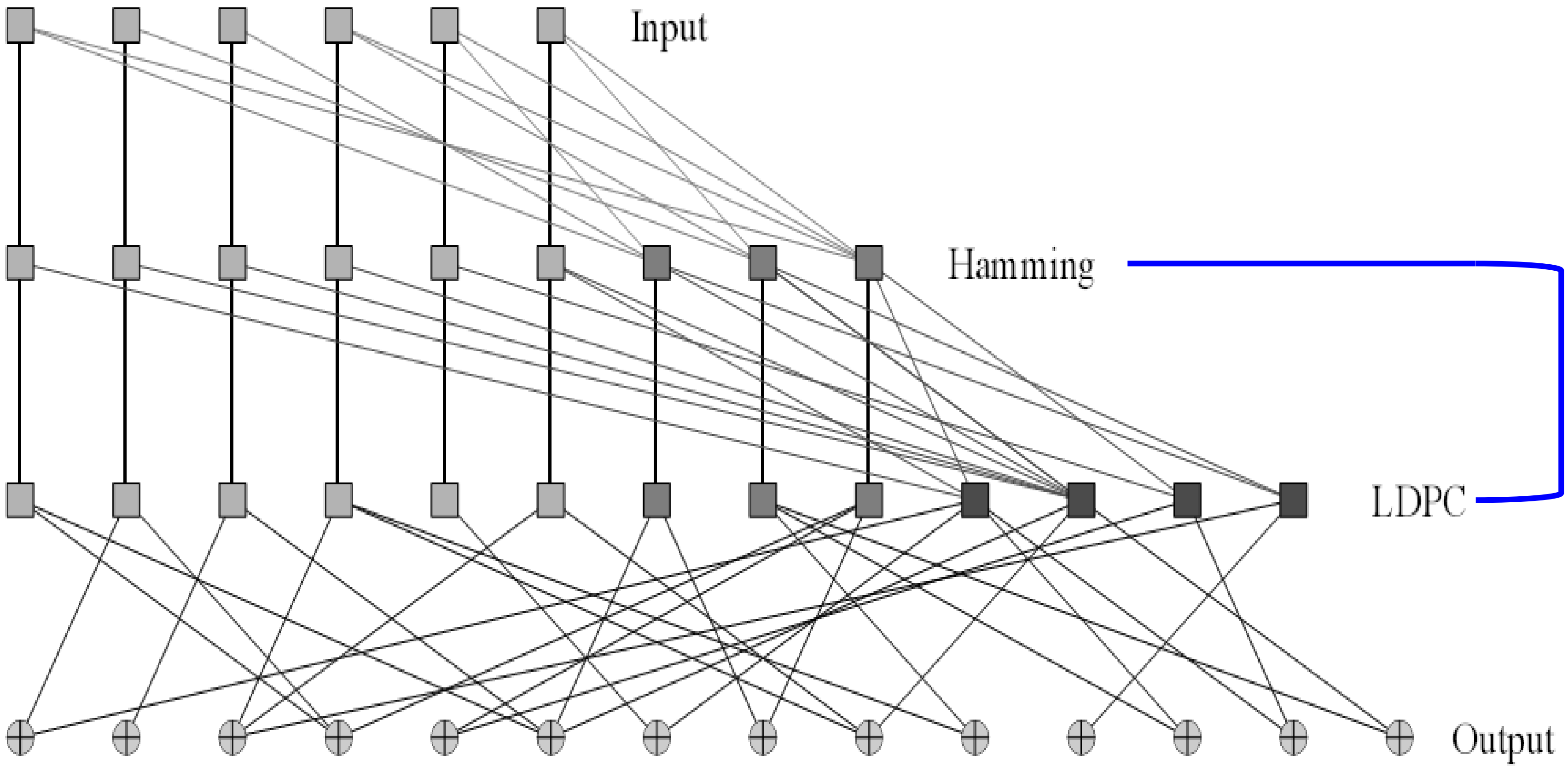
A variety of codes can be used as the pre-code

- **Repeat Accumulate codes**
  - Add redundant for check point to recover but don't add too large overhead
  - More robust error correcting codes



A variety of codes can be used as the pre-code

- **Concatenated Pre-codes (3GPP)**



Summary

- With many good properties, **fountain codes** have been applied to a variety of engineering applications, such as hybrid ARQ, scalable video streaming, and sensor networks
- **Raptor codes have been adopted in several standards**
  - 3GPP, where it is used for a mobile cellular wireless broadcast and multicast
  - DVB-H standards, where it is used for IP datacast to handheld devices
- The characteristics of various codes that are designed for the digital fountain ideal

	Tornado	**LT**	**Raptor**
Rateless	No	Yes	Yes
Overhead	ϵ	$\epsilon \to 0$	$\epsilon \to 0$
Encoding complexity per symbol	$O(\epsilon \ln(1/\epsilon))$	$O(\ln(k))$	$O(1)$
Decoding complexity per symbol	$O(\epsilon \ln(1/\epsilon))$	$O(\ln(k))$	$O(1)$
Space per symbol	$O(1)$	$O(1)$	$O(1)$, with a larger constant.

REFERENCES

- J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," *SIGCOMM Comput. Commun. Rev.*, vol. 28, pp. 56-67, 1998.
- J. W. Byers, M. Luby, and M. Mitzenmacher, "A digital fountain approach to asynchronous reliable multicast," *Selected Areas in Communications, IEEE Journal on*, vol. 20, pp. 1528-1540, 2002.
- M. Mitzenmacher, "Digital fountains: a survey and look forward," in *Information Theory Workshop, 2004. IEEE*, 2004, pp. 271-276.
- D. J. C. MacKay, "Fountain codes," *Communications, IEE Proceedings-*, vol. 152, pp. 1062-1068, 2005.
- M. Luby, M. Mitzenmacher, and A. Shokrollahi, "Analysis of random processes via And-Or tree evaluation," presented at the Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms, San Francisco, California, United States, 1998.
- M. Luby, "LT codes," in *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, 2002, pp. 271-280.
- A. Shokrollahi, "Raptor codes," in *Information Theory, 2004. ISIT 2004. Proceedings. International Symposium on*, 2004, p. 36.
- A. Shokrollahi, "Raptor codes," *Information Theory, IEEE Transactions on*, vol. 52, pp. 2551-2567, 2006.
- O. Etesami, M. Molkaraie, and A. Shokrollahi, "Raptor codes on symmetric channels," in Information Theory, 2004. ISIT 2004. Proceedings. International Symposium on, 2004, p. 38.
- R. Palanki and J. S. Yedidia, "Rateless codes on noisy channels," in Information Theory, 2004. ISIT 2004. Proceedings. International Symposium on, 2004, p. 37.
- O. Etesami and A. Shokrollahi, "Raptor codes on binary memoryless symmetric channels," Information Theory, IEEE Transactions on, vol. 52, pp. 2033-2051, 2006.
- N. Rahnavard and F. Fekri, "Finite-length unequal error protection rateless codes: design and analysis," in Global Telecommunications Conference, 2005. GLOBECOM '05. IEEE, 2005, p. 5 pp.

- N. Rahnavard, B. N. Vellambi, and F. Fekri, "Rateless Codes With Unequal Error Protection Property," Information Theory, IEEE Transactions on, vol. 53, pp. 1521-1532, 2007.
- M. C. O. Bogino, P. Cataldi, M. Grangetto, E. Magli, and G. Olmo, "Sliding-Window Digital Fountain Codes for Streaming of Multimedia Contents," in Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on, 2007, pp. 3467-3470.
- D. Sejdinovic, D. Vukobratovic, A. Doufexi, V. Senk, and R. Piechocki, "Expanding window fountain codes for unequal error protection," Communications, IEEE Transactions on, vol. 57, pp. 2510-2516, 2009.
- C. Yu, S. Blostein, and C. Wai-Yip, "Unequal error protection rateless coding design for multimedia multicasting," in Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on, 2010, pp. 2438-2442.
- W. Nan and Z. Zhiji, "The impact of application layer Raptor FEC on the coverage of MBMS," *Radio and Wireless Symposium, 2008 IEEE*, pp. 223-226, 22-24 Jan. 2008 2008.
- C. Berger, Z. Shengli, W. Yonggang, P. Willett, and K. Pattipati, "Optimizing Joint Erasure- and Error-Correction Coding for Wireless Packet Transmissions," *Wireless Communications, IEEE Transactions on*, vol. 7, pp. 4586-4595, 2008.
- C. Yu and S. D. Blostein, "Cross-layer optimization of rateless coding over wireless fading channels," in *Communications (QBSC), 2010 25th Biennial Symposium on*, 2010, pp. 144-149.
- S. Puducheri, J. Kliewer, and T. E. Fuja, "Distributed LT Codes," in *Information Theory, 2006 IEEE International Symposium on*, 2006, pp. 987-991.
- S. Puducheri, J. Kliewer, and T. E. Fuja, "The Design and Performance of Distributed LT Codes," *Information Theory, IEEE Transactions on*, vol. 53, pp. 3740-3754, 2007.

- D. Sejdinovic, R. J. Piechocki, and A. Doufexi, "AND-OR tree analysis of distributed LT codes," in *Networking and Information Theory, 2009. ITW 2009. IEEE Information Theory Workshop on*, 2009, pp. 261-265.
- M. L. Champel, K. Huguenin, A. M. Kermarrec, and N. Le Scouarnec, "LT Network Codes," in *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, 2010, pp. 536-546.
- D. J. C. MacKay and R. M. Neal, “Near Shannon limit performance of low density parity check codes,” *Electron. Lett.*, vol. 32, no. 18, pp. 1645–1646, March 1996, reprinted *Electron. Lett*, vol. 33(6), pp. 457–458, March 1997.
- M. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann, “Practical loss-resilient codes,” in *Proc. 29th Annual ACM Symposium on Theory of Computing*, 1997, pp. 150–149.
- P. Dohyung and C. Sae-Young, "Performance & complexity tradeoffs of rateless codes," in *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, 2008, pp. 2056-2060.

Tutorial review
on

Locally Repairable Codes

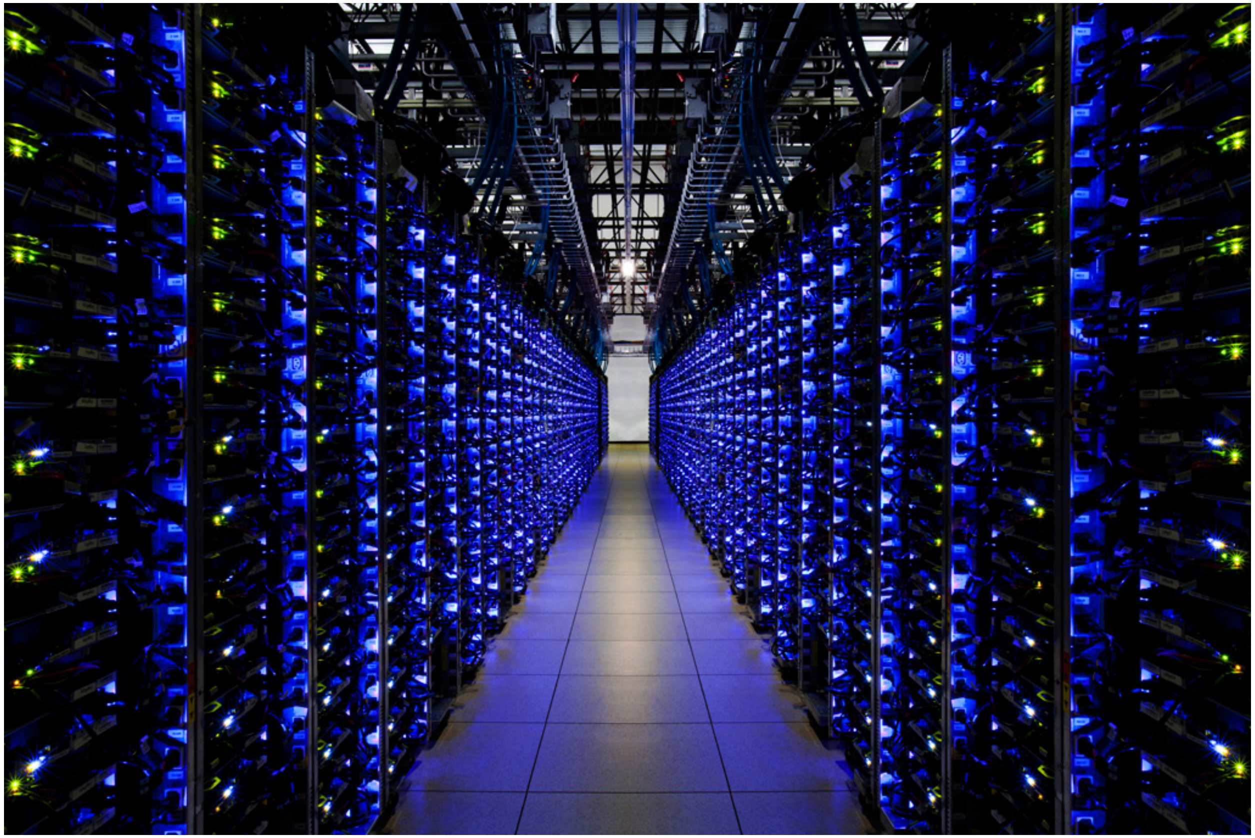
Codes for Distributed Storage Systems

Introduction to distributive storage codes

Information Era

- Large-Scale Storage System
  - Warehouse-scale data center
  - Thousands of servers, Petabytes (or Exabytes) of disk space

※ Peta: 10^{15}
Exa: 10^{18}



http://www.google.com/about/datacenters/

Google
Data Center

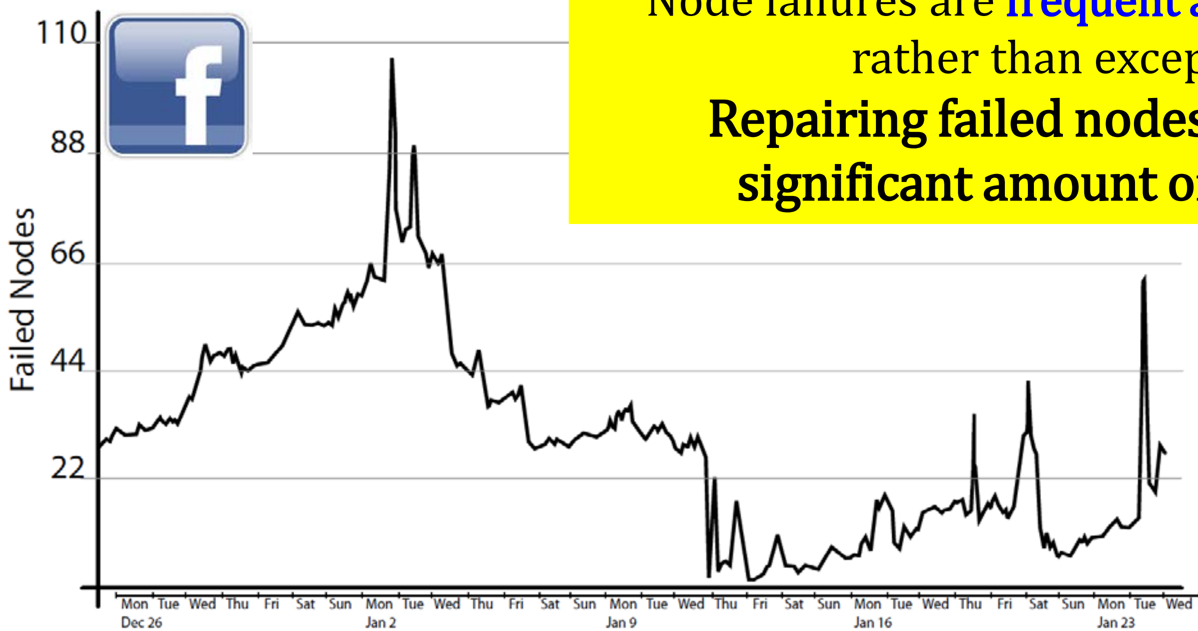
The Data Explosion

- We generate a **huge amount** of digital data
- We expect it to be **stored reliably** and **accessible anytime, anywhere for free**
  - Total data in the cloud is of the order of **few hundred exabytes**
  - Even storing raw data **costs** hundreds of millions
  - Hardware is no longer cheap
- Currently, **data centers** consume up to 3 percent of all global electricity production while producing 200 million metric tons of carbon dioxide

A Trace of Node Failures

- The number of failed nodes over a single month in a 3000 node production cluster of **Facebook**

Node failures are **frequent and normal** event rather than exceptional.
Repairing failed nodes consumes a significant amount of resources.

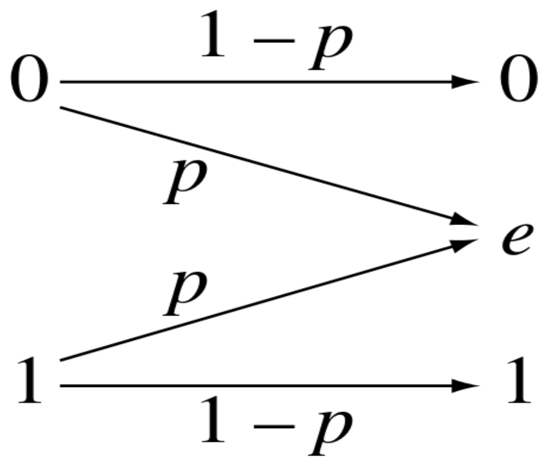


- More than 20 nodes fail daily on average

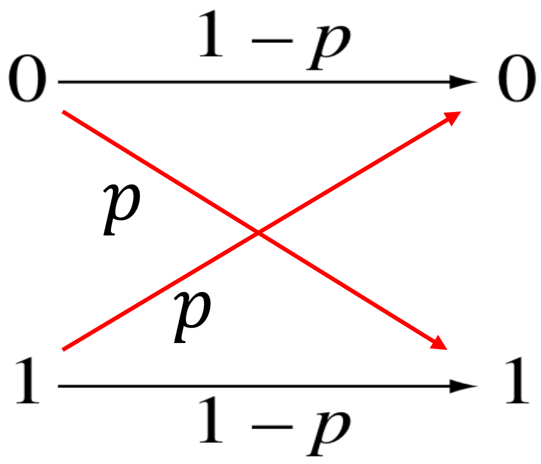
M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, D. Borthakur, "XORing Elephants: Novel Erasure Codes for Big Data," in Proc. of the 39th International Conf. on Very Large Data Bases, Aug. 2013.

Discrete coding channel model

- A discrete coding channel is a model of communication channel including digital modem, rx(tx) antenna and analog physical (RF/CABLE) channel.
- It is characterized by (1) input alphabet (2) output alphabet, and (3) transition probabilities between these two.
- Famous examples are BSC, BEC, etc.



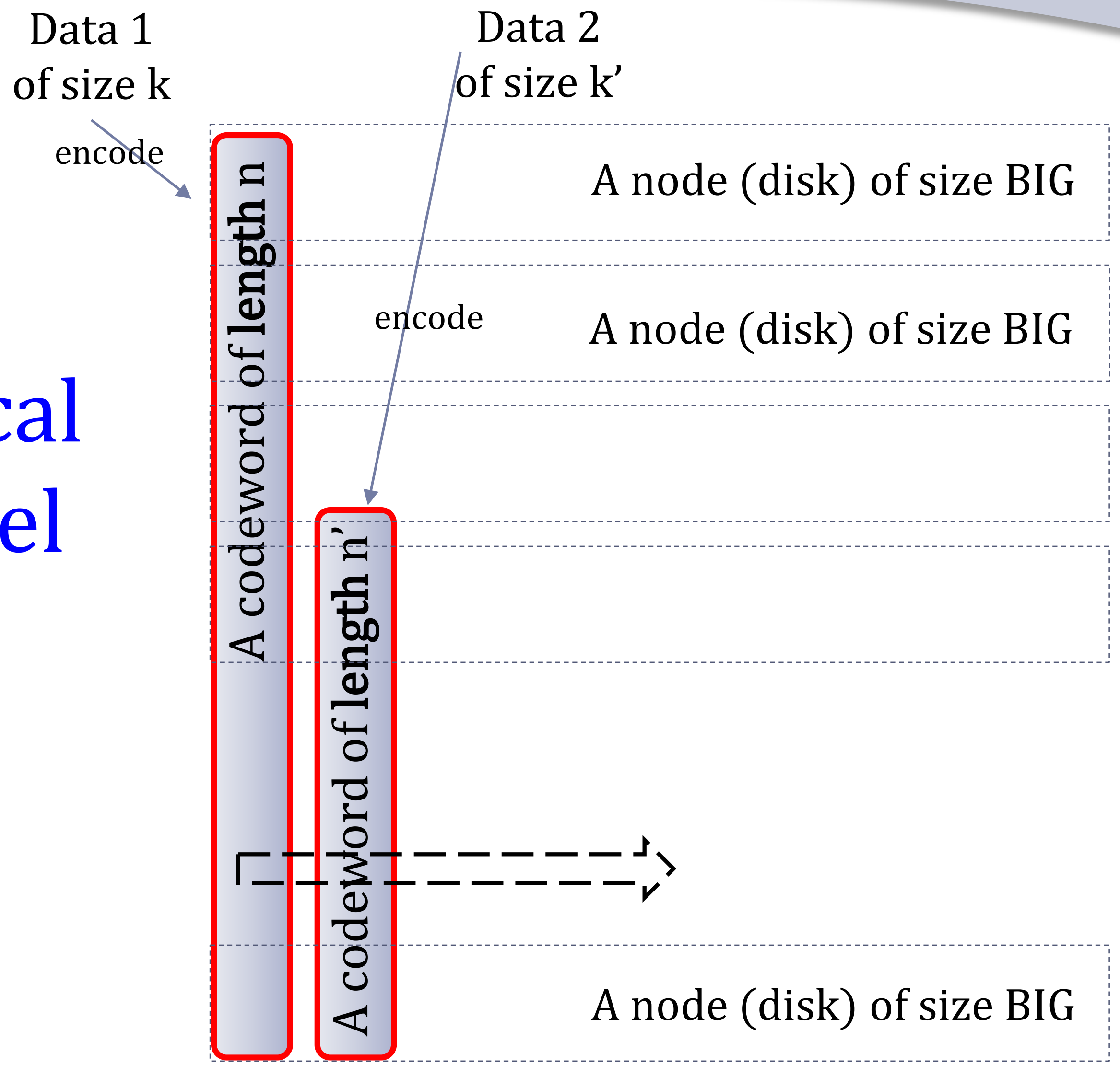
BEC(p)



BSC(p)

Disk failure is best modelled by BEC

Typical model

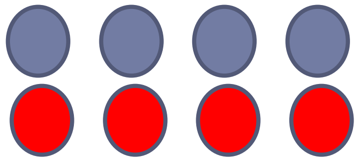


Reliable Data Storage

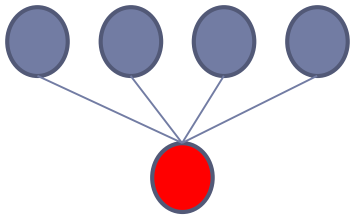
Channel model

Erasure only channel without errors

- Goal: Tolerate **one** disk failure
- Solution:
  - Duplication
    - 2x overhead
    - Quick recovery



  - Simple XOR – RAID 5
    - Treat each disk as a bit vector
    - 1.2x overhead
    - Slower recovery

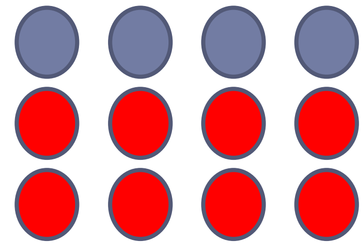


Reliable Data Storage

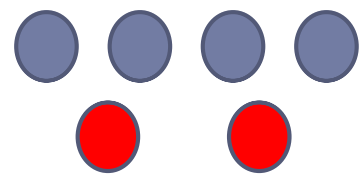
Channel model

Erasure only channel without errors

- Goal: Tolerate **two** disk failures
- Solution:
  - Triplication
    - 3x overhead
    - Quick recovery



  - (6,4) Reed Solomon Code – RAID 6
    - 1.5x overhead
    - Slower recovery
    - Need a larger field: each disk is a byte-vector



$\boldsymbol{d = 3}$

1-error correcting

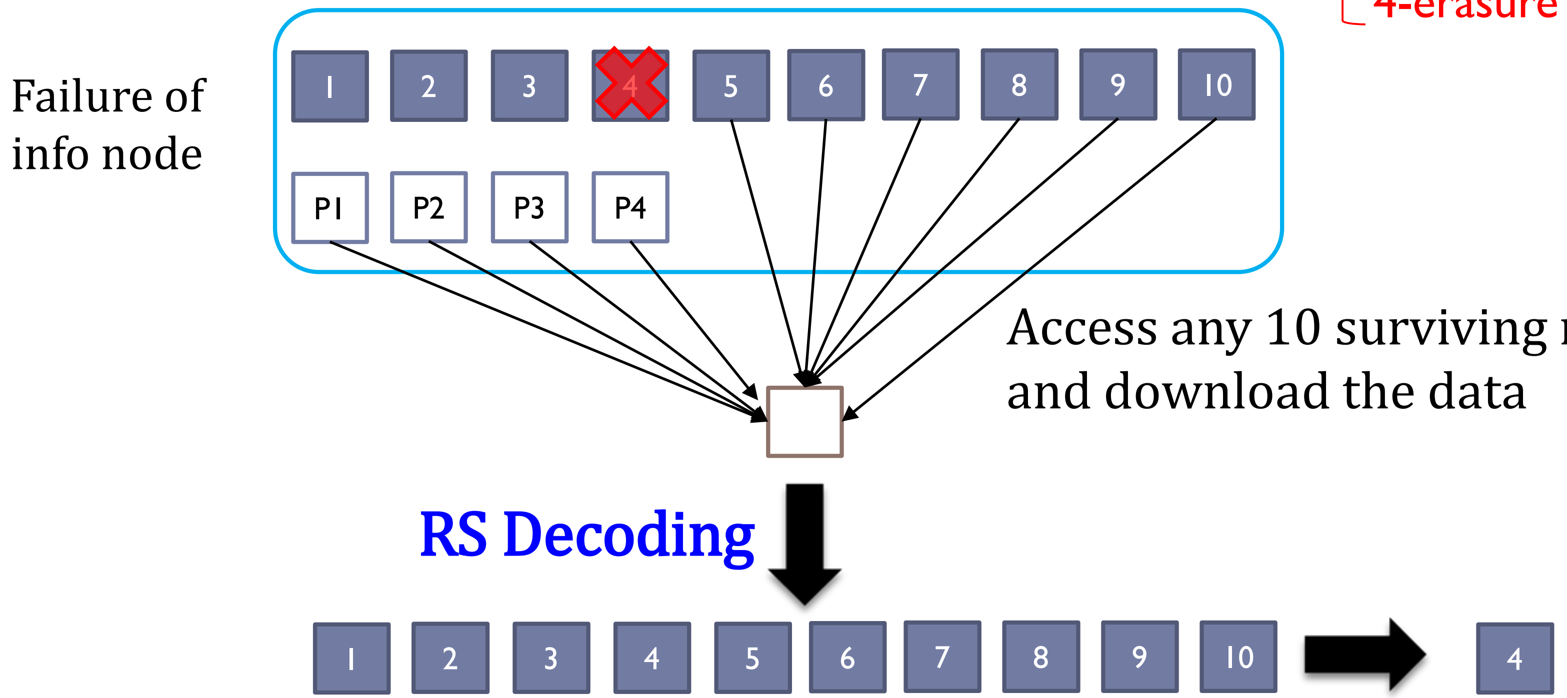
or

2-erasure correcting

Limitations of Reed-Solomon Codes

- Traditional erasure-correcting codes are optimized for **regeneration of the original message**
  - But **not** for regeneration of individual lost parts
- Example: (14, 10) RS code with d=14-10+1=5

2-error correcting
or
4-erasure correcting



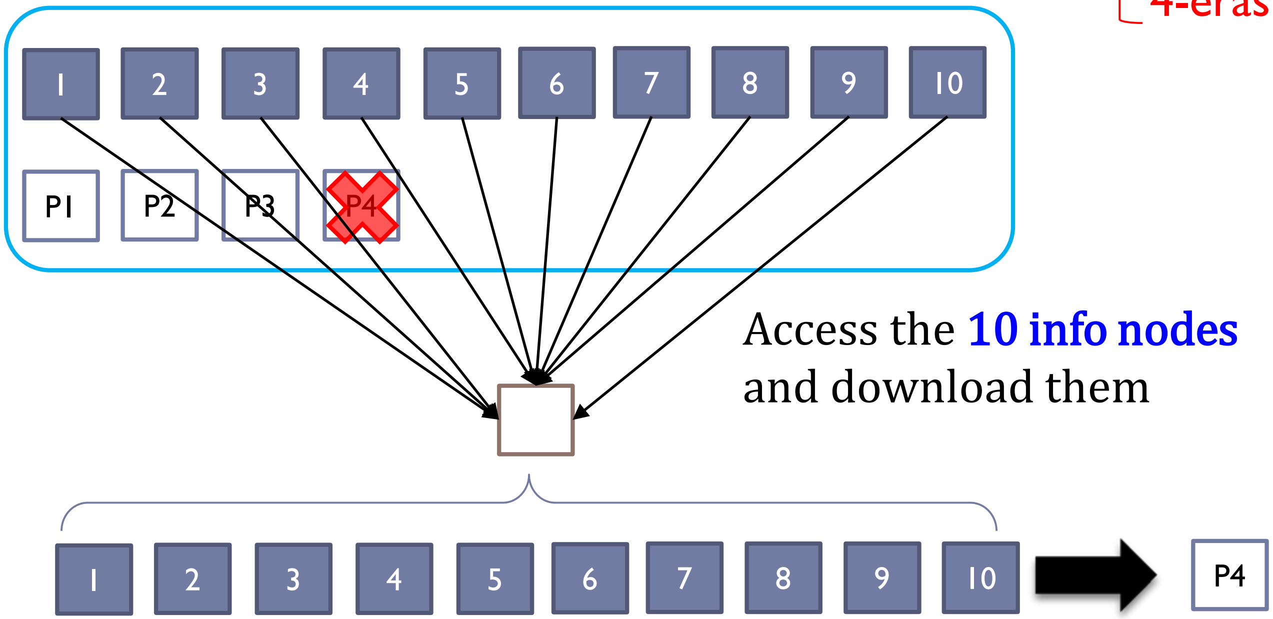
Reconstruct the whole file

Limitations of Reed-Solomon Codes

- Example: (14, 10) RS code with d=14-10+1=5

2-error correcting
or
4-erasure correcting

Failure of parity node



Access the 10 info nodes and download them

access the whole message file

RS Encoding

Do we really have to **reconstruct the whole message** in order to recover **a single node failure**?

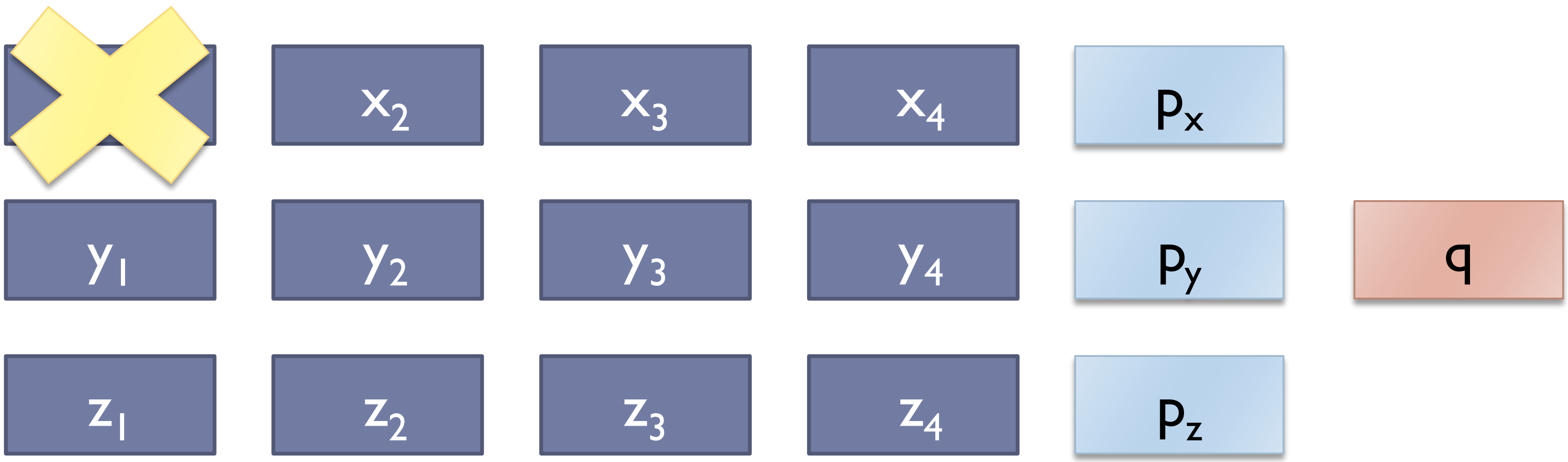
Good Repair Process

- **How many nodes(disks) have to be accessed and how much data from each node must be downloaded for the repair?**
- **Metric**: Total **bandwidth** during the repair
  - Regenerating Codes [Dimakis-Godfrey-Wu-Wainwright-Ramchandran `10]
  - Model allows connecting many disks

- **How many nodes(disks) have to be accessed?**
- **Metric**: Total **number of disks** participating in the repair
  - Locally Repairable Codes (LRC) [Gopalan-Huang-Simitci-Yekhanin `12]

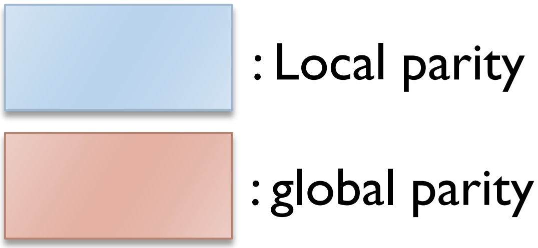
Locally Repairable Codes

LRC Example



- Local reconstruction for x_1

  $$\mathbf{x_1 = p_x - (x_2+x_3+x_4)}$$



Locality

[Chen-Huang-Li`07, Oggier-Datta`11, Gopalan-Simitci-Huang-Yekhanin`12, Papailiopoulos-Luo-Dimakis-Huang-Li'12]

- Let $\mathcal{C}$ be an $(n, k)_q$ code of length n, dimension k over a finite field $\mathbb{F}_q$
- The locality of the i-th coordinate of $\mathcal{C}$ is r if the value of the i-th symbol of a codeword of $\mathcal{C}$ is a function of r other coordinates and no such a set of coordinates of cardinality less than r exists
- The set of such r coordinates that can repair the i-th symbol is called **a repair set**
- The locality of the code $\mathcal{C}$ is r if the symbol locality of every symbol in a codeword of $\mathcal{C}$ is at most r

An (n, k) code $\mathcal{C}$ with locality $r(\ll \boldsymbol{k})$ is defined as an (n, k, r) locally repairable code

Locality

[Chen-Huang-Li`07, Oggier-Datta`11, Gopalan-Huang-Simitci-Yekhanin`12, Papailiopoulos-Luo-Dimakis-Huang-Li'12]

- A coordinate in a linear code has locality r if it can be expressed as a linear combination of r other coordinates

- If an i-th symbol c_i is lost, it can be recovered by reading just r other symbols
  - Information locality r: all information symbols have locality r
  - All-symbol locality r: all symbols have locality r

- Decouples typical decoding complexity r from length n
  - r reads **for single failures**, degraded reads
  - No guarantees for more worst-case failures

Parameters of LRCs
[Gopalan-Huang-Simitci-Yekhanin`12]

- Let $\mathcal{C}$ be an (n, k, r) LRC
  - Assume that $r|k$ and $r+1|n$
  - The rate is bounded by

$$\boldsymbol{\frac{k}{n} \le \frac{r}{r+1}}$$

Proof:

There exist at most $\frac{nr}{r+1}$ coordinates that determine the exact codeword

$$\boldsymbol{\Leftrightarrow k \le \frac{nr}{r+1}}$$

For example,

x_1	x_2	x_3	x_4	p_x
y_1	y_2	y_3	y_4	p_y
z_1	z_2	z_3	z_4	p_z

achieves the bound
since k/n=4/5 and r=4

Parameters of LRCs
[Gopalan-Huang-Simitci-Yekhanin`12]

- Let $\mathcal{C}$ be an (n, k, r) LRC
  - Assume that $r|k$ and $r+1|n$
  - The minimum distance is bounded by

$$d \leq n - k - \left\lceil \frac{k}{r} \right\rceil + 2$$

Remarks:

- Generalization of **the singleton bound** $(r = k)$

$$d \leq n - k + 1$$

- **An optimal (n, k, r) LRC** achieves the bound with equality

Optimal LRCs – Trivial (extreme) cases

- $\boldsymbol{r = k}$
  - $d \leq n - k + 1$
  - **An $(\boldsymbol{n}, \boldsymbol{k})$ RS code is an $(\boldsymbol{n}, \boldsymbol{k}, \boldsymbol{r} = \boldsymbol{k})$ optimal LRC**
  - $|\mathbb{F}| = O(n)$

$$\boldsymbol{d \leq n - k - \left\lceil \frac{k}{r} \right\rceil + 2}$$

- $\boldsymbol{r = 1}$
  - $d \leq 2\left(\frac{n}{2} - k + 1\right)$
  - **Duplication of an $(\boldsymbol{n}/2, \boldsymbol{k})$ RS code is an $(\boldsymbol{n}, \boldsymbol{k}, \boldsymbol{r} = 1)$ optimal LRC**
  - $|\mathbb{F}| = O(n)$

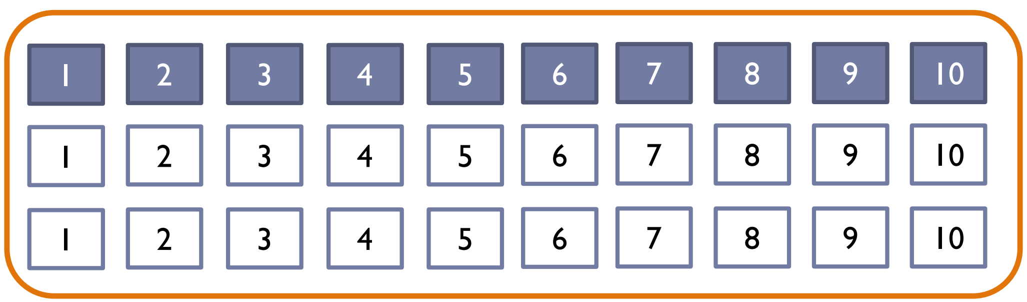
- What happens for $1 < r < k$?

Availability

- A symbol has **availability t** if it can be read in parallel by $t+1$ disjoint groups of symbols
  - These t reads have locality r if they involve up to r symbols each

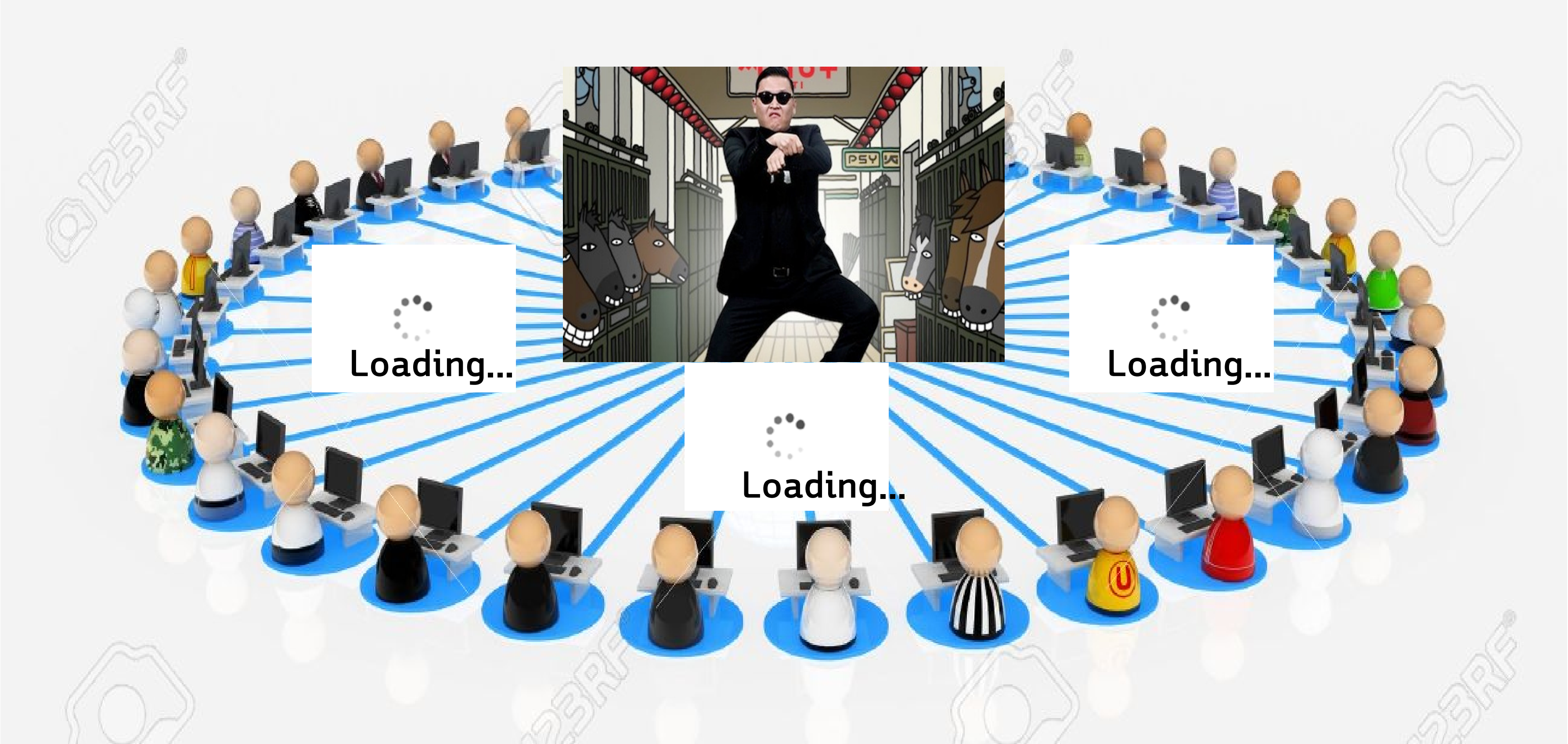
- **Replication** provides **high availability** for **hot data**
  - Example: 3x replication

• Each symbol can be read in parallel $t+1=3$ times.
• Availability $t=2$.
• Locality $r=1$.



Availability for Hot Data

- "Hot data" is accessed simultaneously by a very large number of users

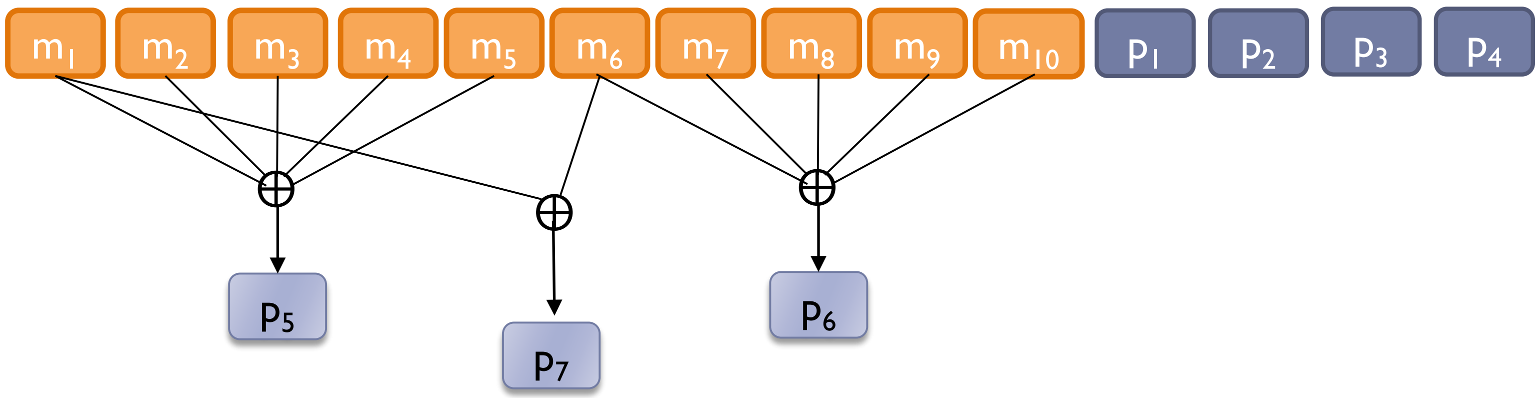


Availability

- Goal
  - A code with **high availability** and **small storage overhead**
- Solution
  - **LRC code with multiple disjoint recovery sets**
  - **This is a code with availability**

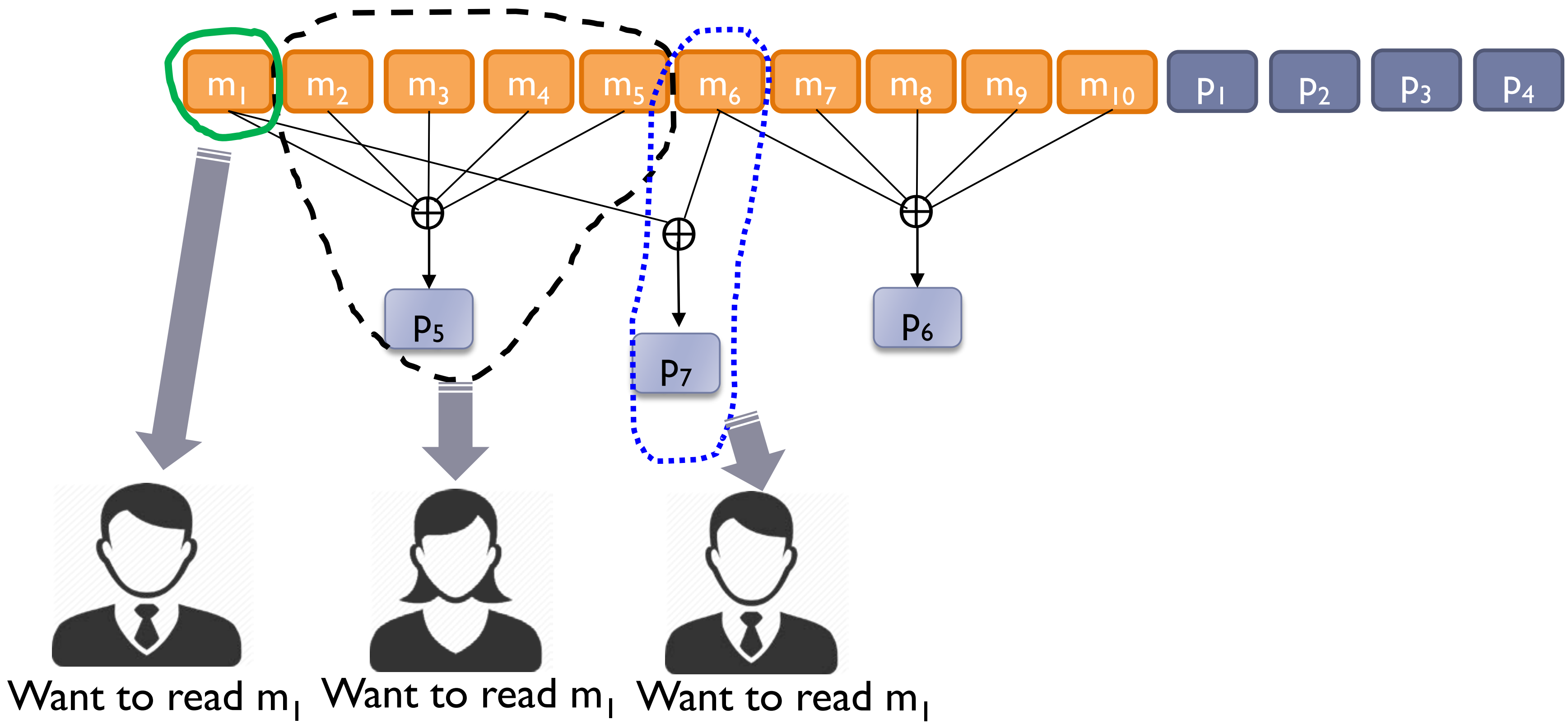
Locality and Availability

- (14, 10) RS code
- Information locality 5
- Availability for m_1



Locality and Availability

- Block m_1 can be read by 1 systematic read and 2 repair reads **simultaneously**



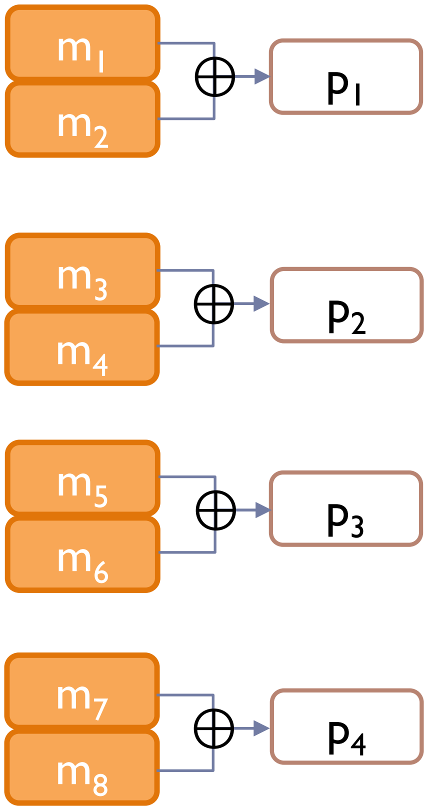
Some constructions for LRC

Local Parity

- An (n, k, r) LRC when $r|k$
  - **Single parity check code** is an LRC with $\boldsymbol{d} = \mathbf{2}$
  - $n = k + \frac{k}{r}$ and $r = 2$ *(with k even)*
  - The minimum distance bound for an (n, k, r) LRC

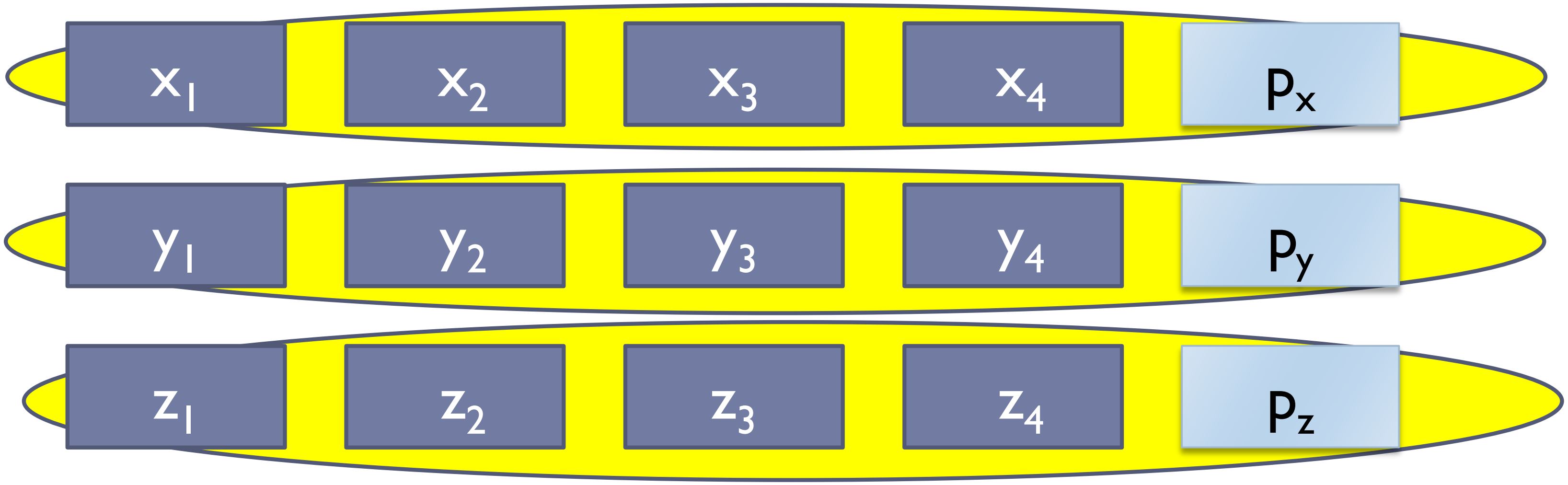
$$d \leq \left(k + \frac{k}{r}\right) - k - \frac{k}{r} + 2 = 2$$

  - This is **an optimal LRC** for given n and k



Optimal LRC Example

3 repair groups



Single parity check code is an LRC with $\boldsymbol{d = 2}$

$n = k + \frac{k}{r} = 12 + 3 = 15$ and $r = 4$

The minimum distance bound for an (n, k, r) LRC

$$d \leq \left(k + \frac{k}{r}\right) - k - \frac{k}{r} + 2 = 2$$

Single parity check code is optimal (**when** $r|k$)

- Take r info symbols at a time and add a single parity check for them
- Repeat this for all info symbols of size r times some number, say, m.

- Length $\boldsymbol{n} = \boldsymbol{m}(\boldsymbol{r} + \mathbf{1})$
- Dimension $\boldsymbol{k} = \boldsymbol{rm}$ so $(k/r = m)$

- Minimum distance $\boldsymbol{d} = \mathbf{2}$ and RHS=$(n-k) - \frac{k}{r} + 2 = m - m + 2 = 2$

  $\rightarrow$ optimal
- Coderate $k/n = rm/m(r+1) = r/(r+1)$

  $\rightarrow$ optimal in the sense of coderate

Global Parity and Local Parity

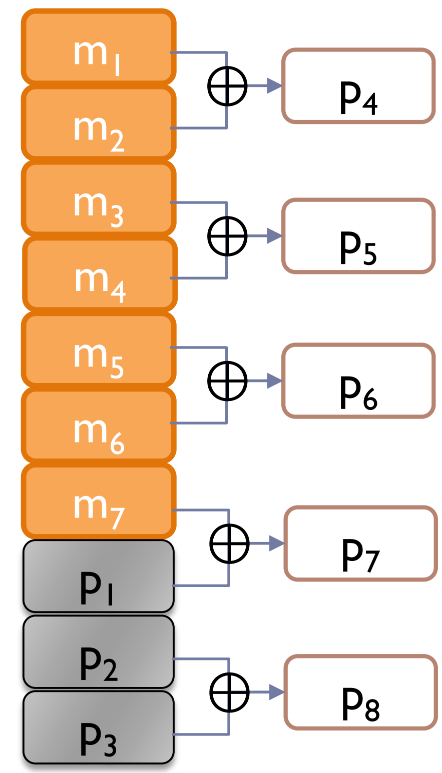
- An (n, k, r) LRC when $r + 1 | n$
  - (n', k) MDS code, and then, use
    $(r + 1, r)$ single parity check code (for every r of them)
  - $n = n' \frac{(r+1)}{r}$ and $n' - k + 1 \leq d \leq n' - k + 2$
  - The minimum distance bound for an (n, k, r) LRC

$$d < \left(n' \frac{r+1}{r}\right) - k - \left\lceil \frac{k}{r} \right\rceil + 2$$

  - This is **not an optimal** LRC

(15,10) SPC

(10,7) MDS



d = 4 **r = 2**

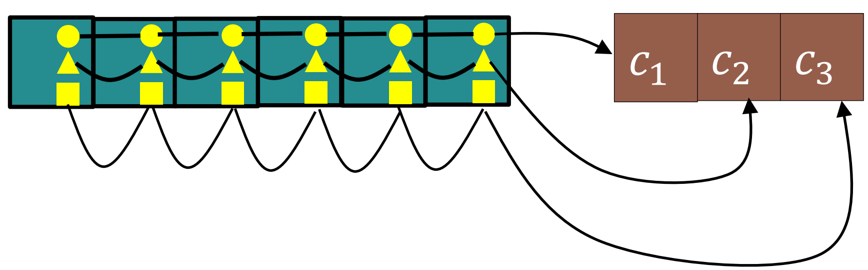
$4 < 15 - 7 - 2 = 6$

Pyramid Code – Information Locality
[Chen-Huang-Li'07]

- Information Locality
  - Take an $(k+d-1,\ k)_q$ Reed-Solomon code

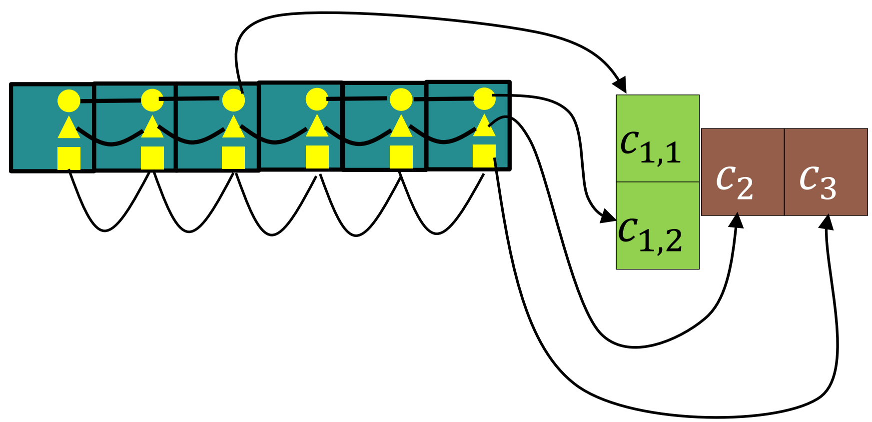
○△□ represents some suitable coefficients when the parity symbol is generated.

(9,6)
MDS



- **Split the first parity so that each cover 1/2 of info symbols**
  - This gives $n = (k+d-1) + \frac{k}{r} - 1 = k\left(1+\frac{1}{r}\right) + d - 2$

(10,6,3)
Pyramid



Information Locality

$r = \frac{\boldsymbol{k}}{\boldsymbol{2}} = 3$

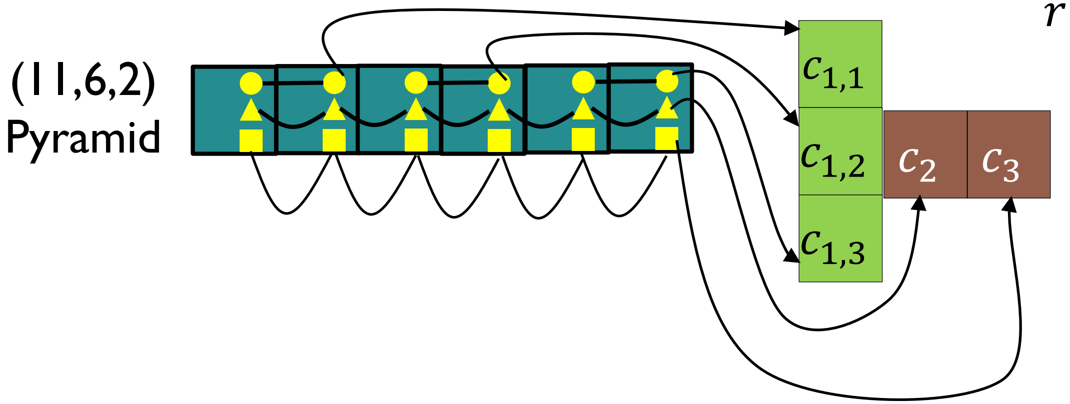
Pyramid Code – Information Locality

[Chen-Huang-Li'07]

- Or, cover 1/3 of info symbols

Information Locality

$$r = \frac{\boldsymbol{k}}{\mathbf{3}} = 2$$



- Pyramid codes can be obtained from **any systematic MDS codes** with d.
- Assume that the first parity check symbol is the sum $\sum_{i=1}^{k} x_i$ of info symbols.
- Replace this with $\left\lceil \frac{k}{r} \right\rceil$ parity checks each of size at most r on disjoint info symbols.
- Then, the resulting code $\mathcal{C}$ has information locality r and distance d, while the redundancy is given by

$$\boldsymbol{n - k = \left\lceil \frac{k}{r} \right\rceil + d - 2.}$$

Therefore, Pyramid codes are optimal.

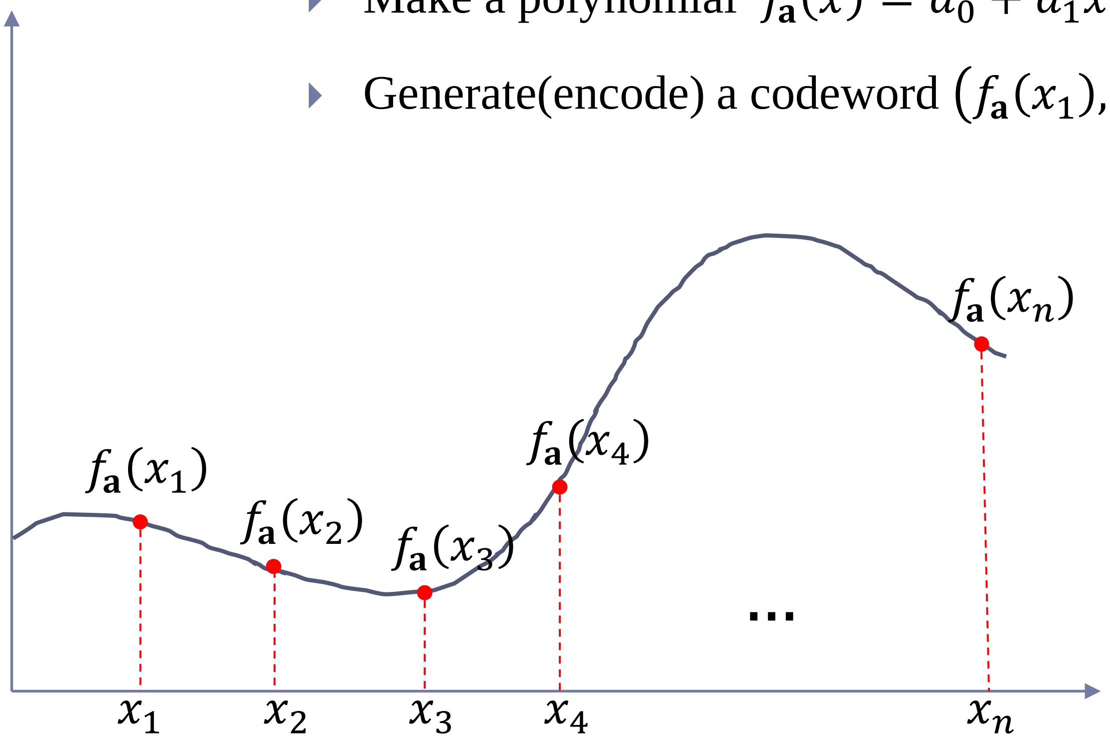
Explicit Codes with All-symbol Locality

- [Silberstein-Rawat-Koyluoglu-Vishwanath`13]
  - Optimal length codes with all-symbol locality for $q = 2^n$
  - Constructions based on **Gabidulin codes** (maximum rank distance code)

- [Tamo-Barg`14]
  - Optimal length codes with all-symbol locality for $q = O(n)$
  - Constructions based on **RS codes**

Original view of RS codes

- Fix n points: $x_1, x_2, x_3, \ldots, x_n$
- Given message vector $\mathbf{a} = \{a_0, a_1, a_2, a_3\}$
- Make a polynomial $f_{\mathbf{a}}(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$
- Generate(encode) a codeword $\left(f_{\mathbf{a}}(x_1), f_{\mathbf{a}}(x_2), \ldots, f_{\mathbf{a}}(x_n)\right)$



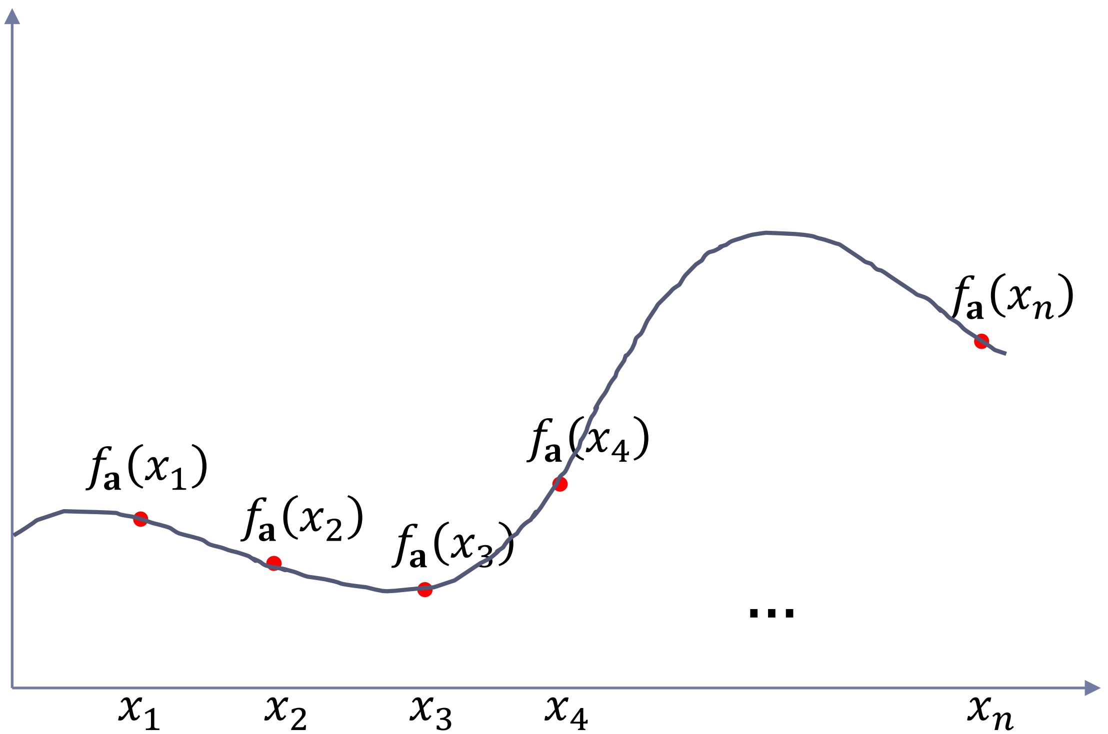
Intuition behind the RS-like LRC

[Tamo-Barg`14]

- A symbol $f_{\mathbf{a}}(x_1)$ is erased

$$\left(\cancel{f_{\mathbf{a}}(x_1)}, f_{\mathbf{a}}(x_2), \ldots, f_{\mathbf{a}}(x_n)\right)$$

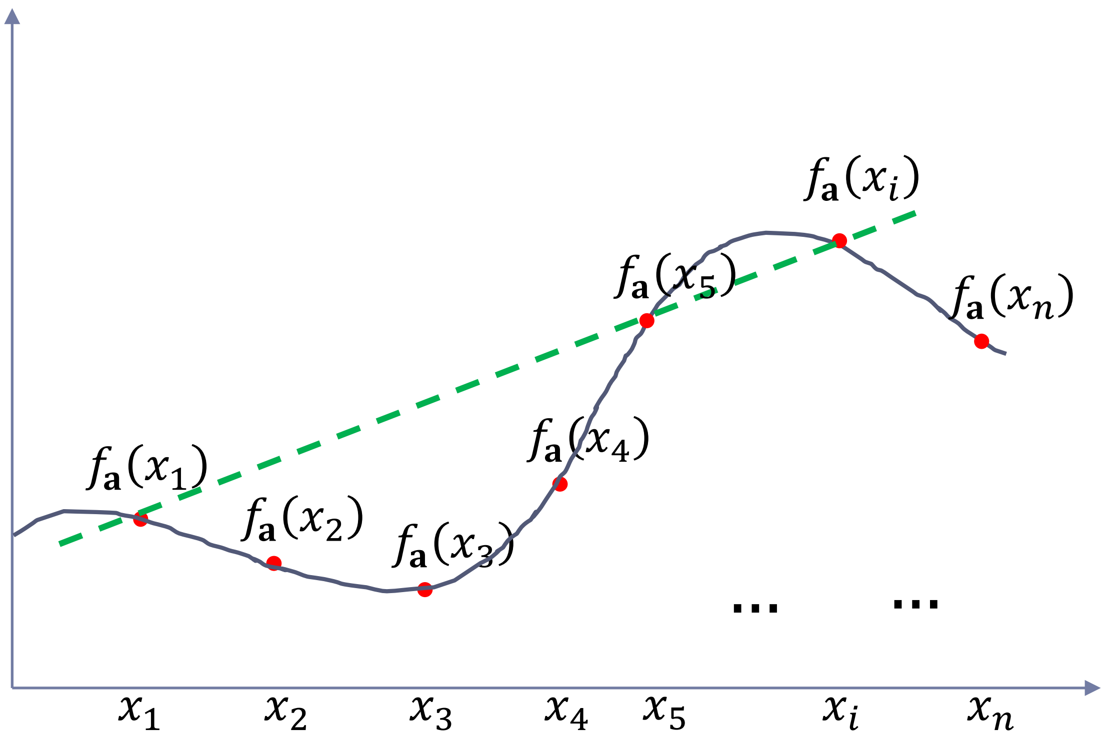
- $f_{\mathbf{a}}(x_1)$ can be recovered by RS decoding



Intuition behind the RS-like LRC

[Tamo-Barg`14]

- Assume that there **is a linear polynomial that passes through the points $f_{\mathbf{a}}(x_1), f_{\mathbf{a}}(x_5)$ and $f_{\mathbf{a}}(x_i)$** of the codeword
  - Only two points suffice to recover the lost point $f_{\mathbf{a}}(x_1)$



RS-like Codes

- $A_1, A_2, \dots, A_{\frac{n}{r+1}}$ are **disjoint subsets,** each of size $\boldsymbol{r+1}$
- $\boldsymbol{g(x) \in \mathbb{F}[x]}$ **is a polynomial** such that
  - $\deg(g(x)) = \boldsymbol{r+1}$
  - $g(x)$ is **constant** on each subset $\boldsymbol{A_j}$: $g(\alpha) = g(\beta)$, for $\alpha, \beta \in A_j$
- **Encoding:**
  - Given $\boldsymbol{k}$ information symbols $\boldsymbol{a_{i,j}}$, $i = 0,1,\dots,r-1, j = 0,1,\dots,\frac{k}{r}-1$
  - Define the polynomial

$$f_{\boldsymbol{a}}(x) = \sum_{i=0}^{r-1} \boldsymbol{x^i} \sum_{j=0}^{\frac{k}{r}-1} \boldsymbol{a_{i,j}}\, g(x)^j$$

  - The codeword of length n is $\left(f(\alpha):\ \alpha \in \bigcup_{i=1}^{\frac{n}{r+1}} A_i\right)$

Example : $(n = 9, k = 4, r = 2)$ LRC code over $\mathbb{F}_q$

- Since we need 9 distinct evaluation points of the field, we **must** choose $q \geq 9$.
- We will define the code $\mathcal{C}$ over $\mathbb{F}_{13}$.
- Let the partition of all the nonzero elements of $\mathbb{F}_{13}$ be as follows:

$$\boldsymbol{A_1 = \{1, 3, 9\}}, \qquad \boldsymbol{A_2 = \{2, 6, 5\}} = 2A_1, \qquad \boldsymbol{A_3 = \{4, 12, 10\}} = 4A_1$$

- Note that $g(x) = x^3$ is **constant** on any set A_i

NOT a magic

- Let $a = (a_{0,0}, a_{0,1}, a_{1,0}, a_{1,1})$ be the information vector of length $k = 4$ over $\mathbb{F}_{13}$
- Define the encoding polynomial

degree $< r$

$$f_a(x) = \left(a_{0,0}g(x)^0 + a_{0,1}g(x)^1\right) + x\left(a_{1,0}g(x)^0 + a_{1,1}g(x)^1\right)$$

$$= x^0\underline{\left(a_{0,0} + a_{0,1}x^3\right)} + x^1\underline{\left(a_{1,0} + a_{1,1}x^3\right)}$$

$$= a_{0,0} + a_{1,0}x + a_{0,1}x^3 + a_{1,1}x^4$$

$a_{0,0} + a_{0,1}x^3$ is constant on all $x \in A_i$

$a_{1,0} + a_{1,1}x^3$ is constant on all $x \in A_i$

Locality

- **It is one less than the size of each subset A_i** **Therefore, it is r**
- How to recover $\boldsymbol{f(\alpha)}$ for $\boldsymbol{\alpha \in A_1}$?
- Define $\boldsymbol{f_i(x) = \sum_{j=0}^{\frac{k}{r}-1} a_{i,j}\, g(x)^j}$ so that the encoding polynomial is

$$f(x) = \sum_{i=0}^{r-1} x^i \sum_{j=0}^{\frac{k}{r}-1} a_{i,j}\, g(x)^j = \sum_{i=0}^{r-1} x^i\, \boldsymbol{f_i(x)}$$

  - ✓ **Observe that $\boldsymbol{f_i(x)}$ is constant on the any set $\boldsymbol{A_j}$ ($\because$so is $\boldsymbol{g(x)}$).**
- Define $\delta(x) = \sum_{i=0}^{r-1} x^i\, \boldsymbol{f_i(\alpha)}$.
  - ✓ Then $\boldsymbol{f(\alpha) = \delta(\alpha)}$, and $\deg(\delta(x)) \leq r-1$
  - ✓ Any $\boldsymbol{r}$ points on $\delta(x)$ will suffice to recover $\delta(x)$
- Use the r values $\{\delta(\beta) = f(\beta) : \beta \in A_1 \setminus \alpha\}$ to recover $\delta(x)$

Example (continued):

- The codeword c that corresponds to a is found as **the evaluation of the polynomial** f_a at all the points of the sets of the partition $\mathcal{A}$
- If $\mathbf{a} = (1, 1, 1, 1)$, then the codeword becomes
$$(\boldsymbol{f}_{\mathbf{a}}(\mathbf{1}), \boldsymbol{f}_{\mathbf{a}}(\mathbf{3}), \boldsymbol{f}_{\mathbf{a}}(\mathbf{9}), \boldsymbol{f}_{\mathbf{a}}(\mathbf{2}), \boldsymbol{f}_{\mathbf{a}}(\mathbf{6}), \boldsymbol{f}_{\mathbf{a}}(\mathbf{5}), \boldsymbol{f}_{\mathbf{a}}(\mathbf{4}), \boldsymbol{f}_{\mathbf{a}}(\mathbf{12}), \boldsymbol{f}_{\mathbf{a}}(\mathbf{10}))$$
$$= (\mathbf{4}, \mathbf{8}, \mathbf{7}, \mathbf{1}, \mathbf{11}, \mathbf{2}, \mathbf{0}, \mathbf{0}, \mathbf{0})$$
- suppose that the value $c_1 = f_{\mathbf{a}}(1)$ is erased

using polynomial interpolation

- ✓ Find the unique polynomial $\delta(x)$ of degree less than $r = 2$ such that $\delta(\beta) = f_{\mathbf{a}}(\beta)$ for all $\beta \in A_1 \backslash 1$
- ✓ It can be recovered by accessing two other codeword symbols at locations corresponding to 3 and 9

Example (continued): Polynomial Interpolation

- Given a set of $k+1$ points
$$(x_0, y_0), (x_1, y_1), \ldots, (x_j, y_j), \ldots, (x_k, y_k)$$
where no two x_j's are the same,
- the interpolation polynomial in the Lagrange form is a linear combination
$$L(x) = \sum_{j=0}^{k} y_j \boldsymbol{l_j(x)}$$
of degree at most $\boldsymbol{k}$

of Lagrange basis polynomials

$$\boldsymbol{l_j(x)} = \prod_{\substack{0 \le m \le k \\ m \ne j}} \frac{x - x_m}{x_j - x_m} = \frac{x - x_0}{x_j - x_0} \cdot \frac{x - x_1}{x_j - x_1} \cdot \ldots \cdot \frac{x - x_{j-1}}{x_j - x_{j-1}} \cdot \frac{x - x_{j+1}}{x_j - x_{j+1}} \cdot \ldots \cdot \frac{x - x_k}{x_j - x_k}$$

with $\boldsymbol{l_j(x_j) = 1}$ **and** $\boldsymbol{l_{j \ne i}(x_i) = 0}$

- It follows that $\boldsymbol{y_j l_j(x_j) = y_j}$
- Therefore, at each point $\boldsymbol{x_i}$,
$$\boldsymbol{L(x_i) = y_i + 0 + 0 + \cdots + 0 = y_i}$$

Example (continued): recovery

- To find one erased symbol, we need to perform polynomial interpolation from $r = 2$ known symbols in its recovery set $\{\cancel{(1,4)}, (\mathbf{3}, \mathbf{8}), (\mathbf{9}, \mathbf{7})\}$
  - Now, the interpolation polynomial $\delta(x)$ is

    $$\boldsymbol{\delta(x) = f_{\mathrm{a}}(3) l_3(x) + f_{\mathrm{a}}(9) l_9(x)},$$

    $$\boxed{l_3(x) = \frac{x-9}{3-9}, \qquad l_9(x) = \frac{x-3}{9-3}}$$

    where $l_i(x) = \prod_{j \in \{3,9\} \setminus i} \frac{x-j}{i-j}$
  - $$\boldsymbol{\delta(x) = 8 \cdot \frac{x-9}{3-9} + 7 \cdot \frac{x-3}{9-3} = 2x + 2}$$
- Therefore, we can find the erased value $\boldsymbol{f_{\mathrm{a}}(1) = \delta(1) = 4}$

Optimality

- Encoding polynomial

$$\boldsymbol{f}_{\mathbf{a}}(\boldsymbol{x}) = \sum_{i=0}^{r-1} x^i f_i(x) = \sum_{i=0}^{r-1} x^i \sum_{j=0}^{\frac{k}{r}-1} a_{i,j}\, g(x)^j,$$

  - k polynomials $g(x)^j x^i$ all are of distinct degrees, and therefore are linearly independent over $\mathbb{F}$

- The deg of $\boldsymbol{f}_{\mathbf{a}}(\boldsymbol{x})$ is at most $\left(\frac{k}{r} - 1\right)(r+1) + r - 1 = k + \frac{k}{r} - 2 = n - 2$
  - Two distinct encoding polynomials gives rise to two distinct code-vectors
  - So the dimension of the code is $\boldsymbol{k}$

- The code distance satisfies

$$\boldsymbol{d(C) \geq n - \max_{f_a,\, a \in \mathbb{F}_q^k} \deg(f_a) = n - k - \frac{k}{r} + 2}$$

Binary Locally Repairable Codes

[Shahabinejad-Khabbanzian-Ardakani`14]

- Using a Parity-check matrix
  - $n = 15, k = 10$
  - This code has $\boldsymbol{d_{min} = 4}$ and locality $\boldsymbol{r = 6}$
    - Best achievable minimum distance for given n and k

$\boldsymbol{d_{min} = 4}$ because

no 3 columns are linearly dependent
But some 4 columns are dependent

$$G = \begin{bmatrix} 00111\,1000000000 \\ 01011\,0100000000 \\ 01101\,0010000000 \\ 01110\,0001000000 \\ 10011\,0000100000 \\ 10101\,0000010000 \\ 10110\,0000001000 \\ 11100\,0000000100 \\ 11010\,0000000010 \\ 11001\,0000000001 \end{bmatrix} \Leftrightarrow H = \begin{bmatrix} 10000\,0000111111 \\ 01000\,0111000111 \\ 00100\,1011011100 \\ 00010\,1101101010 \\ 00001\,1110110001 \end{bmatrix}$$

G: $\boxed{P^T}\ \boxed{I_{10\times10}}$

H: $\boxed{I_{5\times5}}\ \boxed{P}$

Every row of H has weight 7

1st row: $\boldsymbol{c_1 + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} + c_{15} = 0}$

➔ $\boldsymbol{c_1 = c_{10} + c_{11} + c_{12} + c_{13} + c_{14} + c_{15}}$

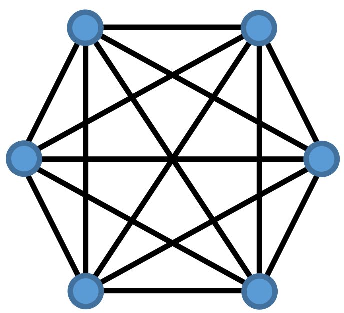
Binary Locally Repairable Codes

[Kim-Nam-Song`15]

- Using a Generator Matrix
  - Complete graph: $d = k, t = d - 1$

This gives all-symbol availability

Example: $n = 21, k = 6, d = 6, r = 2$

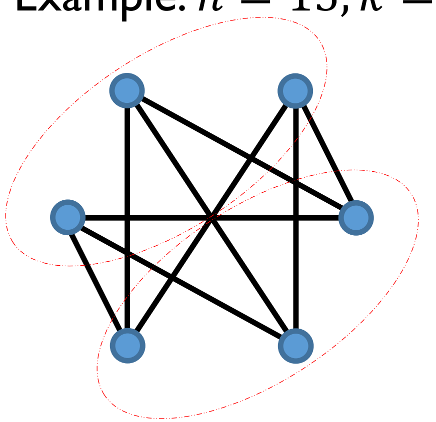


$$G = \left(\begin{array}{cccccc|ccccccccccccccc}
1&0&0&0&0&0&1&1&0&1&0&0&1&0&0&0&1&0&0&0&0\\
0&1&0&0&0&0&1&0&1&0&1&0&0&1&0&0&0&1&0&0&0\\
0&0&1&0&0&0&0&1&1&0&0&1&0&0&1&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0&0&1&1&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&1&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&1\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1
\end{array}\right)$$

  - p-partite graph: $d = k - \frac{k}{p} + 1, t = d - 1$

This gives information-symbol availability

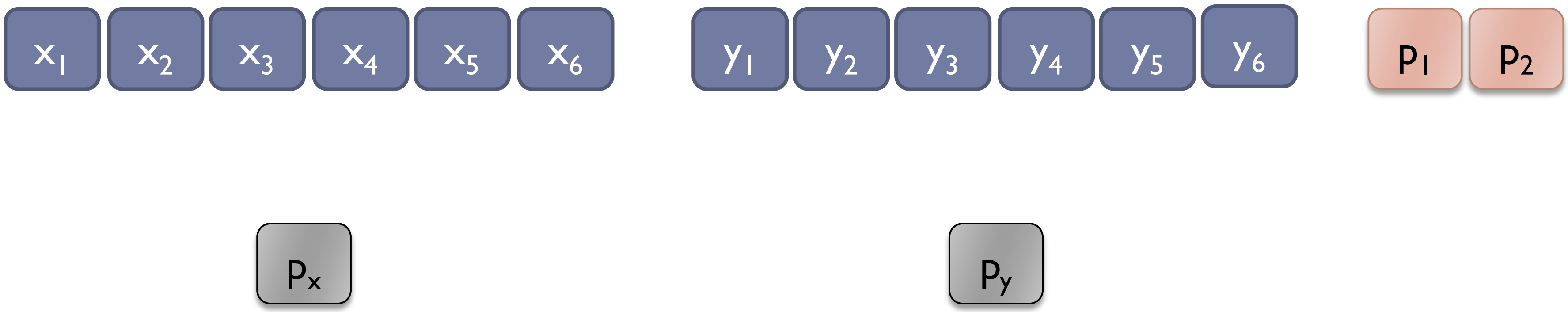
Example: $n = 15, k = 6, d = 4, r = 2$



$$G = \left(\begin{array}{cccccc|ccccccccc}
1&0&0&0&0&0&1&1&1&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&1&1&1&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&1&1&1\\
\hline
0&0&0&1&0&0&1&0&0&1&0&0&1&0&0\\
0&0&0&0&1&0&0&1&0&0&1&0&0&1&0\\
0&0&0&0&0&1&0&0&1&0&0&1&0&0&1
\end{array}\right)$$

Innovative Proposals

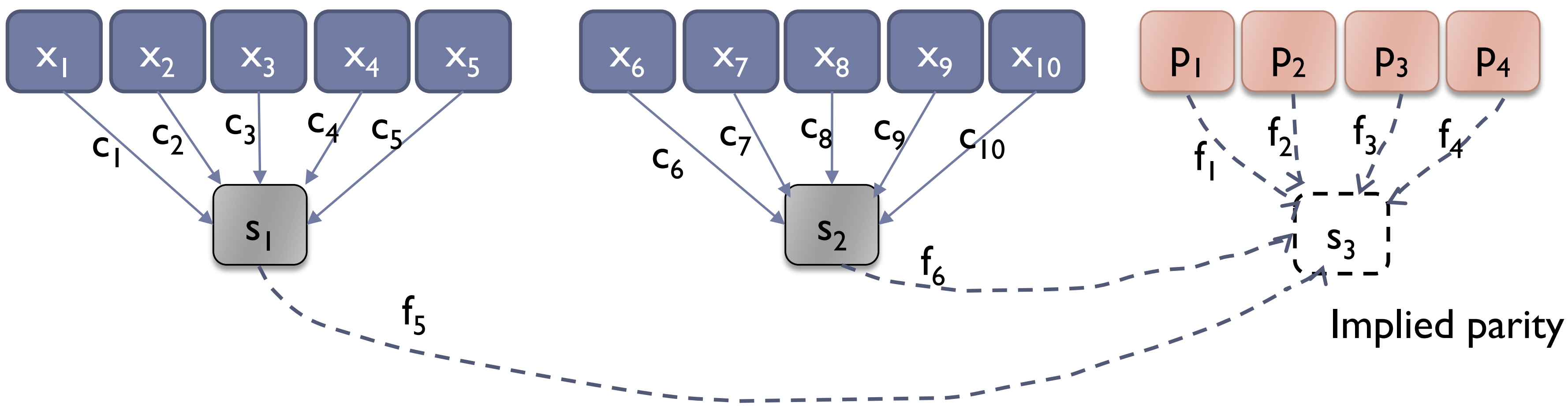
- Erasure codes with some form of locality
- Microsoft Azure Storage
  - **(16, 12, 6) LRC**



- Global parities p_1,p_2 are found from all x_i,y_i, $i=1,2,\ldots,6$
- Local parities p_x, p_y provide local recovery for the information symbols

Innovative Proposals

- HDFS Xorbas (Facebook)
  - **(16, 10, 5) LRC**



- Global parity

  (14,10) RS code

- Local parity

  $s_1=c_1x_1+c_2x_2+c_3x_3+c_4x_4+c_5x_5$

  $s_2=c_6x_6+c_7x_7+c_8x_8+c_9x_9+c_{10}x_{10}$

One additional optimization

$s_1+s_2+s_3=0$

$x_3=c_3^{-1}(s_1-c_1x_1-c_2x_2-c_4x_4-c_5x_5)$

$p_1=f_1^{-1}(-s_1-s_2-f_2p_2-f_3p_3-f_4p_4)$

References

- A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, K. Ramchandran, “Network coding for distributed storage systems,” IEEE Trans. Infor. Theory, vol. 56, no. 9, pp. 4539-4551, Sep. 2010.
- P. Gopalan, C. Huang, H. Simitci, S. Yekhanin, “On the locality of codeword symbols,” IEEE Trans. Infor. Theory, vol. 58, no. 11, pp. 6925-6934, Nov. 2012.
- C. Huang, M. Chen, J. Li, “Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems,” in Proc. IEEE Intern. Symp. Network Comput. App. (NCA), 2007.
- F. Oggier, A. Datta, “Self-repairing homomorphic codes for distributed storage systems,” in Proc. IEEE INFOCOM, 2011.
- D. S. Papailiopoulos, J. Luo, A. G. Dimakis, C. Huang, J. Li, “Simple regenerating codes: Network coding for cloud storage,” in Proc. IEEE INFOCOM, 2012.
- I. Tamo, D. S. Papailiopoulos, A. G. Dimakis, “Optimal locally repairable codes and connections to matroid theory,” IEEE Trans. Infor. Theory vol. 62, no. 12, pp. 6661-6671, Dec. 2016.
- N. Silberstein, A. S. Rawat, O. O. Koyluoglu, S. Vishwanath, “Optimal locally repairable codes via rank-metric codes,” in Proc. IEEE Intern. Symp. Infor. Theory (ISIT), 2013.
- I. Tamo, A. Barg, “A family of optimal locally recoverable codes,” IEEE Trans. Infor. Theory, vol. 60, no. 8, pp. 4661-4676, Aug. 2014.
- M. Shahabinejad, M. Khabbazian, M. Ardakani, “An efficient binary locally repairable code for hadoop distributed file system,” IEEE Commun. Letters, vol. 18, no. 8, pp. 1287-1290, Jul. 2014.
- S. E. Rouayheb and K. Ramchandran, “Fractional repetition codes for repair in distributed storage systems,” 48th Annu. Allerton Conf. Control, Comput., Commun., Sep. 2010.
- Jung-Hyun Kim and Hong-Yeop Song, "Hypergraph-based Binary Locally Repairable Codes with Availability," IEEE Communication Letters ,vol. 21, no. 11, pp.2332-2335, November 2017.
- Mi-Young Nam and Hong-Yeop Song, "Binary Locally Repairable Codes with Minimum Distance at least 6 based on Partial t-Spreads ," IEEE Communications Letters ,vol. 21, no. 8, pp.1683-1686, August 2017.
- Jung-Hyun Kim and Hong-Yeop Song, "Alphabet-Dependent Bounds for Locally Repairable Codes With Joint Information Availability ," IEEE Communications Letters ,vol. 21, no. 8, pp.1687-1690, August 2017.