T1: Erasure Codes for Storage Applications

James S. Plank

Associate Professor
Department of Computer Science
University of Tennessee
plank@cs.utk.edu

USENIX FAST 2005 4th USENIX Conference on File and Storage Technologies
DEC. 13-16 / 2005 / Holiday Inn Golden Gateway / San Francisco / CA / USA

---

http://www.cs.utk.edu/~plank
"Research Papers"

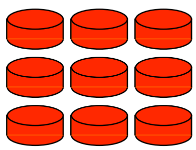
James S. Plank

Associate Professor
Department of Computer Science
University of Tennessee
plank@cs.utk.edu

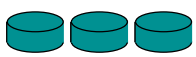
USENIX FAST 2005 4th USENIX Conference on File and Storage Technologies
DEC. 13-16 / 2005 / Holiday Inn Golden Gateway / San Francisco / CA / USA

What is an Erasure Code?

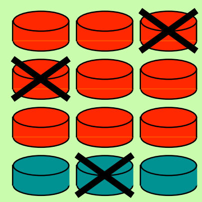
A technique that lets you take *n* storage devices:



Encode them onto *m* additional storage devices:



And have the entire system be resilient to up to *m* device failures:



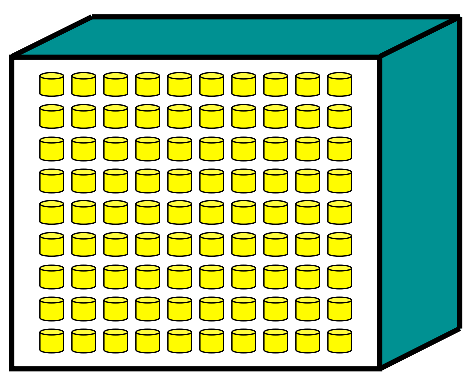
When are they useful?

Anytime you need to tolerate failures.

<u>*For example*</u>:

Disk Array Systems

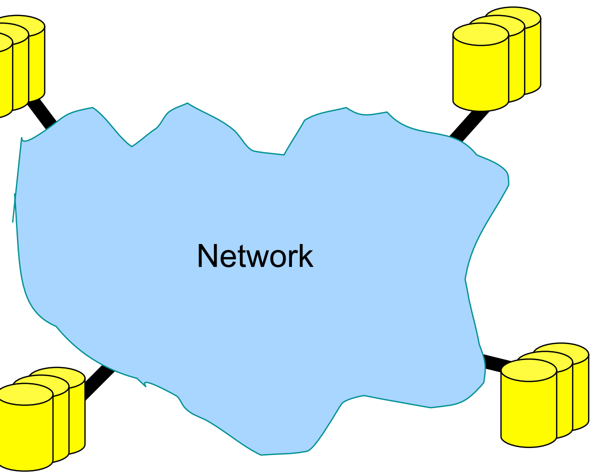
$MTTF_{first} = MTTF_{one}/n$



When are they useful?

Anytime you need to tolerate failures.

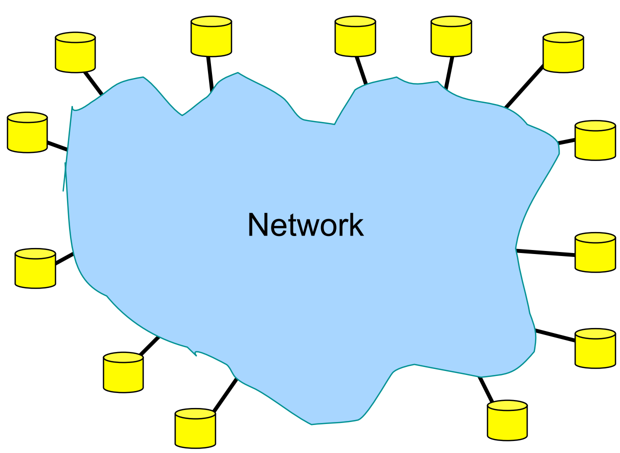
Data Grids



When are they useful?

Anytime you need to tolerate failures.

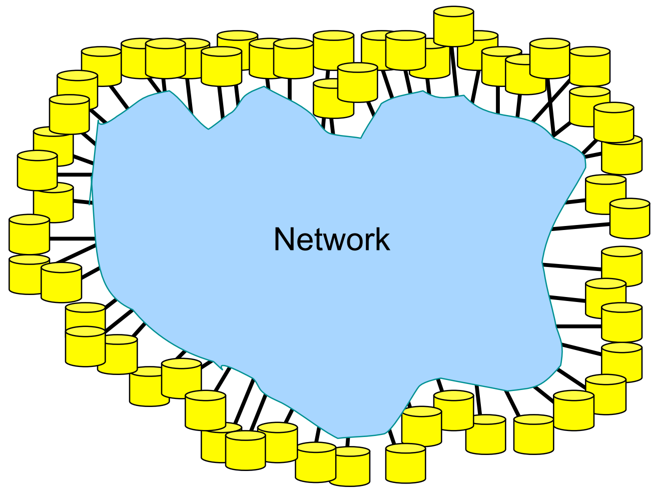
Collaborative/
Distributed
Storage
Applications



When are they useful?

Anytime you need to tolerate failures.

Peer-to-peer applications.

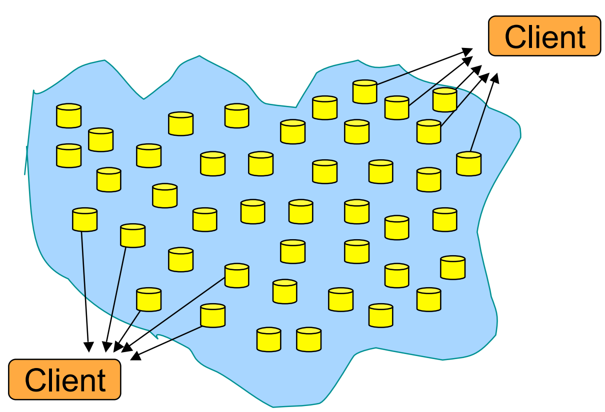


When are they useful?

Anytime you need to tolerate failures.

Distributed Data
or
Object Stores:

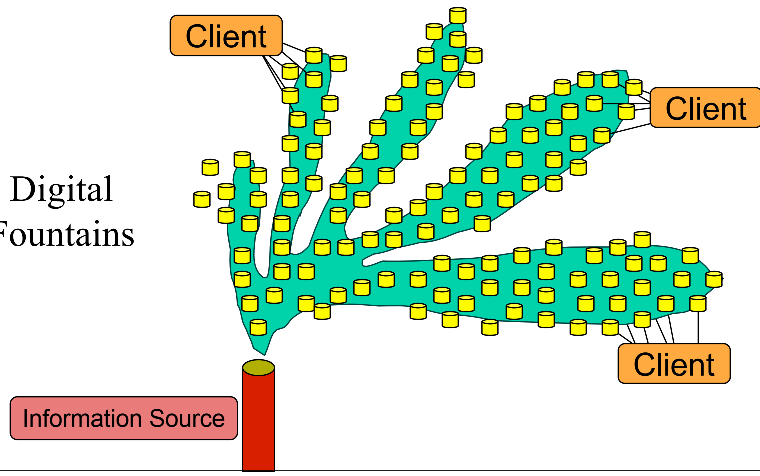
(Logistical Apps.)



When are they useful?

Anytime you need to tolerate failures.

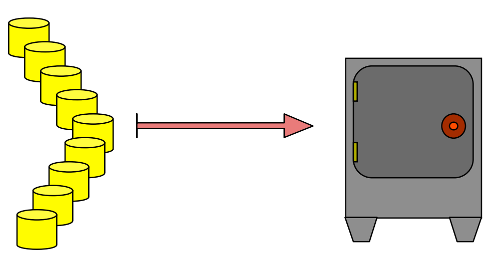
Digital Fountains



When are they useful?

Anytime you need to tolerate failures.

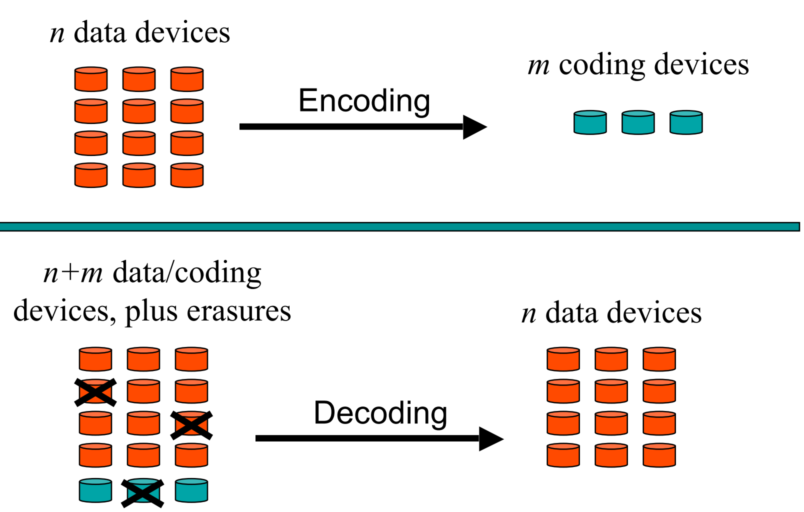
Archival Storage.



Terms & Definitions

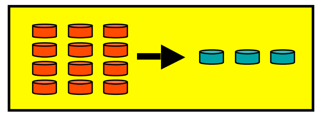
- Number of data disks: *n*
- Number of coding disks: *m*
- Rate of a code: $R = n/(n+m)$
- Identifiable Failure: "Erasure"

The problem, once again

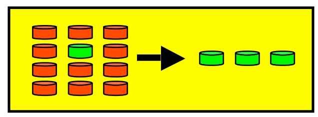


Issues with Erasure Coding

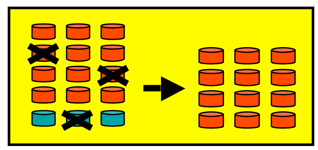
- Performance
  - Encoding
    - Typically *O(mn)*, but not always.



  - Update
    - Typically *O(m)*, but not always.



  - Decoding
    - Typically *O(mn)*, but not always.



Issues with Erasure Coding

- Space Usage
  - Quantified by two of four:
    - Data Devices: n
    - Coding Devices: m
    - Sum of Devices: $(n+m)$
    - Rate: $R = n/(n+m)$
  - Higher rates are more space efficient, but less fault-tolerant.

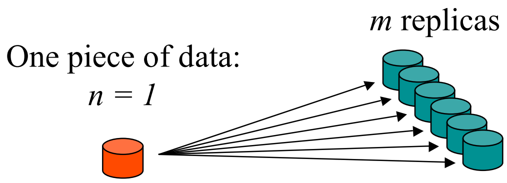
Issues with Erasure Coding

- Failure Coverage - Four ways to specify
  - Specified by a threshold:
    - (e.g. 3 erasures always tolerated).
  - Specified by an average:
    - (e.g. can recover from an average of 11.84 erasures).
  - Specified as MDS (Maximum Distance Separable):
    - MDS: Threshold = average = m.
    - Space optimal.
  - Specified by Overhead Factor f:
    - f = factor from MDS = m/average.
    - f is always >= 1
    - f = 1 is MDS.

Issues with Erasure Coding

- Flexibility
  - Can you arbitrarily add data / coding nodes?
  - (Can you change the rate)?
  - How does this impact failure coverage?

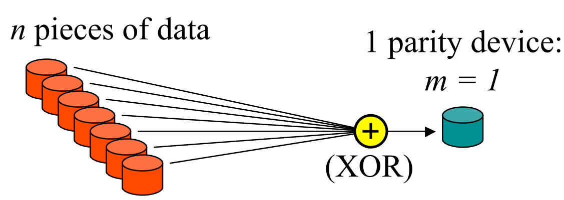
Trivial Example: Replication



Can tolerate any *m* erasures.

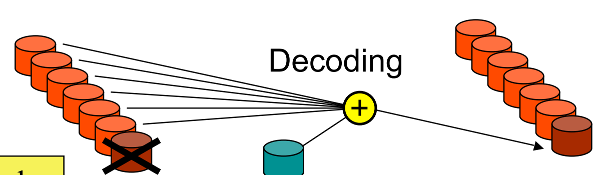
- MDS
- Extremely fast encoding/decoding/update.
- Rate: $R = 1/(m+1)$ - Very space inefficient
- There are many replication/based systems *(P2P especially).*

Less Trivial Example: Simple Parity

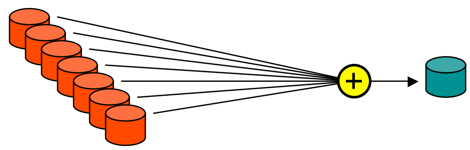


Can tolerate any single erasure.

For example:



Evaluating Parity



- MDS
- Rate: $R = n/(n+1)$ - Very space efficient
- Optimal encoding/decoding/update:
  - n-1 XORs to encode & decode
  - 2 XORs to update
- Extremely popular (RAID Level 5).
- Downside: $m = 1$ is limited.

Unfortunately

- Those are the last easy things you'll see.
- For $(n > 1, m > 1)$, there is no consensus on the best coding technique.
- They *all* have tradeoffs.

The Point of This Tutorial

- To introduce you to the various erasure coding techniques.
  - Reed Solomon codes.
  - Parity-array codes.
  - LDPC codes.
- To help you understand their tradeoffs.
- To help you evaluate your coding needs.
  - This too is not straightforward.

Why is this such a pain?

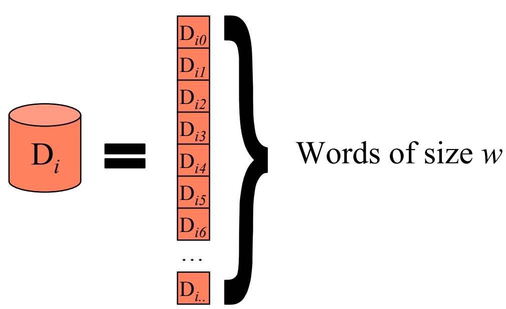
- Coding theory historically has been the purview of coding theorists.
- Their goals have had their roots elsewhere (noisy communication lines, byzantine memory systems, etc).
- They are not systems programmers.
- (They don't care…)

Part 1: Reed-Solomon Codes

- The only MDS coding technique for arbitrary *n* & *m*.
- This means that *m* erasures are always tolerated.
- Have been around for decades.
- Expensive.
- I will teach you standard & Cauchy variants.

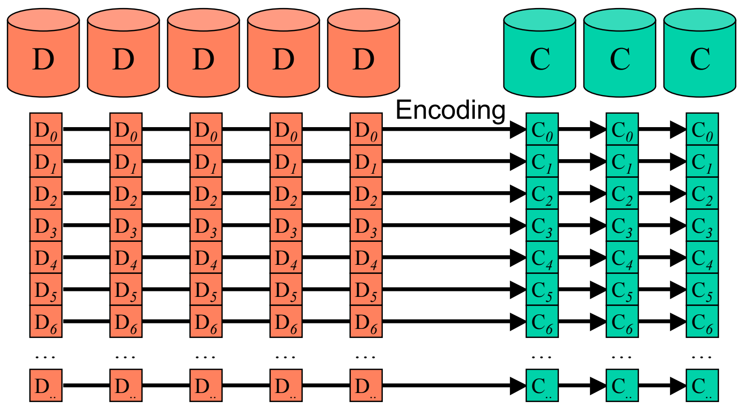
Reed-Solomon Codes

- Operate on binary words of data, composed of *w* bits, where $2^w \geq n+m$.



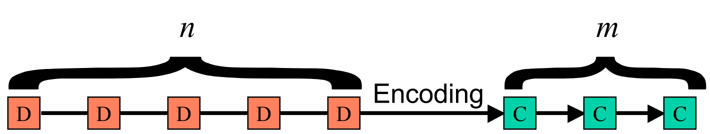
Reed-Solomon Codes

- Operate on binary words of data, composed of w bits, where $2^w \geq n+m$.



Reed-Solomon Codes

- This means we only have to focus on words, rather than whole devices.



- Word size is an issue:
  - If $n+m \leq 256$, we can use bytes as words.
  - If $n+m \leq 65{,}536$, we can use shorts as words.

Reed-Solomon Codes

- Codes are based on linear algebra.
  - First, consider the data words as a column vector *D*:

$$\underbrace{\left.\begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \end{bmatrix}\right\} n}_{D}$$

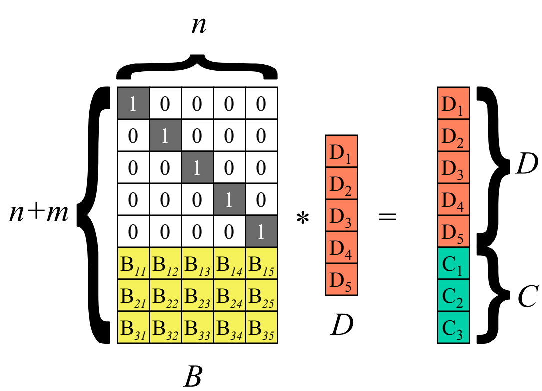
Reed-Solomon Codes

- Codes are based on linear algebra.
  - Next, define an *(n+m)*n* "Distribution Matrix" *B,* whose first *n* rows are the identity matrix:

$$\underbrace{n{+}m\left\{\overbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ B_{11} & B_{12} & B_{13} & B_{14} & B_{15} \\ B_{21} & B_{22} & B_{23} & B_{24} & B_{25} \\ B_{31} & B_{32} & B_{33} & B_{34} & B_{35} \end{bmatrix}}^{n}\right.}_{B} \quad \underbrace{\left.\begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \end{bmatrix}\right\} n}_{D}$$

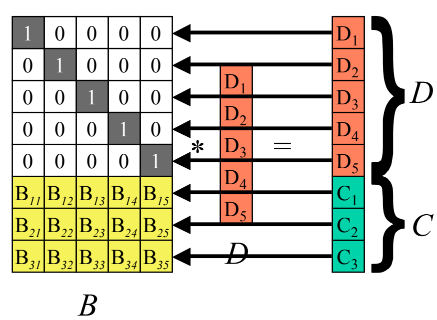
Reed-Solomon Codes

- Codes are based on linear algebra.
  - *B*D* equals an *(n+m)*1* column vector composed of *D* and C (the coding words):



Reed-Solomon Codes

- This means that each data and coding word has a corresponding row in the distribution matrix.



Reed-Solomon Codes

- Suppose *m* nodes fail.
- To decode, we create *B'* by deleting the rows of *B* that correspond to the failed nodes.

$$
\underbrace{\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 \\
B_{11} & B_{12} & B_{13} & B_{14} & B_{15} \\
B_{21} & B_{22} & B_{23} & B_{24} & B_{25} \\
B_{31} & B_{32} & B_{33} & B_{34} & B_{35}
\end{bmatrix}}_{B}
\qquad
\begin{bmatrix}
\times \\ D_2 \\ D_3 \\ \times \\ D_5 \\ C_1 \\ \times \\ C_3
\end{bmatrix}
$$

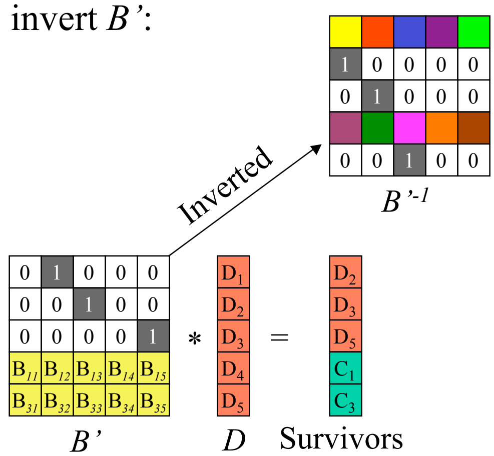
Reed-Solomon Codes

- Suppose *m* nodes fail.
- To decode, we create *B'* by deleting the rows of *B* that correspond to the failed nodes.
- You'll note that *B'*D* equals the survivors.

$$
\underbrace{\begin{bmatrix}
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 \\
B_{11} & B_{12} & B_{13} & B_{14} & B_{15} \\
B_{31} & B_{32} & B_{33} & B_{34} & B_{35}
\end{bmatrix}}_{B'}
*
\underbrace{\begin{bmatrix}
D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5
\end{bmatrix}}_{D}
=
\underbrace{\begin{bmatrix}
D_2 \\ D_3 \\ D_5 \\ C_1 \\ C_3
\end{bmatrix}}_{\text{Survivors}}
$$

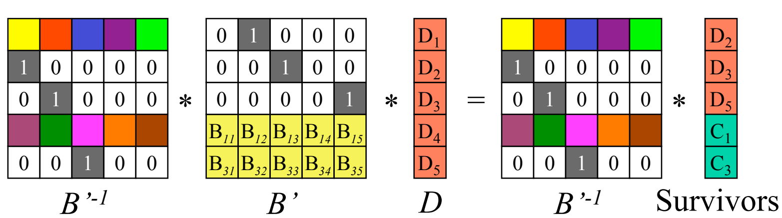
Reed-Solomon Codes

- Now, invert B':



Reed-Solomon Codes

- Now, invert B':
- And multiply both sides of the equation by B'^{-1}



Reed-Solomon Codes

- Now, invert *B'*:
- And multiply both sides of the equation by B'^{-1}
- Since $B'^{-1} * B' = I$, You have just decoded *D*!

$$\underbrace{\begin{bmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{bmatrix}}_{I} * \underbrace{\begin{bmatrix}D_1\\D_2\\D_3\\D_4\\D_5\end{bmatrix}}_{D} = \underbrace{\begin{bmatrix}\ &\ &\ &\ &\ \\1&0&0&0&0\\0&1&0&0&0\\\ &\ &\ &\ &\ \\0&0&1&0&0\end{bmatrix}}_{B'^{-1}} * \underbrace{\begin{bmatrix}D_2\\D_3\\D_5\\C_1\\C_3\end{bmatrix}}_{\text{Survivors}}$$

Reed-Solomon Codes

- Now, invert *B'*:
- And multiply both sides of the equation by B'^{-1}
- Since $B'^{-1} * B' = I$, You have just decoded *D*!

$$\underbrace{\begin{bmatrix}D_1\\D_2\\D_3\\D_4\\D_5\end{bmatrix}}_{D} = \underbrace{\begin{bmatrix}\ &\ &\ &\ &\ \\1&0&0&0&0\\0&1&0&0&0\\\ &\ &\ &\ &\ \\0&0&1&0&0\end{bmatrix}}_{B'^{-1}} * \underbrace{\begin{bmatrix}D_2\\D_3\\D_5\\C_1\\C_3\end{bmatrix}}_{\text{Survivors}}$$

Reed-Solomon Codes

- To Summarize: Encoding
  - Create distribution matrix *B*.
  - Multiply *B* by the data to create coding words.
- To Summarize: Decoding
  - Create *B'* by deleting rows of *B*.
  - Invert *B'*.
  - Multiply B'^{-1} by the surviving words to reconstruct the data.

Reed-Solomon Codes

Two Final Issues:

- #1: How to create *B?*
  - All square submatrices must be invertible.
  - Derive from a Vandermonde Matrix [Plank,Ding:2005].
- #2: Will modular arithmetic work?
  - NO!!!!! (no multiplicative inverses)
  - Instead, you must use *Galois Field* arithmetic.

Reed-Solomon Codes

Galois Field Arithmetic:

- *GF(2^w)* has elements *0, 1, 2, …,* 2^{w-1}.
- Addition = XOR
  - *Easy to implement*
  - *Nice and Fast*
- Multiplication hard to explain
  - If *w* small (≤ 8), use multiplication table.
  - If *w* bigger (≤ 16), use log/anti-log tables.
  - Otherwise, use an iterative process.

Reed-Solomon Codes

Galois Field Example: *GF(2³)*:

- Elements: *0, 1, 2, 3, 4, 5, 6, 7.*
- Addition = XOR:
  - (3 + 2) = 1
  - (5 + 5) = 0
  - (7 + 3) = 4
- Multiplication/Division:
  - Use tables.
  - (3 * 4) = 7
  - (7 ÷ 3) = 4

Multiplication

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	3	1	7	5
3	0	3	6	5	7	4	1	2
4	0	4	3	7	6	2	5	1
5	0	5	1	4	2	7	3	6
6	0	6	7	1	5	3	2	4
7	0	7	5	2	1	6	4	3

Division

	0	1	2	3	4	5	6	7
0	–	–	–	–	–	–	–	–
1	0	1	2	3	4	5	6	7
2	0	5	1	4	2	7	3	6
3	0	6	7	1	5	3	2	4
4	0	7	5	2	1	6	4	3
5	0	2	4	6	3	1	7	5
6	0	3	6	5	7	4	1	2
7	0	4	3	7	6	2	5	1

Reed-Solomon Performance

- Encoding: *O(mn)*
  - More specifically: mS [$(n\text{-}1)/B_{XOR}$ + n/B_{GFMult}]
  - S = Size of a device
  - B_{XOR} = Bandwith of XOR (3 GB/s)
  - B_{GFMult} = Bandwidth of Multiplication over $GF(2^w)$
    - $GF(2^8)$: 800 MB/s
    - $GF(2^{16})$: 150 MB/s

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ B_{11} & B_{12} & B_{13} & B_{14} & B_{15} \\ B_{21} & B_{22} & B_{23} & B_{24} & B_{25} \\ B_{31} & B_{32} & B_{33} & B_{34} & B_{35} \end{bmatrix} * \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \end{bmatrix} = \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}$$

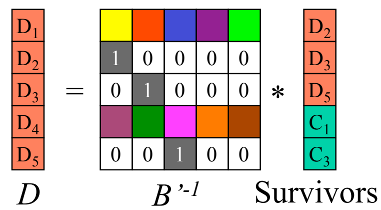
Reed-Solomon Performance

- Update: *O(m)*
  - More specifically: $m+1$ XORs and m multiplications.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ B_{11} & B_{12} & B_{13} & B_{14} & B_{15} \\ B_{21} & B_{22} & B_{23} & B_{24} & B_{25} \\ B_{31} & B_{32} & B_{33} & B_{34} & B_{35} \end{bmatrix} * \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \end{bmatrix} = \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}$$

Reed-Solomon Performance

- Decoding: $O(mn)$ or $O(n^3)$
  - Large devices: dS [$(n\text{-}1)/B_{XOR} + n/B_{GFMult}$]
  - Where d = number of data devices to reconstruct.
  - Yes, there's a matrix to invert, but usually that's in the noise because $dSn >> n^3$.



Reed-Solomon Bottom Line

- Space Efficient: MDS
- Flexible:
  - Works for any value of n and m.
  - Easy to add/subtract coding devices.
  - Public-domain implementations.
- Slow:
  - n-way dot product for each coding device.
  - GF multiplication slows things down.

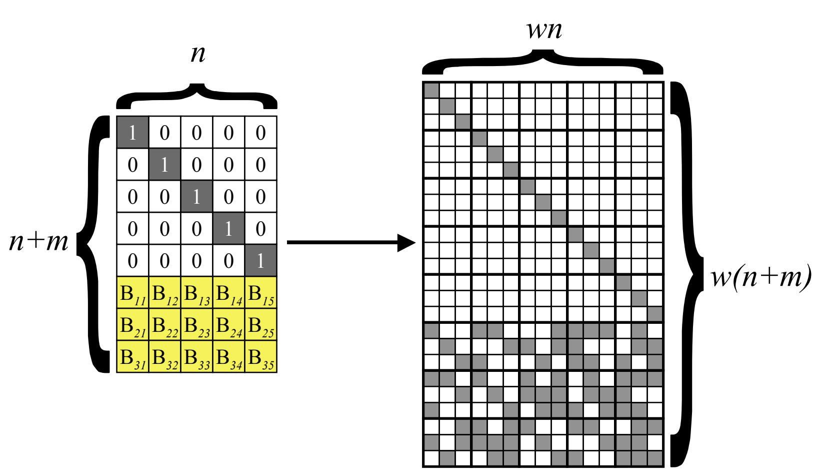
Cauchy Reed-Solomon Codes

[Blomer et al:1995] gave two improvements:

- #1: Use a *Cauchy* matrix instead of a Vandermonde matrix: Invert in $O(n^2)$.
- #2: Use neat projection to convert Galois Field multiplications into XORs.
  - *Kind of subtle, so we'll go over it.*

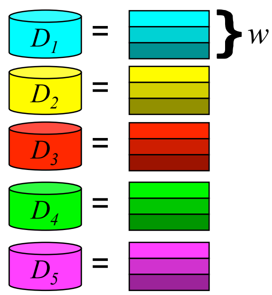
Cauchy Reed-Solomon Codes

- Convert distribution matrix from $(n+m)*n$ over $GF(2^w)$ to $w(n+m)*wn$ matrix of 0's and 1's:



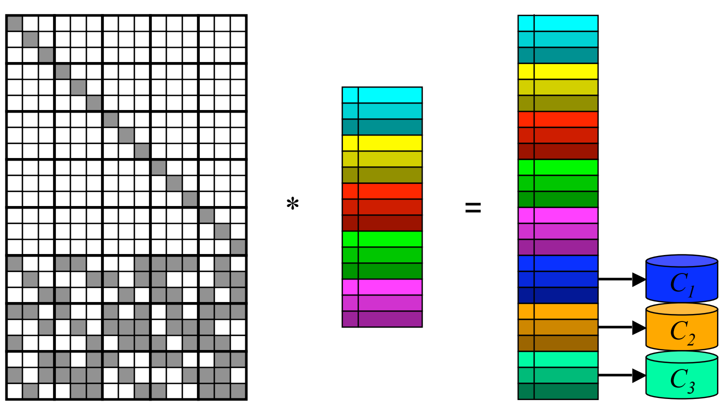
Cauchy Reed-Solomon Codes

- Now split each data device into *w* “packets” of size *S/w*.



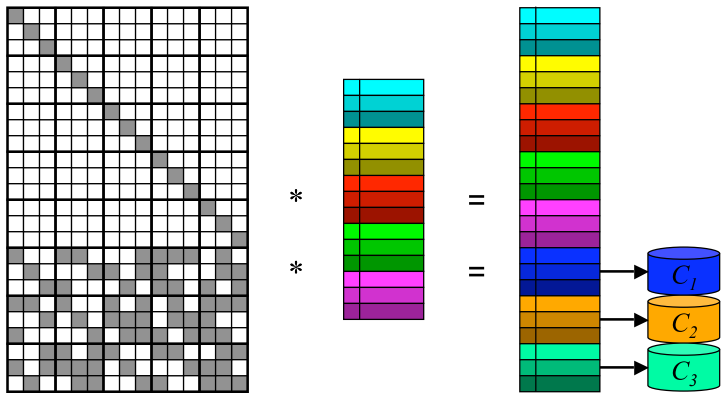
Cauchy Reed-Solomon Codes

- Now the matrix encoding can be performed with XORs of whole packets:



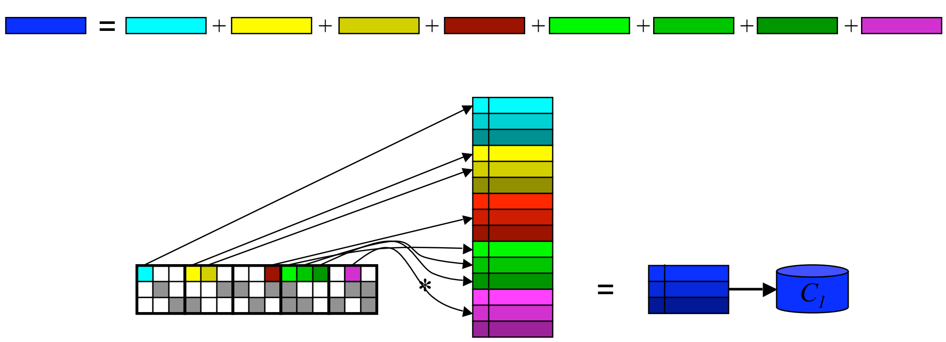
Cauchy Reed-Solomon Codes

- More Detail: Focus solely on C_1.



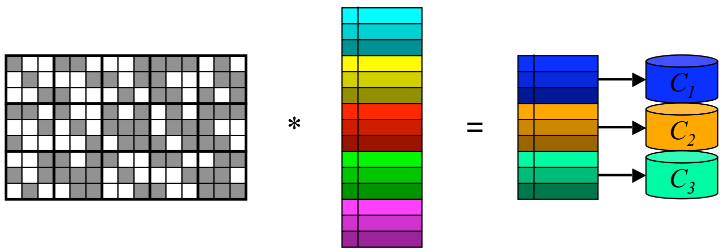
Cauchy Reed-Solomon Codes

- Create a coding packet by XORing data packets with 1's in the proper row & column:



Cauchy Reed-Solomon Performance

- Encoding: $O(wmn)$
  - Specifically: $O(w)*mSn/B_{XOR}$ [Blomer et al:1995]
  - Actually: $mS(o\text{-}1)/B_{XOR}$
  - Where o = average number of 1's per row of the distribution matrix.
- Decoding: Similar: $dS(o\text{-}1)/B_{XOR}$

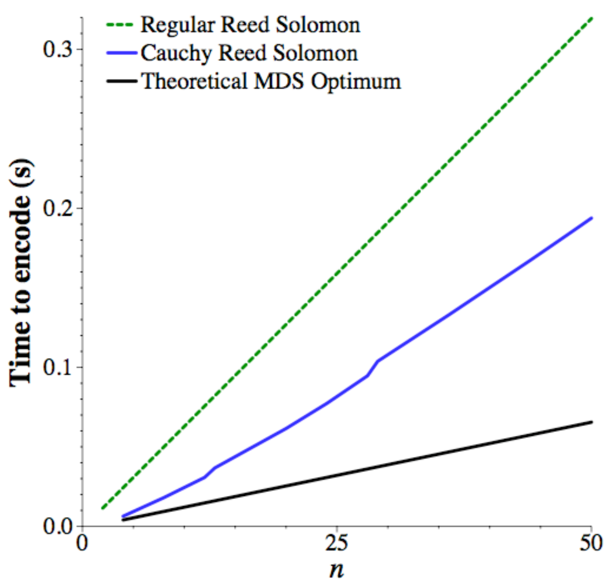


Does it matter?

Encoding time:

- $m = 4$
- $S = 1$ MB
- $B_{XOR} = 3$ GB/s
- $B_{GFMult} = 800$ MB/s
- Cauchy Matrices from [Plank:2005]

We'll discuss more performance later

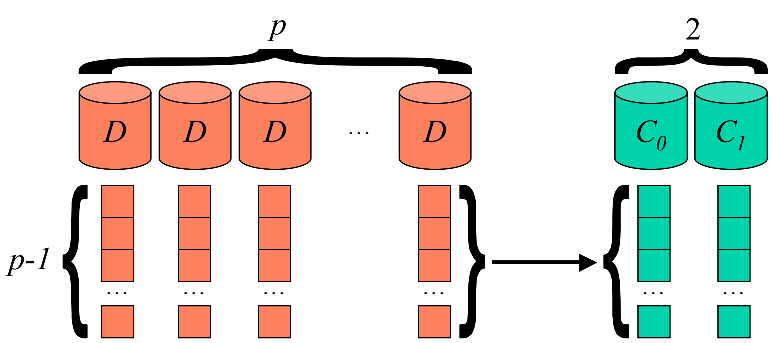


Part 2: Parity Array Codes

- Codes based solely on parity (XOR).
- MDS variants for $m = 2$, $m = 3$.
- Optimal/near optimal performance.
- What I'll show:
  - EVENODD Coding
  - X-Code
  - Extensions for larger m
    - STAR
    - WEAVER
    - HoVer
    - (Blaum-Roth)

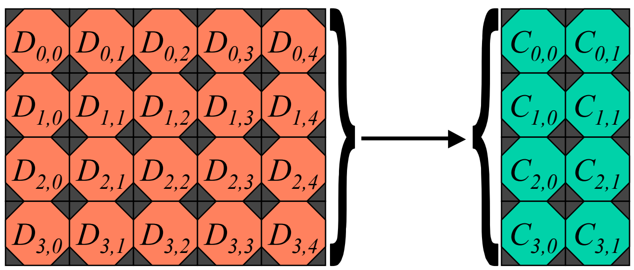
EVENODD Coding

- The "grandfather" of parity array codes.
- [Blaum et al:1995]
- $m = 2$. $n = p$, where p is a prime > 2.
- Partition data, coding devices into blocks of p-1 rows of words:



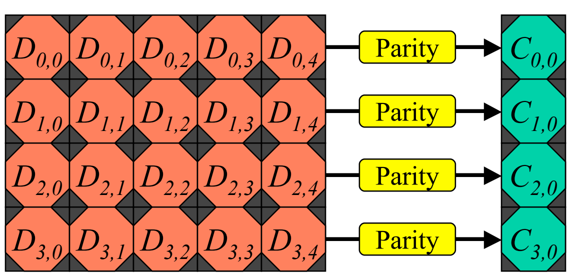
EVENODD Coding

- Logically, a word is a bit.
- In practice, a word is larger.
- Example shown with $n = p = 5$:
  - Each column represents a device.



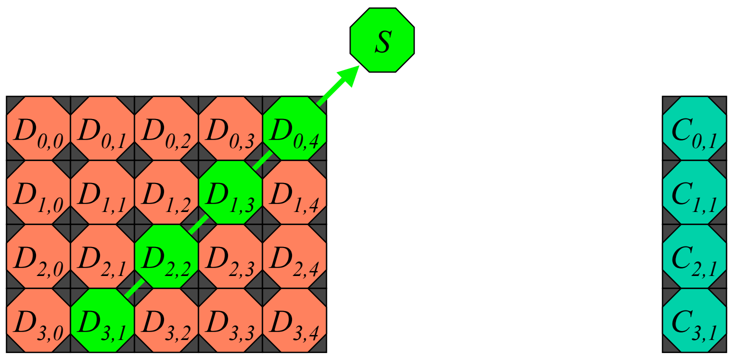
EVENODD Coding

- Column C_0 is straightforward
  - Each word is the parity of the data words in its row:



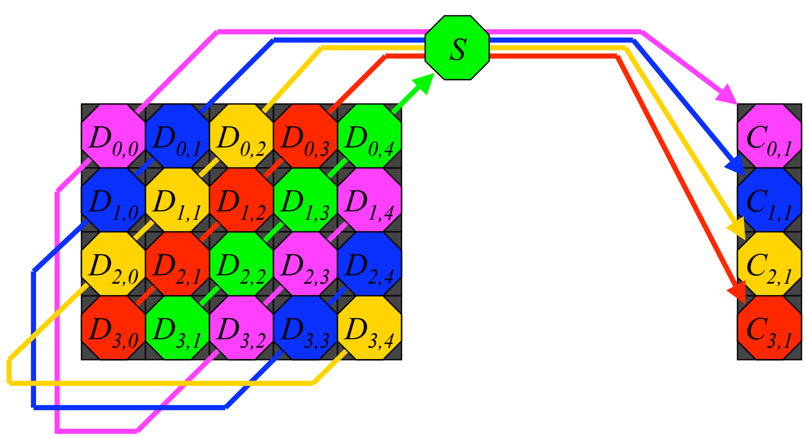
EVENODD Coding

To calculate column C_1, first calculate S (the *"Syndrome"*), which is the parity of one of the diagonals:



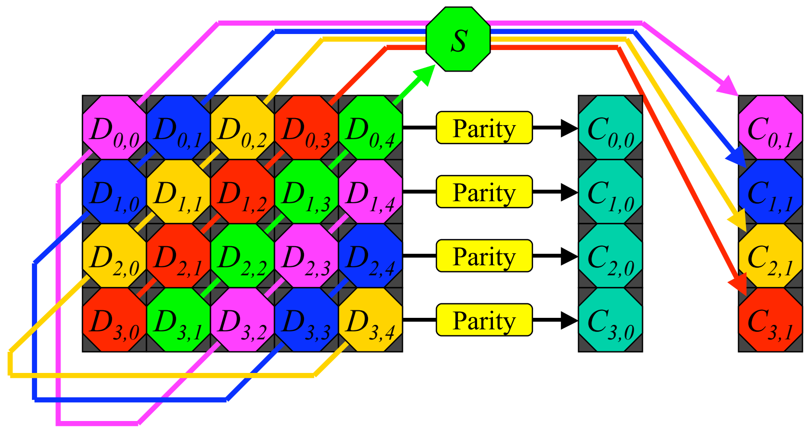
EVENODD Coding

Then, $C_{i,1}$ is the parity of S and all data words on the diagonal containing $D_{i,0}$:



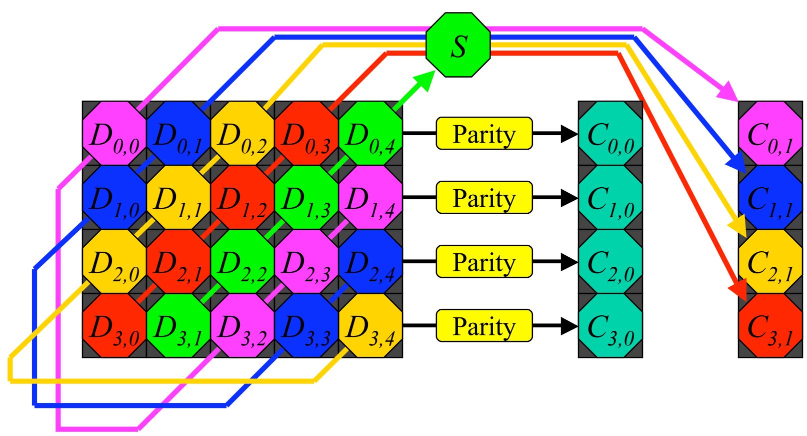
EVENODD Coding

Here's the whole system:



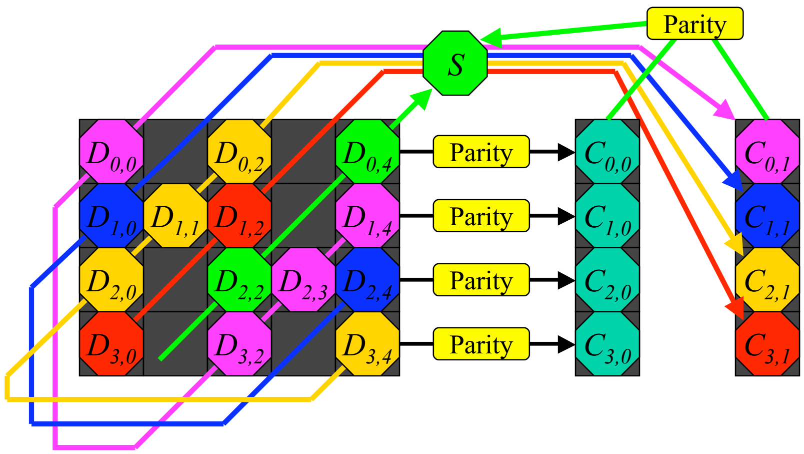
EVENODD Coding

Now, suppose two data devices fail
(This is the hard case).



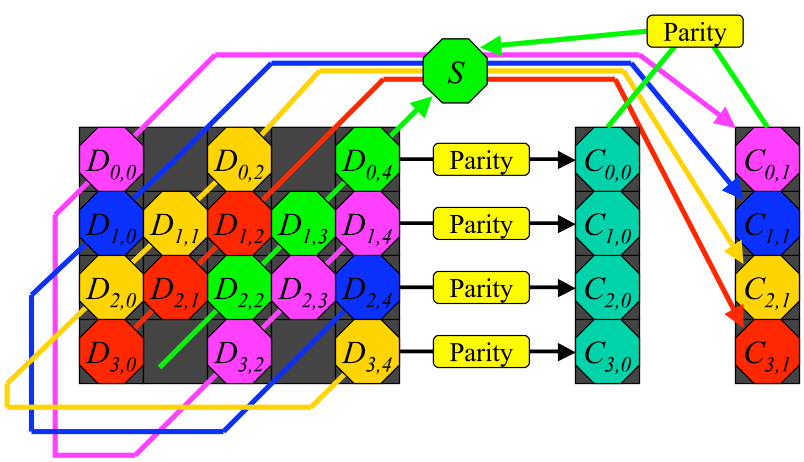
EVENODD Coding

- First, note that S is equal to the parity of all $C_{i,j}$.
- Next, there will be at least one diagonal that is missing just one data word.
- Decode it/them.



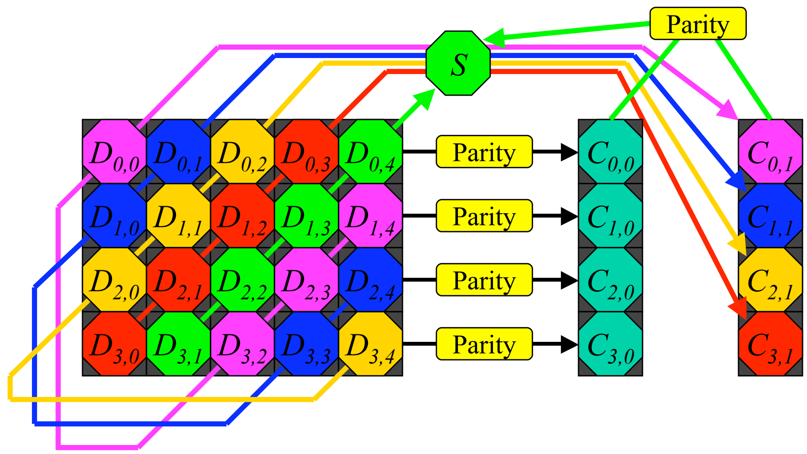
EVENODD Coding

- Next, there will be at least one row missing just one data word:
- Decode it/them.



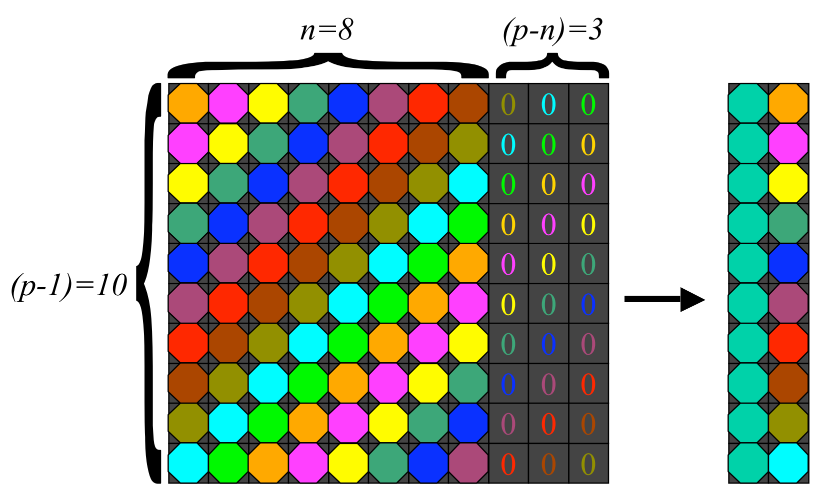
EVENODD Coding

- Continue this process until all the data words are decoded:



EVENODD Coding

If n is not a prime, then find the next prime p, and add p-n "virtual" data devices: - E.g. $n=8$, $p=11$.



EVENODD Performance

- Encoding: $O(n^2)$ XORs per big block.
  - More specifically: $(2n-1)(p-1)$ per block.
  - This means $(n-1/2)$ XORs per coding word.
    - Optimal is $(n-1)$ *XORs* per coding word.
  - Or: $mS\,[n-1/2]/B_{XOR}$, where
    - S = size of a device
    - B_{XOR} = Bandwith of XOR

EVENODD Performance

- Update: *Depends.*
  - If not part of the calculation of S, then 3 XORs (optimal).
  - If part of the calculation of S, then $(p+1)$ XORS (clearly not optimal).

EVENODD Performance

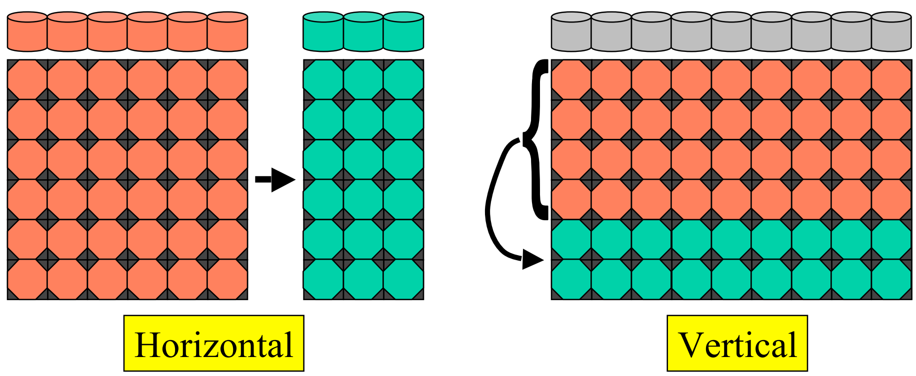
- Decoding:
  - Again, it depends on whether you need to use C_1 to decode. If so, it's more expensive and not optimal.
  - Also, when two data devices fail, decoding is serialized.

EVENODD Bottom Line

- Flexible: works for all values of *n*.
- Excellent encoding performance.
- Poor update performance in $1/(n-1)$ of the cases.
- Mediocre decoding performance.
- Much better than Reed Solomon coding for everything except the pathelogical updates (average case is fine).

Horizontal vs Vertical Codes

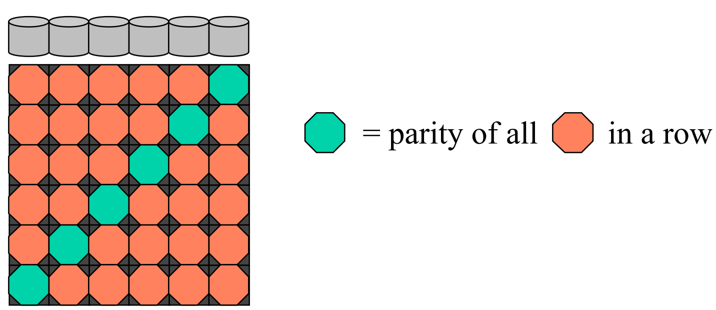
- Horizontal: Devices are all data or all coding.
- Vertical: All devices hold both data and coding.



Horizontal vs Vertical Codes

"Parity Striping"

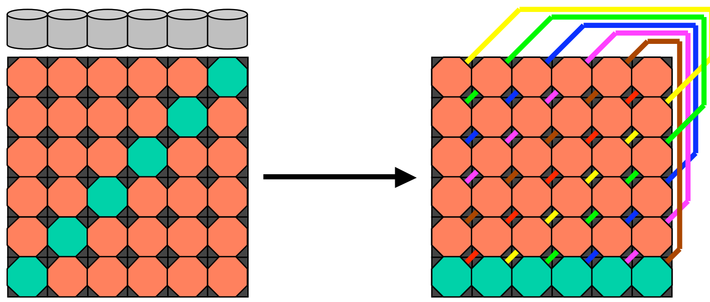
A simple and effective vertical code for $m=1$:



- Good: Optimal coding/decoding.
- Good: Distributes device access on update.
- Bad (?): All device failures result in recovery.

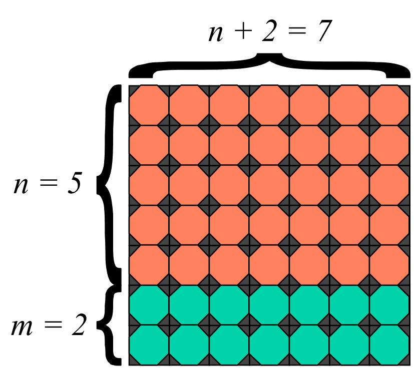
Horizontal vs Vertical Codes

- We can lay out parity striping so that all code words are in the same row:
- (This will help you visualize the X-Code…)



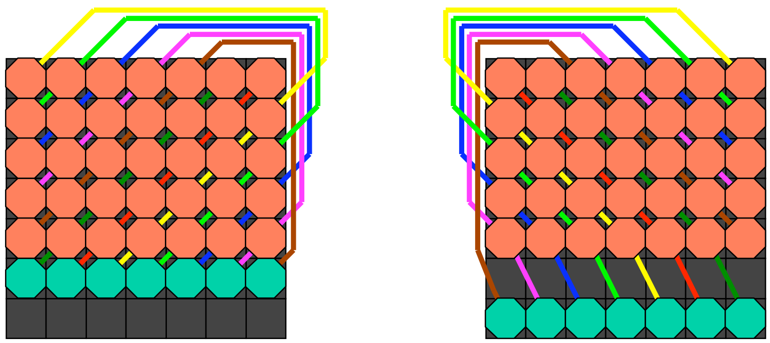
The X-Code

- MDS parity-array code with optimal performance.
- [Xu,Bruck:1999]
- $m = 2$. $n = p-2$, where p is a prime.
  - n rows of data words
  - 2 rows of coding words
  - $n+2$ columns
- For example: $n = 5$:



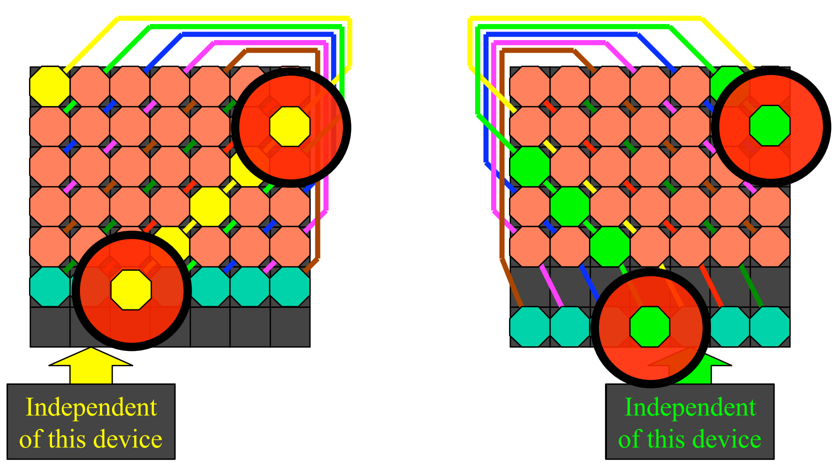
The X-Code

- Each coding row is calculated by parity-striping with opposite-sloped diagonals:



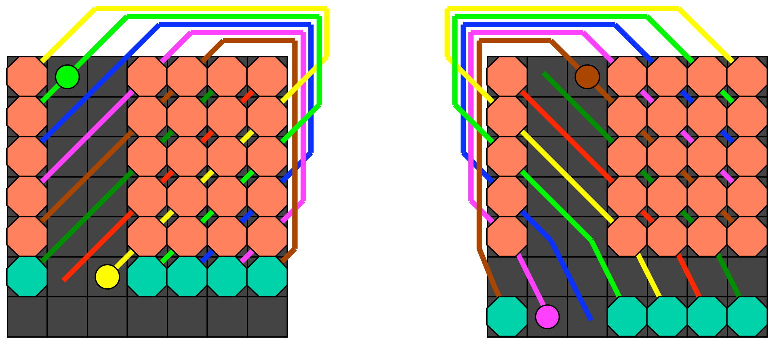
The X-Code

- Each coding word is the parity of n data words.
  - Therefore, each coding word is independent of one data device.
  - And each data word is independent of two data devices:



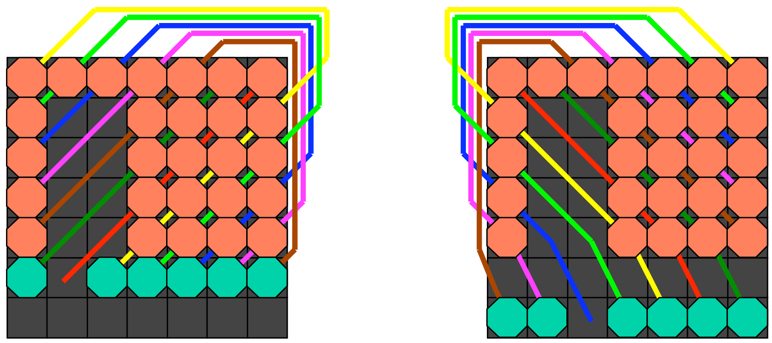
The X-Code

- Suppose we have two failures.
- There will be four words to decode.



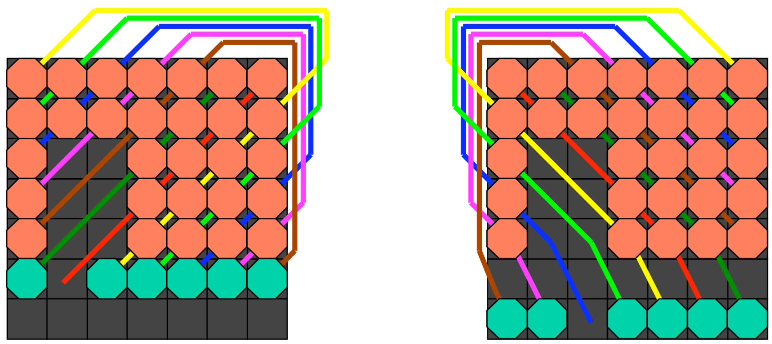
The X-Code

- Suppose we have two failures.
- There will be four words to decode.



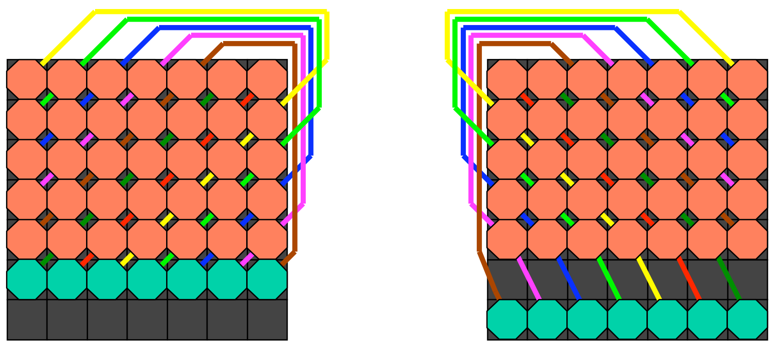
The X-Code

- We can now iterate, decoding two words at every iteration:



The X-Code

- We can now iterate, decoding two words at every iteration:



X-Code Performance

- Encoding: $O(n^2)$ XORs per big block.
  - More specifically: $2(n-1)(n+2)$ per big block.
  - This means $(n-1)$ XORs per coding word.
    - Optimal.
  - Or: $mS\,[n-1]/B_{XOR}$, where
    - S = size of a device
    - B_{XOR} = Bandwith of XOR

X-Code Performance

- Update: *3* XORs - Optimal.
- Decoding: $S\,[n-1]/B_{XOR}$ per failed device.

So this is an excellent code.

Drawbacks:

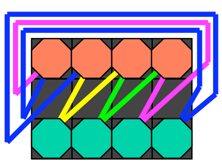
- $n+2$ must be prime.
- (All erasures result in decoding.)

Other Parity-Array Codes

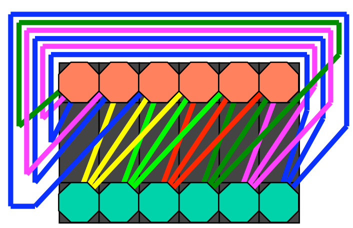
- STAR [Huang,Xu:2005]:
  - Extends EVENODD to $m = 3$.
- WEAVER [Hafner:2005W]:
  - Vertical codes for higher failures.
- HoVer [Hafner:2005H]:
  - Combination of Horizontal/Vertical codes.
- Blaum-Roth [Blaum,Roth:1999]:
  - Theoretical results/codes.

Two WEAVER Codes

m = 2, n = 2:



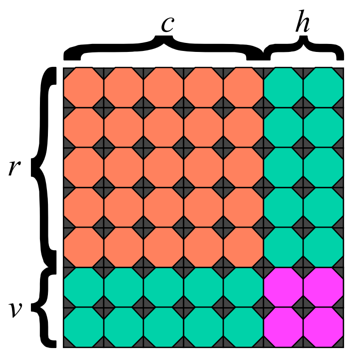
m = 3, n = 3:



- Both codes are MDS.
- Both codes are optimal.
- No X-Code for $n = 2$.
- Other WEAVER codes- up to 12 erasures, but not MDS.

HoVer Codes

- Generalized framework for a blend of horizontal and vertical codes.
- HoVer$^{t}_{v,h}$*[r,c]:*



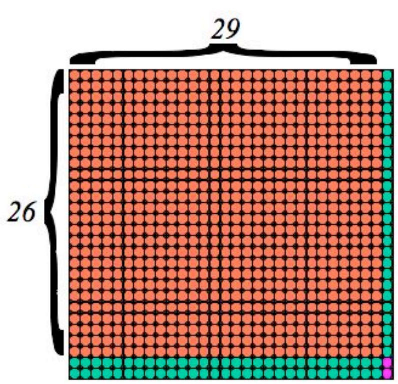
t = fault-tolerance

Not MDS, but interesting nonetheless.

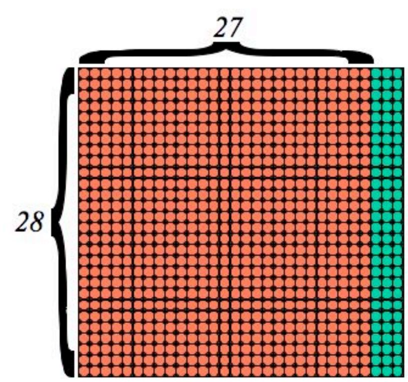
(unused)

HoVer Codes

- For example, there exists: HoVer$^{3}_{2,1}$*[26,29]:*
  - From [Hafner:2005H,Theorem 5, Bullet 6]



HoVer$^{3}_{2,1}$*[26,29]:* Rate .897



MDS Code with same # of devices: Rate .900

Blaum-Roth Codes

- Codes are Minimum Density.
- Optimal encoding and decoding?
- Writing is Maximum Density.
- Will be distilled for the systems programmer someday…

Abstract— **Let $\mathbb{F}_q$ denote the finite field $GF(q)$ and let b be a positive integer. MDS codes over the symbol alphabet $\mathbb{F}_q^b$ are considered that are linear over $\mathbb{F}_q$ and have sparse ("low-density") parity-check and generator matrices over $\mathbb{F}_q$ that are systematic over $\mathbb{F}_q^b$. Lower bounds are presented on the number of nonzero elements in any systematic parity-check or generator matrix of an $\mathbb{F}_q$-linear MDS code over $\mathbb{F}_q^b$, along with upper bounds on the length of any MDS code that attains those lower bounds. A construction is presented that achieves those bounds for certain redundancy values. The building block of the construction is a set of sparse nonsingular matrices over $\mathbb{F}_q$ whose pairwise differences are also nonsingular. Bounds and constructions are presented also for the case where the systematic condition on the parity-check and generator matrices is relaxed to be over $\mathbb{F}_q$, rather than over $\mathbb{F}_q^b$.**

Index Terms— **Disk arrays, group codes, low-density codes, MDS codes, sparse matrices.**

I. INTRODUCTION

LET $\mathbb{F}_q$ denote the finite field $GF(q)$. A code $\mathcal{C}$ over $\mathbb{F}_q^b$ is said to be $\mathbb{F}_q$-*linear* if $\mathcal{C}$ is a vector space over $\mathbb{F}_q$. Such a code is a group code with $\mathbb{F}_q^b$ as the underlying group (see [5], [12], and the references therein). Clearly, every linear code over $GF(q^b)$ is an $\mathbb{F}_q$-linear code over $\mathbb{F}_q^b$. The converse, however, is not true.

Let $\mathcal{C}$ be a code of length n over $\mathbb{F}_q^b$ and minimum Hamming distance d, where the distance is measured with respect to symbols of $\mathbb{F}_q^b$. By the Singleton bound for (not necessarily linear) codes over $\mathbb{F}_q^b$ we have

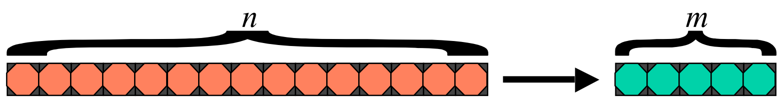
$$d \leq n + 1 - \log_{q^b} |\mathcal{C}|$$

Part 3: LDPC - Low-Density Parity-Check Codes

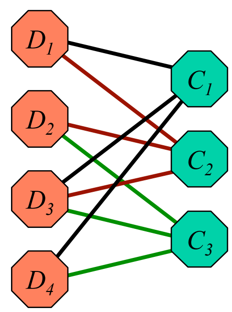
- Codes based solely on parity.
- Distinctly non-MDS.
- Performance far better than optimal MDS.
- Long on theory / short on practice.
- What I'll show:
  - Standard LDPC Framework & Theory
  - Optimal codes for small *m*
  - Codes for fixed rates
  - LT codes

LDPC Codes

- One-row, horizontal codes:



- Codes are defined by *bipartite graphs* - Data words on the left, coding on the right:



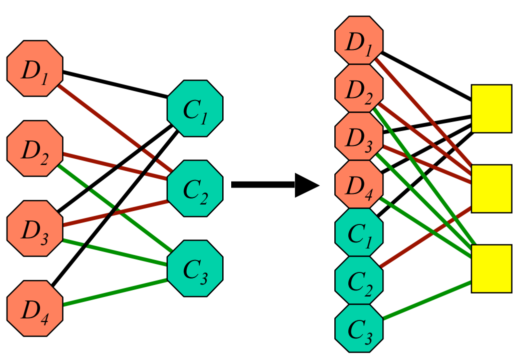
$C_1 = D_1 + D_3 + D_4$

$C_2 = D_1 + D_2 + D_3$

$C_3 = D_2 + D_3 + D_4$

LDPC Codes

- Typical representation is by a *Tanner Graph*
  - Also bipartite.
  - *(n+m)* left-hand nodes: Data + coding
  - *m* right-hand nodes: Equation constraints



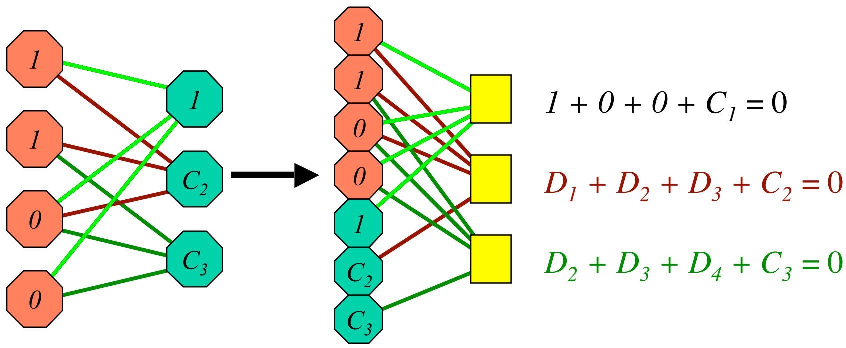
$D_1 + D_3 + D_4 + C_1 = 0$

$D_1 + D_2 + D_3 + C_2 = 0$

$D_2 + D_3 + D_4 + C_3 = 0$

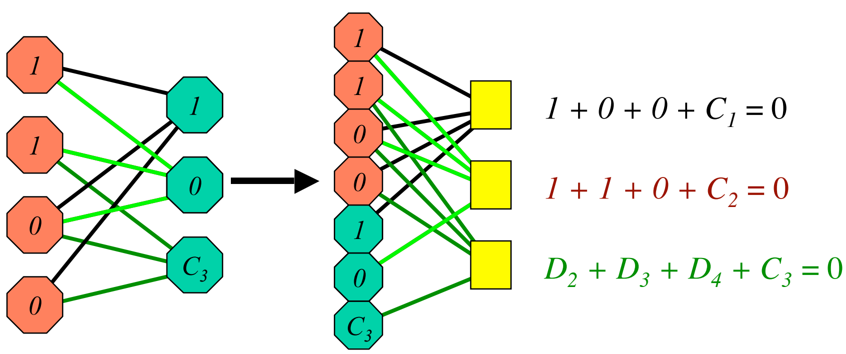
LDPC Codes

- Example coding



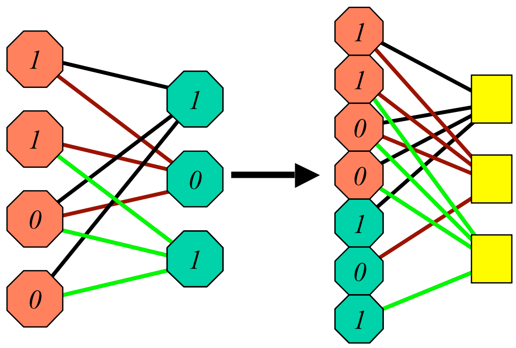
LDPC Codes

- Example coding



LDPC Codes

- Example coding



$1 + 0 + 0 + C_1 = 0$

$1 + 1 + 0 + C_2 = 0$

$1 + 0 + 0 + C_3 = 0$

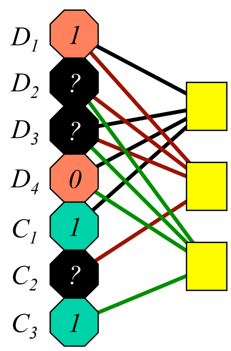
LDPC Codes

- Tanner Graphs:
  - More flexible
  - Allow for straightforward, graph-based decoding.
- Decoding Algorithm:
  - Put 0 in each constraint.
  - For each non-failed node *i*:
    - XOR *i*'s value into each adjacent constraint.
    - Remove that edge from the graph.
    - If a constraint has only one edge, it holds the value of the one node adjacent to it. Decode that node.

LDPC Codes

- Decoding example:

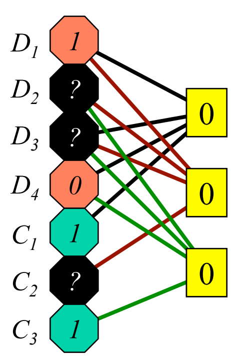
Suppose D_2, D_3 and C_2 fail:



LDPC Codes

- Decoding example:

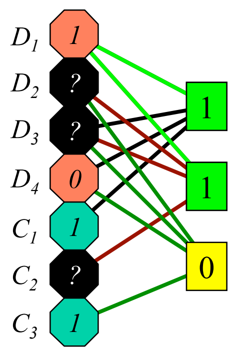
First, put zero into the constraints.



LDPC Codes

- Decoding example:

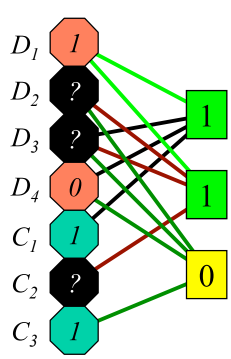
Next, XOR D_1 into its constraints:



LDPC Codes

- Decoding example:

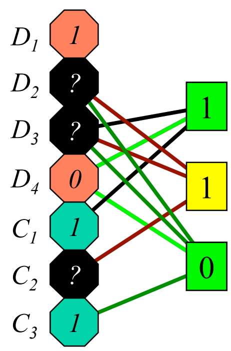
And remove its edges from the graph



LDPC Codes

- Decoding example:

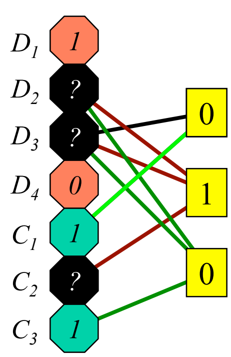
Do the same for D_4:



LDPC Codes

- Decoding example:

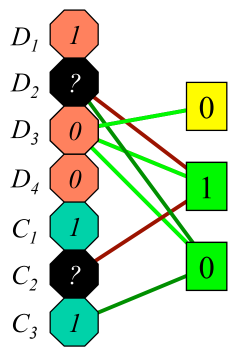
And with C_1



LDPC Codes

- Decoding example:

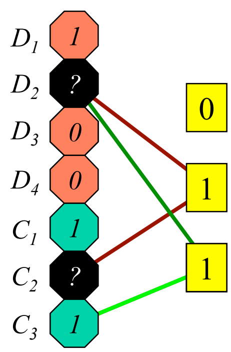
Now, we can decode D_3, and process its edges.



LDPC Codes

- Decoding example:

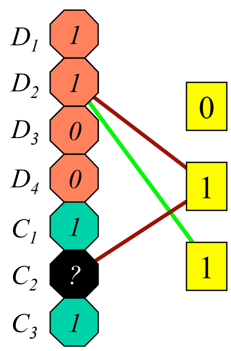
Finally, we process C_3 and finish decoding.



LDPC Codes

- Decoding example:

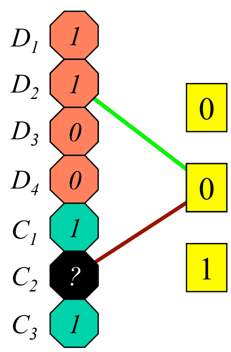
Finally, we process C_3 and finish decoding.



LDPC Codes

- Decoding example:

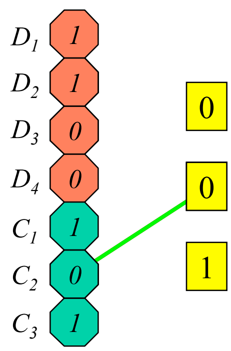
Finally, we process C_3 and finish decoding.



LDPC Codes

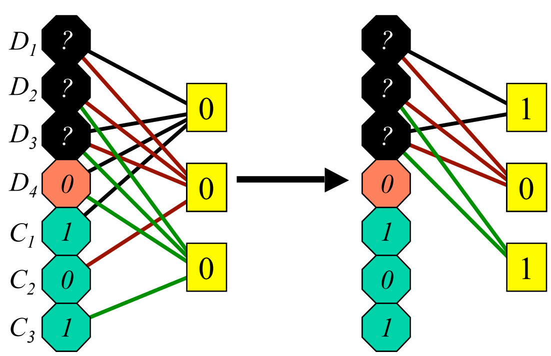
- Decoding example:

We're done!



LDPC Codes

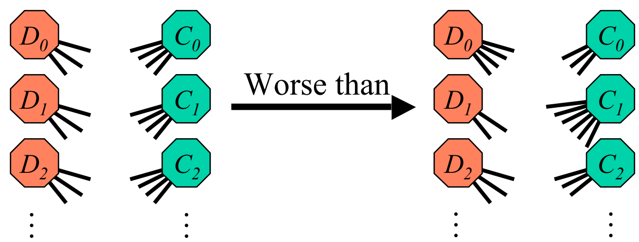
- Encoding:
  - Just decode starting with the data nodes.
- Not MDS:
  - For example: Suppose D_1, D_2 & D_3 fail:



You cannot decode further.

LDPC Codes

- History:
  - Gallager's PhD Thesis (MIT): 1963
  - Landmark paper: Luby *et al:* 1997
    - *Result #1*: Irregular codes perform better than regular codes (in terms of space, not time).



LDPC Codes

- History:
  - Gallager's PhD Thesis (MIT): 1963
  - Landmark paper: Luby *et al:* 1997
    - *Result #2*: Defined LDPC codes that are:

Asymptotically MDS!

LDPC Codes: Asymptotically MDS

- Recall:
  - The rate of a code: $R = n/(n+m)$.
  - The *overhead factor* of a code: f = factor from MDS:
    - $f = m$/(average nodes required to decode).
    - $f \geq 1$.
    - If $f = 1$, the code is MDS.
- You are given R.

LDPC Codes: Asymptotically MDS

- Define:
  - *Probability distributions* λ and ρ for cardinality of left-hand and right-hand nodes.

- Prove that:
  - As $n \to \infty$, and m defined by R,
  - If you construct random graphs where node cardinalities adhere to λ and ρ,
  - Then $f \to 1$.

LDPC Codes: Asymptotically MDS

- Let's reflect on the significance of this:
  - Encoding and decoding performance is *O(1)* per coding node ("Low Density").
  - Update performance is *O(1)* per updated device.
  - Yet the codes are asymptotically MDS.
  - Wow. Spurred a flurry of similar research.
  - Also spurred a startup company, "Digital Fountain," which applied for and received a flurry of patents.

LDPC Codes: Asymptotically MDS

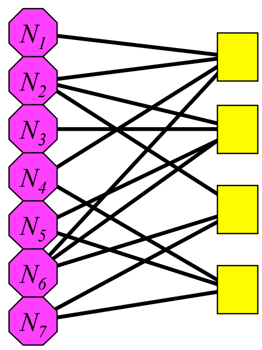
- However:
  - You can prove that:
    - While f does indeed approach 1 as $n \rightarrow \infty$,
    - f is always strictly > 1.
  - Moreover, my life is not asymptotic!
    - Question 1: How do I construct codes for finite n?
    - Question 2: How will they perform?
    - Question 3: Will I get sued?
  - As of 2003:

No one had even attempted to answer these questions!!

LDPC Codes: Small *m*

- [Plank *et al*:2005]
- #1: Simple problem:
  - Given a Tanner Graph, is it *systematic*?
  - I.e: Can n of the left-hand nodes hold the data?

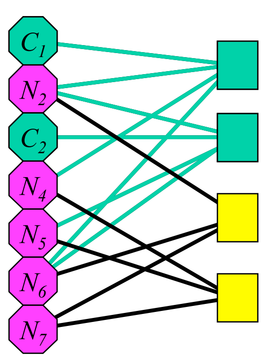
Is this a systematic code for $n=3,\ m=4$?



LDPC Codes: Small *m*

- Simple algorithm:
  - Find up to m nodes N_i with one edge, each to different constraints.
  - Label them coding nodes.
  - Remove them, their edges, and all edges to their constraints.
  - Repeat until you have m coding nodes.

Is this a systematic code for $n=3,\ m=4$?

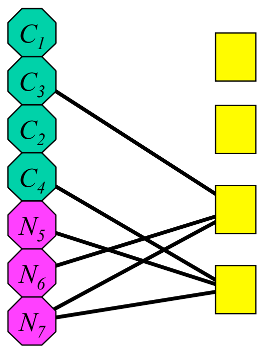


Start with $N_{1,}$ and N_3:

LDPC Codes: Small *m*

- Simple algorithm:
  - Find up to m nodes N_i with one edge, each to different constraints.
  - Label them coding nodes.
  - Remove them, their edges, and all edges to their constraints.
  - Repeat until you have m coding nodes.

Is this a systematic code for $n=3$, $m=4$?



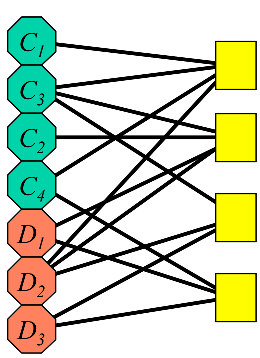
N_2, and N_4 are the final coding nodes.

LDPC Codes: Small *m*

- Simple algorithm:
  - Find up to m nodes N_i with one edge, each to different constraints.
  - Label them coding nodes.
  - Remove them, their edges, and all edges to their constraints.
  - Repeat until you have m coding nodes.

Is this a systematic code for $n=3$, $m=4$?

Yes!

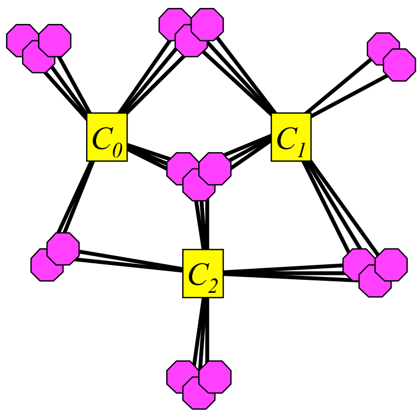


N_2, and N_4 are the final coding nodes.

LDPC Codes: Small *m*

- #2: Define graphs by partitioning nodes into *Edge Classes*:

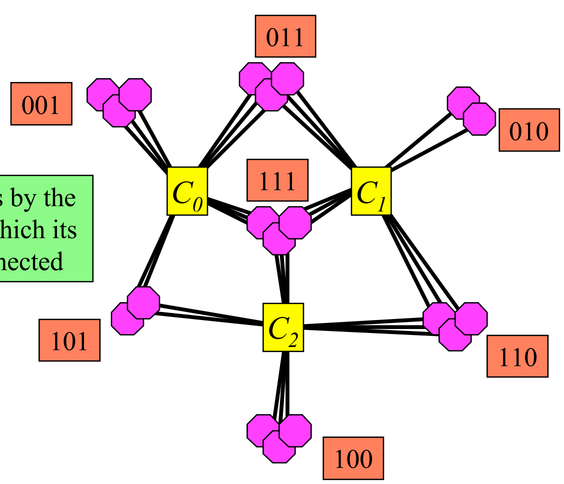
E.g. $m = 3$



LDPC Codes: Small *m*

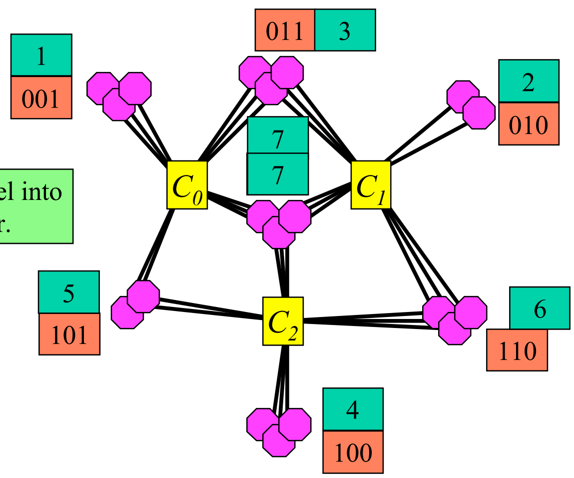
- #2: Define graphs by partitioning nodes into *Edge Classes*:

Label each class by the constraints to which its nodes are connected



LDPC Codes: Small *m*

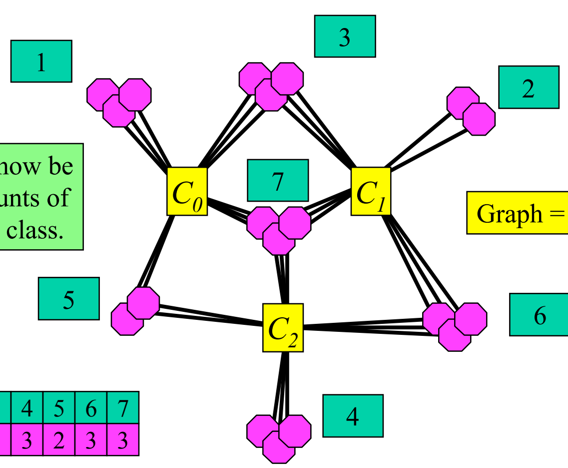
- #2: Define graphs by partitioning nodes into *Edge Classes*:



Turn each label into an integer.

LDPC Codes: Small *m*

- #2: Define graphs by partitioning nodes into *Edge Classes*:



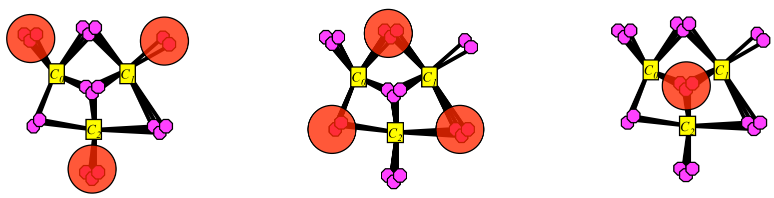
Graph is may now be defined by counts of nodes in each class.

Graph = <3,2,3,3,2,3,3>

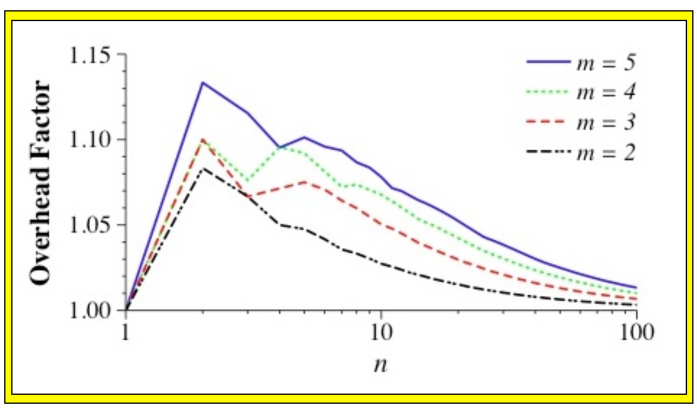
Class	1	2	3	4	5	6	7
Count	3	2	3	3	2	3	3

LDPC Codes: Small *m*

- Best graphs for $m \in [2:5]$ and $n \in [1:1000]$ in [Plank:2005].
- Features:
  - Not balanced. E.g. $m=3$, $n=50$ is <9,9,7,9,7,7,5>.
  - Not loosely left-regular
    - LH nodes' cardinalities differ by more than one.
  - Loosely right-regular
    - RH nodes' (constraints) cardinalities differ at most by one.
  - Loose Edge Class Equivalence
    - Counts of classes with same cardinality differ at most by one.

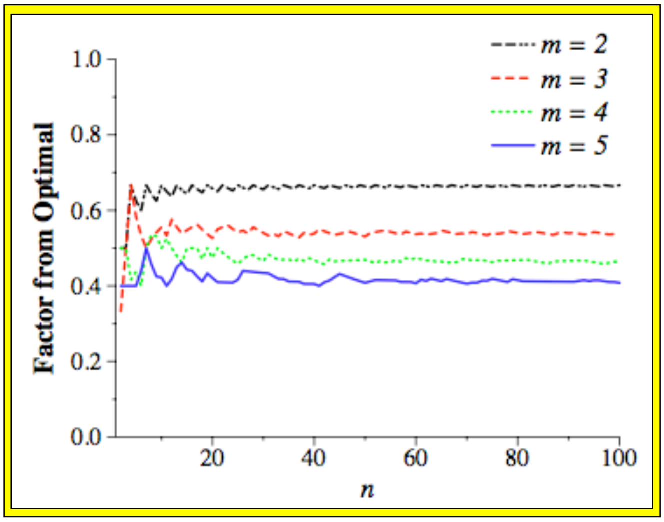


LDPC Codes: Small *m*



- f does *not* decrease monotonically with n.
- $f \rightarrow 1$ as $n \rightarrow \infty$
- f is pretty small (under 1.10 for $n \geq 10$).

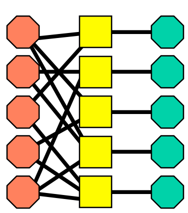
LDPC Codes: Small *m*



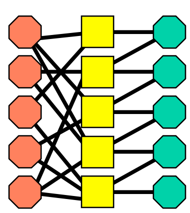
Encoding Performance: 40 - 60 % Better than optimal.

LDPC Codes: Larger *m*

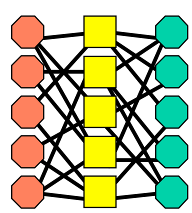
- [Plank,Thomason:2004]
- A lot of voodoo - Huge Monte Carlo simulations.
- Use 80 published values of λ and ρ, test R = 1/3, 1/2, 2/3.
- Three type of code constructions:



Simple Systematic

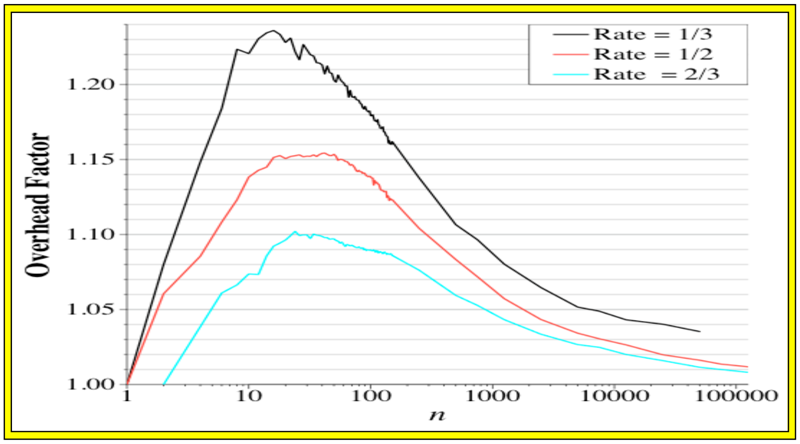


IRA:
Irregular
Repeat-Accumulate



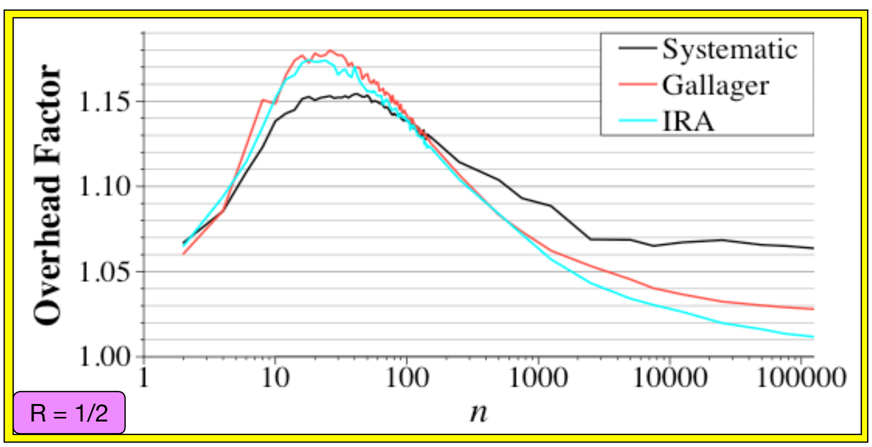
Gallager
Unsystematic

LDPC Codes: Larger *m*



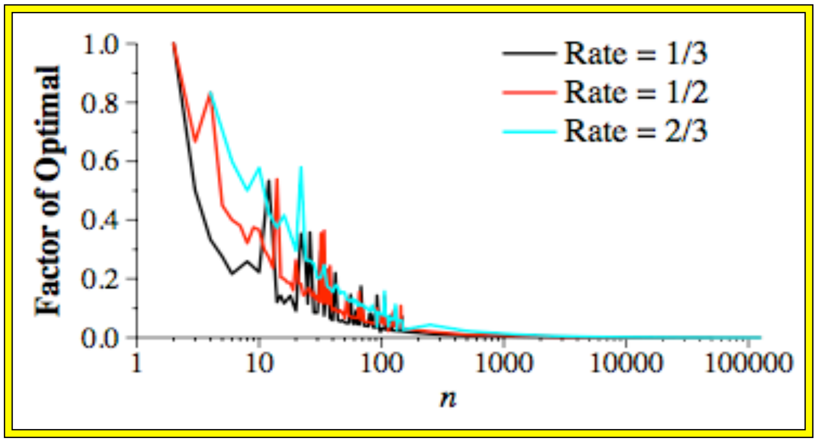
- Lower rates have higher *f*.
- $f \to 1$ as $n \to \infty$
- *f* at their worst in the useful ranges for storage applications.

LDPC Codes: Larger *m*



- Simple systematic perform better for smaller *n*.
- IRA perform better for larger *n*.
- (Not in the graph - Theoretical λ and ρ didn't match performance),

LDPC Codes: Larger *m*



- Improvement over optimal MDS coding is drastic indeed.

LDPC Codes: LT Codes

- Luby-Transform Codes: [Luby:2002]
- Rateless LDPC codes for large n,m.
- Uses an implicit graph, created on-the-fly:
  - When you want to create a coding word, you randomly select a weight w. This is the cardinality of the coding node.
  - w's probability distribution comes from a "weight table."
  - Then you select w data words at random (uniform distribution), and XOR them to create the coding word.
  - As before, theory shows that the codes are asymptotically MDS.
  - [Uyeda et al:2004] observed $f \approx 1.4$ for $n = 1024$, $m = 5120$.
- Raptor Codes [Shokrollahi:2003] improve upon LT-Codes.

LDPC Codes: Bottom Line

- For large *n, m* - Essential alternatives to MDS codes.
- For smaller *n, m* - Important alternatives to MDS codes:
  - Improvement is not so drastic.
  - Tradeoffs in space / failure resilience must be assessed.

LDPC Codes: Bottom Line

- "Optimal" codes are only known in limited cases.
  - Finite theory much harder than asymptotics.
  - "Good" codes should still suffice.
- Patent issues cloud the landscape.
  - Tornado codes (specific λ and ρ) patented.
  - Same with LT codes.
  - And Raptor codes.
  - Scope of patents has not been defined well.
  - Few published codes.
- Need more research!

Part 4: Evaluating Codes

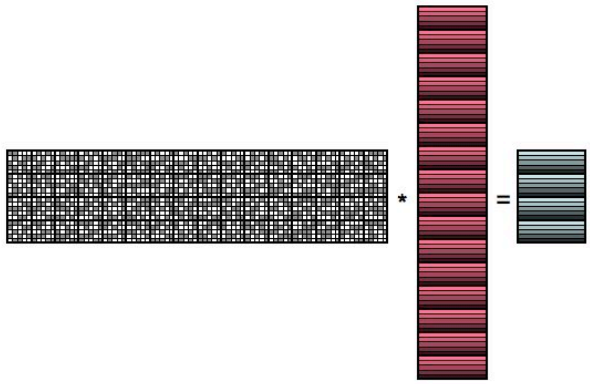
- Defining "fault-tolerance"
- Encoding - impact of the system
- Decoding - impact of the system
- Related work

Defining "fault-tolerance"

- Historical metrics:
  - E.g: "Safe to *x* failures"
  - E.g: "99.44% pure"
  - Makes it hard to evaluate/compare codes.
- Case study:
  - Suppose you have 20 storage devices.
  - 1 GB each.
  - You want to be resilient to 4 failures.

Defining “fault-tolerance”

- 20 storage devices (1GB) resilient to 4 failures:
- Solution #1: The only MDS alternative:
  Reed-Solomon Coding:

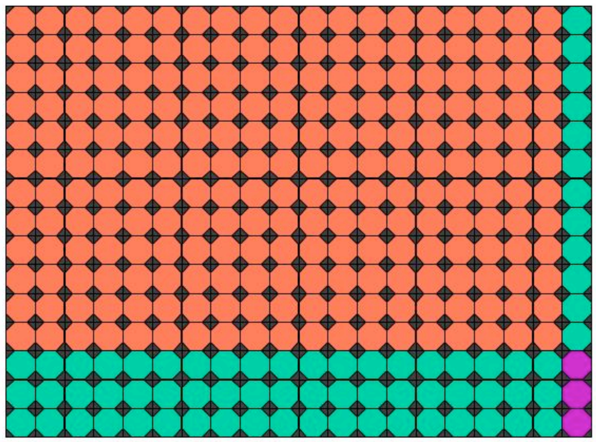


Defining “fault-tolerance”

- 20 storage devices (1GB) resilient to 4 failures:
- Solution #1: The only MDS alternative:
  Reed-Solomon Coding:
  - 80% of storage contains data.
  - Cauchy Matrix for *w*=5 has 912 ones.
  - 44.6 XORs per coding word.
  - Encoding: 59.5 seconds.
  - Decoding: roughly 14.9 seconds per failed device.
  - Updates: 12.4 XORs per updated node.

Defining “fault-tolerance”

- 20 storage devices (1GB) resilient to 4 failures :
- Solution #2: $\text{HoVer}^4_{3,1}[12,19]$:

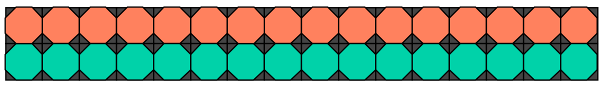


Defining “fault-tolerance”

- 20 storage devices (1GB) resilient to 4 failures :
- Solution #2: $\text{HoVer}^4_{3,1}[12,19]$:
  - 228 data words, 69 coding words (3 wasted).
  - 76% of storage contains data.
  - Encoding: (12*18 + 3*19*11)/69 = 12.22 XORs per coding word: 18.73 seconds.
  - Decoding: Roughly 5 seconds per device.
  - Update: 5 XORs

Defining “fault-tolerance”

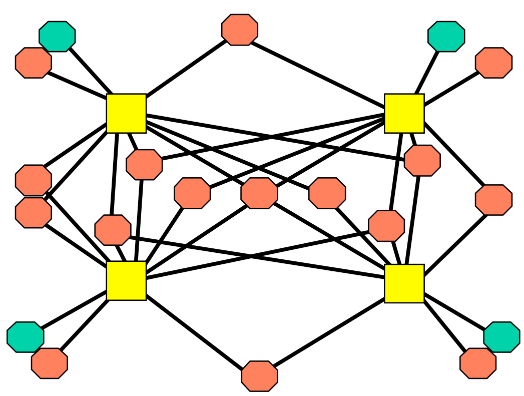
- 20 storage devices (1GB) resilient to 4 failures:
- Solution #3: 50% Efficiency WEAVER code



  - 50% of storage contains data.
  - Encoding: 3 XORs per coding word: 10 seconds.
  - Decoding: Roughly 1 second per device.
  - Update: 5 XORs

Defining “fault-tolerance”

- 20 storage devices (1GB) resilient to 4 failures:
- Solution #4: LDPC <2,2,2,2,1,1,1,2,1,1,1,1,1,1,1>

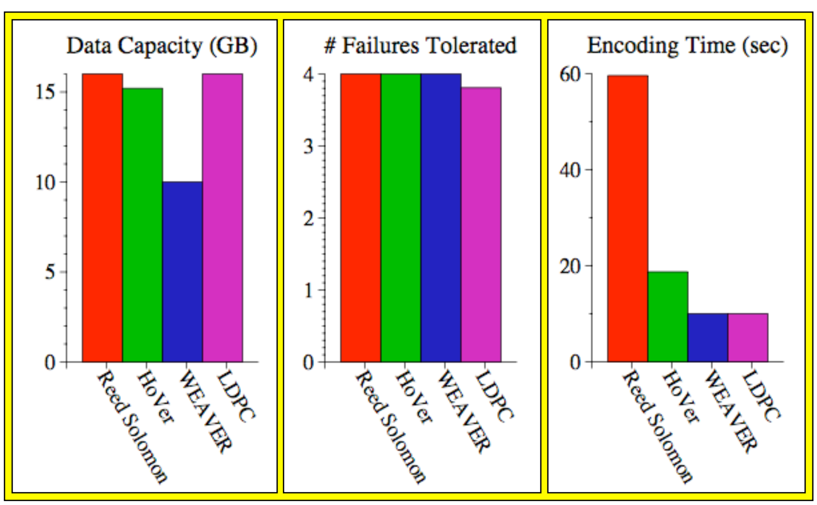


Defining “fault-tolerance”

- 20 storage devices (1GB) resilient to 4 failures:
- Solution #4: LDPC <2,2,2,2,1,1,1,2,1,1,1,1,1,1,1>
  - 80% of storage for data
  - *f* = 1.0496 (Resilient to 3.81 failures…)
  - Graph has 38 edges: 30 XORs per 4 coding words.
  - Encoding: 10 seconds.
  - Decoding: Roughly 3 seconds per device.
  - Update: 3.53 XORs

Defining “fault-tolerance”

- 20 storage devices (1GB) resilient to 4 failures:



Encoding Considerations

- Decentralized Encoding:
  - Not reasonable to have one node do all encoding.
  - E.g. Network Coding [Ahlswede et al:2000].
  - Reed-Solomon codes work well, albeit with standard performance.
  - Randomized constructions [Gkantsidis,Rodriguez:2005].

Decoding Considerations

- Scheduling - Content Distribution Systems:
  - All blocks are not equal - data vs. coding vs. proximity: [Collins,Plank:2005].
  - LDPC: All blocks are not equal #2 - don't download a block that you've already decoded [Uyeda *et al*:2004].
  - Simultaneous downloads & aggressive failover [Collins,Plank:2004].

Resources (Citations)

- Reed Solomon Codes:
  - [Plank:1997] J. S. Plank, “A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems,” *Software -- Practice & Experience*, 27(9), September, 1997, pp. 995-1012. http://www.cs.utk.edu/~plank/plank/papers/papers.html.
  - [Rizzo:1997] L. Rizzo, “Effective erasure codes for reliable computer communication protocols,” *ACM SIGCOMM Computer Communication Review*, 27(2), 1997, pp. 24-36. http://doi.acm.org/10.1145/263876.263881.
  - [Plank,Ding:2005] J. S. Plank, Y. Ding, “Note: Correction to the 1997 Tutorial on Reed-Solomon Coding,” *Software -- Practice & Experience*, 35(2), February, 2005, pp. 189-194. http://www.cs.utk.edu/~plank/plank/papers/papers.html. *(Includes software)*

Resources (Citations)

- Reed Solomon Codes:
  - [Blomer et al:1995] J. Blomer, M. Kalfane, M. Karpinski, R. Karp, M. Luby and D. Zuckerman, “An XOR-Based Erasure-Resilient Coding Scheme,” Technical Report TR-95-048, International Computer Science Institute, August, 1995. http://www.icsi.berkeley.edu/~luby/. *(Includes software)*
  - [Plank:2005] J. Plank “Optimizing Cauchy Reed-Solomon Codes for Fault-Tolerant Storage Applications,” Submitted for publication, http://www.cs.utk.edu/~plank/plank/papers/papers.html. *(Includes good Cauchy Matrices)*

Resources (Citations)

- Parity Array Codes:
  - [Blaum et al:1995] M. Blaum, J. Brady, J. Bruck and J. Menon, EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures, *IEEE Transactions on Computing*, 44(2), February, 1995, pp. 192-202.
  - [Blaum,Roth:1999] M. Blaum and R. M. Roth "On Lowest Density MDS Codes," *IEEE Transactions on Information Theory*, 45(1), January, 1999, pp. 46-59.
  - [Xu,Bruck:1999] L. Xu and J. Bruck, X-Code: MDS Array Codes with Optimal Encoding, *IEEE Transactions on Information Theory*, 45(1), January, 1999, pp. 272-276.

Resources (Citations)

- Parity Array Codes:
  - [Huang,Xu:2005] C. Huang and L. Xu, "STAR: An Efficient Coding Scheme for Correcting Triple Storage Node Failures," *FAST-2005: 4th Usenix Conference on File and Storage Technologies*, December, 2005, http://www.usenix.org/events/fast05.
  - [Hafner:2005H] J. L. Hafner, "HoVer Erasure Codes for Disk Arrays," Research Report RJ10352 (A0507-015), IBM Research Division, July, 2005, http://domino.research.ibm.com/library.
  - [Hafner:2005W] J. L. Hafner, "WEAVER Codes: Highly Fault Tolerant Erasure Codes for Storage Systems," *FAST-2005: 4th Usenix Conference on File and Storage Technologies*, December, 2005, http://www.usenix.org/events/fast05.

Resources (Citations)

- LDPC Codes:
  - [Gallager:1963] R. G. Gallager, Low-Density Parity-Check Codes, MIT Press, Cambridge, MA, 1963.
  - [Wicker,Kim:2005] S. B. Wicker and S. Kim, Fundamentals of Codes, Graphs, and Iterative Decoding, Kluwer Academic Publishers, Norwell, MA, 2003.
  - [Luby *et al*:1997] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman and V. Stemann, "Practical Loss-Resilient Codes," *29th Annual ACM Symposium on Theory of Computing*, El Paso, TX, 1997, pages 150-159, http://www.icsi.berkeley.edu/~luby/.
  - [Plank *et al*:2005] J. S. Plank, A. L. Buchsbaum, R. L. Collins and M. G. Thomason, "Small Parity-Check Erasure Codes - Exploration and Observations," *DSN-2005: International Conference on Dependable Systems and Networks,* Yokohama, Japan, IEEE, 2005, http://www.cs.utk.edu/~plank/plank/papers/papers.html.
    (Includes enumeration of best codes for m = 2-5, n = 2-1000)

Resources (Citations)

- LDPC Codes:
  - [Plank,Thomason:2004] J. S. Plank and M. G. Thomason, "A Practical Analysis of Low-Density Parity-Check Erasure Codes for Wide-Area Storage Applications," *DSN-2004: The International Conference on Dependable Systems and Networks*, IEEE, Florence, Italy, June, 2004, pp. 115-124, http://www.cs.utk.edu/~plank/plank/papers/papers.html.
  - [Collins,Plank:2005] R. L. Collins and J.. S. Plank, "Assessing the Performance of Erasure Codes in the Wide-Area," *DSN-2005: International Conference on Dependable Systems and Networks*, Yokohama, Japan, June, 2005, http://www.cs.utk.edu/~plank/plank/papers/papers.html.
  - [Luby:2002] M. Luby, LT Codes, IEEE Symposium on Foundations of Computer Science, 2002, http://www.digitalfountain.com.

Resources (Citations)

- LDPC Codes:

  - [Mitzenmacher:2004] M. Mitzenmacher, Digital Fountains: A Survey and Look Forward, *IEEE Information Theory Workshop*, San Antonio, October, 2004, http://wcl3.tamu.edu/itw2004/program.html.
  - [Shokrollahi:2003] A. Shokrollahi, "Raptor Codes," Digital Fountain Technical Report DR2003-06-001, 2003, http://www.digitalfountain.com/technology/researchLibrary/.
  - [Uyeda *et al*:2004] F. Uyeda, H. Xia and A. Chien, "Evaluation of a High Performance Erasure Code Implementation," University of California, San Diego Technical Report CS2004-0798, 2004, http://www.cse.ucsd.edu.