Coding and computation in distributed storage for dynamic networks

Muriel Médard

Department of EECS

MIT.

Collaborators

- MIT: Supratim Deb (now AT&T research), Tracey Ho (now Speedy Packets), Ben Leong (now National University of Singapore), Prakash Narayana Moorthy, Weifei Zeng
- University of Illinois Urbana-Champaign: Ralf Koetter (later Technical University of Munich)
- University of Aalborg: Frank Fitzek (now Technical University Dresden), Daniel E. Lucani
- Budapest University of Technology and Economics: Hassan Charaf, Marton Sipos, Aron Szabados, Thomas Toth
- Steinwurf: Janus Heide, , Morten Pedersen, Peter Vingelmann
- University of Warsaw: Szymon Acedanski.

Overview

- Random linear network coding
- Distributed storage random linear network coding
- Coding in dynamic systems
- Coding for updating functions

Random Linear Network Coding (RLNC)

Algebraic equations more efficiently input data into IP packets

An IP packet payload

0011010010100001

= Vector of elements of a finite field

4 packets

P_1

P_2

P_3

P_4

ENCODE
(using algebra)

Random linear combinations are highly likely to be recoverable

$P_1 + 7P_2 + 4P_3 + 3P_4$

$2P_1 + 8P_2 + 9P_3 + 5P_2$

$8P_1 + 6P_2 + P_3 + 2P_4$

$7P_1 + 2P_2 + 3P_3 + 4P_4$

packets are more versatile

T. Ho, Médard, M., Koetter, R., Karger, D.R., Effros, M., Shi, J., and Leong, B., “A Random Linear Network Coding Approach to Multicast,” IEEE Transactions on Information Theory, Vol. 52, Issue 10, pp. 4413-4430, October 2006

Coding Algorithm Evolution

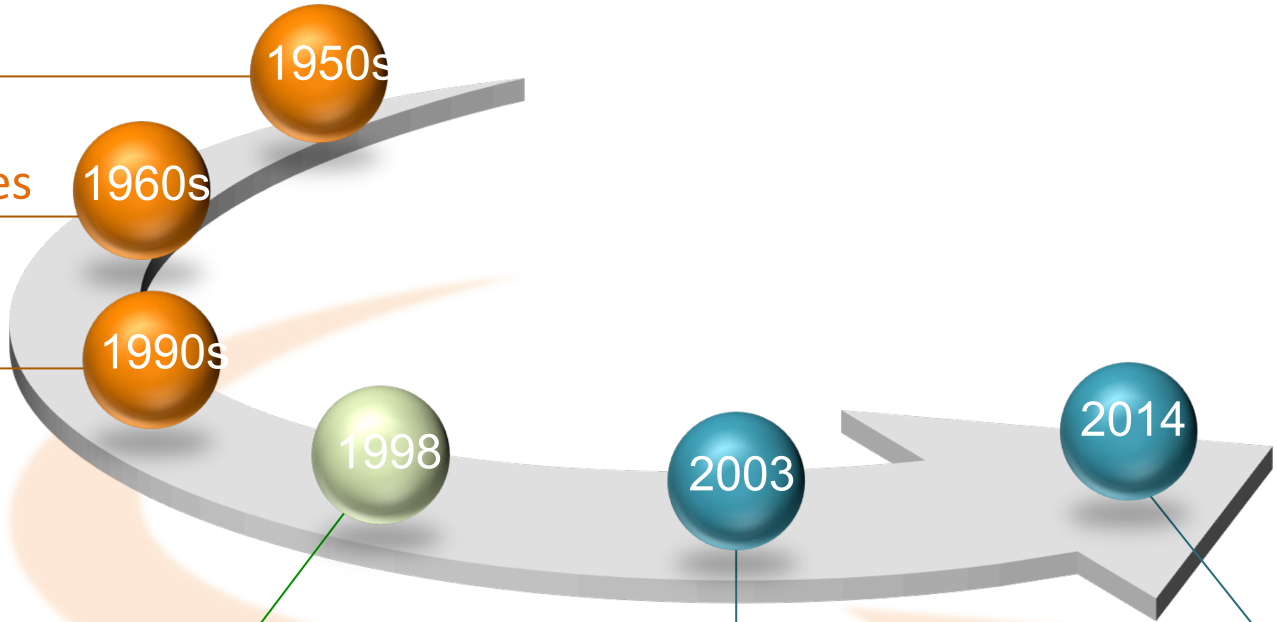
Block Codes

- Reed-Solomon (RS)

Convolutional Codes

Modern Codes

- Low-Density Parity Check Codes (LDPCs)
- Turbo



Rateless Codes QUALCOMM

Raptor and related codes

- Rate-less (refinement to End-to-End)
- Still E2E, still static

Network Codes

- RLNC enables network coding
- Some special cases allow deterministic codes
  - Index Coding
  - CATWOMAN (Linux 3.10)

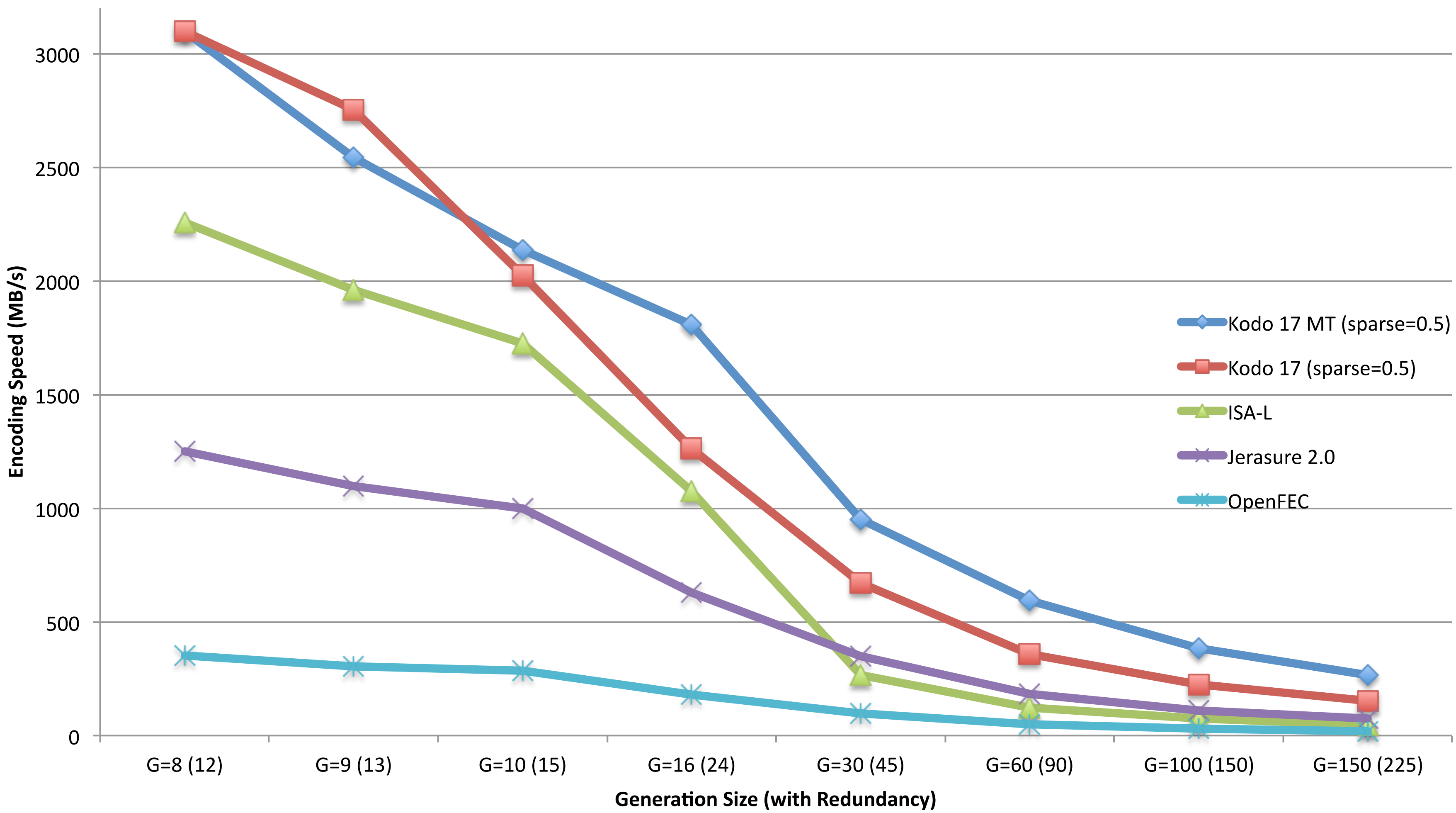
Fulcrum Codes

- RLNC-enabled
  - *Fluid complexity* (flexible field size)
  - Breaks performance-overhead trade-off

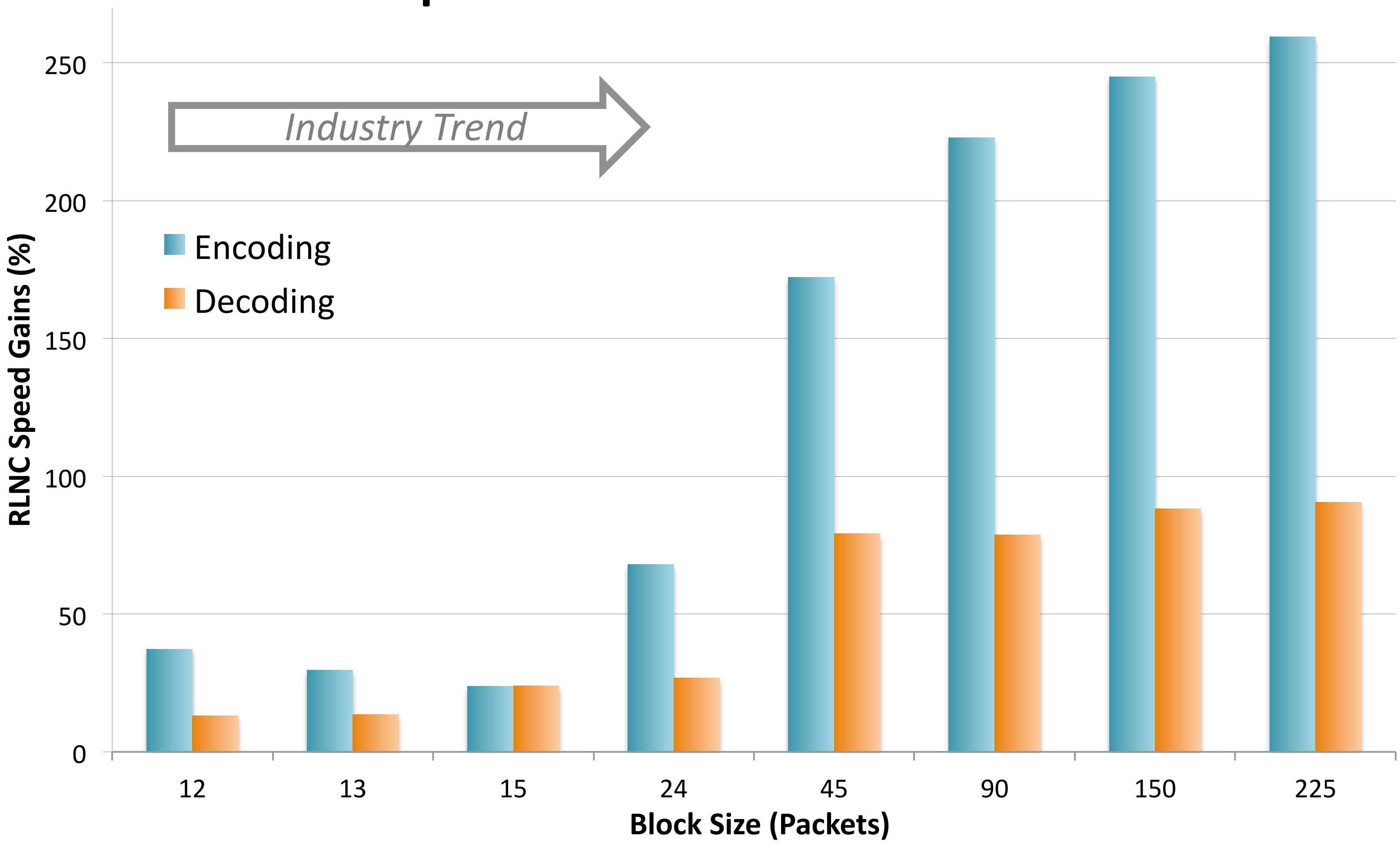
Commercial Library Benchmarking

- Jerasure 1.2 *by James Plank*
- Jerasure 2.0 *by James Plank*
- OpenFEC *by INRIA*
- ISA-L *by INTEL*
- KODO *by Steinwurf*

Comparison with State of the Art



Comparison with State of the Art

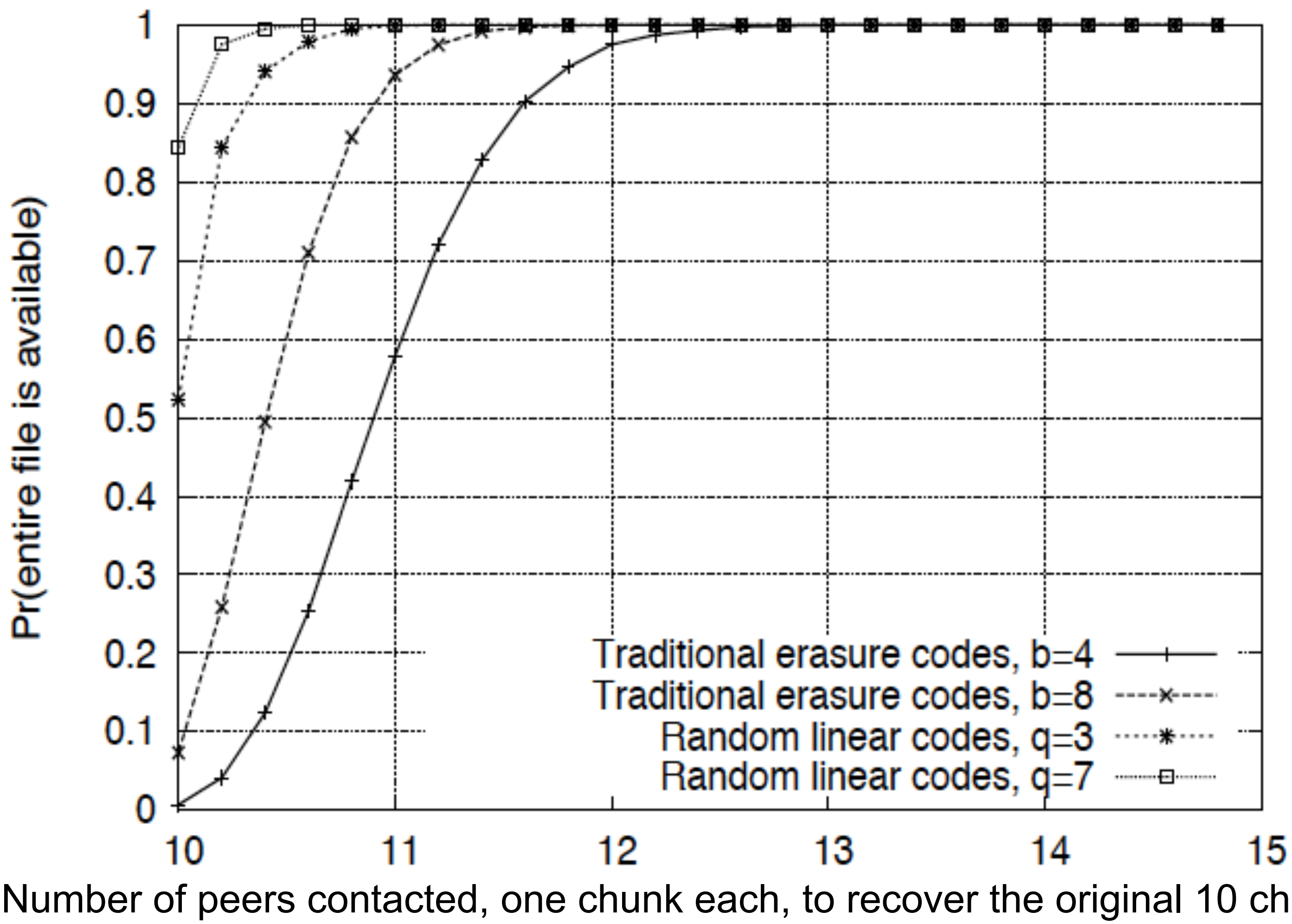


- *Benchmarking: Kodo 17 MT (0.5-sparse) vs. best of ISA-L and Jerasure 2.0*
- *1MB Packets*
- *Field Size = 2^8*
- *Code Rate = 2/3*

Overview

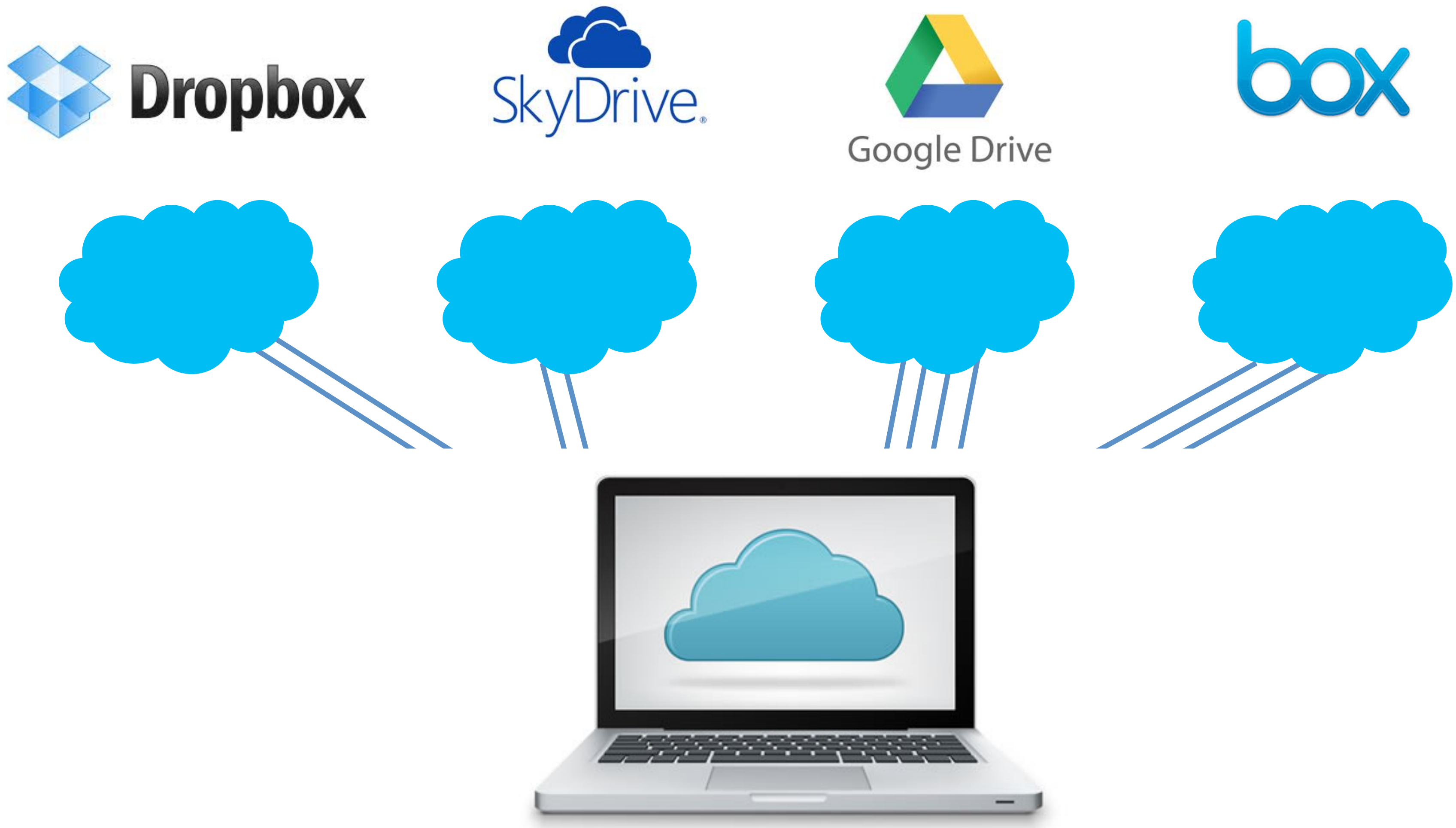
- Random linear network coding
- Distributed storage random linear network coding
- Coding in dynamic systems
- Coding for updating functions

Availability with Coding



S. Acedanski, S. Deb, Médard, M., and Koetter, R., “How Good is Random Linear Coding Based Distributed Networked Storage?”, First Workshop on Network Coding, Theory, and Applications, 2005.

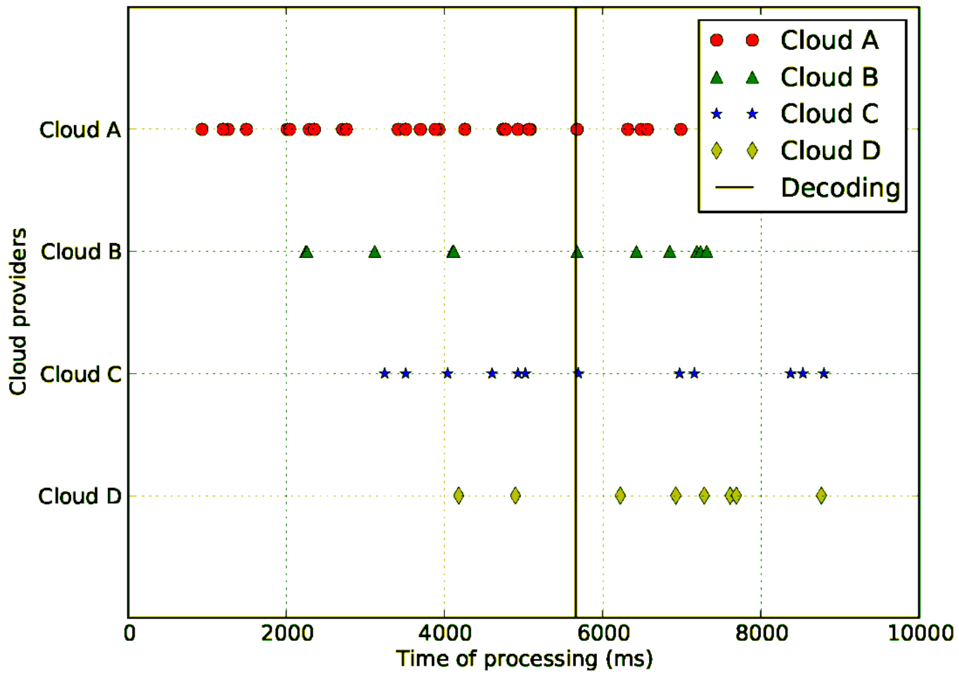
Distributed Clouds



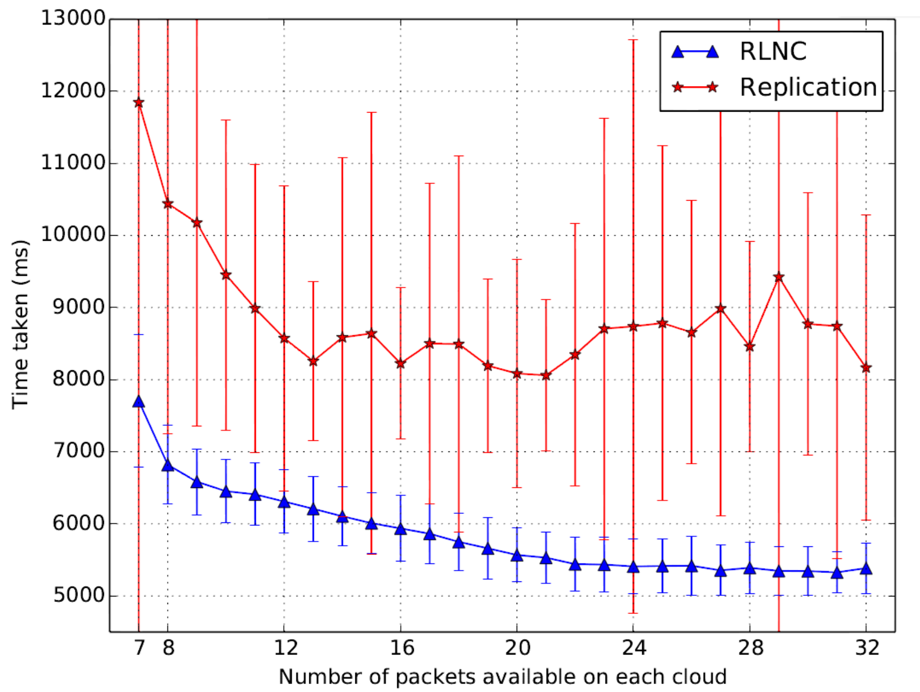
Distributed Clouds

Heterogenity (4 clouds)

- Clouds behave differently



Speed-Up (5 clouds)

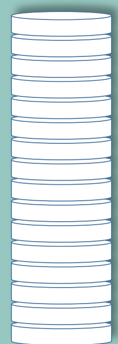


Overview

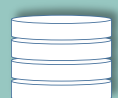
- Random linear network coding
- Distributed storage random linear network coding
- Coding in dynamic systems
- Coding for updating functions

Dynamic Robustness and Repair

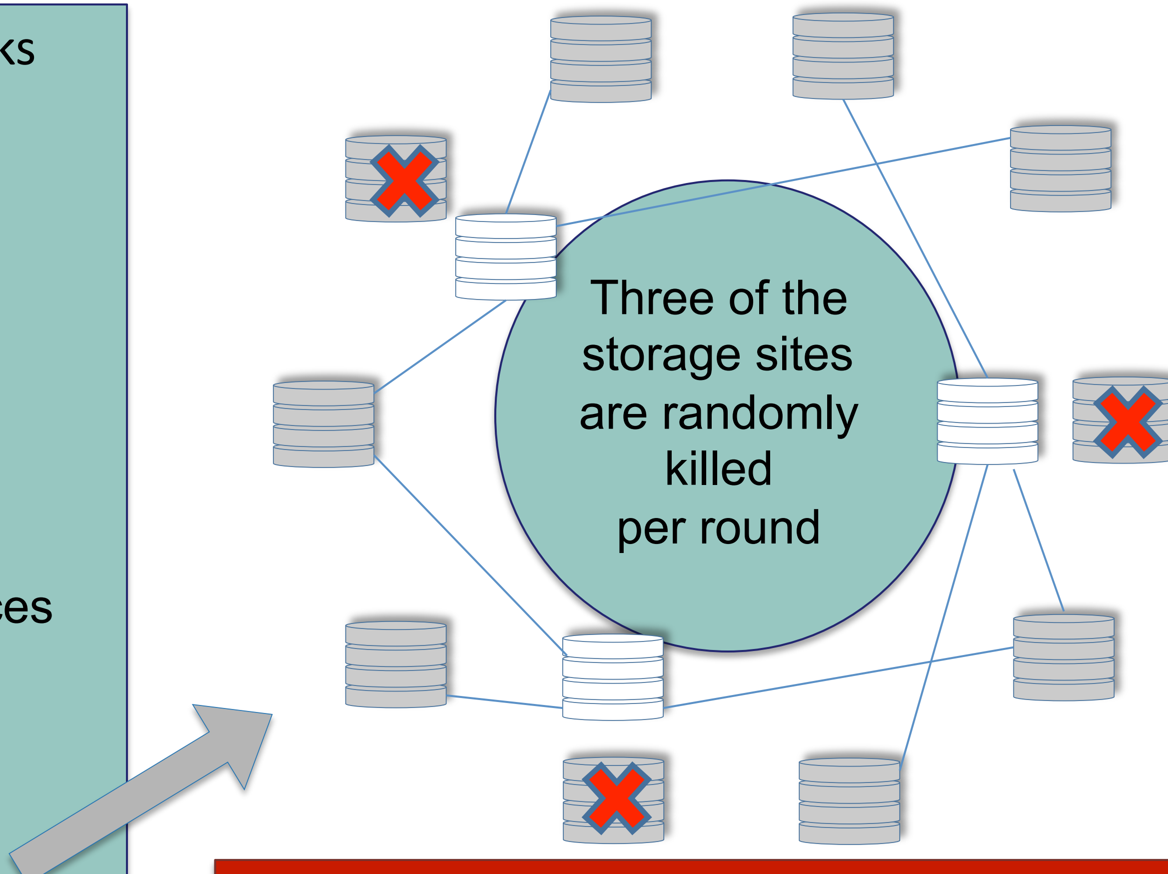
File made up of 16 chunks



Broken into 4 chunk pieces



Stored in 10 data center locations in the cloud

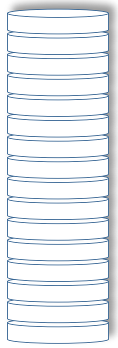


How reliably can the data be reconstructed?

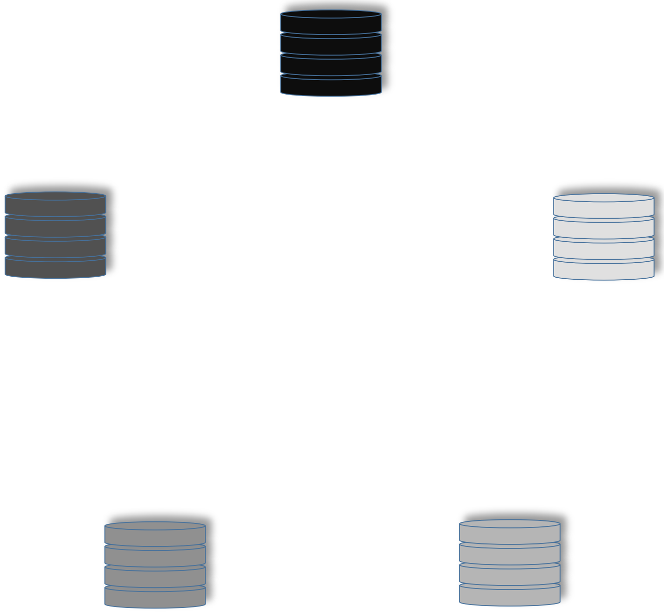
F.H.P. Fitzek, Toth, T., Szabados, A., Pedersen, M.V., Lucani, D.E., Sipos, M., Charaf, H., and Médard, M., “Implementation and Performance Evaluation of Distributed Cloud Storage Solutions using Random Linear Network Coding”, *IEEE CoCoNet 2014*

Example

File made up of 15 chunks

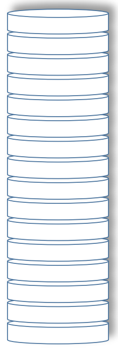


Stored in 5 racks,
4 chunks each
Redundancy 33%

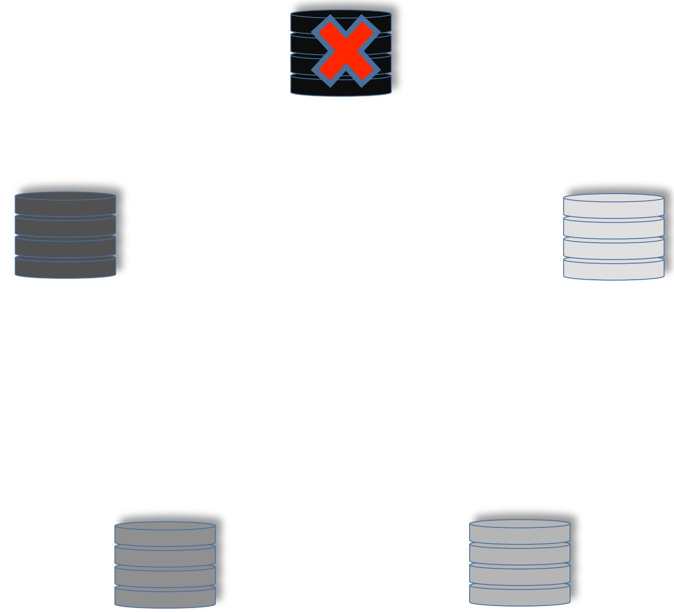


Reed-Solomon

File made up of 15 chunks

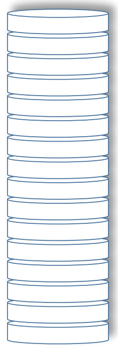


Stored in 5 racks,
4 chunks each
Redundancy 33%

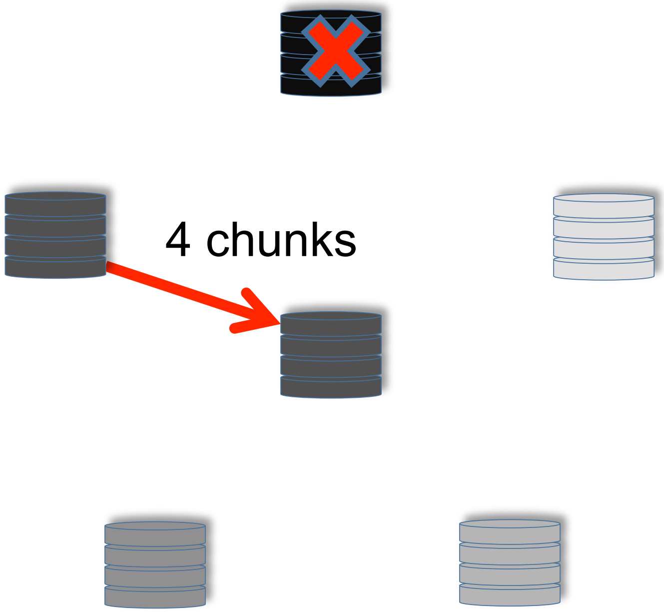


Reed-Solomon

File made up of 15 chunks

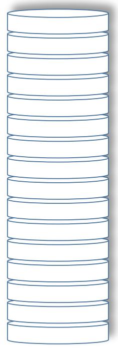


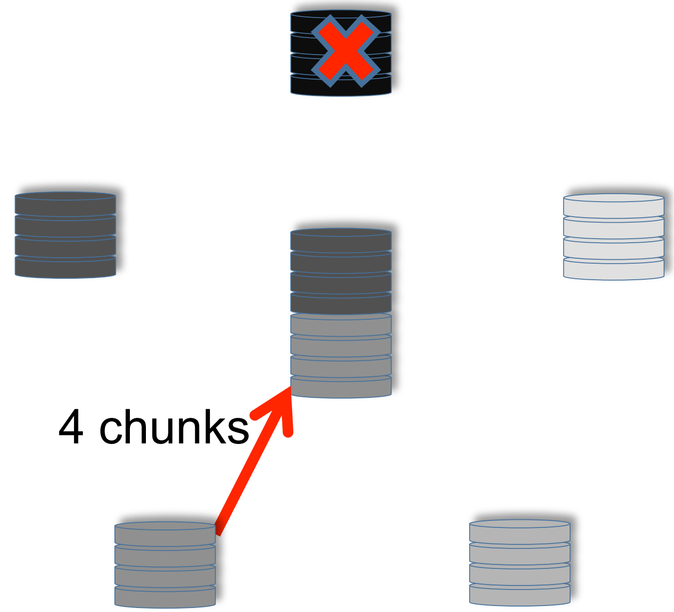
Stored in 5 racks,
4 chunks each
Redundancy 33%



Reed-Solomon

File made up of 15 chunks

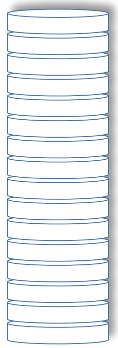




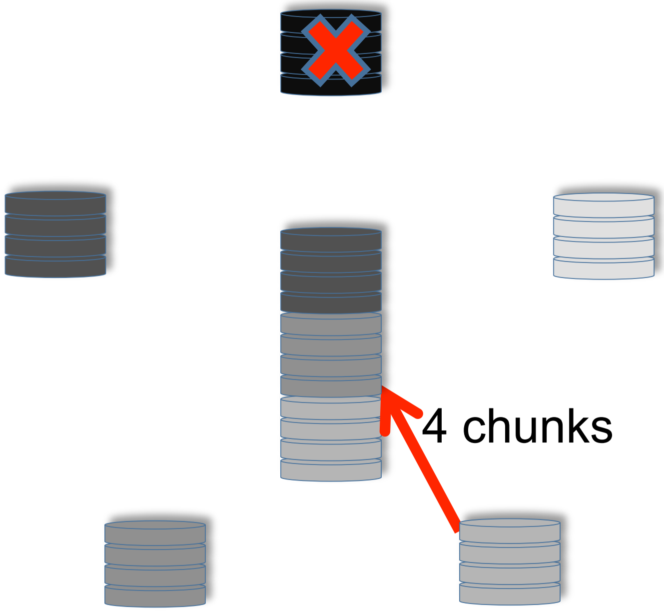
Stored in 5 racks,
4 chunks each
Redundancy 33%

Reed-Solomon

File made up of 15 chunks

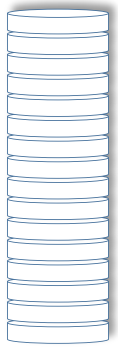


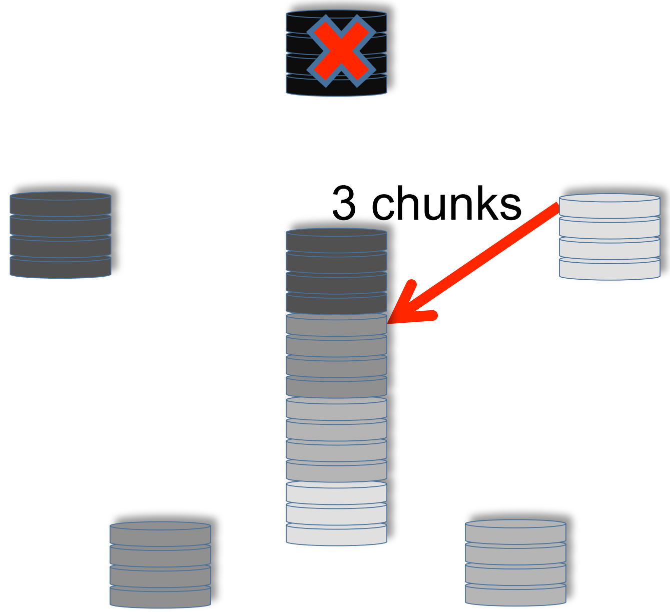
Stored in 5 racks,
4 chunks each
Redundancy 33%



Reed-Solomon

File made up of 15 chunks

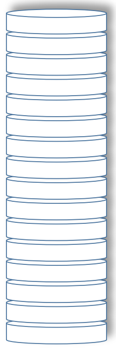




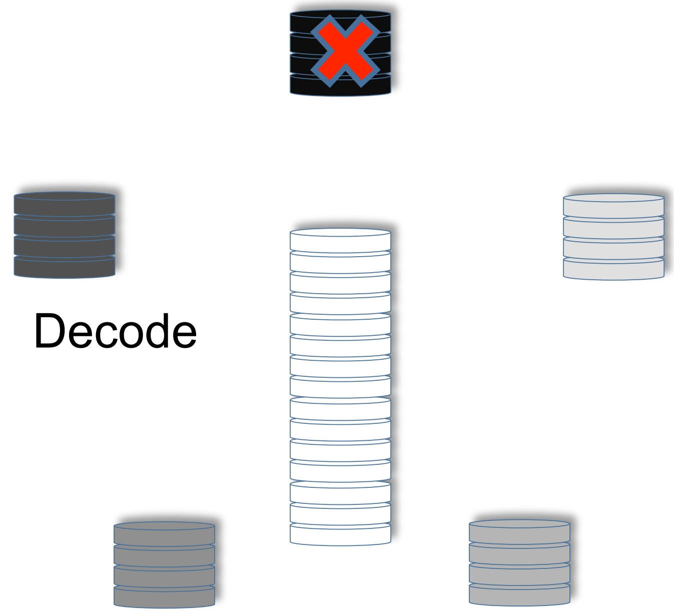
Stored in 5 racks,
4 chunks each
Redundancy 33%

Reed-Solomon

File made up of 15 chunks

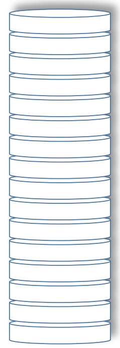


Stored in 5 racks,
4 chunks each
Redundancy 33%

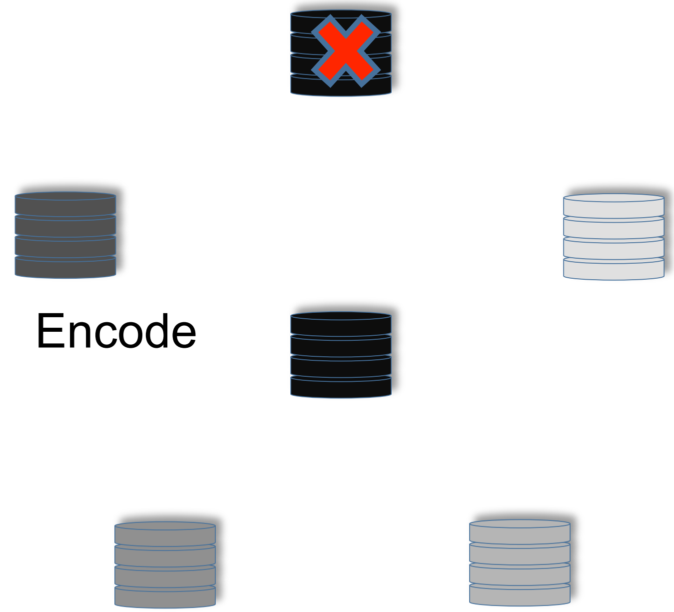


Reed-Solomon

File made up of 15 chunks

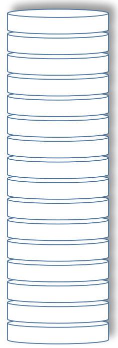


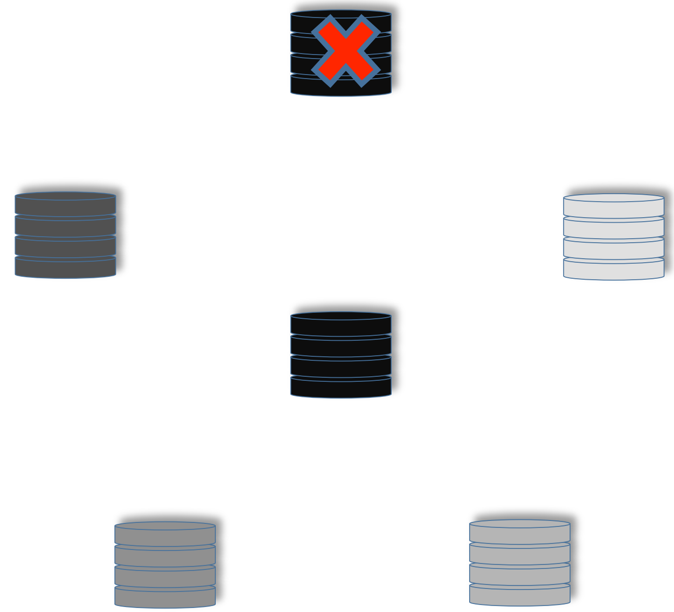
Stored in 5 racks,
4 chunks each
Redundancy 33%



Reed-Solomon

File made up of 15 chunks





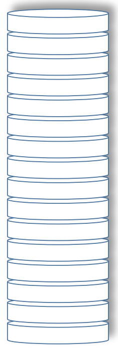
Stored in 5 racks,
4 chunks each
Redundancy 33%

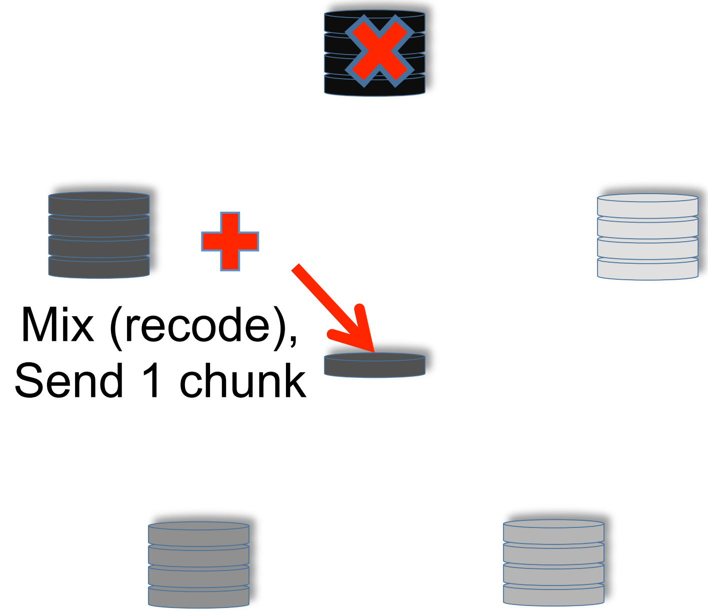
	I/O	Network: Intra-Rack	Inter-Rack	Processing
RS:	15	0*	15	Decode + Encode 15x15 matrix (new rack)
RLNC:				

* May require some intra-rack transfer depending on structure

RLNC

File made up of 15 chunks



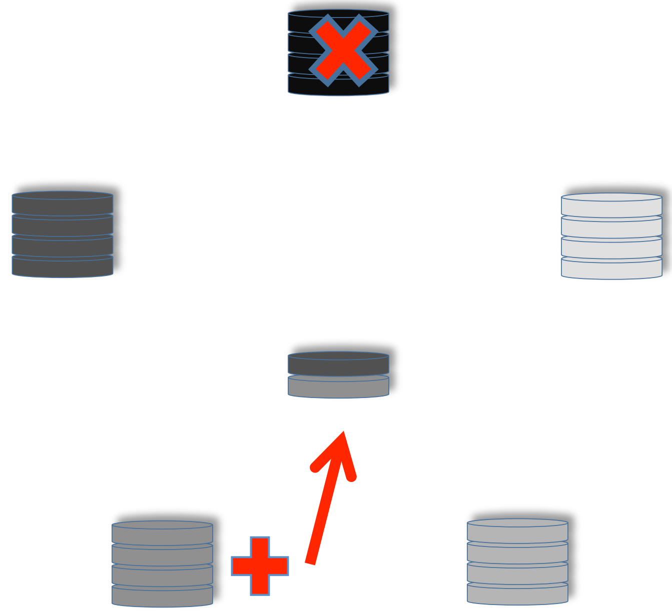


Stored in 5 racks,
4 chunks each
Redundancy 33%

RLNC

File made up of 15 chunks

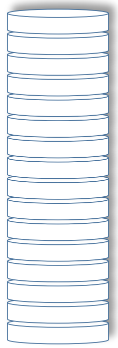
Stored in 5 racks,
4 chunks each
Redundancy 33%



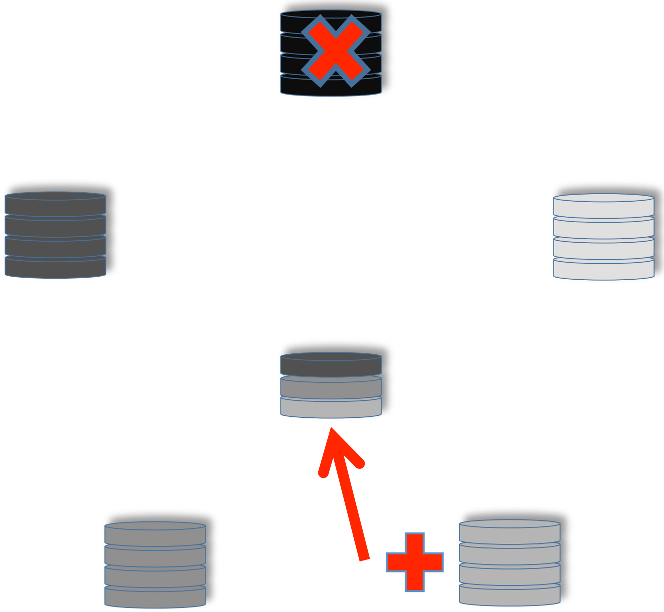
Mix (recode),
Send 1 chunk

RLNC

File made up of 15 chunks



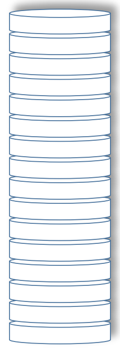
Stored in 5 racks,
4 chunks each
Redundancy 33%



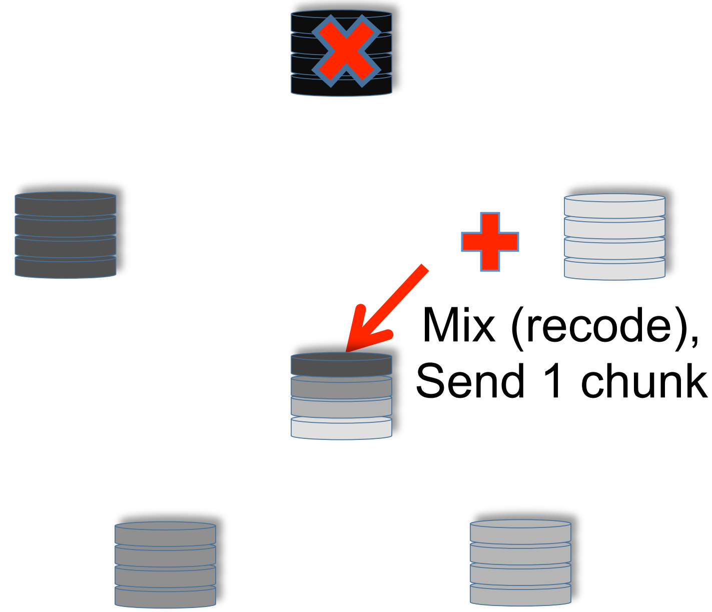
Mix (recode),
Send 1 chunk

RLNC

File made up of 15 chunks

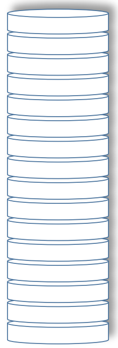


Stored in 5 racks,
4 chunks each
Redundancy 33%

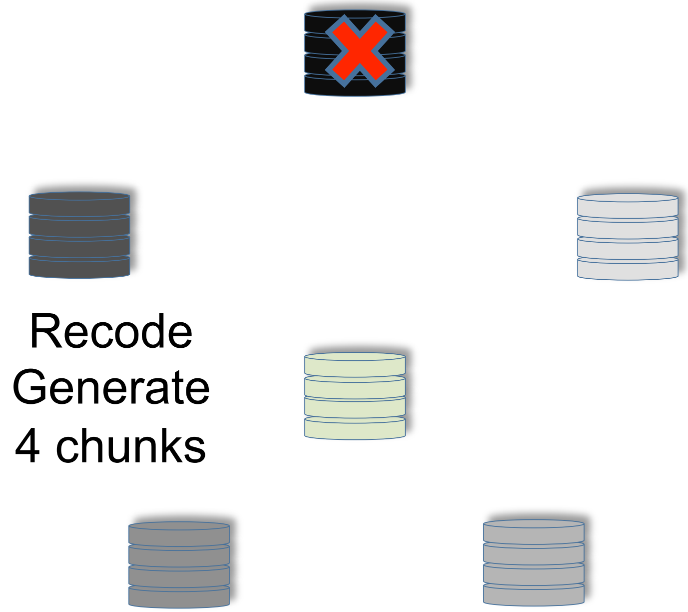


RLNC

File made up of 15 chunks

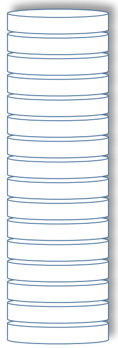


Stored in 5 racks,
4 chunks each
Redundancy 33%

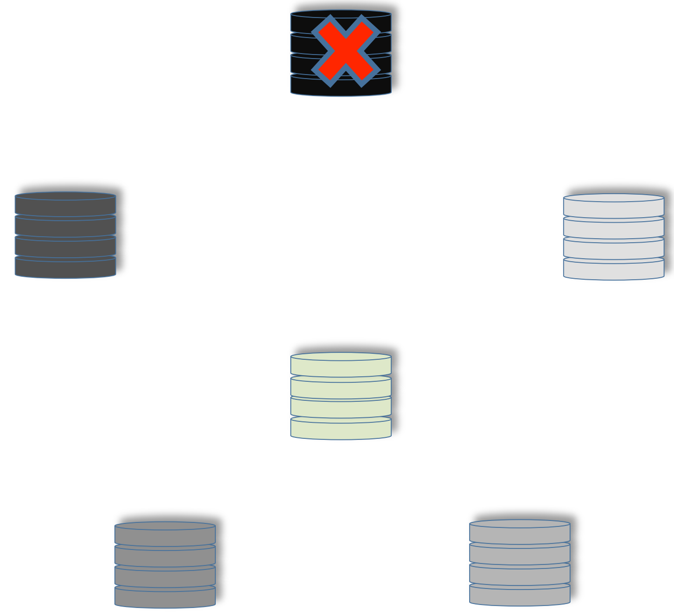


RLNC

File made up of 15 chunks



Stored in 5 racks,
4 chunks each
Redundancy 33%

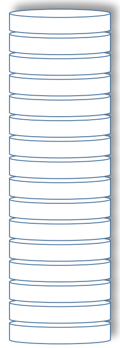


	I/O	Network: Intra-Rack	Inter-Rack	Processing
RS:	15	0*	15	Decode + Encode 15x15 matrix (new rack)
RLNC:	15	11	4	Encode 4x4 matrices (4 times), and one 3x3 matrix

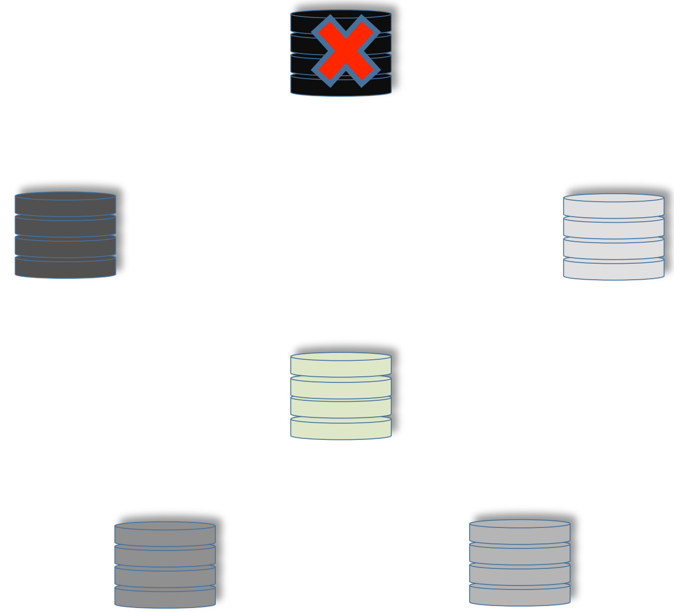
* May require some intra-rack transfer depending on structure

RLNC

File made up of 15 chunks



Stored in 5 racks,
4 chunks each
Redundancy 33%

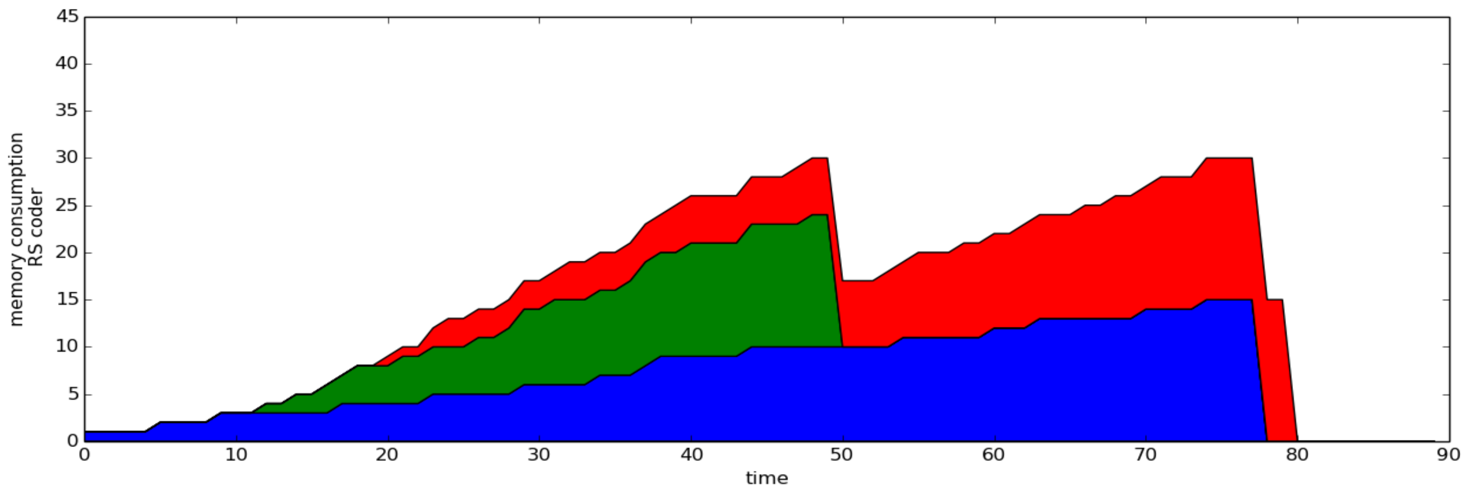


	I/O	Network: Intra-Rack	Inter-Rack	Processing
RS:	15	0*	15	Centralized in new rack
RLNC:	15	11	4	Distributed in old and new racks

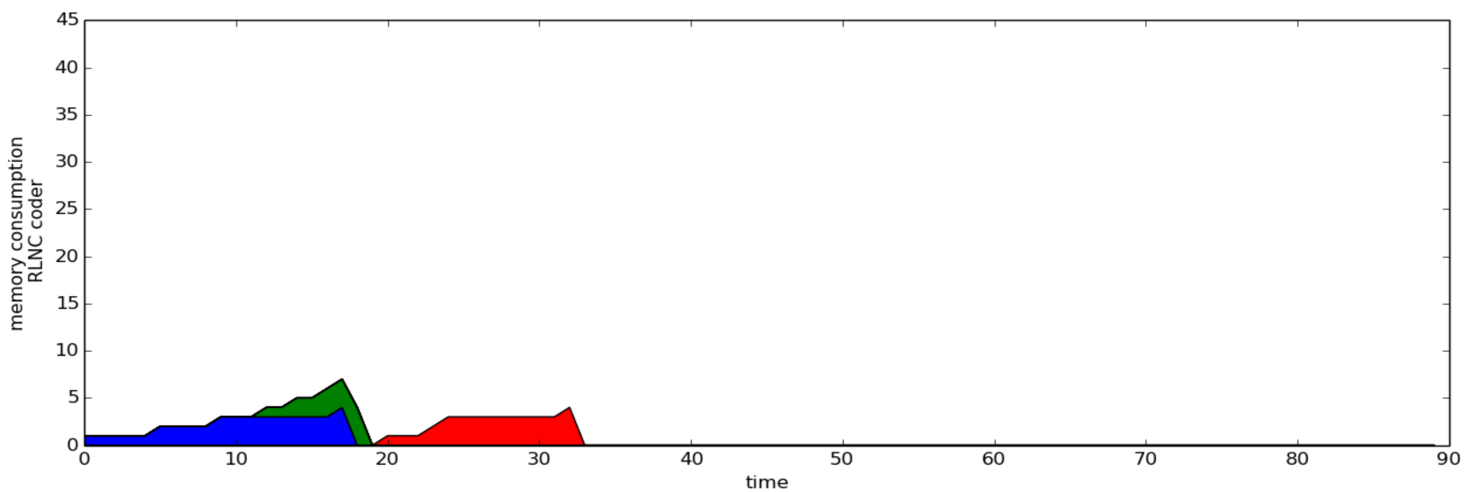
* May require some intra-rack transfer depending on structure

Memory Consumption RS vs RLNC

Reed-Solomon

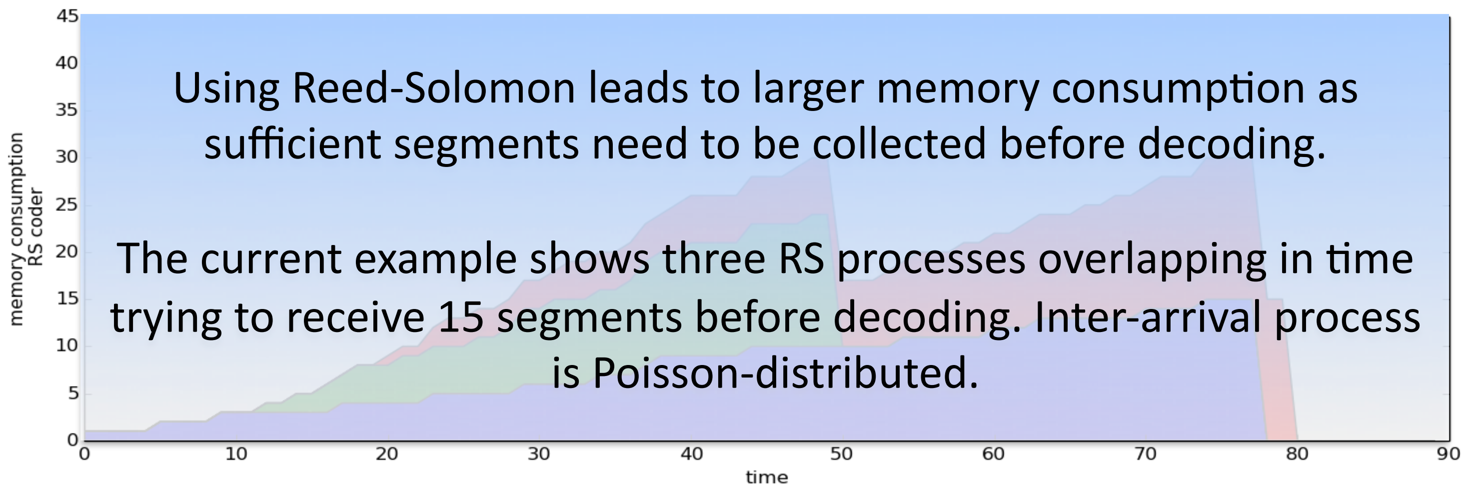


RLNC



Memory Consumption RS vs RLNC

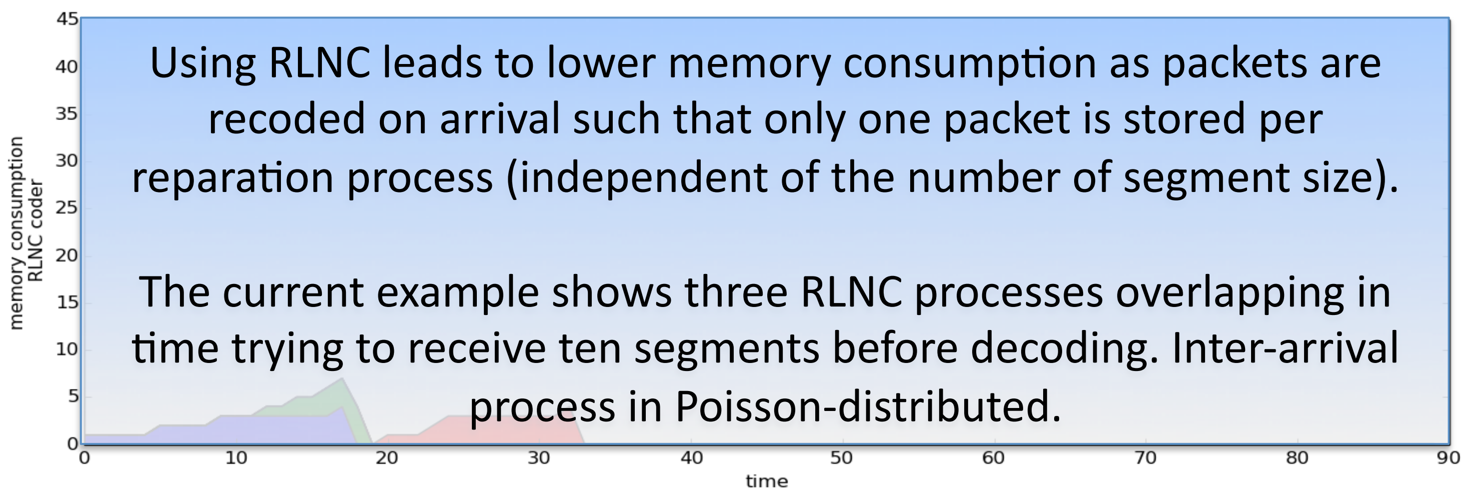
Reed-Solomon



Using Reed-Solomon leads to larger memory consumption as sufficient segments need to be collected before decoding.

The current example shows three RS processes overlapping in time trying to receive 15 segments before decoding. Inter-arrival process is Poisson-distributed.

RLNC

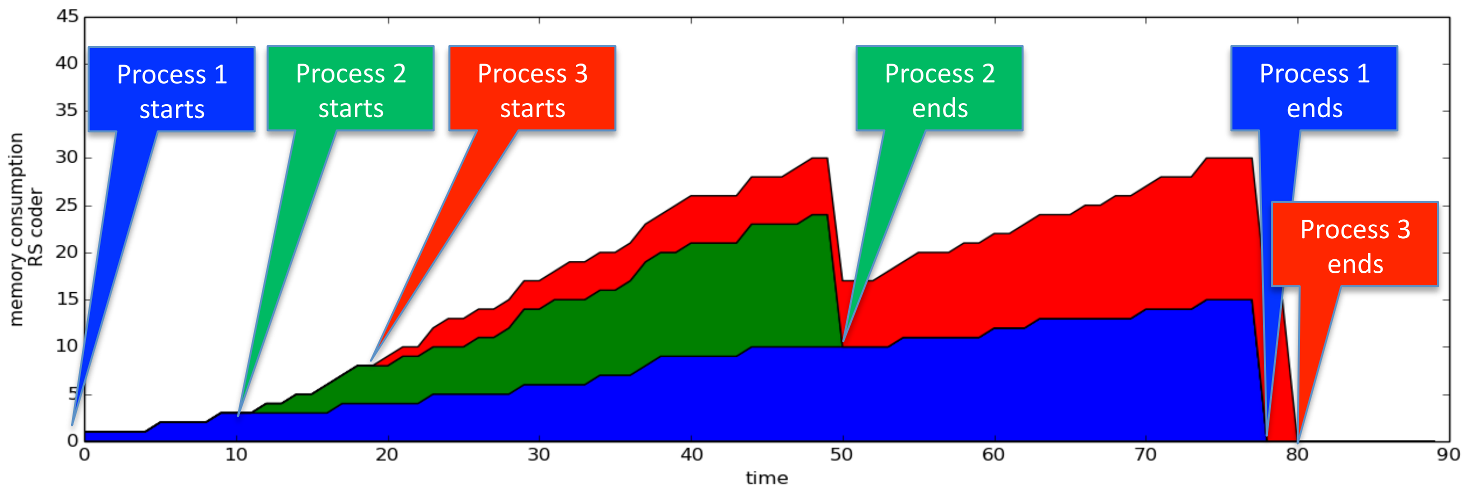


Using RLNC leads to lower memory consumption as packets are recoded on arrival such that only one packet is stored per reparation process (independent of the number of segment size).

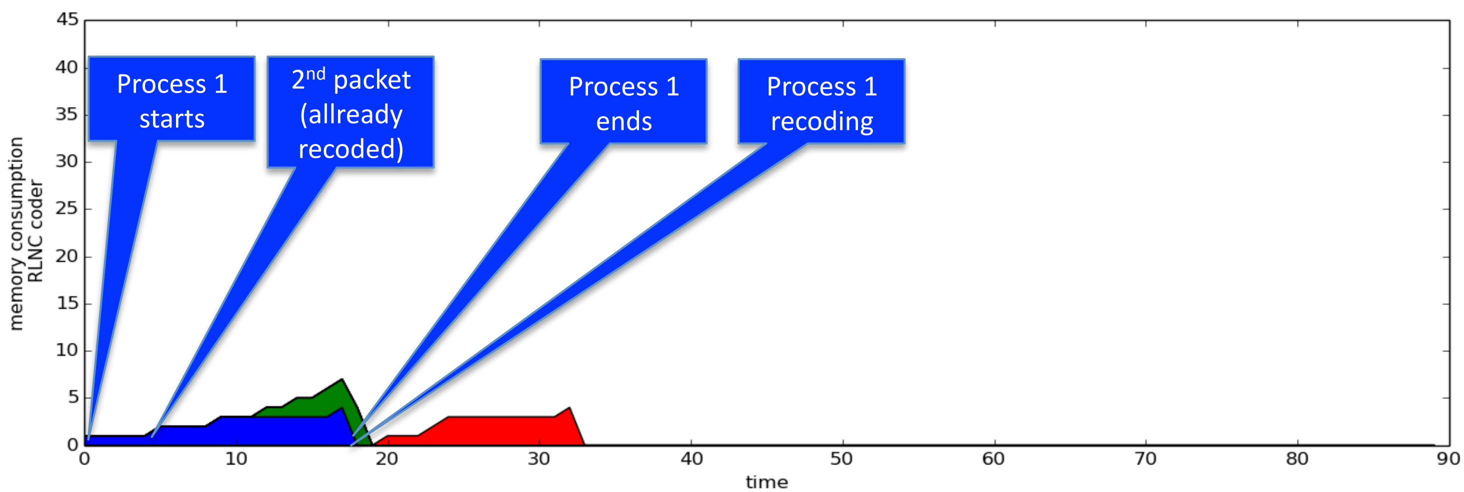
The current example shows three RLNC processes overlapping in time trying to receive ten segments before decoding. Inter-arrival process in Poisson-distributed.

Memory Consumption RS vs RLNC

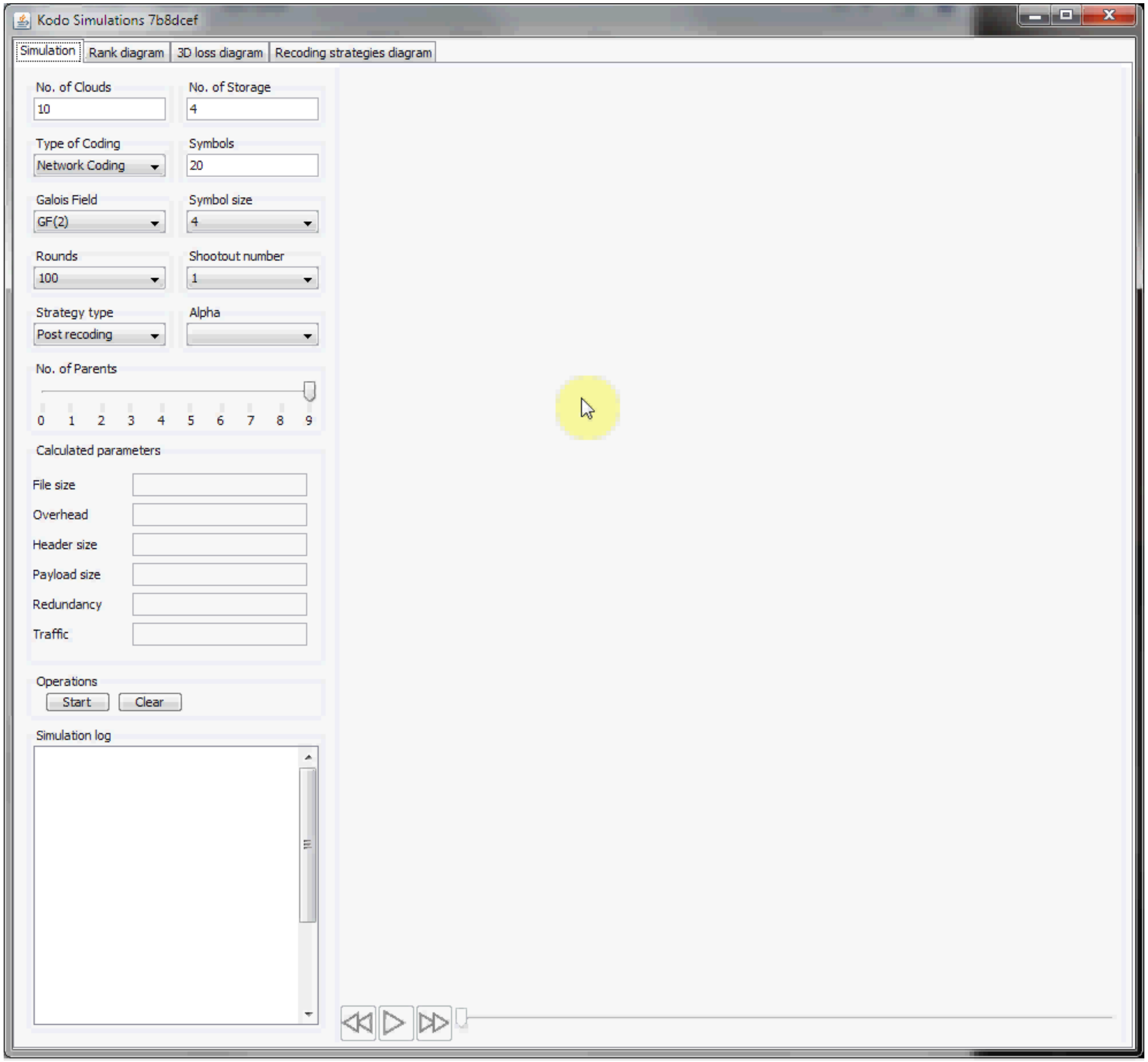
Reed-Solomon



RLNC

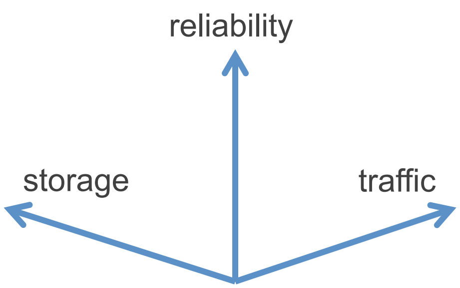


Video

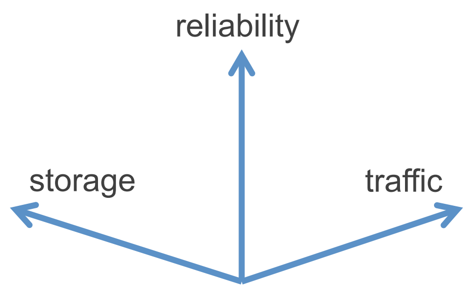

Kodo Simulations 7b8dcef
Simulation
Rank diagram
3D loss diagram
Recoding strategies diagram
No. of Clouds
10
No. of Storage
4
Type of Coding
Network Coding
Symbols
20
Galois Field
GF(2)
Symbol size
4
Rounds
100
Shootout number
1
Strategy type
Post recoding
Alpha
No. of Parents
0 1 2 3 4 5 6 7 8 9
Calculated parameters
File size
Overhead
Header size
Payload size
Redundancy
Traffic
Operations
Start
Clear
Simulation log

Dynamic Robustness and Repair

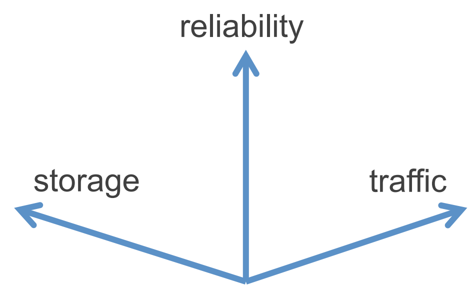
No Coding

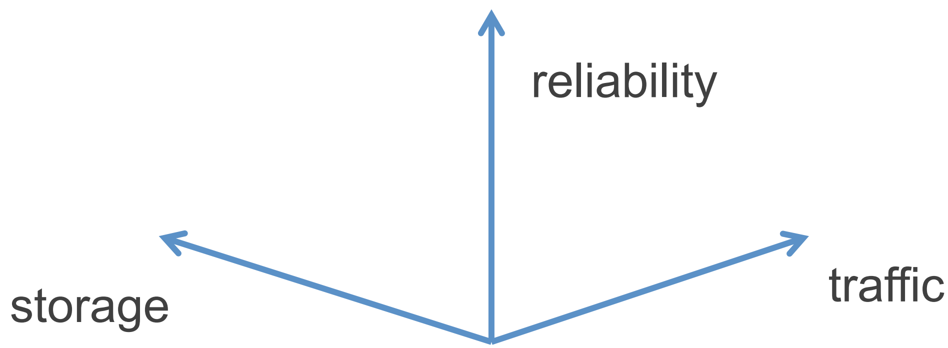


RS Coding



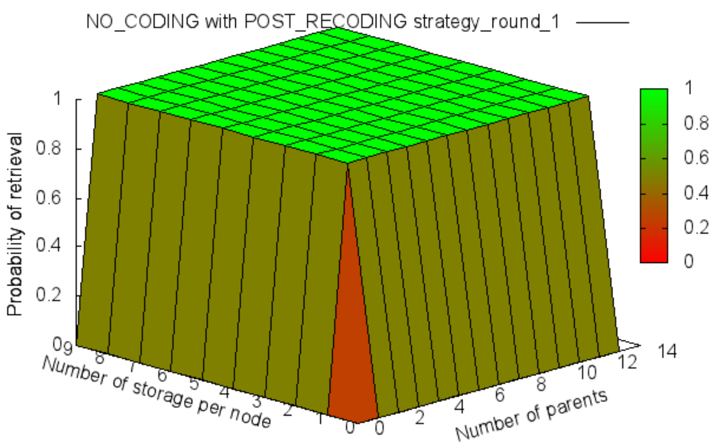
Network Coding



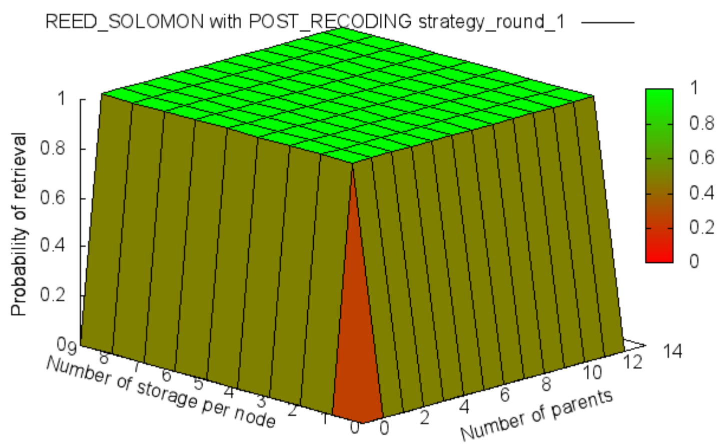


Dynamic Robustness and Repair

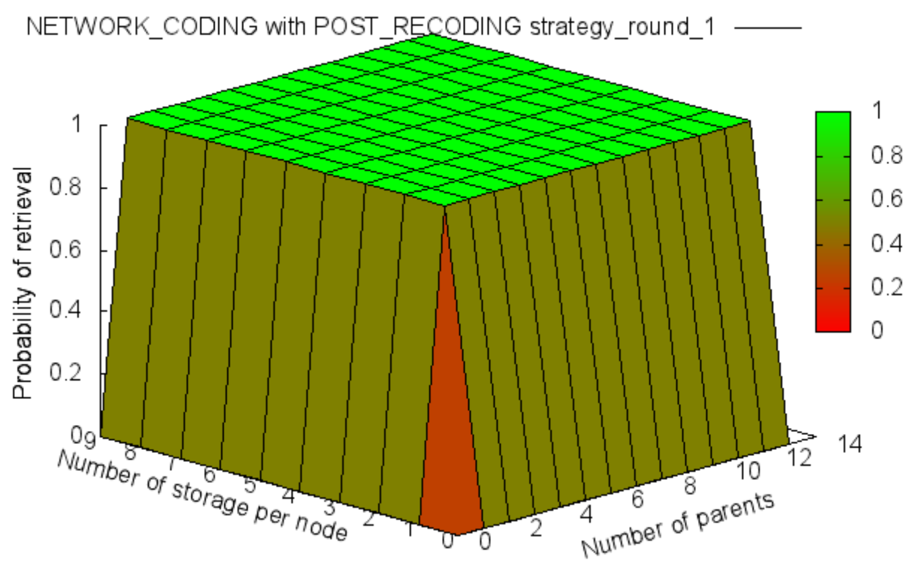
No Coding



RS Coding



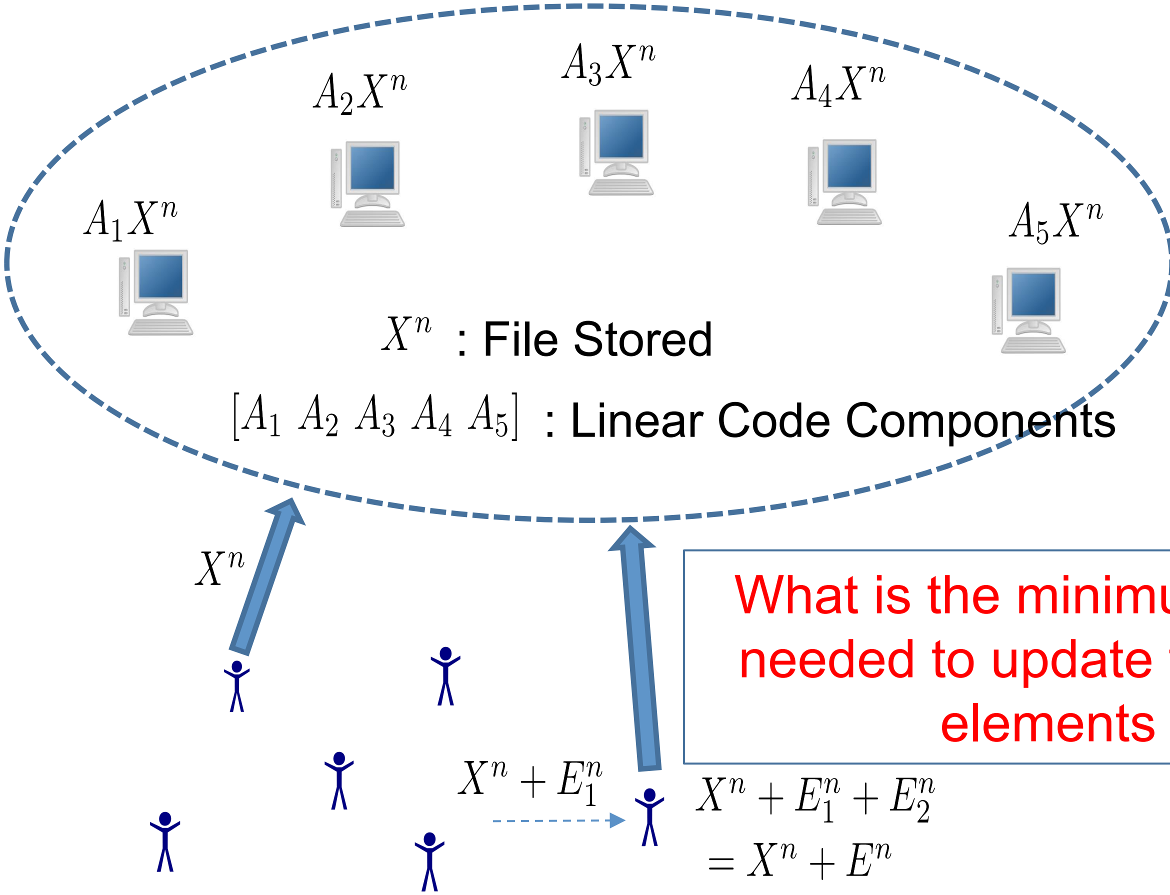
Network Coding



Overview

- Random linear network coding
- Distributed storage random linear network coding
- Coding in dynamic systems
- Coding for updating functions

Motivation

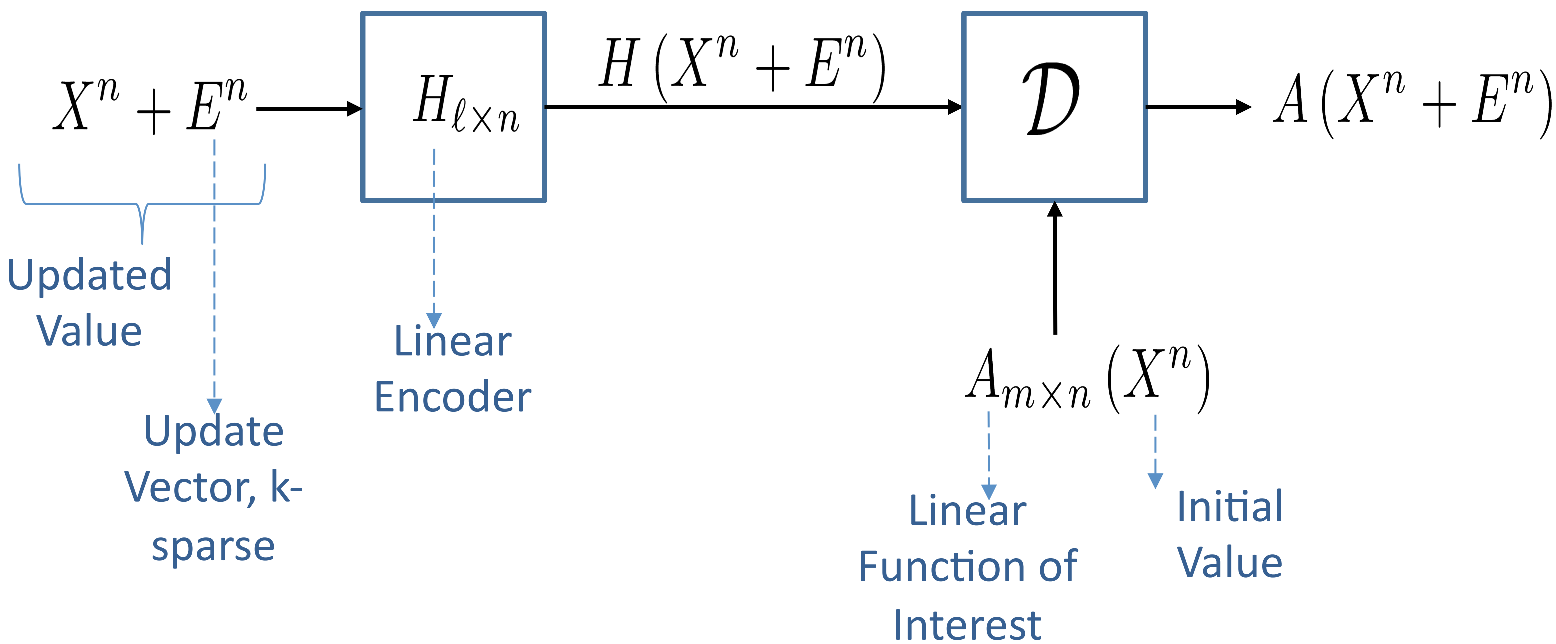


Decentralized, Distributed Storage Nodes - e.g., P2P or Cloud based

What is the minimum upload needed to update the coded elements ?

- Current solutions require precise knowledge/tracking of the update vectors
- Our solution relies only on estimates of sparsity of the update vectors

What about Computation?



What is the minimum communication necessary for the update ?

- Zero probability of error, worst-case scenario
- The function A and sparsity-parameter k are known at the source

P. Narayana Moorthy and Médard, M. “Communication Cost for Updating Functions when Message Updates are Sparse: Connections to Maximally Recoverable Codes”, **invited paper,** *Allerton 2015*

Illustrating Matrix for Striped Data File

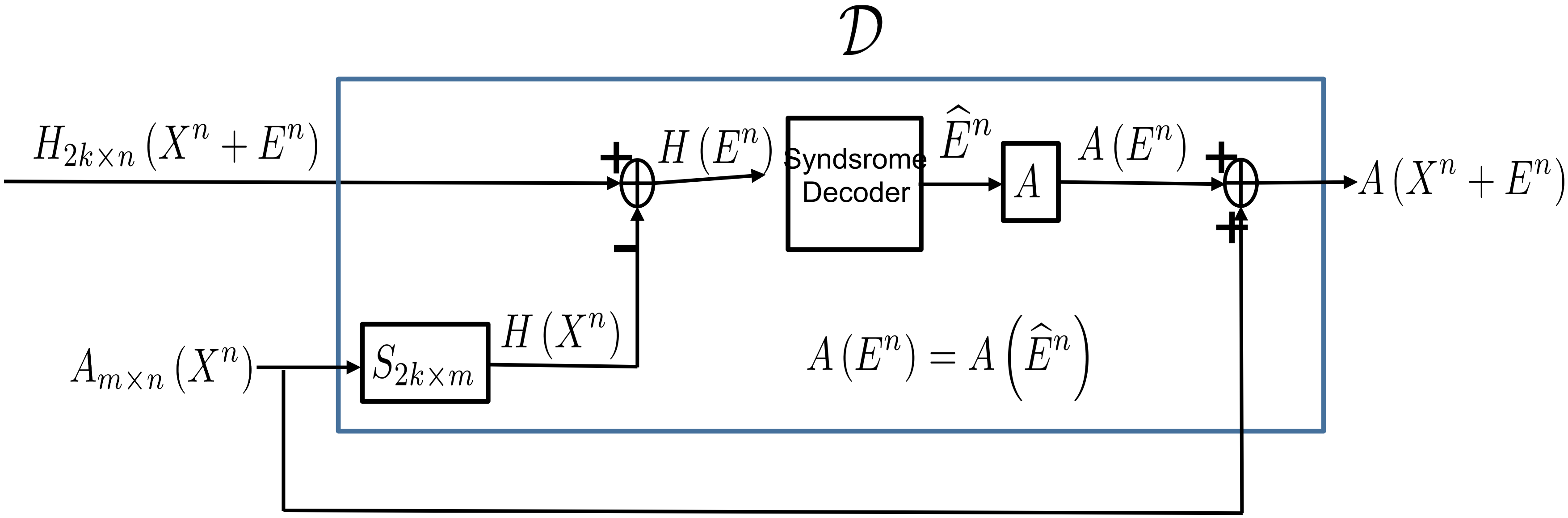
- E.g. [Length = 5, Dimension = 3] scalar linear code for storage
- $\mathbf{a}_1 = [a_{1,1} \;\; a_{1,2} \;\; a_{1,3}]$ coding coefficients for first storage node

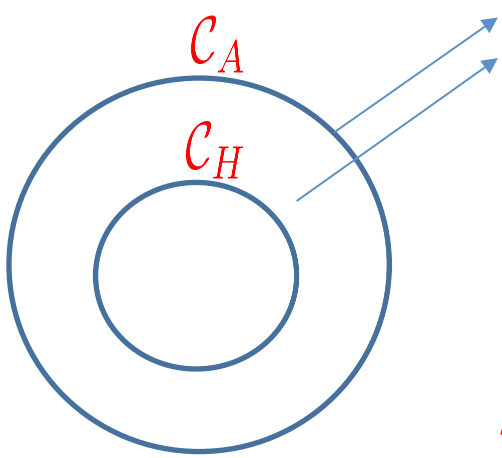
$m =$ number of stripes

$$\underbrace{\begin{bmatrix} \mathbf{a}_1 & & & \\ & \mathbf{a}_1 & & \\ & & \ddots & \\ & & & \mathbf{a}_1 \end{bmatrix}}_{A_{m\times n} = A_1} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ \vdots \\ X_{n-2} \\ X_{n-1} \\ X_n \end{bmatrix} \quad \begin{matrix} \left.\begin{matrix} X_1 \\ X_2 \\ X_3 \end{matrix}\right\} 1^{\text{st}} \text{ stripe} \\ \vdots \\ \left.\begin{matrix} X_{n-2} \\ X_{n-1} \\ X_n \end{matrix}\right\} m^{\text{th}} \text{ stripe} \end{matrix}$$

$n = 3m$

Point-to-Point : Achievable Scheme with $\ell = 2k$





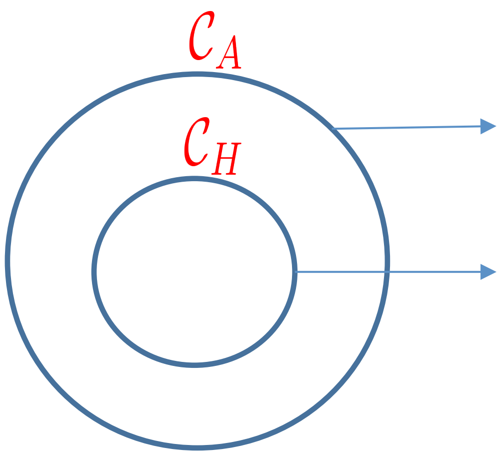
$[n, m]$ code generated by rows of A

$[n, 2k]$ subcode generated by rows of H $\quad m > 2k$

1. $H\left(E^n\right) = H\left(\widehat{E}^n\right) \iff A\left(E^n\right) = A\left(\widehat{E}^n\right)$

2. Matrix A always exists under sufficiently large field size

Maximally Recoverable Codes : Definition



$[n, m]$ code generated by rows of A

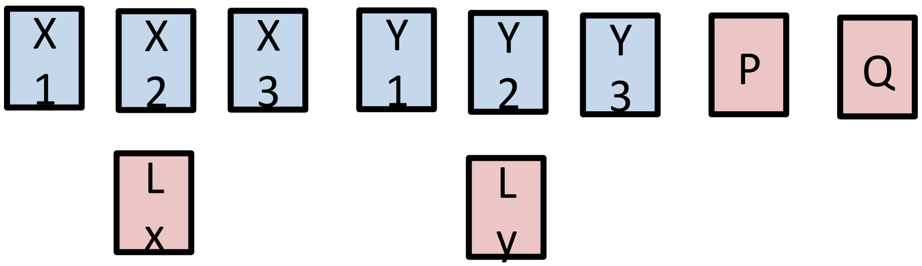
$[n, 2k]$ subcode generated by rows of H

$\mathcal{C}_H$ is a Maximally Recoverable Subcode of $\mathcal{C}_A$ if

$$\mathrm{rank}\left(A|_S\right) = 2k \implies \mathrm{rank}\left(H|_S\right) = 2k, \quad \forall S, |S| = 2k$$

$$A = \begin{bmatrix} 1 & 1 & 1 & & & & & & \\ & & & 1 & 1 & 1 & & & \\ & & & & & & 1 & 1 & 1 \end{bmatrix} \qquad H = \begin{bmatrix} 1 & 1 & 1 & & & & 1 & 1 & 1 \\ & & & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

MRCs with Locality in Windows Azure Storage



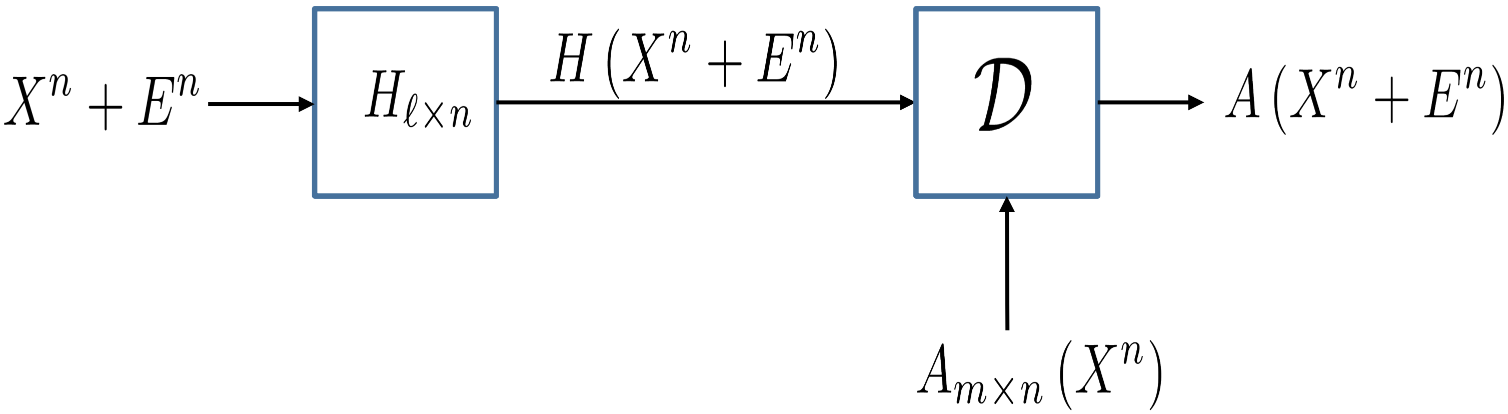
$\mathcal{C}_H \; : \; [n = 10, 2k = 6]$ code

$$A = \begin{bmatrix} 1 & & & 1 & & & & & & \\ & 1 & & 1 & & & & & & \\ & & 1 & 1 & & & & & & \\ & & & & 1 & & & 1 & & \\ & & & & & 1 & & 1 & & \\ & & & & & & 1 & 1 & & \\ & & & & & & & & 1 & \\ & & & & & & & & & 1 \end{bmatrix} \quad H = \begin{bmatrix} 1 & & & 1 & & & & & p_1 & q_1 \\ & 1 & & 1 & & & & & p_2 & q_2 \\ & & 1 & 1 & & & & & p_3 & q_3 \\ & & & & 1 & & & 1 & p_4 & q_4 \\ & & & & & 1 & & 1 & p_5 & q_5 \\ & & & & & & 1 & 1 & p_6 & q_6 \end{bmatrix}$$

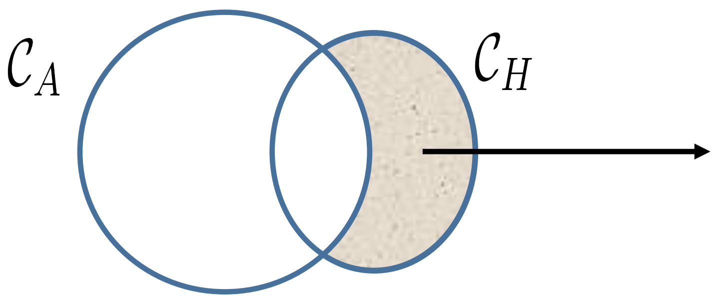
Property of MRCs with Locality

- Data decodable from any 6 symbols that are not “dominated” by either of the two local codes
  - E.g. {X1, Lx, Y1, Ly, P, Q}
- For this reason, MRCs with locality are better known as Partial MDS codes
  - “as MDS as possible” given the locality constraints

Point-to-Point : Converse Statements

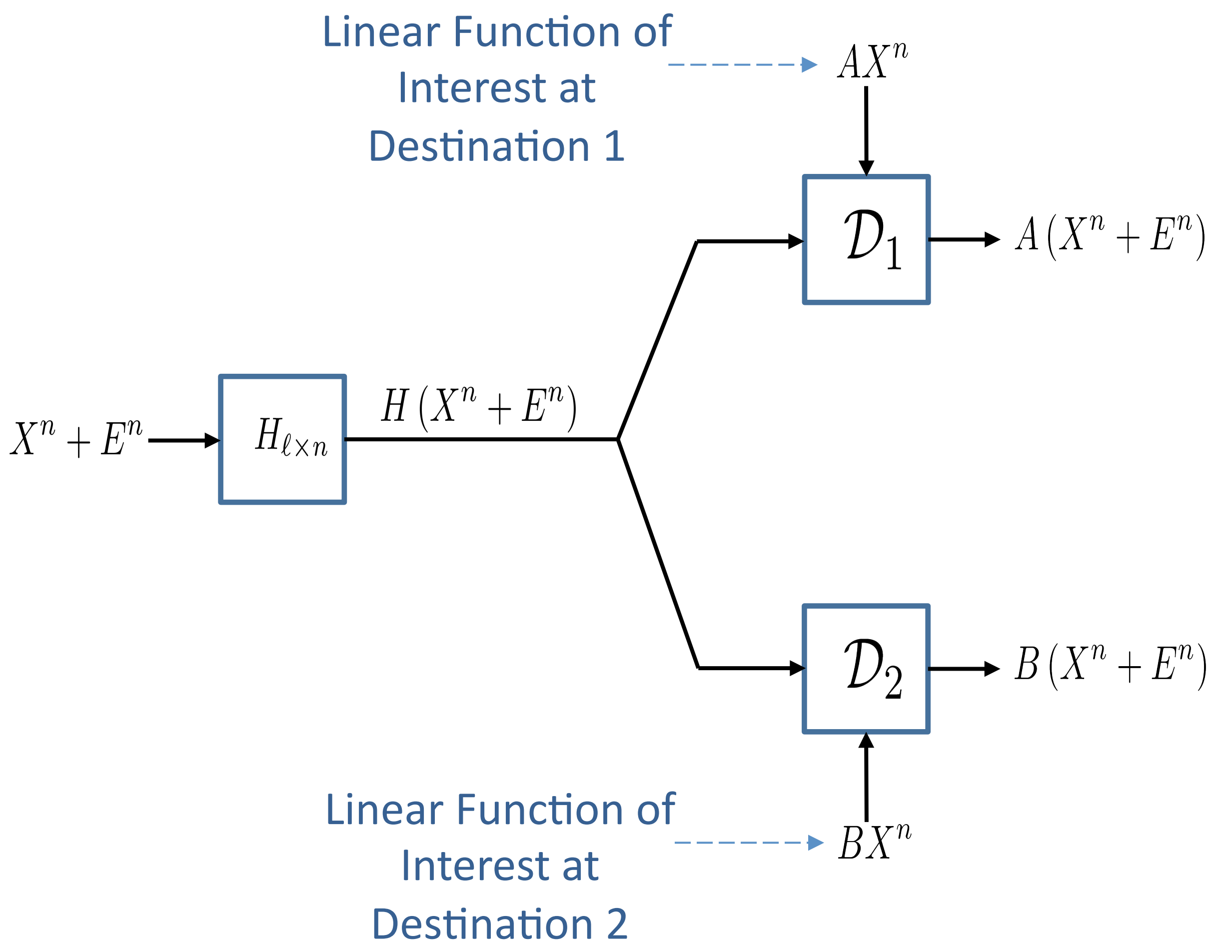


- $\ell \geq 2k$ (assuming $\operatorname{rank}(A) \geq 2k$)
- Under optimality, $\mathcal{C}_H$ must be a $2k$ dimensional maximally recoverable subcode of $\mathcal{C}_A$



Components outside of $\mathcal{C}_A$ are not useful

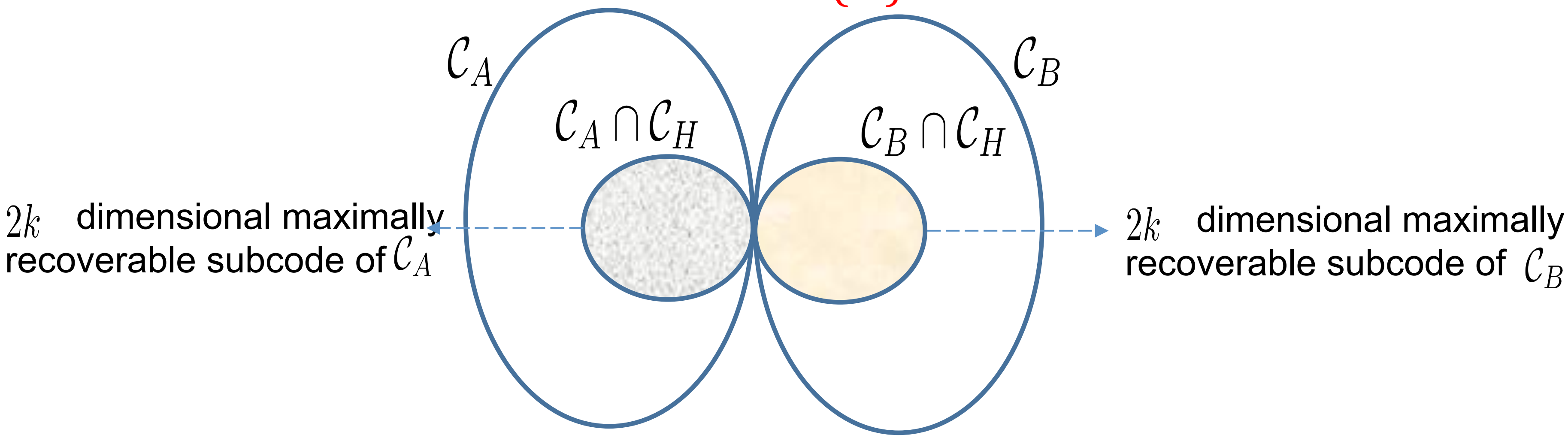
Broadcast Setting : Problem Statement



What is the minimum communication necessary for updating both destinations simultaneously ?

Special Case:

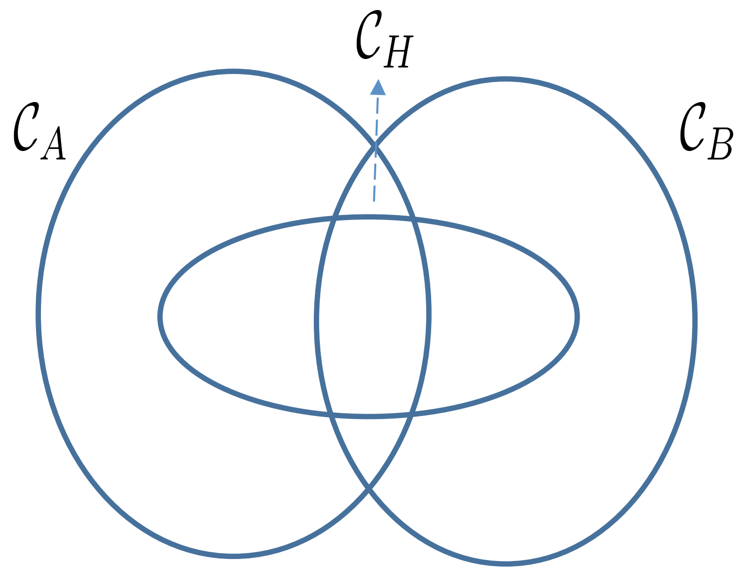
$$\mathcal{C}_A \cap \mathcal{C}_B = \{\underline{0}\}$$



- $\ell \geq 4k$

- Optimal to transmit individually to the two destinations – No benefit from broadcasting

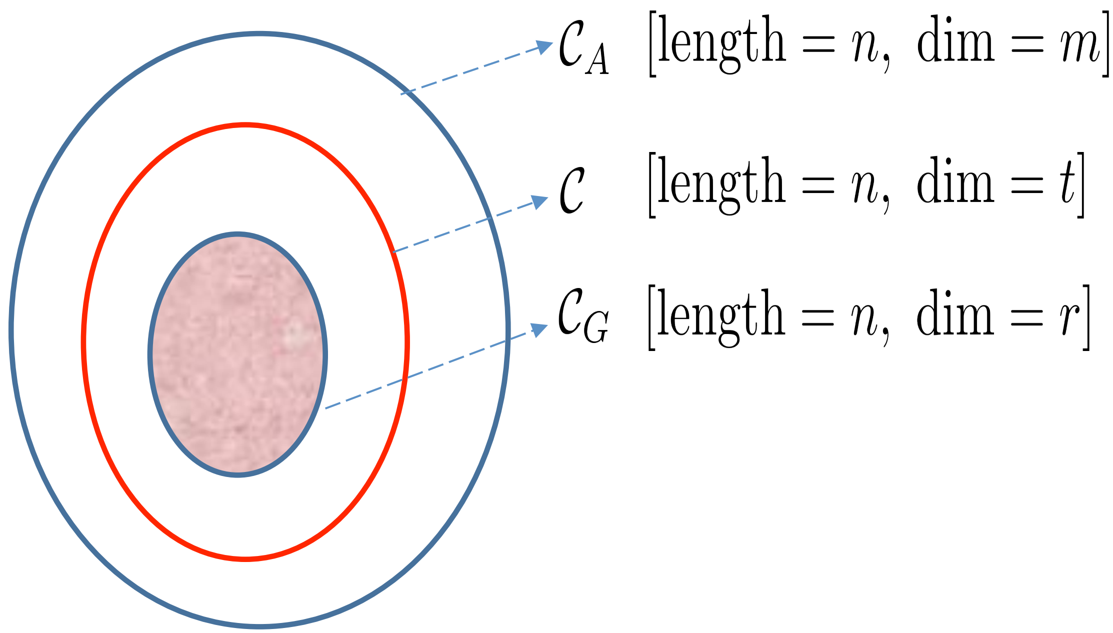
Broadcast : Approach for General Case



- Pick $\mathcal{C}_A \cap \mathcal{C}_H$ $2k$ as a $\mathcal{C}_A$ - MRSC of
- Pick $\mathcal{C}_B \cap \mathcal{C}_H$ $2k$ as a $\mathcal{C}_B$ - MRSC of
- "Maximize" $\mathcal{C}_H \cap \mathcal{C}_A \cap \mathcal{C}_B$ - we benefit from broadcasting

- Closed form expression for the optimal communication cost can be given

Broadcast: Connection to MRC



Given $\mathcal{C}_A$ $\mathcal{C}_G$ and $\mathcal{C}$, can you construct a maximally recoverable subcode ?

- Necessary Regularity Condition for “sandwiched” MRSC (straightforward) :

$$\operatorname{rank}\left(A|_S\right) = t \implies \operatorname{rank}\left(G|_S\right) = r, \quad \forall S, |S| = t$$

Code Constructions

	Purpose	Field Size	Comments
1	Point-to-Point, A corresponds to stripes of any linear code	$m = \mathrm{rank}(A)$ $q^{m-r},\ A \in F_q^{m \times n}$	**Partial Maximum Distance Separable** codes where local codes are scaled repetition codes
2	Broadcast - "Sandwiched" MRSC, any A and G		Based on Linearized Polynomials
3	A specific family of Partial MDS codes	Better than known constructions	Based on broadcast -"sandwiched" MRSC

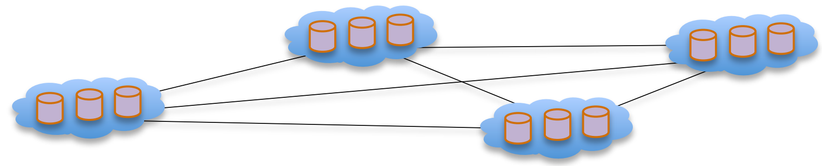
Overview

- Random linear network coding
- Distributed storage random linear network coding
- Coding in dynamic systems
- Coding for updating functions

Repair

8 segments (plus redundancy) in 4 clouds

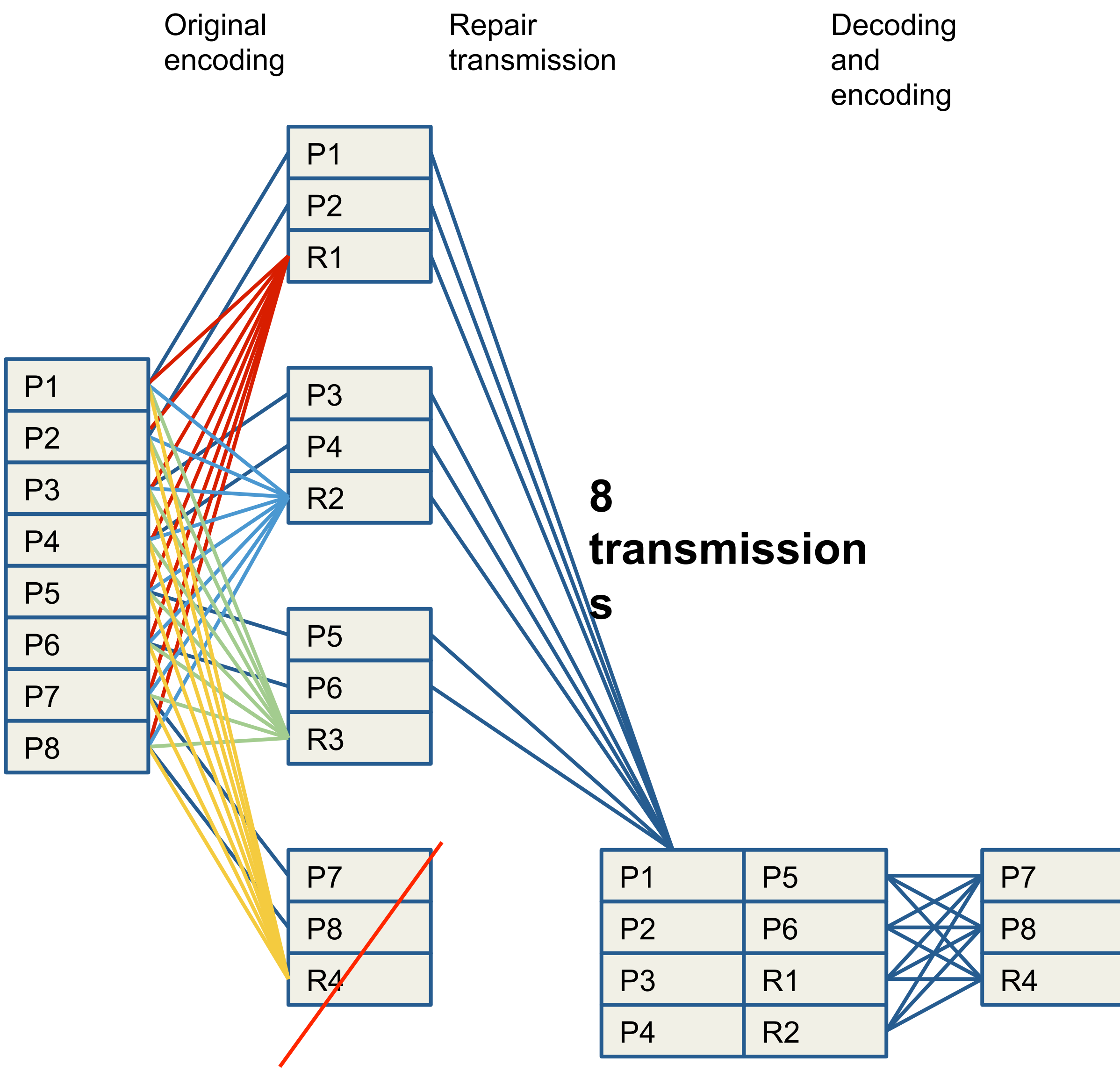
Example: 4 clouds with 3 disks (12 disk storage).



Coding Scheme	Disk Storage (less is better)	Inter (Intra) Cloud Bandwidth (less is better)	
		Cloud failure	Disk failure
RS 8:4	12	8	6
XORBAS 8:4:2	16	8	0(1)
RLNC v1a 8:4 systematic	12	6	2
RLNC v1b 8:8 systematic	16	4	1
RLNC v2 dense	12	3	1

- Conclusion: RLNC approaches will reduce the traffic at comparable storage situations.
- Staircase/LDPC need significant storage - **unable even to reach 16 in storage**

Reed-Solomon - RS (8,4)



Each storage unit holds some original pieces and a redundancy piece, which is coded from all the original pieces

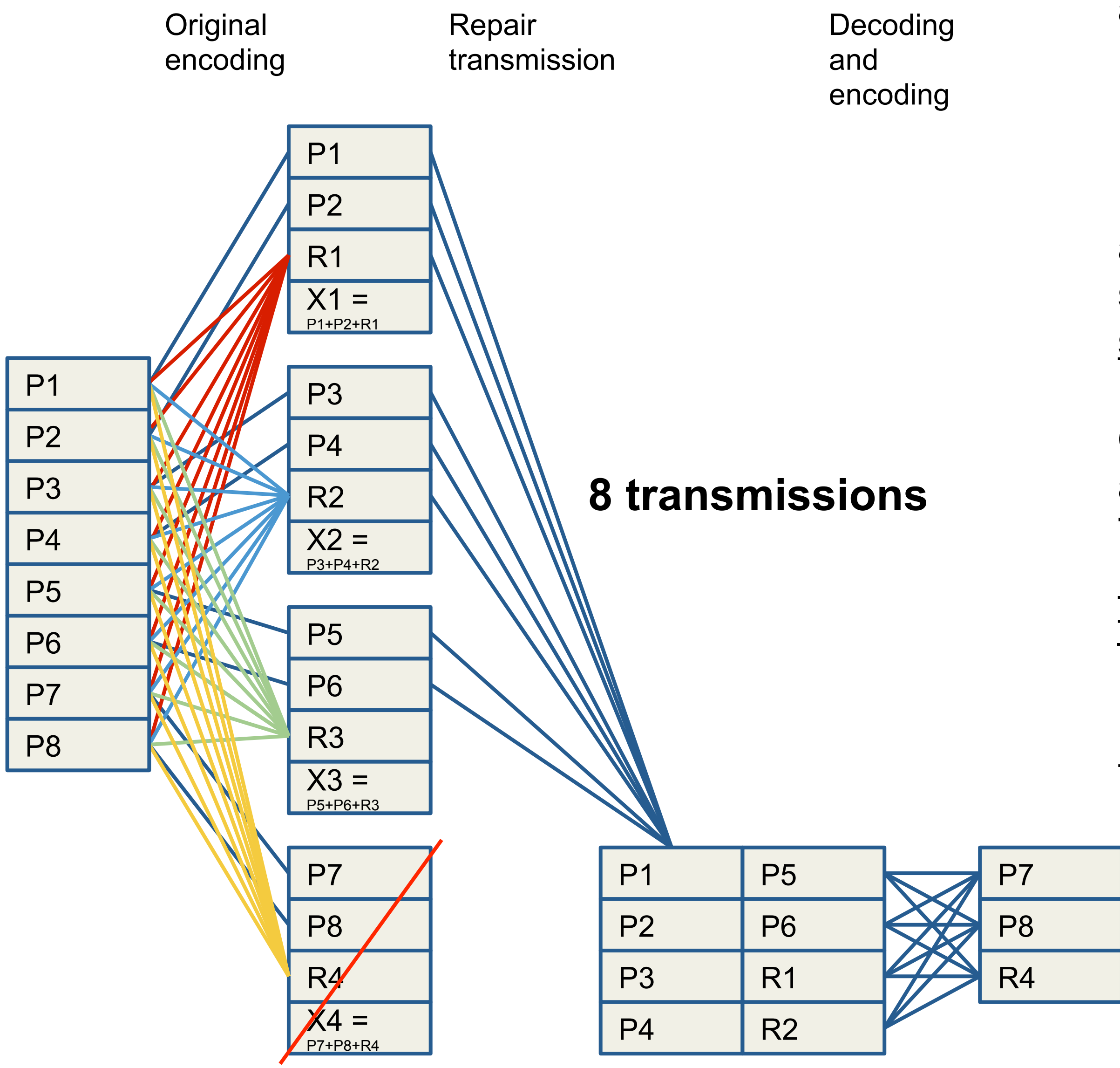
Recovery from unit failure:

1. The substitution node receives enough pieces to decode the original data.
2. The data is decoded.
3. The lost redundancy block is encoded.

Recovering from a unit loss requires complete decoding of all data.

R1 = P1+P2+P3+P4 +P5+P6+P7+P 8	R2 = P1+P2+P3+P4 +P5+P6+P7+P 8
R3 = P1+P2+P3+P4 +P5+P6+P7+P 8	R4 = P1+P2+P3+P4 +P5+P6+P7+P 8

XORBAS - like (8,4,3)

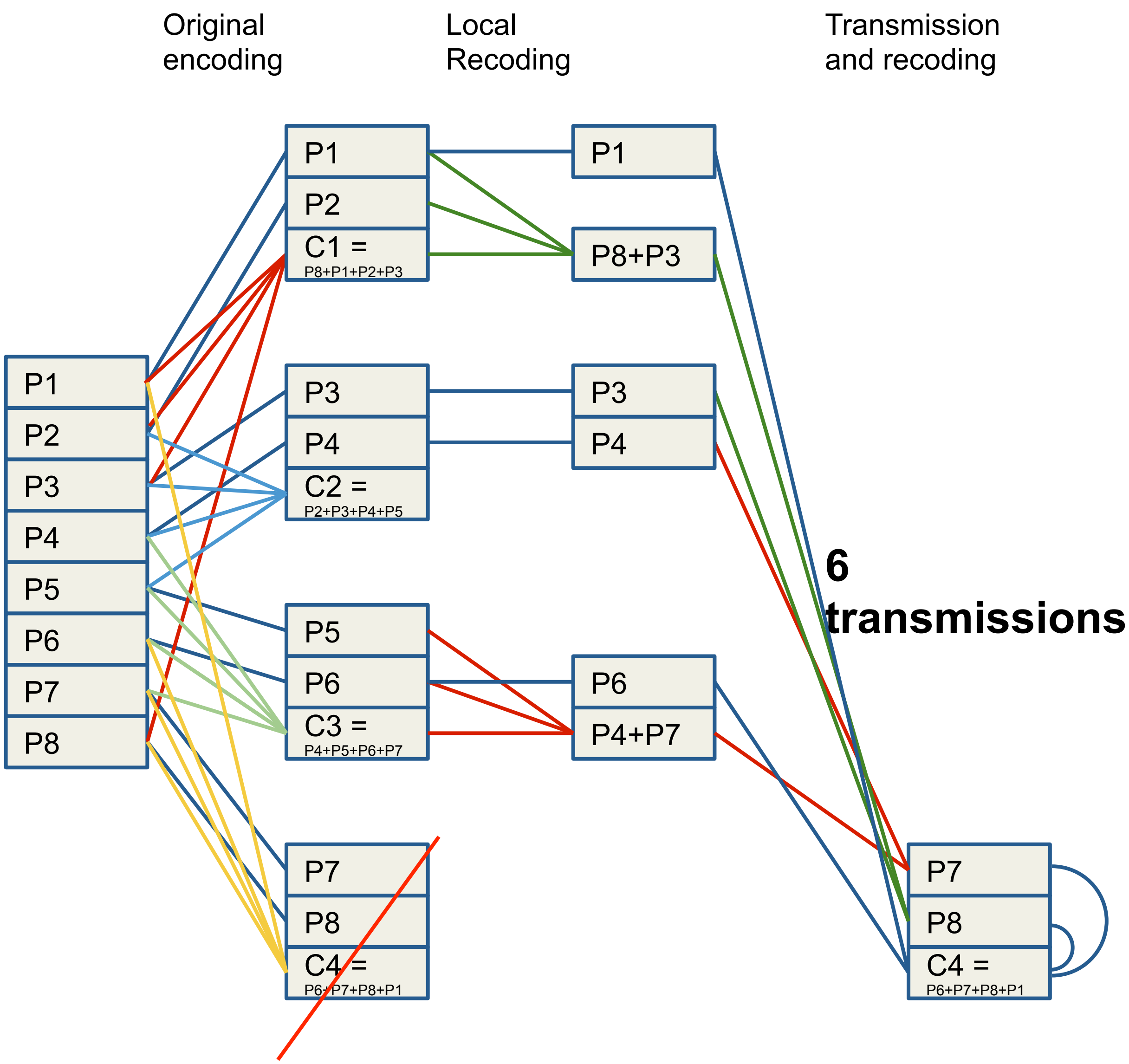


Each storage unit holds, In addition to original and redundancy pieces, a local redundancy block.

By adding local redundancy at the cost of additional spent storage, recovery from single block failures requires no transmissions. This "trick" can be applied to other approaches.
This enables all units to recover from a single block failure locally, i.e., within the unit.
For a unit failure, the cost is the same as for RS.

R1 = P1+P2+P3+P4+P5+P6+P7+P8	R2 = P1+P2+P3+P4+P5+P6+P7+P8
R3 = P1+P2+P3+P4+P5+P6+P7+P8	R4 = P1+P2+P3+P4+P5+P6+P7+P8

Perpetual-RLNC (8,4)



Each storage unit holds a perpetually coded block, which is a combination of a subset of the original pieces.

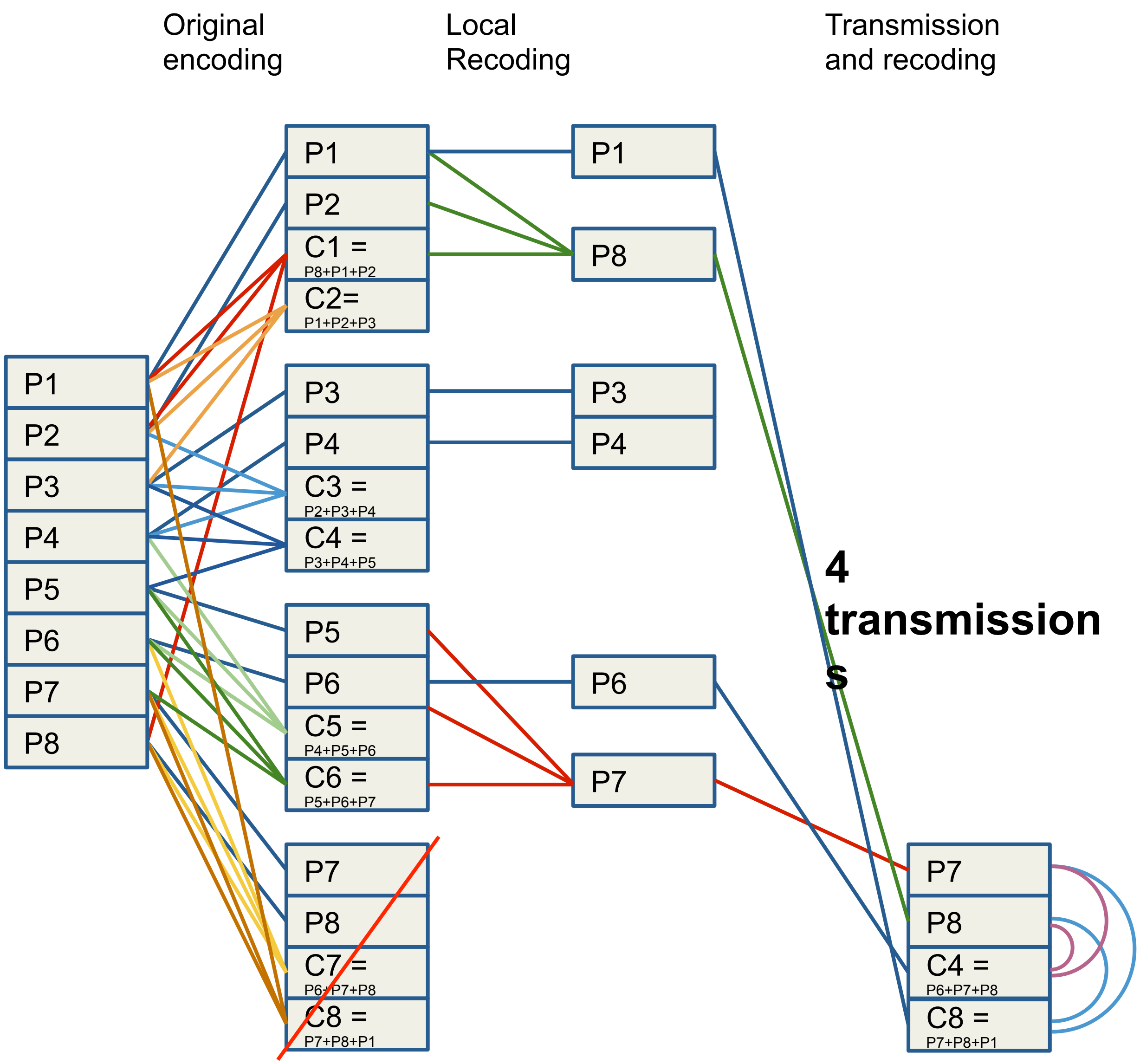
Recovery from unit failure:

1. The remaining units perform recoding to obtain the most useful pieces for the substitution unit
2. The resulting pieces are transmitted
3. The lost original pieces are decoded.
4. The lost redundancy block is encoded.

By adding an extra coding step at the sending units, the number of transmissions are reduced and the coding performed at the substitution node simplified.

Perpetual-RLNC* (8,8)

*Random Linear Network Coding



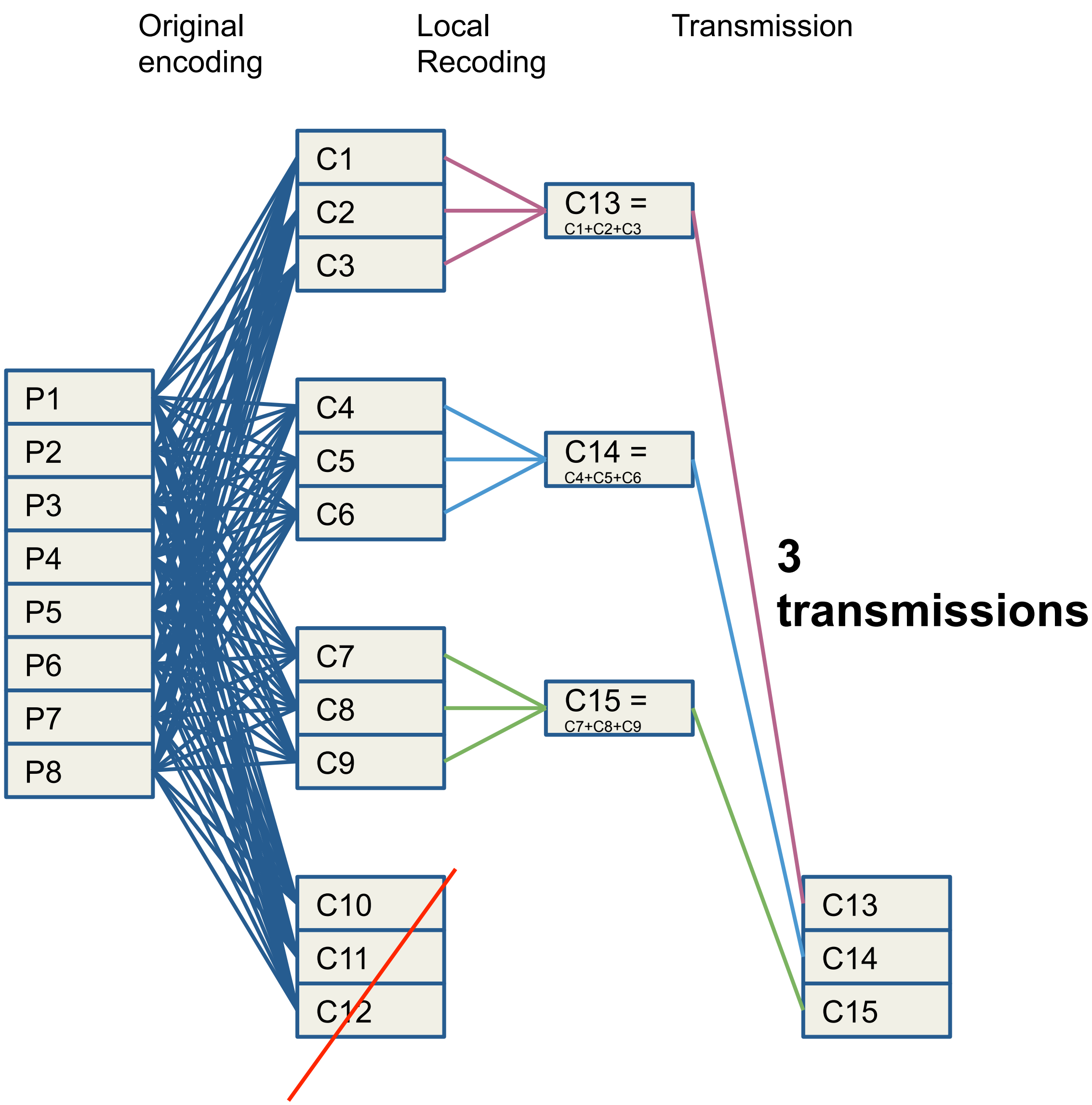
Extra storage can also be spent on decreasing the cost of unit failure repair.
Each storage unit holds two perpetually coded blocks.
This example considers a smaller subset of original pieces in each coded packet.

Recovery from unit failure:

1. Remaining units perform recoding to obtain pieces for the substitution unit
2. The resulting pieces are transmitted
3. The lost redundancy block is encoded

By utilizing additional storage at each storage unit the number of transmissions can be further reduced.

RLNC* (0,12)



So far we have considered exact repair if we accept functional repair we can apply RLNC. With RLNC all stored pieces are combinations of all original pieces.

Recovery from unit failure:

1. The remaining units perform uncoordinated recoding combining all pieces they hold.
2. The resulting pieces are transmitted

By utilizing RLNC the number of transmissions is further reduced and the need for coding at the substitution node removed.

C1 = P1+P2+P3+P4 +P5+P6+P7+P 8	C2 = P1+P2+P3+P4 +P5+P6+P7+P 8	C3 = P1+P2+P3+P4 +P5+P6+P7+P 8	…