Preserving data

Sébastien Ponce
sebastien.ponce@cern.ch

CERN

Thematic CERN School of Computing 2015

In the previous episodes...

- We've discussed data formats and addressing
- We've talked about finding our data
- We've found out how to store it efficiently

In the previous episodes...

- We've discussed data formats and addressing
- We've talked about finding our data
- We've found out how to store it efficiently

Today

Let's make sure we do not lose or corrupt our nice data !

Outline

1. Risks of data loss and corruption
2. Data consistency
   - Checksums
   - Block checksums
3. Data safety
   - Redundancy
   - Parity
   - Erasure coding
4. Conclusion

Risks of data loss and corruption

1 Risks of data loss and corruption
2 Data consistency
3 Data safety
4 Conclusion

Risks for my data - Hardware

some numbers for disks

- probability of losing a disk per year : few %, up to 10%
  - with 60K disks, it's around 10 per day
  - and all files are lost
- one unrecoverable bit error in 10^{14} bits read/written
  - for 1GB files, that's one file corrupted per 10K files written

some numbers for tapes

- probability of losing a tape per year : 10^{-4}
  - and you recover most of the data on it
  - net result is 10^{-7} file loss per year
- one unrecoverable bit error in 10^{19} bits read/written
  - for 1GB files, that's one file corrupted per 1G files written

Risks for my data - Software

BUGS !

- in your software
  - e.g. scheduling twice a transfer, not receiving data on the second run and overwriting the correct file with an empty one
- in your dependencies
  - e.g. the transfer protocol used does not support checksum and data may be corrupted by TCP (checksum is only 16 bit, one corrupted packet in 65536 will go through)
- in the OS or common libraries
  - e.g. libc locks not being atomic
- in the hardware - that is in the micro code running inside
  - e.g. RAID controllers
- in your admin tools
  - e.g. recycling a tape that was not empty

Risks for my data - Human factor

Real life cases that went wrong

- reinstall (and wipe) old machine p23425a4752
  - Oh no, I actually meant p42532a8779... bad cut and paste
- rm -rf /top/data/alltimes /2015/04/crap
  - one space too much and all data are gone....
- activate garbage collection on pool XYZ, it's full
  - wasn't it tape backed up ? no ? oups....

Risks for my data - conclusion

Risks for my data - conclusion

You will lose/corrupt data !

Risks for my data - conclusion

You will lose/corrupt data !

- better to be able to know when and what
- even better if you can repair

Data consistency

1 Risks of data loss and corruption

2 Data consistency
- Checksums
- Block checksums

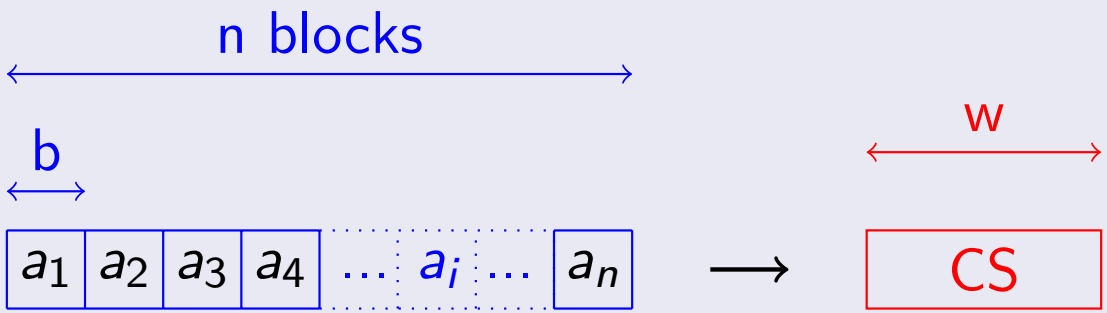
3 Data safety

4 Conclusion

Checksum

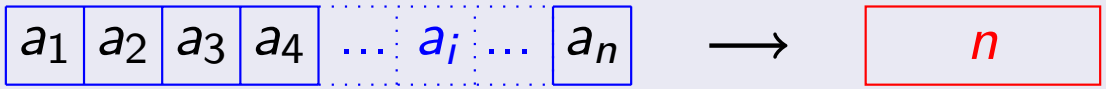
Definition

"small-size datum from a block of digital data for the purpose of detecting errors"



Most basic checksum : data size

Computation



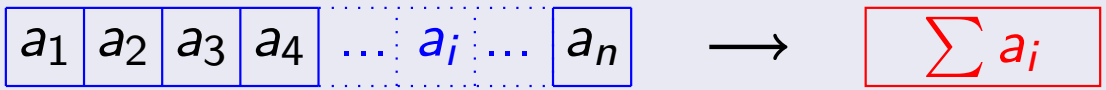
$$b = 8\,\text{bit} \quad w = 64\,\text{bit} \quad CS = n$$

Pros and Contra

- easy to compute
- detects erasures and additions
- does not detect any corruption

Basic checksum : sum/xor

Computation



$$b \quad w = b \quad CS = \sum_{i=1}^{n} a_i$$

Pros and Contra

- easy to compute
- detects most corruptions
- does not detect any inversions/change of order

Adler like checksums

Computation

$$b = 8\,\text{bit} \quad w = 32\,\text{bit} \quad CS_{high} = \sum_{i=1}^{n} a_i \quad CS_{low} = \sum_{i=1}^{n} i a_i$$

Pros and Contra

- easy to compute
- detects most corruptions and inversions
- weak for small files
- easy to fake in case of intentional corruption

(Crypt)Analysis of adler

Weaknesses

- 32 bits is short
  - one per 4 billion corruption will go through
- it's actually worse for small files
  - all bits of the sum are not even used for less than 256 bytes
- they can be easily bypassed
  - one can easily change the last 16 bytes and reach any checksum
  - so intentional corruptions are not covered

Cryptographic checksums

What is it ?

- checksums that cannot be faked (easily)
- they are based on non reversible cryptographic functions

Most used ones

md5	1991, 128 bits, by Rivest. Not considered secure anymore as complete collisions have been discovered.
sha1	1995, 160 bits, by NSA. Collision in 2^{61} operations
sha256	2001, 256 bits, by NSA. Collision in 2^{128} operations
sha512	2001, 512 bits, by NSA. Collision in 2^{256} operations

Drawback

- more costful to compute
- although modern processors have dedicated instructions

Comparison of main checksums

Name	MB/s on intel core 2	Cycles Per Byte
Adler32	920	1.9
MD5	255	6.8
SHA-1	153	11.4
SHA-256	111	15.8
SHA-512	99	17.7

Practical usage of checksums

Simple approach

- compute checksum in memory when creating/writing file
- store checksum in a DB
- check it in memory on full file reads

Problems

- corrupted data only found when read back, unnoticed otherwise
- one needs to fully read the file to be able to check
- file updates not supported. Need to read back the whole file
- file append suffers the same limitation
- multi stream, out of order writing not supported
- losing the DB loses all checksums
- renaming a file implies changing the entry in the DB, with double commit issue

Recommendations

When to compute checksums

- opportunistically
- plus regular scans of the whole data set

How to store checksums

- always close to the file
- next to it on the filesystem
- in most cases, using external attributes

Block checksumming

One checksum per file piece

- split the file into smaller pieces
- compute on checksum per piece

Advantages

- updates only need recompilation of modified blocks
- adding blocks is trivial
- concurrent write to different blocks can be handled

Disadvantages

- the metadata management becomes complex

Combining checksums

Idea

- allow to combine checksums of subparts of a file
- allows to handle updates (if block checksums is used)
- allows easy support of parallel writing into a file

Combining checksums

Idea

- allow to combine checksums of subparts of a file
- allows to handle updates (if block checksums is used)
- allows easy support of parallel writing into a file

- supported by trivial checksums (size, xor) and adler32

- not supported by strong checksums obviously

The adler case

$$Adler_{1,n} = (A_{1,n}, B_{1,n}) \qquad with \qquad A_{1,n} = \sum_{i=1}^{n} a_i \qquad B_{1,n} = \sum_{i=1}^{n} i a_i$$

The adler case

$$Adler_{1,n} = (A_{1,n}, B_{1,n}) \qquad with \qquad A_{1,n} = \sum_{i=1}^{n} a_i \qquad B_{1,n} = \sum_{i=1}^{n} i a_i$$

Cutting at p, we have :

$$A_{1,n} = A_{1,p} + A_{p+1,n} \qquad B_{1,n} = \sum_{i=1}^{n} i a_i$$

$$= B_{1,p} + \sum_{i=p+1}^{n} (i-p) a_i + p \sum_{i=p+1}^{n} a_i$$

$$= B_{1,p} + B_{p+1,n} + p\ A_{p+1,n}$$

The adler case

$$Adler_{1,n} = (A_{1,n}, B_{1,n}) \qquad with \qquad A_{1,n} = \sum_{i=1}^{n} a_i \qquad B_{1,n} = \sum_{i=1}^{n} i a_i$$

Cutting at p, we have :

$$A_{1,n} = A_{1,p} + A_{p+1,n} \qquad B_{1,n} = \sum_{i=1}^{n} i a_i$$

$$= B_{1,p} + \sum_{i=p+1}^{n} (i-p) a_i + p \sum_{i=p+1}^{n} a_i$$

$$= B_{1,p} + B_{p+1,n} + p\ A_{p+1,n}$$

Practically

- you need the length on top of the adler32 checksum

Data safety

1 Risks of data loss and corruption

2 Data consistency

3 Data safety
- Redundancy
- Parity
- Erasure coding

4 Conclusion

How to correct corrupted data ?

So far

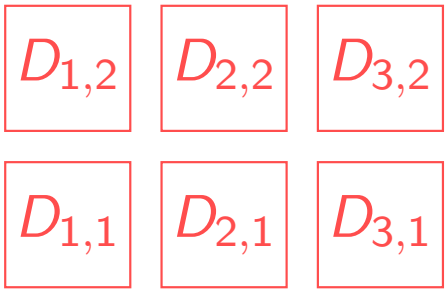
- checksum allow corruption detection
- but no correction is possible
- erasure of data is not covered either
- for this we need some data redundancy

Existing techniques

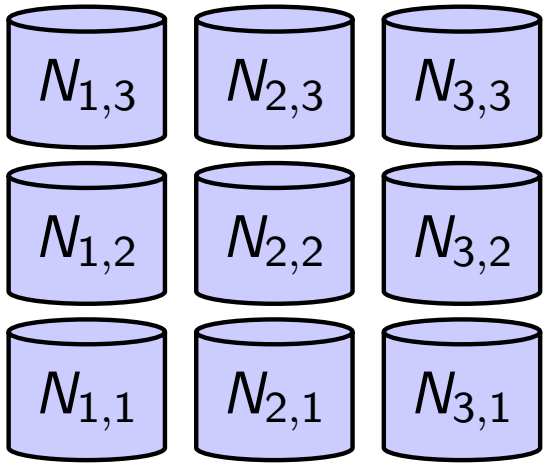
- mirroring
- parity and RAID systems
- erasure coding

Context

k pieces of data to be stored



n nodes

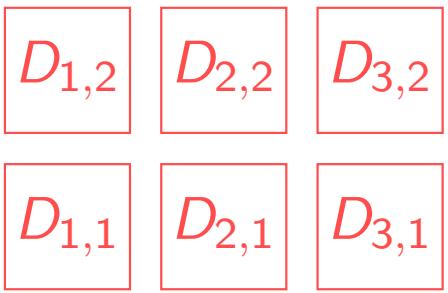


Goal

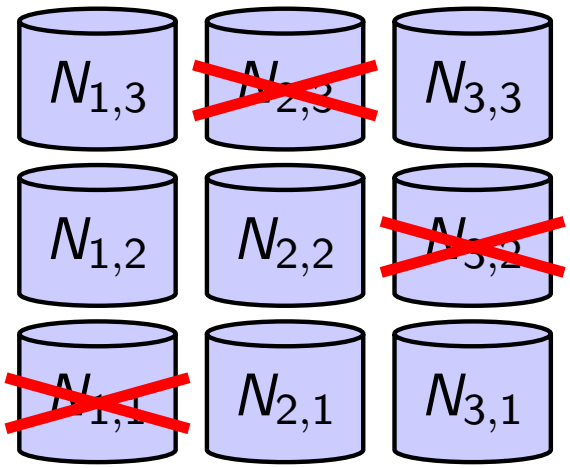
- some nodes will fail/disappear
- we want our data back anyway
- we want the cheapest price

Context

k pieces of data to be stored



n nodes



Goal

- some nodes will fail/disappear
- we want our data back anyway
- we want the cheapest price

Replication

Idea

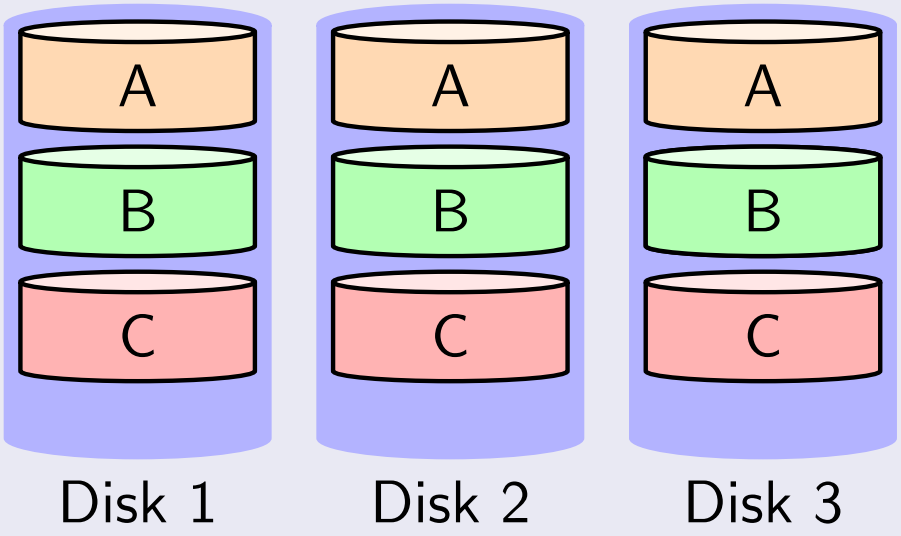
- replicate each piece on data on p disks
- you can afford losing $p - 1$ disks
- you pay p times the original price

Replication

Idea

- replicate each piece on data on p disks
- you can afford losing $p - 1$ disks
- you pay p times the original price

Replication - RAID 1

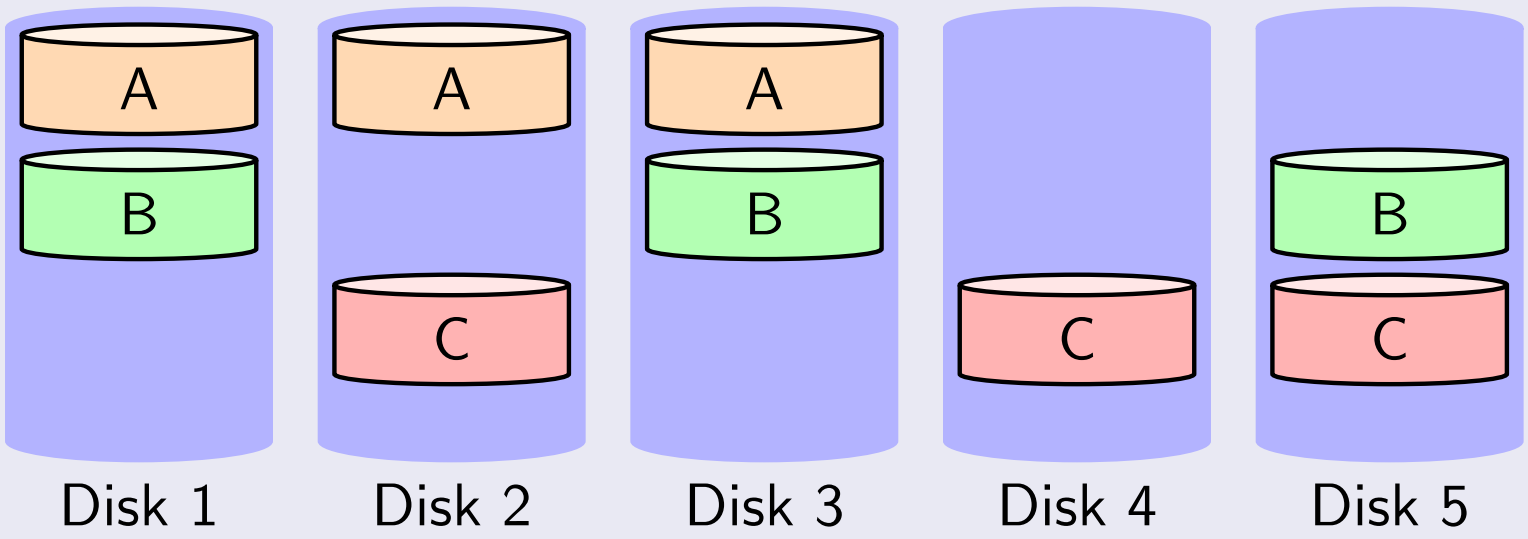


Replication

Idea

- replicate each piece on data on p disks
- you can afford losing $p - 1$ disks
- you pay p times the original price

Replication - RAID 1



Limitations of replication

With 2 replicas

- expensive : effective disk space divided by 2
- when corruption occur, no way to know which copy is corrupted
  - unless you have checksums on top

With 3 and more replicas

- horribly expensive, actually unaffordable

Parity

Idea

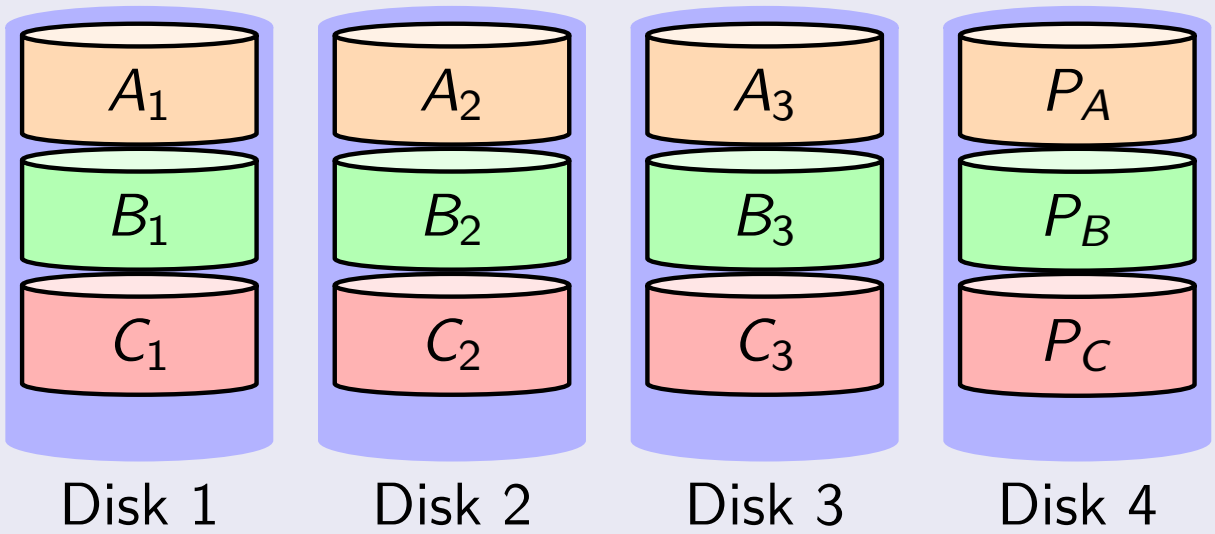
- create a parity piece of data for each k pieces (xor)
- store it as an extra piece of data
- you can afford losing 1 disk
- you pay $1 + \frac{1}{k}$ times the original price

Parity

Idea

- create a parity piece of data for each k pieces (xor)
- store it as an extra piece of data
- you can afford losing 1 disk
- you pay $1 + \frac{1}{k}$ times the original price

Parity Node - RAID 4

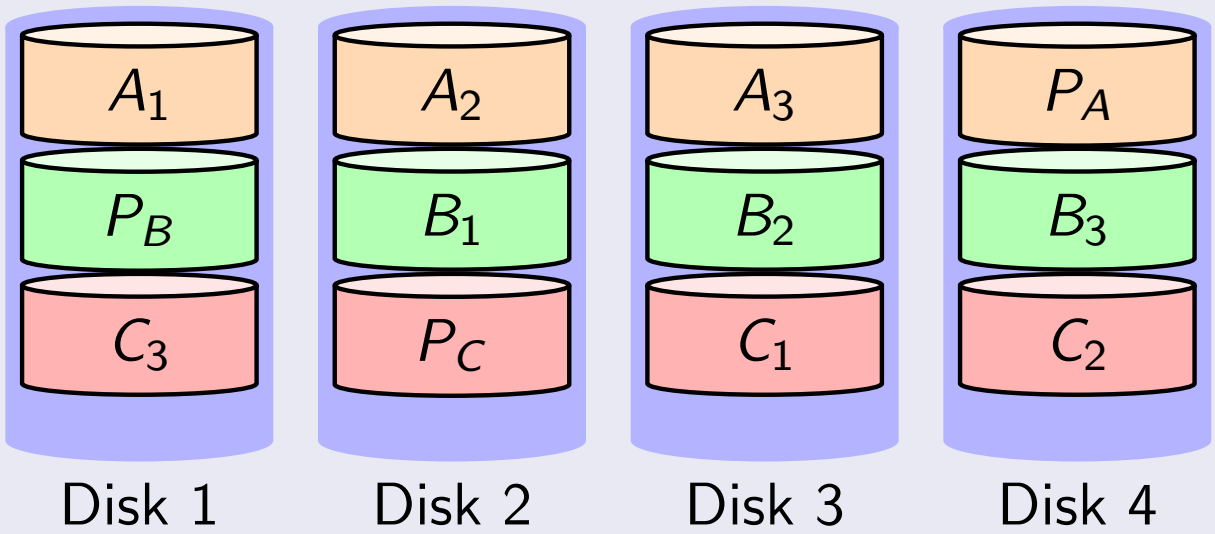


Parity

Idea

- create a parity piece of data for each k pieces (xor)
- store it as an extra piece of data
- you can afford losing 1 disk
- you pay $1 + \frac{1}{k}$ times the original price

Parity Node - RAID 5



Main issues

With 2 replicas

- writes are slightly more costful
- small updates are costful
  - one has to read back the old data to recompute parity
- only one corruption/erasure allowed
  - take care that losing a disk drastically increases the probability to lose another one
  - especially in RAID due to locality. RAIN is much better from this point of view
  - reconstructing the lost disk can take days
    e.g. 6 TB at 100 MB s^{-1} is 17 hours

Double parity

Idea : same spirit

- compute 2 parity pieces of data for each k pieces

$$P = \sum D_i \qquad Q = \sum g^i D_i$$

- you can lose 2 disks and still recover (less easily though)
- you pay $1 + \frac{2}{k}$ times the original price

Double parity

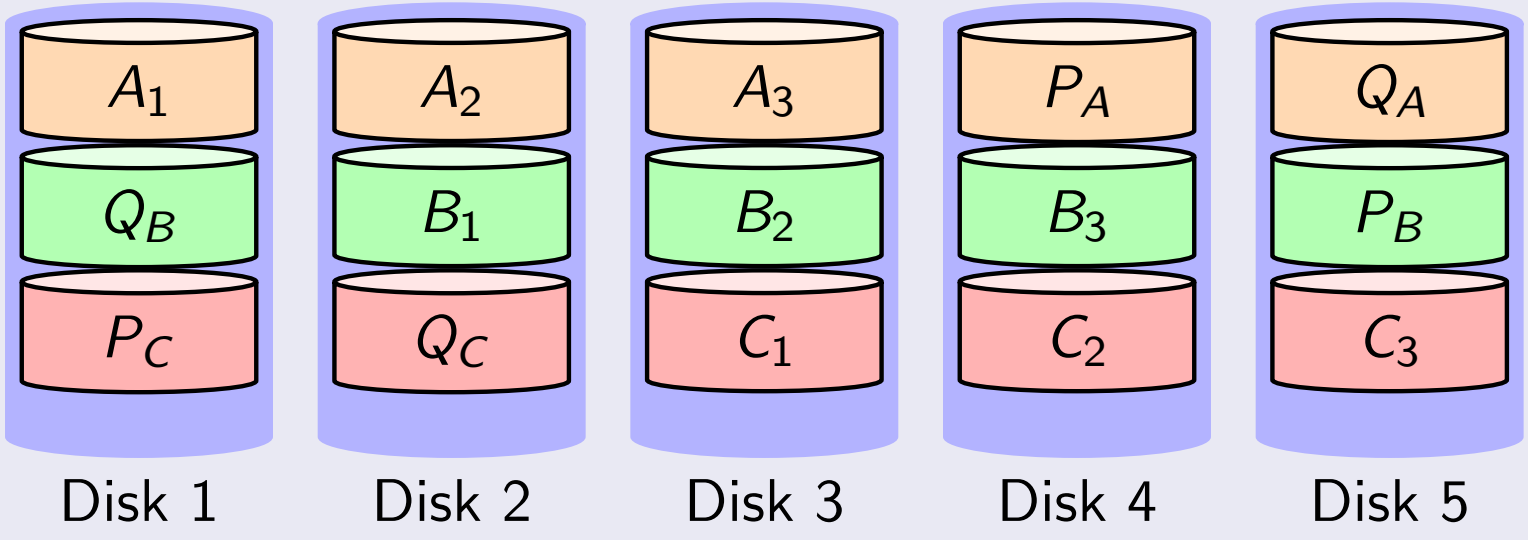
Idea : same spirit

- compute 2 parity pieces of data for each k pieces

$$P = \sum D_i \qquad Q = \sum g^i D_i$$

- you can lose 2 disks and still recover (less easily though)
- you pay $1 + \frac{2}{k}$ times the original price

Double parity - RAID 6



Cost vs Risk computations

Cost of parity

we call r is the ratio of disk space compared to data size
for k blocks with one parity, we have

$$r = 1 + \frac{1}{k}$$

Cost vs Risk computations

Cost of parity

we call r is the ratio of disk space compared to data size
for k blocks with one parity, we have

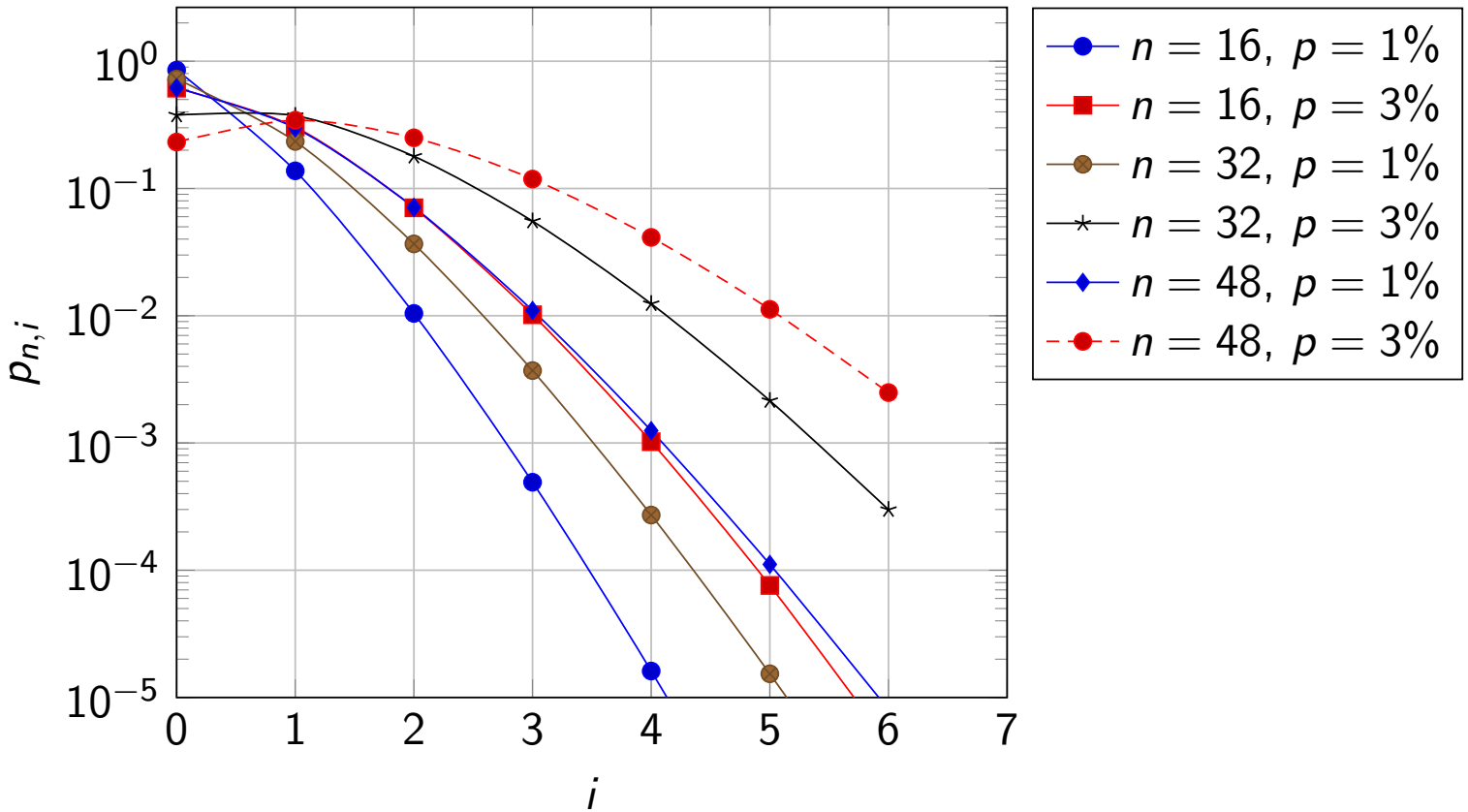
$$r = 1 + \frac{1}{k}$$

Risk reduction

- if p the probability to lose a given disk within a given period
- we call $p_{n,i}$ the probability of losing any i disks among n

$$p_{n,0} = (1-p)^n \tag{1}$$
$$p_{n,1} = np(1-p)^n \tag{2}$$
$$p_{n,i} = \binom{n}{i} p^i (1-p)^{n-i} \tag{3}$$

Graphically, for disks within a year



Generic case (erasure coding)

Ideas

- compute more "parities" : m for each k blocks
- work on r rows of k blocks at a time

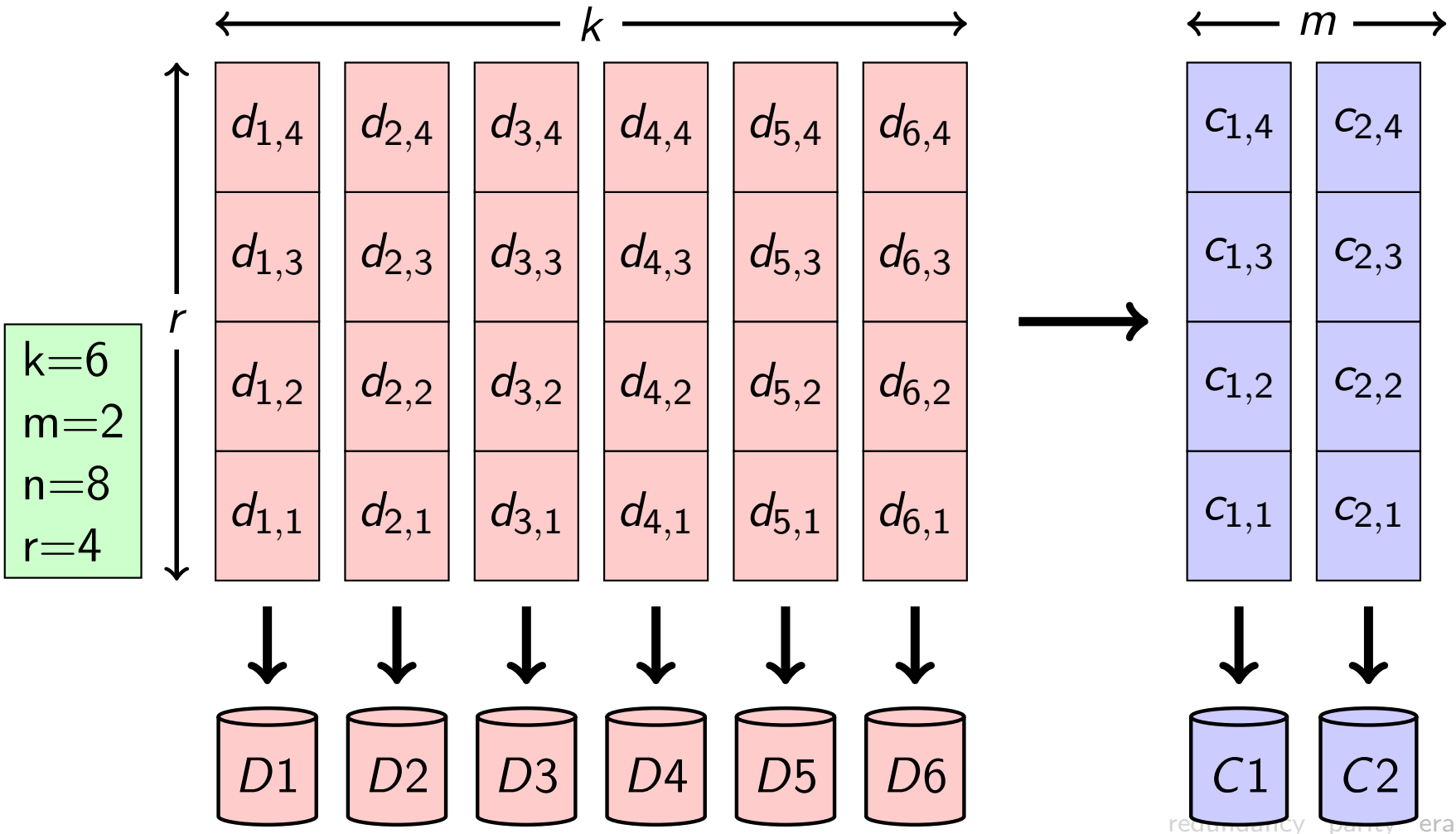
Goal

- be able to lose any m disks
- be able to reconstruct from any k disks

"Maximum Distance Separable" code (MDS)

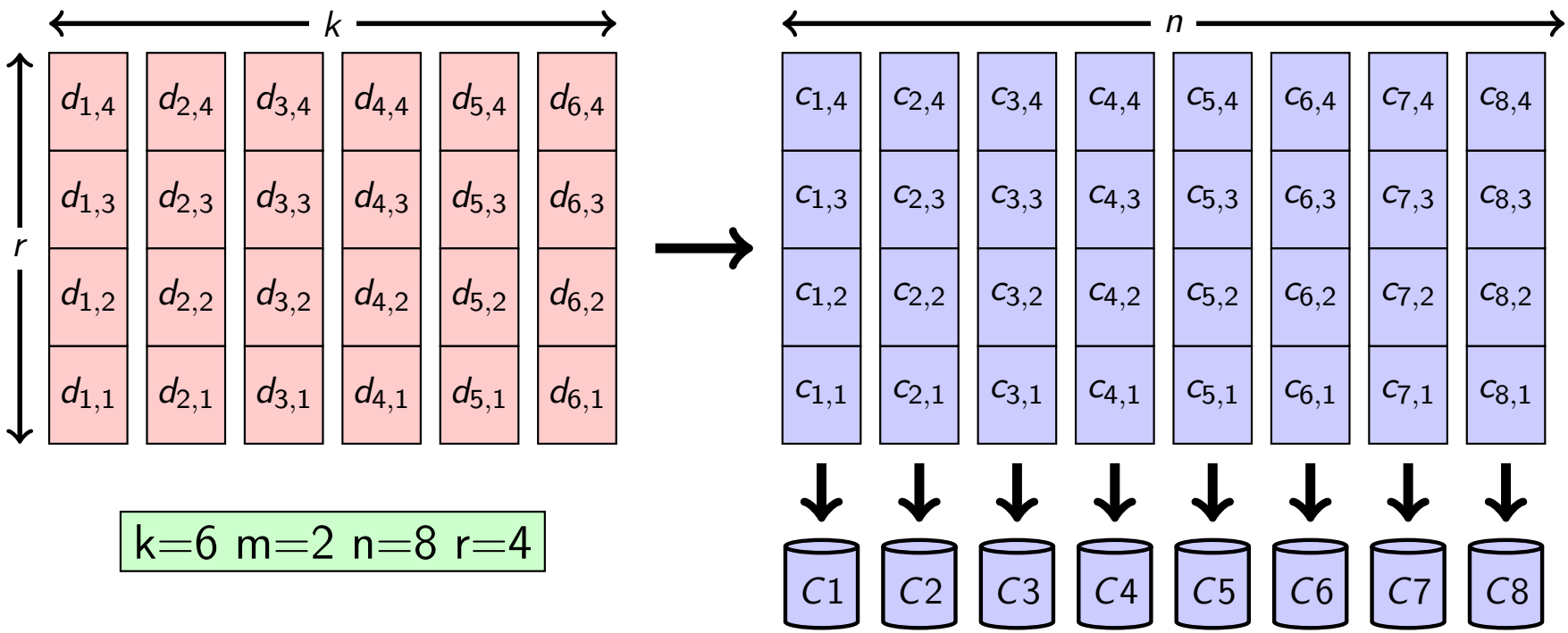
First example

Systematic code



More generic example

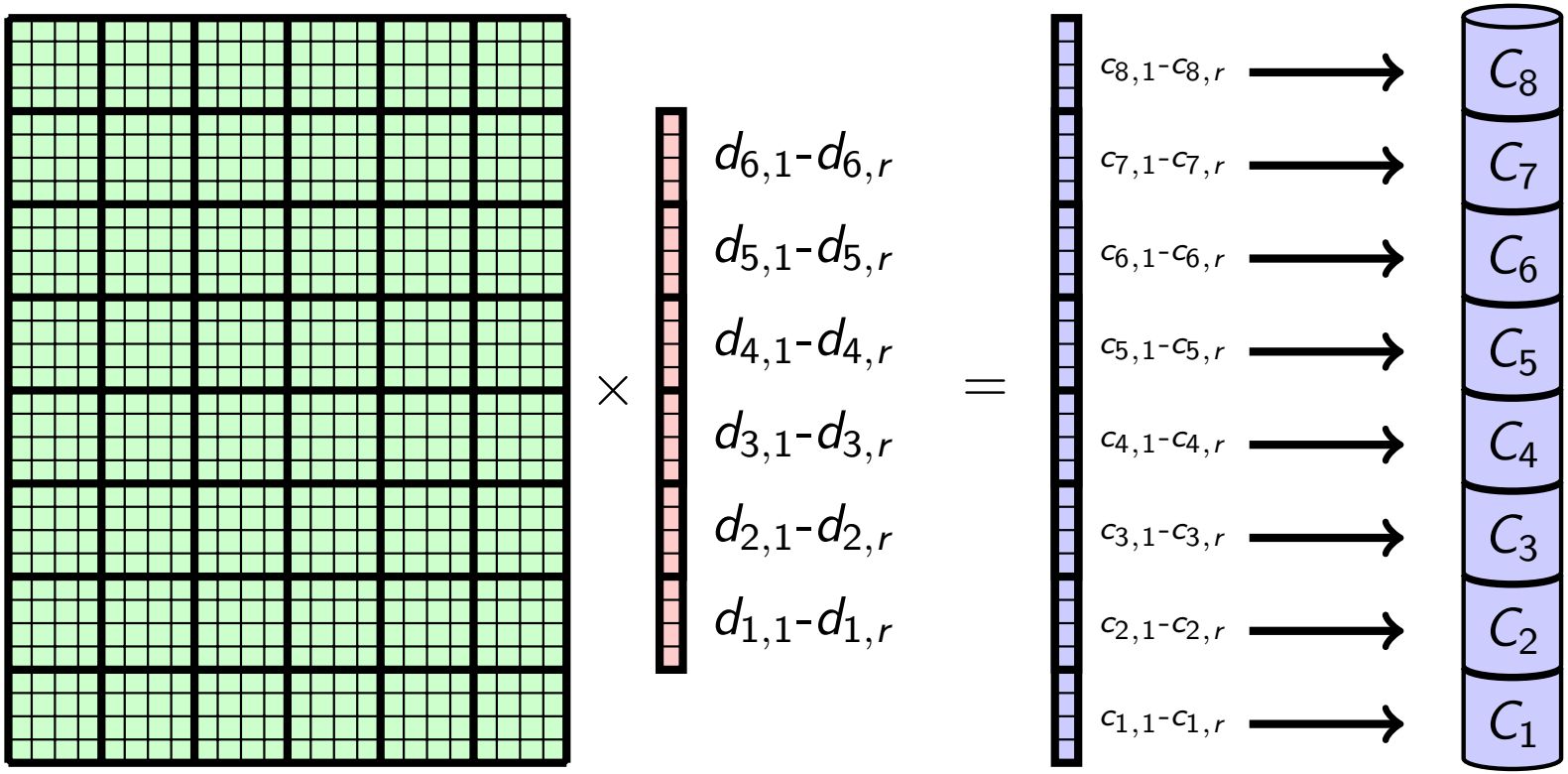
Non systematic code



k=6 m=2 n=8 r=4

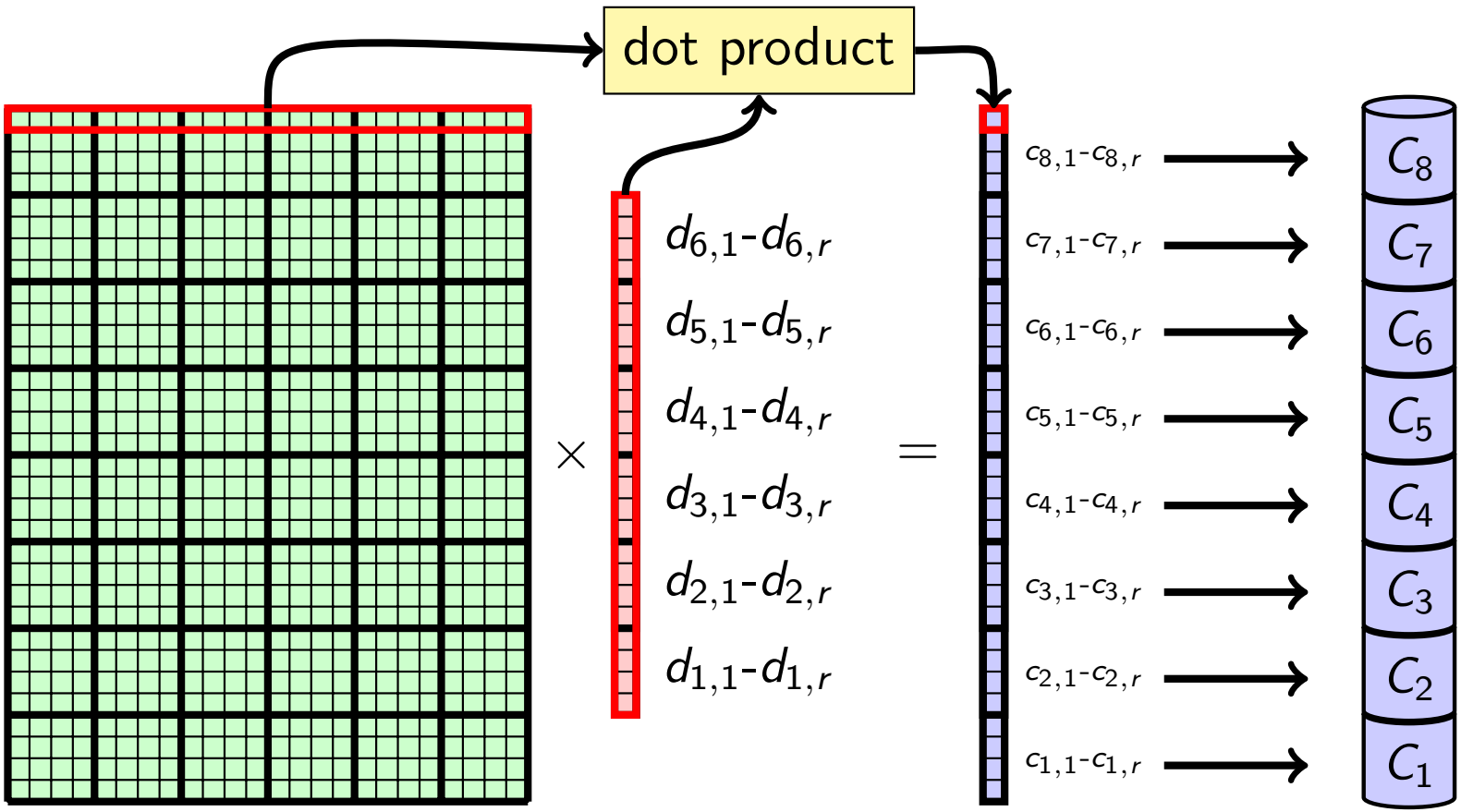
The matrix view

Generator Matrix G is nr x kr



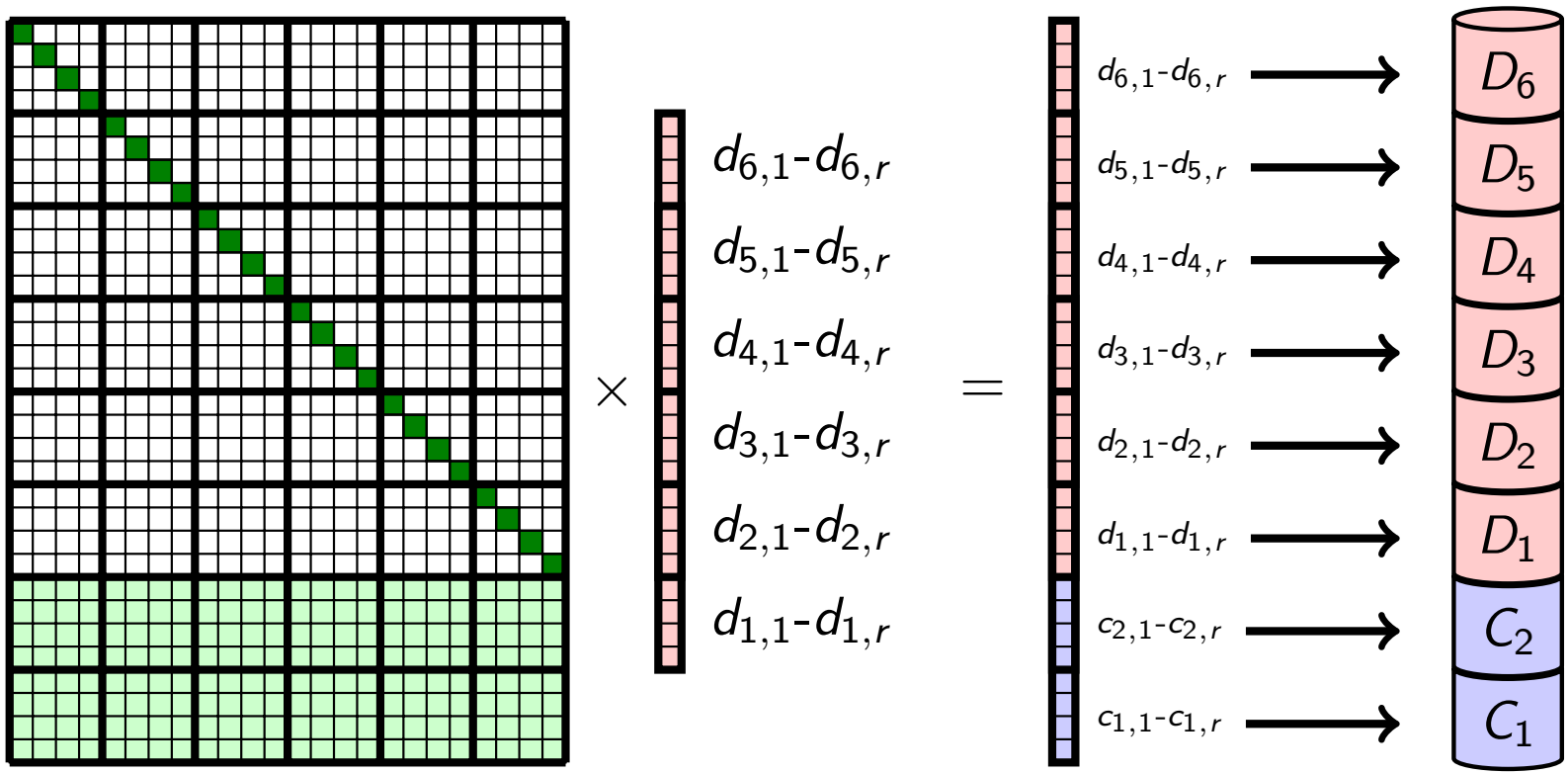
The matrix view

Generator Matrix G is nr x kr

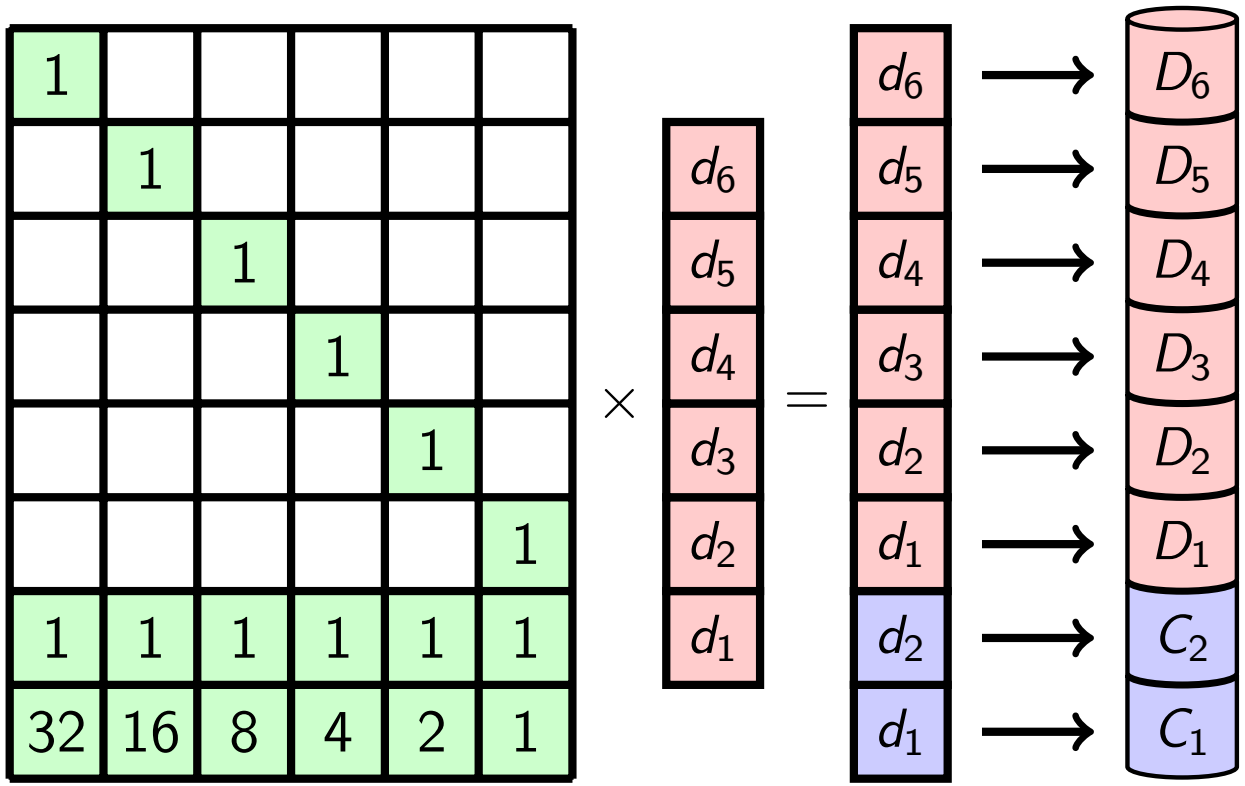


Back to systematic code

Systematic means G includes identity



Practical example : RAID 6



Systematic Reed-Solomon Codes

1					
	1				
		1			
			1		
				1	
					1
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,6}$
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,6}$
$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$

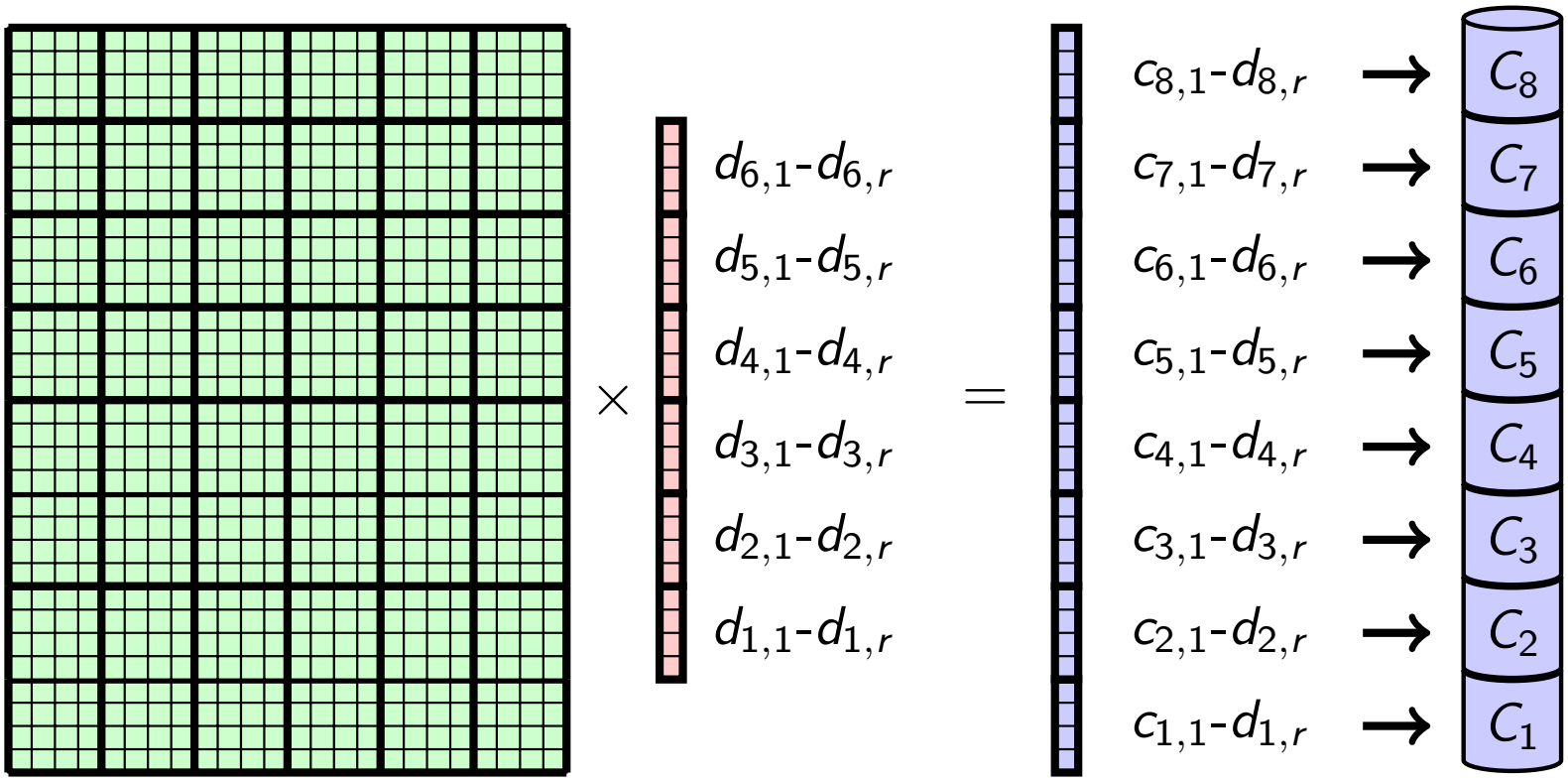
$k = 6$ and $n = 9$

- Create two sets X and Y
- X has m elements: x_0 to x_{m-1}
- Y has k elements: y_0 to y_{k-1}
- Elements in $X \cup Y$ are distinct
- $a_{i,j} = \frac{1}{x_i + y_j}$ in $GF(2^w)$ (Galois Field)

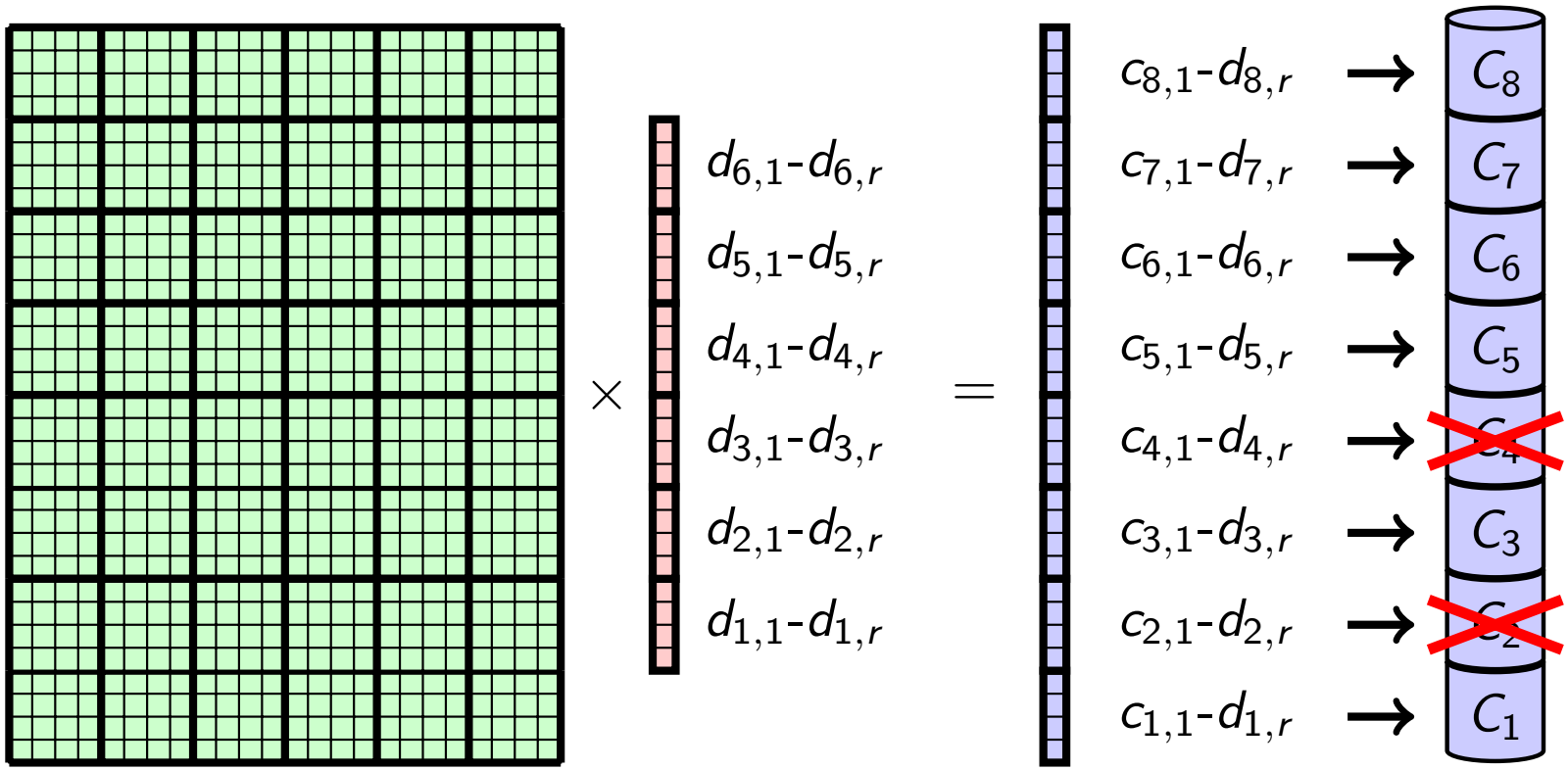
$X = 1, 2, 3$

$Y = 4, 5, 6, 7, 8, 9$

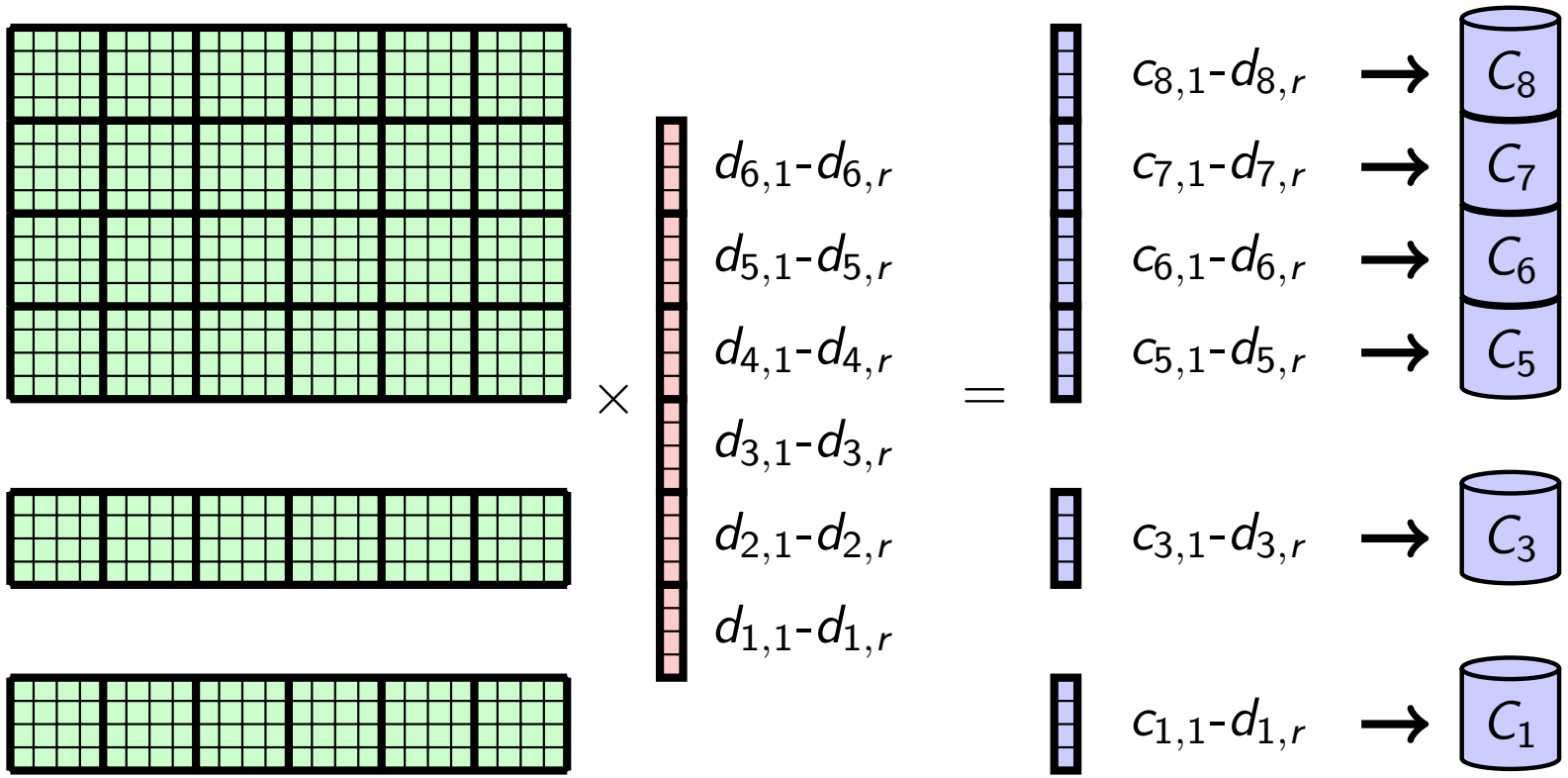
Generic reconstruction



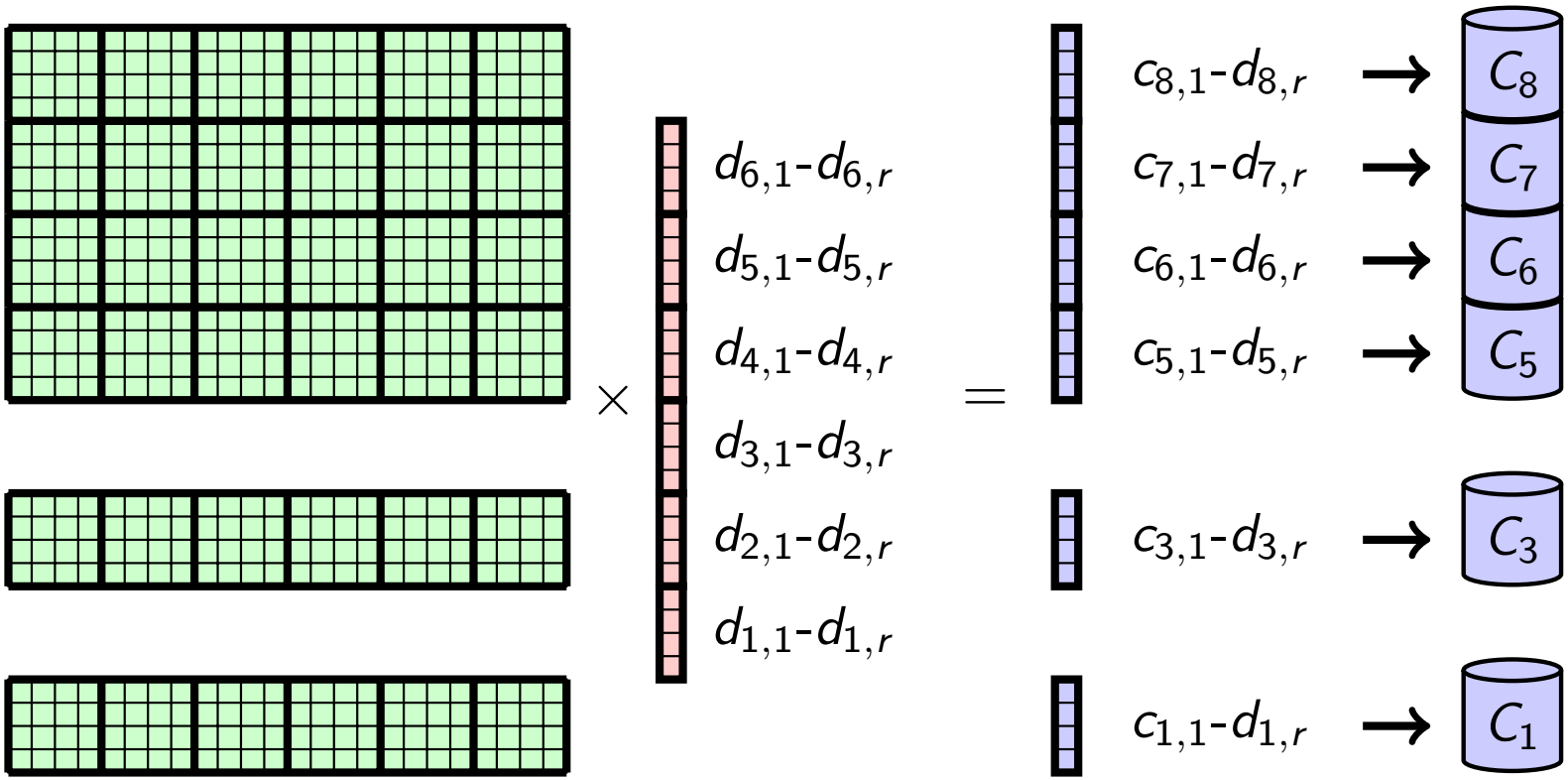
Generic reconstruction



Generic reconstruction

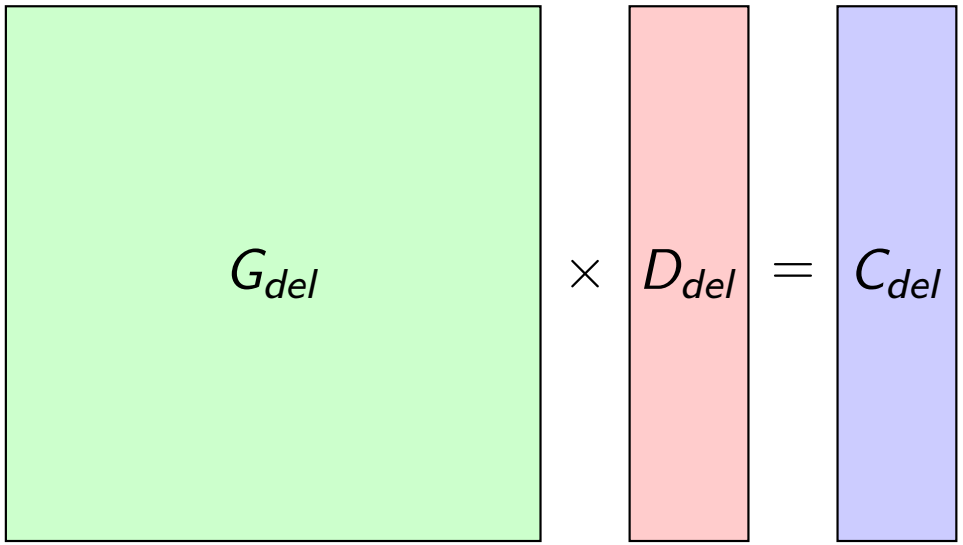


Generic reconstruction

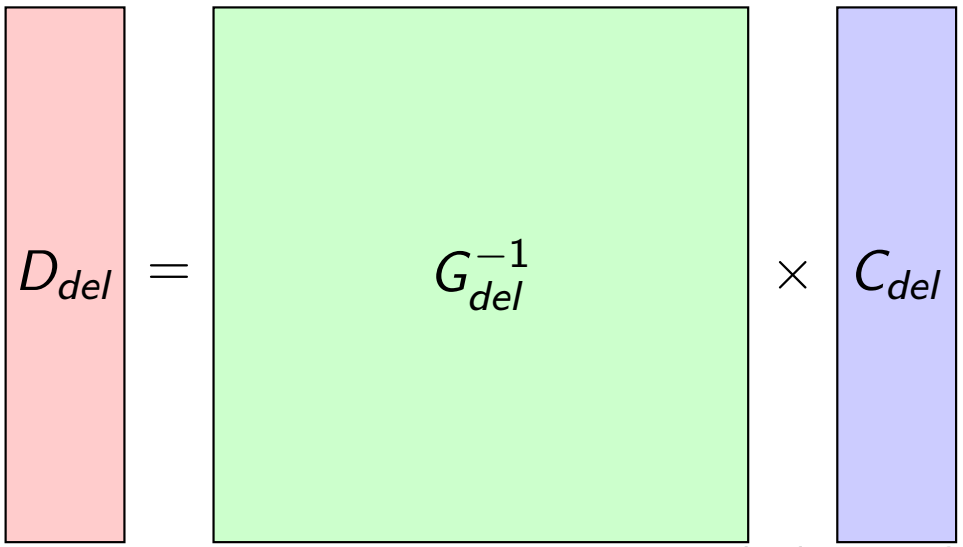


$$G_{del} \times D_{del} = C_{del}$$

Generic reconstruction



"Simple" matrix inversion



Reconstruction cost

Important parameters

- systematic codes are very interesting for no error case
- parity is very cheap for a single erasure/corruption : simple xor
- galois fields are more costful, especially when m grows

Reconstruction cost

Important parameters

- systematic codes are very interesting for no error case
- parity is very cheap for a single erasure/corruption : simple xor
- galois fields are more costful, especially when m grows

Improvements are possible

- pyramid codes bring improvements of reconstruction time of errors in different subparts
- Cauchy Reed-Solomon : replaces Galois fields with pure xors
- evenodd, RDP : fast single error recovery

Conclusion

1 Risks of data loss and corruption
2 Data consistency
3 Data safety
4 Conclusion

Conclusion

Key messages of the day

- **You will lose and corrupt data** - better be prepared
- Checksums will allow you to know
- RAID, RAIN and erasure coding will allow to recover
  - choose your security level, pay the price