Low-rate coding using incremental redundancy for GLDPC codes

Cunche M., Savin V., Roca V., Kraidy G., Soro A., Lacan J.

Work supported by the CAPRI-FEC ANR project

IWSSC'08

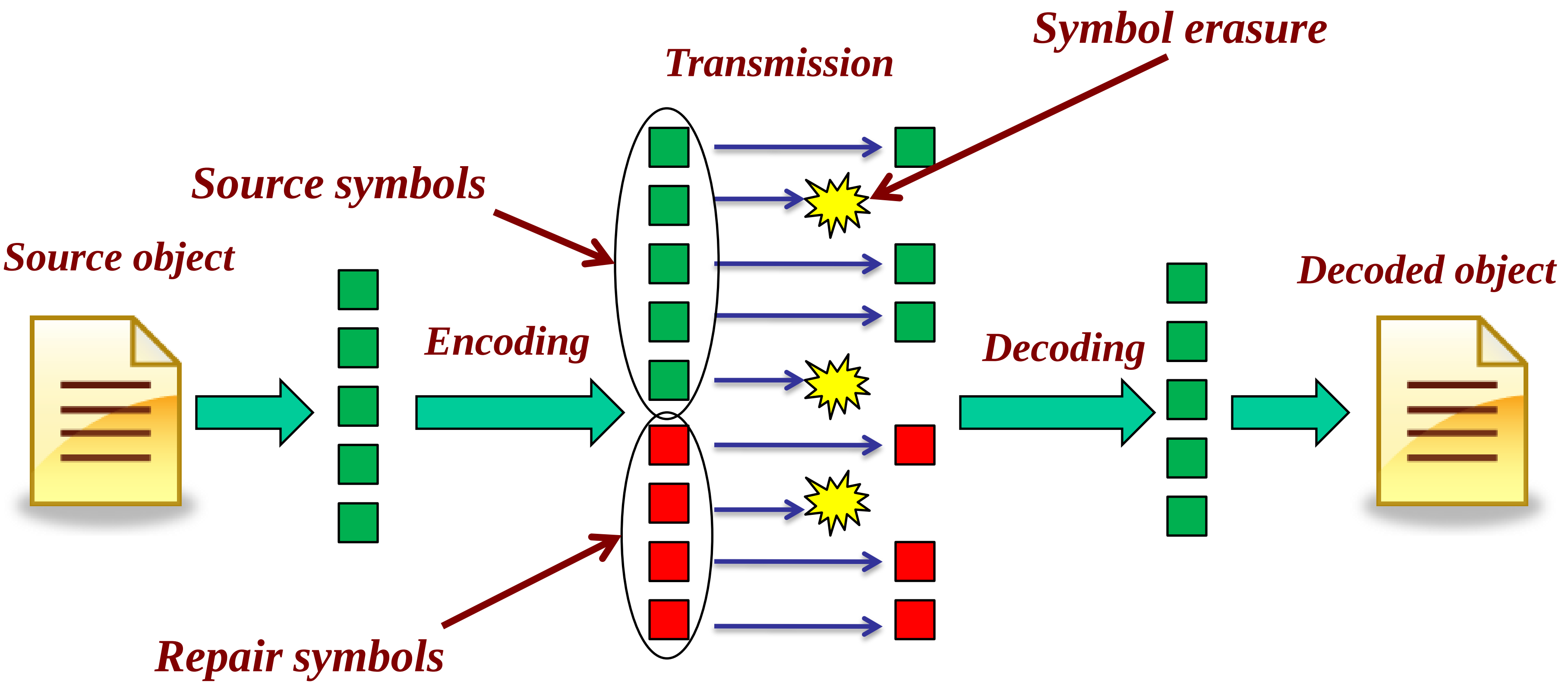
October 3rd, Toulouse

Introduction

- FEC codes for the **erasure** channel
  - Symbols either erased or received without error
- **Low rate** coding (i.e., add a lot of redundancy)
  - to improve carousel-based transmissions (e.g., with FLUTE/ALC), or to counter with very high loss rates
- Proposal based on LDPC-staircase codes
  - Belong to “regular repeat accumulate” codes
  - Now an **IETF standard** (RFC5170) http://www.rfc-editor.org/rfc/rfc5170.txt
- **Extended** with a Generalized LDPC scheme

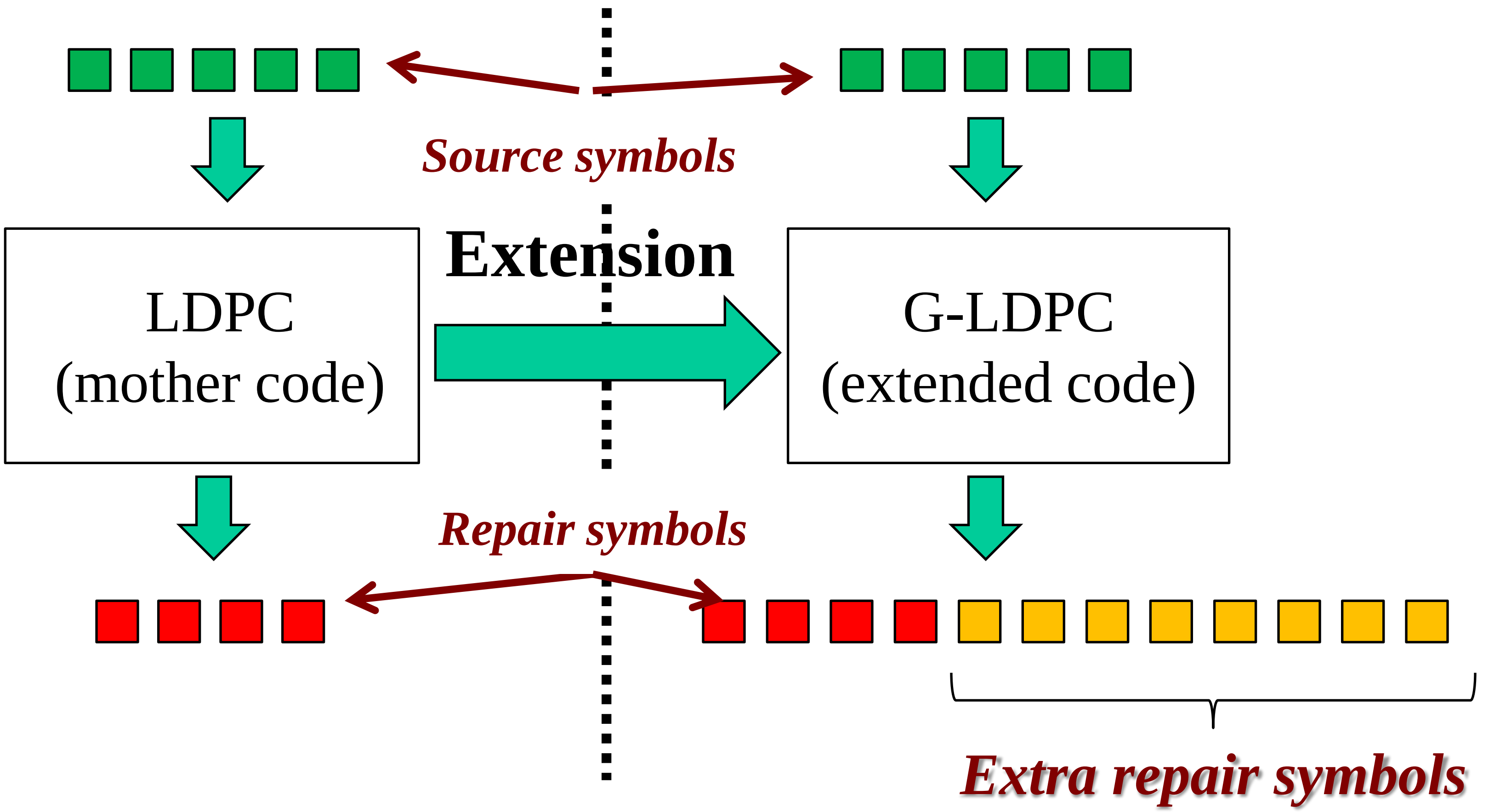
What is a FEC code for the erasure channel?

- Source object is divided into k symbols
- Encoding: add redundancy with (N-K) repair symbols
- Decoding: rebuild the source object from the K(1+ε) symbols received



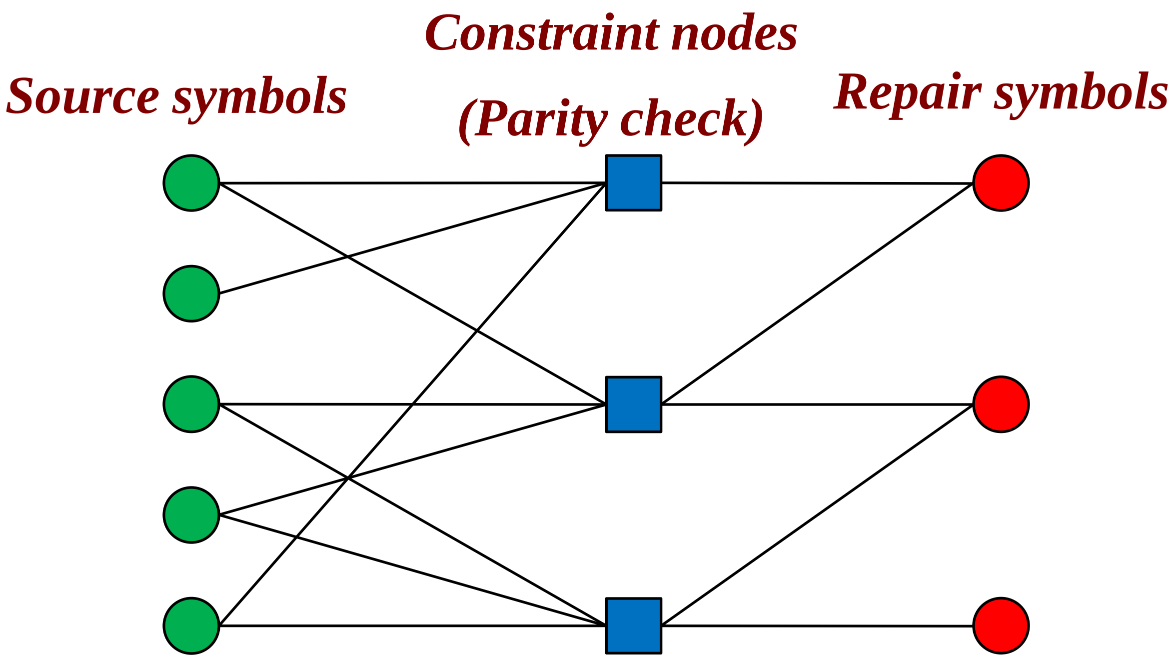
Proposed coding scheme (1/6)

- Extend a “Mother code” for low rate coding
  - Use a Generalized-LDPC construction to add **extra repair symbols**



Proposed coding scheme (2/6)

- « Mother » code: LDPC-Staircase
  - Based on Simple parity checksum (XOR)
    - **1 repair symbol created per constraint node**



Constraint_node 1:
$S_1 \oplus S_2 \oplus S_5 \oplus P_1 = 0$

Constraint_node 2:
$S_1 \oplus S_3 \oplus S_4 \oplus P_1 \oplus P_2 = 0$

Constraint_node 3:
$S_3 \oplus S_4 \oplus S_5 \oplus P_2 \oplus P_3 = 0$

Proposed coding scheme (3/6)

- Encoding
  - **Linear time encoding** thanks to an appropriate code structure
- Iterative Decoding
  - **If a constraint node has all but one symbol known, the latter is equal to the sum of the others. Reiterate if possible…**
  - **Linear time decoding** ☺

$$S_1 \oplus S_2 \oplus S_5 \oplus P_1 = 0 \quad S_5$$

$$S_1 \oplus S_3 \oplus S_4 \oplus P_1 \oplus P_2 = 0 \quad S_3$$

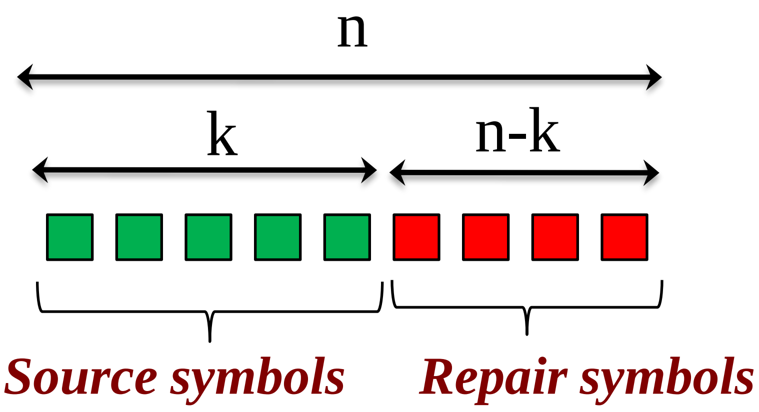
$$S_3 \oplus S_4 \oplus S_5 \oplus P_2 \oplus P_3 = 0$$

S2

S_1	S_2	S_3	S_4	S_5	P_1	P_2	P_3

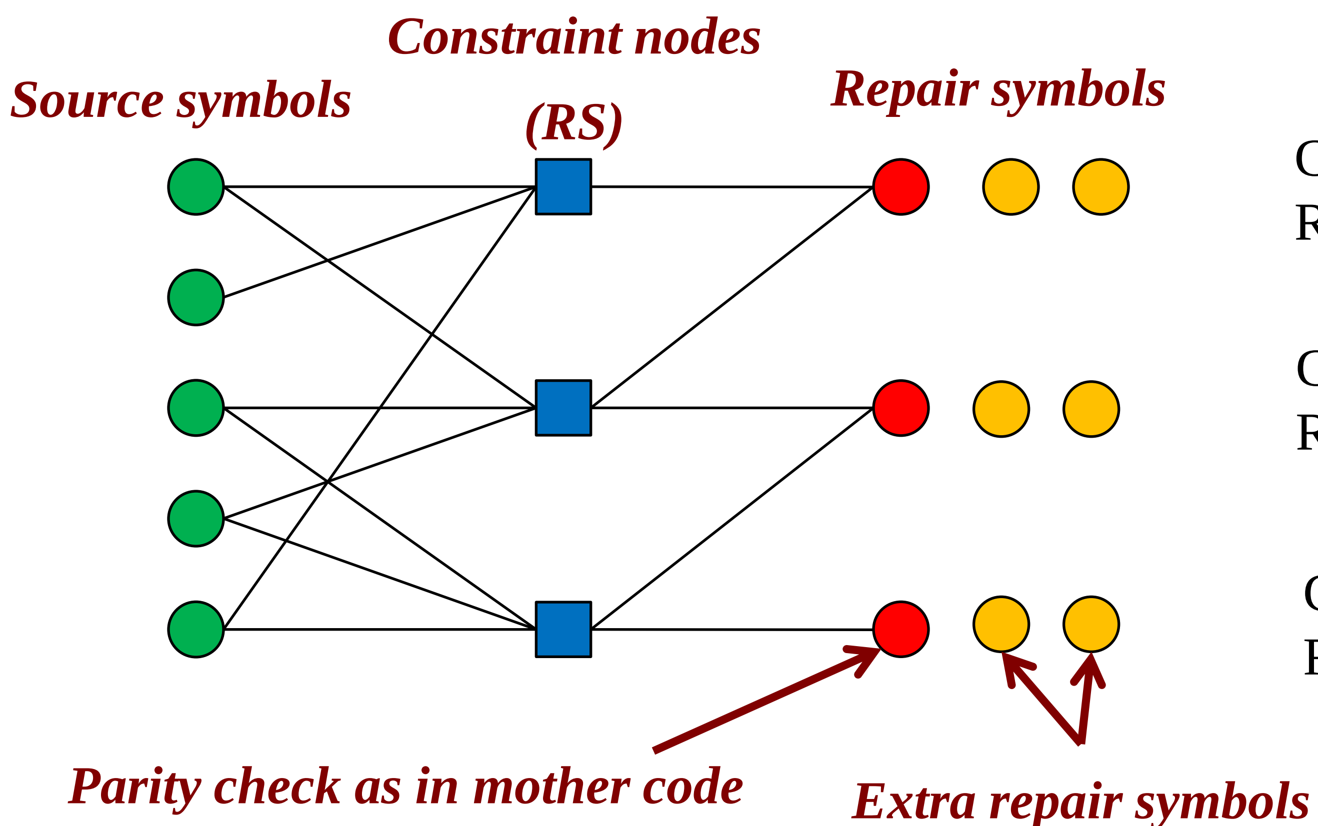
Proposed coding scheme (4/6)

- Extended with Reed Solomon (RS) codes
  - **They are ideal codes**
  - **Practical limit on *n* due to encoding/decoding complexity**
    - *Cannot be applied directly on the whole source object*
  - **In our case n is small and we can use small Galois Fields (e.g., GF(2^4)) that are easily encoded/decoded**



Proposed coding scheme (5/6)

- Extended G-LDPC code based on Reed-Solomon
  - (1 + E) repair symbols created by constraint node
    - **With appropriate RS codes, the first repair symbol remains the parity check (idem LDPC-staircase codes)**



Constraint_node 1:
RS(S_1,S_2,S_5,P_1, P_4, P_5)

Constraint_node 2:
RS(S_1,S_3,S_4,P_1,P_2, P_6, P_7)

Constraint_node 3:
RS(S_3,S_4,S_5,P_2,P_3, P_8, P_9)

Proposed coding scheme (6/6)

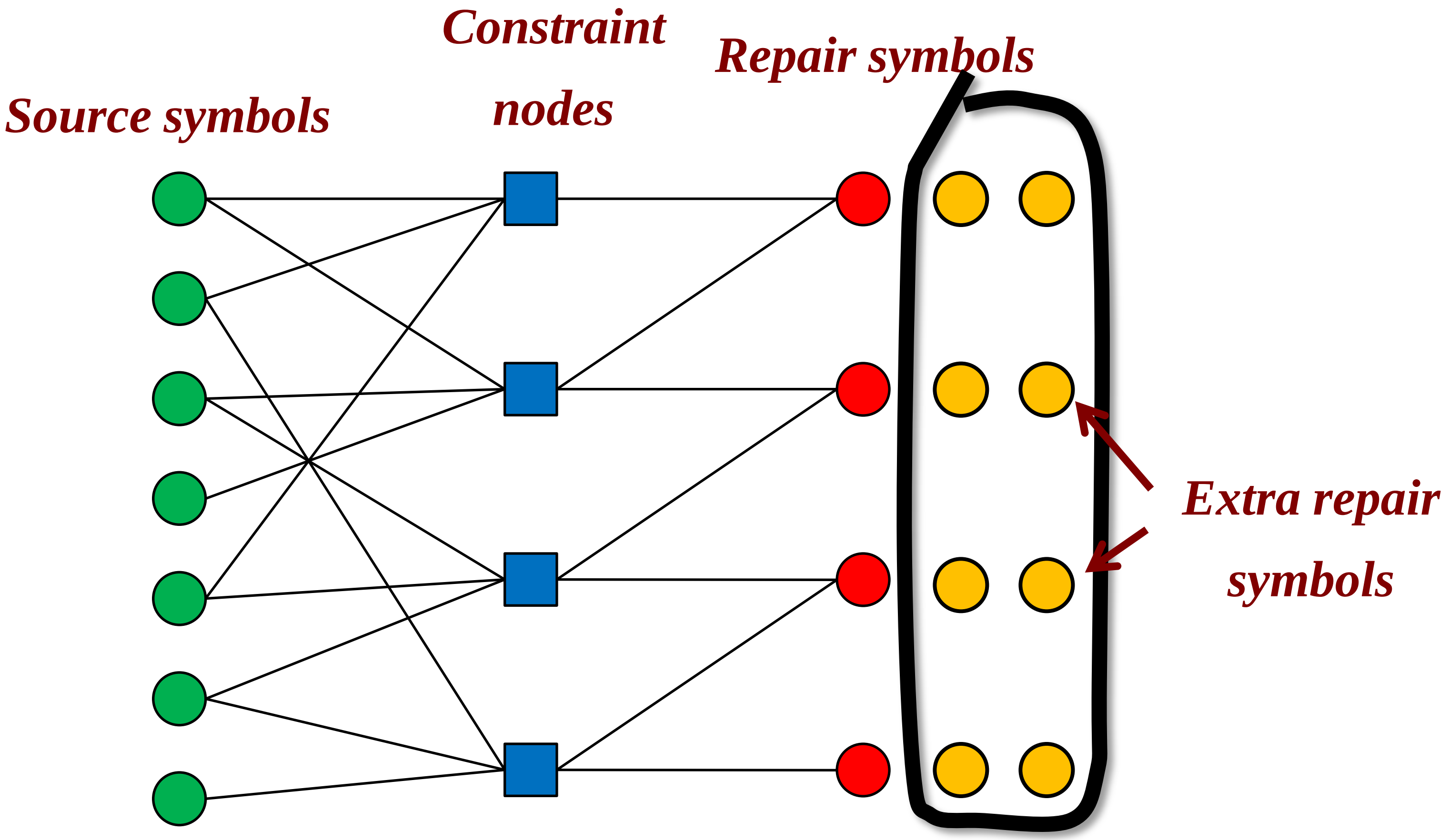
- Encoding
  - **First round: "Parity check" repair symbols created**
  - **Additional rounds: Extra repair symbols created on demand**
  - **Linear complexity**
- Decoding
  - Iterative Decoding (ID) for G-LDPC codes:
    - **Idea: If a constraint node of dimension k, has k symbols known, rebuild the other symbols. And reiterate …**
  - Complexity: linear in the number of source symbols ☺

Distribution of the Extra repair symbols (1/5)

- Is it appropriate to produce the same number of Extra Repair Symbol per constraint node?
  - Not necessarily!
  - We show that a non constant number can help improving the erasure recovery capabilities…
    - **We tested 3 distributions**

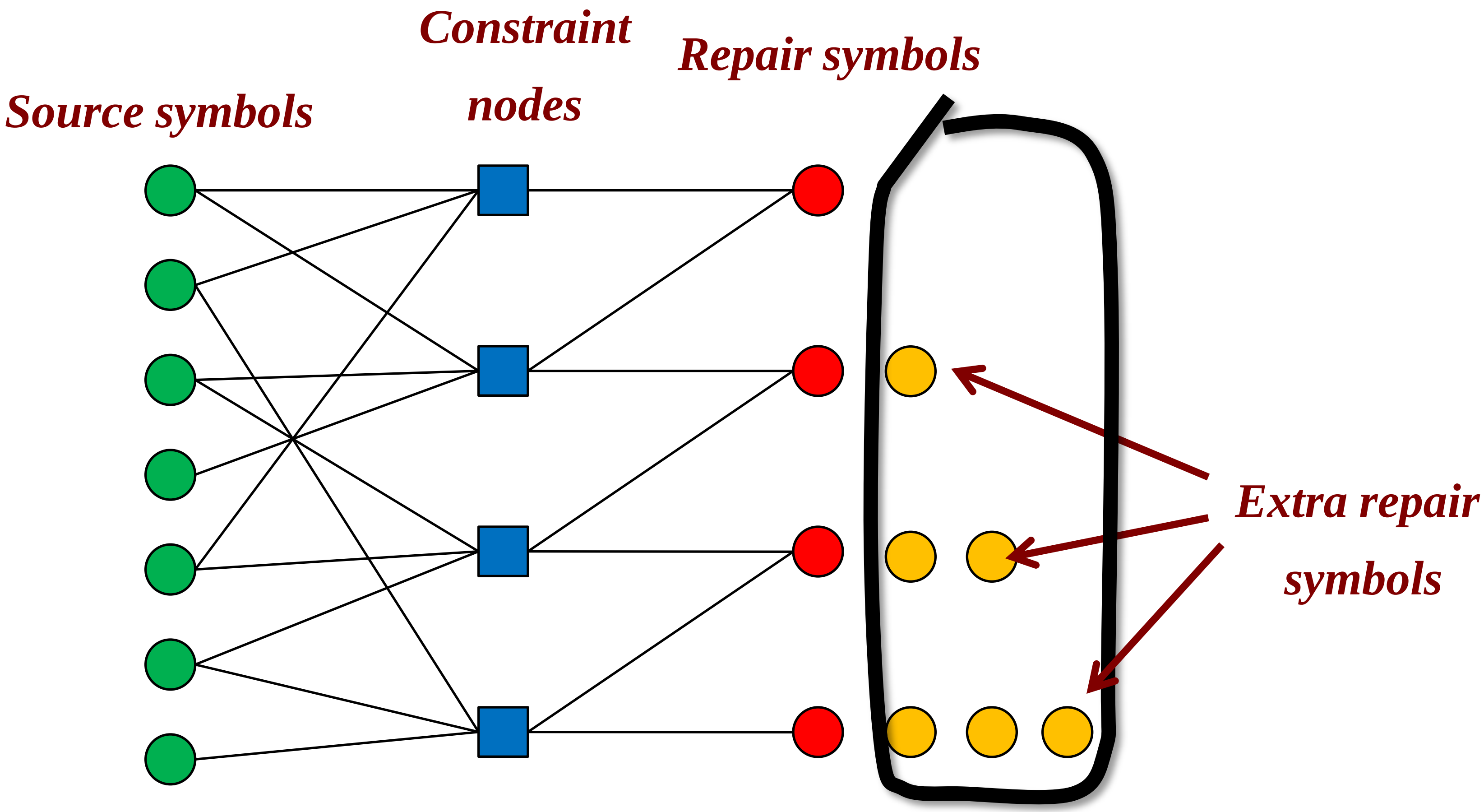
Distribution of the Extra repair symbols (2/5)

1/ Constant : the number of extra repair symbols connected to a constraint node is constant.



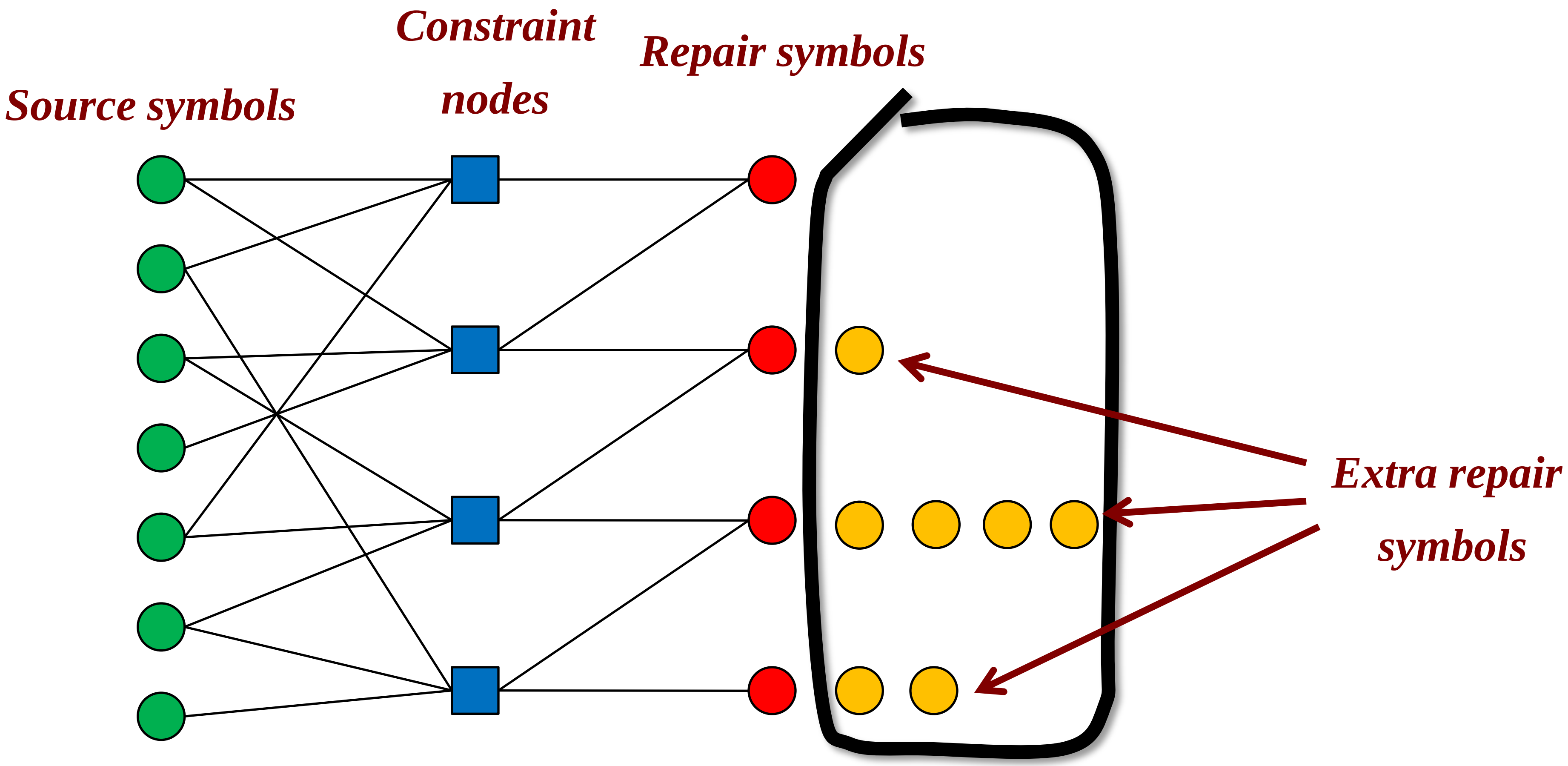
Distribution of the Extra repair symbols (3/5)

2/ Uniform: the number of extra repair symbols connected to a constraint node is uniformly distributed between 0 and a maximum value E_{max}.



Distribution of the Extra repair symbols (4/5)

3/ Irregular: the number of extra repair symbols connected to a constraint node is irregularly distributed between 0 and a maximum value E_{max}.



Distribution of the Extra repair symbols (5/5)

- Density evolution analysis
  - Find a good irregular distribution (#3) of the extra repair symbols produced
  - We found the best irregular distribution (see paper)
- In fact, uniform distribution…
  - …is very close to the best irregular distribution
  - …is better than constant distribution
  - …is fairly simple

We use uniform distribution !

Results (1/2)

Conditions: **K=5,000 source symbols,**

mother code rate=1/2

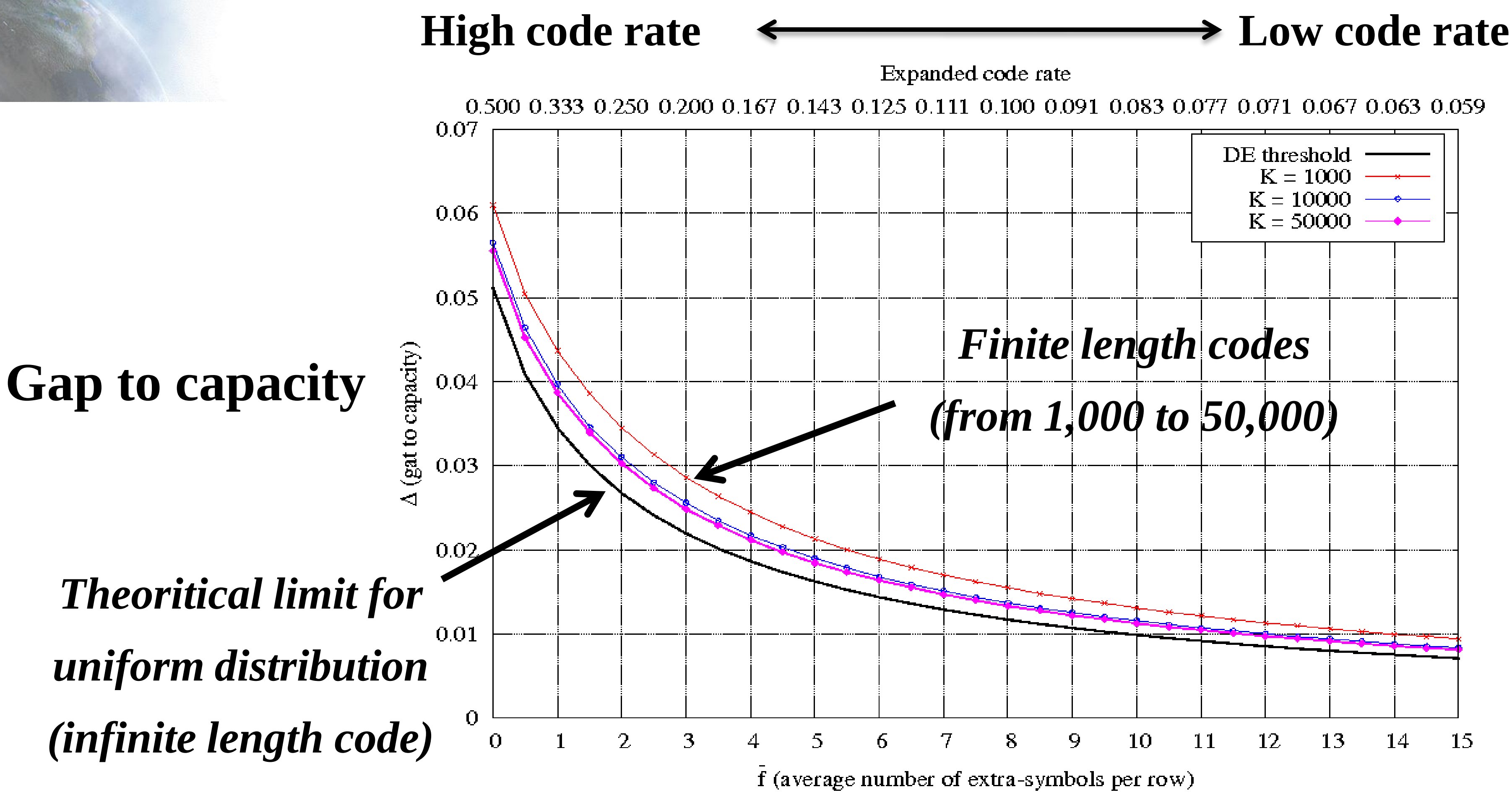
code rate	Average overhead		
	Extended codes		LDPC-Staircase
	Uniform distrib.	Constant distrib.	
1/2	11.4%	11.4%	11.4%
1/5	13.0%	13.4%	32.8%
1/10	14.0%	16.5%	84.6%
1/17	14.4%	18.2%	144.0%

Fairly stable performances, even at small code rates ☺

Unusable with iterative decoding at small rates (use ML decoding…)

Results (2/2)

- Uniform distribution



Gap to capacity (i.e., distance to ideal code perf.) decreases with the code rate ☺

Additional advantages (1/2)

Advantages at the encoder…

- Flexibility on the encoder side: Extra repair symbols can be produced on demand, in “rounds”
  - **To adapt dynamically to the loss rate**
  - **To start transmissions earlier (no need to wait for all repair symbol creation) and to reduce the delay**
  - **To save resources (no need to remember all extra repair symbols)**

Additional advantages (2/2)

Advantages at the decoder…

- Limited memory requirements
  - No need to store the whole matrix, the mother code matrix (much smaller) is sufficient
  - No need to re-build extra repair symbols during decoding ($\neq$ ID with LDPC codes)
- Backward compatibility…
  - An RFC5170 compliant decoder can decode with source/parity symbols, ignoring extra repair symbols

To conclude

- An efficient **small rate** coding scheme
  - **good erasure recovery capabilities at very very low rates**
- Relies on an iterative decoding scheme
  - **Guaranties linear decoding complexity,**
  - **Decoding remains fast even with huge source objects (≠ ML decoding)**
- Incremental redundancy added **on demand**
  - **Provides a high flexibility**

To conclude

- A very **simple** design
  - **Based on well-known and standardized building blocks**

- A possible alternative to rate-less codes
  - **We can easily/efficiently reach very small code rates**

With RS over GF(2^4) we can reach a code rate 1/7

GF(2^8) we can reach a code rate 1/127

……

Questions ?