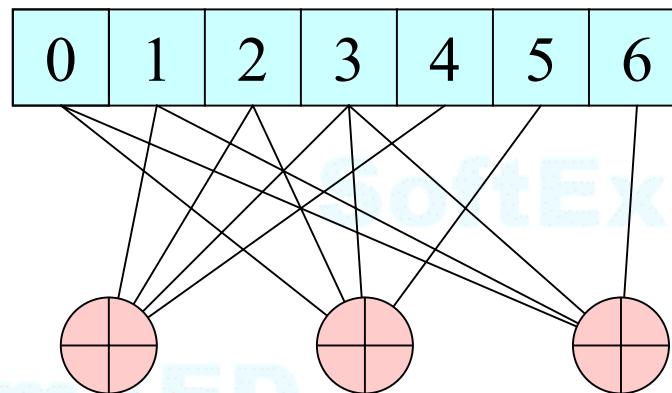


Architecture générique de décodage de code LDPC



F. GUILLOUD

E. BOUTILLON

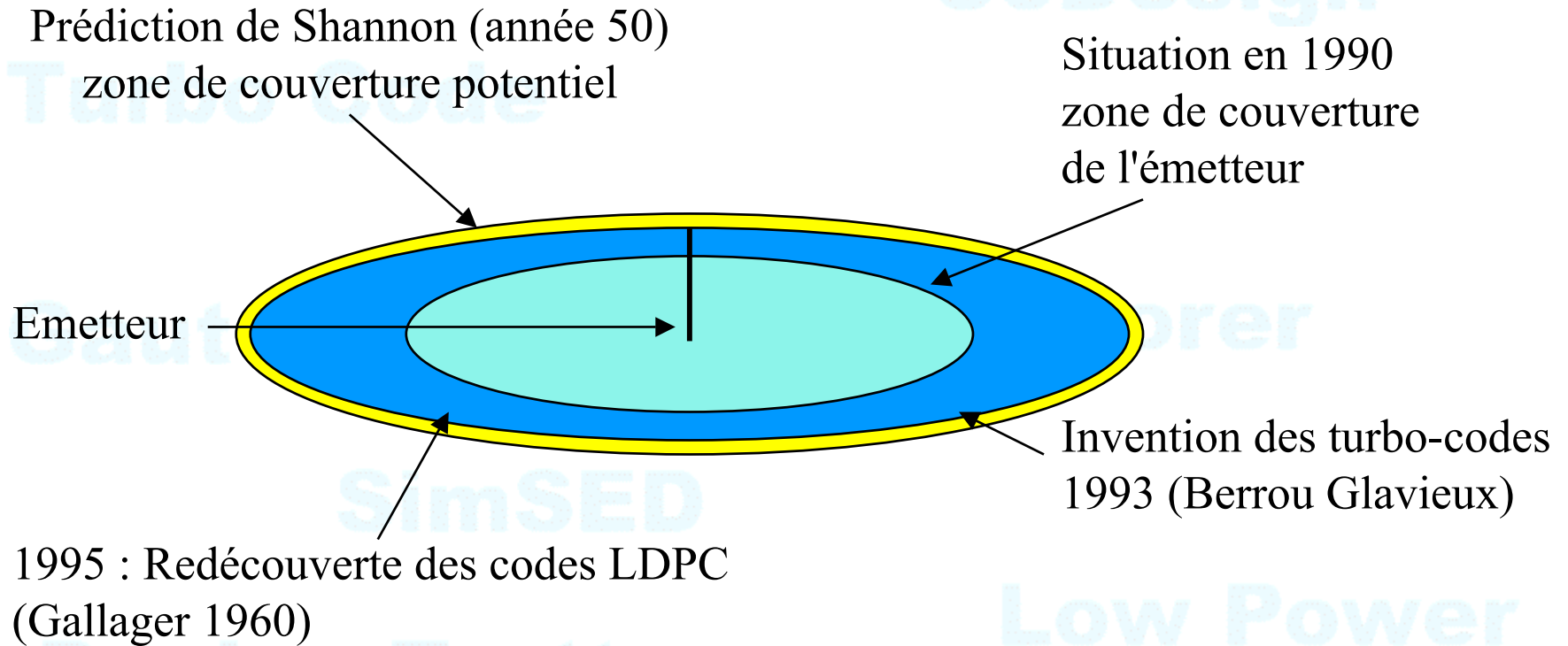
guilloud@enst-bretagne.fr

emmanuel.boutillon@univ-ubs.fr

1/54



Importance du codage...



Compétition LDPC-Turbo-Code...



Low Density Parity Check Code

Inventé en 1961 par Gallager (thèse MIT), oublié, redécouvert en 1995 (MacKay, Ritchardson).

Gallager	LDPC	1962
Berrou, Glavieux	Turbo Codes	1993
MacKay	LDPC	1995
Wiberg	Graphes	1995
Richardson, Urbanke	Density Evolution	2001
Premières architectures de décodeurs		2001
Standard DVB-S2		2003



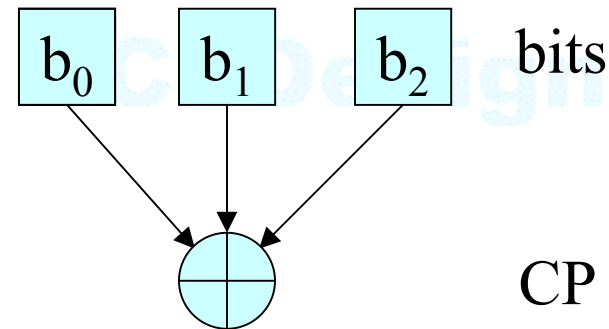
PLAN

- ❶ Code de parité
- ❷ Principe des codes LDPC
- ❸ Architecture de décodeur LDPC
- ❹ En guise de conclusion



Parity check

Parity check (3,2,1) :



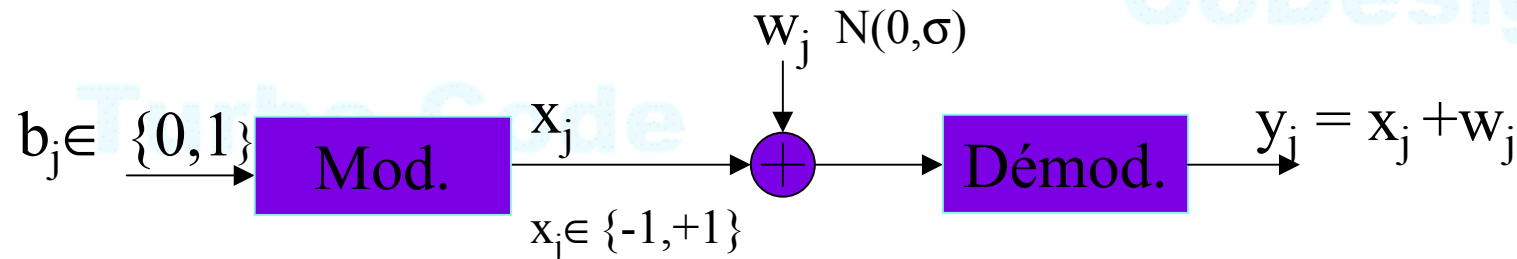
(b_0, b_1, b_2) codeword \Leftrightarrow Sum of 1 = 0 mod 2.

$CP(3,2,1) = \{(0, 0, 0) ; (0, 1, 1) ; (1, 1, 0) ; (1, 0, 1)\}$

Generalization: Parity Check $(n, n-1, 1)$



Décodage d'un code de Parité



L'observation y_j donne l'information *intrinsèque* i_j du bit b_j :

$$i_j = (p(b_j=0/y_j), p(b_j=1/y_j))$$

que l'on exprime aussi par le "log likelihood ratio":

$$i_j = \ln(p(b_j=1/y_j)/p(b_j=0/y_j)) = 2y_j/\sigma^2$$

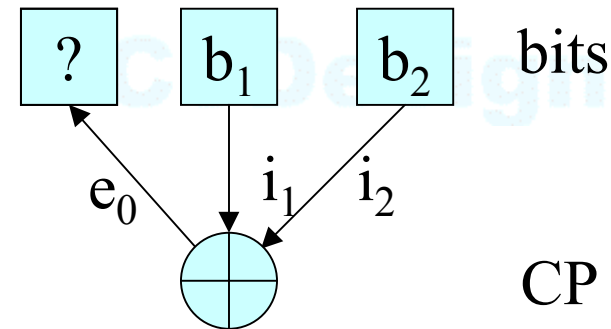
Note: $\text{sign}(i_j) \Rightarrow$ décision dure, $|i_j|$ fiabilité de la décision



Décodage d'un code de Parité

Quelle information i_1 et i_2 donnent sur la valeur b_0 ?

En utilisant:



$$p(b_0=0/y_1, y_2) = p(b_1=0/y_1) \cdot p(b_2=0/y_2) + p(b_1=1/y_1) \cdot p(b_2=1/y_2)$$

$$p(b_0=1/y_1, y_2) = p(b_1=0/y_1) \cdot p(b_2=1/y_2) + p(b_1=1/y_1) \cdot p(b_2=0/y_2)$$

on obtient une observation e_0 de la valeur de b_0 indépendante de i_0 :

$$e_0 = i_1 \oplus i_2 = \ln\left(\frac{1 + e^{i_1} \cdot e^{i_2}}{e^{i_1} + e^{i_2}}\right)$$

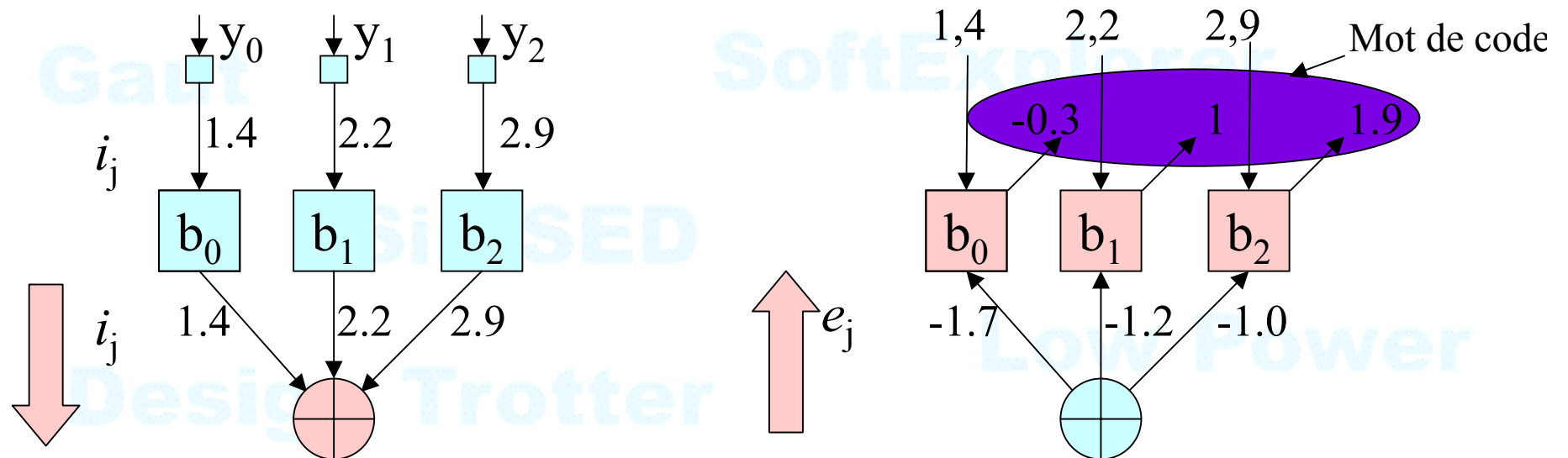
e_0 est appelé information *extrinsèque*



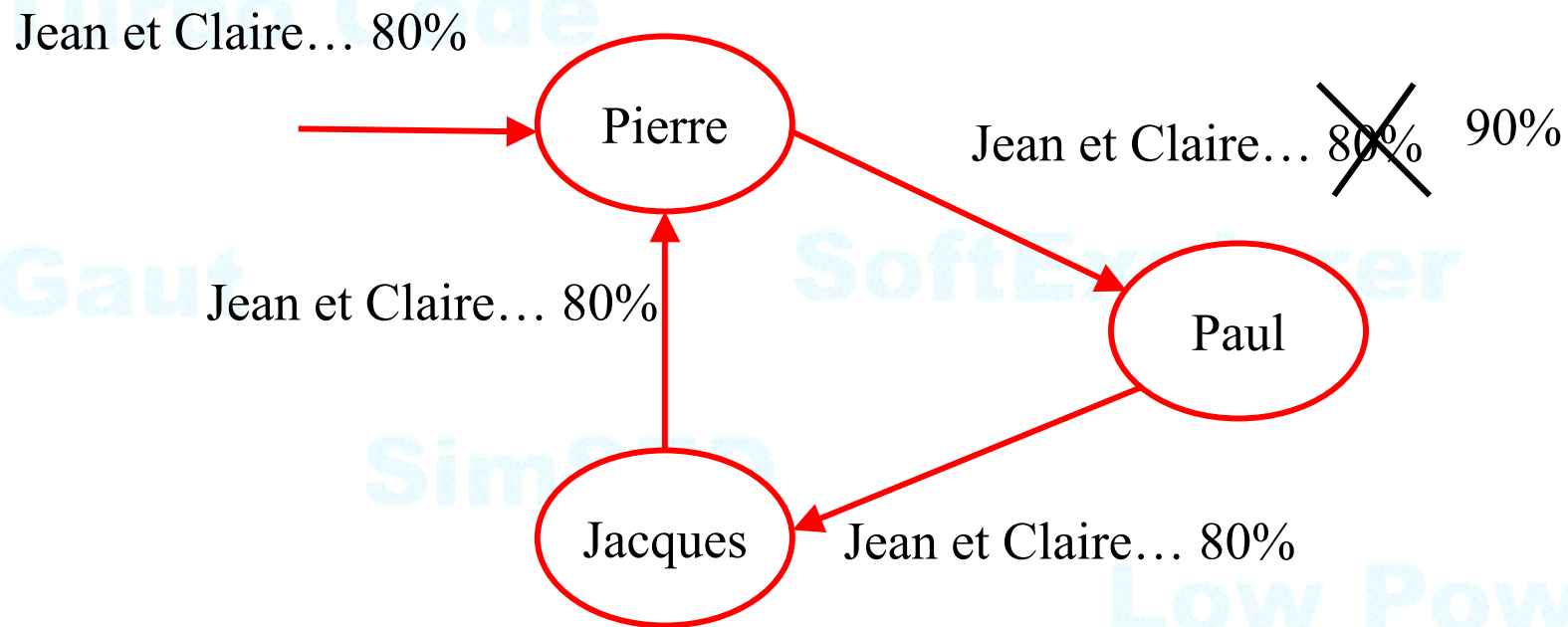
Décodage d'un code de parité

Enfin, i_0 et e_0 sont additionnés pour obtenir la valeur finale.

Le processus est symétrique pour tous les bits.

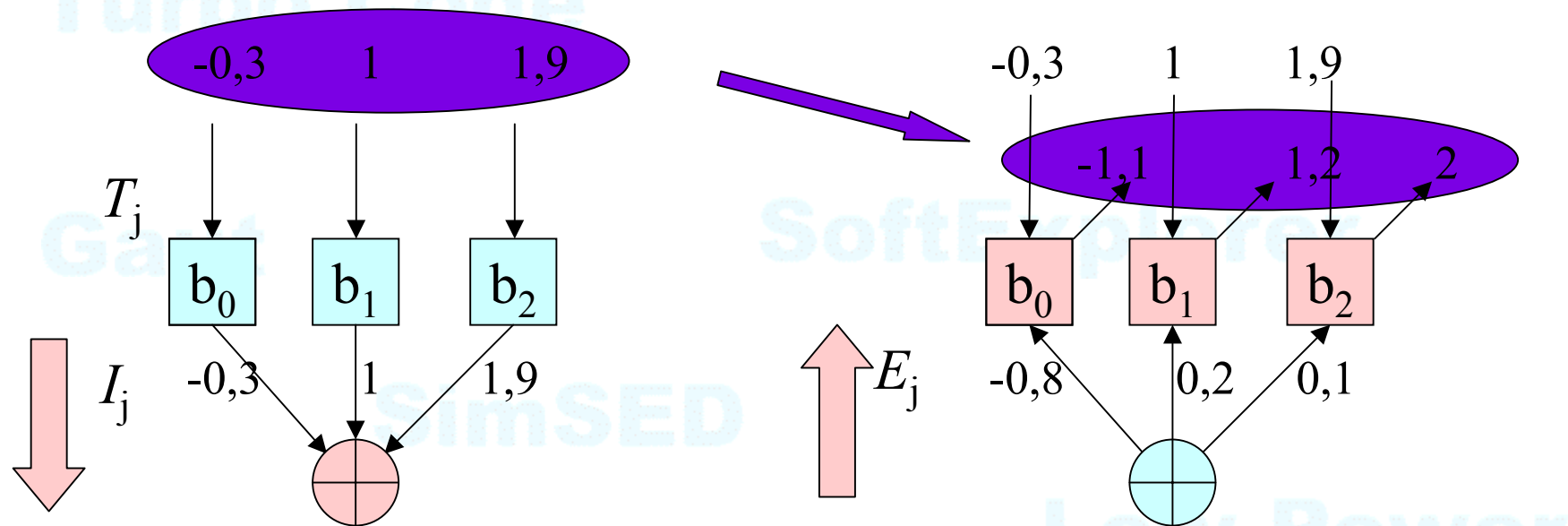


Conseil : éviter les auto-confirmations



Note: itération

Si le processus itère de nouveau, il y a autoconfirmation



Et le système **diverge**...



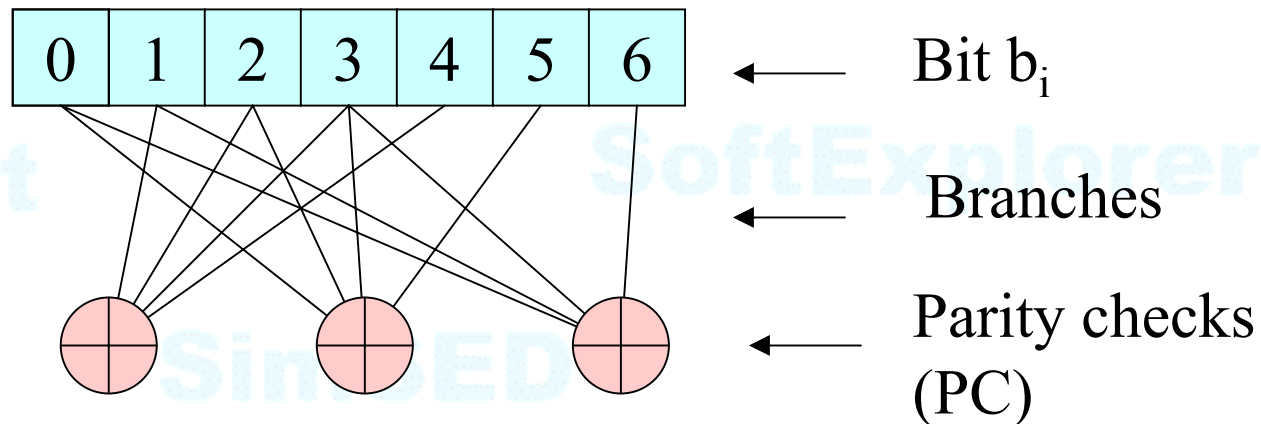
PLAN

- ❶ Code de parité
- ❷ Principe des codes LDPC
- ❸ Architecture de décodeur LDPC
- ❹ En guise de conclusion



LDPC Code

Peut être défini par un graph bi-partite



(b_0, b_1, \dots, b_6) mot de code \Leftrightarrow toutes les PC sont respectées



Pourquoi le nom de LDPC ?

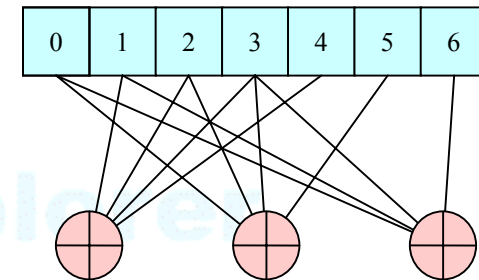
Représentation algébrique :

$X = (x_0, x_1, \dots, x_6)^T$ mot de code si $H.X = 0$

$$H = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

$\xleftarrow{\hspace{10em}} N = \text{nombre de bits}$

$\uparrow P = \text{nombre de PC}$
 Matrice de parité



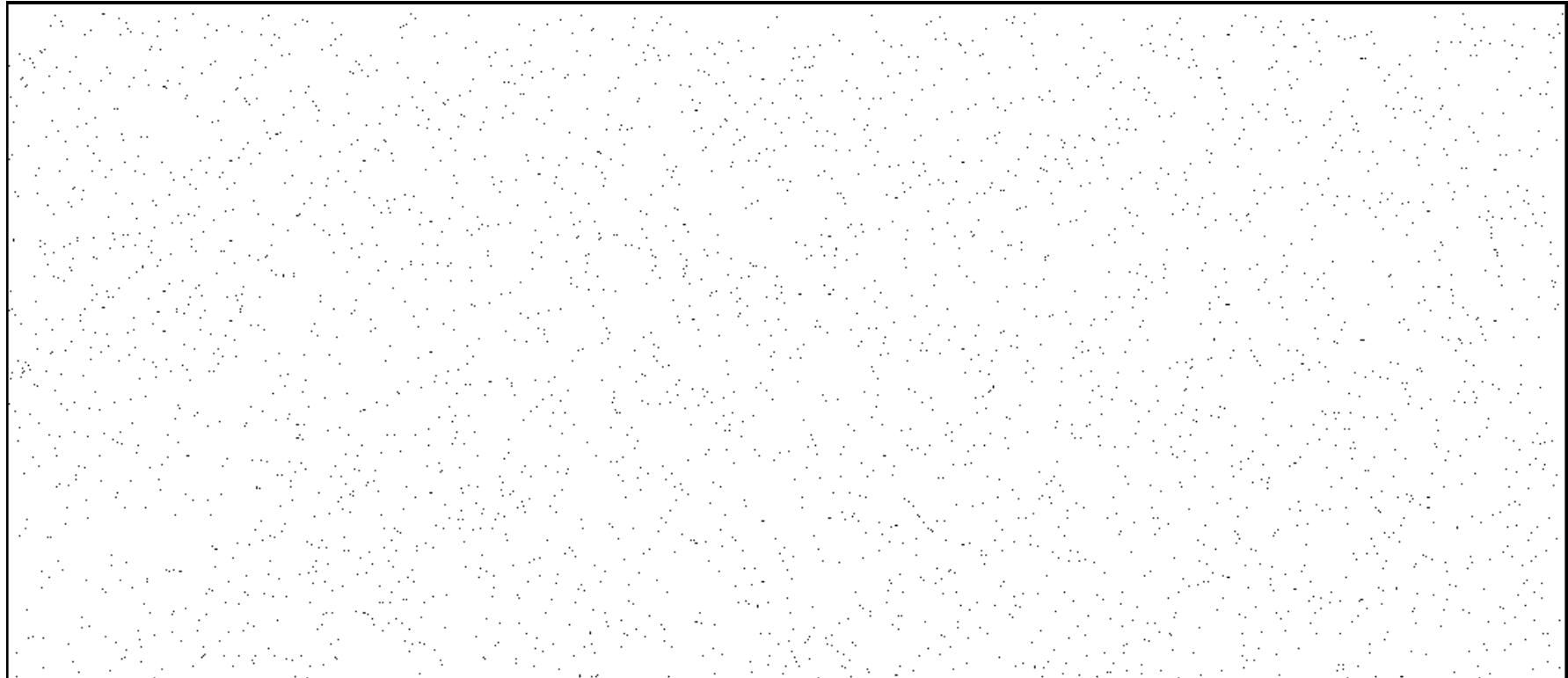
Nombre de 1 sur une ligne = d_c = nombre de bit associé au PC

Nombre de 1 sur une colonne = d_v = nombre de PC associé à la bit

Moins de **1%** des bits de H sont égaux à 1



Exemple de matrice LDPC



Matrice de parité H de taille $(1024, 512)$, $(d_v, d_c) = (3, 6)$

Question : rendement du code ? Nombre de 1 de la matrice ?



Comment construire un bon code LDPC ?

Application => taille du code et rendement

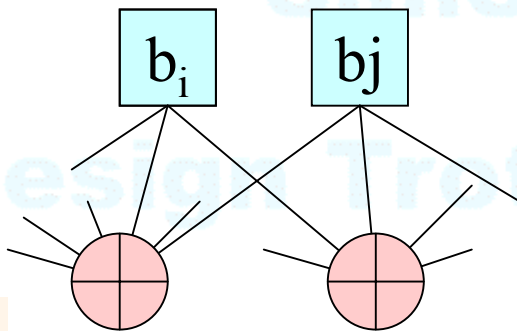
Sélection de la répartition optimale des spectres de répartition des poids des branches (en utilisant des EXIT Chart [ref]):

exemple:

90 % bits => $d_v = 3$ branches, 10 % bits $d_v \Rightarrow 12$ branches

70 % PC => $d_c = 6$ branches, 30 % PC => $d_c = 8$ branches

et on choisi le code aléatoirement... en évitant juste les cycles :



...et on obtient un bon code

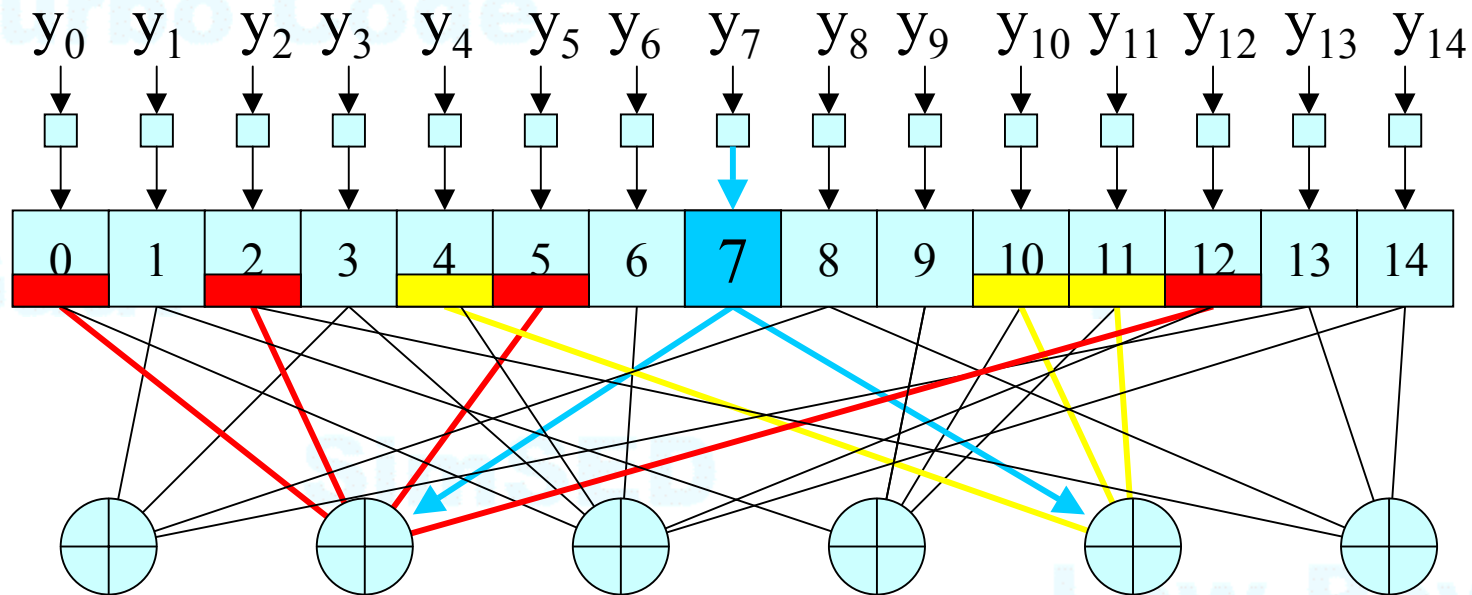


Algorithme à propagation de croyance

- Step1: calcul du LLR des bits reçus (information intrinsèque)
- Step2: Message bit->parité + traitement parité
- Step3: Message parité->bit + traitement bit
- Step4: répéter 2 et 3 jusqu'au décodage correct ou "max iteration".



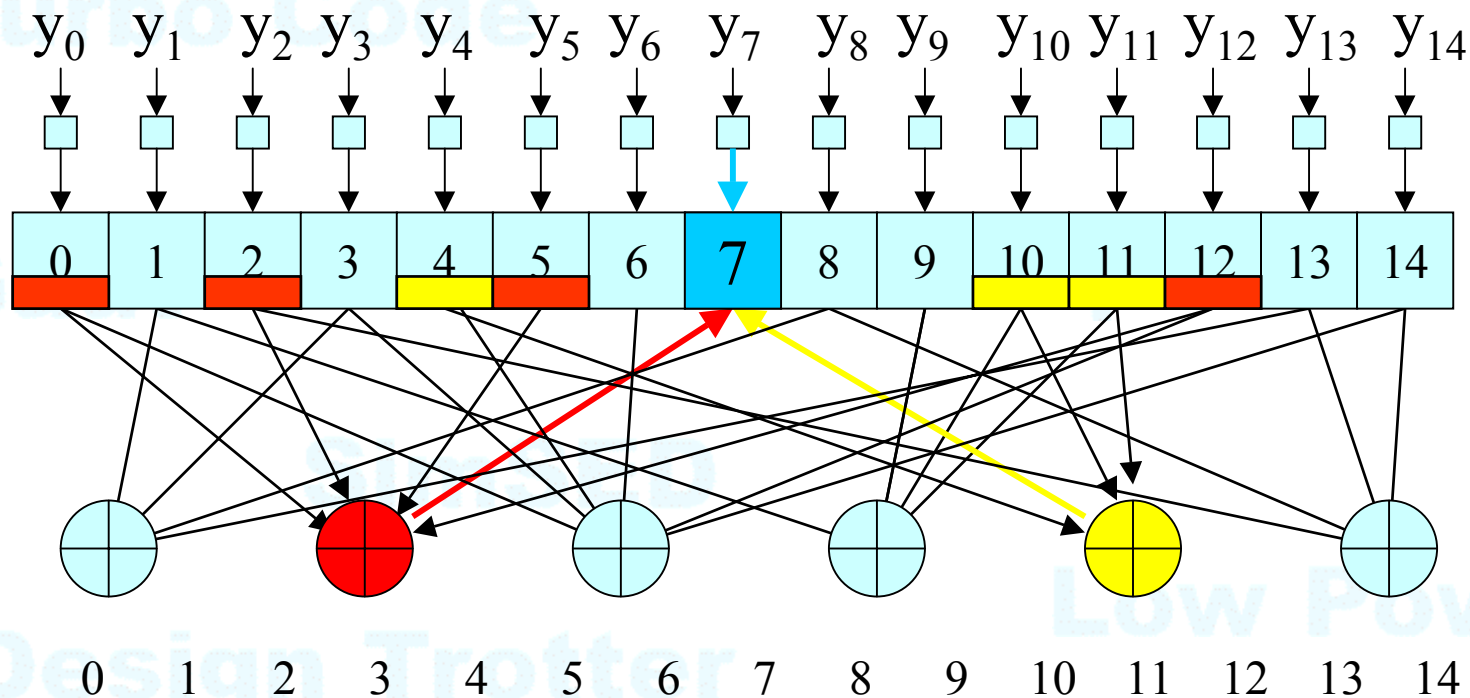
Décodage itératif



Première itération: message bit- \rightarrow parité



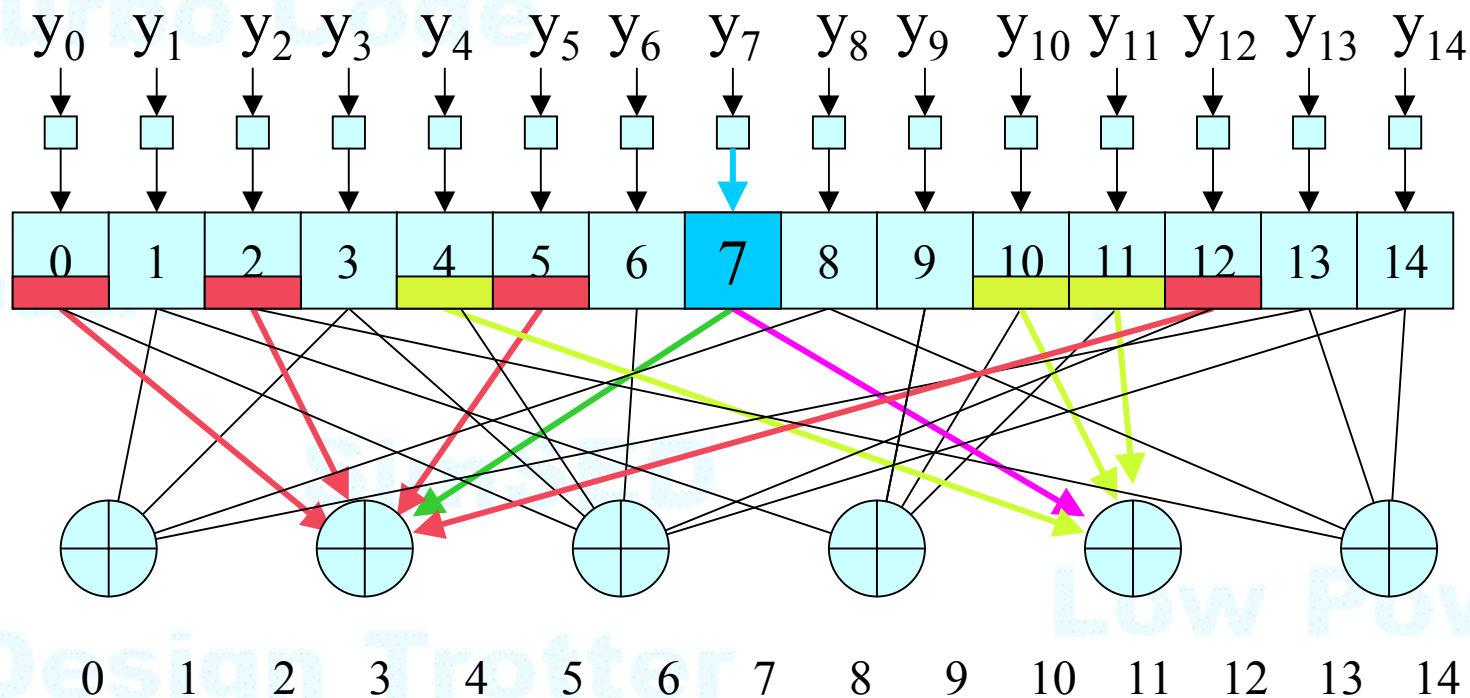
Décodage itératif



Première itération: message parité- \rightarrow bit



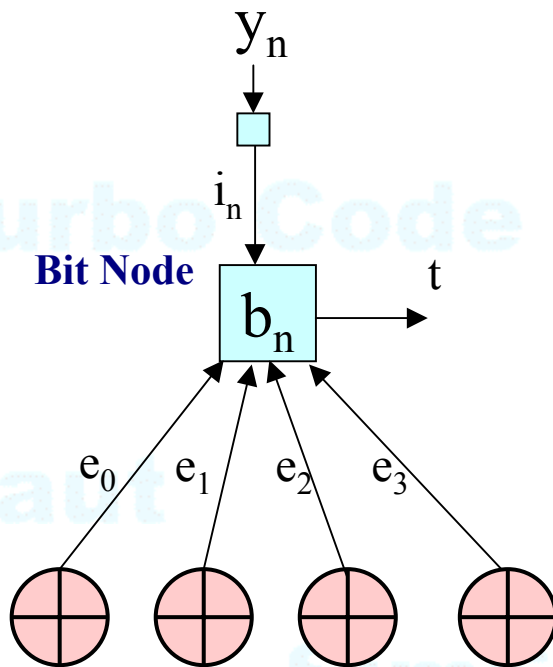
Décodage itératif



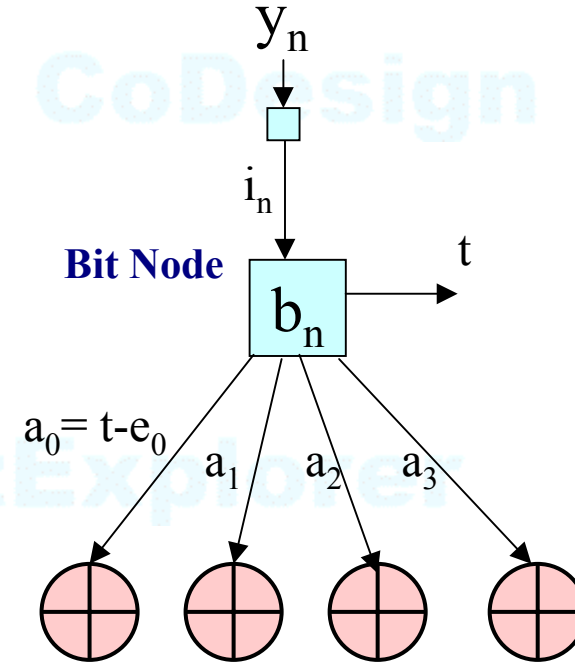
Deuxième itération: message parité \rightarrow bit



Calcul d'un nœud de variable

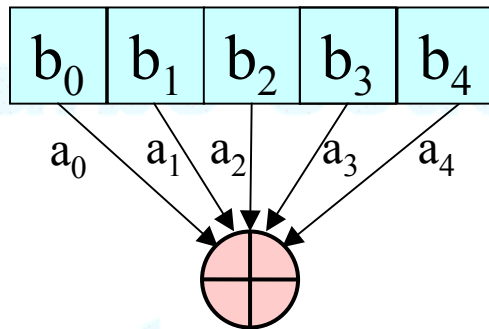


$$t = i_n + \sum_m e_m$$

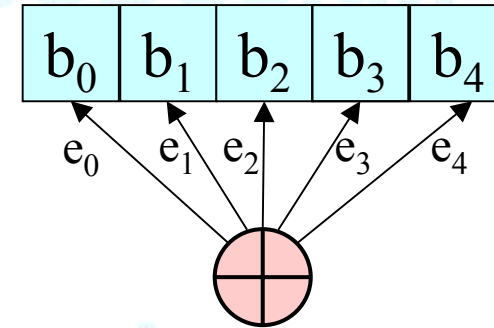


$$a_i = i_n + \sum_{m, m \neq i} e_m = t - a_i$$

Calcul d'un nœud de parité



Check Node



Check Node

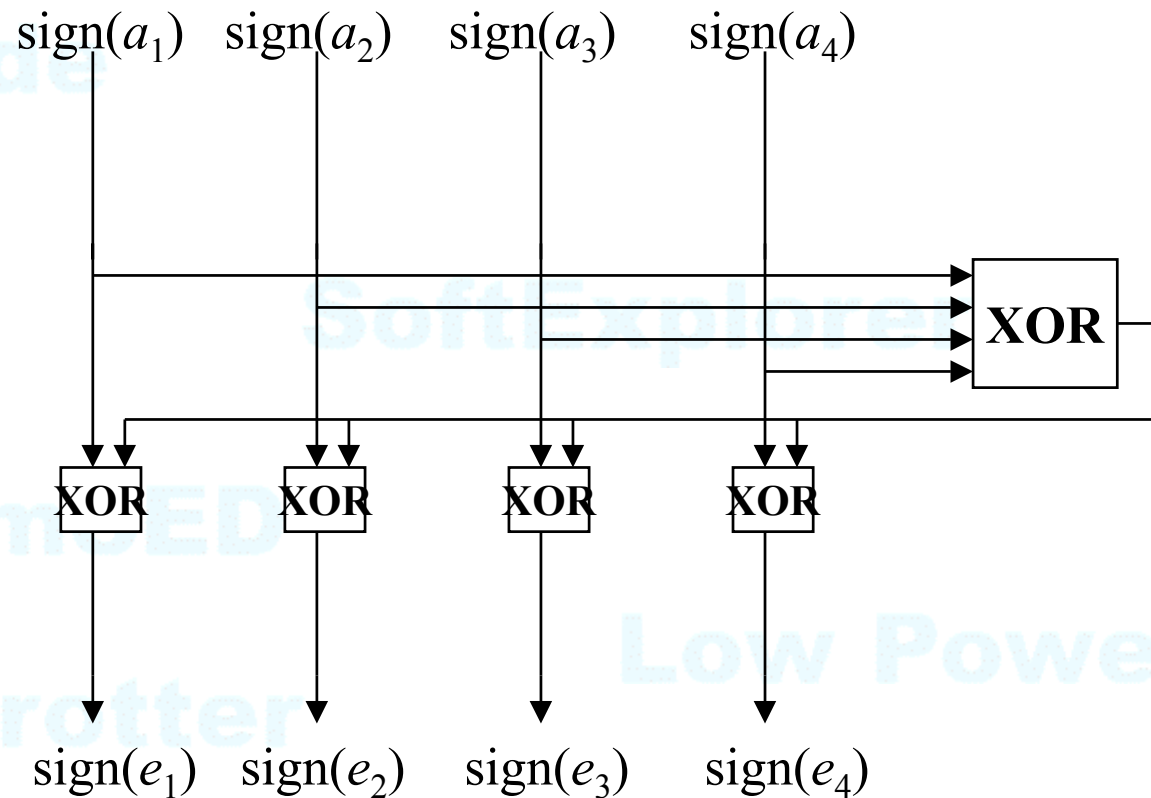
$$e_i = \bigoplus_{j \neq i} a_j = a_1 \oplus a_2 \dots \oplus a_{i-1} \oplus a_{i+1} \dots \oplus a_{dc}$$

Deux méthodes de calcul : directe ou fréquentielle



Calcul parité: méthode fréquentielle

a) calcul du signe



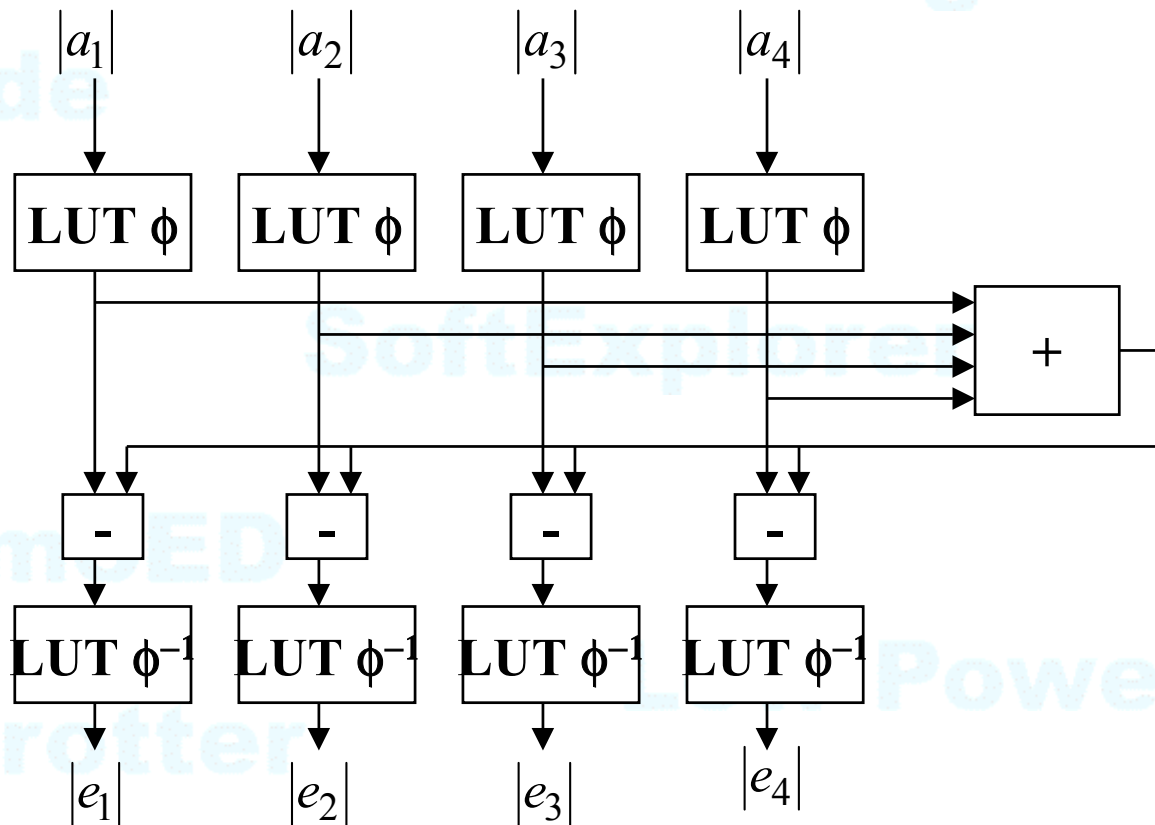
Calcul parité: méthode fréquentielle

b) calcul du module

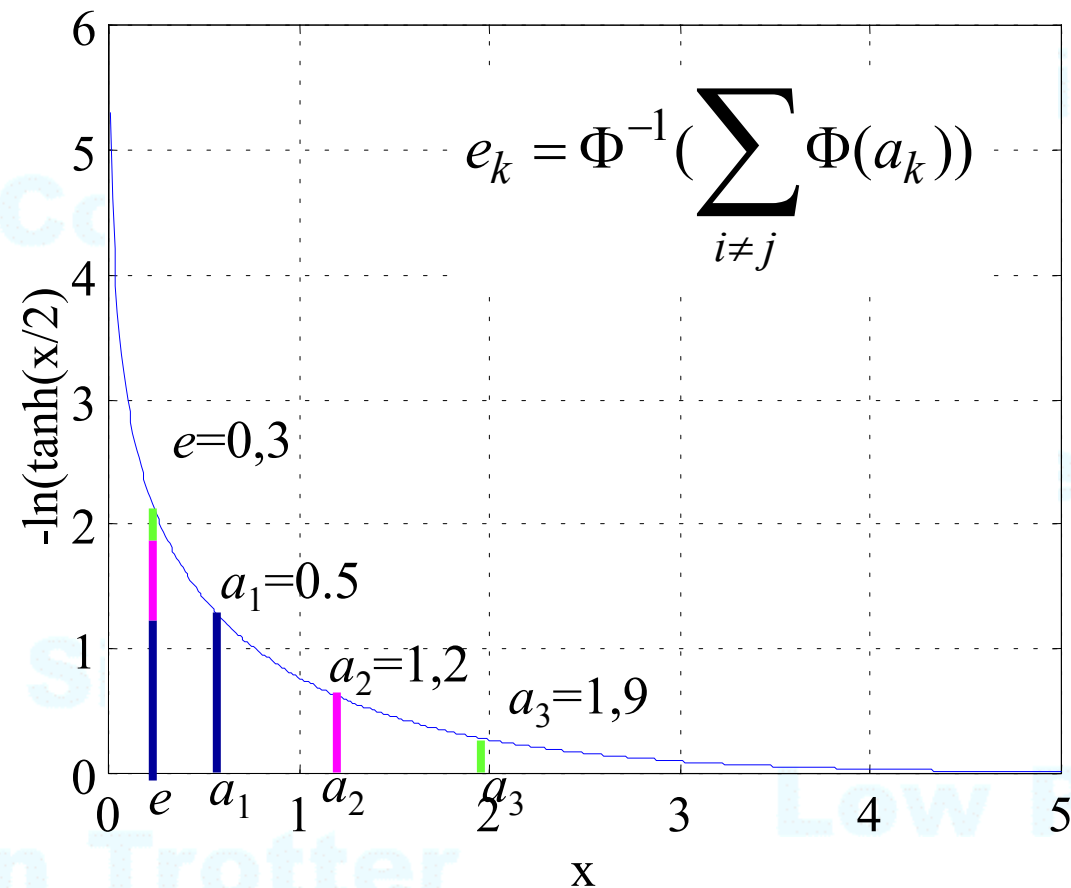
$$|e_k| = \Phi^{-1}\left(\sum_{i \neq j} \Phi(|a_k|)\right)$$

avec

$$\Phi(x) = -\ln(\tanh(x/2))$$



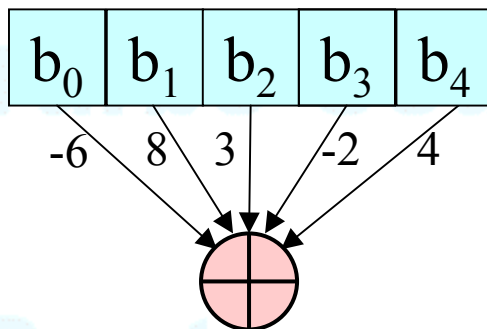
Fonction $\Phi(x) = \Phi^{-1}(x)$



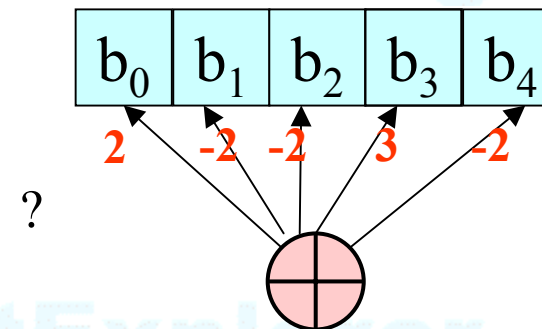
On obtient : $e = \min(a_1, a_2, a_3) - \text{offset}$



Exercice



Processeur de parité



Processeur de parité

La sous-optimalité engendre une perte de 0,5 dB en convergence.

=> solution : on soustrait une constante pour compenser la sur-évaluation de l'extrinsèque : $\min(e_k - \beta, 0)$

=> CONVERGENCE PLUS RAPIDE !!!! (Fossorier)

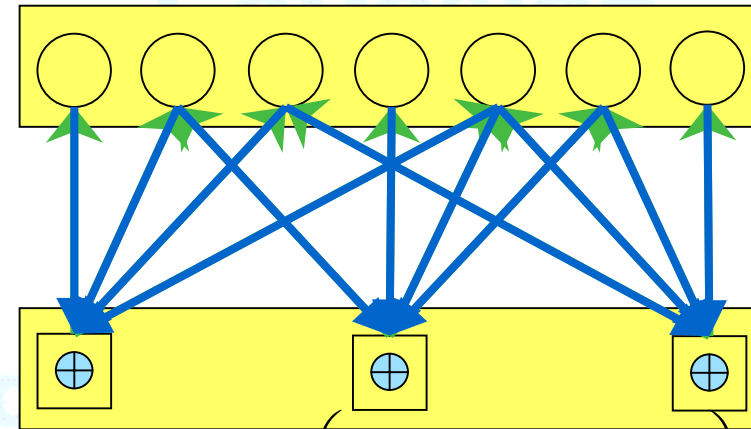


• Algorithme BP

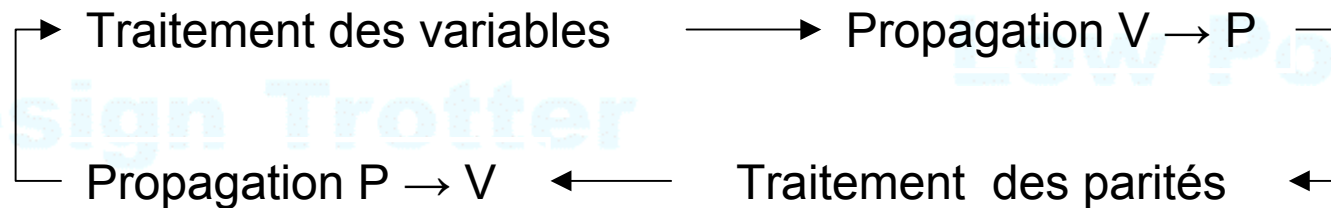
- Propagation de croyance (Belief Propagation)
- Initialisation :
 - ⇒ $I_n = 2y_n/\sigma^2$
 - ⇒ Extrinsicèque branche : $E_{m,n}=0$
 - ⇒ Fiabilités : $T_{m,n}=0$
- Itérations : tous les messages sont traités

$$a_{m,n} = t_n - e_{m,n}$$

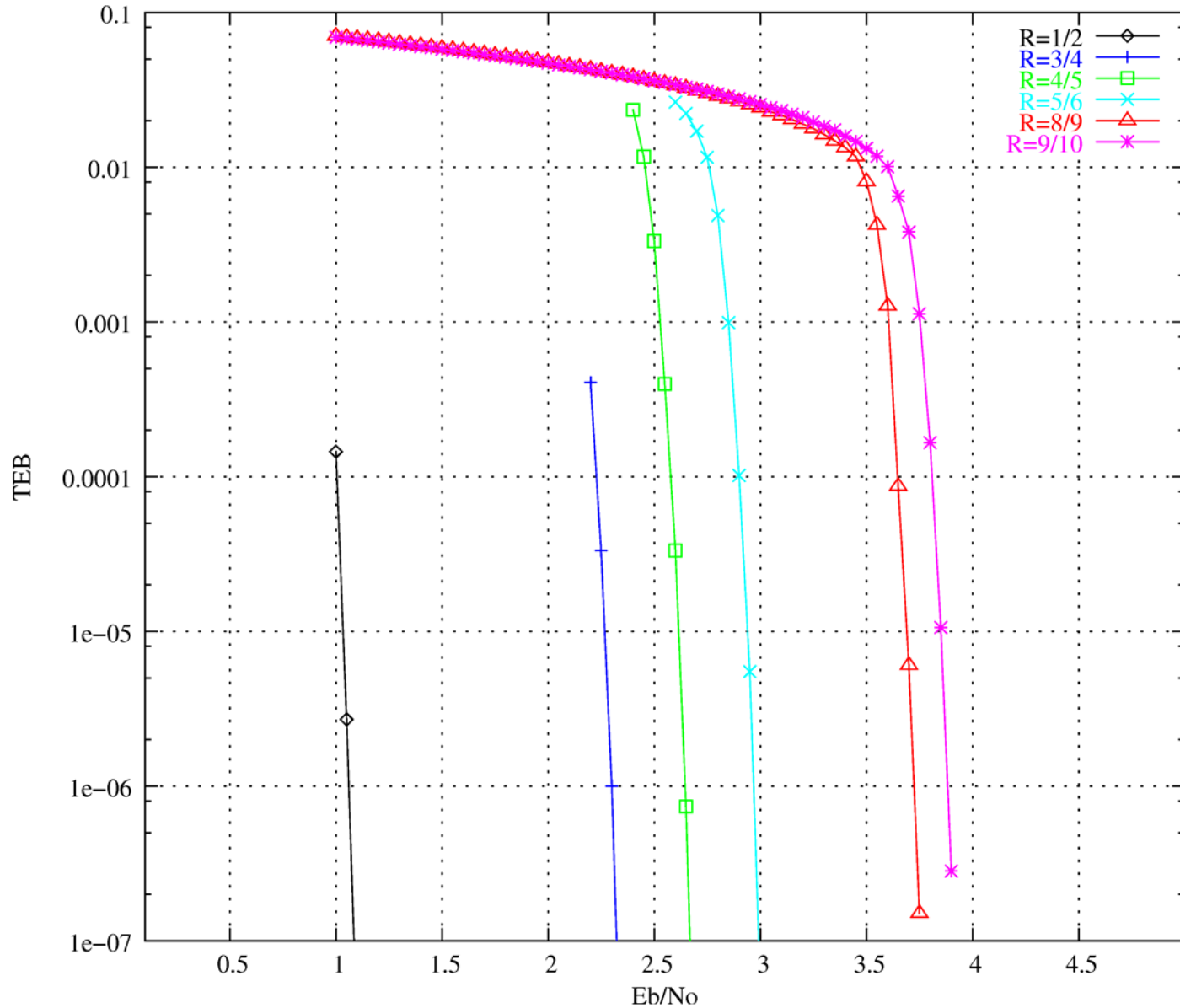
$$= i_n + \sum_{m' \in M(n) \setminus m} e_{m',n}$$



$$e_{m,n} = \Phi^{-1} \left(\sum_{n' \in N(m) \setminus n} \Phi(a_{m,n'}) \right)$$



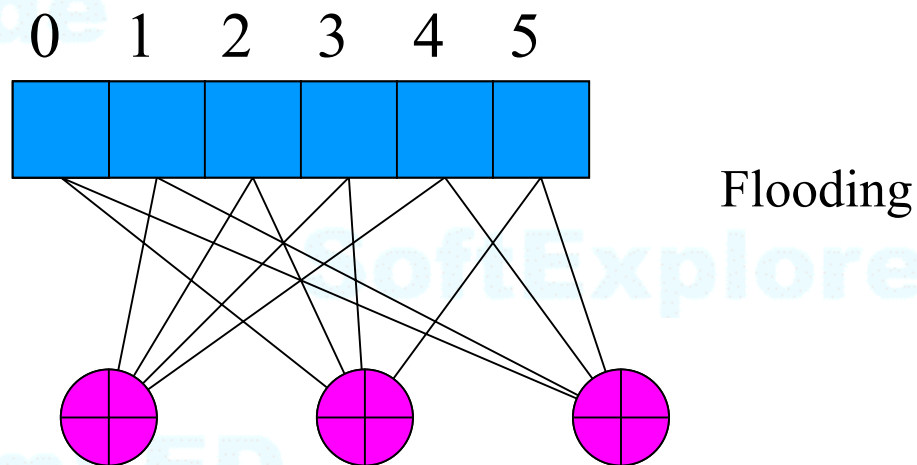
Performances DVB-S2



64k
50 itérations

LDPC + BCH

Sequencement par inondation (Méthode classique)



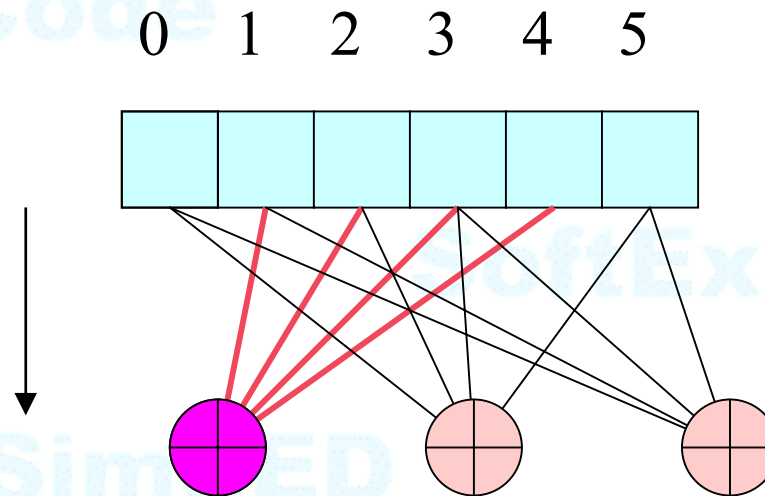
=> postulat faux : unique séquencement efficace



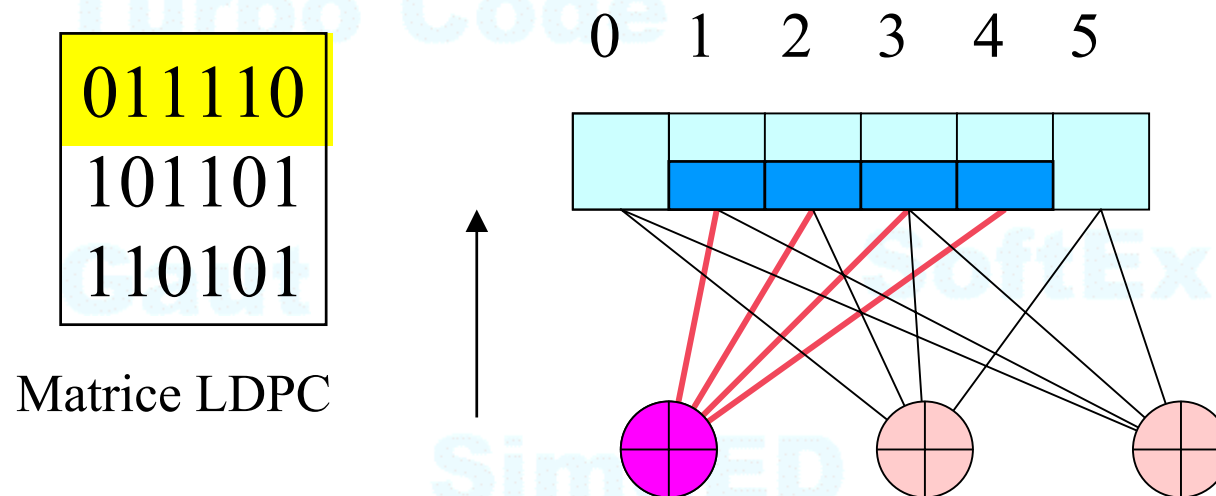
Sequencement horizontal (Mansour)

011110
101101
110101

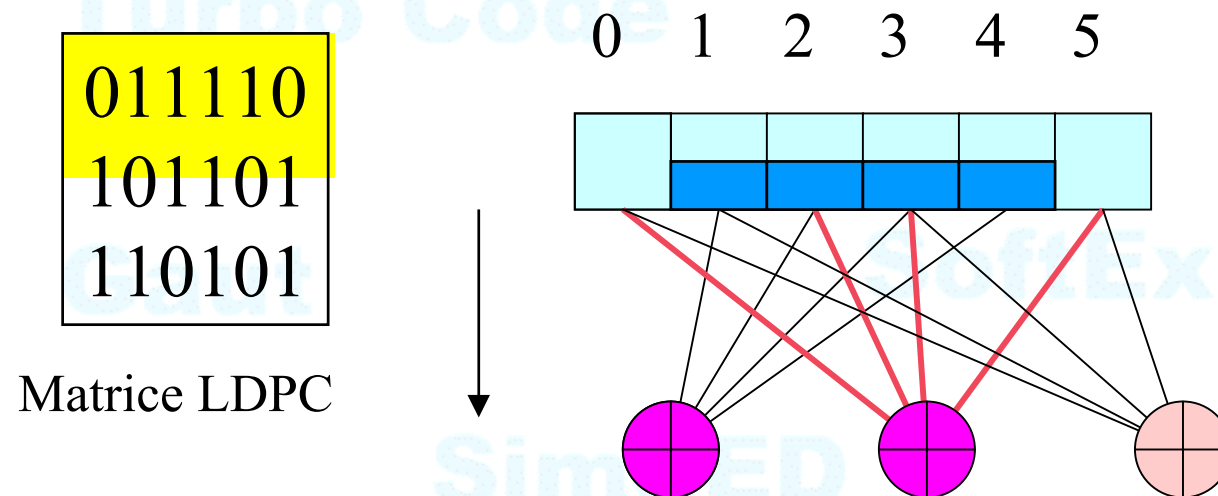
Matrice LDPC



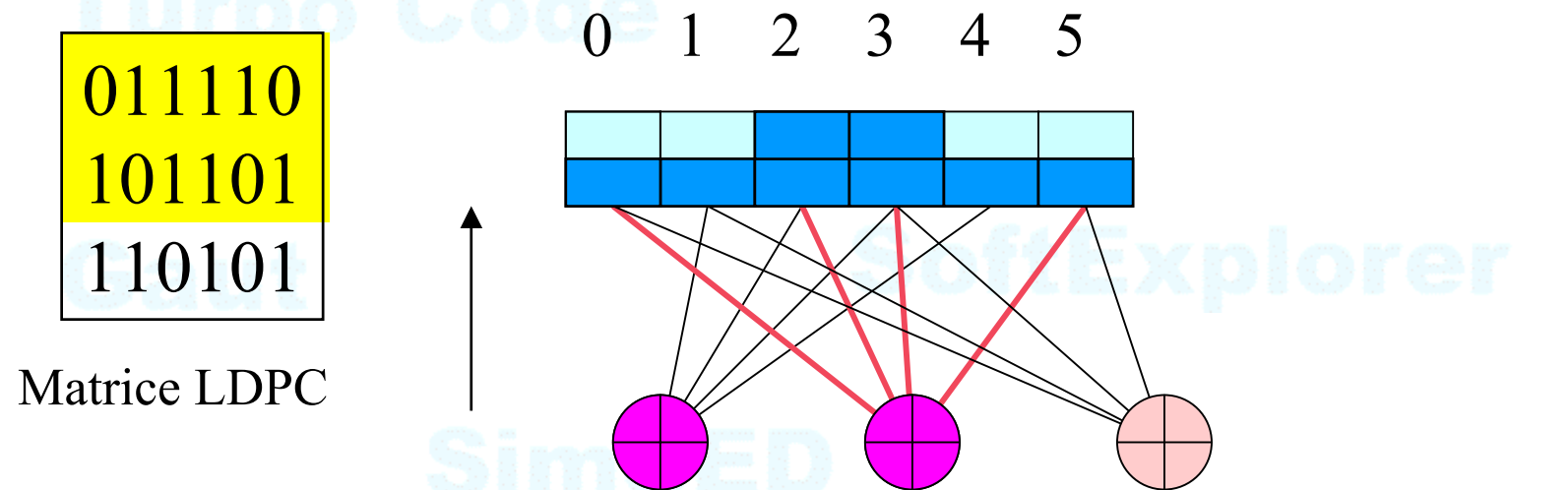
Sequencement horizontal (Mansour)



Sequencement horizontal (Mansour)



Sequencement horizontal (Mansour)



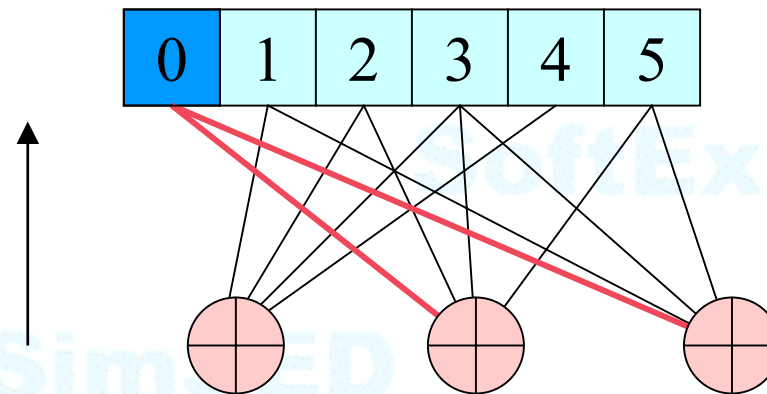
Convergence en moitié moins d'itération



Sequencement vertical (Fossorier)

0	1	1	1	1	0
1	0	1	1	0	1
1	1	0	1	0	1

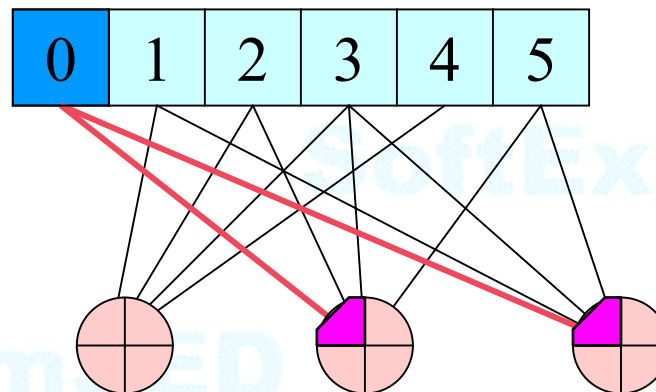
Matrice LDPC



Sequencement vertical (Fossorier)

0	1	1	1	1	0
1	0	1	1	0	1
1	1	0	1	0	1

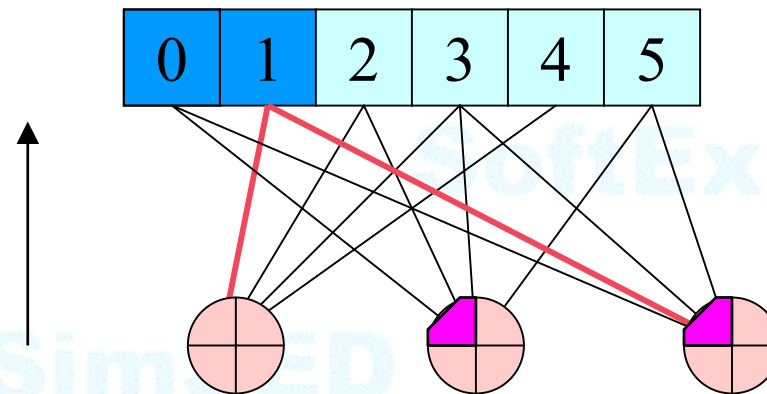
Matrice LDPC



Sequencement vertical (Fossorier)

0	1	1	1	1	0
1	0	1	1	0	1
1	1	0	1	0	1

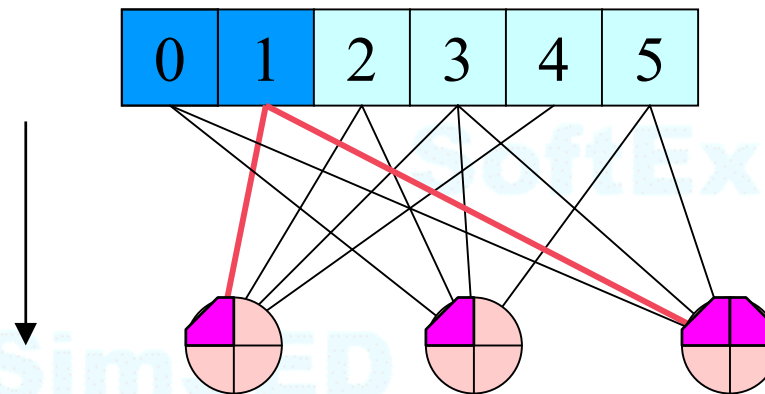
Matrice LDPC



Sequencement vertical (Fossorier)

0	1	1	1	1	0
1	0	1	1	0	1
1	1	0	1	0	1

Matrice LDPC



De nouveau, convergence en moitié moins d'itération !



Problème séquençement H et V

CoDesign

Turbo Code

Trouver une architecture adaptée...

SimSED

Low Power

Design Trotter



PLAN

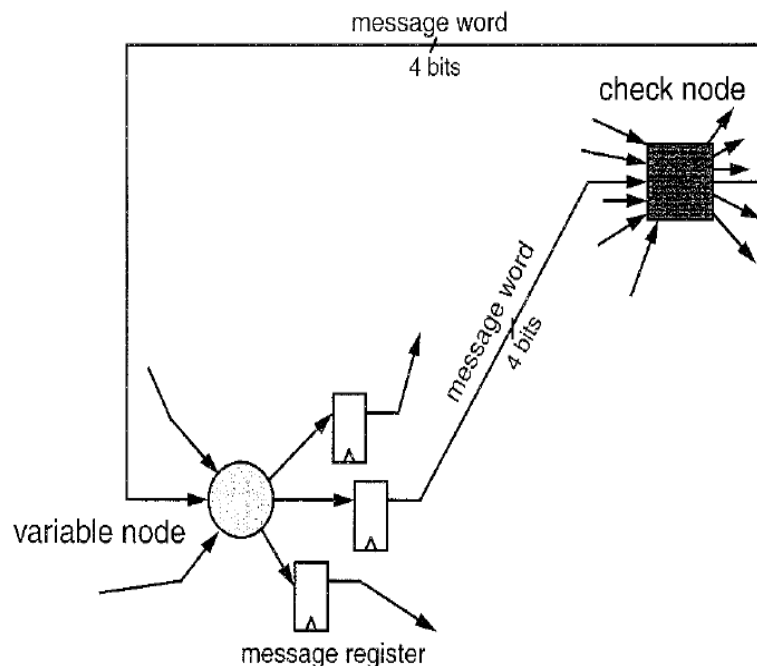
- ❶ Code de parité
- ❷ Principe des codes LDPC
- ❸ Architecture de décodeur LDPC
- ❹ En guise de conclusion



Architecture parallèle

● Exemple

⇒ Blanksby, Howland, « A 690-mW 1-Gb/s 1024-b Rate-1/2 Low-Density Parity-Check Code Decoder » (IEEE Trans. on Solid-State Circuits, 2002.)



● Avantages

- ⇒ performance: 1Gb/s 64 iterations
- ⇒ Power dissipation : 690mW
- ⇒ PER= $2 \cdot 10^{-4}$ @ 2,5 dB

● Drawback

- ⇒ Complex routing =>add oc CAD tool
- ⇒ Fixed code
- ⇒ Size : 52.5mm², 0.16μ Techno



Architecture série-parallèle

Puissance de calcul P_c : nombre de branche à traiter/ cycle d'horloge.

P_c dépend de : code (3,6)

N nombre de symboles du code ;	
E nombre de branche du code	$3N$
R rendement du code ;	$1/2$
D débit d'information (bit/s) ;	10 Mbit/s
N_{it} nombre moyen d'itération ;	20
f_{clk} réquence d'horloge .	100 MHz

$$P_c = \frac{EN_{it}}{N} [\text{branches/variable}] \times \frac{D}{f_{clk}R} [\text{variable/cycle}]$$

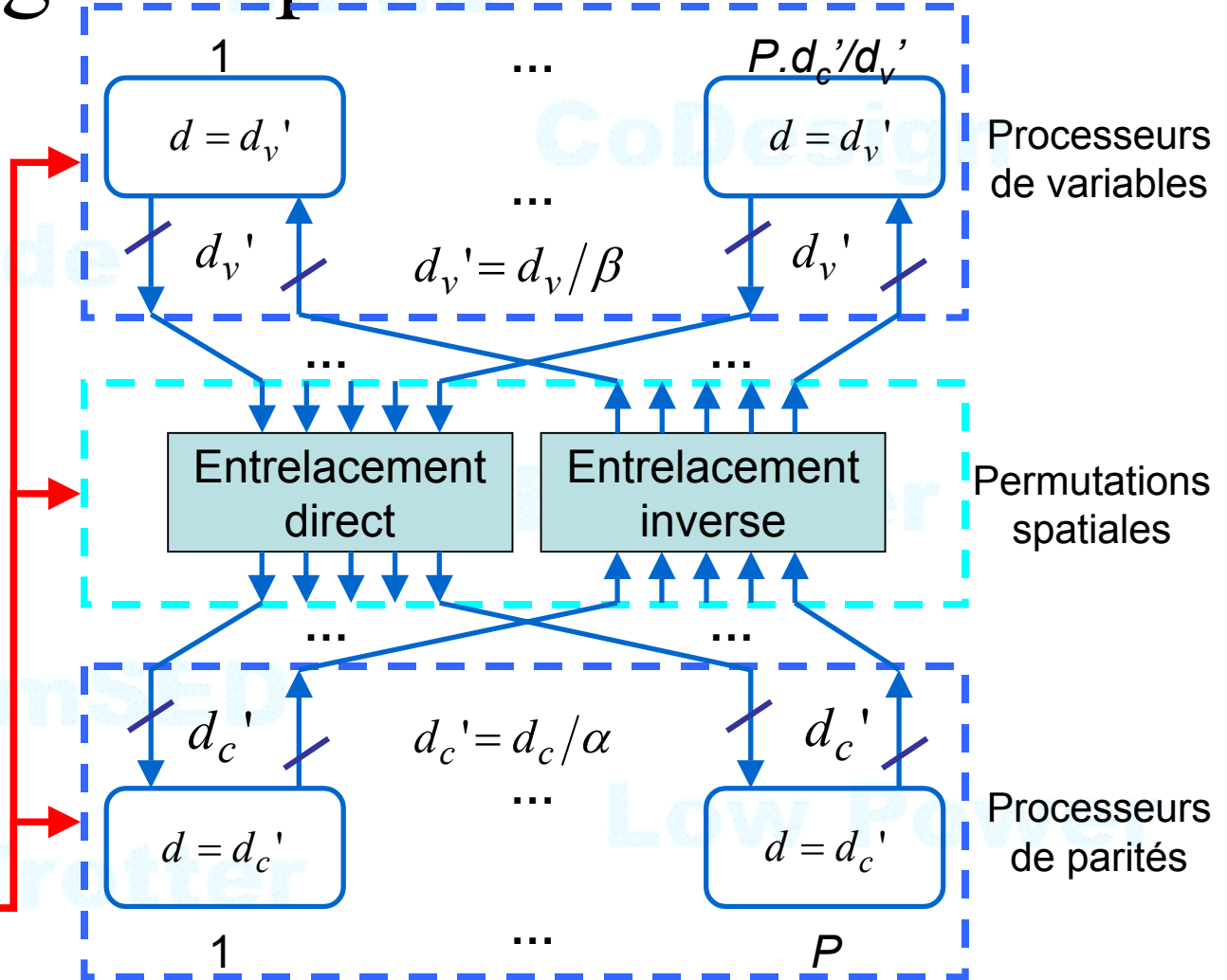
$$= 3 \times 20 \times \frac{10^7}{10^8 \times 0,5} = 12 [\text{branches/cycle}]$$



Degré de parallélisme

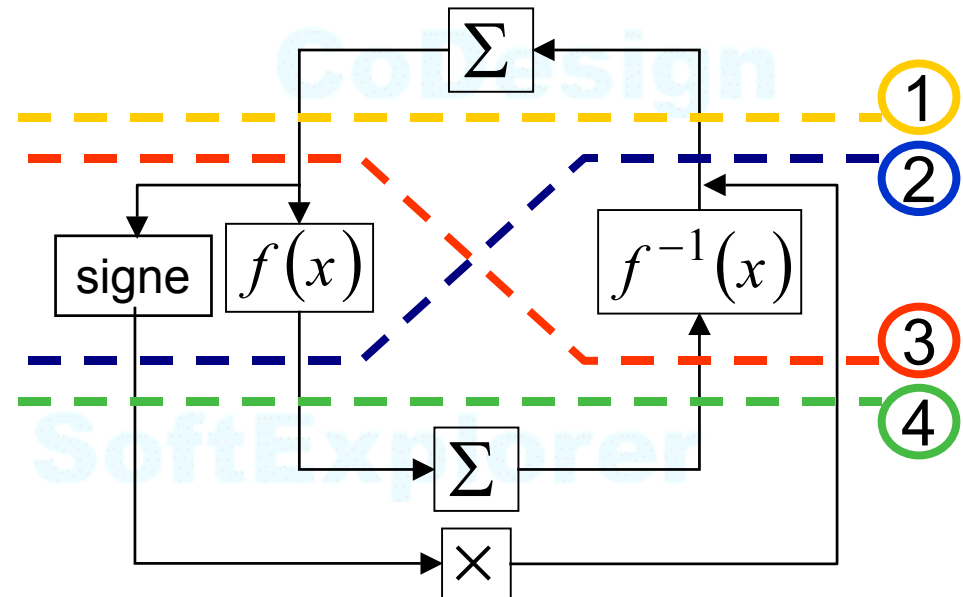
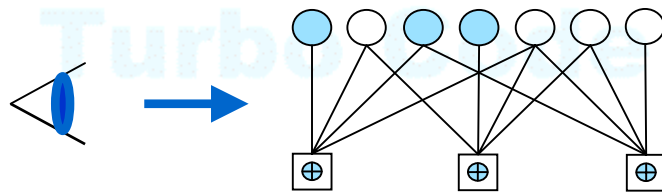
- P processeurs de parités
(α cycles)
- Processeurs de variables
(β cycles)
- Parallèle :
 $P=M, \alpha=1, \beta=1$
- Série :
 $P=1, \alpha=d_c, \beta=d_v$

Contrôle



Position du réseau d'interconnexion

Note : $\Phi=f=f^{-1}$

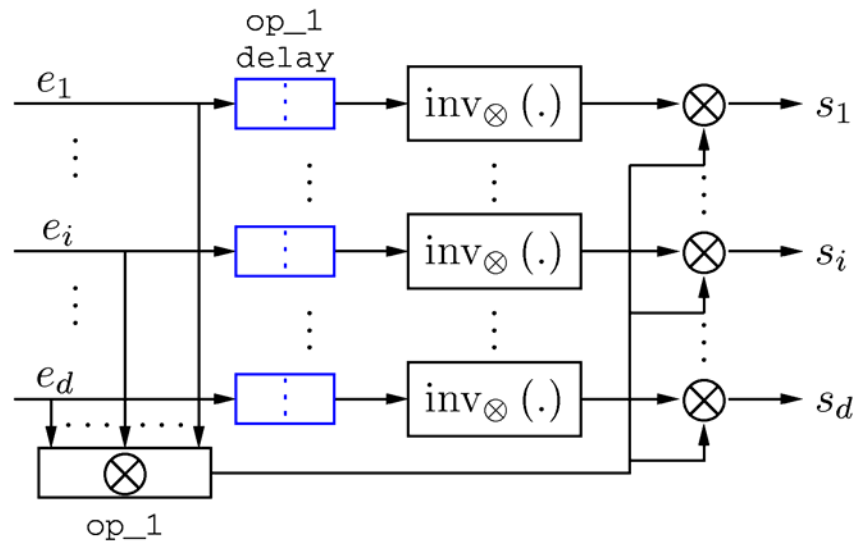


Position du réseau		①	②	③	④
Opérateur générique	Processeur de variable	Σ	$f \circ \Sigma$	$\Sigma \circ f^{-1}$	$f \circ \Sigma \circ f^{-1}$
	Processeur de parité	$f^{-1} \circ \Sigma \circ f$	$f^{-1} \circ \Sigma$	$\Sigma \circ f$	Σ
		\times (produit des signes)			

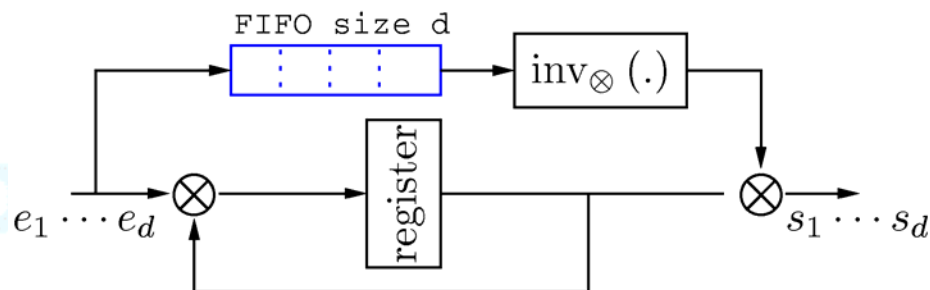


Opérateur Σ ou x

a) Mode compact



a) Mode compact et parallèle
(α ou $\beta = 1$)

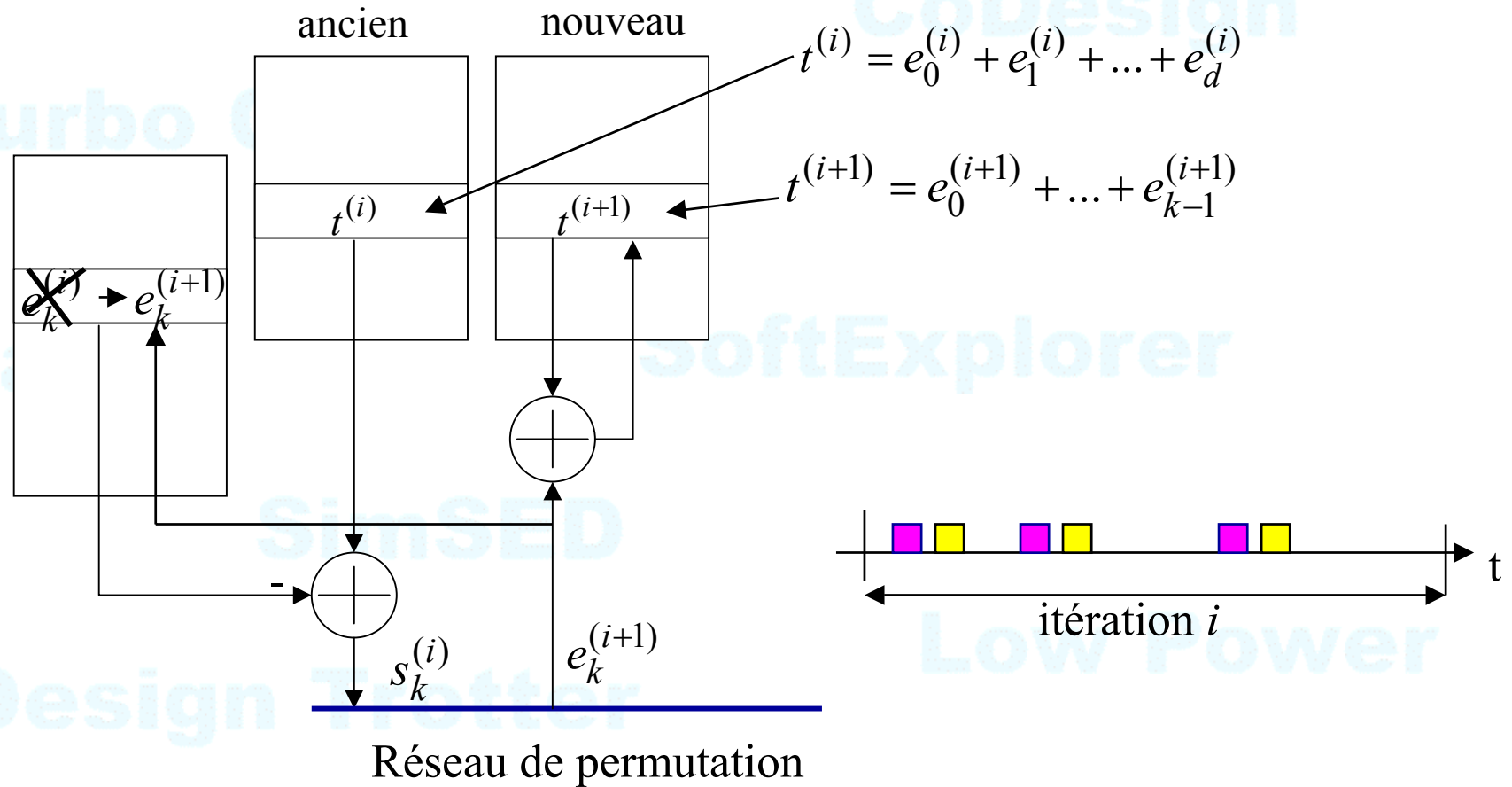


a) Mode compact et série
(α ou $\beta = d$)



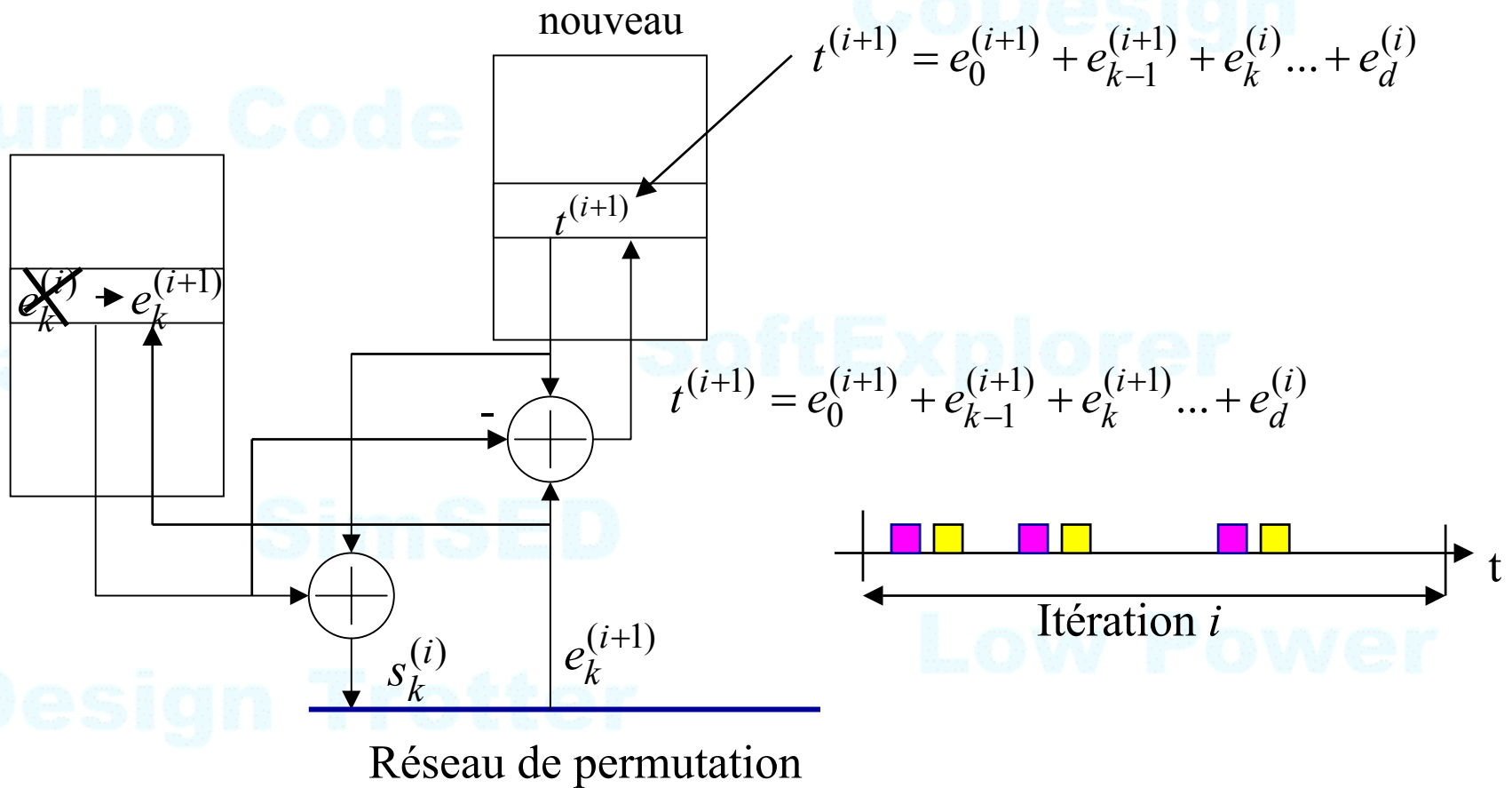
Opérateur Σ ou x

b) Mode distribué mise à jours différée



Opérateur Σ ou x

b) Mode distribué mise à jours immédiate



Modèle générique

Paramètres :

- Processeurs de nœuds:
 - ⇒ Architecture
 - ⇒ Position du réseau d'interconnexion (1,2,3,4)
- Parallélisme : P , α , β
- Contrôle des processeurs

Toutes les combinaisons sont possibles



• IP Combinaison des Contrôles

		Entrelacement (vertical)		VARIABLES			
				Compacte		Distribué	
				Mise à jour différée	Mise à jour immédiate		
P A R I T E S	Compacte	Inondation	Inondation parités	séquencement horizontal			
	Distribué	Mise à jour différée	Inondation variables	Contrôle des branches			
Mise à jour immédiate		Séquencement vertical					



- Etat de l'art

		VARIABLES		
		Compacte	Distribué	
			Mise à jour différée	Mise à jour immédiate
P A R I T E S	Compacte	Blanksby (parallèle)	Boutillon, Chen, Zhang,	Mansour
	Distribué	Mise à jour différée	Nouveau	
		Mise à jour immédiate	Nouveau (algo Fossorier)	



- Etat de l'art : taille mémoire

E : Nombre de branches

F() : compression

		VARIABLES		
		Compacte	Distribué	
			Mise à jour différée	Mise à jour immédiate
P A R I T E S	Compacte	$N + E$	$3N + f(E)$	$N + f(E)$
	Distribué	Mise à jour différée	$N + 2M + E$	-
		Mise à jour immédiate	$N + M + E$	-



PLAN

- ❶ Code de parité
 - ❷ Principe des codes LDPC
 - ❸ Architecture de décodeur LDPC
- ❸ En guise de conclusion



Conclusion

Présentation d'un modèle d'architecture générique
=> classement des architectures existantes

Proposition d'architectures nouvelles pour des séquençements
verticaux ou horizontaux
=> convergence en 1/2 moins d'itération
=> architecture requérant moins de mémoire.



Conclusion sur LDPC

Pour les codes de taille importante ($n > 10^4$) les LDPCs irréguliers ont des meilleures équivalentes à celle des Turbo-Codes

- moins de 0,1 dB de la limite de Shannon pour $n = 10^6$
(travaux de Mac-Kay, Richardson).

De rapide progrès dans la construction des matrices et des algorithmes de décodage.

Travaux en cours:

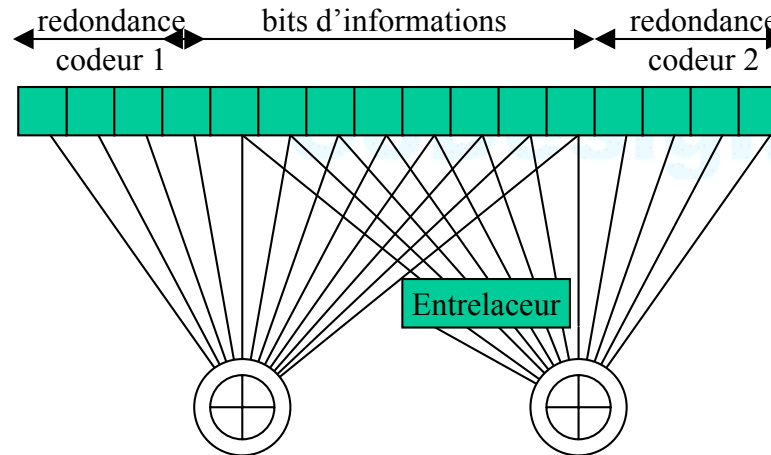
LDPC sur $GF(2^q)$

Digital fountain



LDPC vs Turbo-Code

Une autre vision d'un turbo-code



LDPC et Turbo-code sont aux deux extrémités du spectre des graphes de Tanner

Il existe des solutions intermédiaires (turbo-code produit par exemple)

Compromis performances-complexité : un débat complexe et non tranché (DVB-S2)



Nouvelle technique de décodage

Utilisation de décodeur purement analogique :

- propagation des messages de façon continue
- faible sensibilité des décodeurs à la précision
- les équations du transistor identiques à celle du MAP !

Gain en consommation et vitesse de décodage d'un facteur 10 à 1000 !

=> Solution très prometteuse...



Exemple construction matrice/architecture conjoint

Matrice prototype : $H_i =$

$$\begin{matrix} \xleftarrow{M=mP} & & \xrightarrow{M=mP} \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & \textcircled{1} & 0 & 1 & 0 & \textcircled{0} & 1 & 1 \end{pmatrix} & \begin{matrix} \uparrow \\ \downarrow \\ \uparrow \\ \downarrow \end{matrix} & N = nP \end{matrix}$$

$$\Pi_{i,j}(I_p) = \begin{matrix} \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} & \begin{matrix} \uparrow \\ \downarrow \\ \uparrow \\ \downarrow \\ \uparrow \\ \downarrow \end{matrix} & P \\ \xleftarrow{P} & & \xrightarrow{P} \end{matrix}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Très peu d'information pour mémoriser la matrice



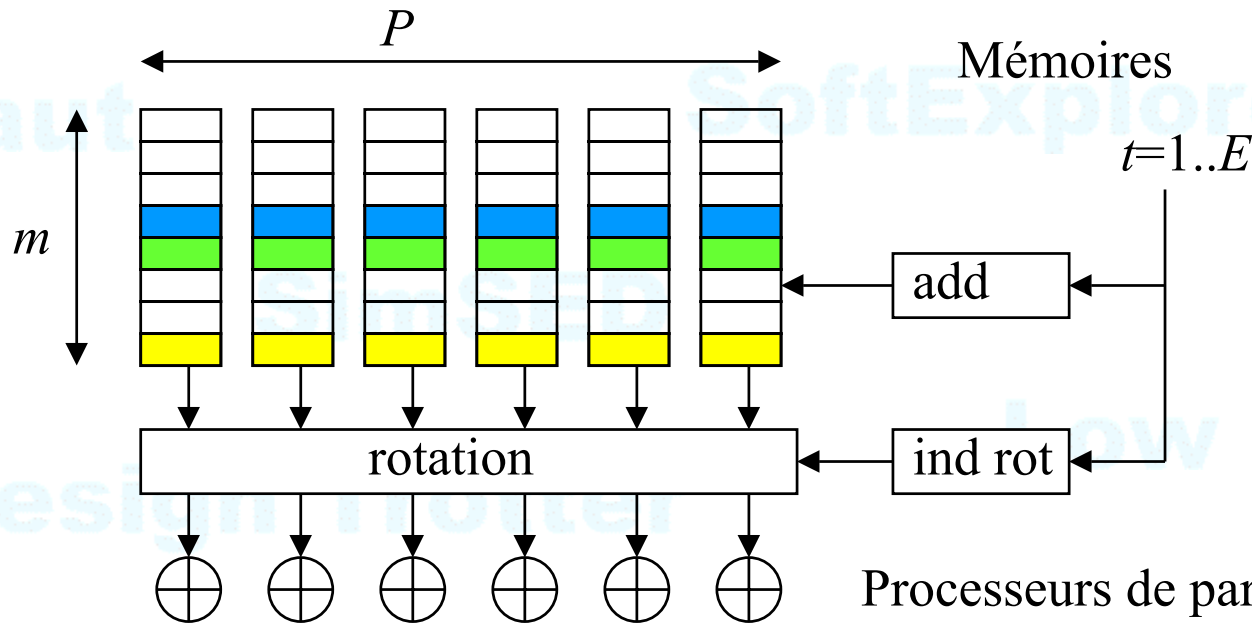
Exemple construction matrice/architecture conjoint

Matrice prototype : $H_i =$

1	0	0	1	1	0	0	0
0	1	1	0	1	1	0	0
1	0	1	0	0	1	1	0
0	1	0	1	0	0	1	1

$M = mP$

$N = nP$



Processeurs de parité : calcul sérié

