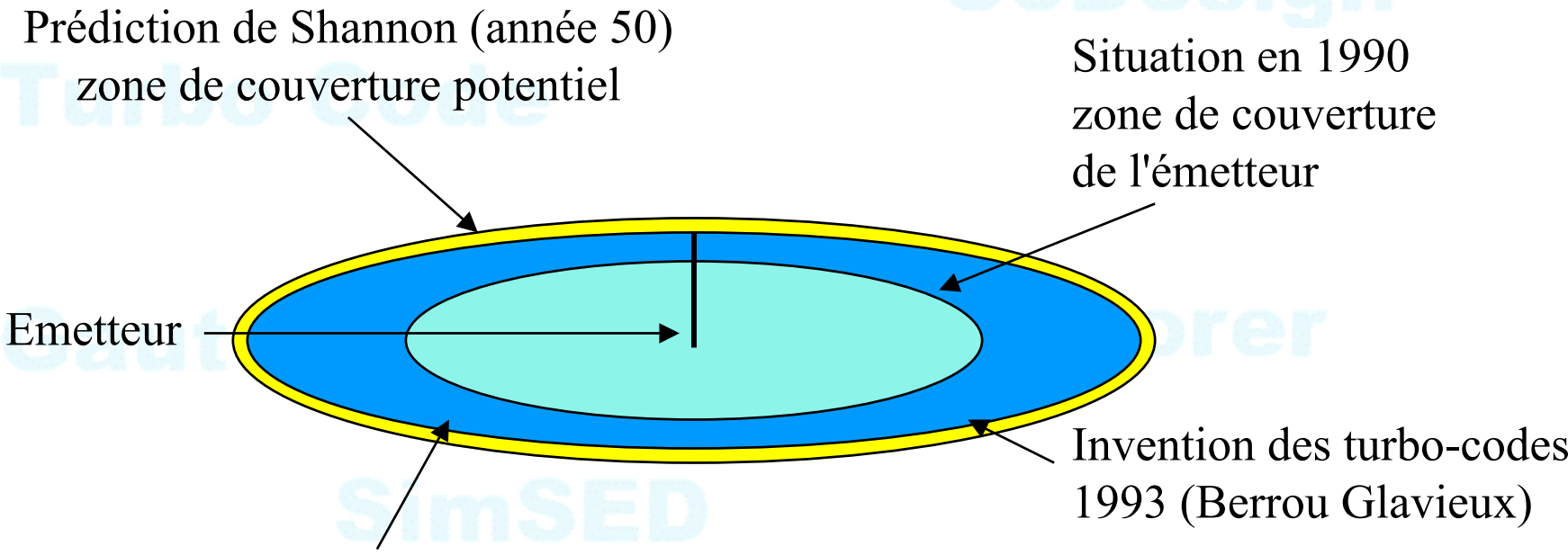
Architecture générique de décodage de code LDPC

F. GUILLOUD
E. BOUTILLON
guilloud@enst-bretagne.fr
emmanuel.boutillon@univ-ubs.fr

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).

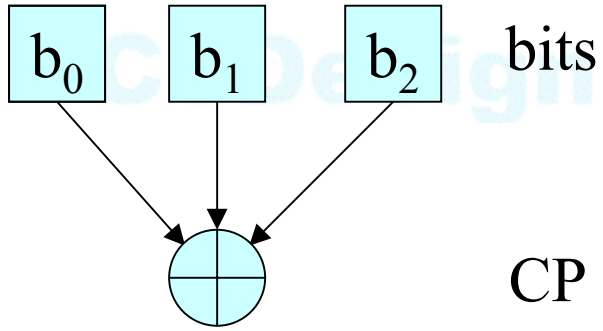
Gallager	LDPC	1962
Berrou, Glavieux	Turbo Codes	1993
MacKay	LDPC	1995
Wiberg	Graphes	1995
Richardson, Urbanke	Density Evolution	2001

Premières architectures de décodeurs	2001
Standard DVB-S2	2003

PLAN

Parity check

Parity check (3,2,1) :

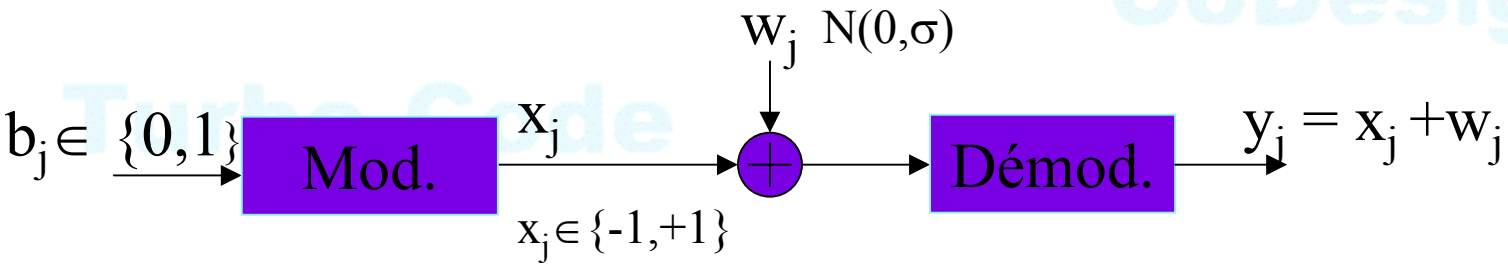


(b_0, b_1, b_2) codeword <=> Sum of 1 = 0 mod 2.

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:

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

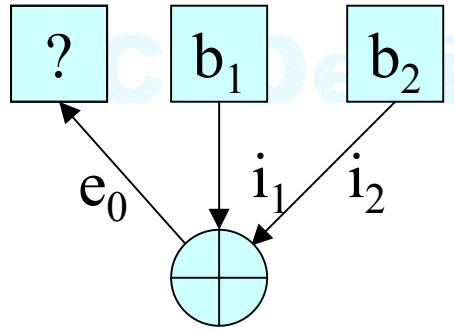
que l'on exprime aussi par le "log likelihood ratio":

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

Note: sign(i_j) => 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:

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

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

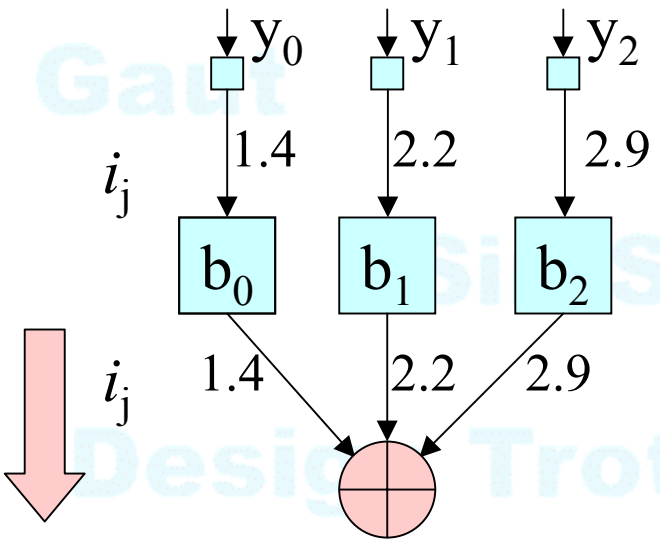
$$e_0 = i_1 \oplus i_2 = \Phi^{-1}(\Phi(i_1)+\Phi(i_2))$$
$$\Phi^{-1}(x) = \Phi(x) = -\ln(\tanh(x/2))$$

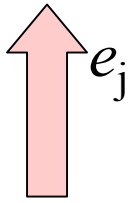
e_0 est appelé information *extrinsèque*

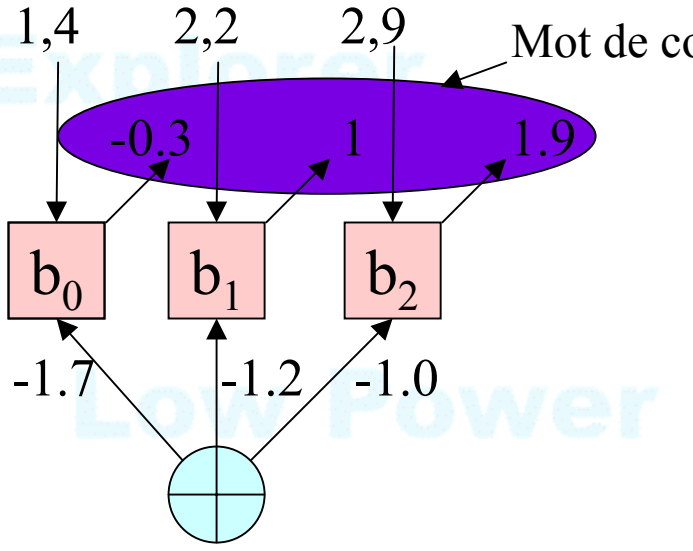
Décodage d'un code de parité

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

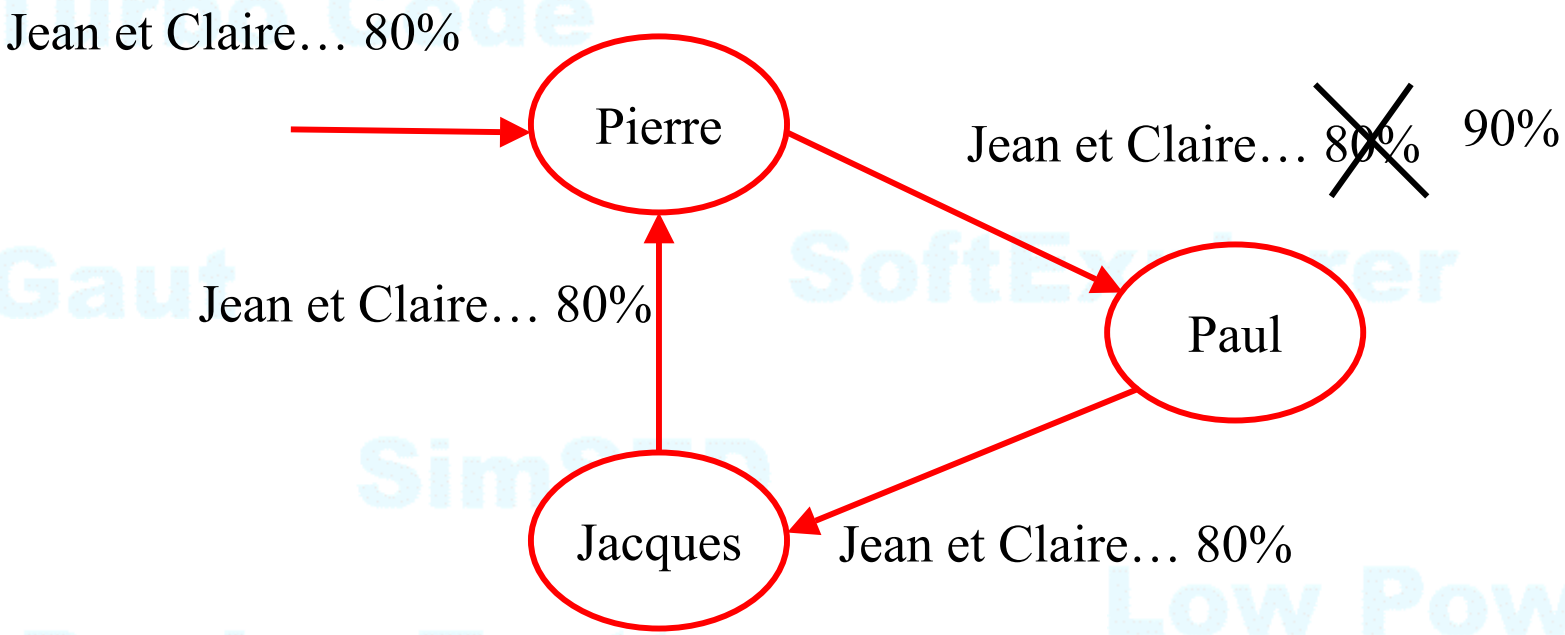
Le processus est symétrique pour tous les bits.





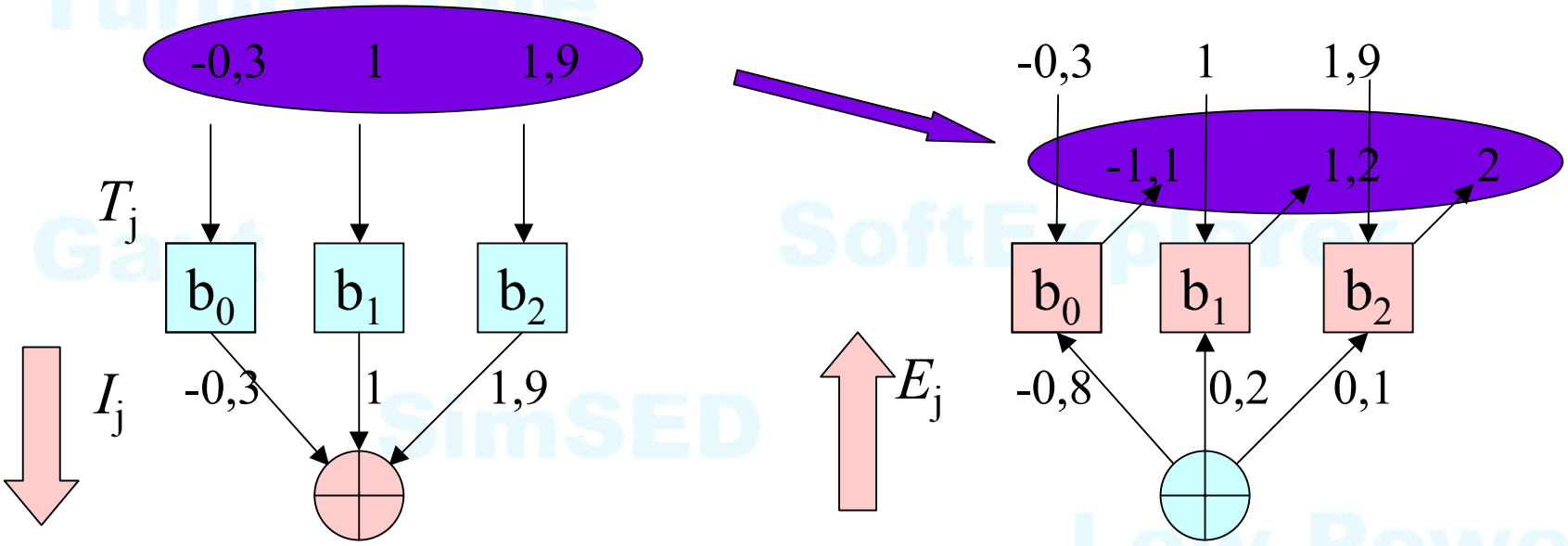


Conseil : éviter les auto-confirmations



Note: itération

Si le processus itère de nouveau, il y a autoconfirmation

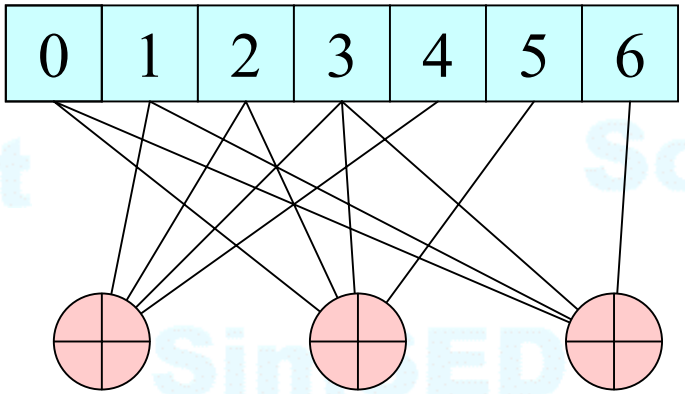


Et le système diverge...

PLAN

LDPC Code

Peut être défini par un graph bi-partite



Bit b_i

Branches

Parity checks (PC)

$(b_0, b_1, \ldots, b_6)$ mot de code <=> toutes les PC sont respectées

Pourquoi le nom de LDPC ?

Représentation algébrique :

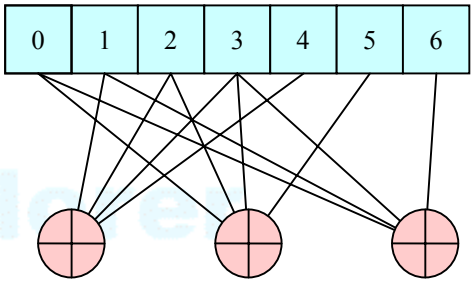
$X = (x_0, x_1, \ldots, x_6)^T$ mot de code si $H.X = 0$

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

P=nombre de PC

N=nombre de bits

Matrice de parité

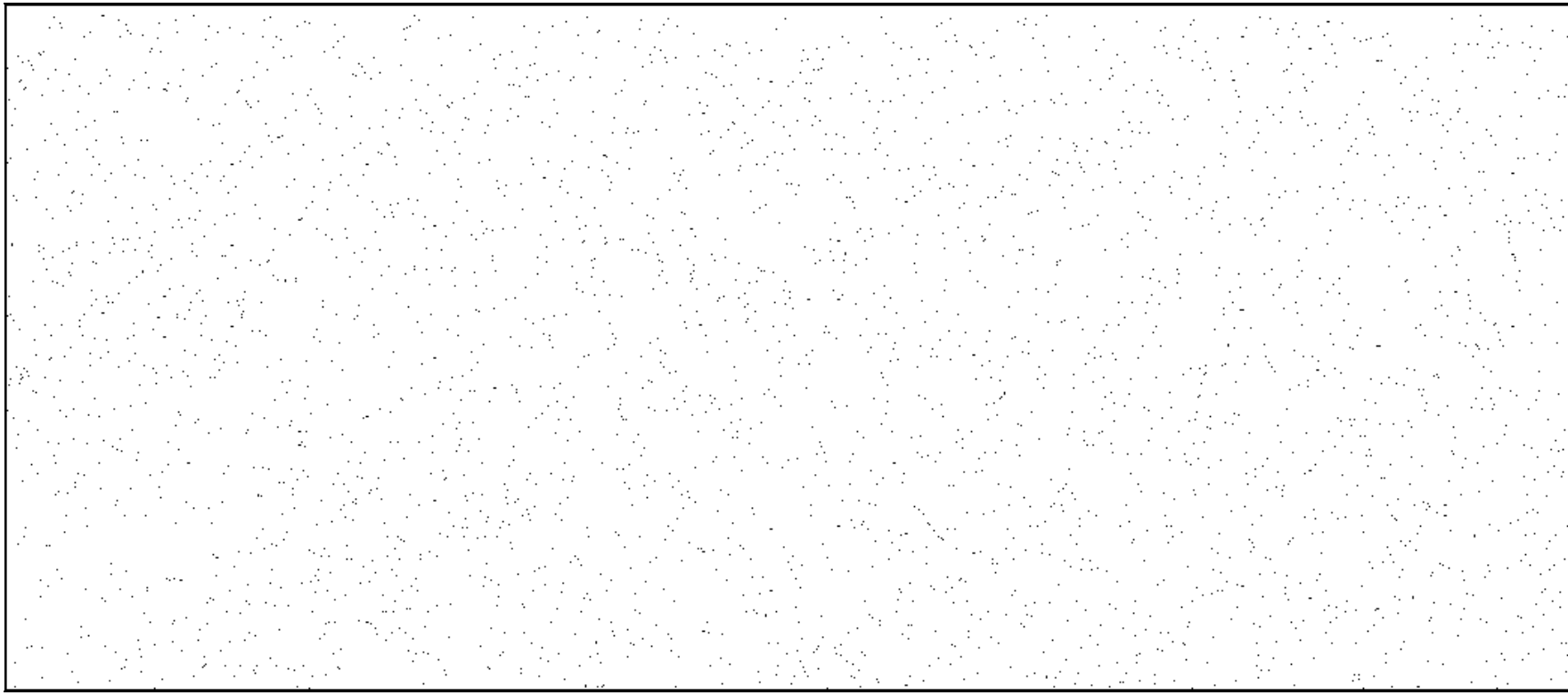


Nombre de 1 sur une ligne = dc=nombre de bit associé au PC

Nombre de 1 sur une ligne = dv=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 ?

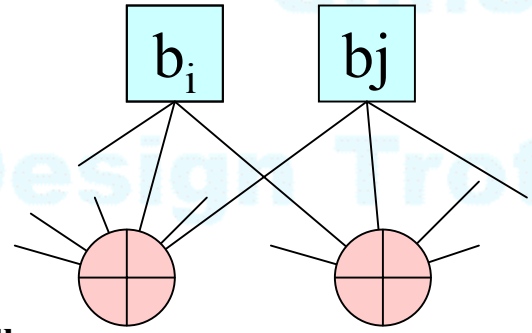
Comment construire un bon code LDPC ?

Application => taille du code et rendement

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

exemple:
90 % bits => dv =3 branches, 10 % bits dv=> 12 branches
70 % PC => dc = 6 branches, 30 % PC => dc = 8 branches

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

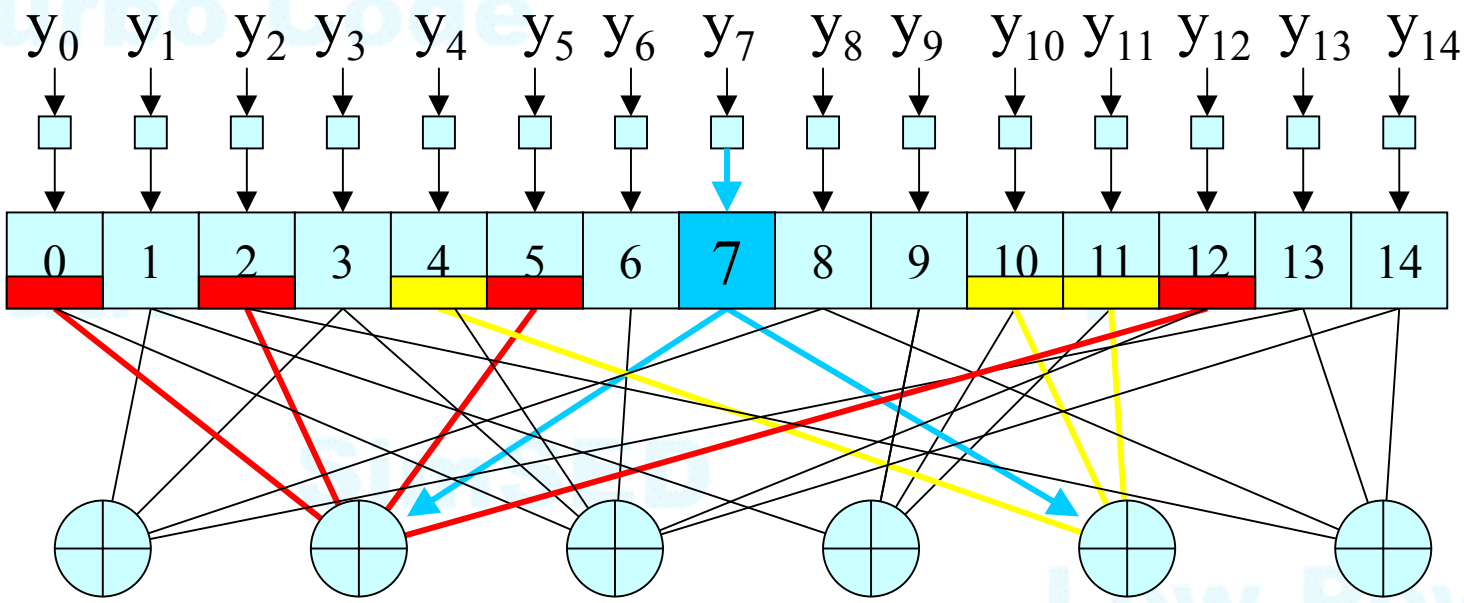


…et on obtient un bon code

Algorithme à propagation de croyance

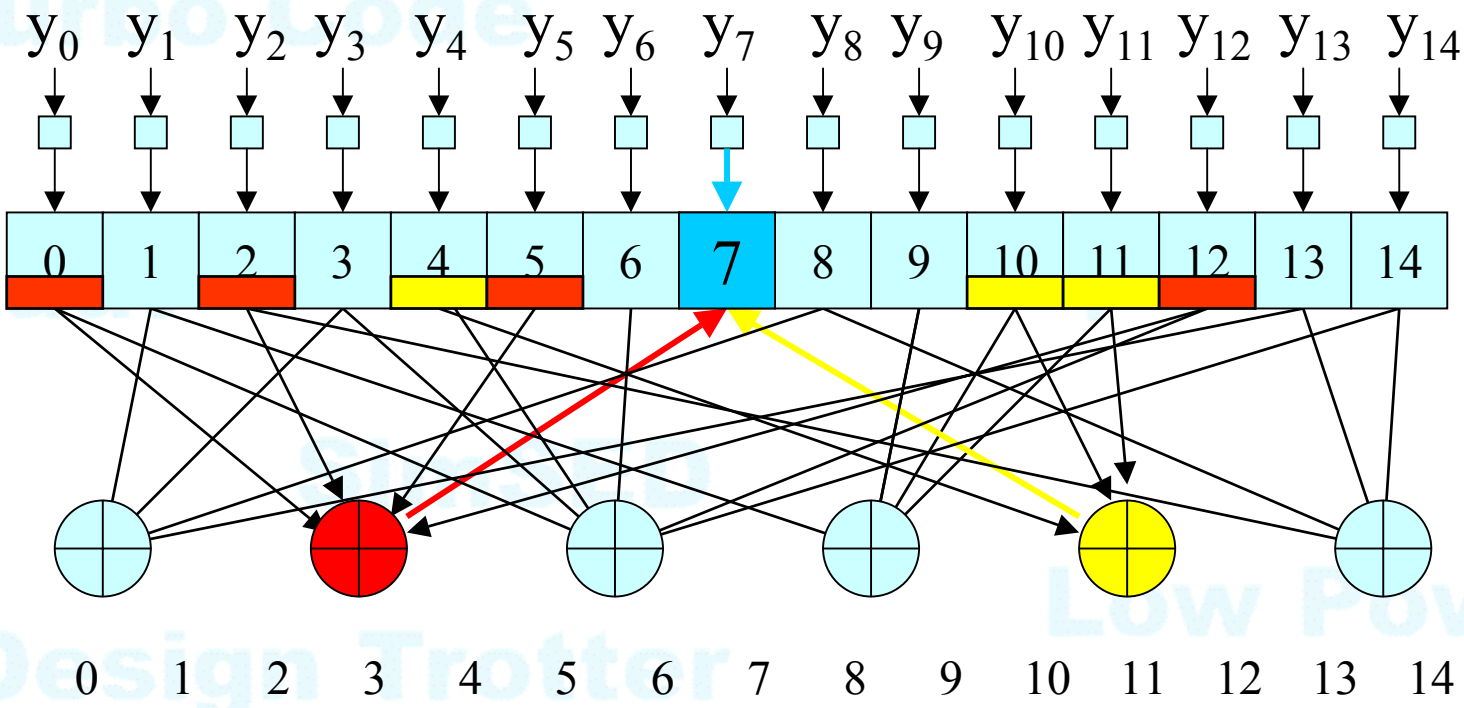
- Step1: calcul du LLR des bits reçus (information intrinsèque)
- Step2: Message bit->parité + traitement parité
- Step3: Message parité->bit + traitement bit
- Step4: répéter 2 et 3 jusqu'au décodage correct ou "max iteration".

Décodage itératif



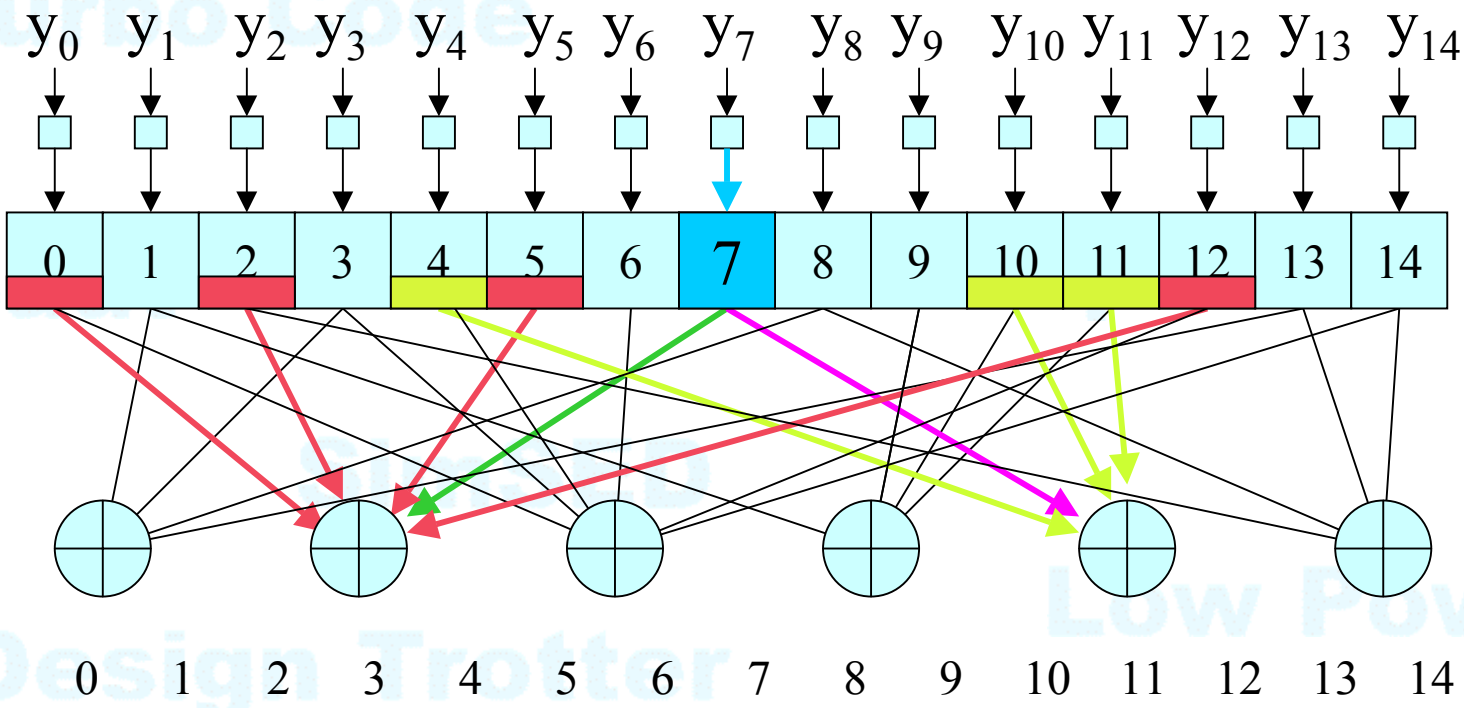
Première itération: message bit-> parité

Décodage itératif



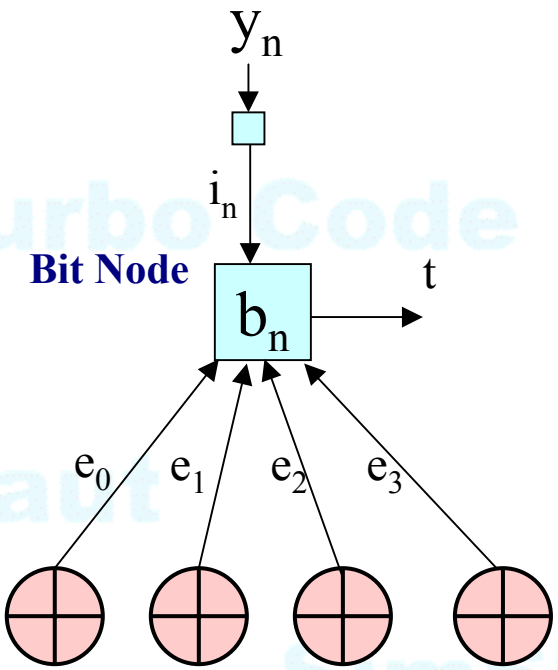
Première itération: message parité-> bit

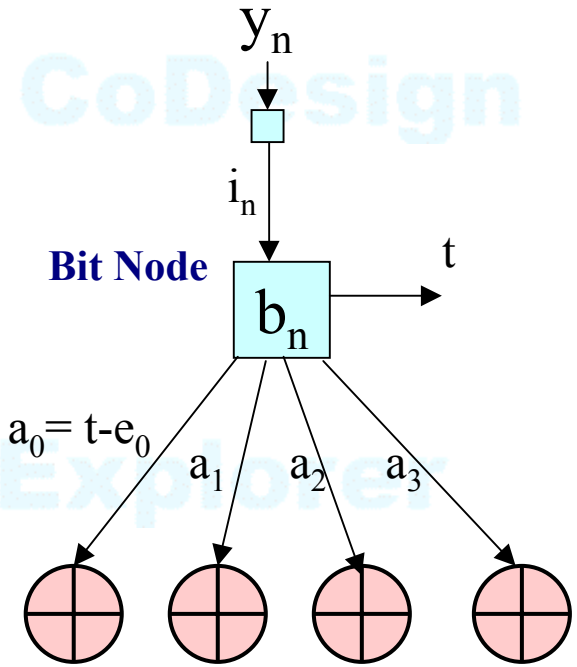
Décodage itératif



Deuxième itération: message parité -> bit

Calcul d'un nœud de variable

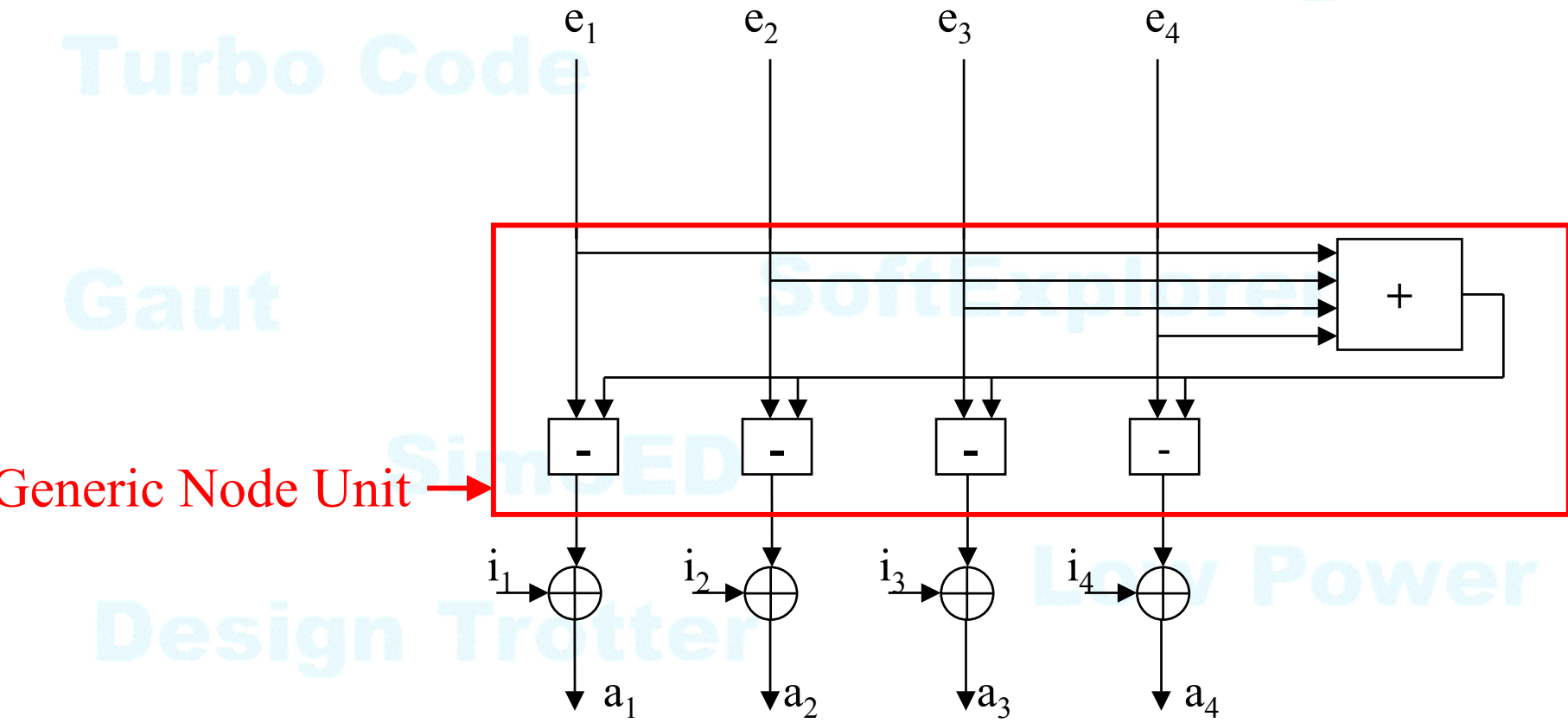




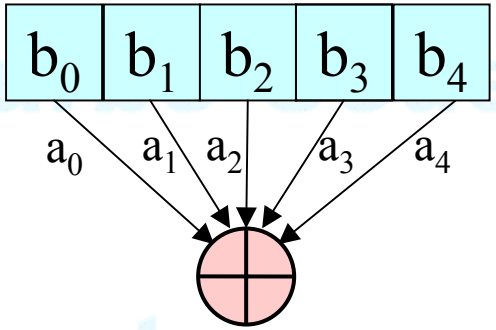
$$t = i_n + \sum_m e_m$$

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

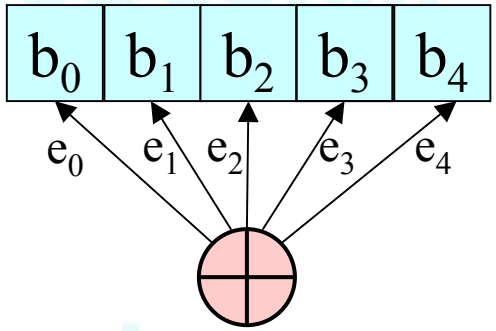
Calcul Variable : graphe de calcul



Calcul d'un nœud de parité



Check Node



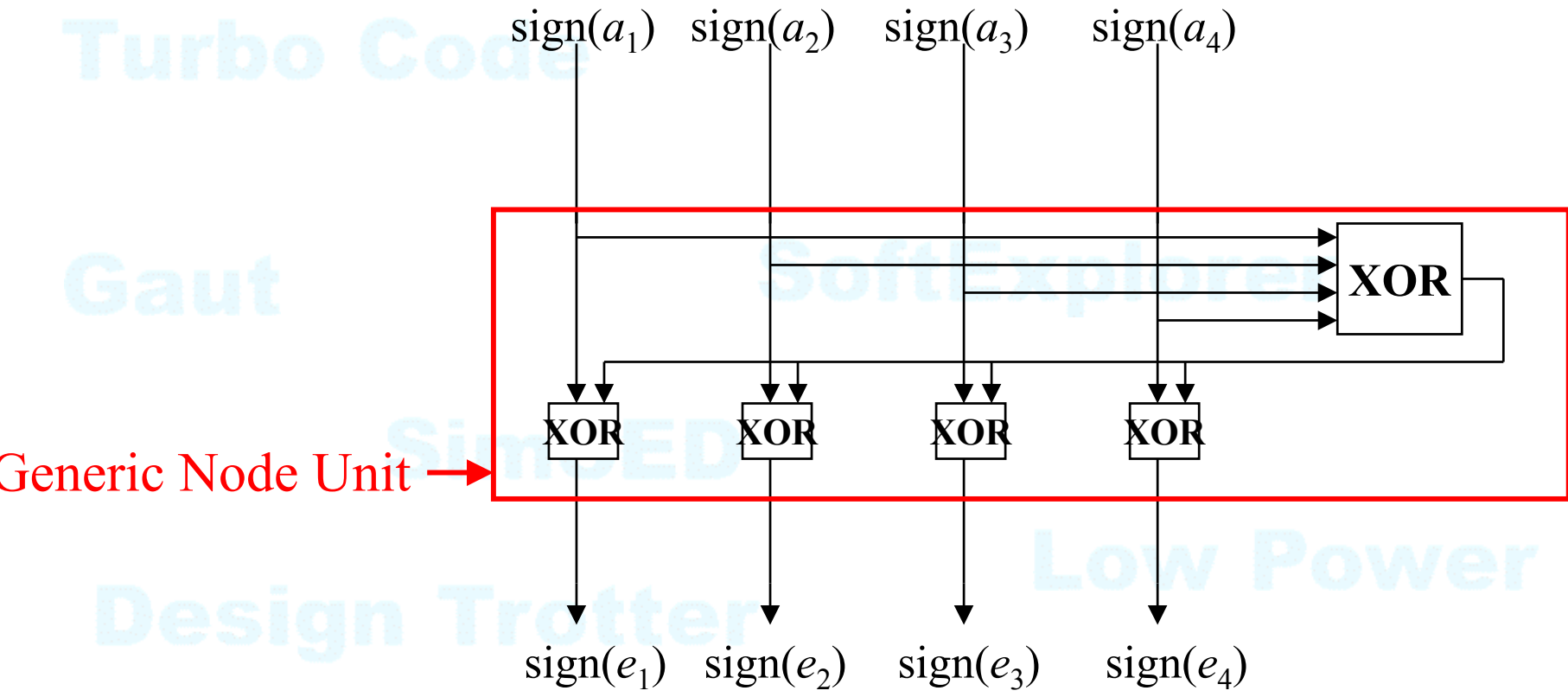
Check Node

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

Différentes méthodes de calcul => exactes : directe ou fréquentielle

=> approchées : min-sum et ses variant

Calcul parité: méthode fréquentielle
a) calcul du signe



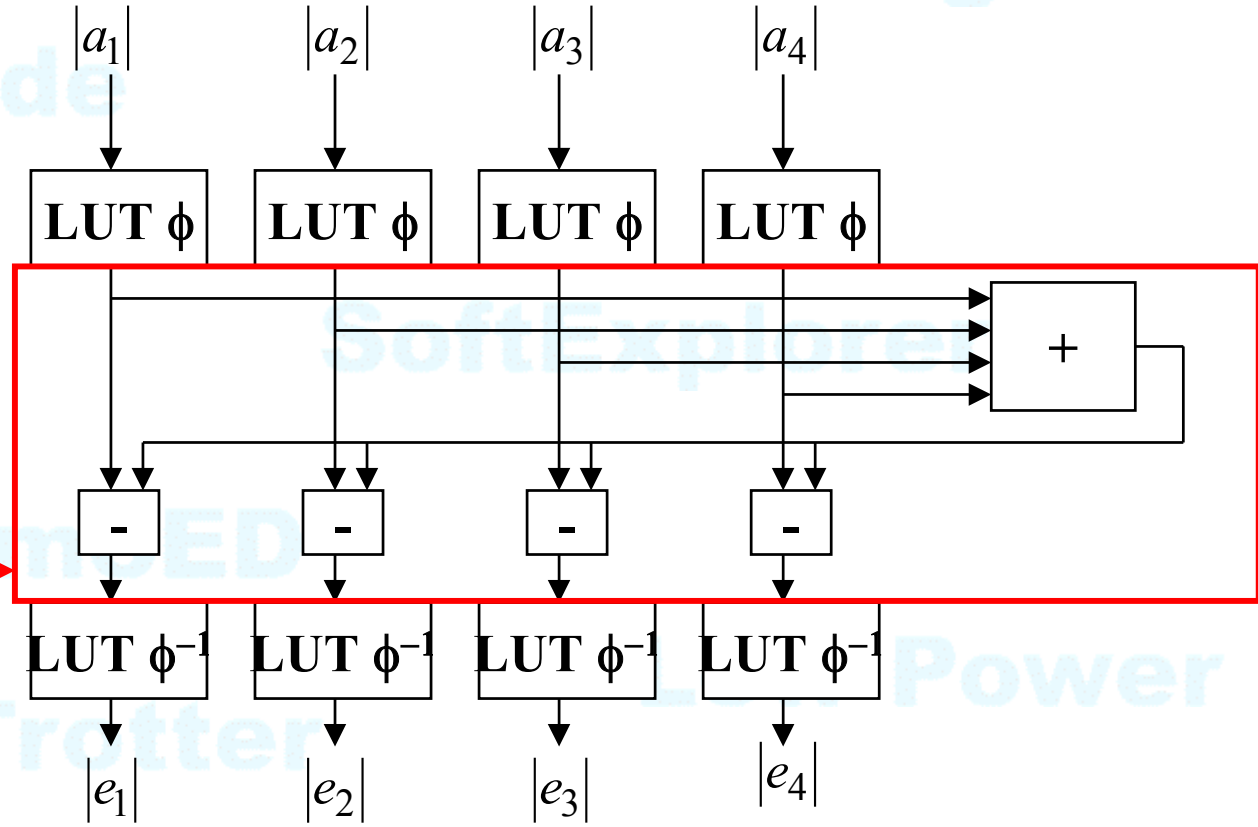
Calcul parité: méthode fréquentielle
b) calcul du module

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

avec

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

Generic Node Unit →

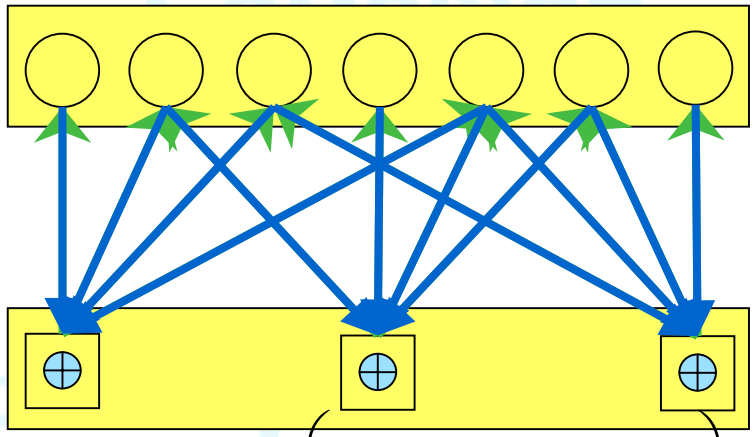


Algorithme BP

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

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

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

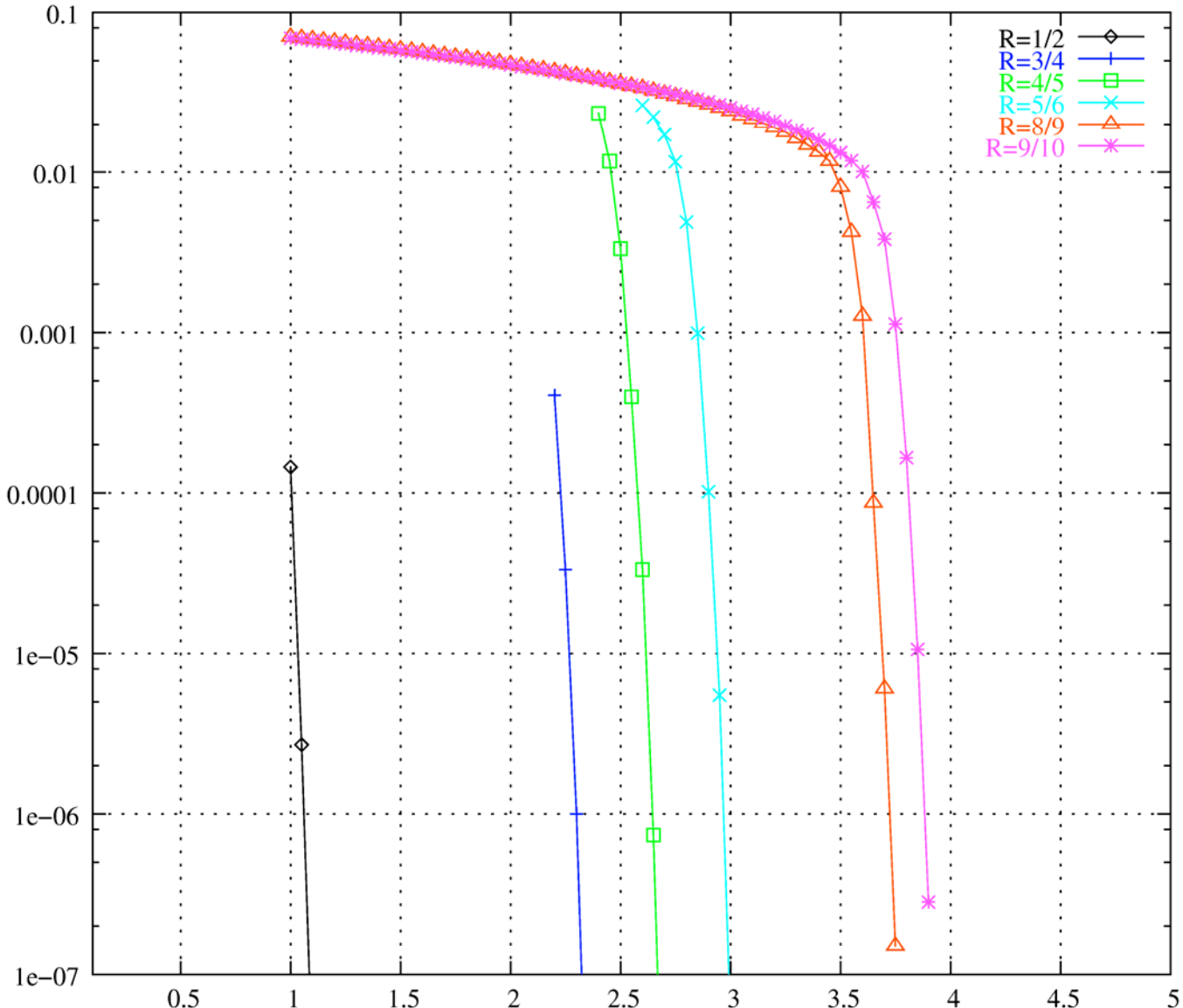


$$e_{m,n} = \Phi^{-1}\left(\sum_{n' \in N(m) \setminus n} \Phi(a_{m,n'}) \right)$$

Traitement des variables → Propagation V → P

Propagation P → V ← Traitement des parités

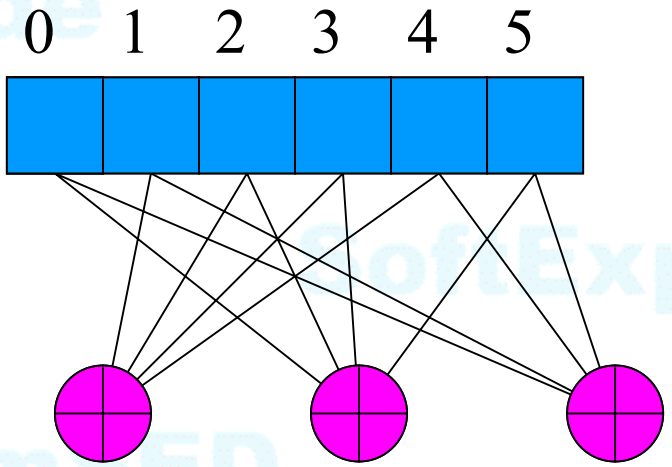
Performances DVB-S2



64k
50 itérations

LDPC + BCH

Sequencement par inondation (Méthode classique)



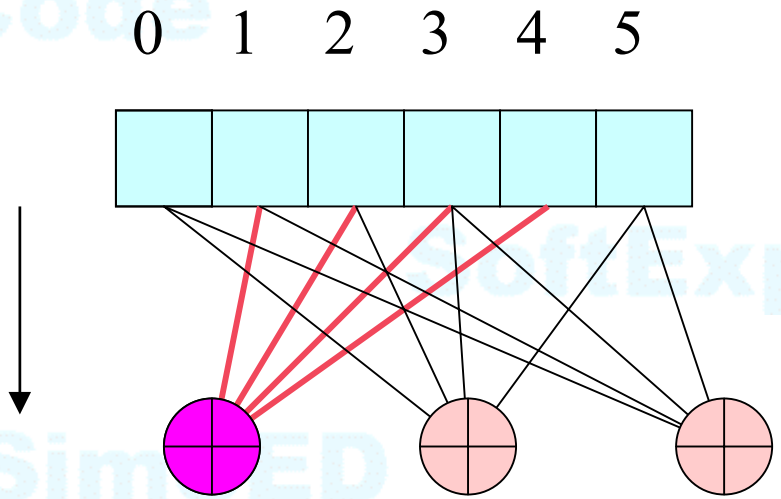
Flooding

=> postulat faux : unique séquencement efficace

Sequencement horizontal (Mansour)

011110
101101
110101

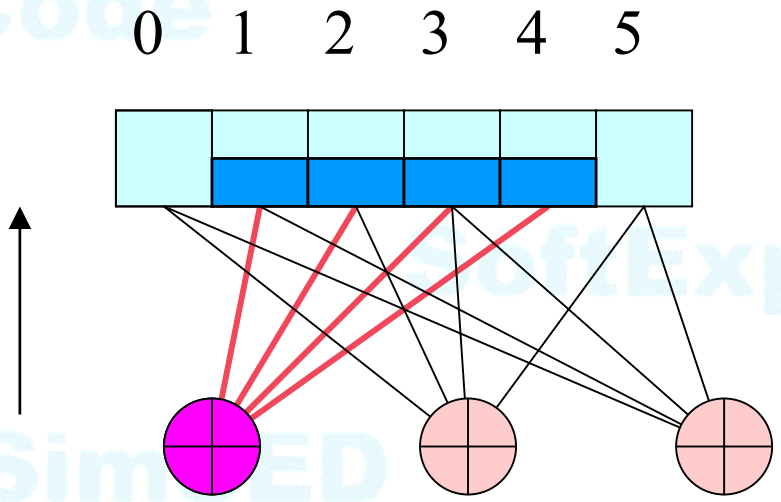
Matrice LDPC



Sequencement horizontal (Mansour)

011110
101101
110101

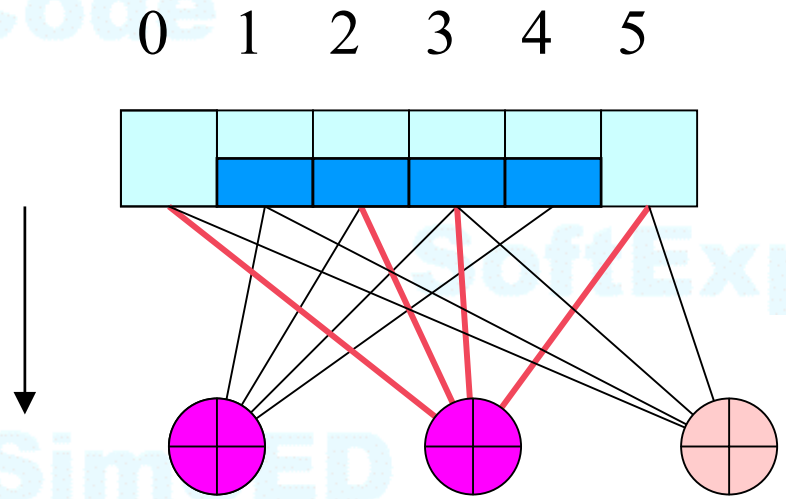
Matrice LDPC



Sequencement horizontal (Mansour)

011110
101101
110101

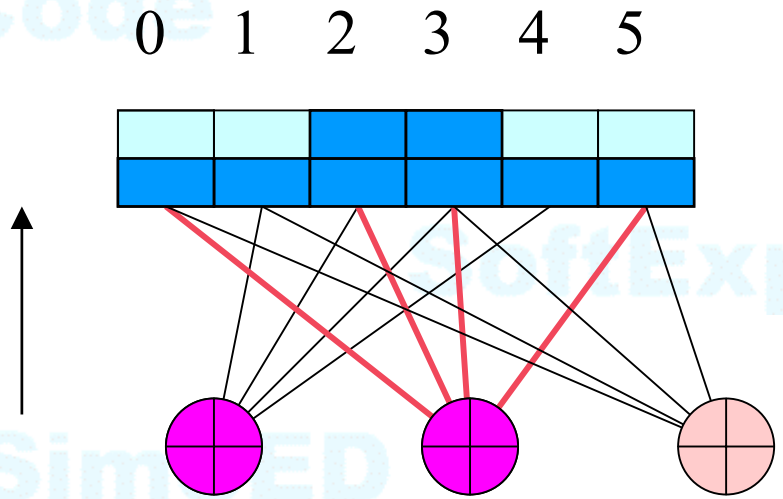
Matrice LDPC



Sequencement horizontal (Mansour)

011110
101101
110101

Matrice LDPC

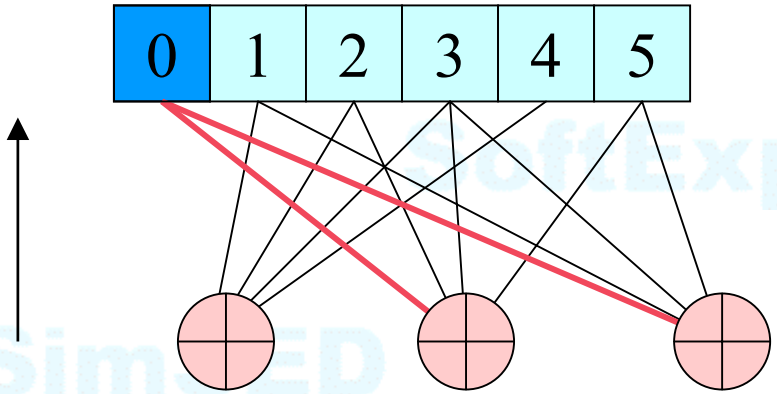


Convergence en moitiée moins d'itération

Sequencement vertical (Fossorier)

011110
101101
110101

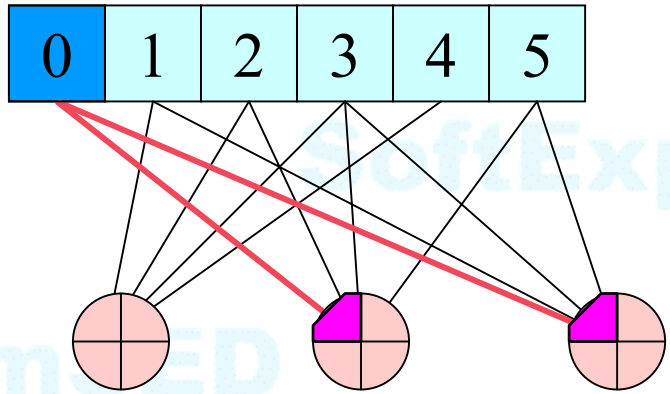
Matrice LDPC



Sequencement vertical (Fossorier)

011110
101101
110101

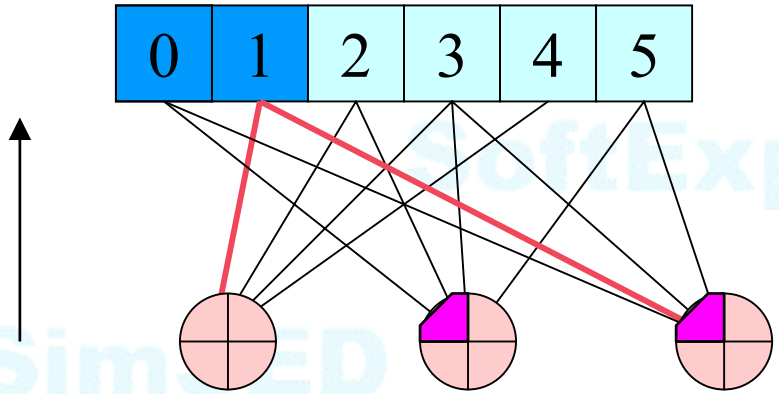
Matrice LDPC



Sequencement vertical (Fossorier)

011110
101101
110101

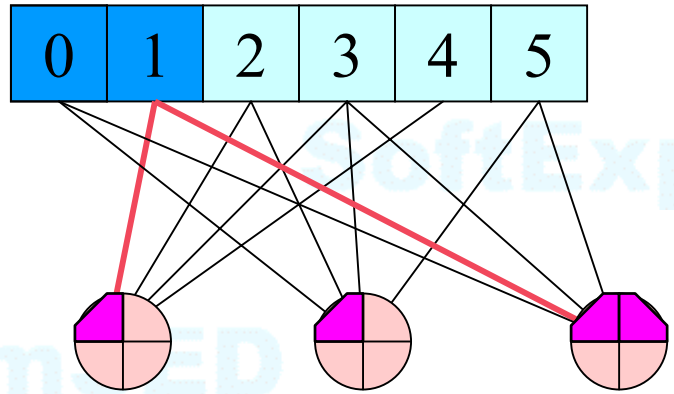
Matrice LDPC



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 moitiée moins d'itération !

Problème séquencement H et V

Trouver une architecture adaptée...

PLAN

❶ Code de parité

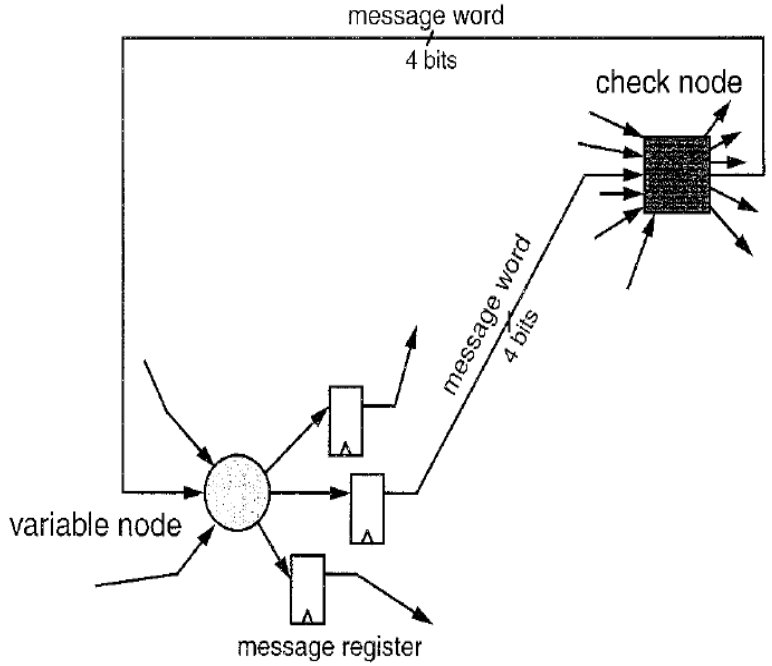
❷ Principe des codes LDPC

❸ Architecture de décodeur LDPC

❸ 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 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

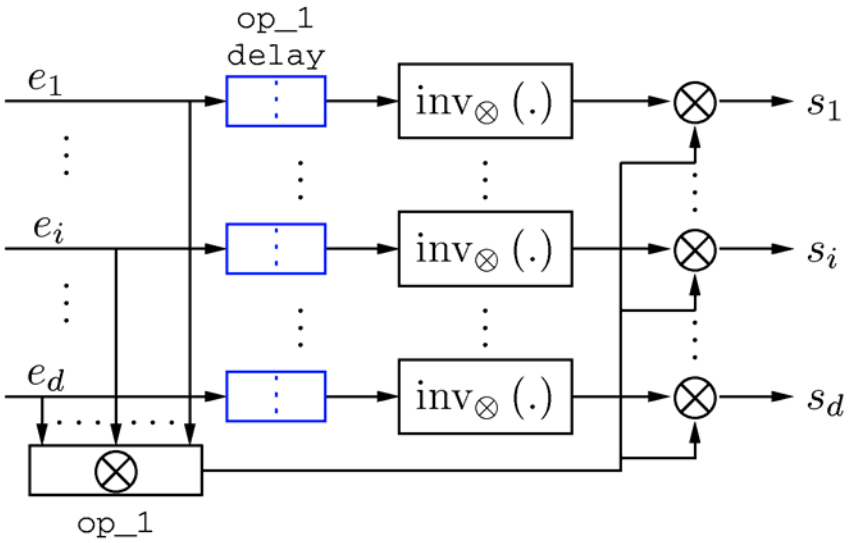
Pc dépend de :

	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

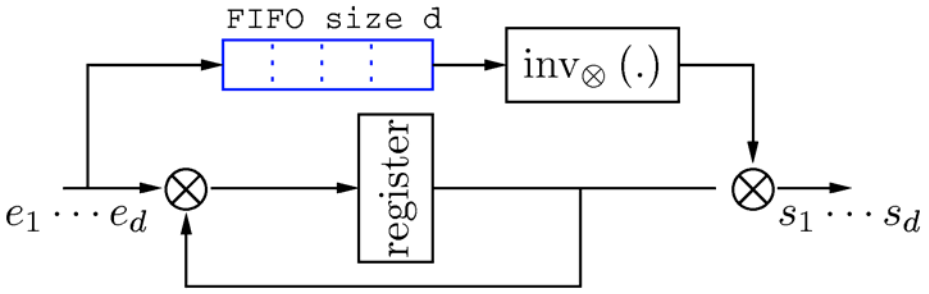
$$P_c = \frac{EN_{it}}{N}\,[\text{branches/variable}] \times \frac{D}{f_{clk}R}\,[\text{variable/cycle}]$$

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

Opérateur Σ ou x
a) Mode compact

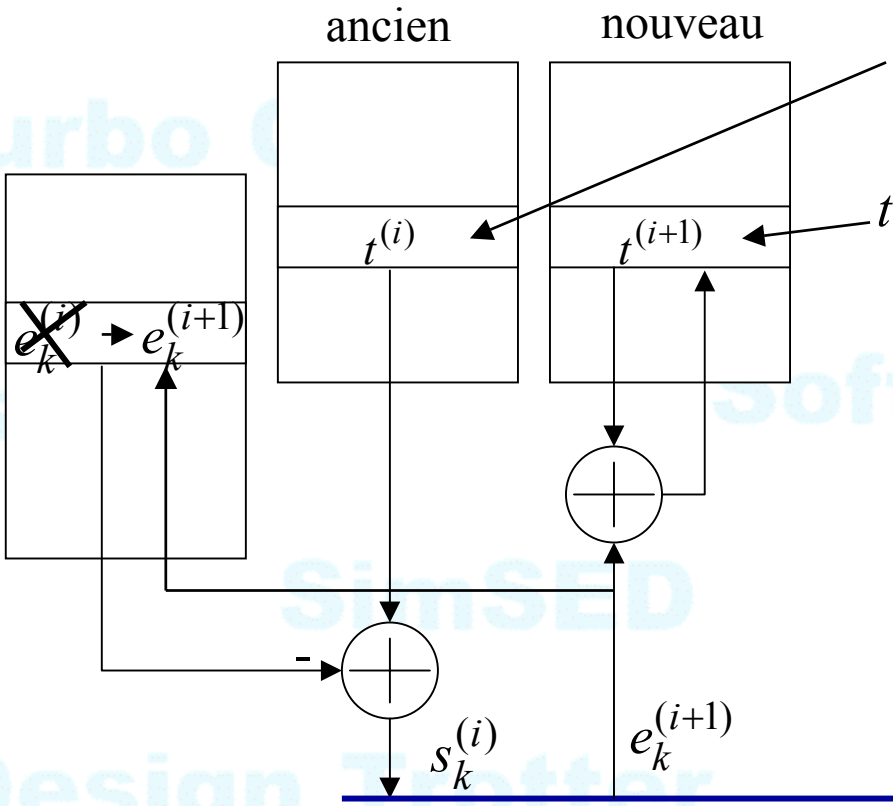


a) Mode compact et parallèle
(calcul en α = 1 cycle d'horlog



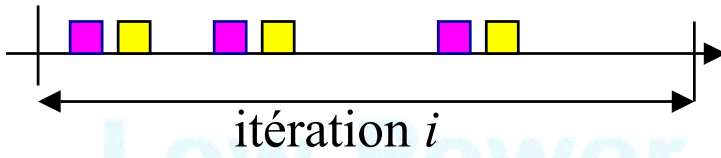
b) Mode compact et série
(calcul en a = dv cycle d'horlog

Opérateur Σ ou x
b) Mode distribué mise à jours différée

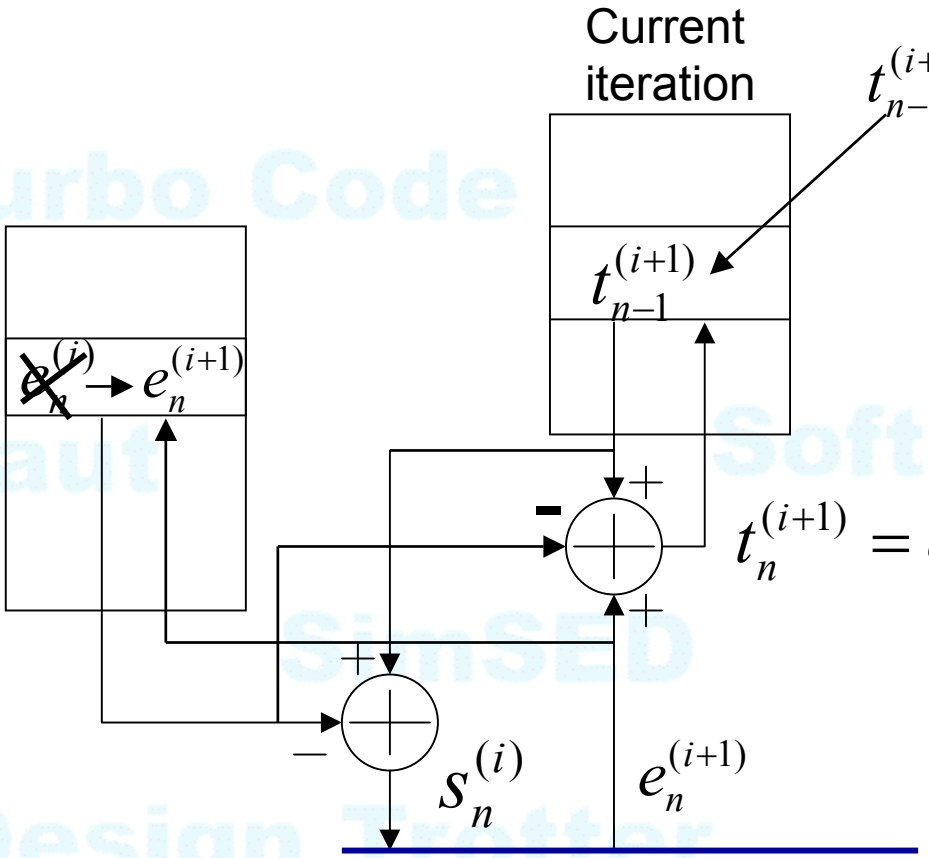


$$t^{(i)} = e_0^{(i)} + e_1^{(i)} + \ldots + e_d^{(i)}$$

$$t^{(i+1)} = e_0^{(i+1)} + \ldots + e_{k-1}^{(i+1)} + e_k^{(i+1)}$$



Implémentation du Generic Node Unit : b) Mode distribué, mis à jour immédiate



$$t_{n-1}^{(i+1)} = e_1^{(i+1)} + e_{n-1}^{(i+1)} + e_n^{(i)} + \ldots + e$$

$$t_n^{(i+1)} = e_1^{(i+1)} + e_{n-1}^{(i+1)} + e_n^{(i+1)} + \ldots + e$$

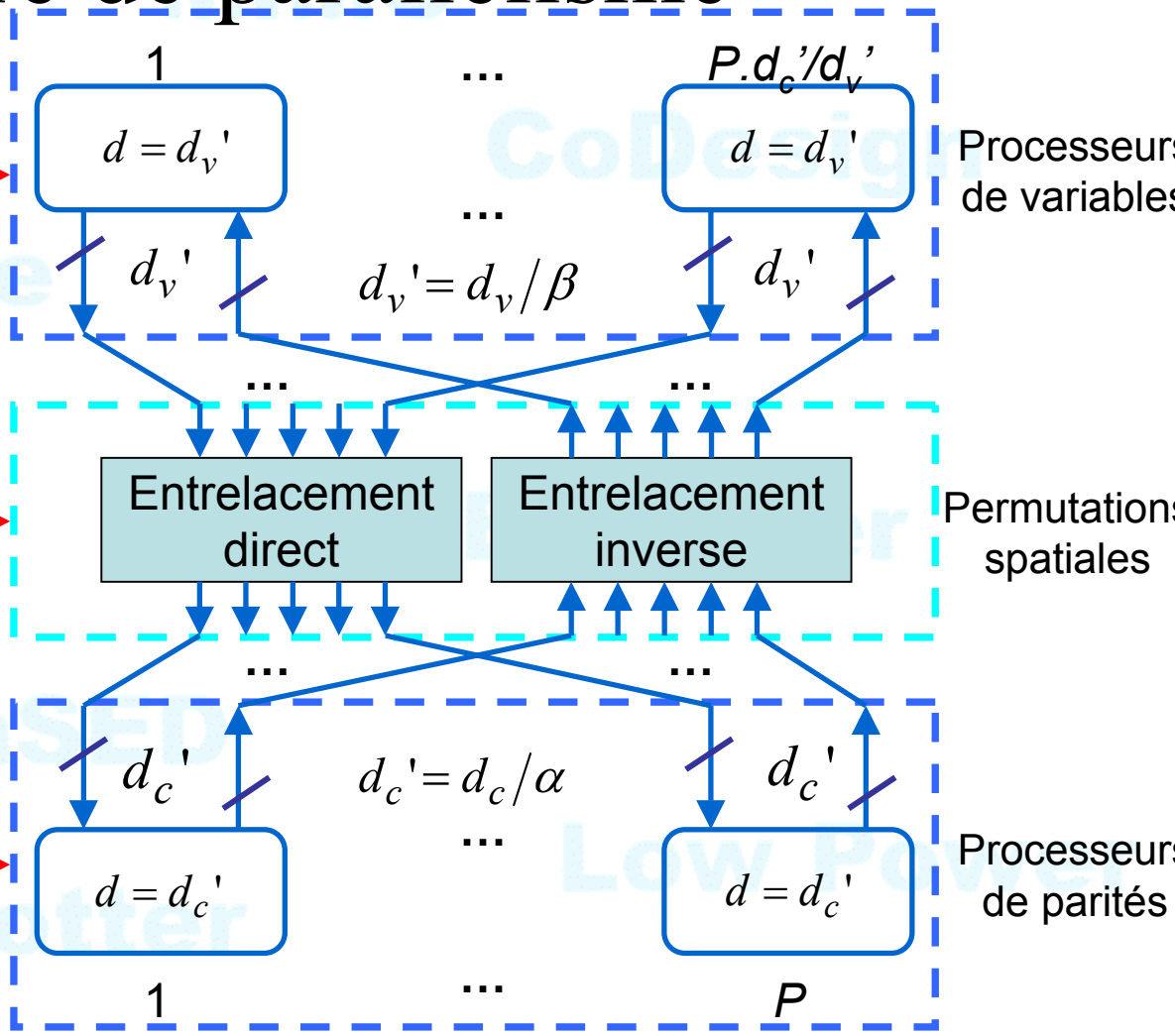
$t^{(i+1)}$ Change *d* fois durant l'itération

Réseau de permutation

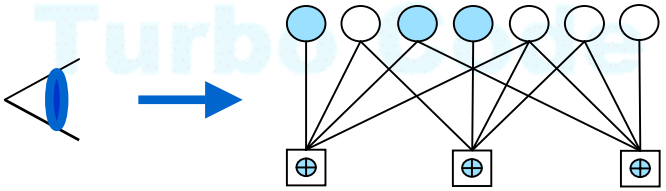
Degré de parallélisme

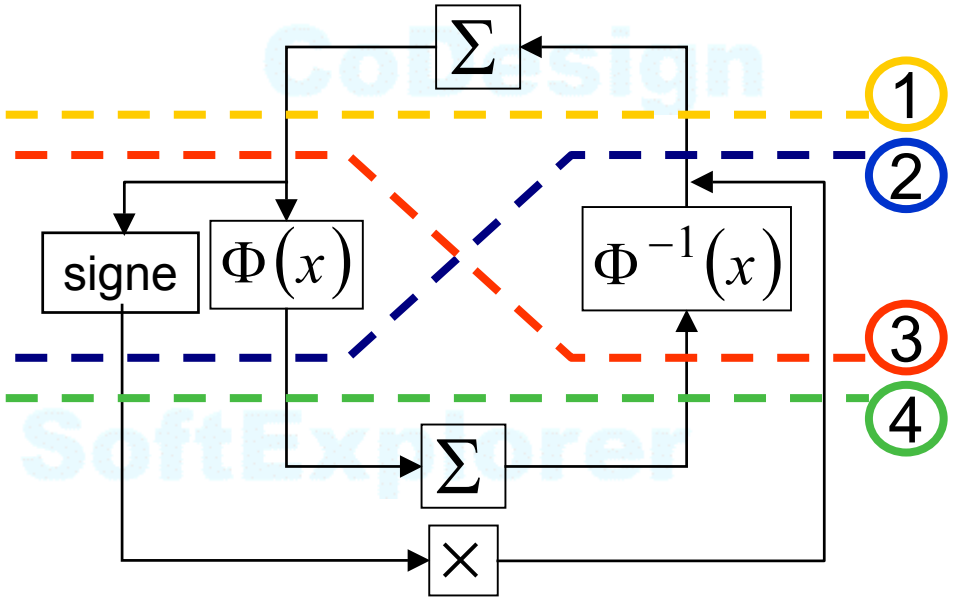
- P processeurs de parités
  (α cycles)
- Processeurs de variables
  (β cycles)
- Parallèle :
  P=M, α=1, β=1
- Série :
  P=1, α=d_c, β=d_v

Contrôle



Position du réseau d'interconnection





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

Modèle générique

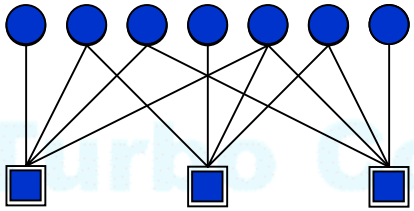
Paramètres :

- 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

• Combinaison des Contrôles

Entrelacement (vertical)



PARITES			VARIABLES: Compacte	VARIABLES: Distribué – Mise à jour différée	VARIABLES: Distribué – Mise à jour immédiate
	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	VARIABLES: Distribué – Mise à jour différée	VARIABLES: Distribué – Mise à jour immédiate
PARITES	Compacte		Blanksby (parallèle)	Boutillon, Chen, Zhang,	Mansour
PARITES	Distribué	Mise à jour différée	Nouveau	-	
PARITES	Distribué	Mise à jour immédiate	Nouveau (algo Fossorier)		

• Etat de l'art : taille mémoire

E : Nombre de branches

F( ) : compression

PARITES			VARIABLES: Compacte	VARIABLES: Distribué – Mise à jour différée	VARIABLES: Distribué – Mise à jour immédiate
Compacte			N + E	3N+ f(E)	N+ f(E)
Distribué	Mise à jour différée		N + 2M + E	-	
Distribué	Mise à jour immédiate		N + M + E		

PLAN

1. Code de parité
2. Principe des codes LDPC
3. Architecture de décodeur LDPC
4. En guise de conclusion

Conclusion

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

Proposition d'architectures nouvelles pour des séquencements verticaux ou horizontaux
=> convergence en 2 fois moins d'itérations
=> architecture requiérant 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 décodag

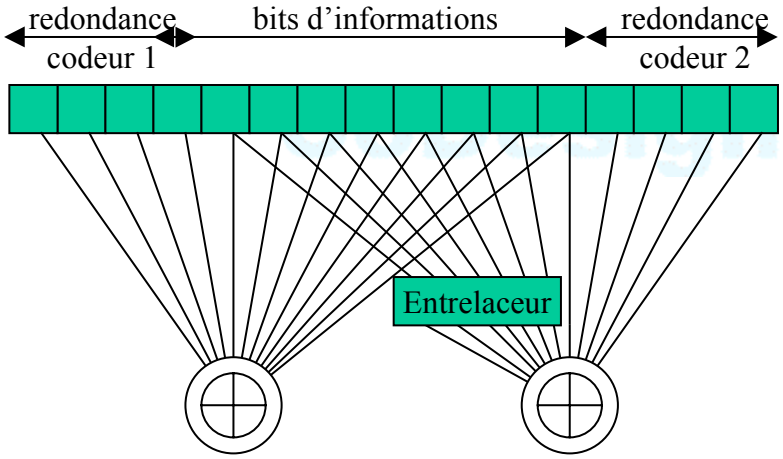
Travaux en cours:

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

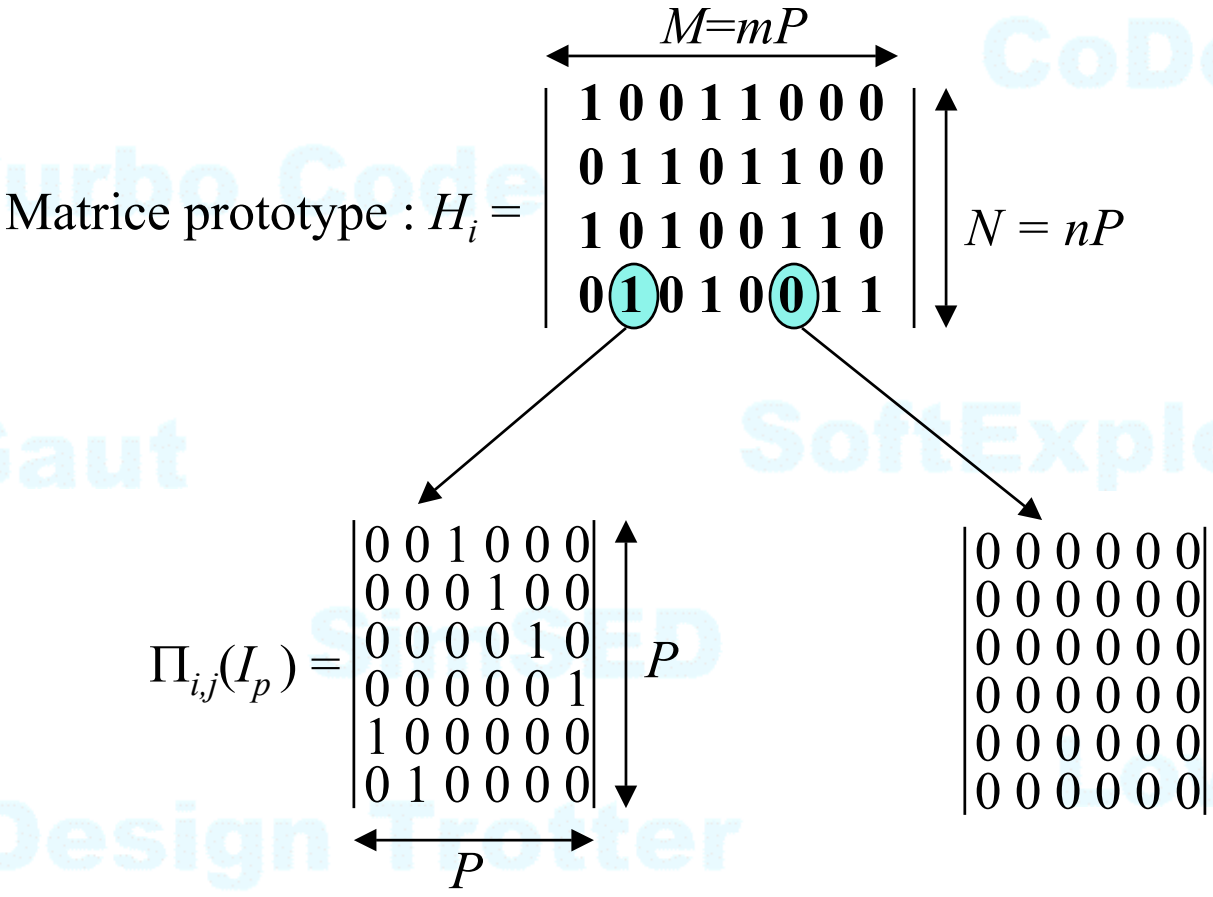
- les équations du transistor identiques à celle du MAP !

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

=> Solution très prometteuse...

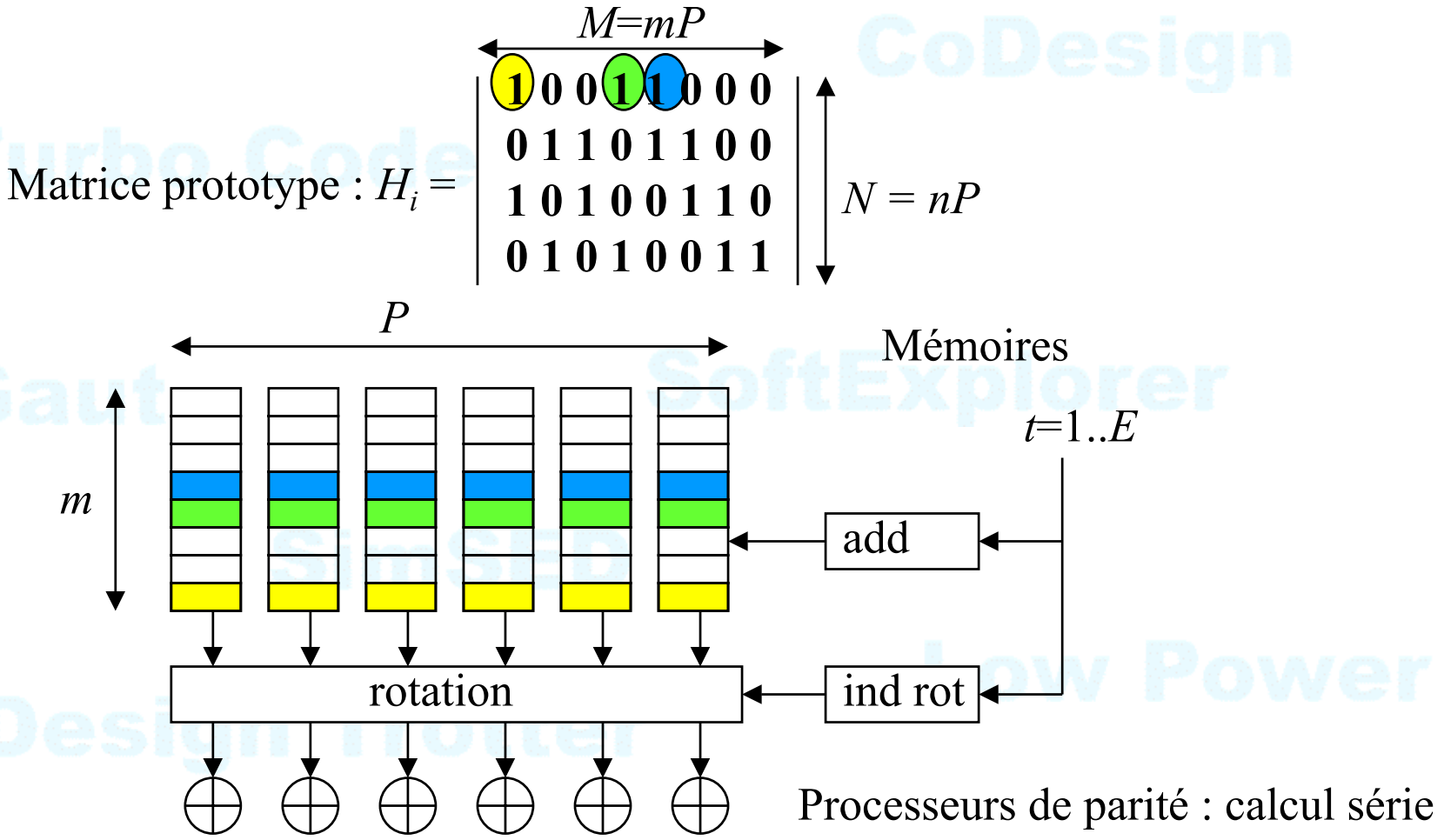
Question ?

Exemple construction matrice/architecture conjoint

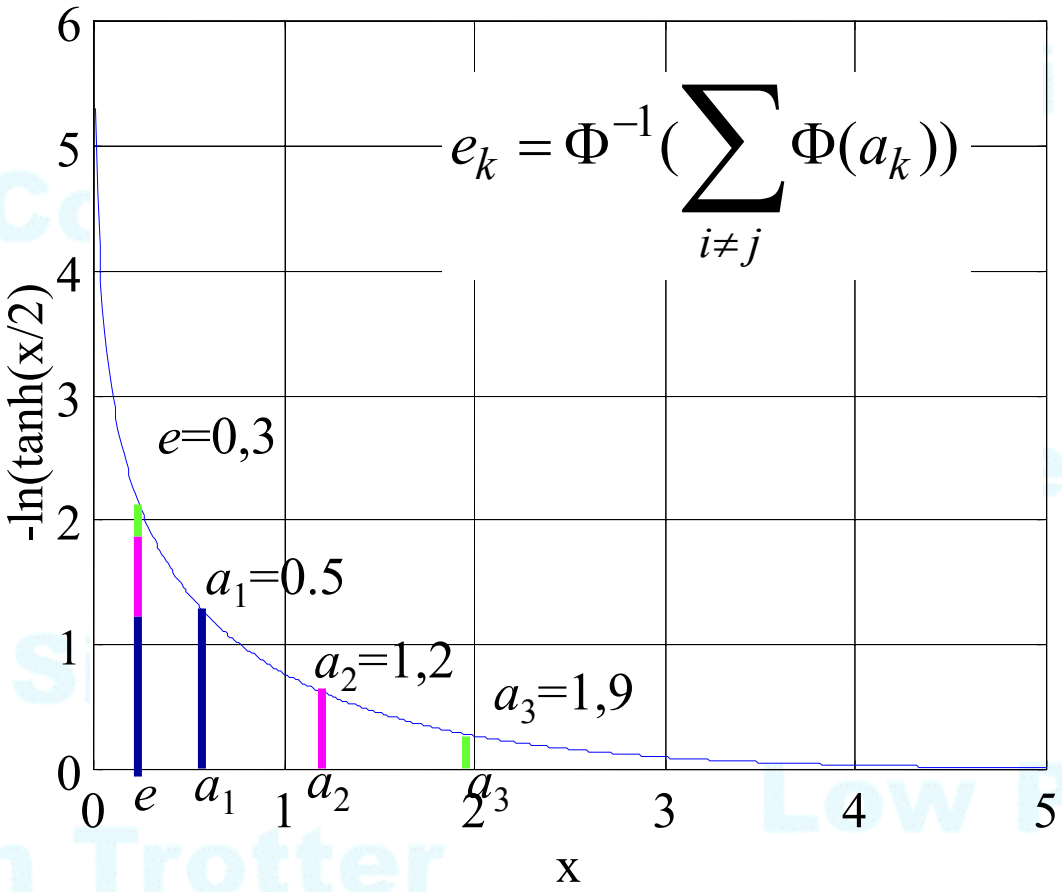


Très peu d'information pour mémoriser la matrice

Exemple construction matrice/architecture conjoint

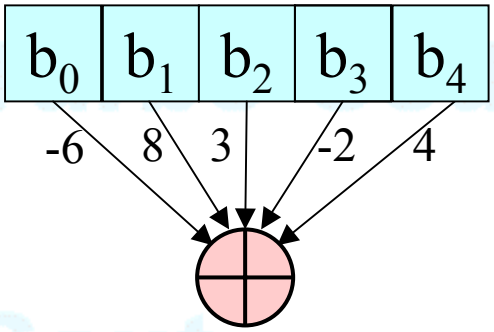


Fonction $\Phi(x)= \Phi^{-1}(x)$



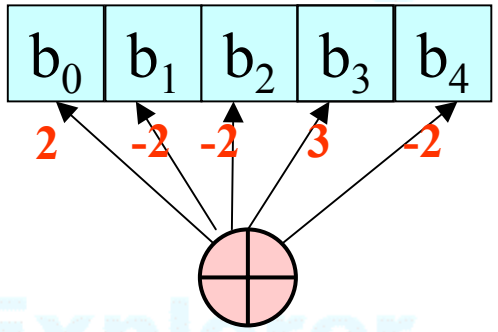
On obtient : $e = \min(a_1,a_2,a_3)$ - offset

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)