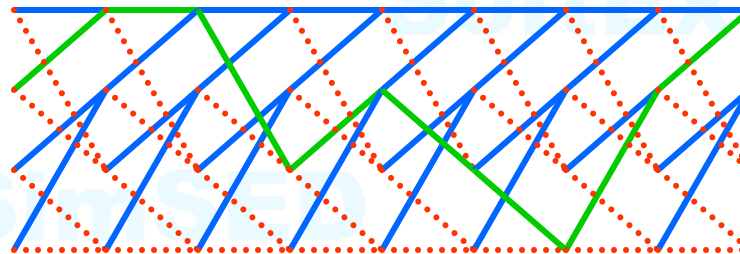


Architectures pour les codes correcteurs d'erreurs

Emmanuel Boutillon



UNIVERSITÉ DE BRETAGNE-SUD

Ecole d'été du GRETSI
Juillet 2007, Peyresq

Remerciement

Les résultats présentés ici sont issus de l'état de l'art et des travaux de thèse de Messieurs David Gnaedig (Turbo-Code), Frederic Guilloud (LDPC) ainsi que plus récemment, Haisheng Liu (Turbo-Code).

Caractéristiques des applications de télécommunications

Application de type "machine à coudre »

Données traitées uniformes (bit, échantillon, trame...)

Fréquence d'arrivée des données constantes

Algorithme à appliquer à chaque donnée uniforme

Algorithme de traitement connu à l'avance (norme).

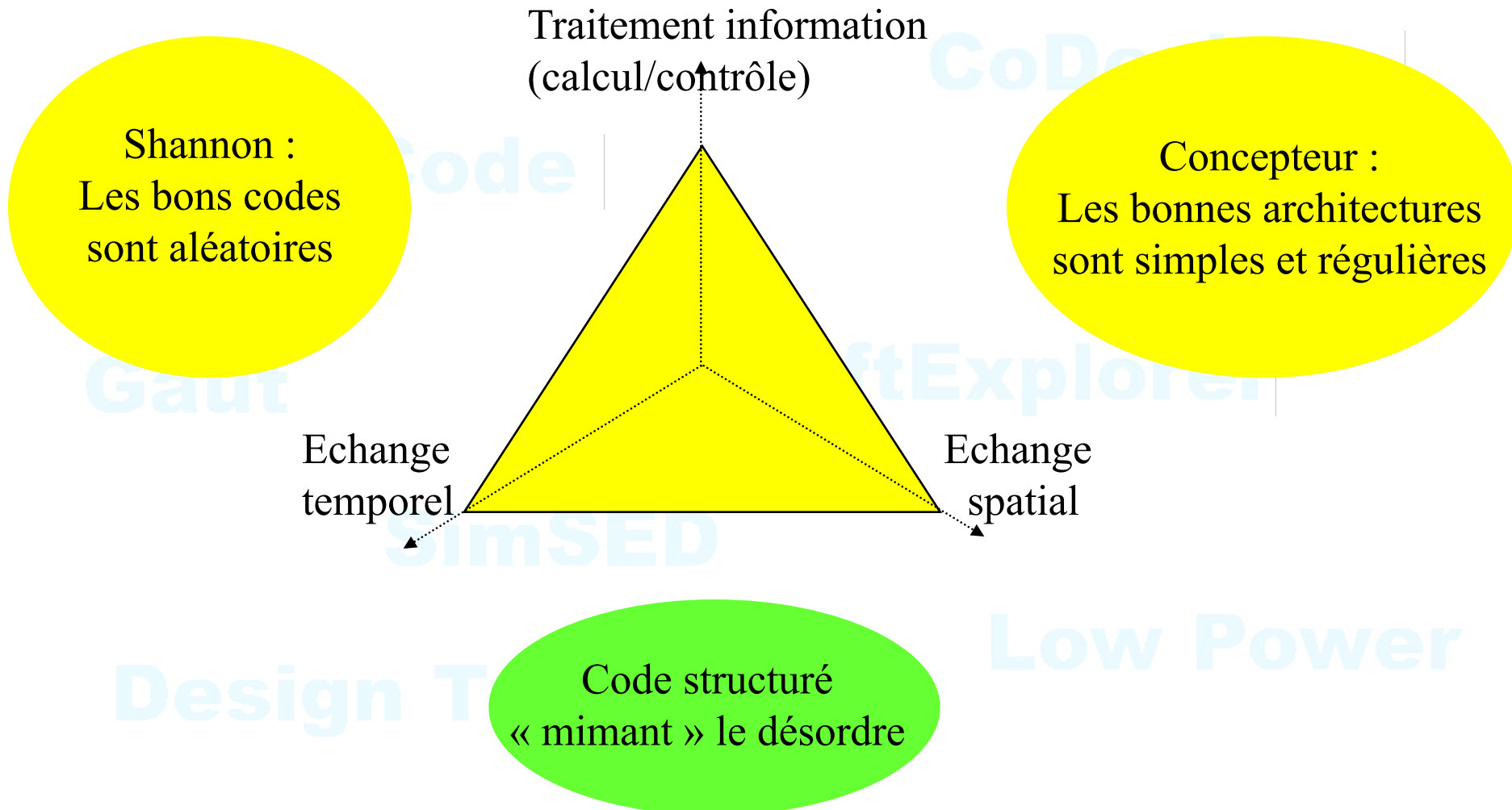
Actuellement => ASIC

- Réduction des coûts (produit de grande consommation)

- Performances (militaire, spatial)

Futur => ASIC et architectures programmables.

Espace de conception



PLAN

❶ Code convolutif et son décodage

❷ Turbo-code

❸ Code LDPC

❹ Décodeurs haut débits

❹ Conclusion

PLAN

❶ Code convolutif et son décodage

❶ Codeur convolutif

❶ Viterbi

❶ MAP

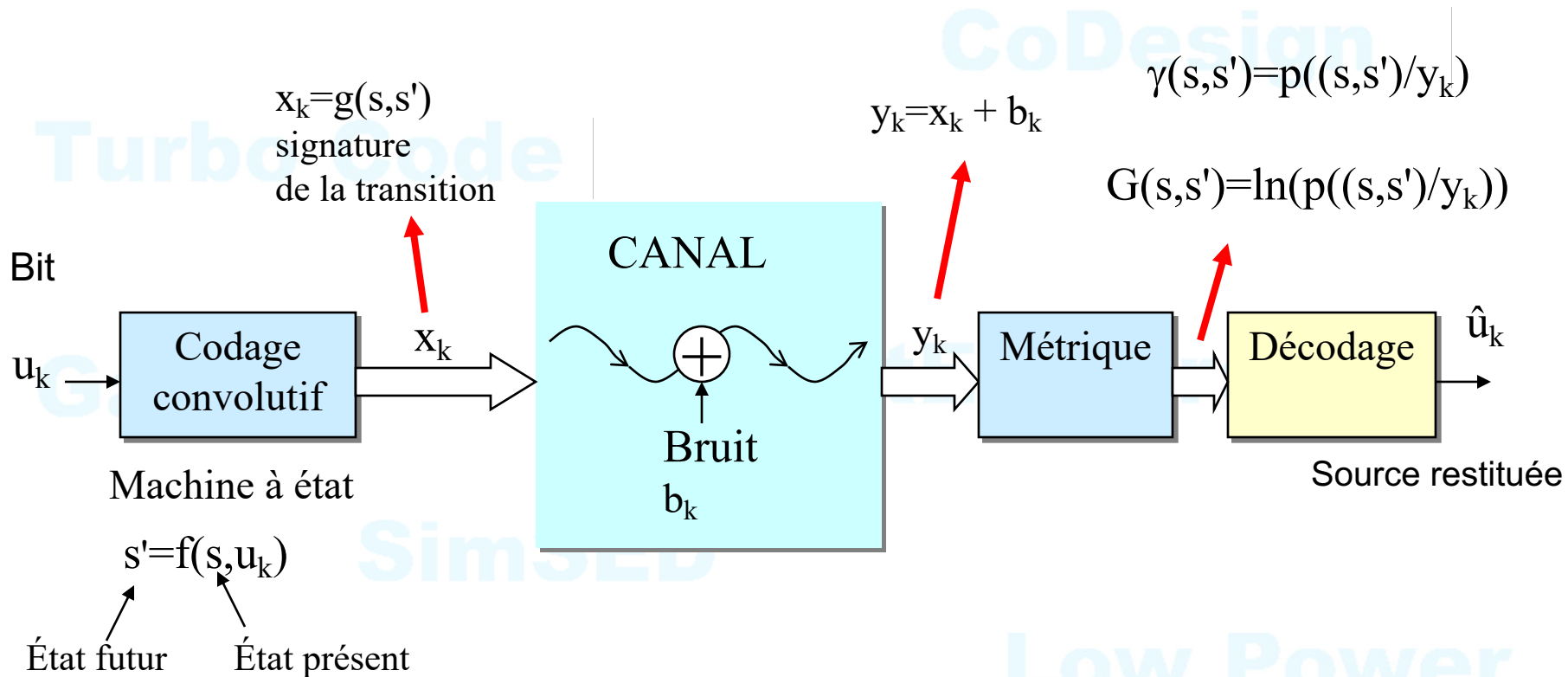
❷ Turbo-code

❸ Code LDPC

❹ Décodeurs haut débits

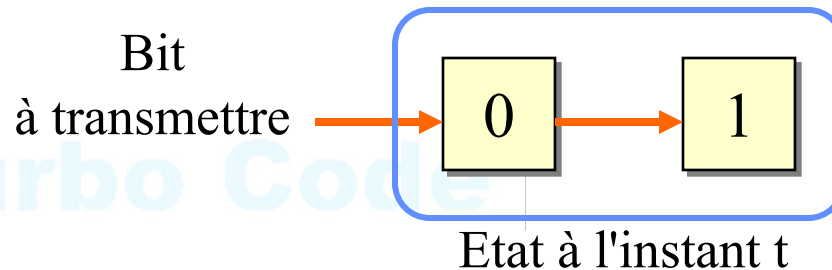
❹ Conclusion

Principe d'une chaîne de transmission avec codage convolutif



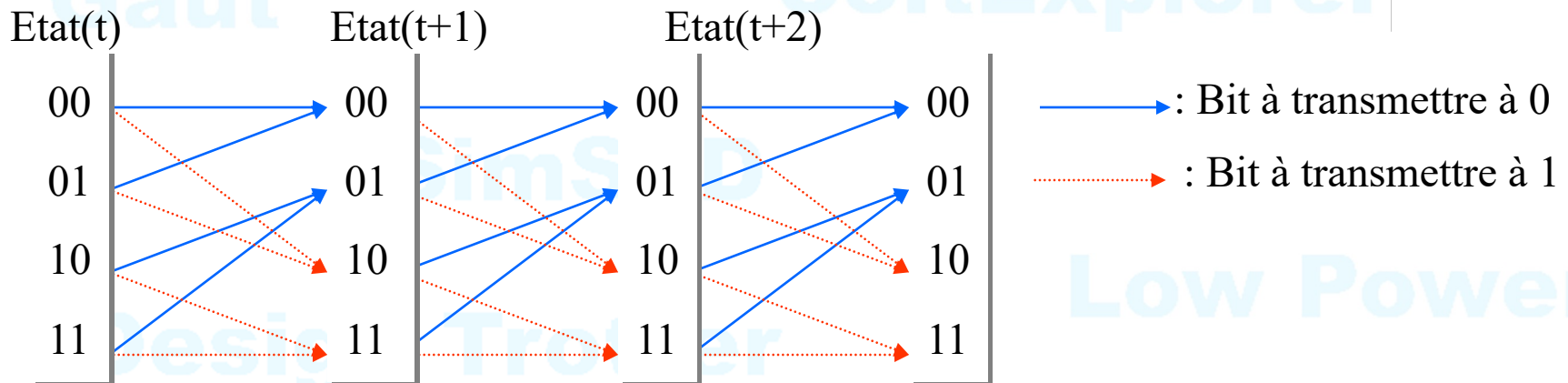
Message : U , message codé : $X = F(s_0, U)$, message reçu = Y

Exemple de code convolutif

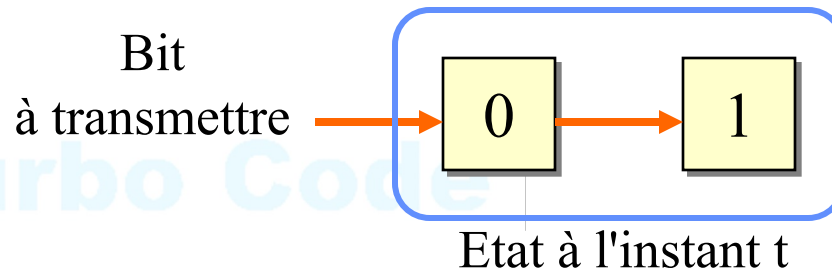


Longueur de contrainte $K=3$
(1 bit à transmettre + 2 bits d'état)

L'évolution de l'état du codeur est représentée par un treillis :

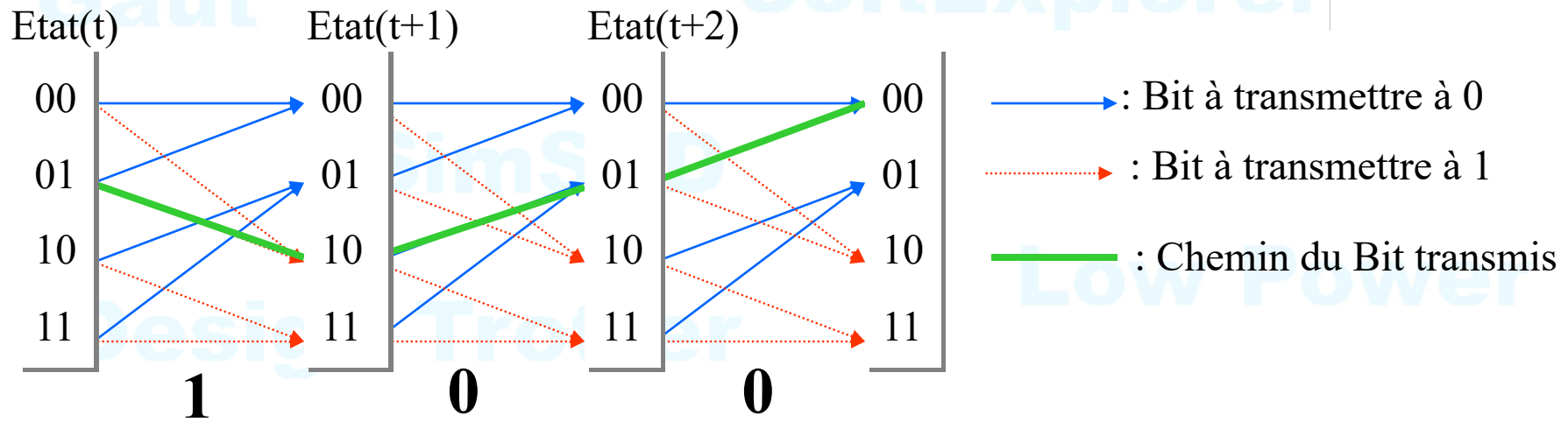


Code convolutif(2)

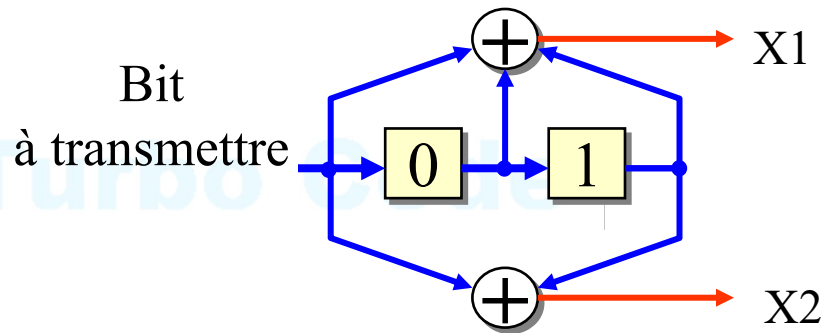


Longueur de contrainte $K=3$
(1 bit à transmettre + 2 bits d'état)

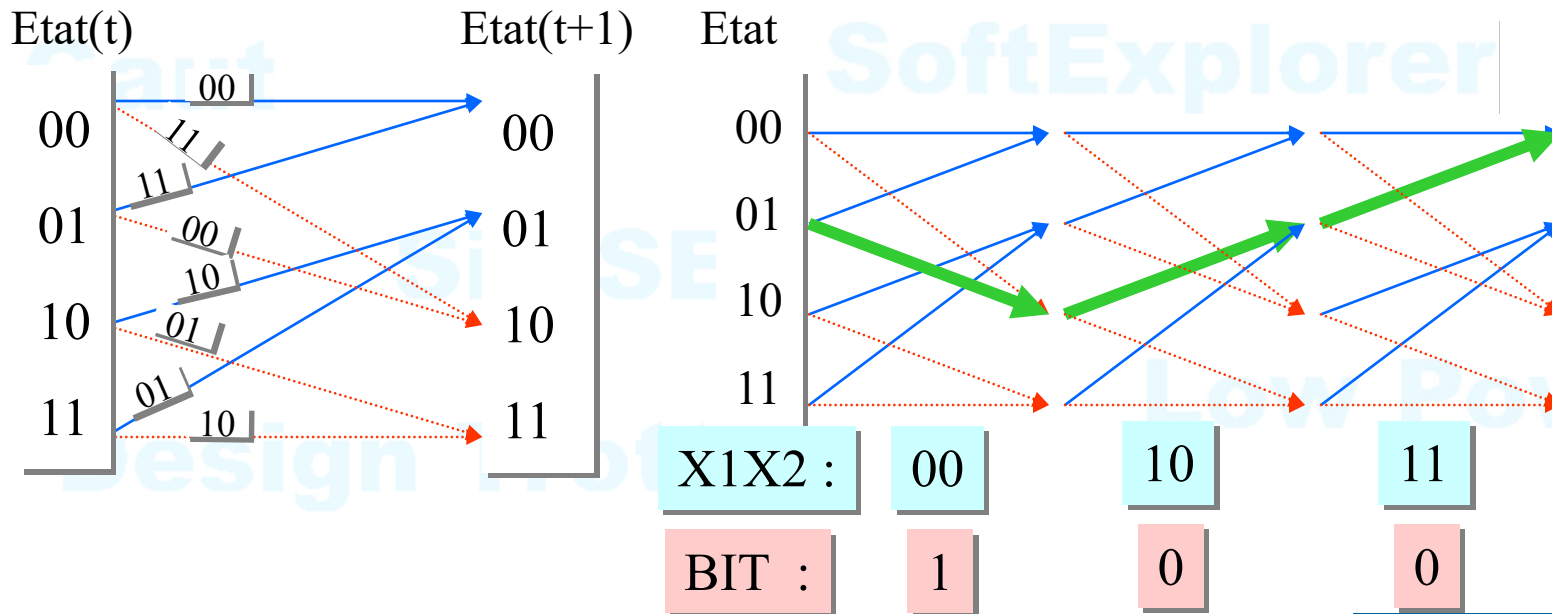
Chemin du bit transmis sur le treillis :



Code convolutif(3)



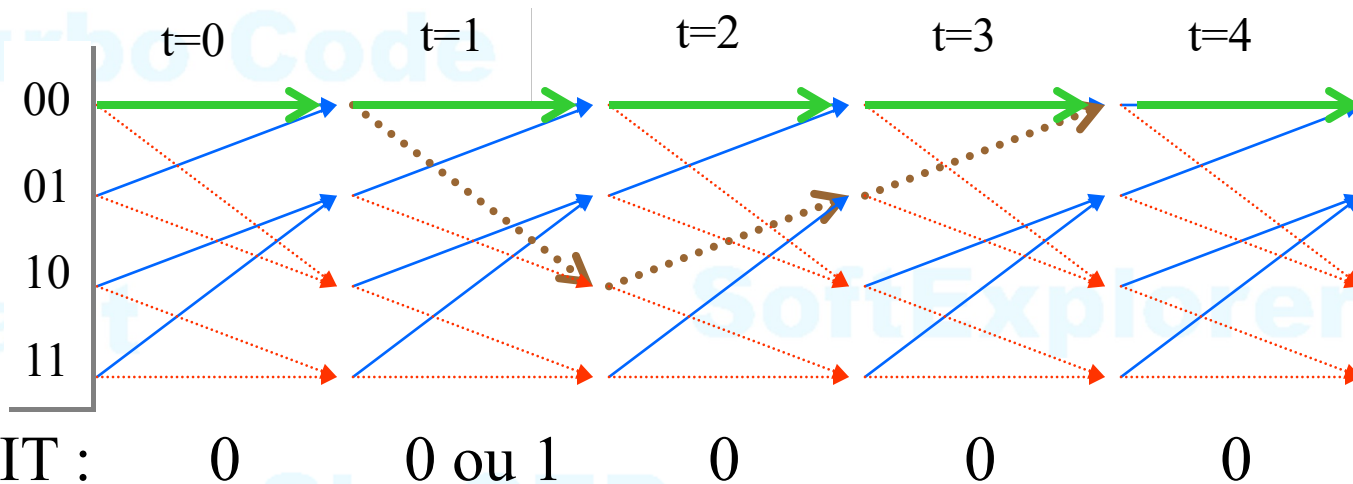
Les sorties X1 et X2 représentent la signature des transitions du bit à transmettre
(redondance $\Rightarrow$ rendement=1/2)



10/180

Code convolutif(4)

 : Chemin lorsque un bit est toujours à 0
 : Chemin lorsque un bit est toujours à 0 sauf 1 fois



X1X2 :
 bit toujours à 0

00

00

00

00

00

Distance libre=5

X1X2 :
 bit à 1 à t=1

00

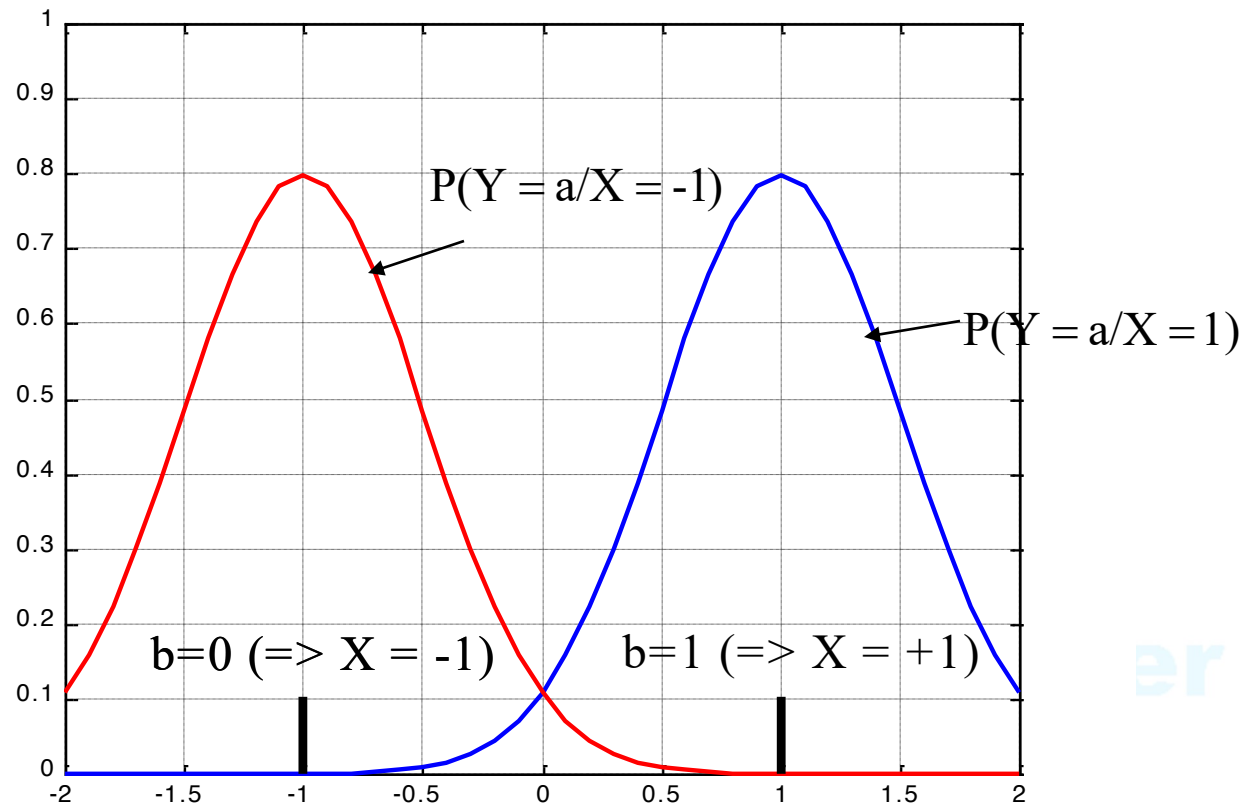
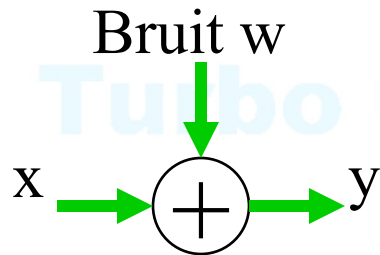
11

10

11

00

Canal de transmission gaussien



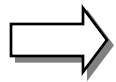
Canal de transmission gaussien

$$P(Y / X = 1) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(Y-1)^2}{2\sigma^2}}$$

σ^2 =variance du bruit

Dans le domaine des logarithmes, on obtient :

$$\text{Log}(P(Y / X = 1)) = K_1 + K_2(Y - 1)^2 = C(Y) - 2K_2Y$$



$$\text{Log}(P(Y / X = -1)) = K_1 + K_2(Y + 1)^2 = C(Y) + 2K_2Y$$

$$K_1 = -\text{Log}(\sigma\sqrt{2\pi}) \quad K_2 = -\frac{1}{2\sigma^2}$$

$$C(Y) = K_1 + K_2(Y^2 + 1)$$

Métrie avec la décision souple

Si la signature émise est X_1X_2 et celle reçue est Y_1Y_2 :

$$P(Y_1Y_2/X_1X_2=00) = P(Y_1/X_1=0) * P(Y_2/X_2=0)$$

$$\text{Log}[P(Y_1Y_2/X_1X_2=00)] = C(Y_1) + C(Y_2) + 2K_2(-Y_1 - Y_2)$$

$$\text{Log}[P(Y_1Y_2/X_1X_2=01)] = C(Y_1) + C(Y_2) + 2K_2(-Y_1 + Y_2)$$

$$\text{Log}[P(Y_1Y_2/X_1X_2=10)] = C(Y_1) + C(Y_2) + 2K_2(+Y_1 - Y_2)$$

$$\text{Log}[P(Y_1Y_2/X_1X_2=11)] = C(Y_1) + C(Y_2) + 2K_2(+Y_1 + Y_2)$$

Commun à
toutes les hypothèses

Métriques de branches

$$\text{Log}[P(Y_1Y_2/X_1X_2)] = F(s(X_1)Y_1 + s(X_2)Y_2)$$

avec

F linéaire

$s(X) = 1$ si $X=1$

$s(X) = -1$ si $X=0$

Calculs simples

Probabilité d'un chemin

Soit C_i , un chemin parmi les 2^N possibles.

Soit $(X_1^0, X_1^0, \dots, X_1^k, X_2^k, \dots, X_1^{N-1}, X_2^{N-1})$ les signatures du chemin C_i

$$H_i = \log(P(\text{Obser.} / C_i))$$

$$H_i = \log(P(Y_1^0, Y_1^0, \dots, Y_1^k, Y_2^k, \dots, Y_2^{N-1} / X_1^0, X_1^0, \dots, X_1^k, X_2^k, \dots, X_1^{N-1}, X_2^{N-1}))$$

$$H_i = \sum_{k=0}^{N-1} \log(P(Y_1^k, Y_2^k / X_1^k, X_2^k)) = F\left(\sum_{k=0}^{N-1} Y_1^k s(X_1^k) + Y_2^k s(X_1^k)\right)$$

**Le chemin C optimal qui maximise $P(\text{Observation}/C)$
est obtenu par l'algorithme de Viterbi.**

Les problèmes de décodage

Viterbi : Trouver le mot de code le plus probable (décision dure)

$$\tilde{C} = \arg \max_C (\log(P(Y / C)))$$

$$\tilde{C} = \arg \min_C (-\log(P(Y / C)))$$

Parmi tous les chemins du treillis, trouver celui expliquant le mieux l'observation

MAP : Calculer le LRR de chaque bits :

$$LRR(bi) = \log \left(\frac{\sum_{C_i^1, b_i=1} P(Y / C_i^1)}{\sum_{C_i^0, b_i=0} P(Y / C_i^0)} \right)$$

Algorithmes de décodage des codes convolutifs

- o **Viterbi** (décision souple en entrée, dure en sortie)
 - > Chemin survivant optimal
- o **SOVA** (décision souple en entrée, souple en sortie)
 - > fiabilité approximative, complexité réduite
 - > permet de décoder les "turbo-codes"
- o **Forward-Backward** (décision souple en entrée, souple en sortie)
 - > fiabilité exacte des bits décodés
 - > +0,5 dB de mieux que le SOVA

PLAN

❶ Code convolutif et son décodage

❶ Codeur convolutif

❶ Viterbi

❶ MAP

❷ Turbo-code

❸ Code LDPC

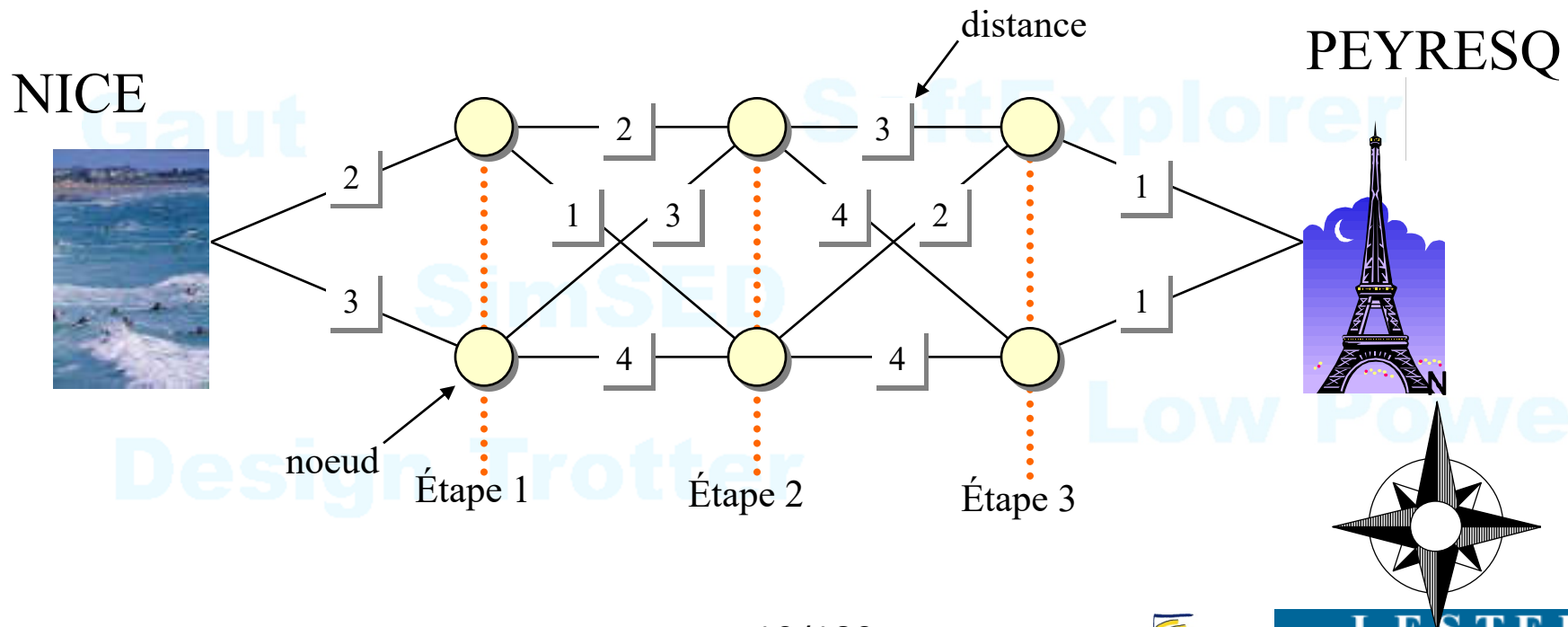
❹ Décodeurs haut débits

❹ Conclusion

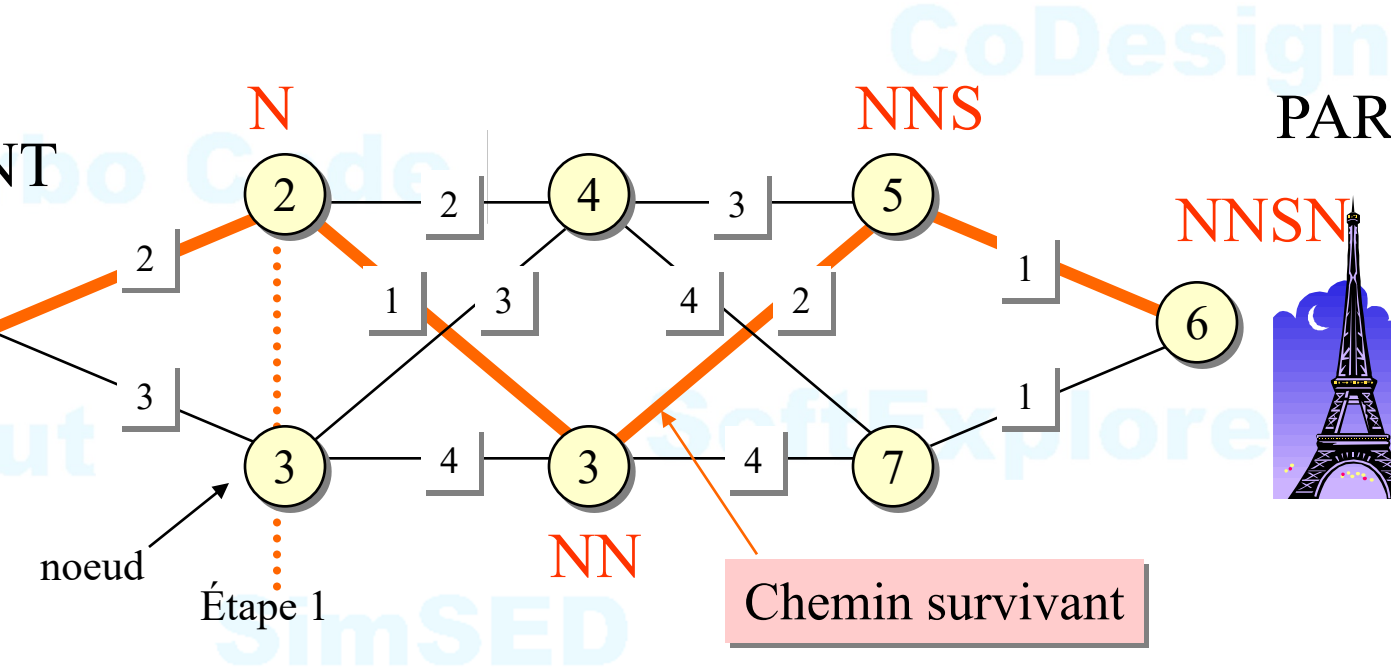
Décodeur de Viterbi

Le décodage consiste à retrouver le chemin parcouru par le bit dans le treillis en utilisant une recherche du maximum de vraisemblance

Exemple : Quel est le plus court chemin ?

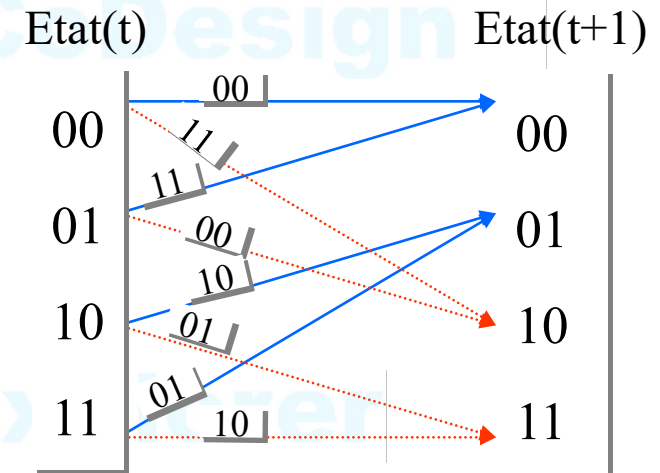
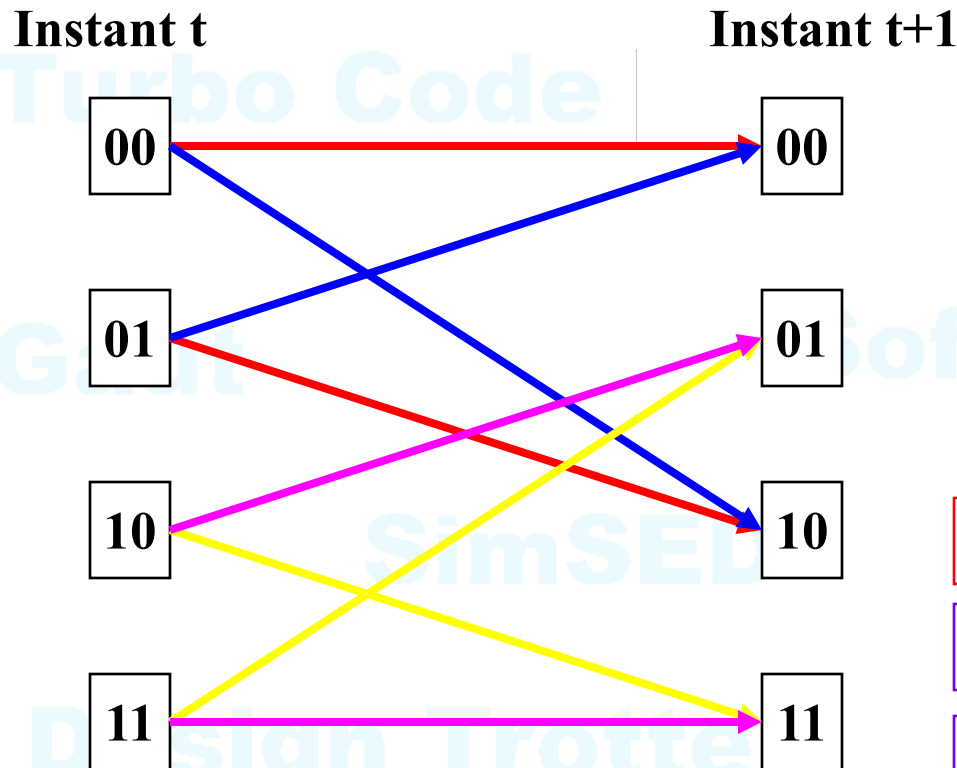


Probleme aa prab eoa



Pour chaque nœud, la distance parcourue la plus courte est calculée et le chemin le plus court est mémorisé. Il faut arriver à destination pour en déduire le chemin le plus court = Chemin survivant

Calcul métrique de branche



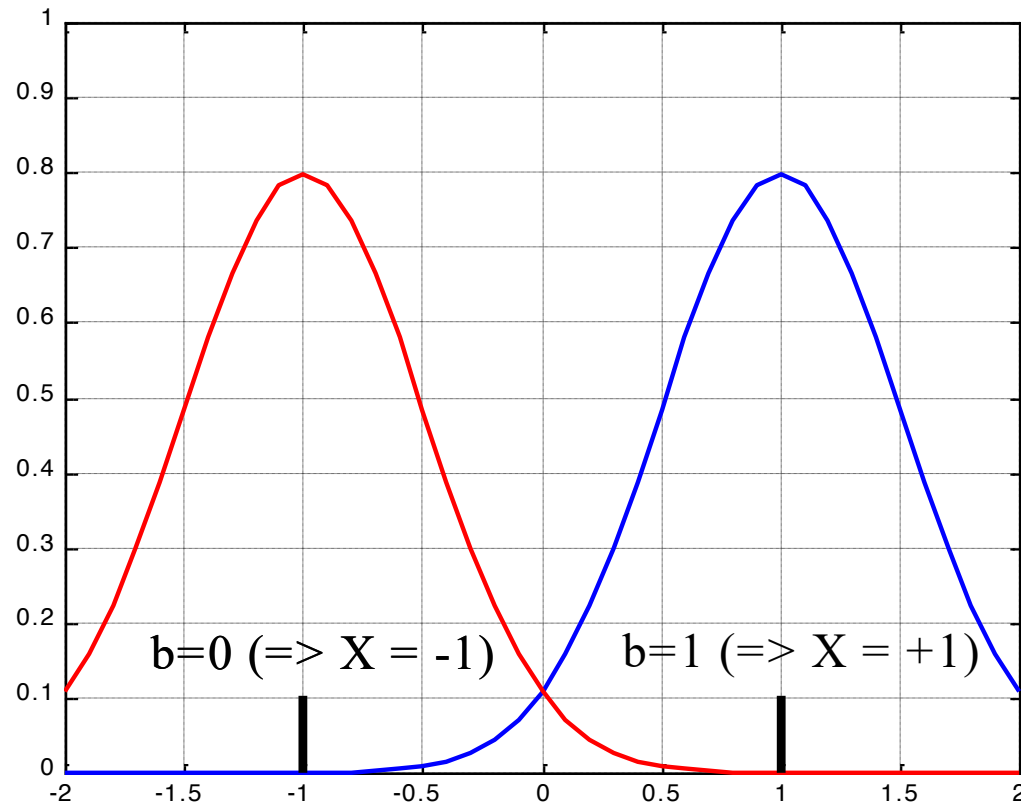
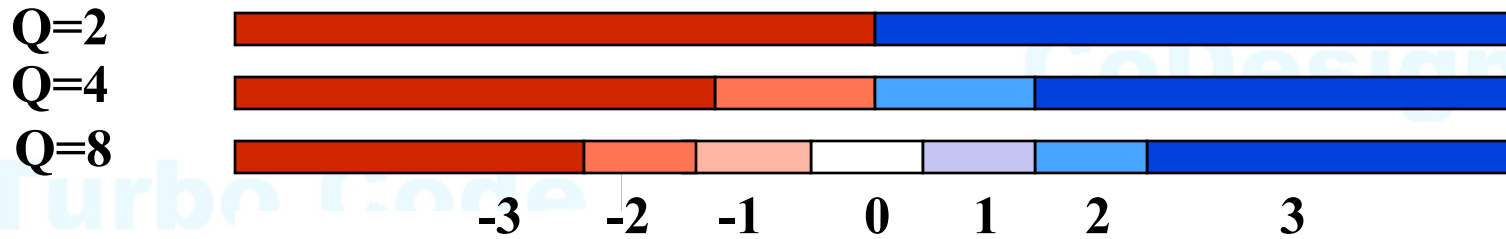
$$MB_{00} = -Y_1 - Y_2$$

$$MB_{01} = -Y_1 + Y_2$$

$$MB_{10} = +Y_1 - Y_2$$

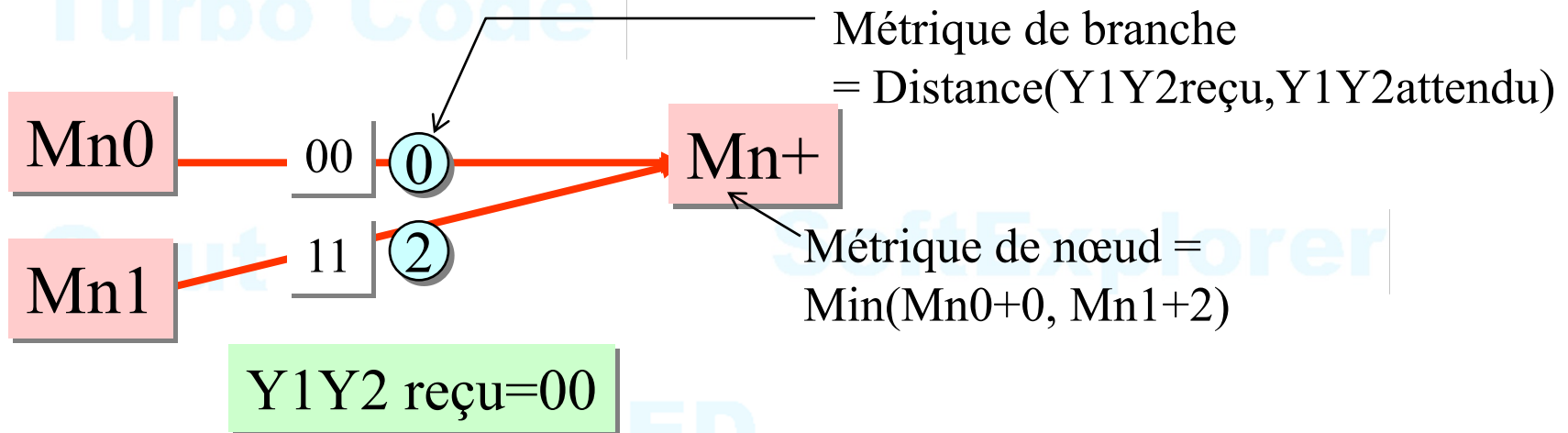
$$MB_{11} = +Y_1 + Y_2$$

Décision souple



Décodeur de Viterbi

La recherche de la séquence émise nécessite le calcul des métriques de branche et métriques de nœud

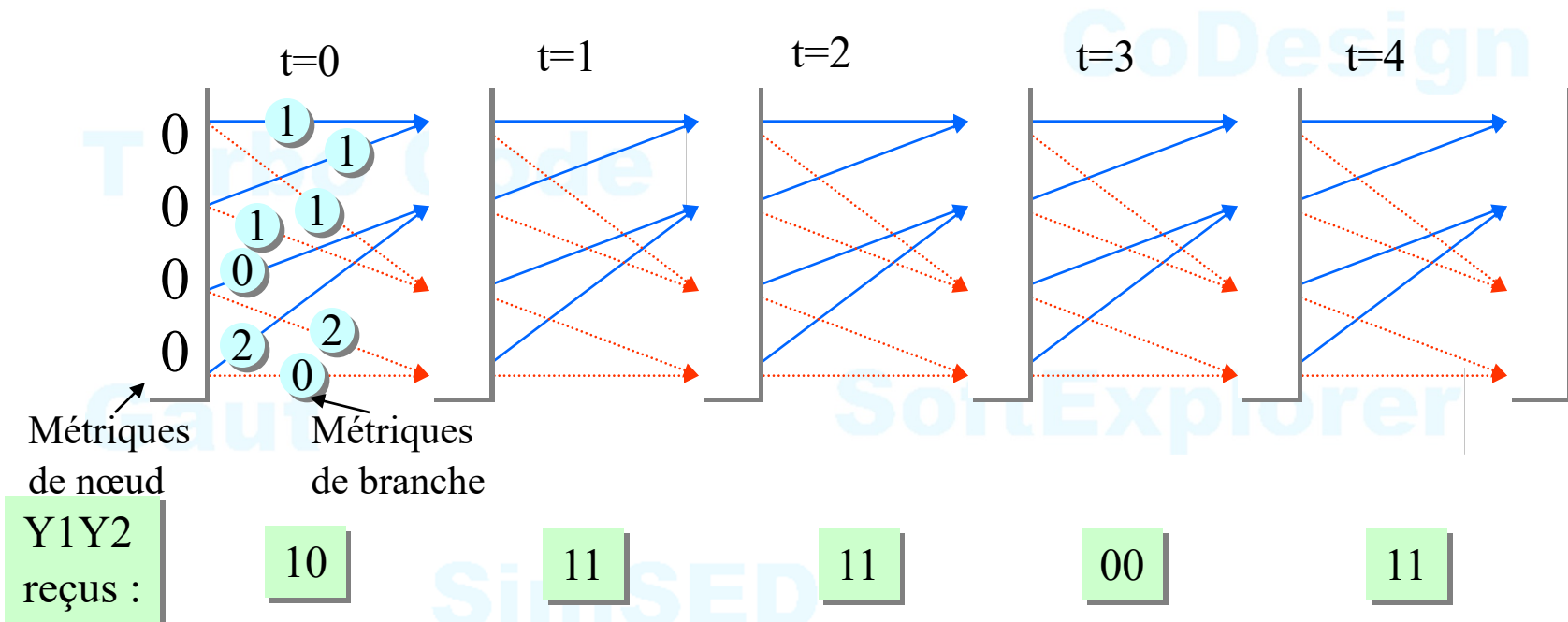


A chaque nœud est associée la décision correspondant à la branche la plus vraisemblable : décision = $[(Mn0+0) < (Mn1+2) ? 0, 1]$

La valeur du BIT est donnée par la position de nœuds . Au NORD le bit est à 0.

Au SUD , le bit est à 1.

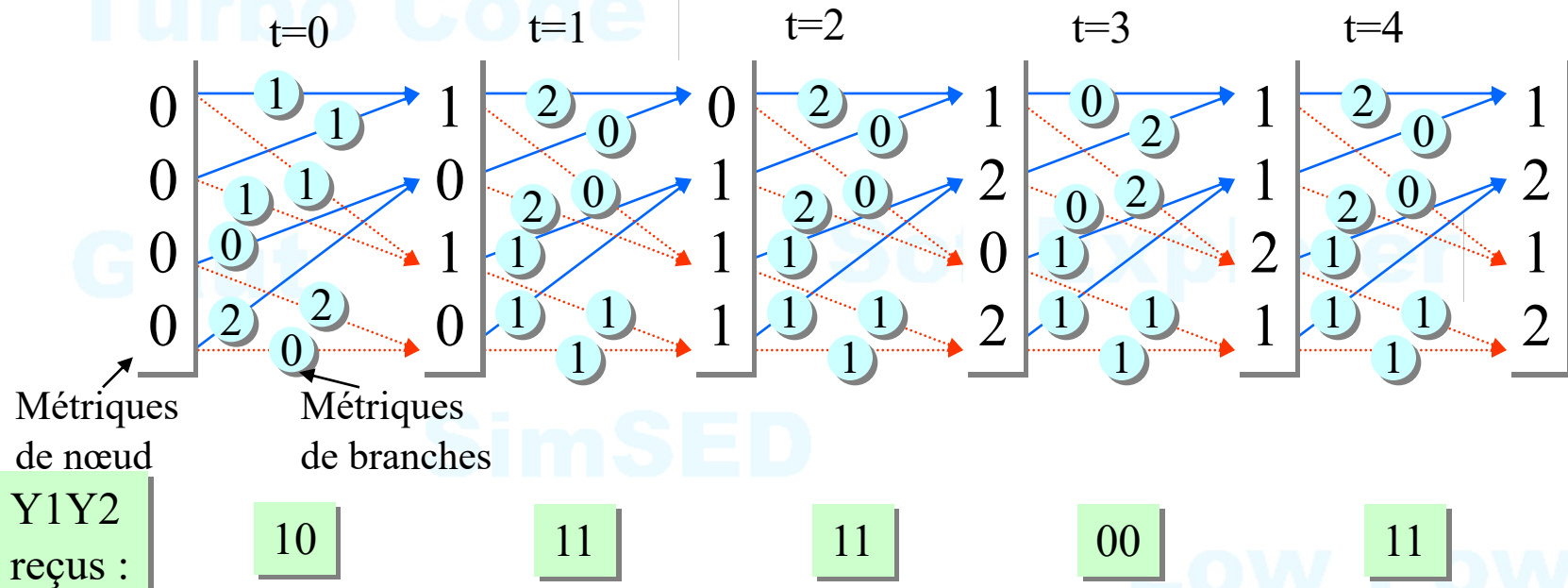
Décodeur de Viterbi(2)



Exercice : Calculer les métriques de branches et métriques de nœuds
note : les nœuds sont équiprobables à $t=0$

Décodeur de Viterbi(3)

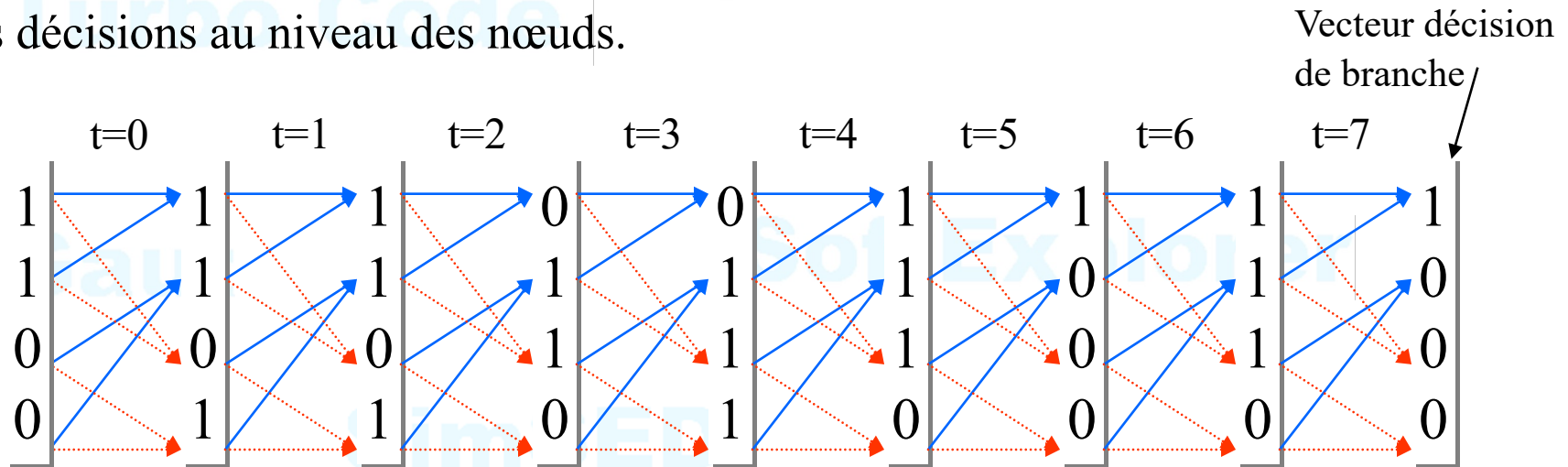
Solution :



Décodeur de Viterbi (4)

Exercice : Rechercher la séquence émise.

Note : Il suffit de remonter le chemin à partir de la destination, en considérant les décisions au niveau des nœuds.



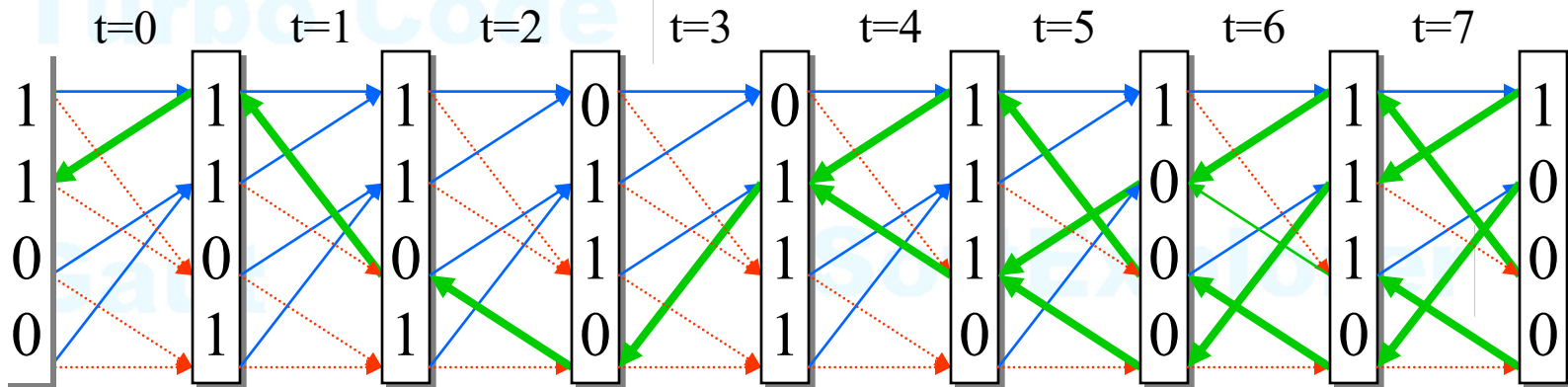
Les chemins convergent en remontant sur une profondeur L suffisamment grande. L = longueur de convergence

$$L \cong 6K$$

K = longueur de contrainte (Nbre d'états = 2^{K-1})

Décodeur de Viterbi(5) Canal de transmission gaussien

Solution :



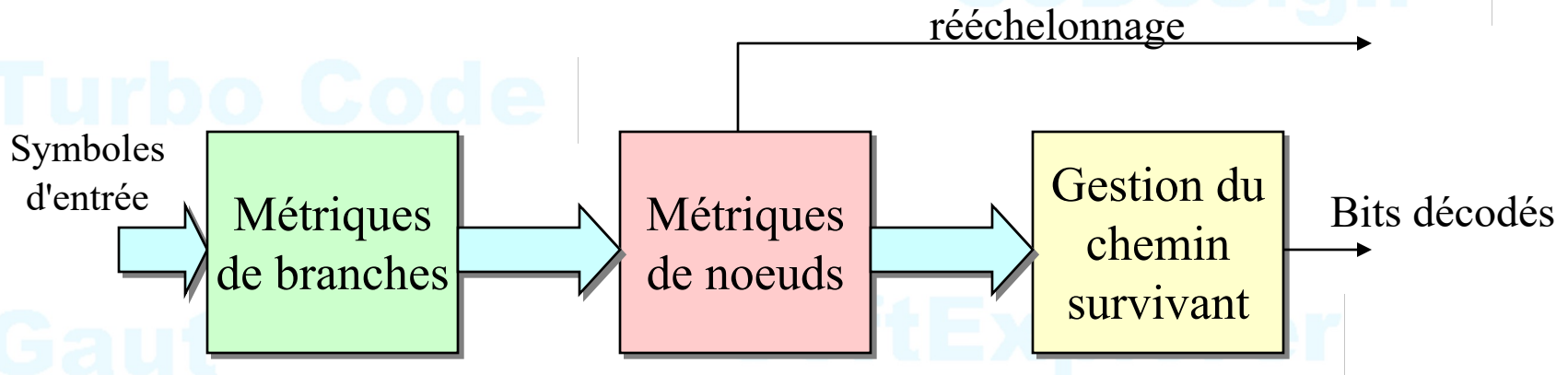
Les chemins convergent en remontant sur une profondeur L suffisamment grande. L = longueur de convergence

$$L \cong 6 K$$

K = longueur de contrainte (Nbre d'états = 2^{K-1})

Architecture du décodeur

3 blocs principaux :



Points critiques :

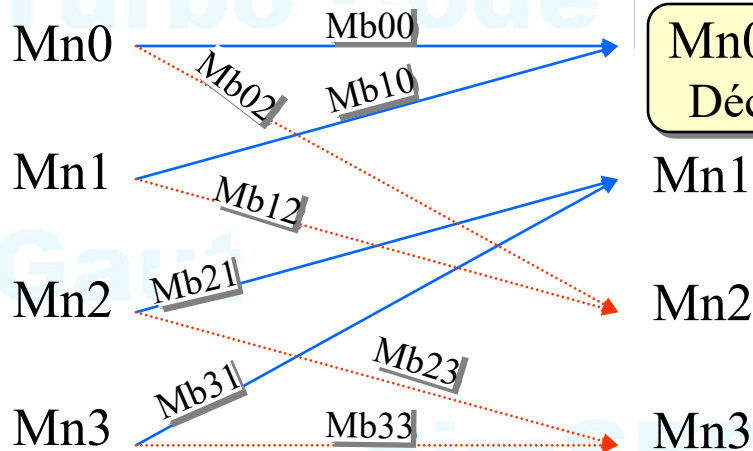
- ⇒ Puissance de calcul pour les métriques de nœud
- ⇒ Accroissement des métriques de nœud infini
- ⇒ Gestion de la mémoire survivante

Métriques de noeuds

Exemple : $K=3$

Métrieque(t)

Métrieque(t+1)



$Mn0 = \text{Min} (Mn0+Mb00, Mn1+Mb10)$
 Décision=[$Mn0+Mb00 < Mn1-Mb10$? 0 , 1]

2 Additionneurs
 1 Comparateur
 1 Sélecteur

Opérateur
 ACS

Si $K=7$ (64 états)
 débit=30Mbits/s

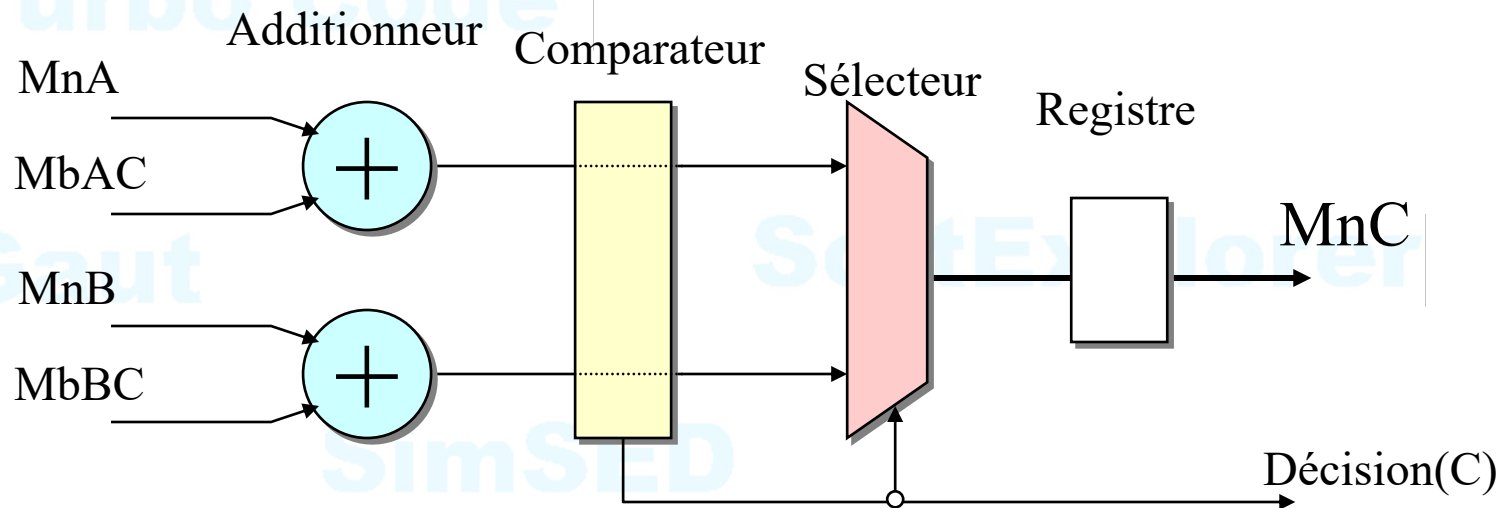


6 G additions/s !

Note : 1 comparateur
 =1 soustracteur

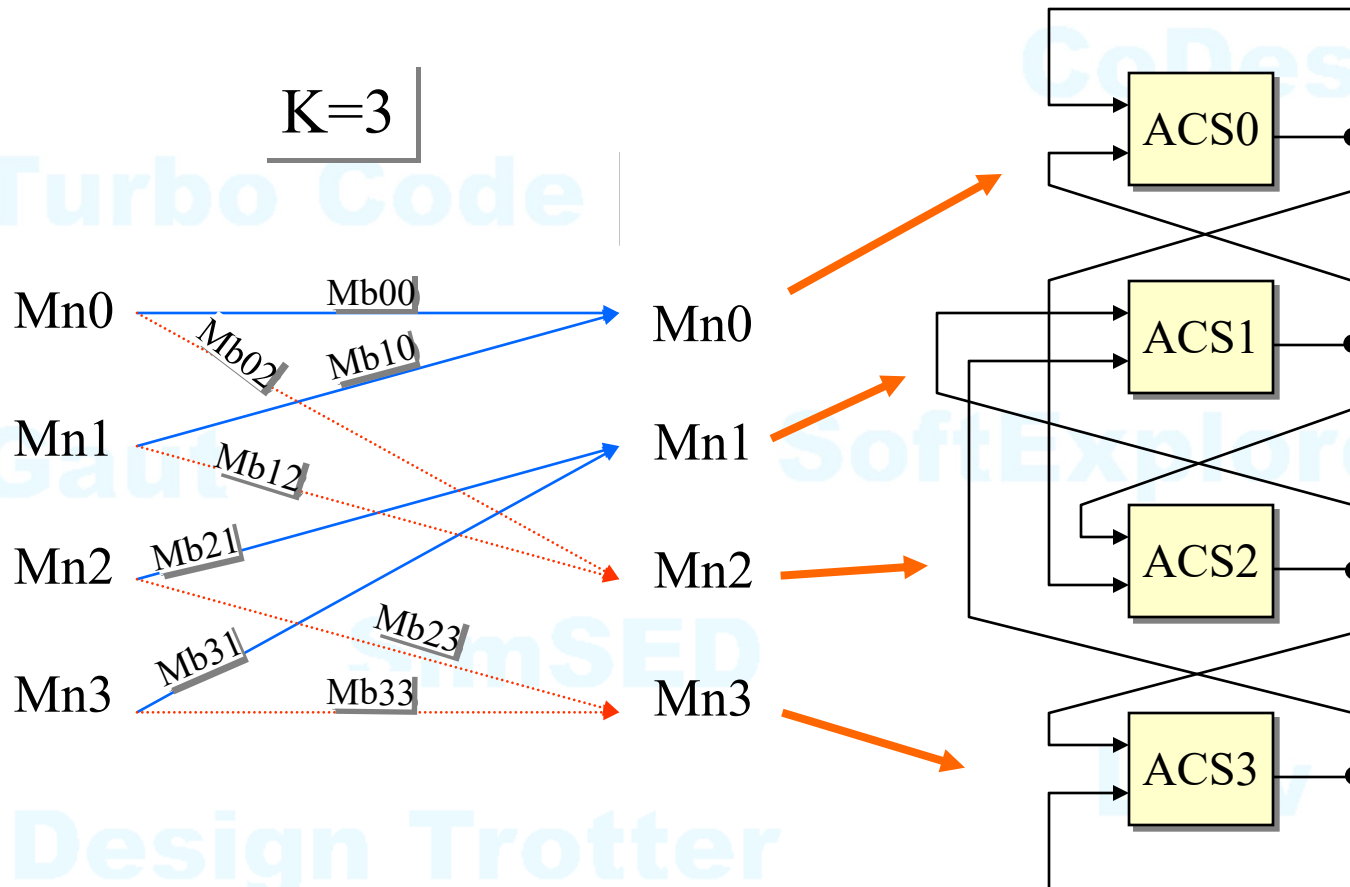
Opérateur ACS

Architecture d'un ACS :



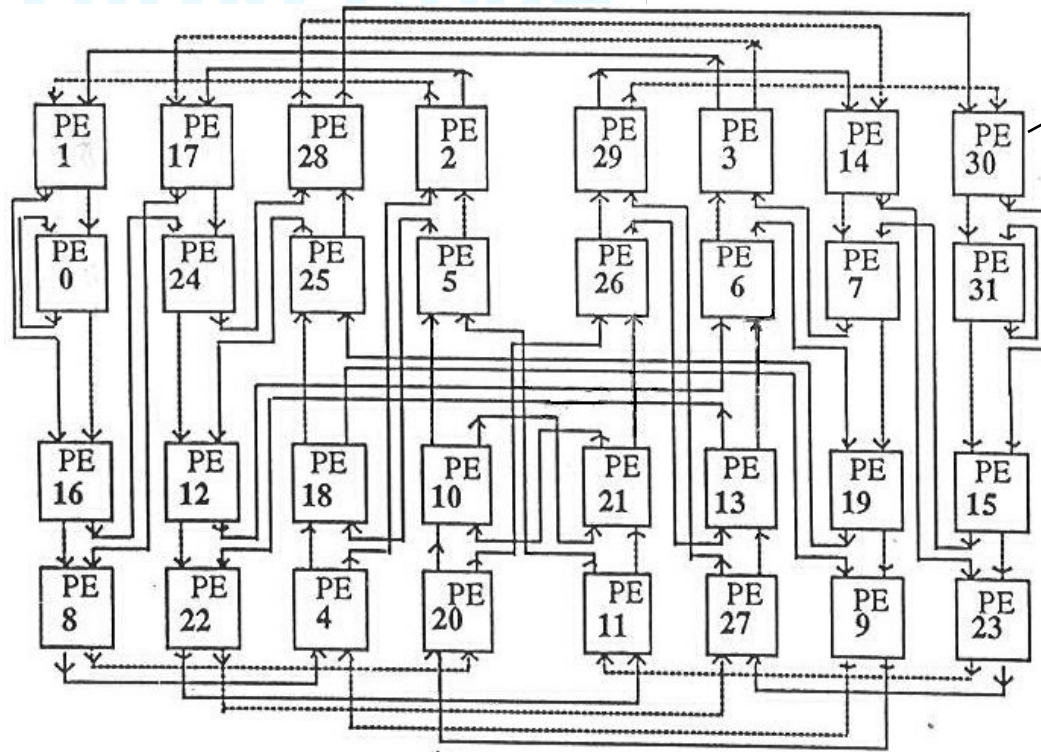
En technologie $0,5\mu \Rightarrow f_{max} = 100\text{MHz}$

Interconnexion des ACS

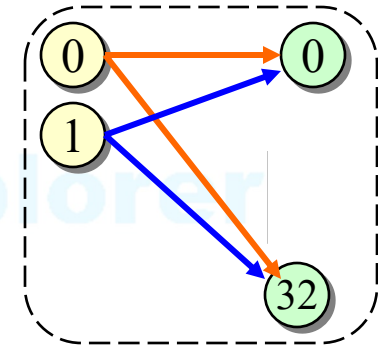


Interconnexion des ACS(2)

$K=7$ $\Rightarrow$ 64 états



1PE=2ACS

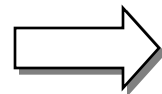
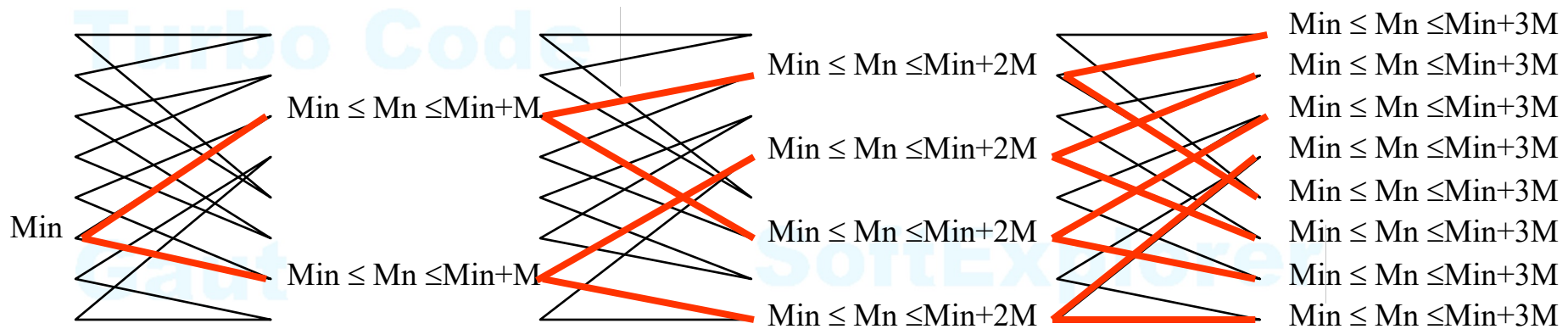


Surface d'interconnexion importante

Dynamique des métriques de noeuds

$K=4$

M =métrique de branche maximale

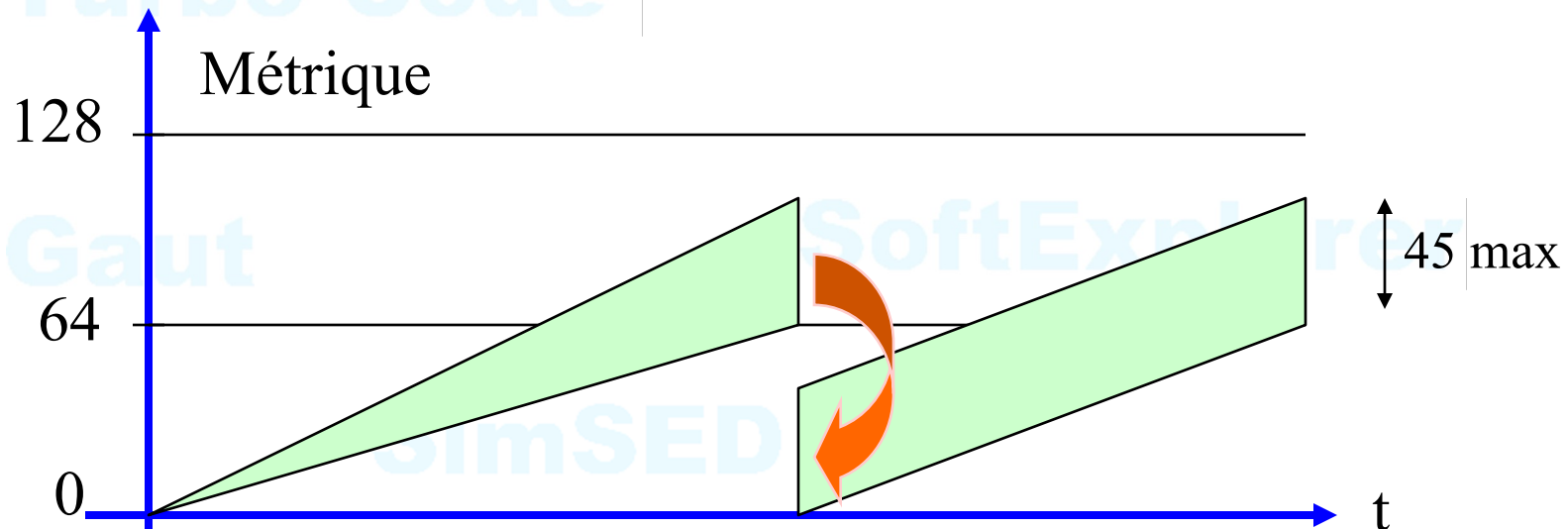


$$\text{Max} - \text{Min} \leq (K-1).M$$

Réechelonnage(2)

Exemple : $M=15 \Rightarrow \text{Max-Min}=45$

Les métriques de nœud sont codées sur 7 bits. Quand tous les MSBs sont à 1 ils sont remis à 0 (réechelonnage).

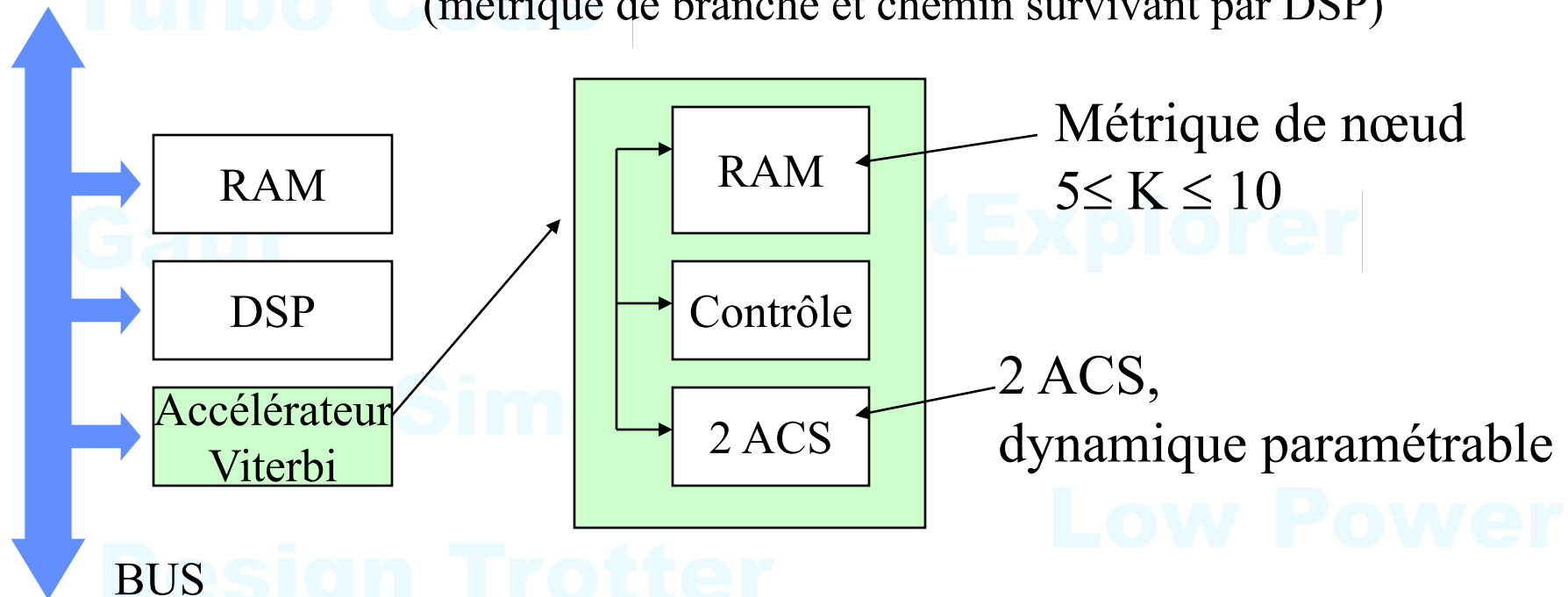


La fréquence de réechelonnage donne une information sur le bruit du canal.

Approche DSP + Accélérateur de Viterbi

Macrofonction paramétrable (IP) en VLSI (téléphonie mobile) :

Gestion automatique des métriques de noeud
(métrique de branche et chemin survivant par DSP)

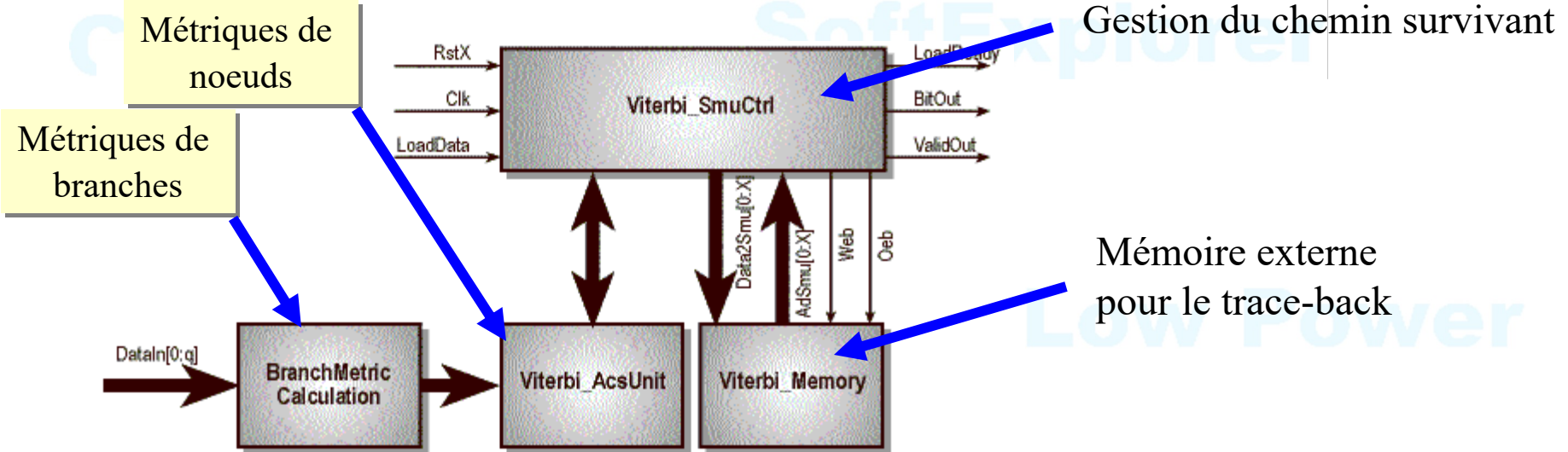


6000 portes($\forall K$) + mémoire, freqmax 80MHz

Exemple de réalisation

Macrofonction paramétrable (IP) en FPGA :

- ⇒ gestion du chemin survivant par trace-back
- ⇒ K, décision/souple/dure, nombre d'ACS paramétrables
- ⇒ Si K=5, décision dure, 4 ACS, mémoire externe => 1000 cellules(10000 portes)



Exemple de circuit

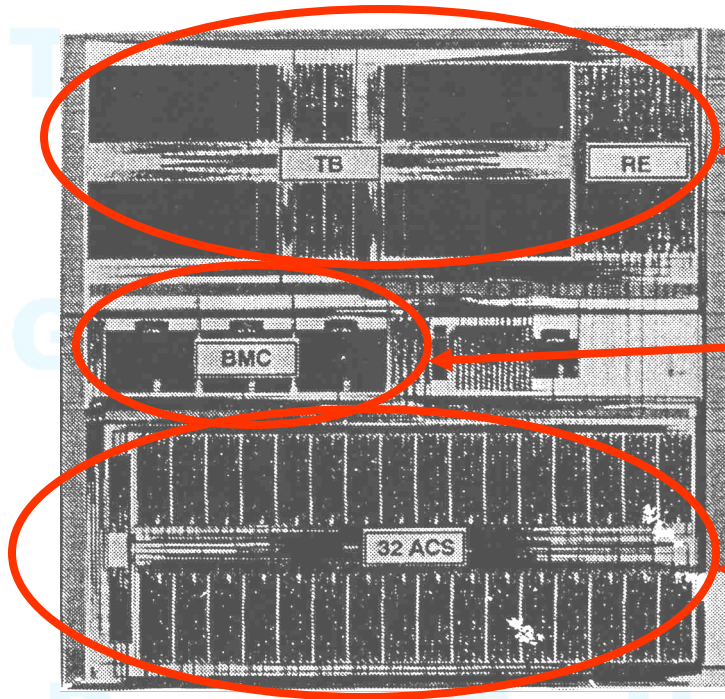
Décodeur pour la norme DVB
 $K=7$ $R=1/2$ à $7/8$
Soft decision

Gestion du chemin survivant
par architecture mixte
(trace-back/exchange register)

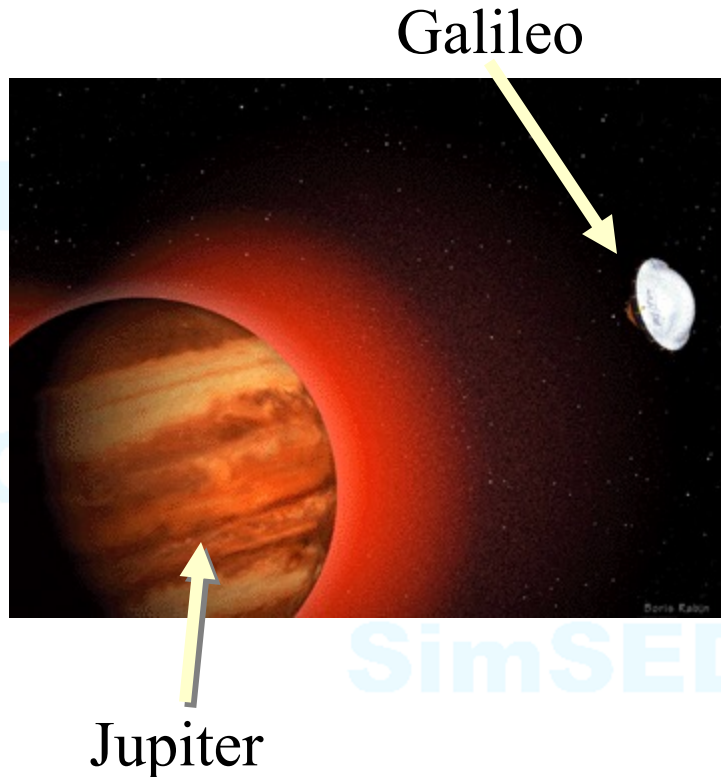
Métriques de
branches

Métriques de
noeuds

Taille 100mm^2
 $F_{\text{max}}=60\text{MHz}$



La sonde Galileo



- Communication 128kb/s
- SNR=0dB
- code convolutif $K=15$, $r=1/7$
=> 16000 états, 10 Gop/s
- 12 stations terrestres avec des décodeurs de Viterbi de 1 MF (128 processeurs)

Anecdote : le décodeur n'était pas faisable à l'époque du lancement (1989)

PLAN

❶ Code convolutif et son décodage

❶ Codeur convolutif

❶ Viterbi

❶ MAP

❷ Turbo-code

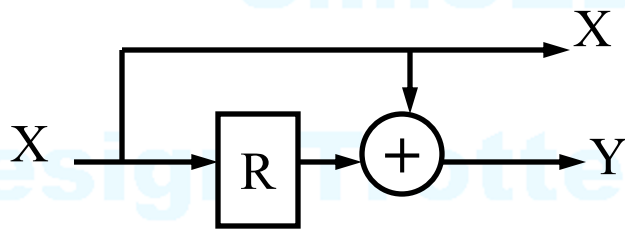
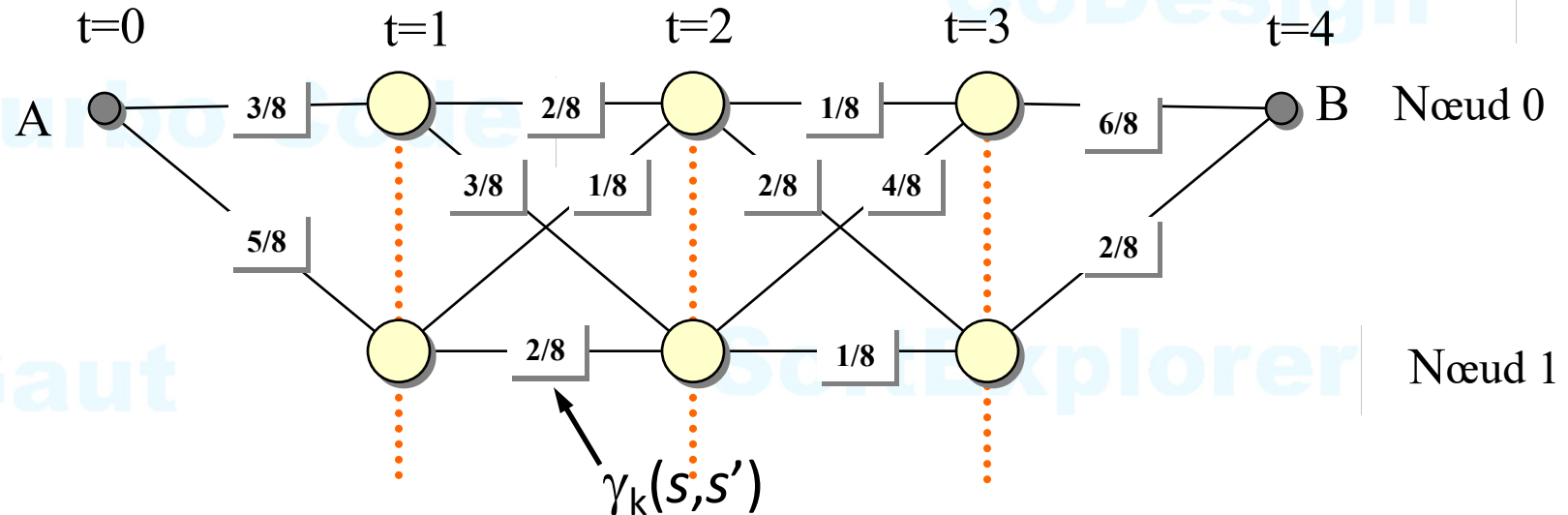
❸ Code LDPC

❹ Décodeurs haut débits

❹ Conclusion

Algorithme Forward-Backward

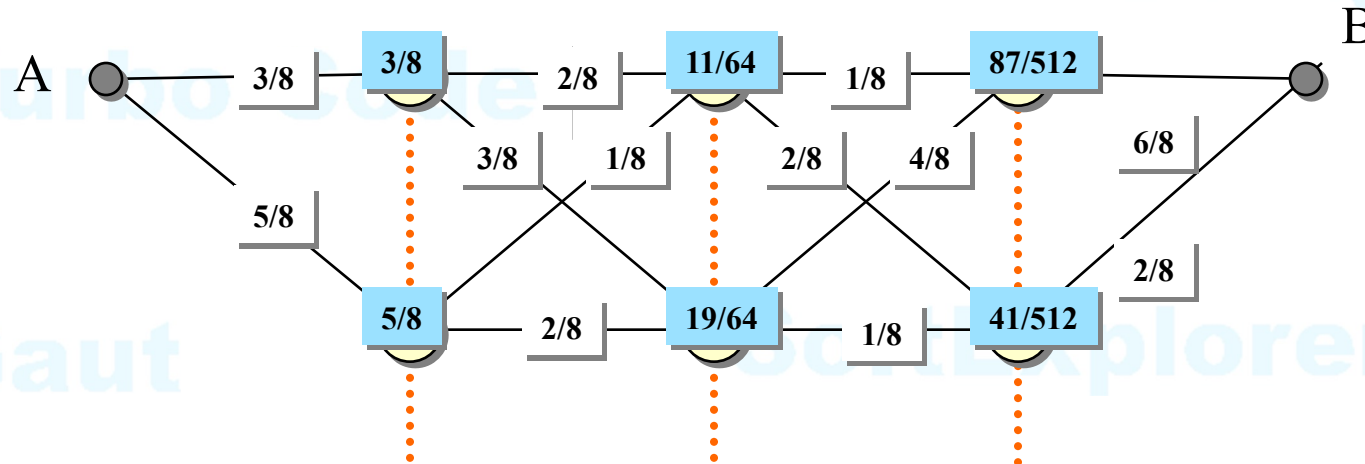
Les métriques de branches correspondent à des probabilités



Probabilité a priori de passer du nœud S au nœud S'

Algorithme Forward-Backward

Les métriques de nœud correspondent à des probabilités



Itération Forward :

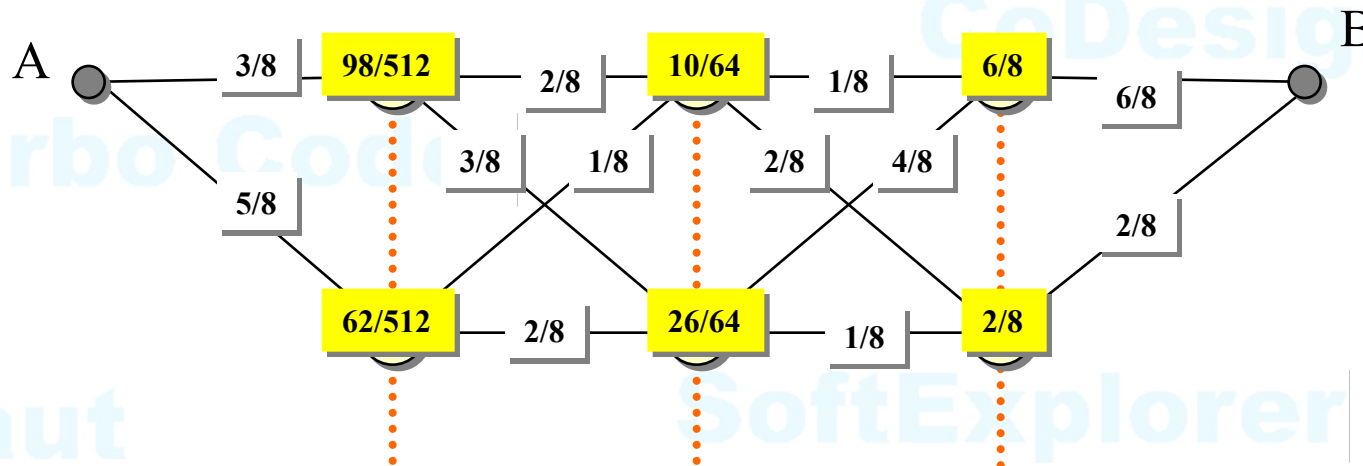


$$\alpha_k(s) = \alpha_{k-1}(s_0) \underbrace{\gamma_{k-1}(s_0, s)}_{\text{Probabilité a priori de passer du nœud } S_0 \text{ au nœud } S} + \alpha_{k-1}(s_1) \gamma_{k-1}(s_1, s)$$

Probabilité d'être au nœud S au temps k

Probabilité a priori de passer du nœud S_0 au nœud S

Algorithme Forward-Backward



Itération Backward :

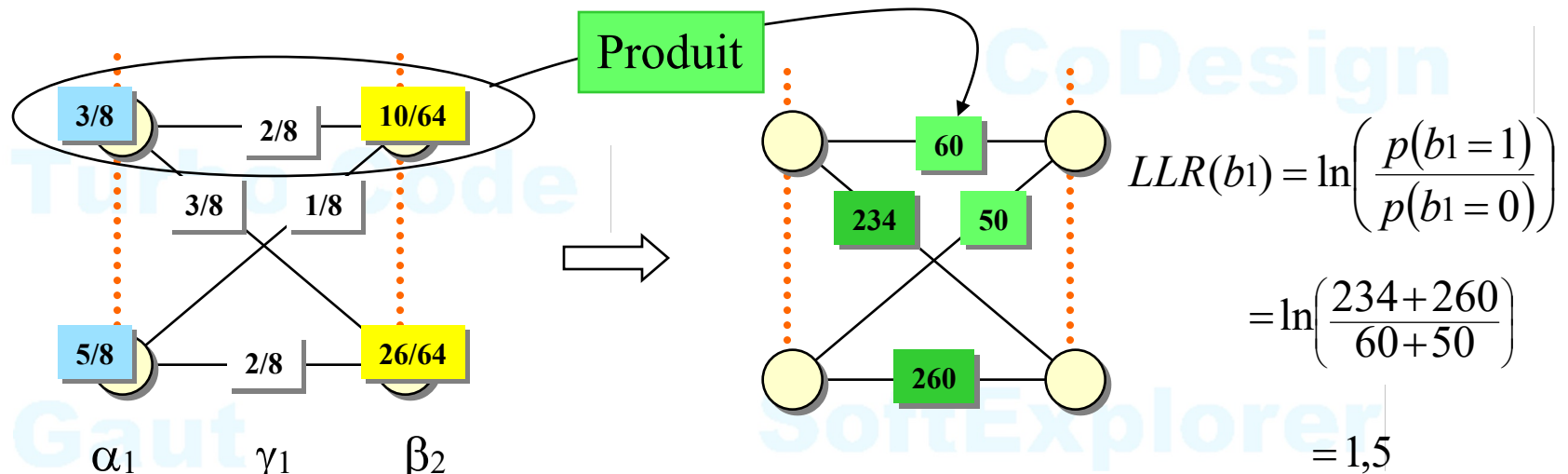


$$\beta_{k-1}(s) = \beta_k(s_0) \underbrace{\gamma_{k-1}(s_0, s)}_{\text{Probabilité à priori de passer du nœud } S_0 \text{ au nœud } S} + \beta_{k-1}(s_1) \gamma_{k-1}(s_1, s)$$

Probabilité d'être au nœud S au temps $k-1$

Probabilité à priori de passer du nœud S_0 au nœud S

Calcul de la fiabilité du bit



$$LLR_k = \ln \left(\frac{\sum_{(s,s')/u_k=1} (\alpha_{k-1}(s') \gamma_k(s',s) \beta_k(s'))}{\sum_{(s,s')/u_k=0} (\alpha_{k-1}(s') \gamma_k(s',s) \beta_k(s'))} \right)$$

IP Problèmes d'architecture

o Puissance de calcul

- métriques de branche : calcul d'exponentielles
- itération F et B : par symbole : 2^{K+1} mult. et 2^K add.
- Calcul du LLR : par symbole : 2^{K+1} mult. et 2^K-2 add.

o Dynamique très importante

- accroissement sans fin

o Besoins élevés en mémoire

- mémoires pour effectuer les itérations F et B
- dépend du temps de convergence

Opérations dans le domaine des log

Les calculs ont une complexité réduite dans le domaine des Log :

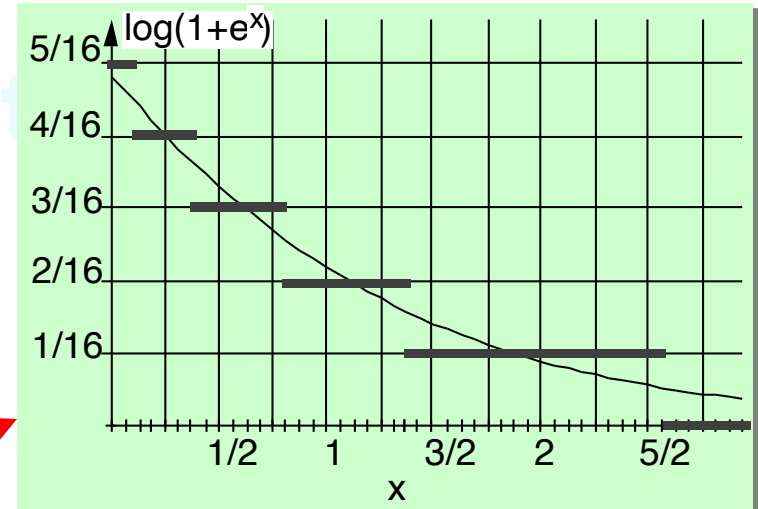
Si $X = \log(x)$ $Y = \log(y)$ $Z = \log(z)$

Multiplication : $z = x * y \Rightarrow Z = X + Y$

Addition : $z = x + y \Rightarrow Z = \text{MAX}^*(X, Y)$

$$\begin{aligned} Z &= \ln(z) = \ln(x + y) = \ln(e^X + e^Y) \\ &= \ln[e^{\text{MAX}(X, Y)} \cdot (1 + e^{\text{MIN}(X, Y) - \text{MAX}(X, Y)})] \\ &= \text{MAX}(X, Y) + \ln(1 + e^{-|X - Y|}) \\ &= \text{MAX}^*(X, Y) \end{aligned}$$

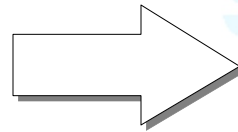
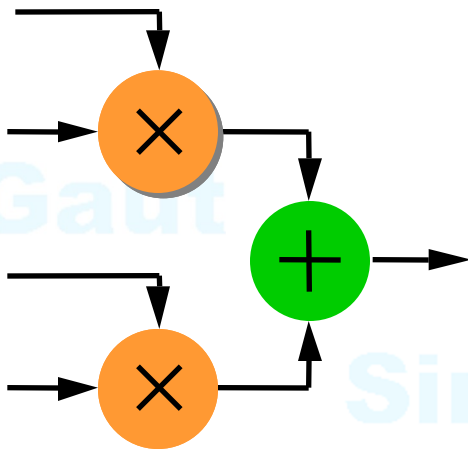
$\text{MAX}^* = \text{MAX} + \text{Facteur de correction}$



Architecture de MAX*

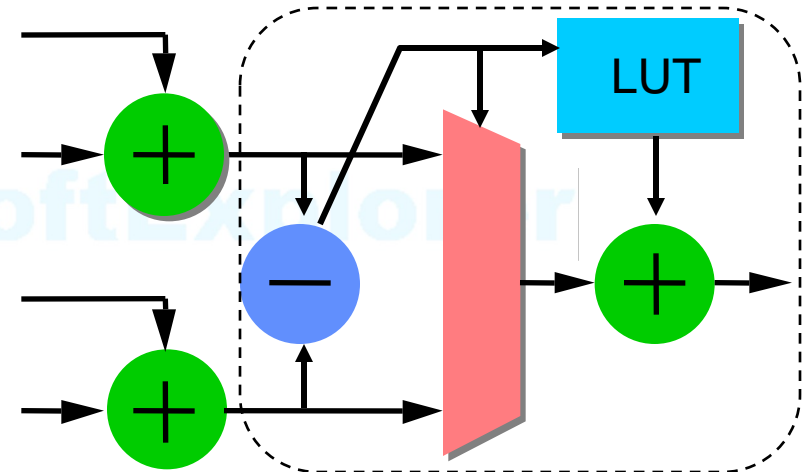
Calculs des récurions A_k et B_k :

$$\alpha_k(s) = \alpha_{k-1}(s_0)\gamma_{k-1}(s_0,s) + \alpha_{k-1}(s_1)\gamma_{k-1}(s_1,s)$$



$$A_k = \log(\alpha_k) \quad B_k = \log(\beta_k) \quad G_k = \log(\gamma_k)$$

$$A_k(s) = \text{MAX}^*[A_{k-1}(s_0) + G_{k-1}(s_0,s), A_{k-1}(s_1) + G_{k-1}(s_1,s)]$$

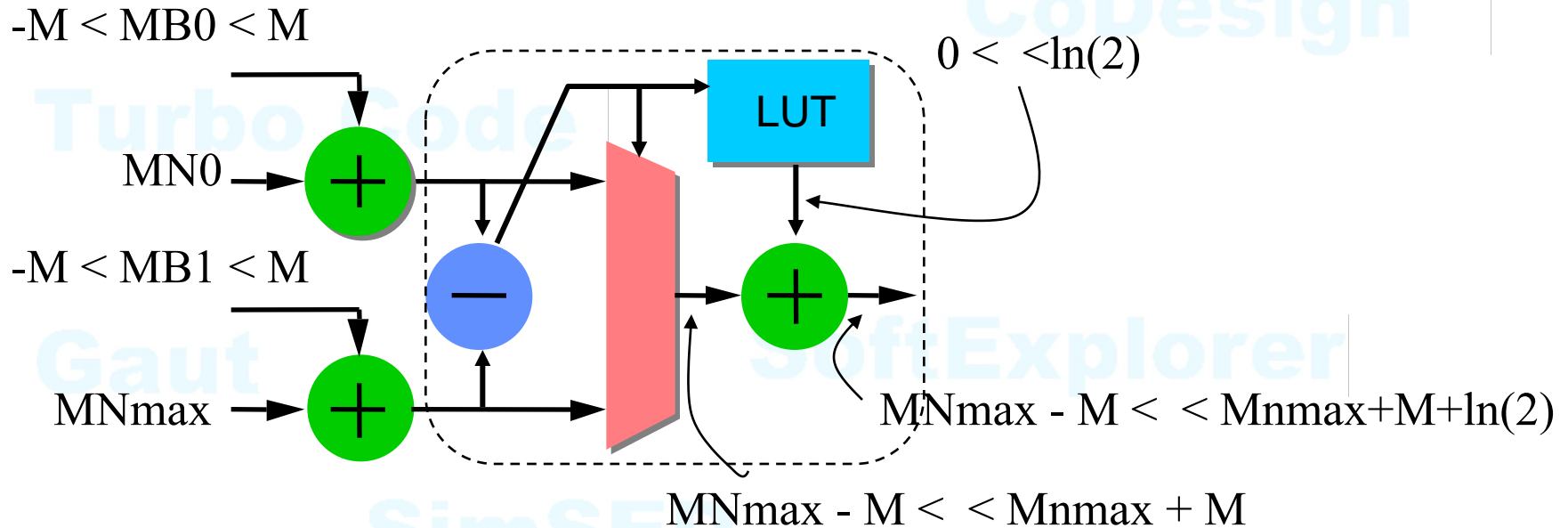


MAX*

Calculs du LLR :

$$L(u_{k-1}) = \text{MAX}^*(s,s')/u_{k=1}[A_{k-1}(s) + G_{k-1}(s,s') + B_{k-1}(s')] \\ - \text{MAX}^*(s,s')/u_{k=0}[A_{k-1}(s) + G_{k-1}(s,s') + B_{k-1}(s')]$$

Dynamique des données



Cela implique (voir Viterbi) la relation $Mn_{max} - Mn_{min} < (K-1).(2M + \ln(2))$

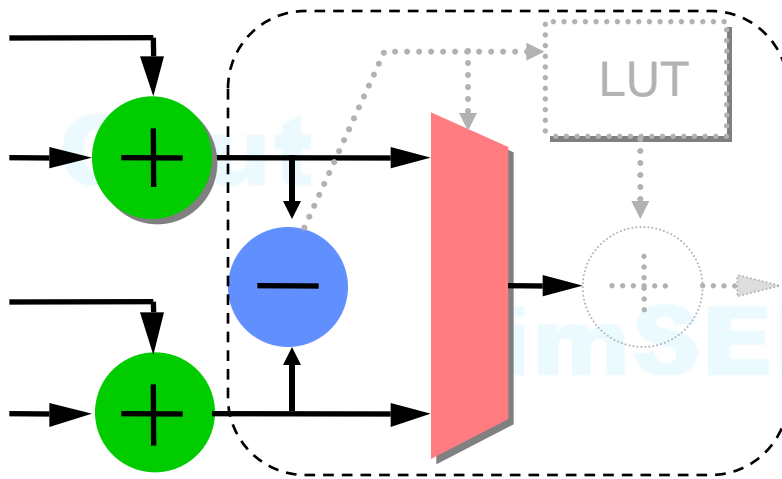
=> Utilisation de la technique de ré-échantillonnage

Note : Qualité canal = λ * Fréquence ré-échantillonnage

Problème de l'opérateur Max*

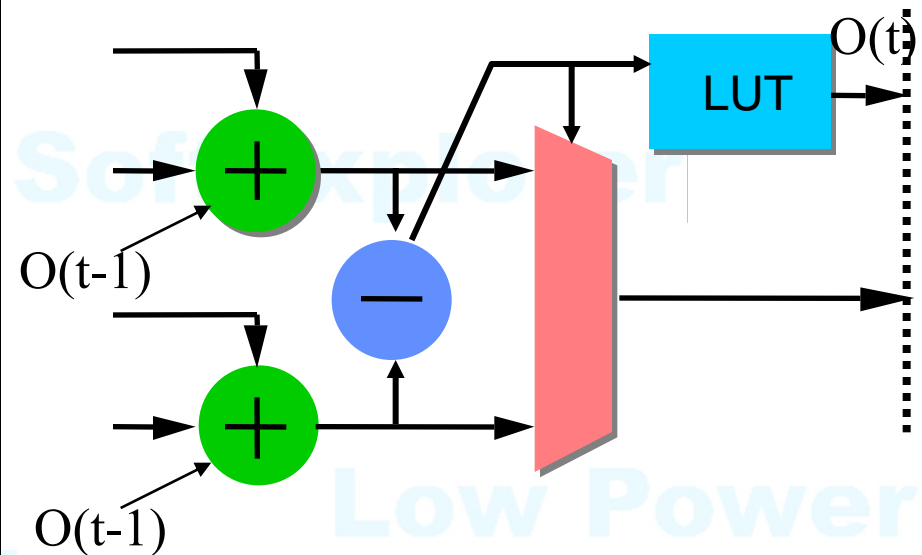
Par rapport à un ACS classique, le chemin critique est allongé

Solution algorithmique :
Supprimer le facteur correctif



Algo : Max-log-Map

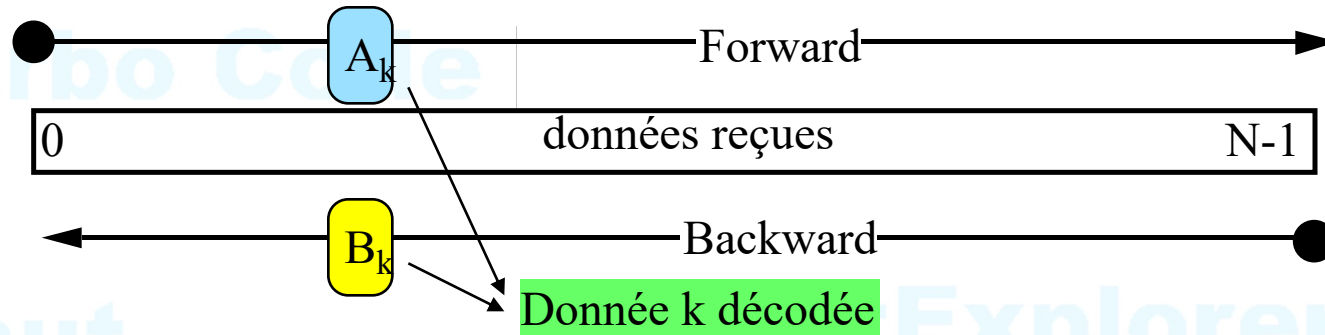
Solution architectural : ACSO -> OACS
Retarder d'un cycle addition Offset



+ 15 % de complexité

Besoin de mémoire

Problème de rendez-vous de données générées en ordre chronologique inverse



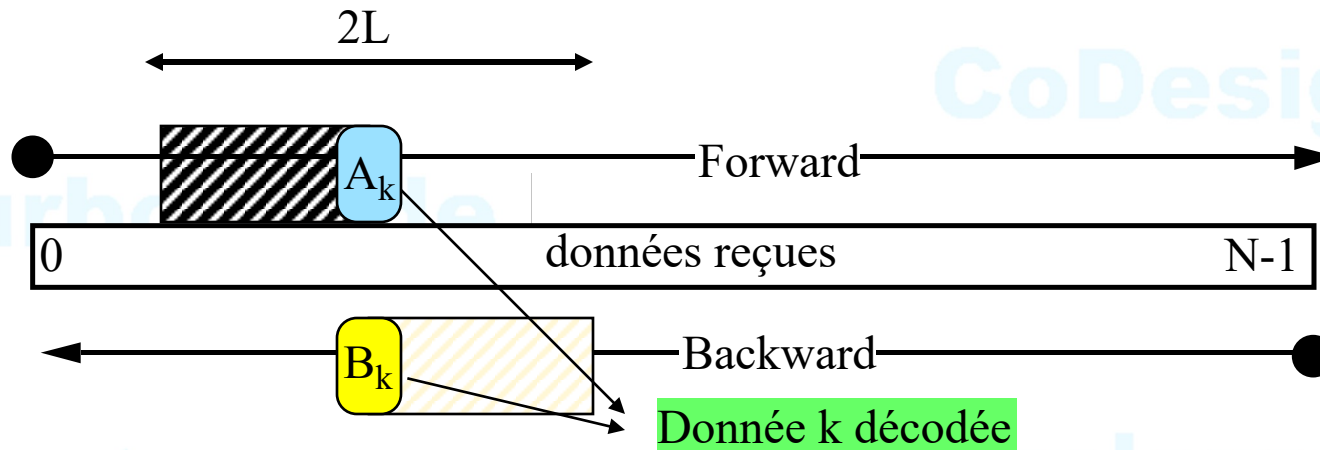
Approche directe : Deux calculs par symbole (avant et arrière)

Latence de décodage maximale de $2N$

Mémoire de N vecteurs A_k .

Typiquement : $N \gg 1000$, A_k est composé de 2^{K-1} métriques de nœuds
($K=5$, MN sur 10 bits $\Rightarrow A_k$ contient 160 bits).

Propriété de convergence

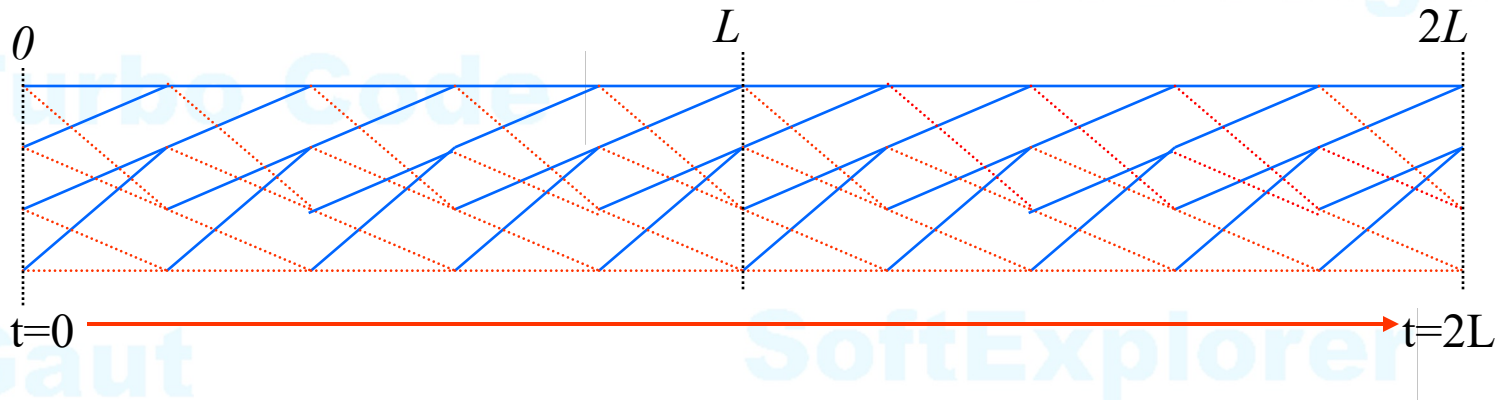


théorème de convergence : Le calcul du bit k n'est pas dégradé si les récursions Forward et Backward démarrent respectivement avant $k-L$ et après $k+L$.

Cette propriété permet d'anticiper le décodage des bits
 $\Rightarrow$ Economie de mémoire.

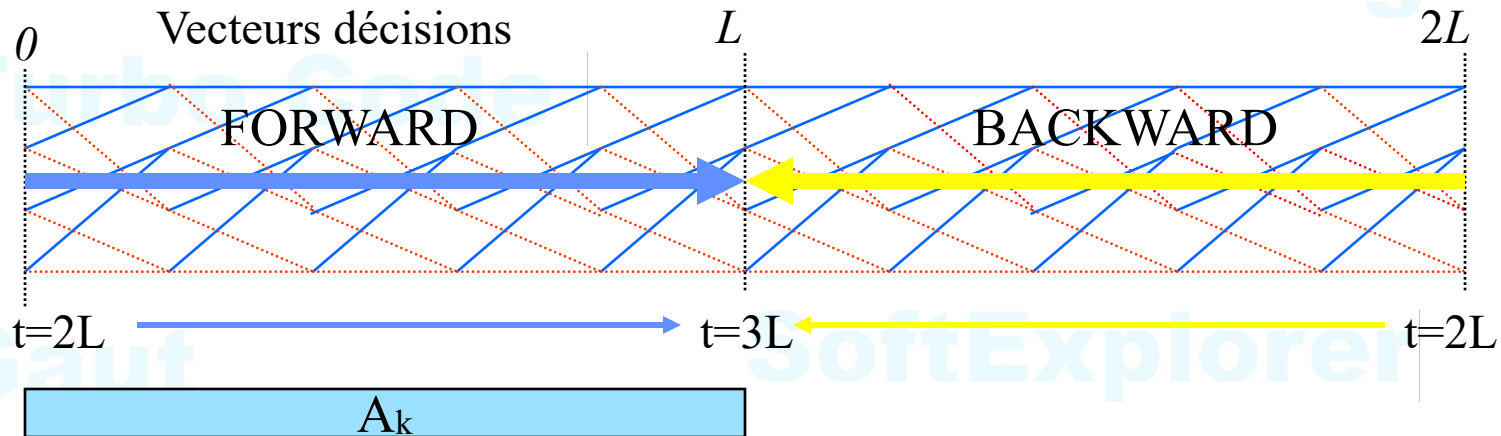
Chronologie des calculs

Période 1 ($0 < t < 2L$) : Les symboles d'entrée sont stockés en mémoire.



Chronologie des calculs

Période 1 ($0 < t < 2L$) : Les symboles d'entrée sont stockés en mémoire.



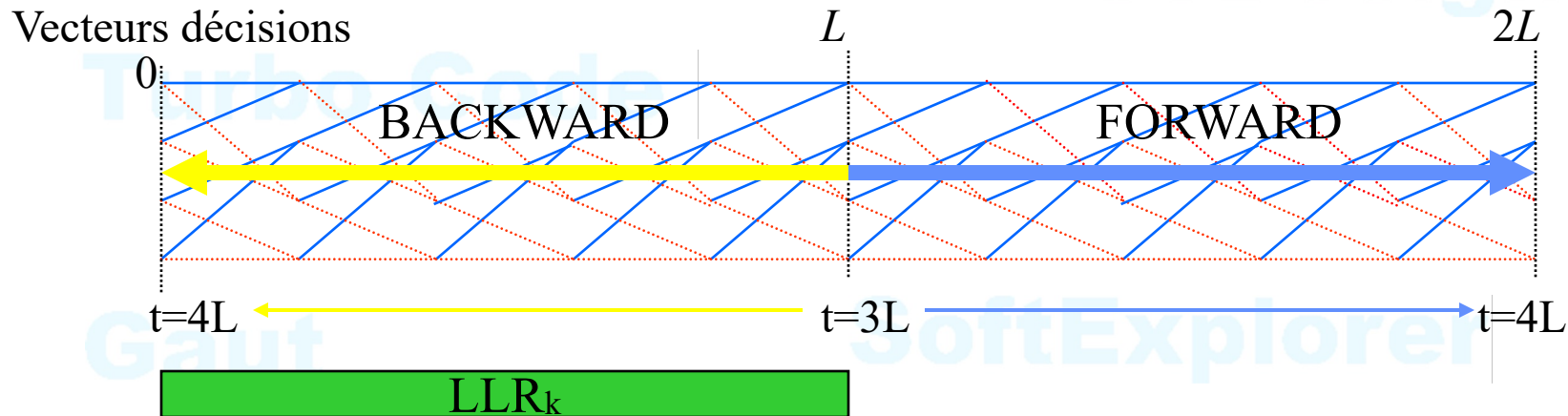
Période 2 ($2L < t < 3L$) : Itération Backward effectuée de $2L$ à $L \Rightarrow B_L$.

Simultanément, itération Forward effectuée entre donnée 0 à L .

Les L vecteurs A_k générés sont mémorisés.

Chronologie des calculs

Période 1 ($0 < t < 2L$) : Les symboles d'entrée sont stockés en mémoire.



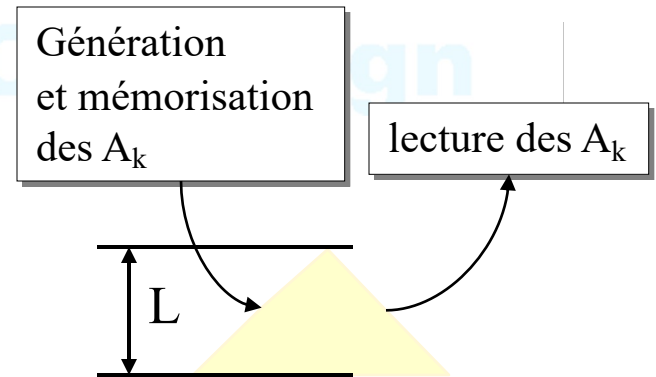
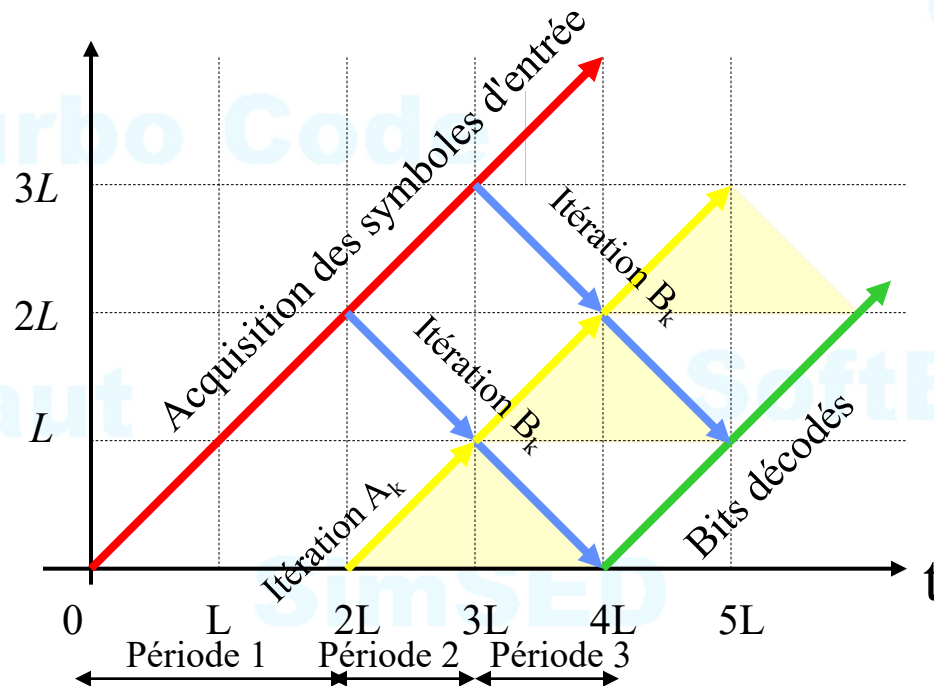
Période 2 ($2L < t < 3L$) : Récursion Backward effectuée de $2L$ à $L \Rightarrow B_L$.

Simultanément, récursion forward effectuée entre données 0 à L . Les L vecteurs A_k générés sont mémorisés.

Période 3 ($3L < t < 4L$) : Récursion Backward effectuée de L à 0 . A chaque itération le vecteur A_k est lu et utilisé avec B_{k+1} pour calculer le $LLR(b_k)$

Chronologie des calculs et mémorisation

Représentation graphique :



Latence de décodage : $4L$,
Trois itérations/symbole
Mémoire : L vecteurs A_k

Typiquement, mémoire $L=24$, $A_k = 160$ bits. Il existe d'autres compromis
Puissance de calcul / mémoire.

Comparaison Viterbi, SOVA, FB

	MB	Itération	Gestion mémoire	Latence	Surface	Sortie
Viterbi (TB)	Appl.	2^{K-1} ACS	3 RAM $L \times 2^{K-1}$	4L	<1	bit
Viterbi (ER)	Appl.	2^{K-1} ACS	$L \times 2^{K-1}$ Registres	L	1	bit
SOVA	Appl.	2^{K-1} ACS 2^{K-1} demi-ACS	$L \times 2^{K-1}$ Registres $L/2 \times 2^{K-1}$ Registres $L/2$ mise à jours W	$L + L/2$	2	bit ~Fiabilité
Forward Backward	Appl.	MAX*=1,2 ACS $3 \times 2^{K-1}$ MAX*	$L \times (n \cdot 2^{K-1})$ (Registre ou RAM)	4L	5	bit Fiabilité

Il existe des algorithmes de Viterbi sous-optimaux

Il existe différents compromis « itération - mémoire - latence »

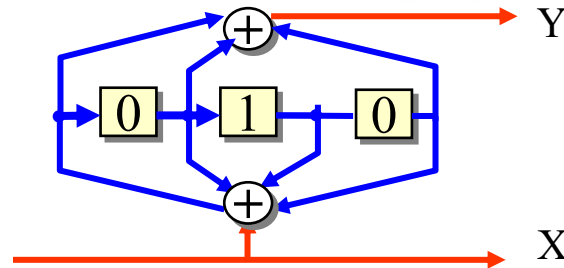
PLAN

- ❶ Code convolutif et son décodage
- ❷ Turbo-code
- ❸ Code LDPC
- ❹ Décodeurs haut débits
- ❹ Conclusion

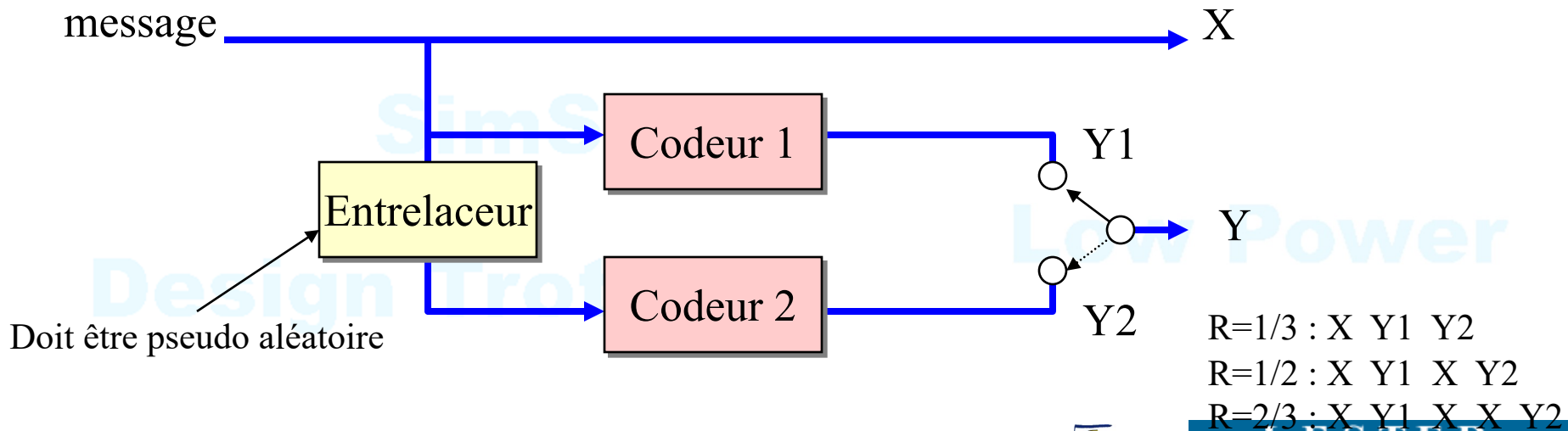
Turbo code

Les codeurs récurifs systématiques permettent de décoder avec 2 SOVA en //

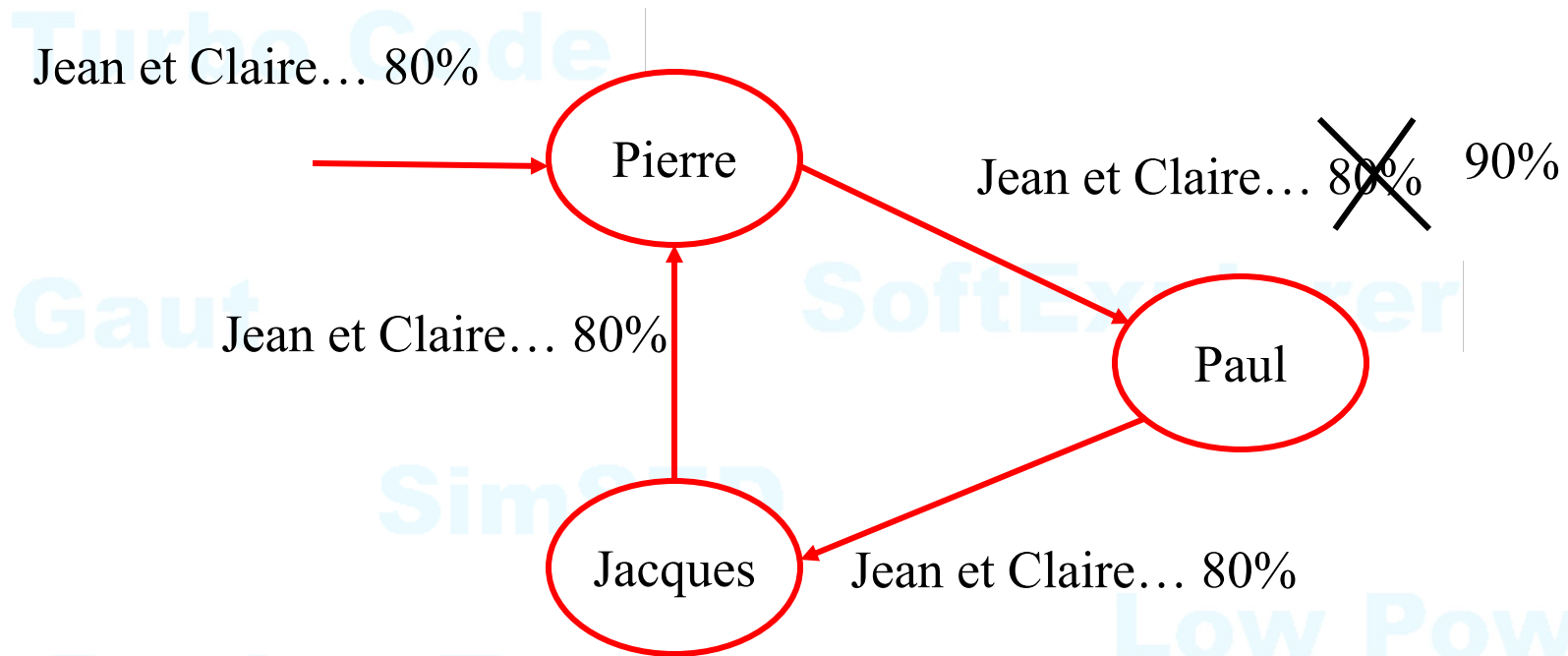
Exemple de codeur :



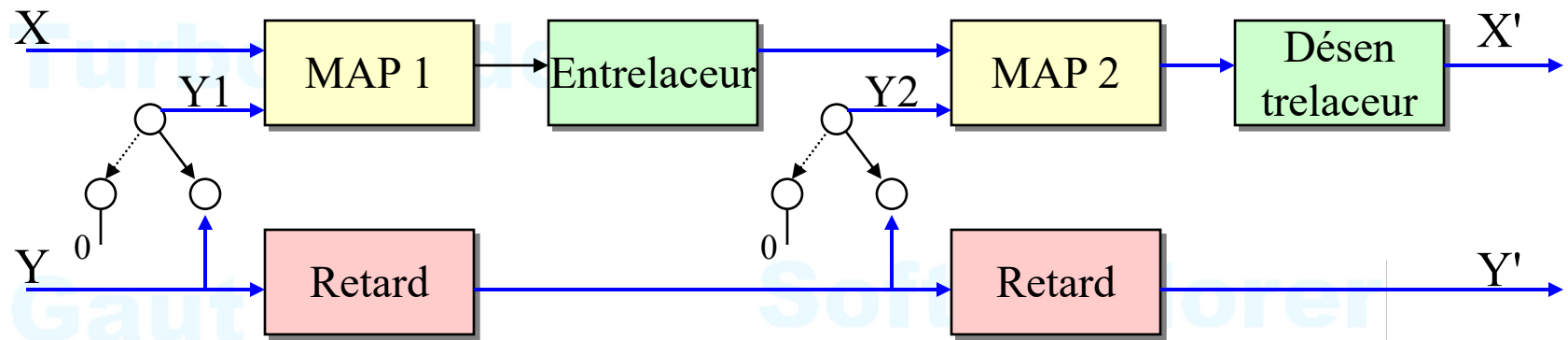
Exemple de codeur turbo :



Conseil : éviter les auto-confirmations



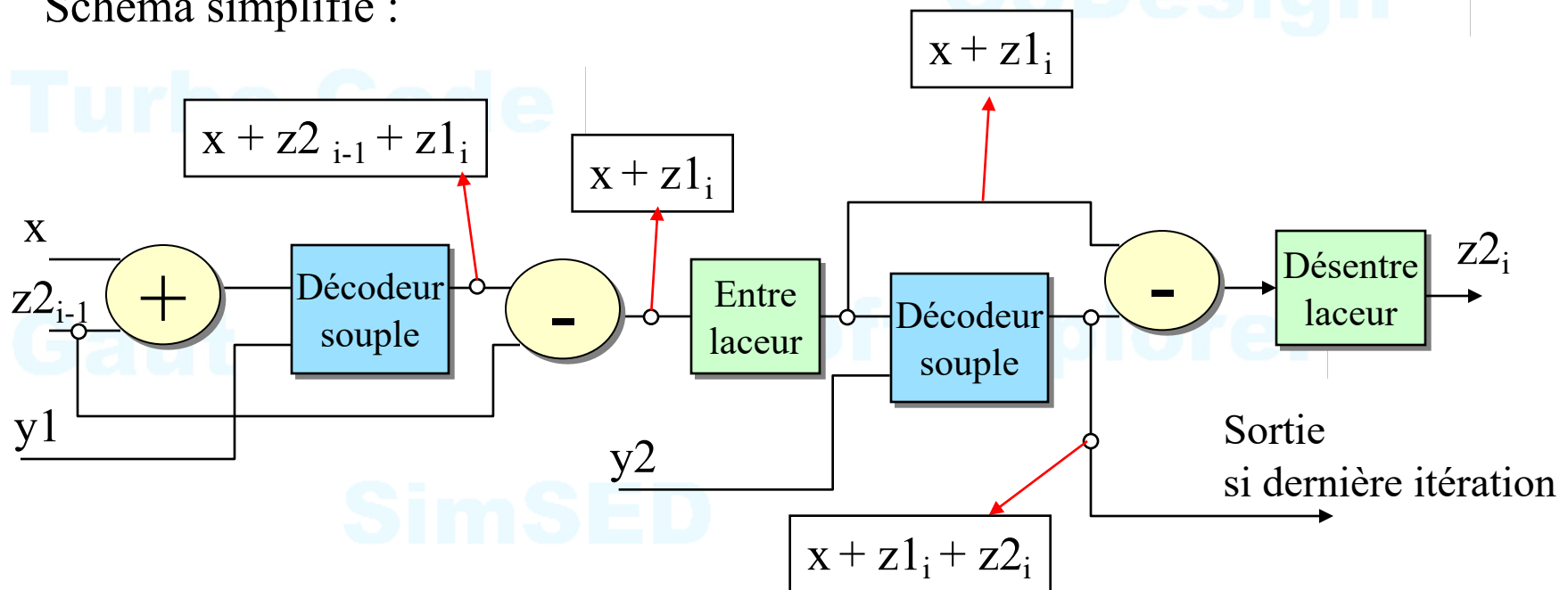
Architecture de décodage



Les décodeurs de Viterbi doivent être à sortie souple (SOVA ou Forward-Backward) pour générer l'information extrinsèque à l'entrée du prochain étage

Architecture détaillée

Schéma simplifié :



Z_i = Information « extrinsèque »

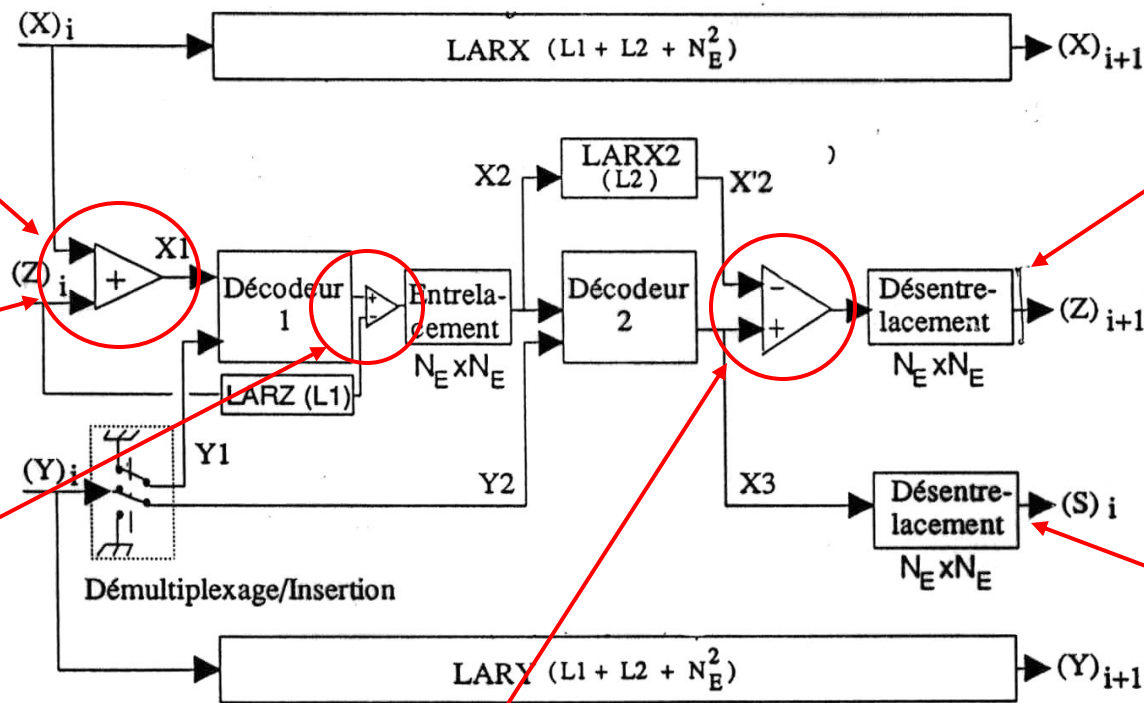
Il s'agit de l'information apportée par la redondance sur la valeur du bit courant

Architecture détaillée

Ajout de la contribution du décodeur précédent par Z

Z=info extrinsèque

On ne garde que la contribution du décodeur 1

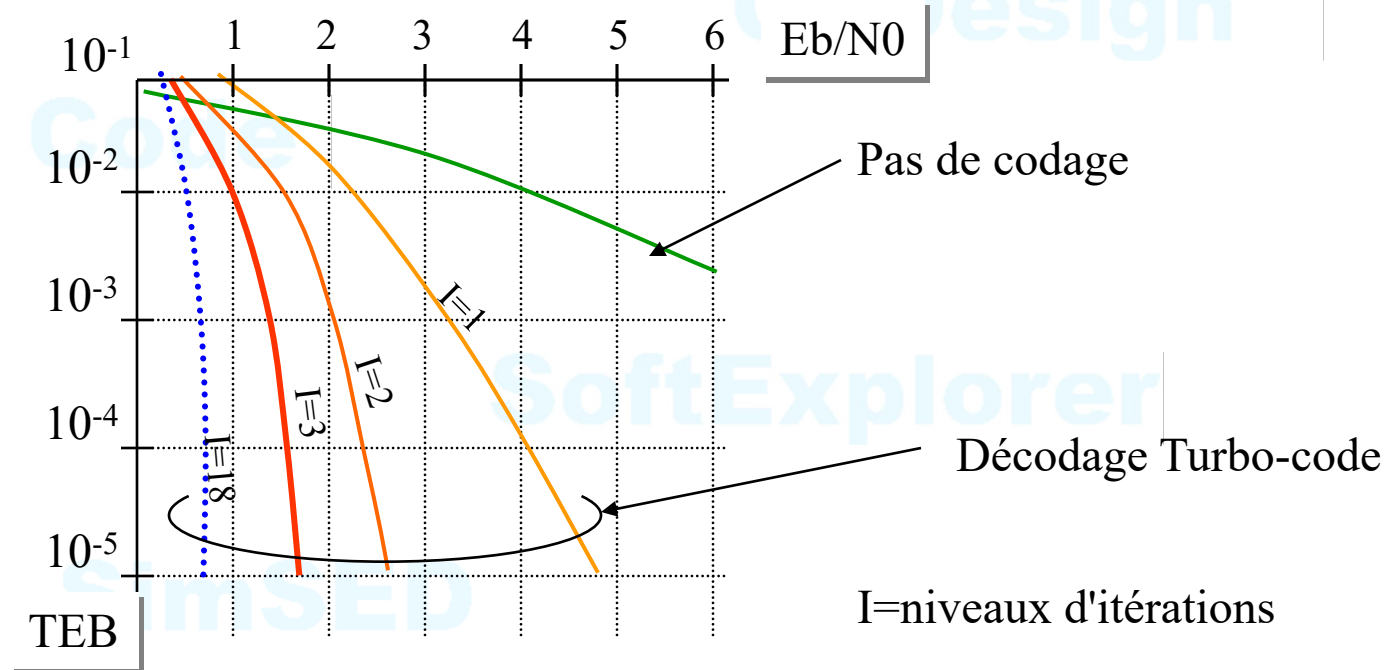


Z=info extrinsèque pour la prochaine itération

Sortie du décodeur

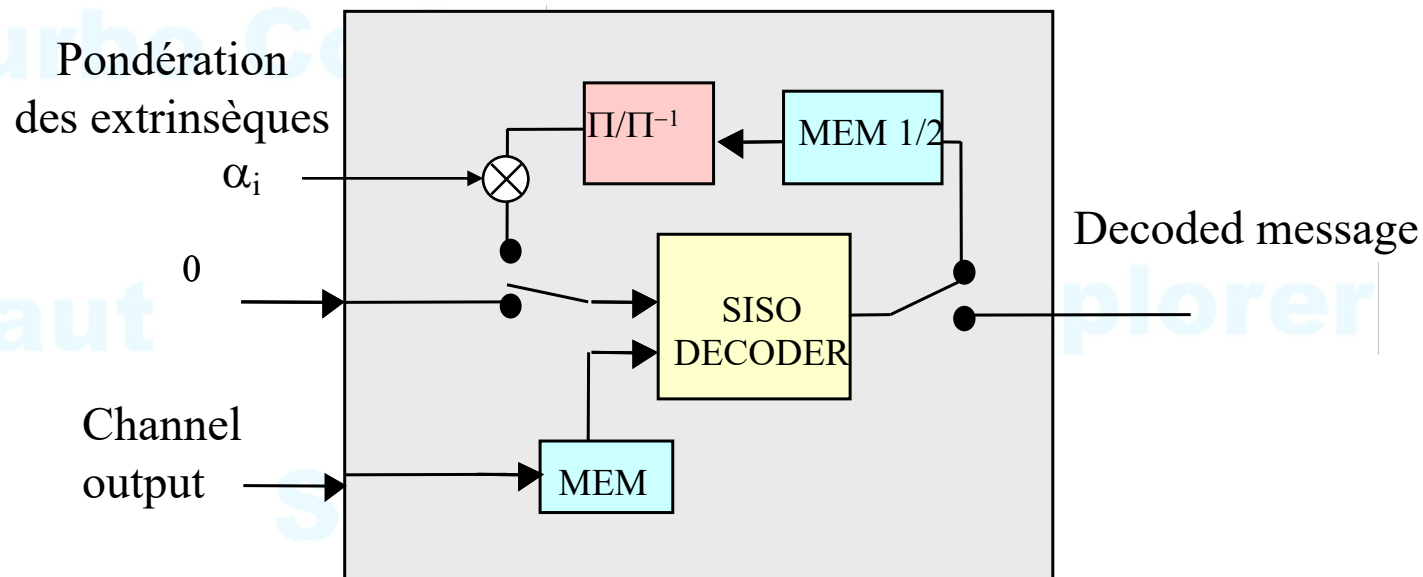
On ne garde que la contribution du décodeur 2

Performances



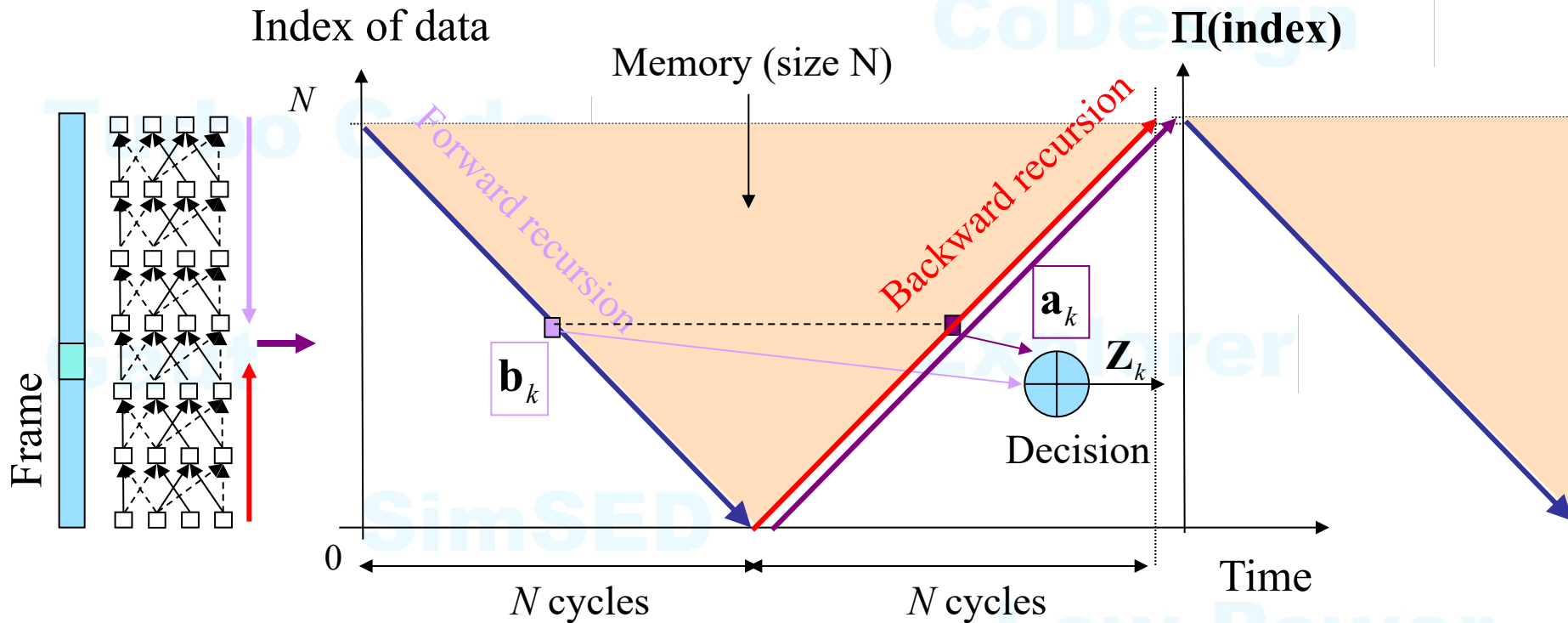
A partir d'un certain nombre d'itérations, les performances dépassent les codes concaténés Viterbi+RS

Classical architecture: serial architecture



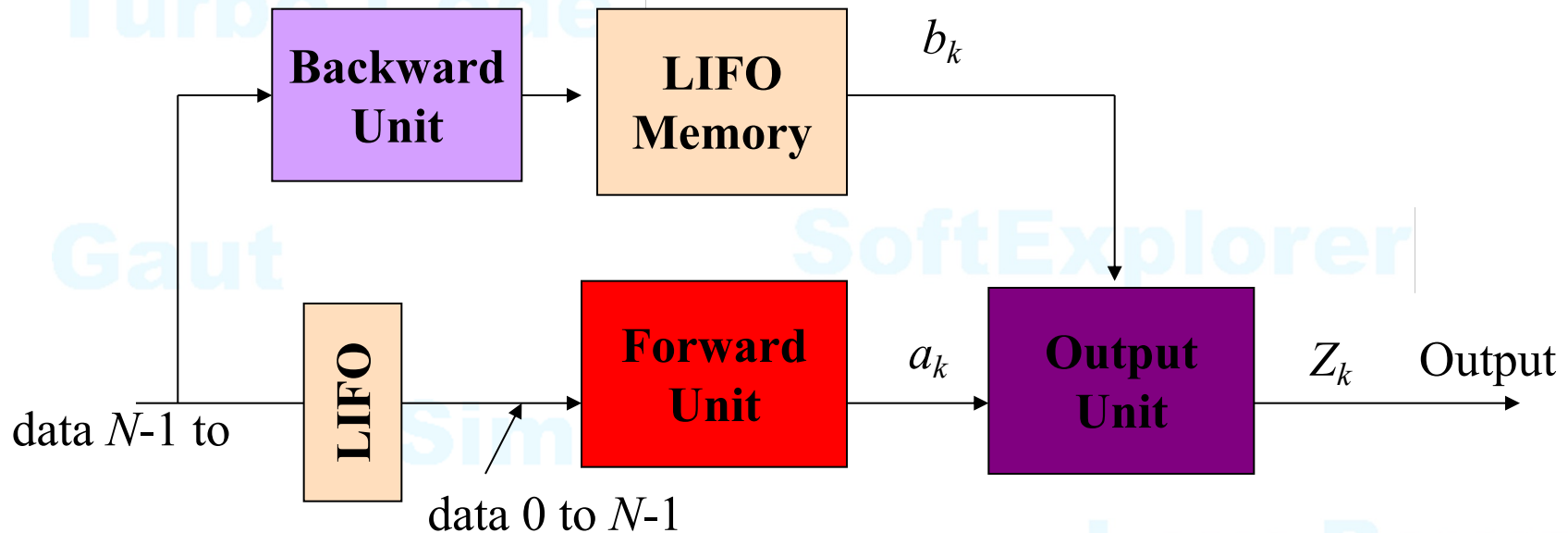
The SISO decoder performs SISO1, then SISO2,
then SISO 1 and so on...

SISO scheduling



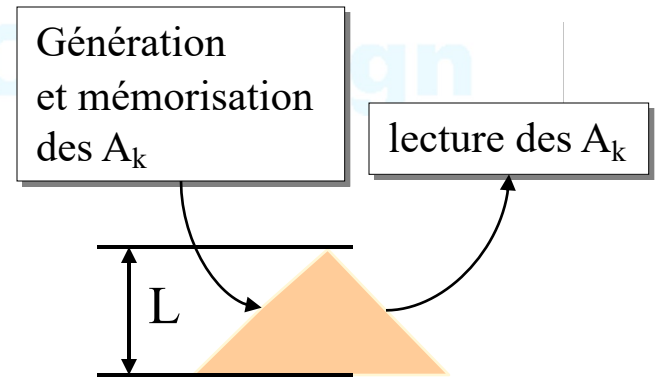
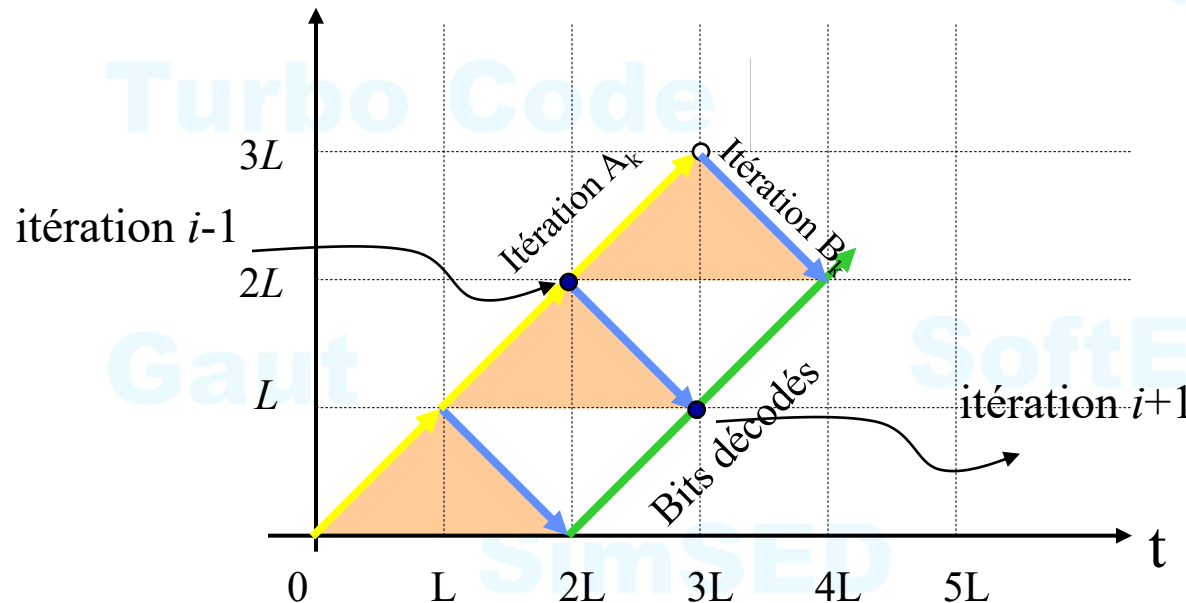
« Triangle scheduling »

SISO architecture



AAA: Technique "sliding window"

Représentation graphique :



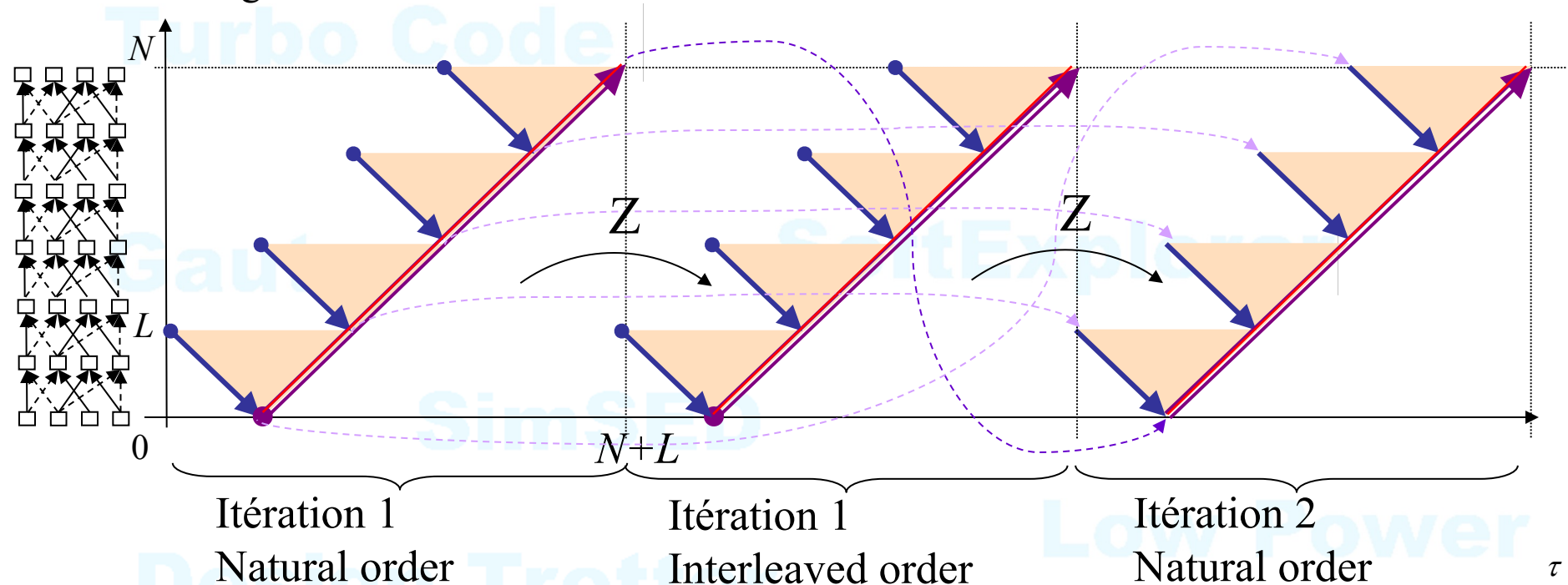
Latence de décodage : $2L$,
deux itérations/symbole
Mémoire : L vecteurs A_k

Gain de calcul sans perte significative de performances.

Sliding window algorithm

The triangle is split in N/L sub-triangles

Trellis stage



Save memory and latency

SISO processor architecture

o Entrée sortie

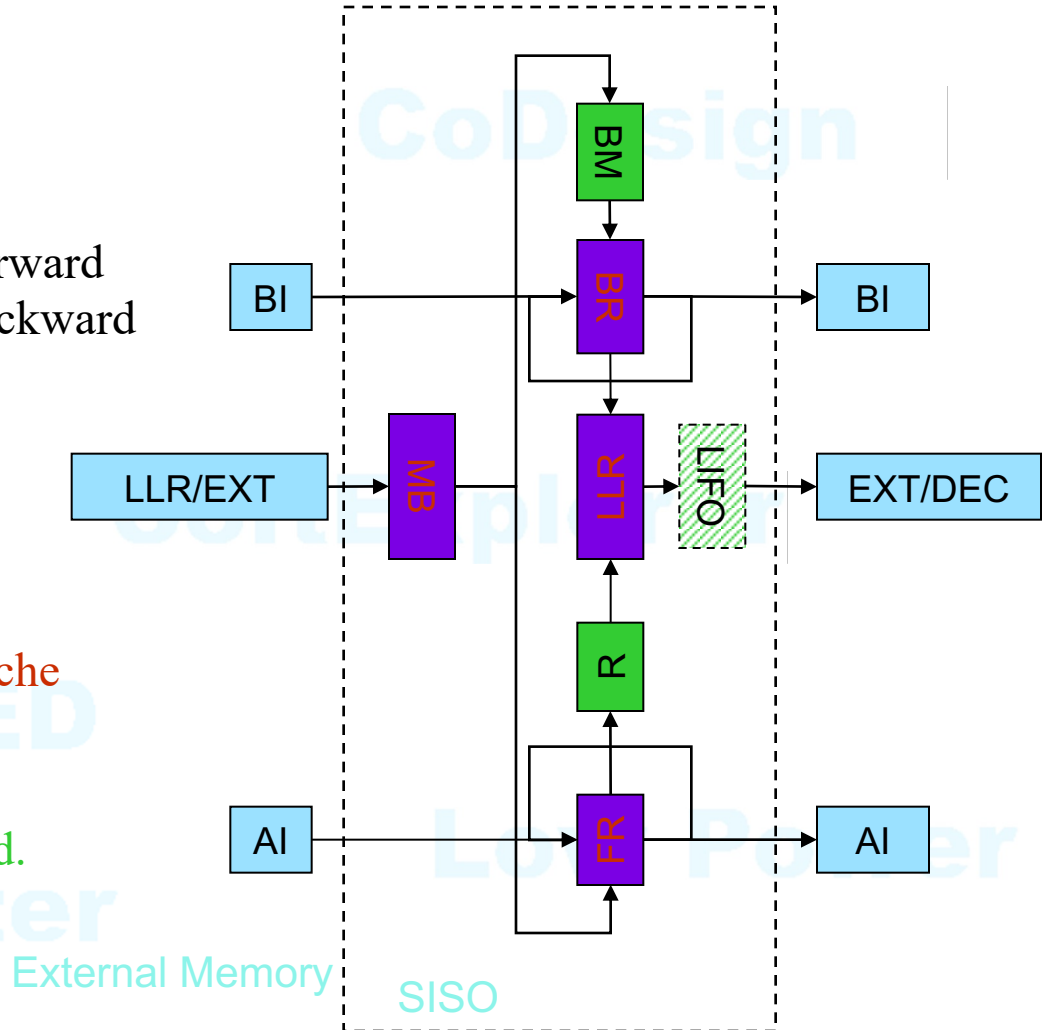
- **AI**: Initialisation récursion forward
- **BI**: Initialisation récursion backward
- **LLR/EXT**: Entrée souple

o Opérateur calcul :

- **BR**: recursion Backward
- **FR**: recursion Forward
- **LLR**: Calcul LLR
- **MB**: Calcul métrique de branche

o Mémoires :

- **BM**: Métrique de branche
- **R**: Métrique de noeud forward.
- **FILO** (ré-ordonnancement)



PLAN

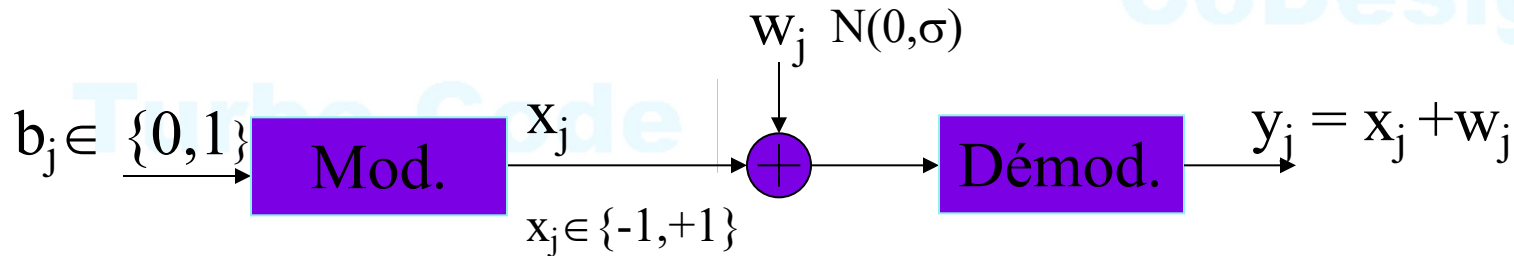
- ❶ Code convolutif et son décodage
- ❷ Turbo-code
- ❸ Code LDPC
- ❸ Contrainte de parité
- ❸ Code LDPC
- ❸ Architecture de décodage
- ❹ Problèmes communs
- ❺ Quelques résultats théoriques

Low Density Parity Check Code

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

- 1) Décodage d'un bit de parité**
- 2) Code LPDC**
- 3) Architecture pour les codes LDPC**

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":

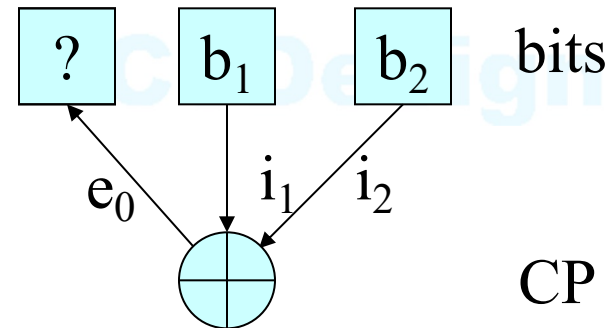
$$i_j = \ln(p(b_j=1/y_j)/p(b_j=0/y_j)) = 2y_j/\sigma^2$$

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 est appelé information *extrinsèque*

Le calcul de e_0 en fonction de i_1 et i_2 peut être fait soit directement, soit dans le domaine fréquentiel.

Calcul direct

On a $i_j = \ln\left(\frac{P(b_j = 1 / y_j)}{P(b_j = 0 / y_j)}\right)$ $p(b_j=0/y_j) + p(b_j=1/y_j) = 1$

Soit $P(b_j = 1 / y_j) = \frac{e^{i_j}}{1 + e^{i_j}}$ et $P(b_j = 0 / y_j) = \frac{1}{1 + e^{i_j}}$

$$e_0 = \ln\left(\frac{P(b_1 = 0 / y_1) \cdot P(b_2 = 1 / y_2) + P(b_1 = 1 / y_1) \cdot P(b_2 = 0 / y_2)}{P(b_1 = 0 / y_1) \cdot P(b_2 = 0 / y_2) + P(b_1 = 1 / y_1) \cdot P(b_2 = 1 / y_2)}\right)$$

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

Calcul fréquentiel

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

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

$$e_0 = \text{ifft}(\text{fft}(\mathbf{i}_1) \cdot \text{fft}(\mathbf{i}_2)) \quad \xrightarrow{\text{Convolution...}}$$

$$\text{fft}(\mathbf{i}_j) = \text{fft}(p(b_j=0/y_j), p(b_j=1/y_j)) = \left(1, \frac{1 - e^{i_j}}{1 + e^{i_j}}\right) = (1, -\tanh(i_j/2))$$

$$e_0 = \text{ifft}(\text{fft}(\mathbf{i}_1) \cdot \text{fft}(\mathbf{i}_2)) = \frac{1}{2} (1 + \tanh(i_0/2) \cdot \tanh(i_1/2), 1 - \tanh(i_0/2) \cdot \tanh(i_1/2))$$

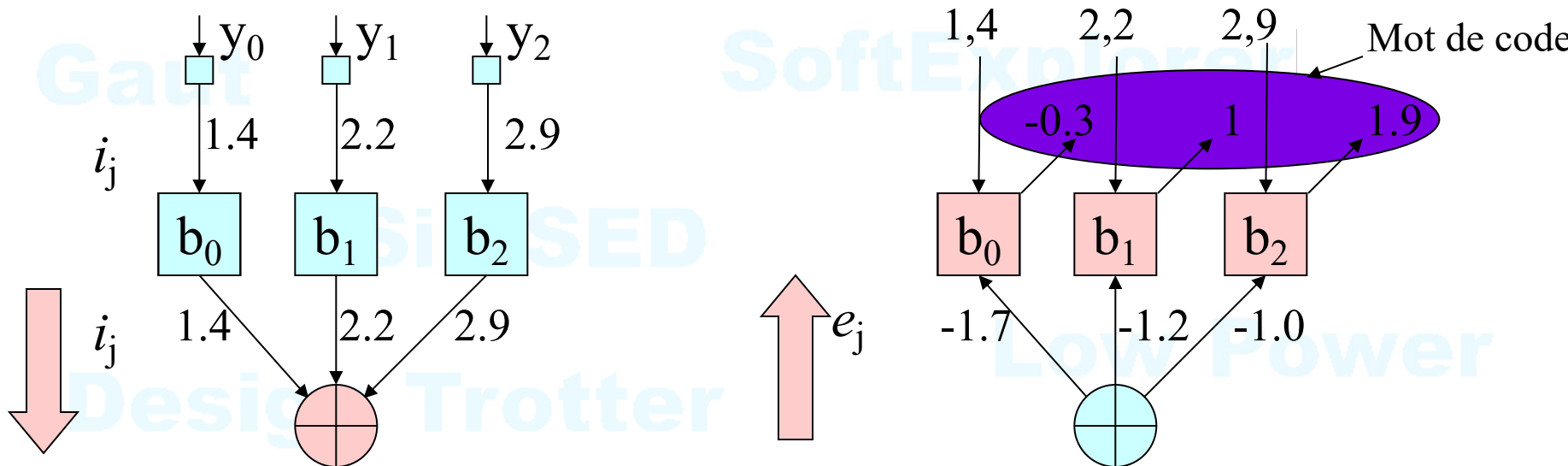
$$e_0 = \ln \left(\frac{1 - \tanh(i_0/2) \cdot \tanh(i_1/2)}{1 + \tanh(i_0/2) \cdot \tanh(i_1/2)} \right) = \Phi(\Phi(i_0) + \Phi(i_1))$$

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

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.



Pourquoi le nom de LDPC ?

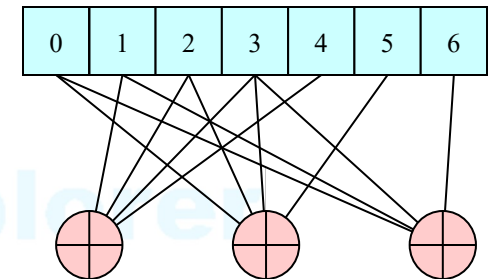
Représentation algébrique :

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

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

$\xleftarrow{\hspace{10em}} N = \text{number of bits}$

$P = \text{number of PC}$
 Check Matrix

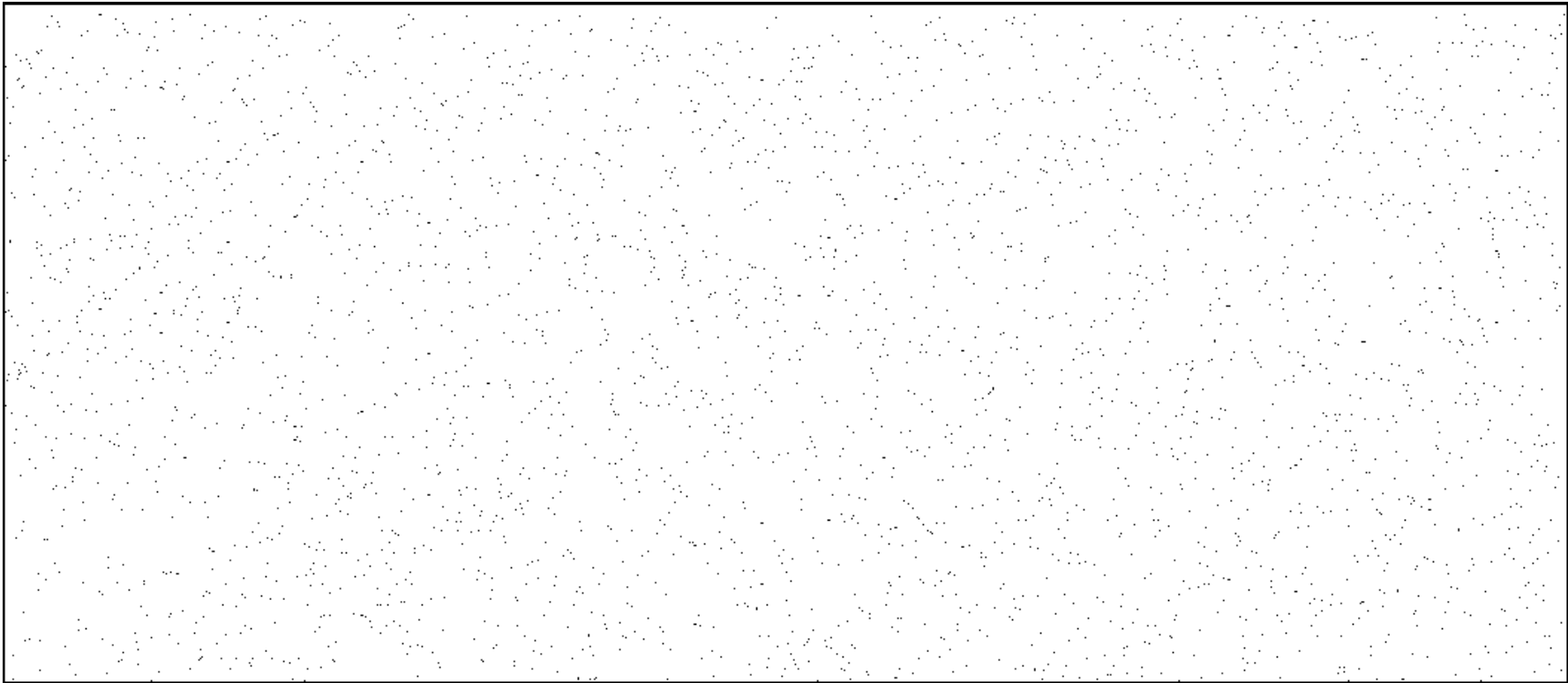


Nombre de 1 sur une ligne = d_c = nombre de bit associé au PC

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

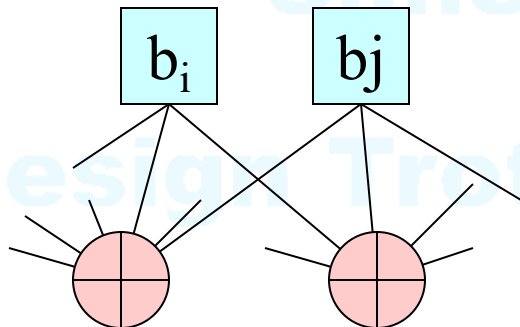
Sélection de la répartition optimale des spectres de répartition des poids des branches (en utilisant des EXIT Chart):

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 :

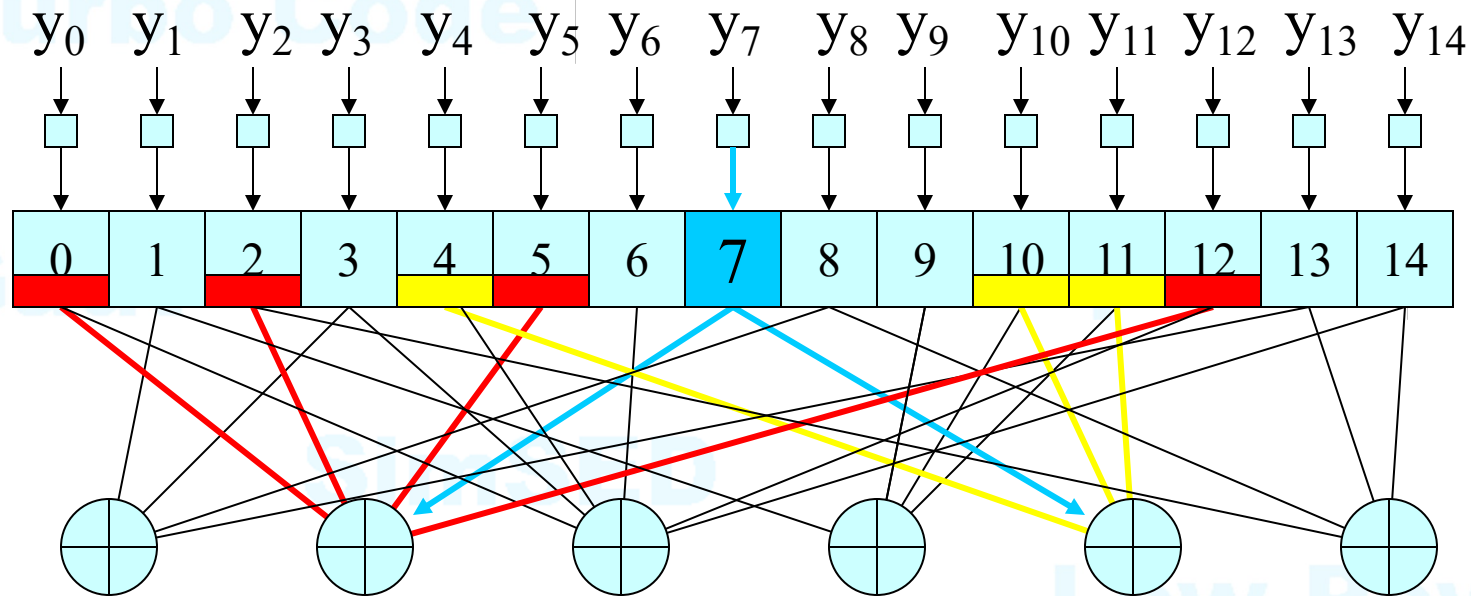


...et on obtient un bon code

Algorithme à propagation de croyance

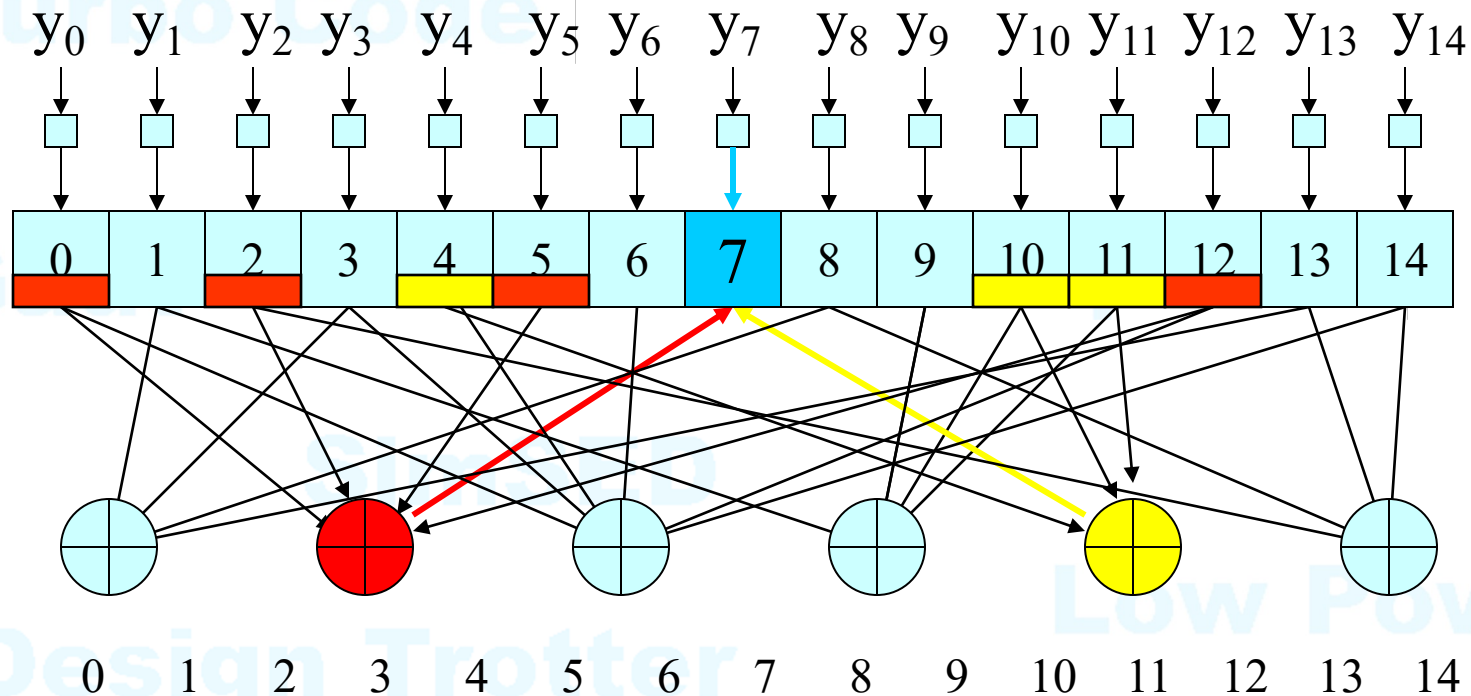
- o Step1: calcul du LLR des bits reçus (information intrinsèque)
- o Step2: Message bit->parité + traitement parité
- o Step3: Message parité->bit + traitement bit
- o Step4: répéter 2 et 3 jusqu'au décodage correct ou "max iteration".

Décodage itératif



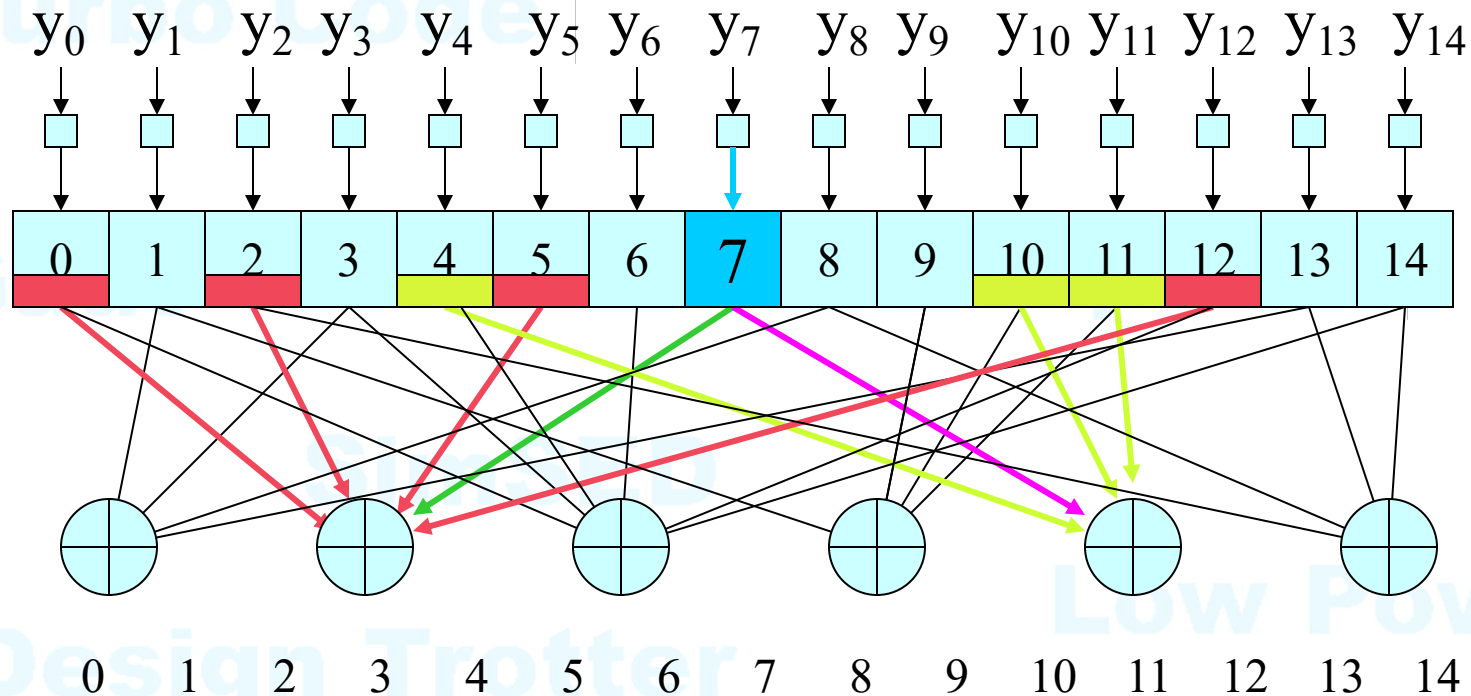
Première itération: message bit- $\rightarrow$ parité

Décodage itératif



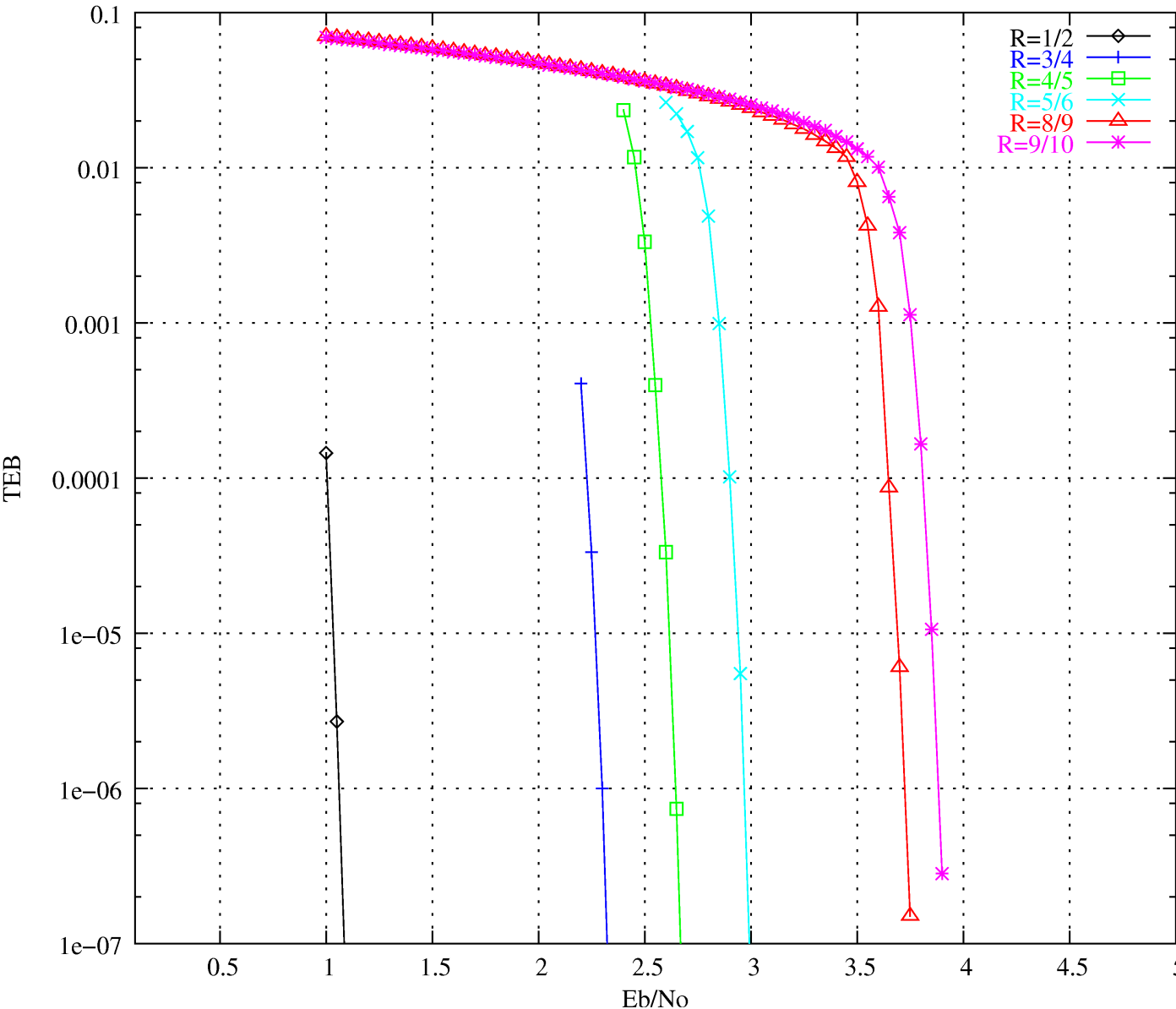
Première itération: message parité $\rightarrow$ bit

Décodage itératif



Deuxième itération: message parité $\rightarrow$ bit

Performances DVB-S2



64k
50 itérations

LDPC + BCH

L.E.S.T.E.R

Laboratoire d'Electronique des Systèmes Temps Réel



Architecture LDPC

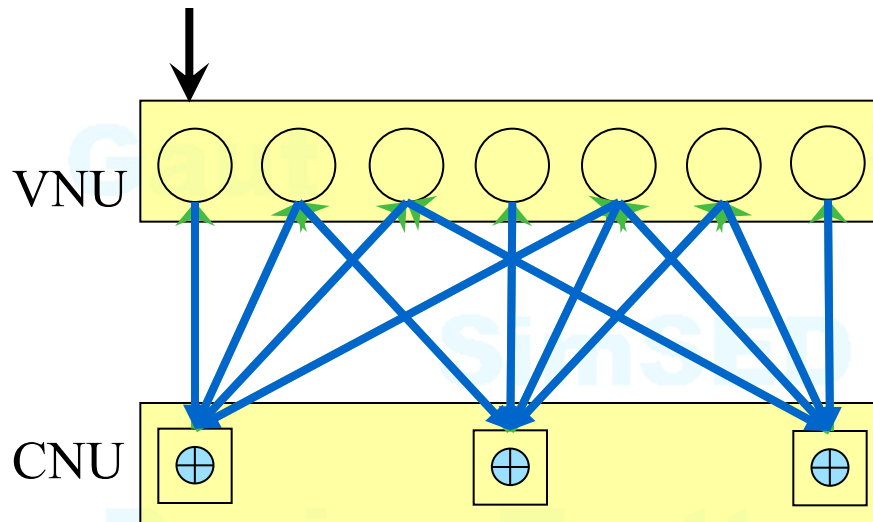
- o Motivation
- o Architecture parameters
- o Example of architecture
- o Conclusion

Motivation

- o Find « generic parameters » to characterize a LDPC architecture.
- o (Try to) classify existing architectures
- o Propose new architectures

Hypothesis 2

- o Regular (j,k) LDPC code
- o Belief Propagation decoding algorithm
- o Check node in the frequency domain



$$E_{m,n} = I_n + \sum_{m' \in M(n) \setminus m} T_{m',n}$$

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

$$\Phi(x) = -\ln \left(\tanh \left(\frac{x}{2} \right) \right)$$

OUTLINE

- o Motivation
- o Architecture parameters
 - Parallelism
 - Scheduling
 - Type of generic node (VNU, GNU)
 - Position of the interconnection network
- o Example of architecture
- o Conclusion

Parallelism of a $(j,k)=(3,6)$ LDPC

P_c : computational power: number of edges computed/cycle

K number of information bit :

K

R rate of the code

$R=(1-j/k)$

E number of edge of the code:

jK/R

F_{clock} Clock frequency:

100 Mbits/s

F_{bit} Bit throughput :

10 Mbit/s

N_{it} average number of iteration :

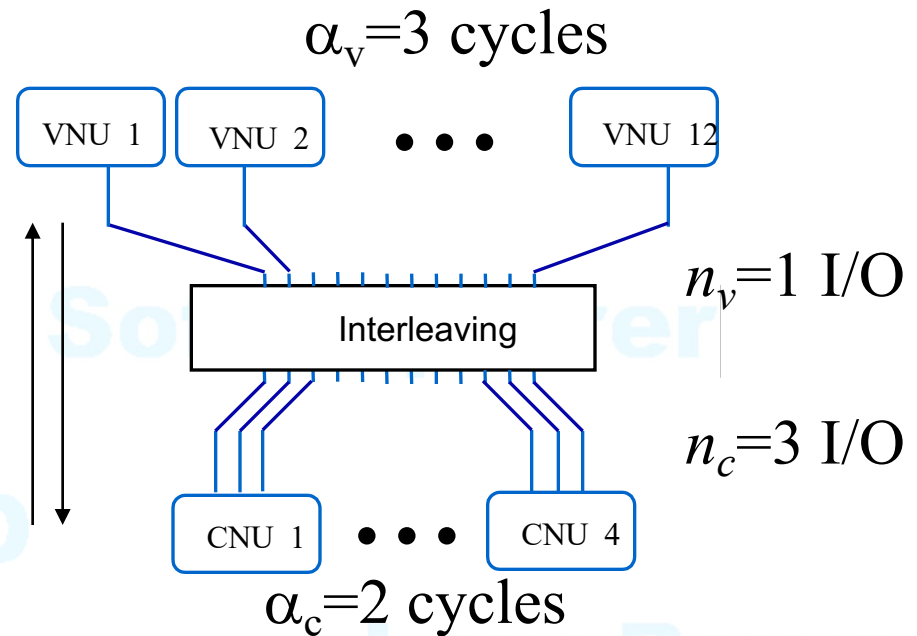
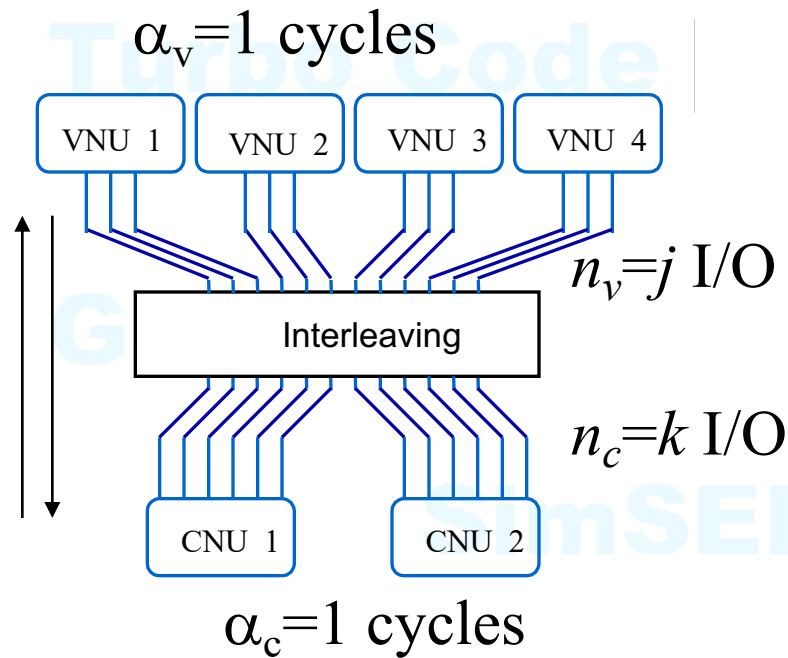
20

$$P_c = EN_{it} [\text{edge/codeword}] \times \left(\frac{F_{bit}}{K} \right) \frac{1}{F_{clock}} [\text{codeword/clock cycle}]$$

$$= \frac{3K}{0.5} \times 20 \times \frac{10}{K} \times \frac{1}{100} = 12 [\text{edge/clock cycle}]$$

Serial/Parallel solution

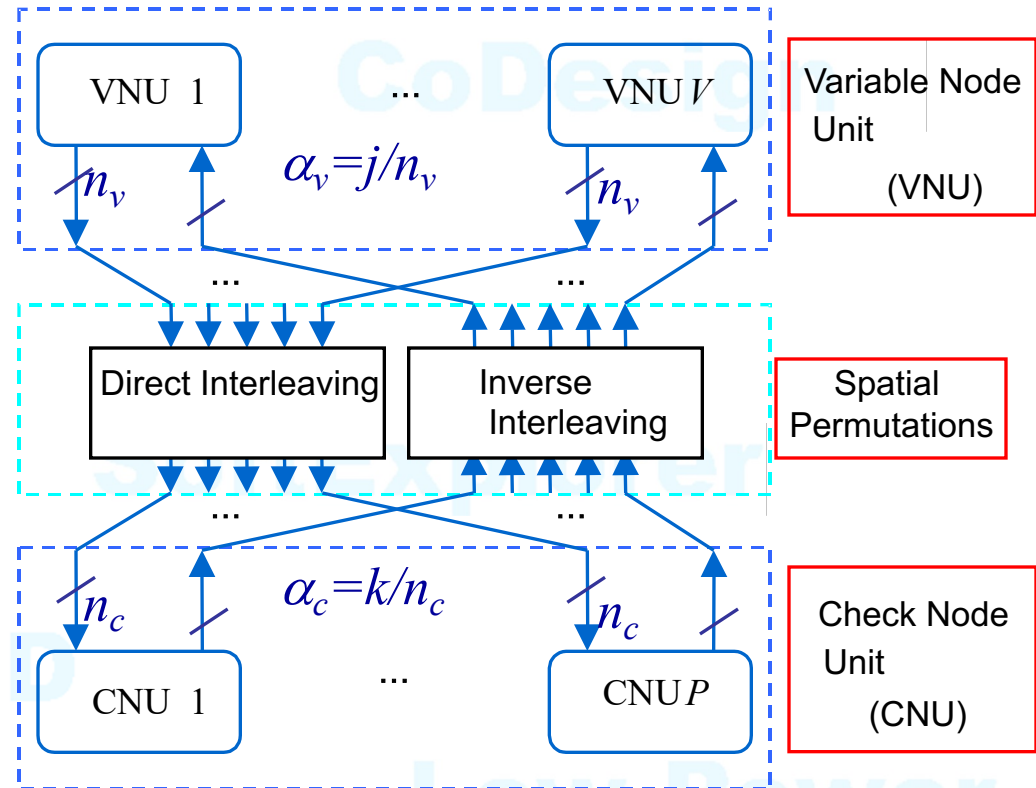
o Several solutions



Generic model

- n_v, n_c : number of input/output ports of the node units

- α_c, α_v : number of clock cycles required to process a CN and a VN

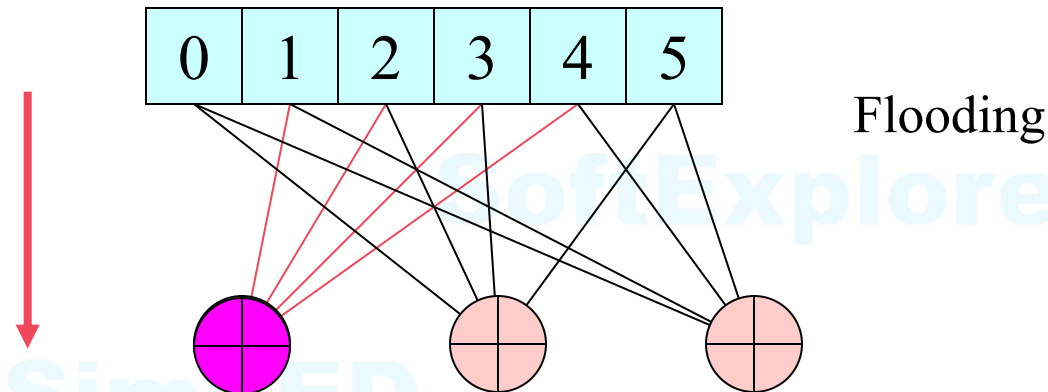


$$P_c = n_c \times P = n_v \times V$$

OUTLINE

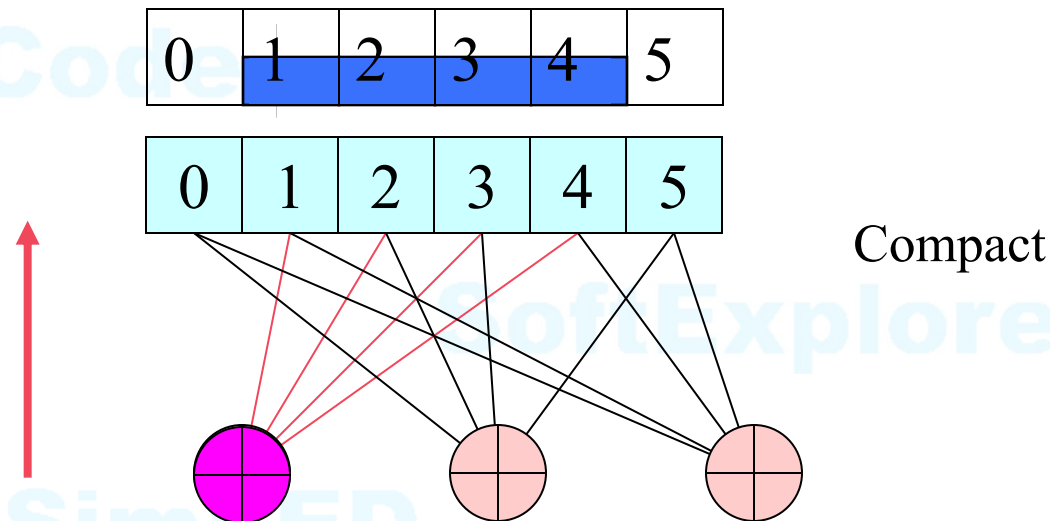
- o Motivation
- o Architecture parameters
 - Parallelism
 - Scheduling
 - Type of generic node (VNU, GNU)
 - Position of the interconnection network
- o Example of architecture
- o Conclusion

Flooding scheduling



Compact scheduling for the check

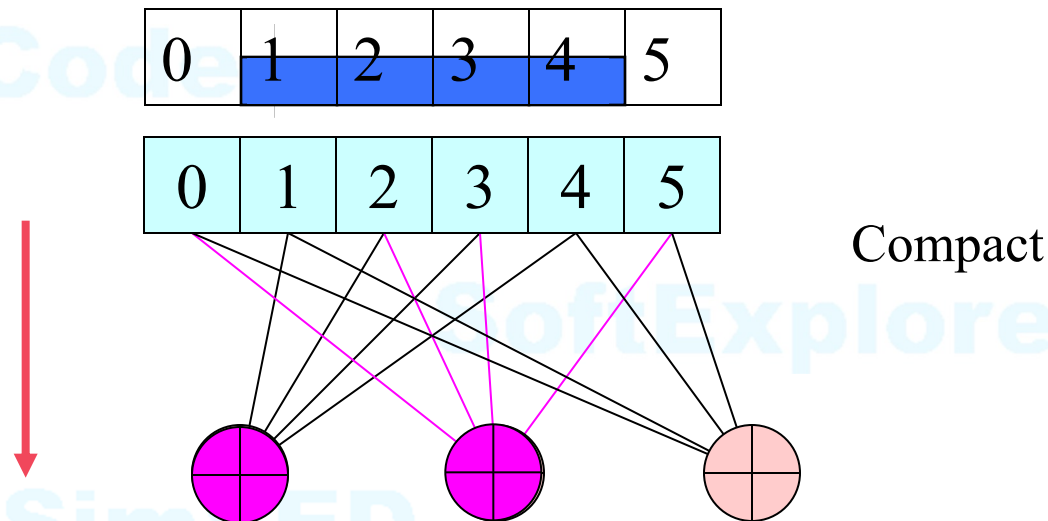
Flooding scheduling



Compact scheduling for the check

Distributed scheduling for the variable, slow update

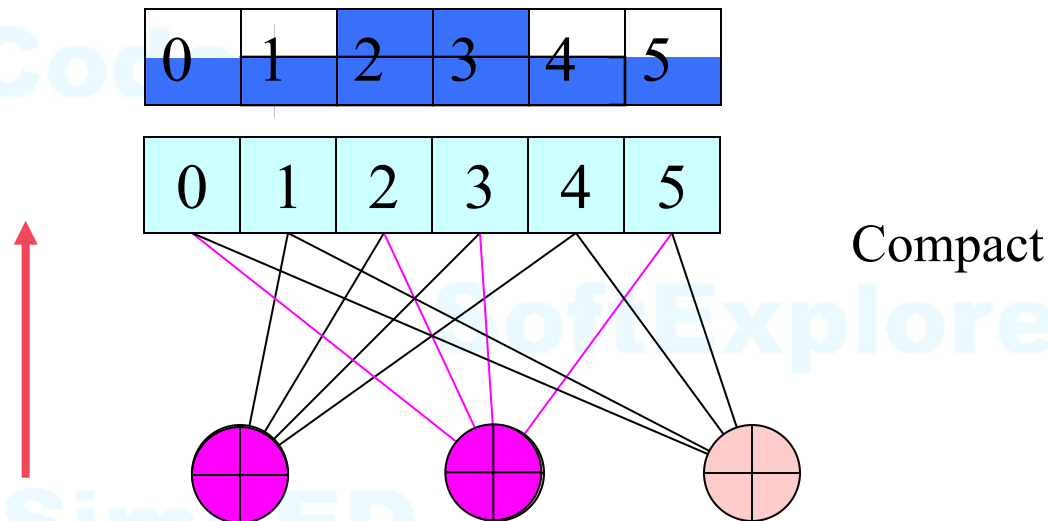
Flooding scheduling



Compact scheduling for the check

Distributed scheduling for the variable, slow update

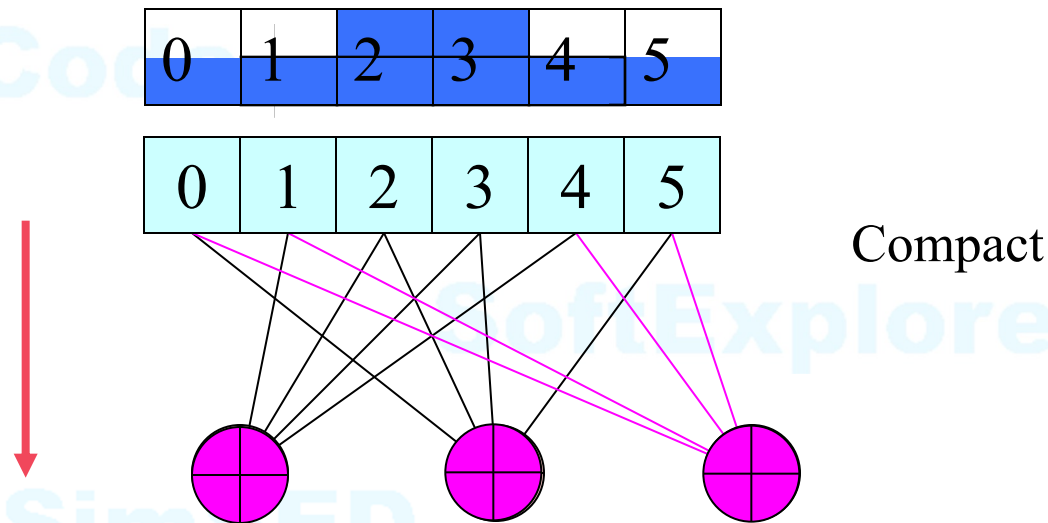
Flooding scheduling



Compact scheduling for the check

Distributed scheduling for the variable, slow update

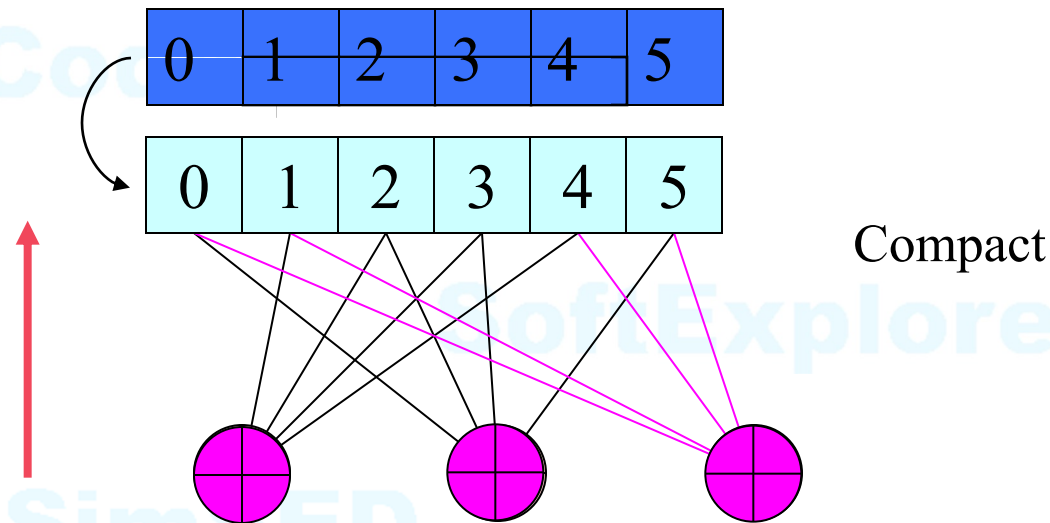
Flooding scheduling



Compact scheduling for the check

Distributed scheduling for the variable, slow update

Flooding scheduling



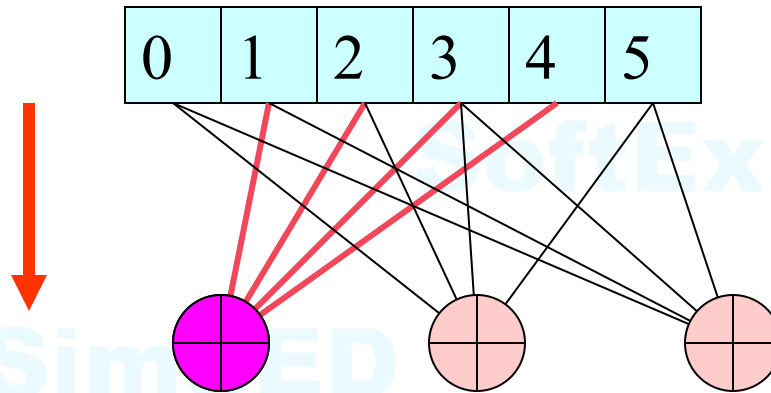
Compact scheduling for the check

Distributed scheduling for the variable, slow update

Horizontal scheduling (turbo-scheduling)

011110
101101
110101

LDPC Matrix

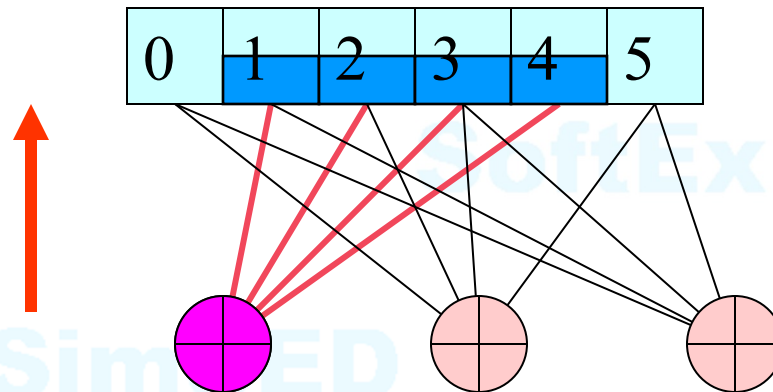


Compact scheduling for the check

Horizontal scheduling (turbo-scheduling)

011110
101101
110101

LDPC Matrix



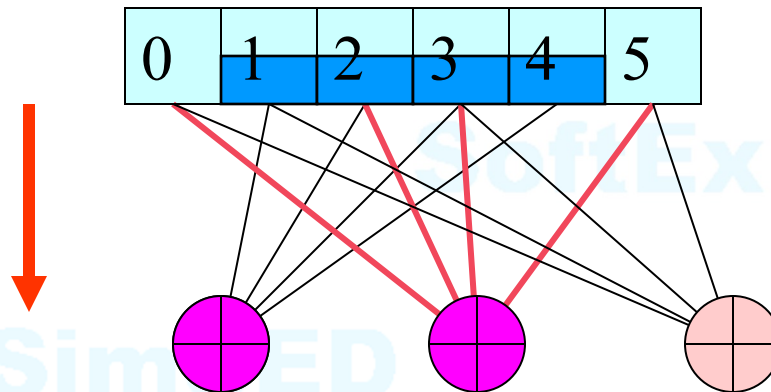
Compact scheduling for the check

Distributed scheduling for the variable, fast update

Horizontal scheduling (turbo-scheduling)

011110
101101
110101

LDPC matrix



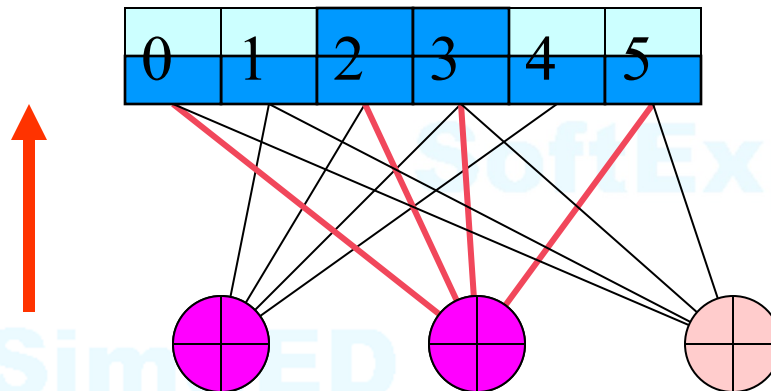
Compact scheduling for the check

Distributed scheduling for the variable, slow update

Horizontal scheduling (turbo-scheduling)

011110
101101
110101

Matrice LDPC



Number of iteration
divided by 1/2

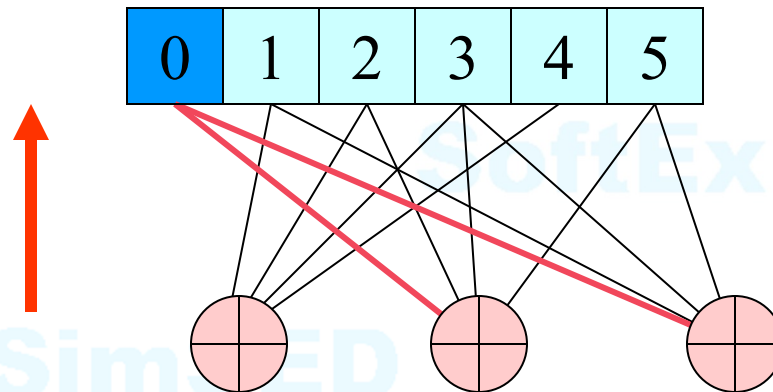
Compact scheduling for the check

Distributed scheduling for the variable, slow update

Vertical scheduling (Fossorier)

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

LDPC matrix



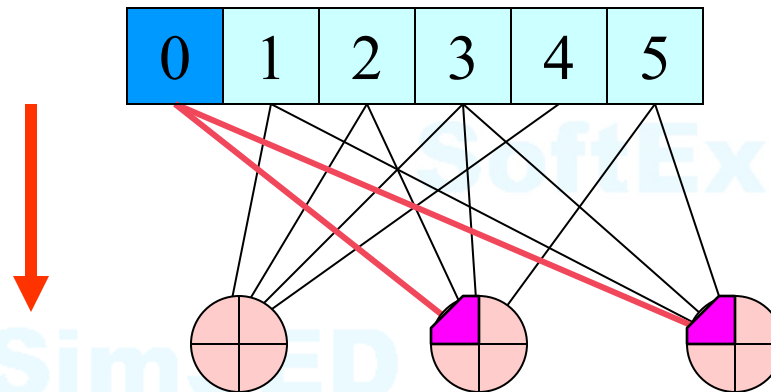
Compact scheduling for the check

Distributed scheduling for the variable, slow update

Vertical scheduling (Fossorier)

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

LDPC Matrix

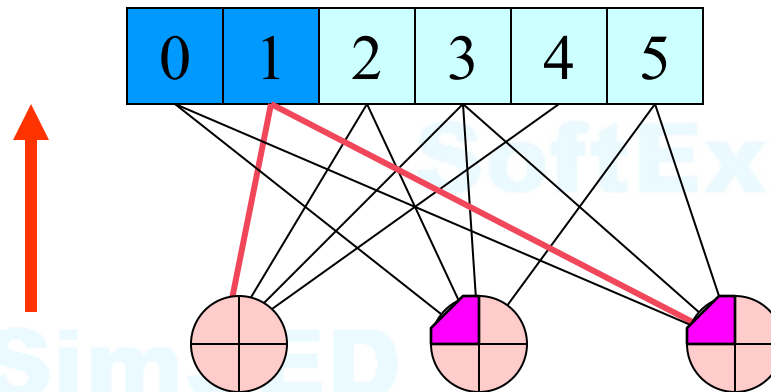


Distributed scheduling for the check, fast update
Compact scheduling for the variable

Vertical scheduling (Fossorier)

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

LDPC Matrix

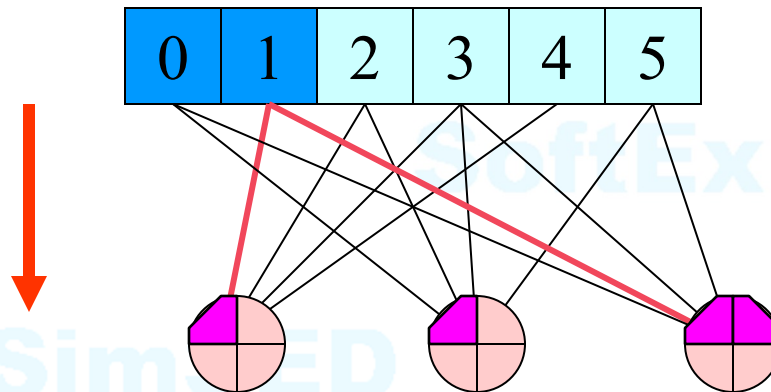


Distributed scheduling for the check, fast update
Compact scheduling for the variable

Vertical scheduling (Fossorier)

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

LDPC matrix



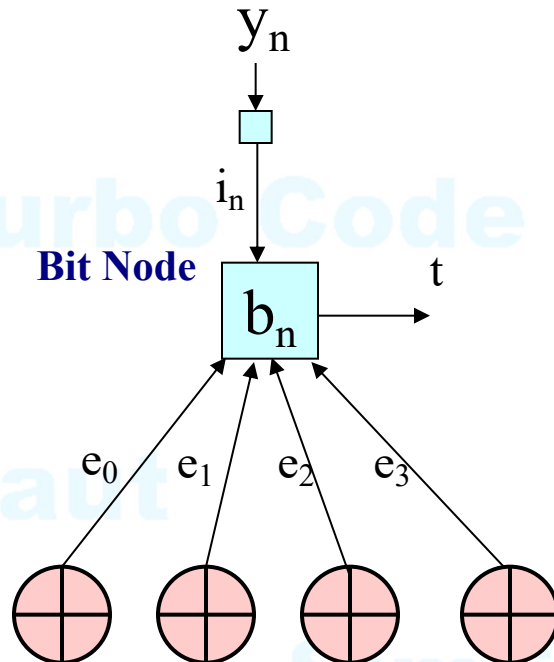
Number of iteration
divided by 1/2

Distributed scheduling for the check, fast update
Compact scheduling for the variable

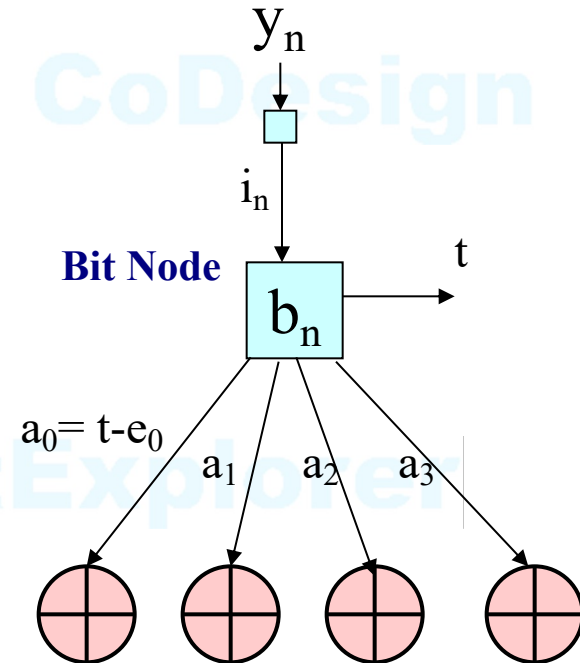
OUTLINE

- o Motivation
- o Architecture parameters
 - Parallelism
 - Scheduling
 - Type of generic node (VNU, GNU)
 - Position of the interconnection network
- o Example of architecture
- o Conclusion

Computation of a variable node

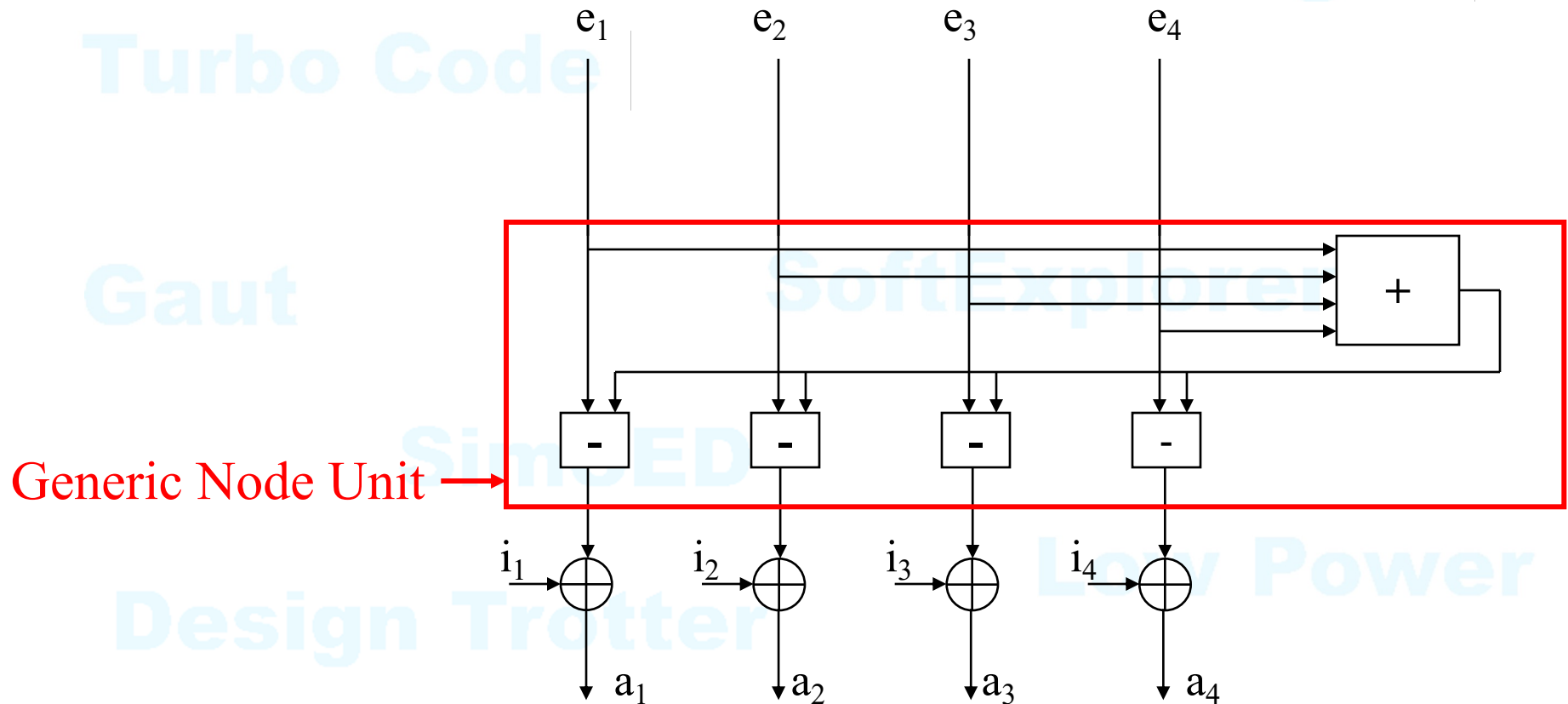


$$t = i_n + \sum_m e_m$$

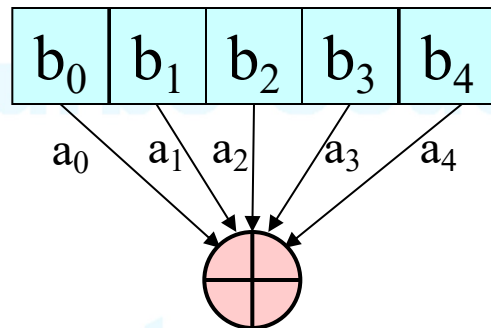


$$a_i = i_n + \sum_{m, m \neq i} e_m = t - a_i$$

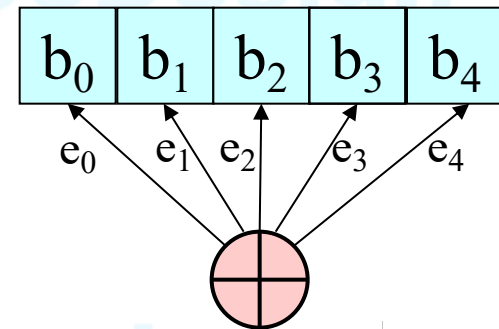
Variable node: data flow graph



Computation of a parity check node



Check Node

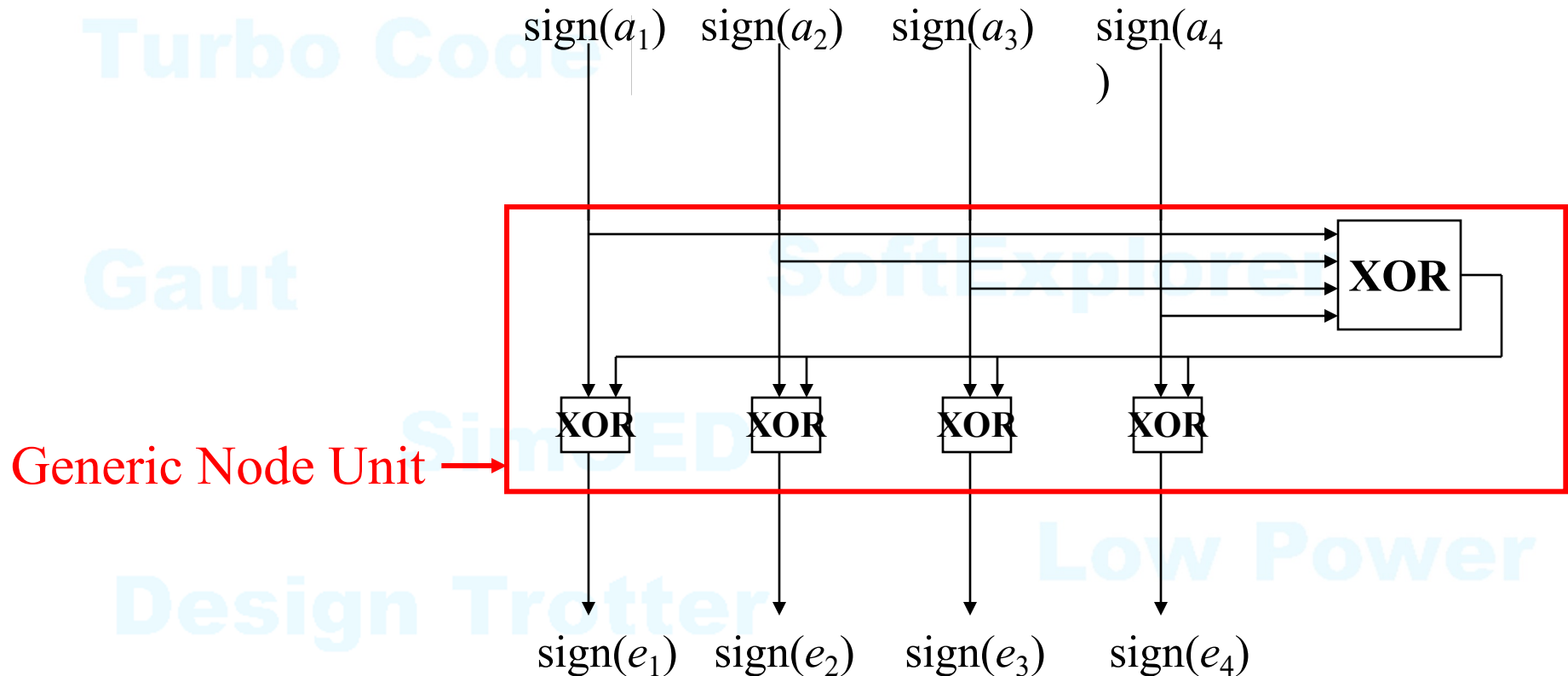


Check Node

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

Two methods $\Rightarrow$ exact: probability or **frequency domain**
 $\Rightarrow$ approximated: min-sum and its variantes

Parity check node in the frequency domain (sign)



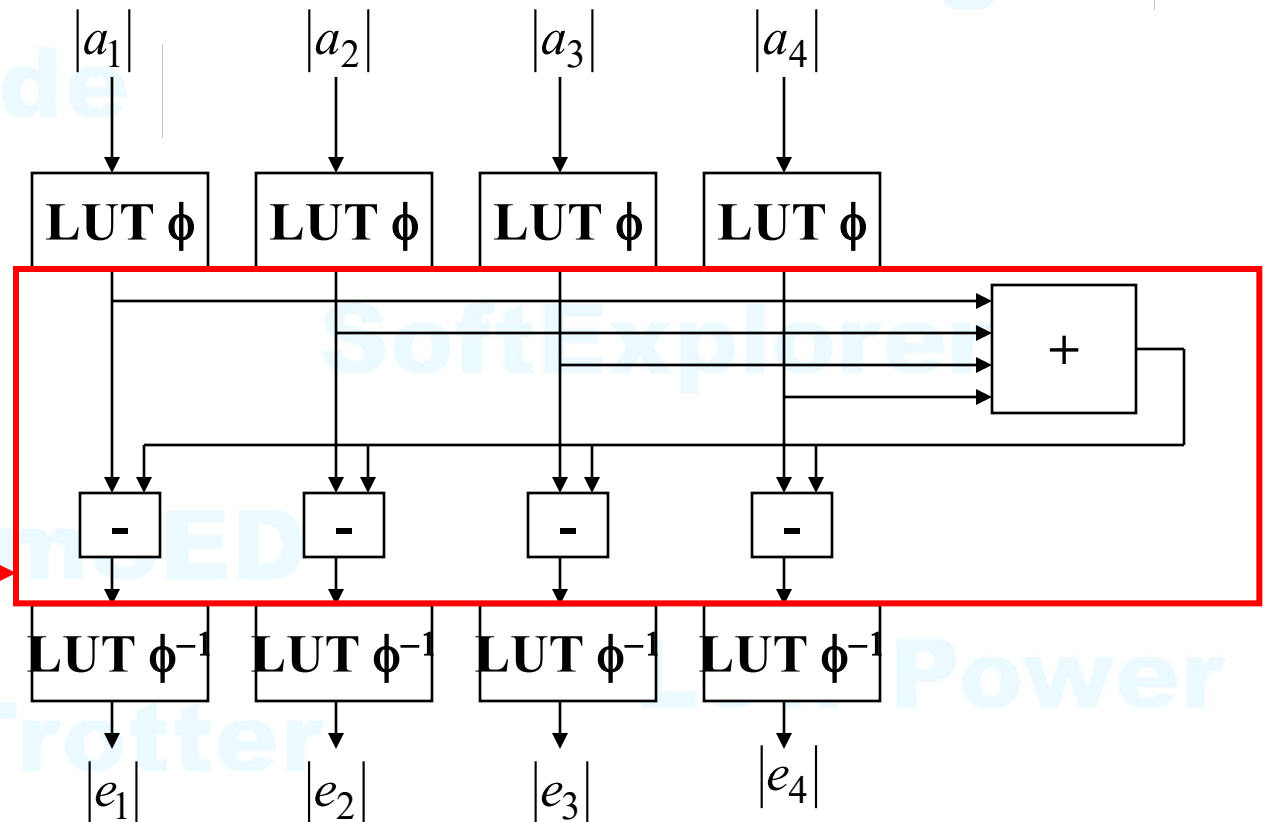
Parity check node in the frequency domain (module)

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

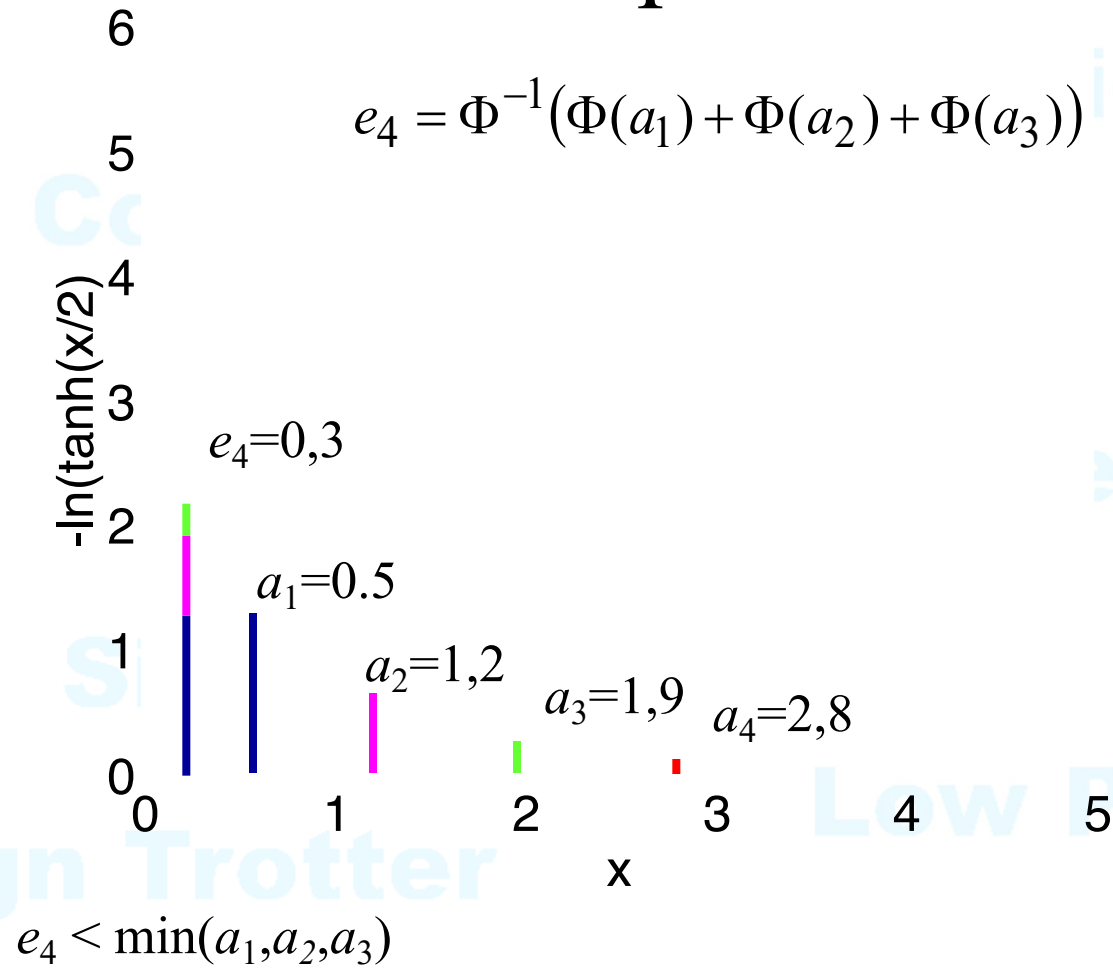
with

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

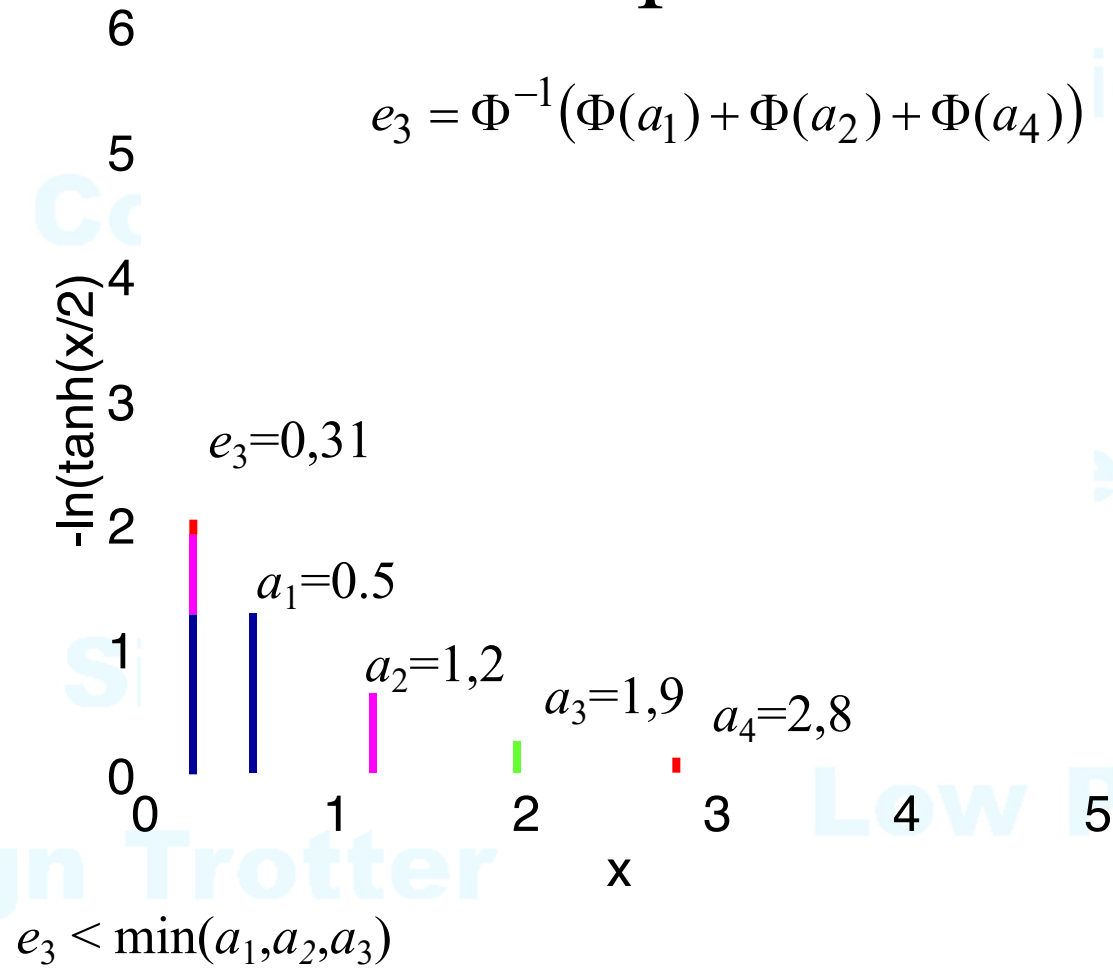
Generic Node Unit →



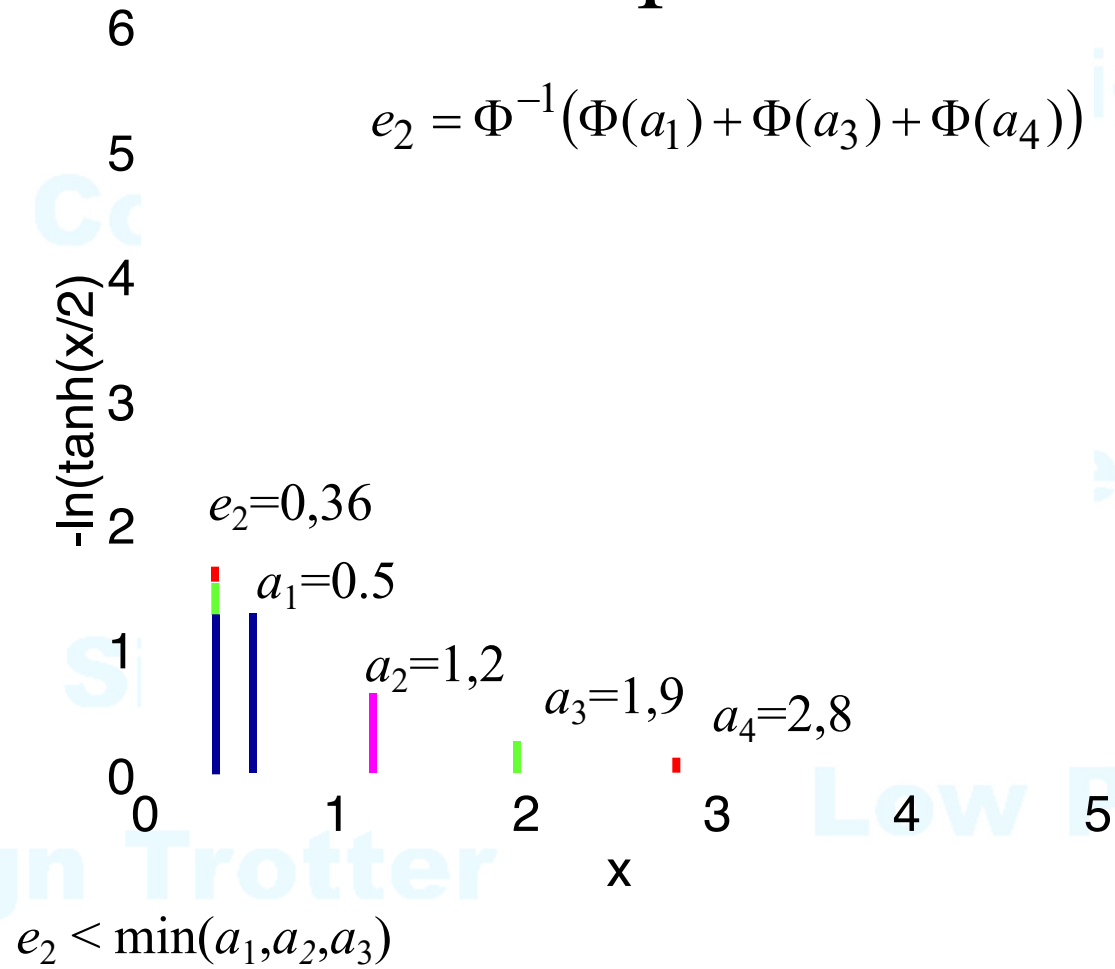
Interprétation graphique calcul extrinsèque



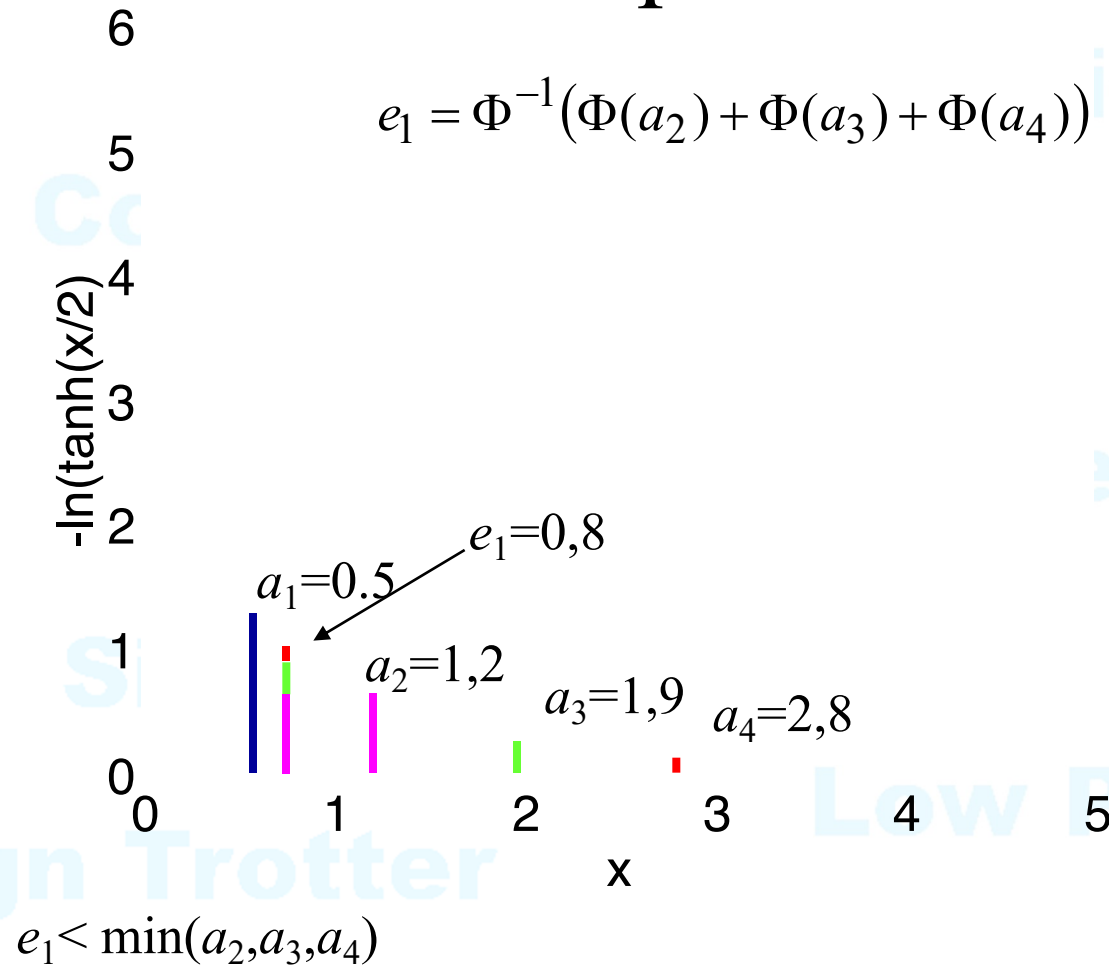
Interprétation graphique calcul extrinsèque



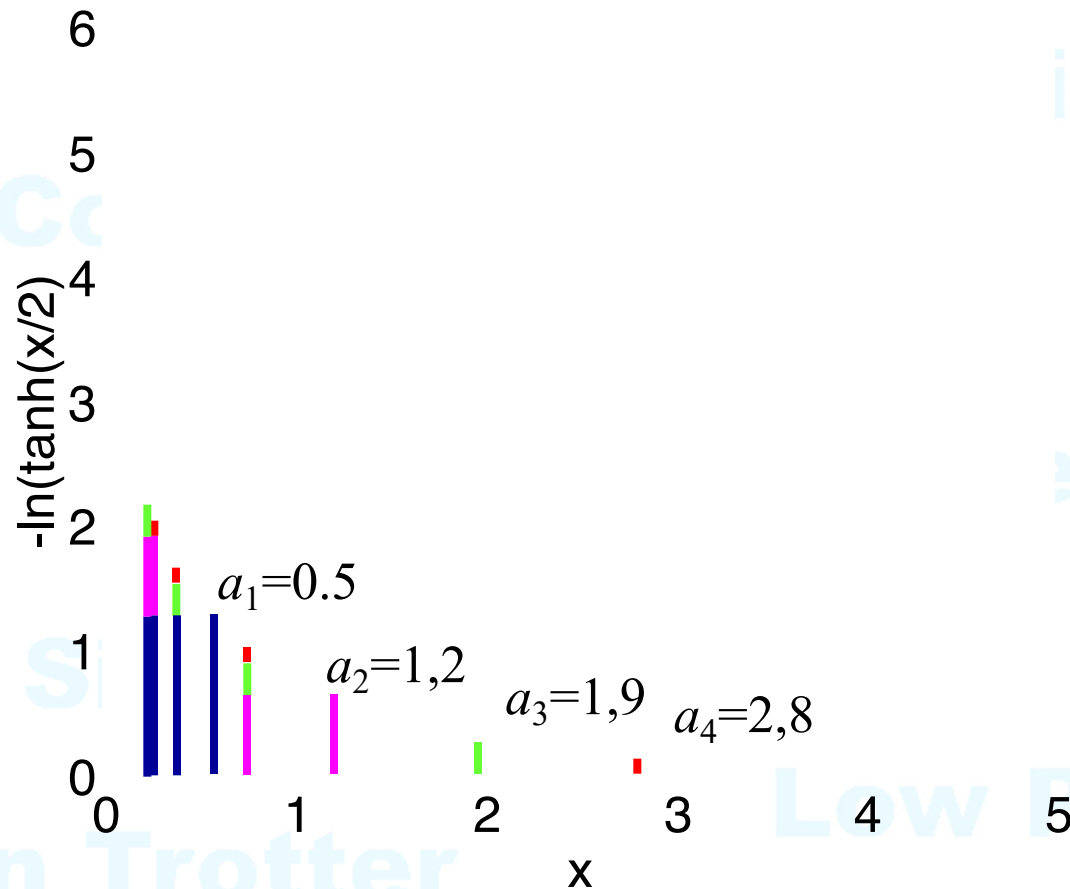
Interprétation graphique calcul extrinsèque



Interprétation graphique calcul extrinsèque



Bilan



Il y a deux (voire trois) groupes d'extrinsèques

Méthodes Sous-optimales

→ Min-sum et ses variantes: $e_k = \alpha \max\left(\min_{j \neq k}(|a_j|) - \beta, 0\right)$

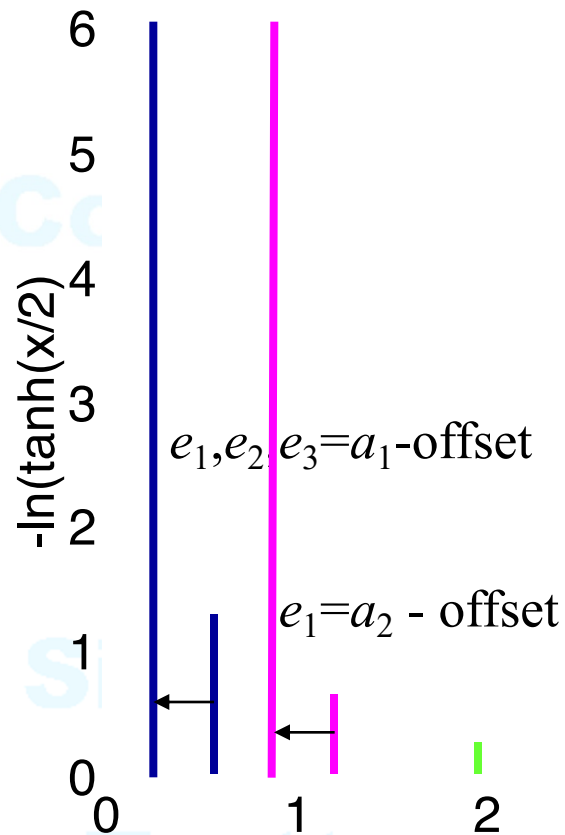
→ λ -min $e_k = \bigoplus_{j \in J(k), j \neq k} |a_j|$

avec $J(k)$ contenant les indices des λ -plus faibles valeurs de $|a_j|$.

→ A-min $e_k = \bigoplus_j |a_j|$ Si $|a_k|$ n'est pas la plus petite valeur

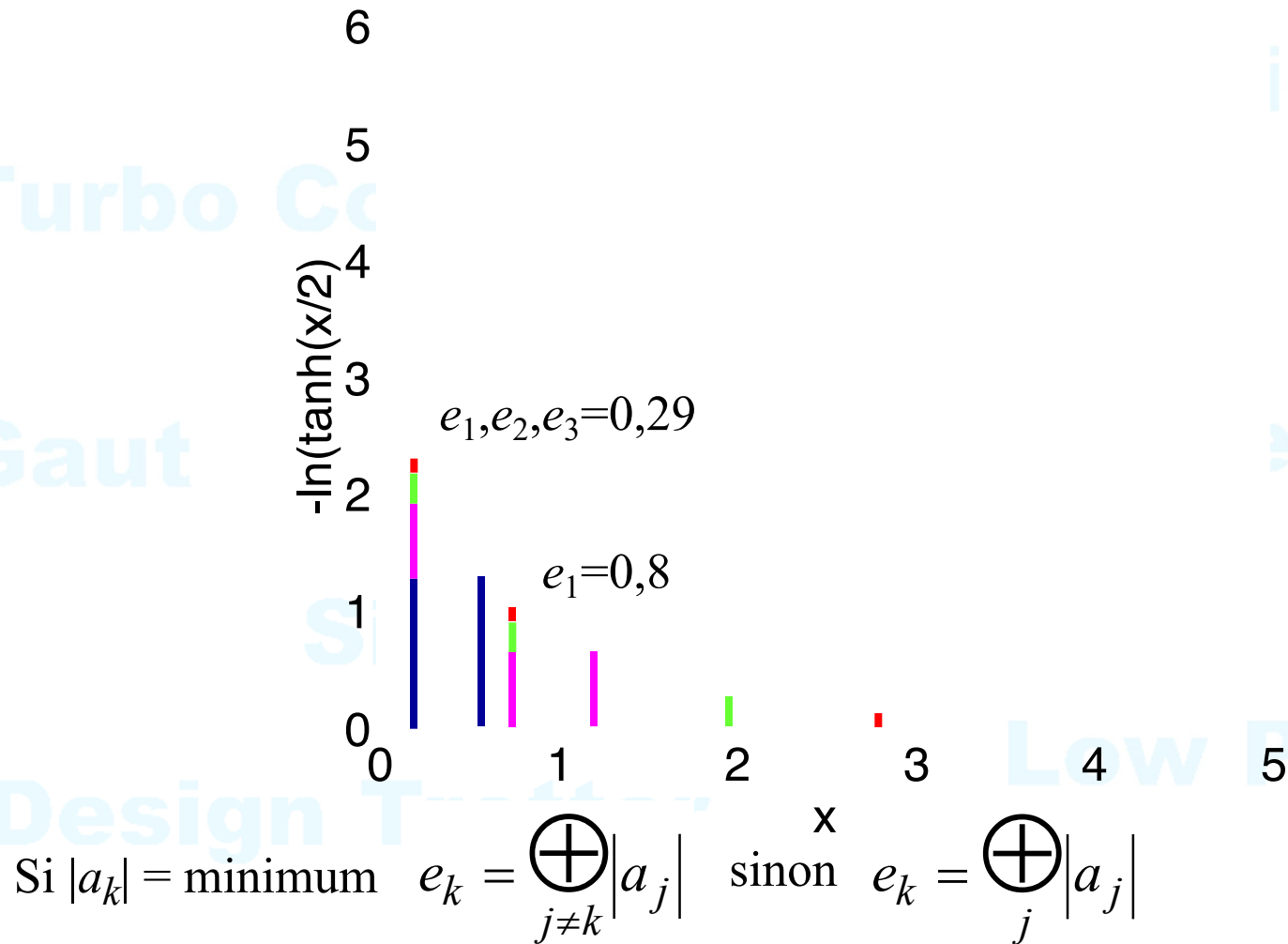
$e_k = \bigoplus_{j \neq k} |a_j|$ Si $|a_k|$ est la plus petite valeur

Méthode min-sum offset

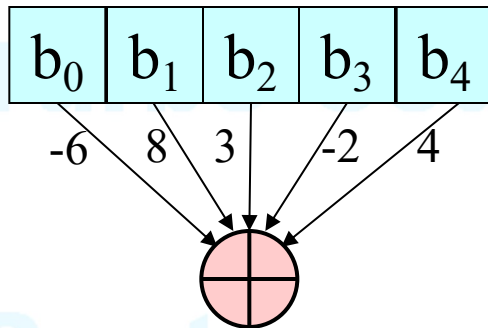


$$e_k = \max\left(\min_{j \neq k}(|a_j|) - \beta, 0\right)$$

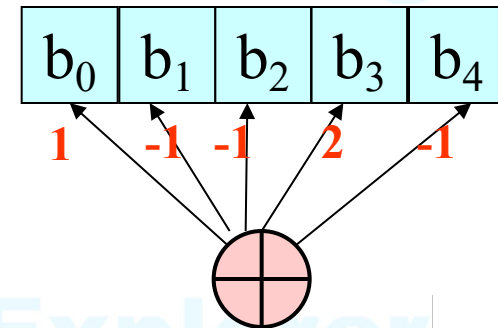
Méthode A-min



Exercice : min-offset ($\beta=1$)



Processeur de parité



Processeur de parité

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

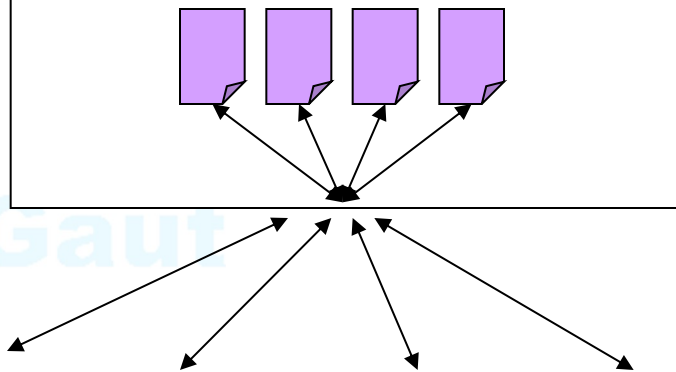
=> solution : on soustrait une constante pour compenser la sur-évaluation de l'extrinsèque : $\min(e_k - \beta, 0)$

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

Generic Node Units (exclude the Φ function)

Generic Node unit

Memories + processing



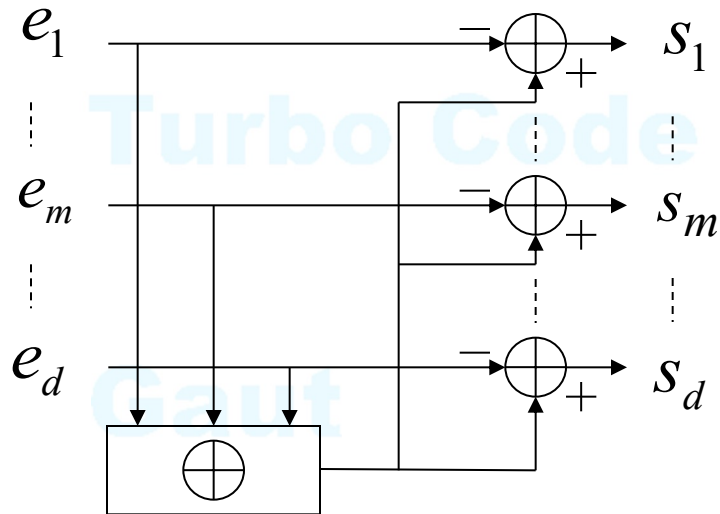
◦ d inputs e_m and d outputs s_n
($d=j$ or k)

◦ Input / output law:

$$s_n = \sum_{m \neq n} e_m$$

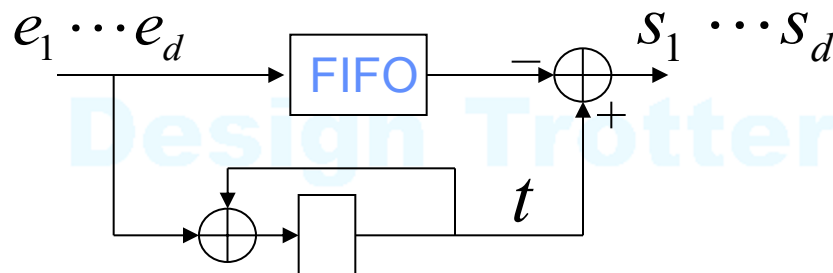
Generic Node^{b)}

a) Compact mode («no memories »)



Compact mode and parallel implementation

$$(\alpha = 1 \rightarrow n = d)$$

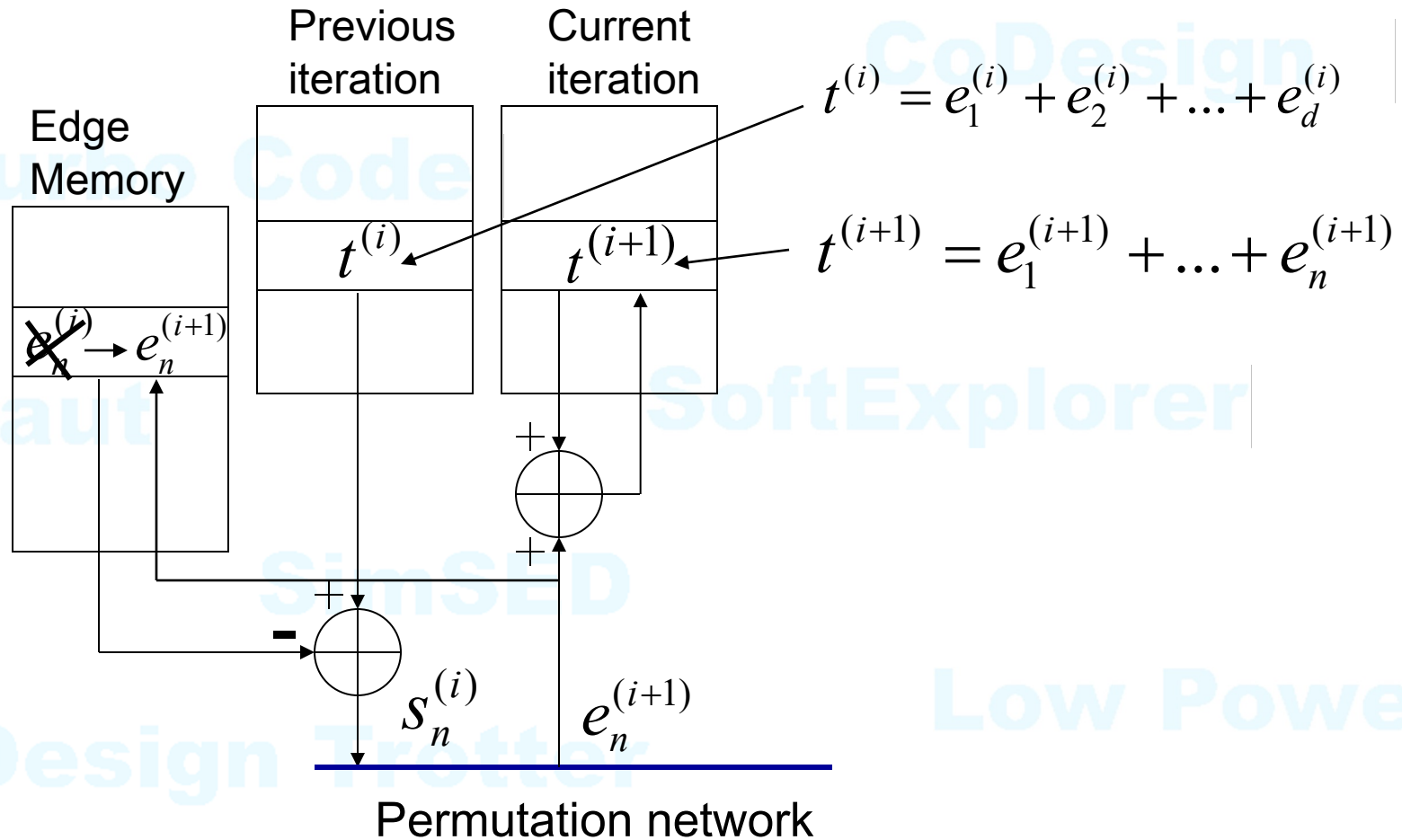


Compact mode and serial implementation

$$(\alpha = d \rightarrow n = 1)$$

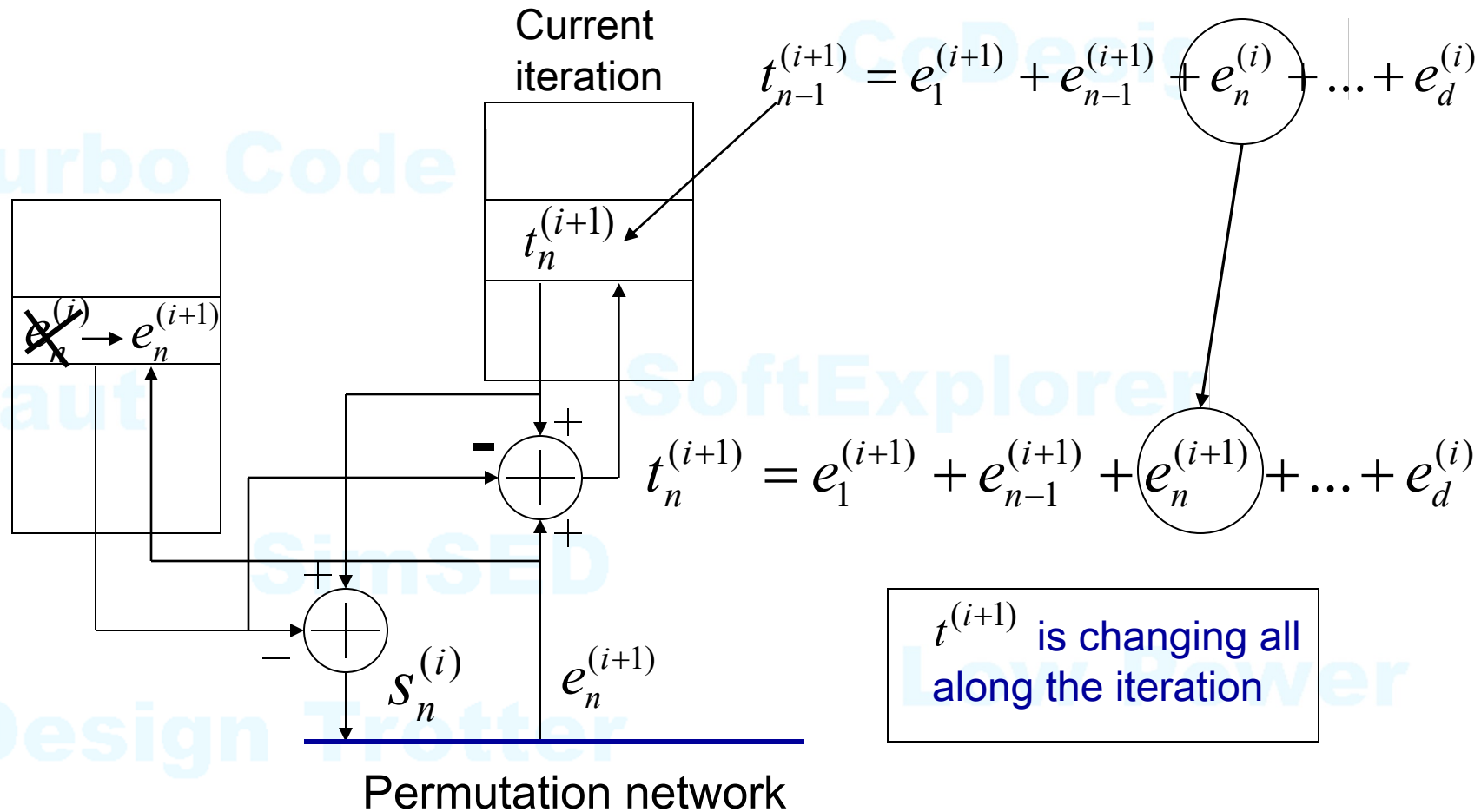
Generic Node

b) Distributed Mode, slow update (Serial)



Generic Node:

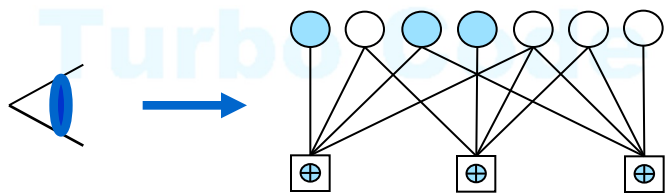
c) Distributed Mode, fast update



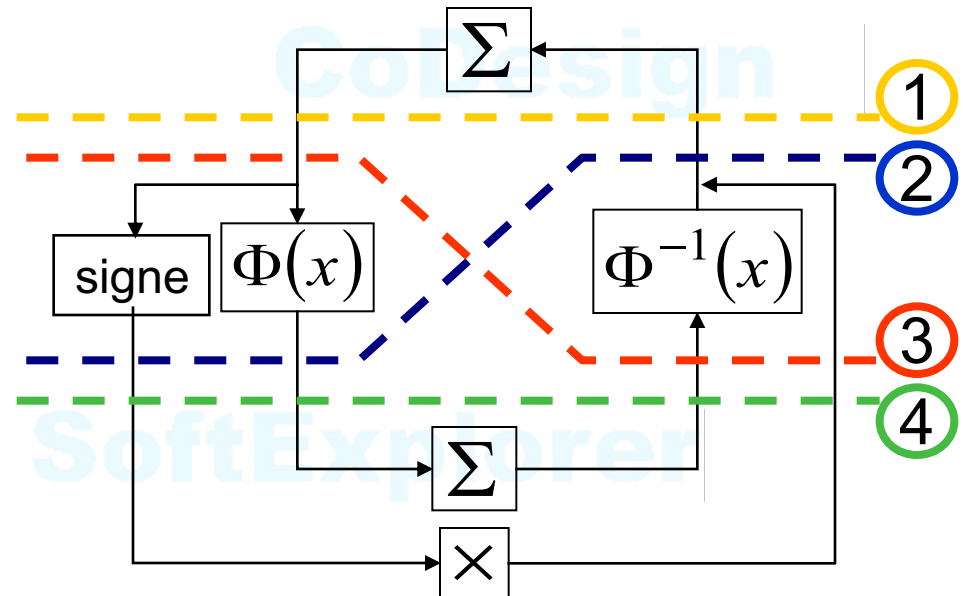
OUTLINE

- o Motivation
- o Architecture parameters
 - Parallelism
 - Scheduling
 - Type of generic node (VNU, GNU)
 - Position of the interconnection network
- o Example of architecture
- o Conclusion

Position of the interconnexion network



or min sum...



Network position		1	2	3	4
Generic Node 	VNU	Σ	$\Phi \circ \Sigma$	$\Sigma \circ \Phi^{-1}$	$\Phi \circ \Sigma \circ \Phi^{-1}$
	CNU	$\Phi^{-1} \circ \Sigma \circ \Phi$	$\Phi^{-1} \circ \Sigma$	$\Sigma \circ \Phi$	Σ
		$\times$ (sign product)			

Result : Generic description of the decoder architecture

- oNode Unit Architecture

- oParallelism : $P, V, \alpha_{CNU}, \alpha_{VNU}$

- oProcesseur control (compact / distributed)

- oShuffle network position
(1,2,3,4)

Many combinations
of the parameters are
possible:

Control mode combinations

			VNU		
			Compact	Distributed	
				Slow Update	Fast Update
CNU	Compact		Parallel Flooding	Flooding along CNU	Horizontal Shuffle
	Distributed	Slow Update	Flooding along VNU	Edge by edge control	
		Fast Update	Vertical Shuffle		

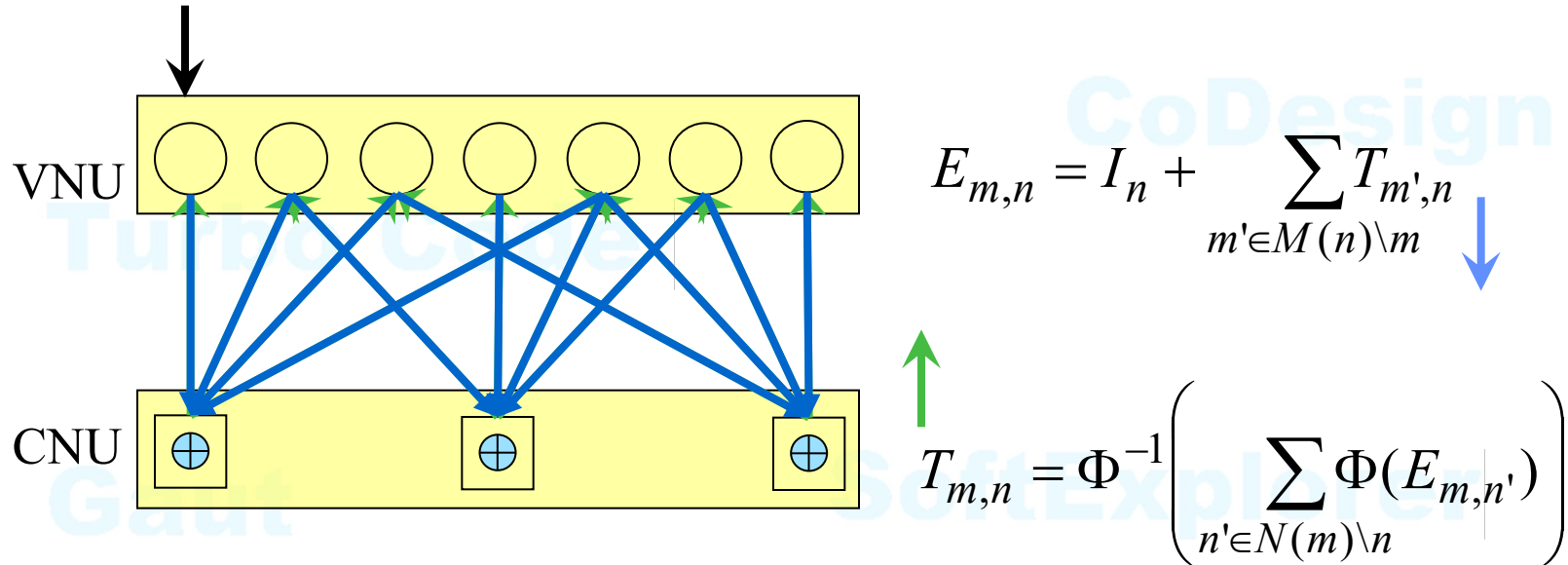
OUTLINE

- o Motivation
- o Architecture parameters
- o Example of architecture
- o Conclusion

Vertical Shuffle ad-hoc architecture (1)

- o Originally proposed as Shuffle-BP (Zhang & Fossorier, 2002)
- o Combination of:
 - Compact Mode for VNU
 - Distributed Mode with fast update for CNU

Vertical Shuffle ad-hoc architecture (1)

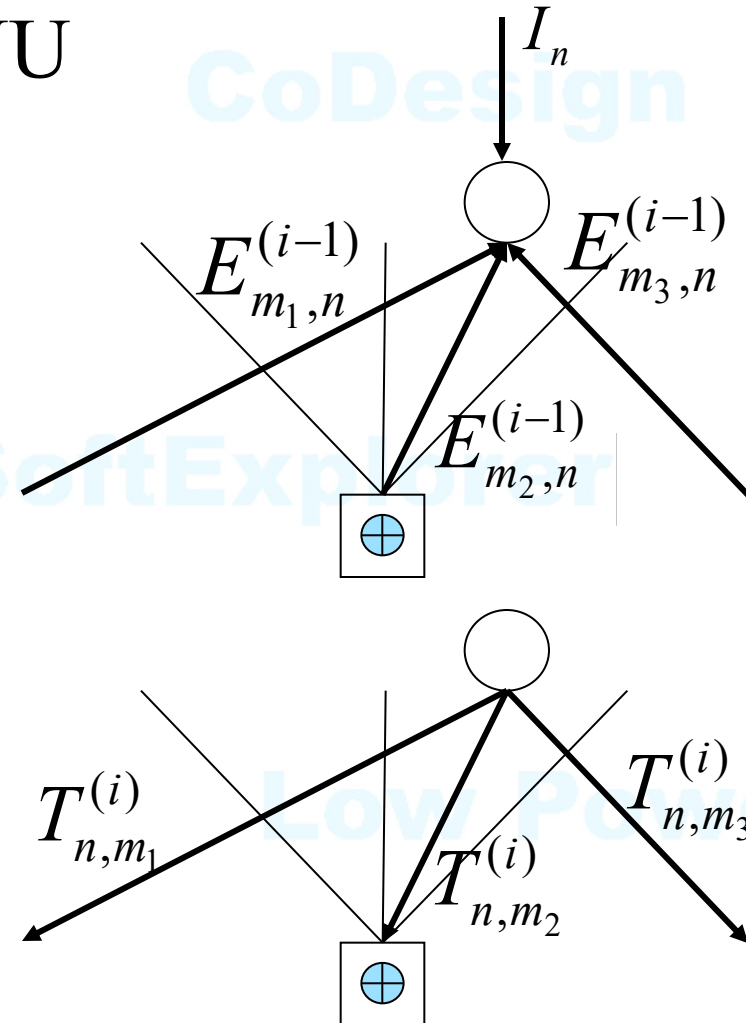
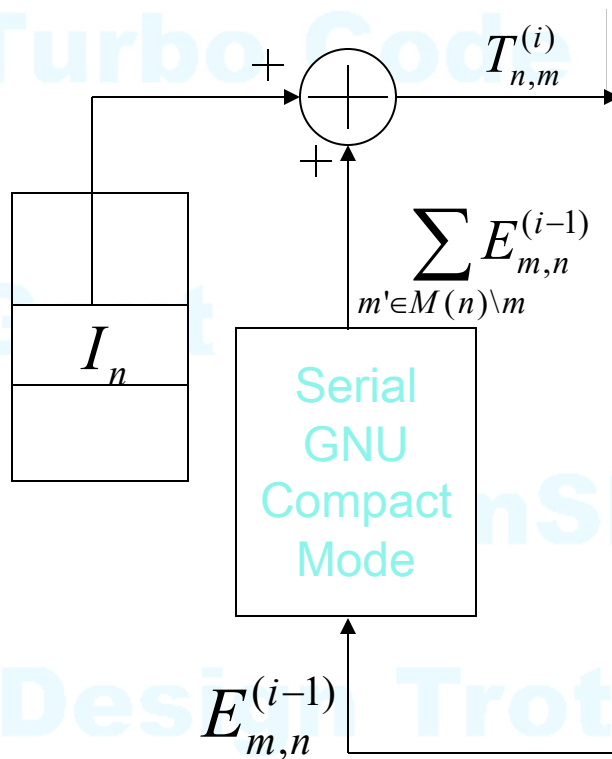


$$\Phi(x) = -\ln\left(\tanh\left(\frac{x}{2}\right)\right)$$

$$Q_{m,n} = \Phi(E_{m,n'}) \quad R_m = \sum_{n \in N(m)} \Phi(E_{m,n}) \quad T_{m,n} = \Phi^{-1}(R_n - Q_{m,n})$$

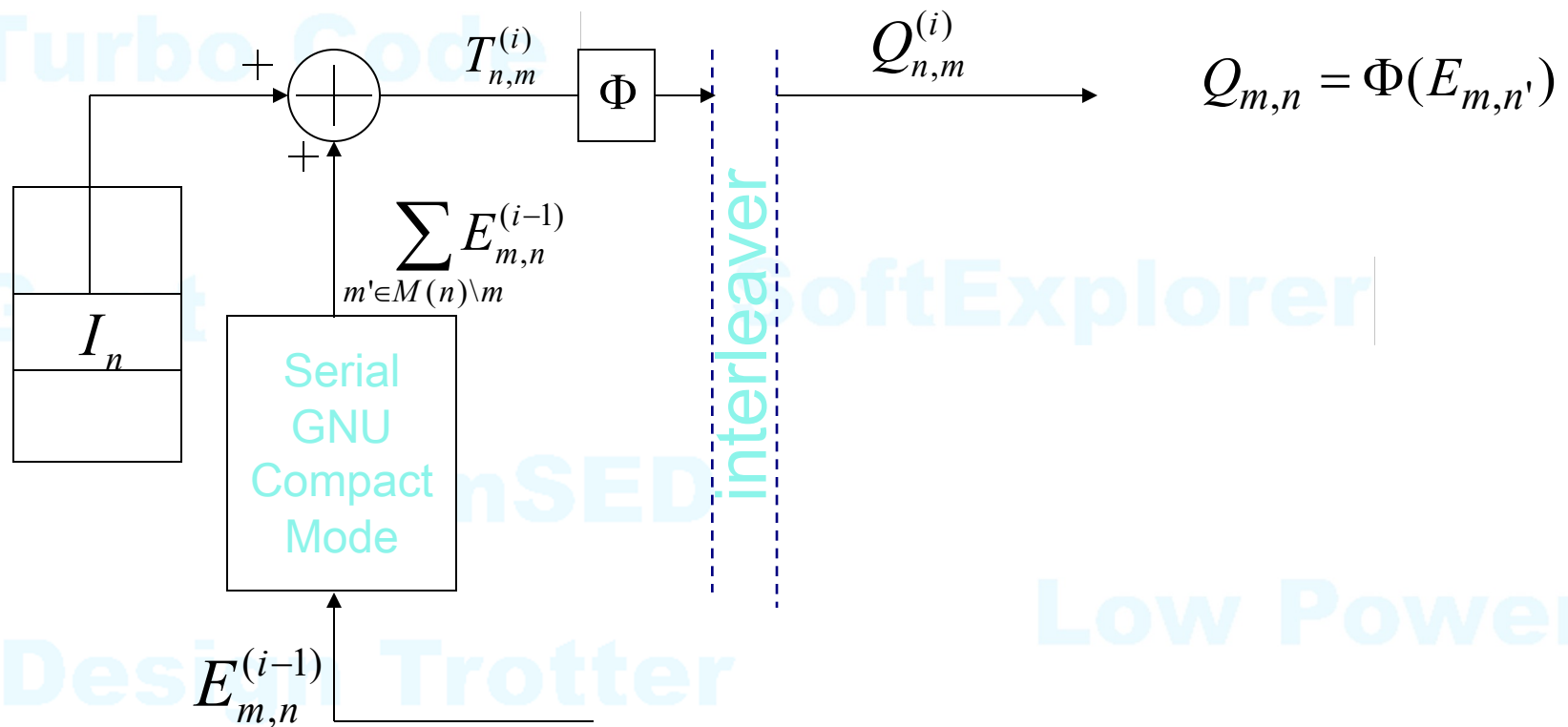
Vertical Shuffle ad-hoc architecture (2)

o Compact, Serial VNU



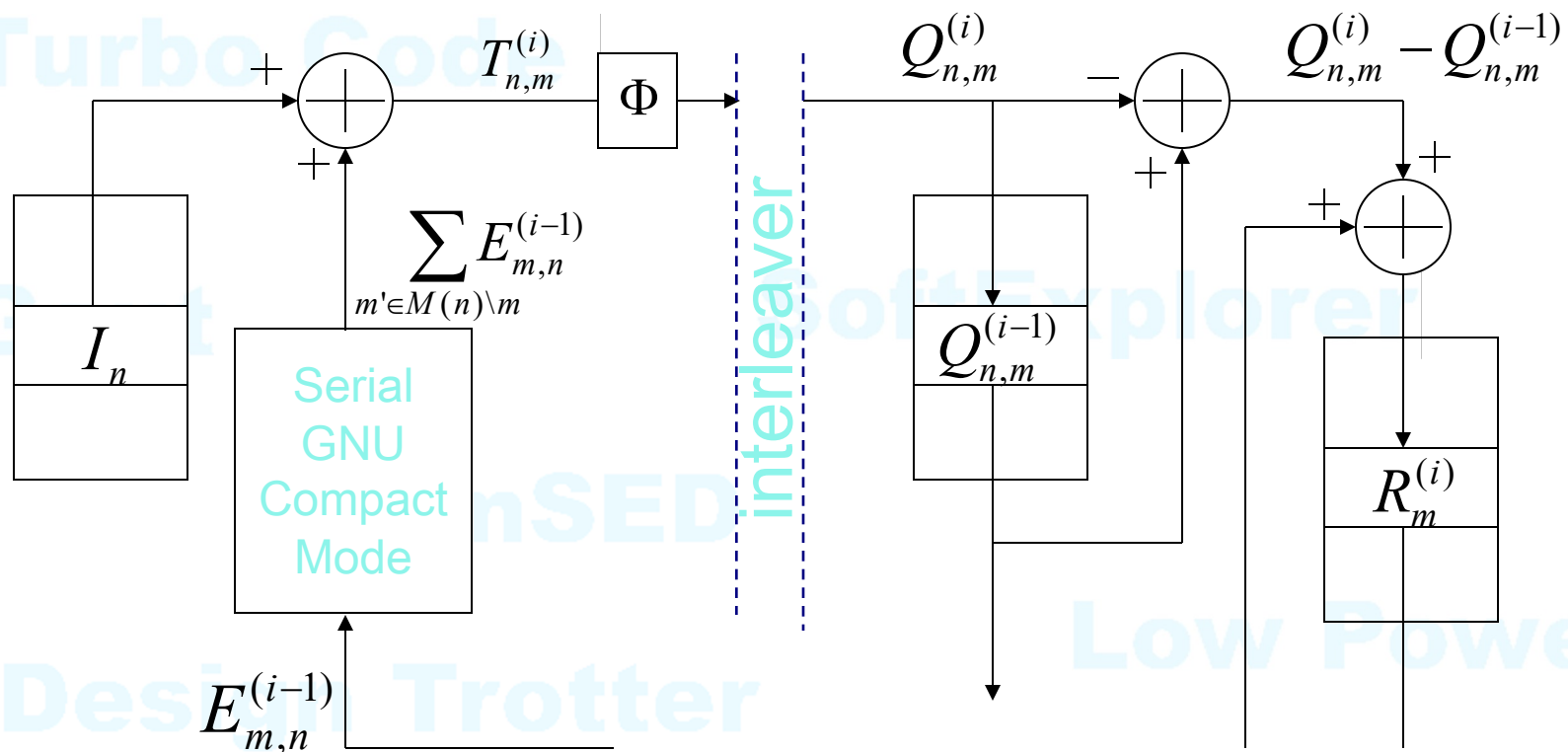
Vertical Shuffle ad-hoc architecture (3)

- Go through the ϕ functions and interleaver



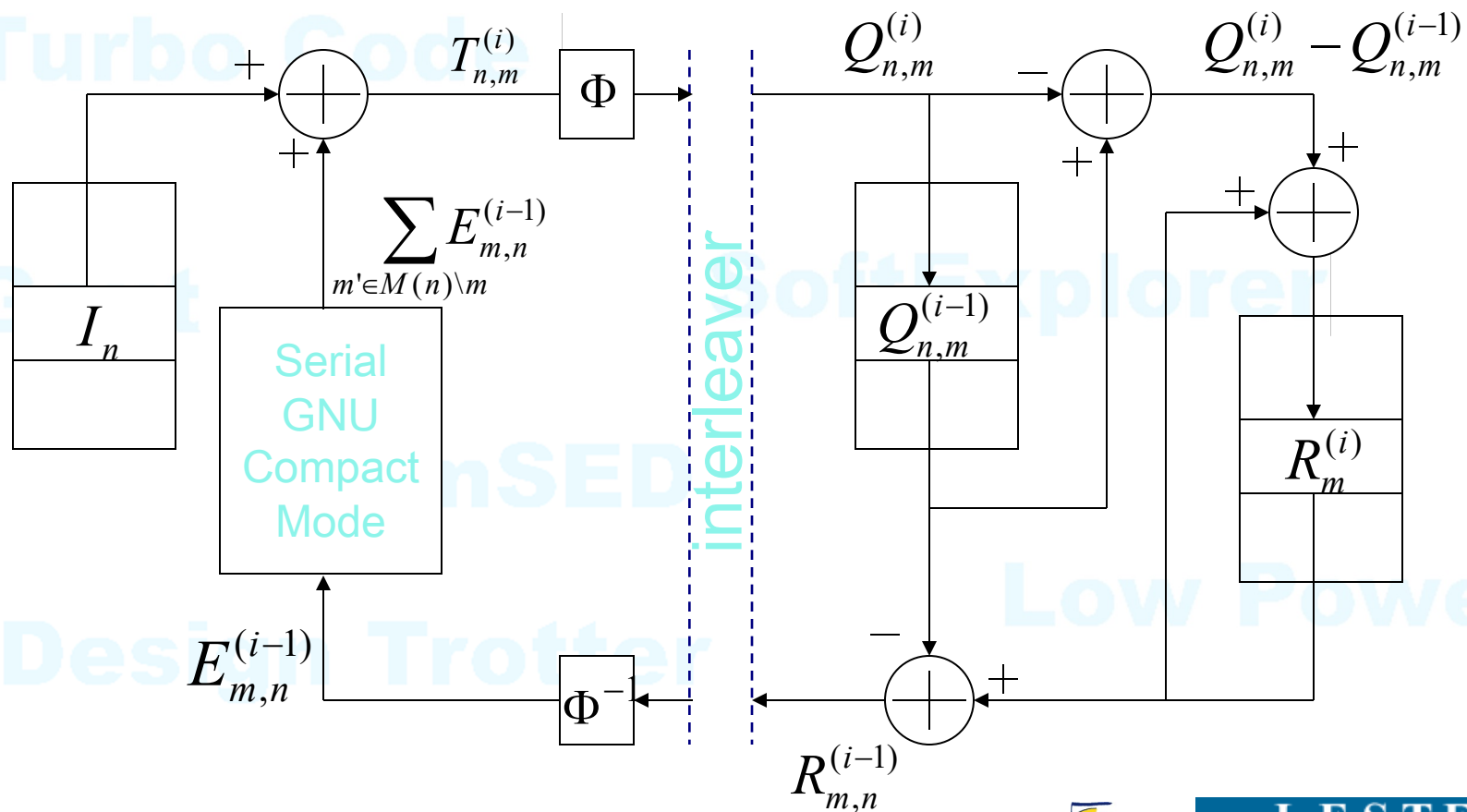
Vertical Shuffle ad-hoc architecture(4)

- Save $Q_{n,m}$ and update R_m (fast update)



Vertical Shuffle ad-hoc architecture (5)

$$R_m = \sum_{n \in N(m)} \Phi(E_{m,n})$$



State of the art : memory size

E : Edge number
 N : codeword size
 M : number of check
 F() : compression

E : Edge number N : codeword size M : number of check 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		

Scheduling vs Architecture

			VNU		
			Compact	Distributed	
				Slow Update	Fast Update
CNC	Compact		Blanksby et al. (parallel)	Boutillon'00, Chen, Zhang,	Mansour
	Distributed	Slow Update	New	-	
		Fast Update	New (Fossorier's Shuffle BP)		

OUTLINE

- o Motivation
- o Architecture parameters
- o Example of architecture
- o Conclusion

Conclusion

Proposition of partial framework for description, analysis and synthesis of LDPC decoder architectures.

Based on the formalism, proposition of an architecture for vertical scheduling

Extend methodology to all LDPC architecture
(to be done)

PLAN

❶ Code convolutif et son décodage

❷ Turbo-code

❸ Code LDPC

❹ Décodeurs haut débit

❹ Gestion du parallélisme

❹ Quantification du signal d'entrée

❹ Critère d'arrêt et bufferisation

❺ Conclusion

Augmenter le débit d'un décodeur

- o Soit N_{op} le nombre d'opérations/bit d'information pour une itération de décodage.

Parallélisme

$$\text{Débit} = \frac{N}{N_{op}} \times F_{clock} \times n_{iter}$$

Critère d'arrêt +
buffer

Précision
Pipe-line
Optimisation fine

PLAN

❶ Code convolutif et son décodage

❷ Turbo-code

❸ Code LDPC

❹ Décodeurs haut débit

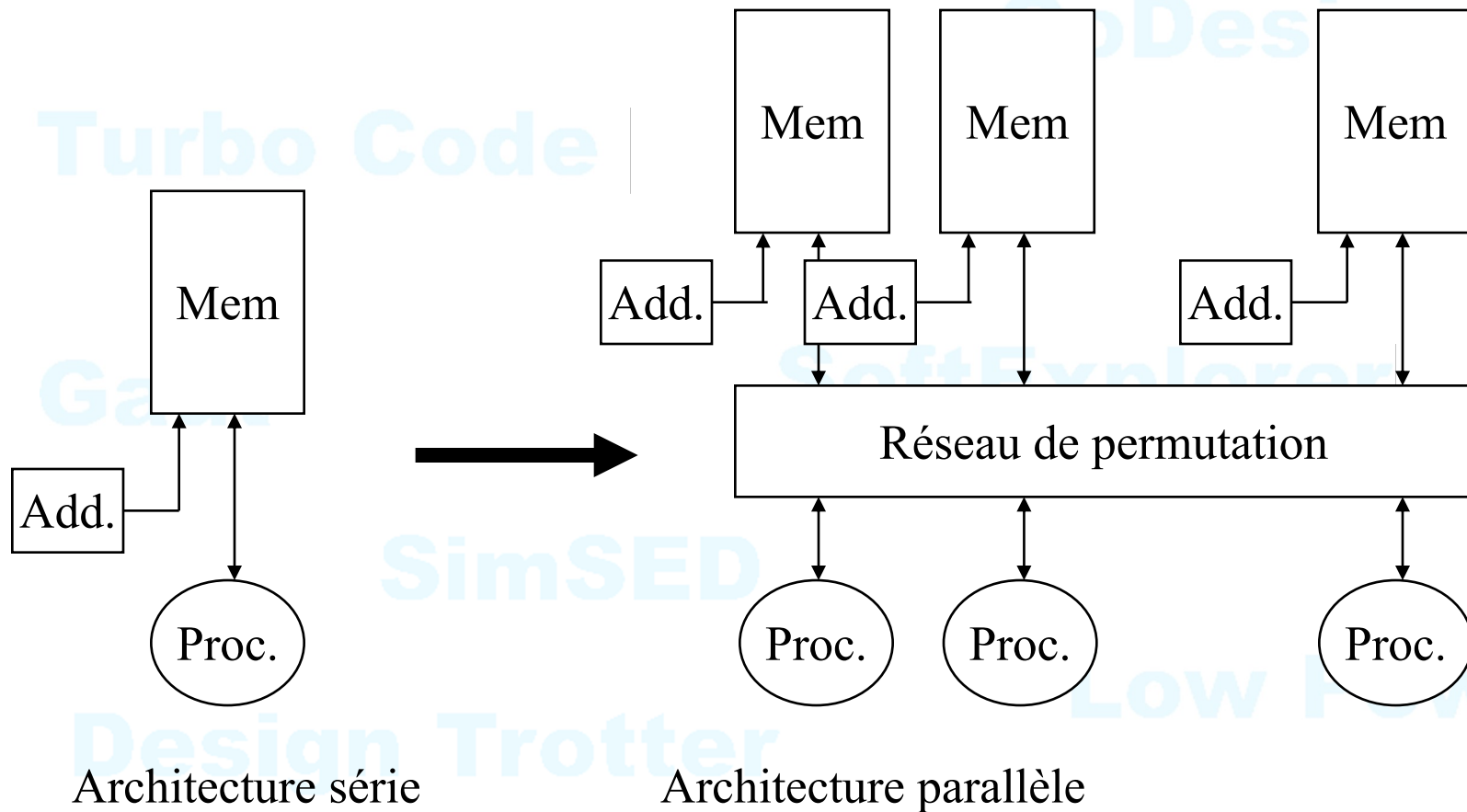
❹ Gestion du parallélisme

❹ Quantification du signal d'entrée

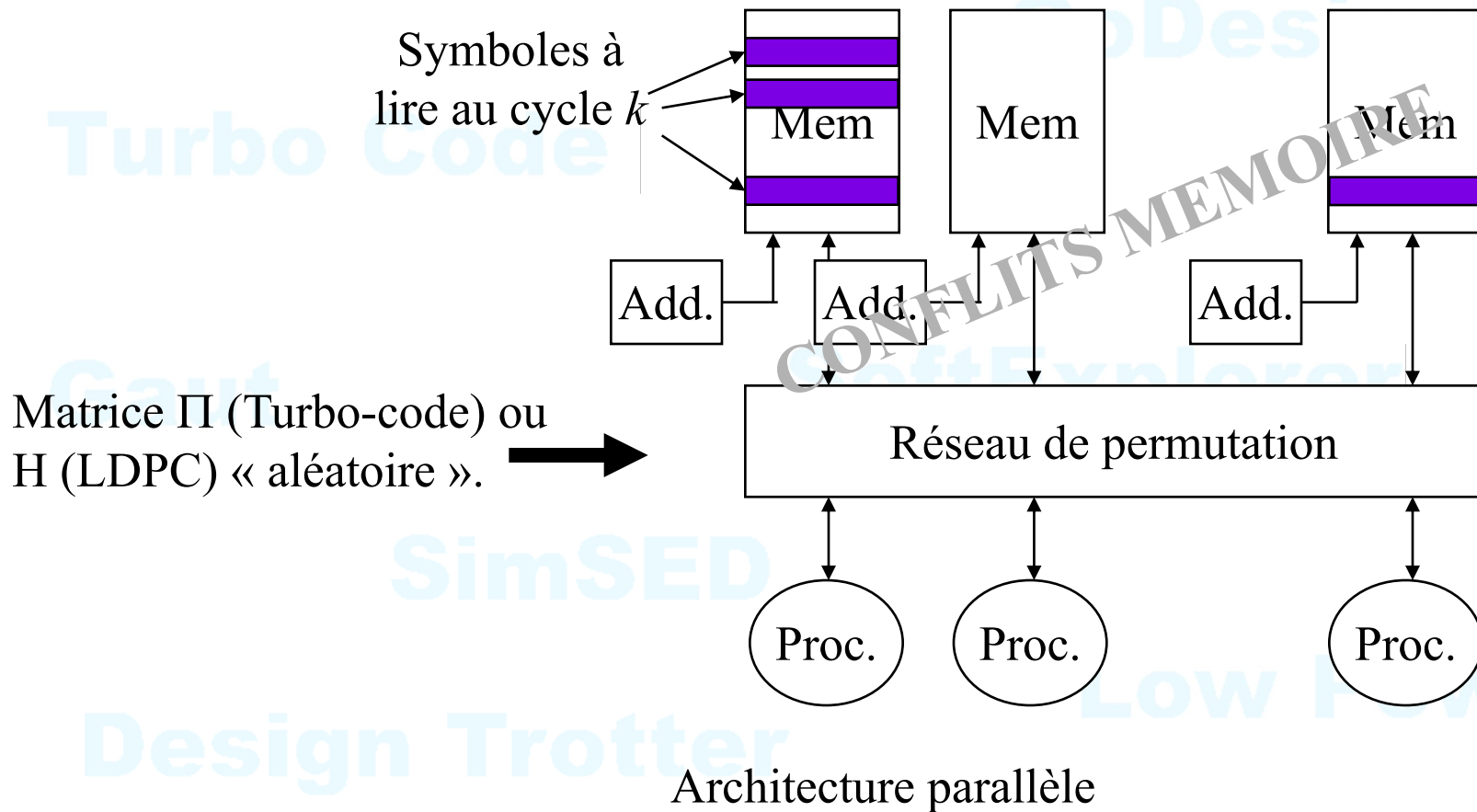
❹ Critère d'arrêt et bufferisation

❹ Conclusion

Augmenter le parallélisme



Problème : conflit mémoire...



Solution à l'exécution

Approche Network on Chip (NOC) : le réseau contient des FIFO en entrée permettant d'anticiper ou retarder les lectures et les écritures.

Quand un conflit est inévitable, les décodeurs sont « gelés » le nombre de cycles nécessaires.

Les messages (adresse + symbole) sont transférés dans le réseau de permutation par des mécanismes dynamiques.

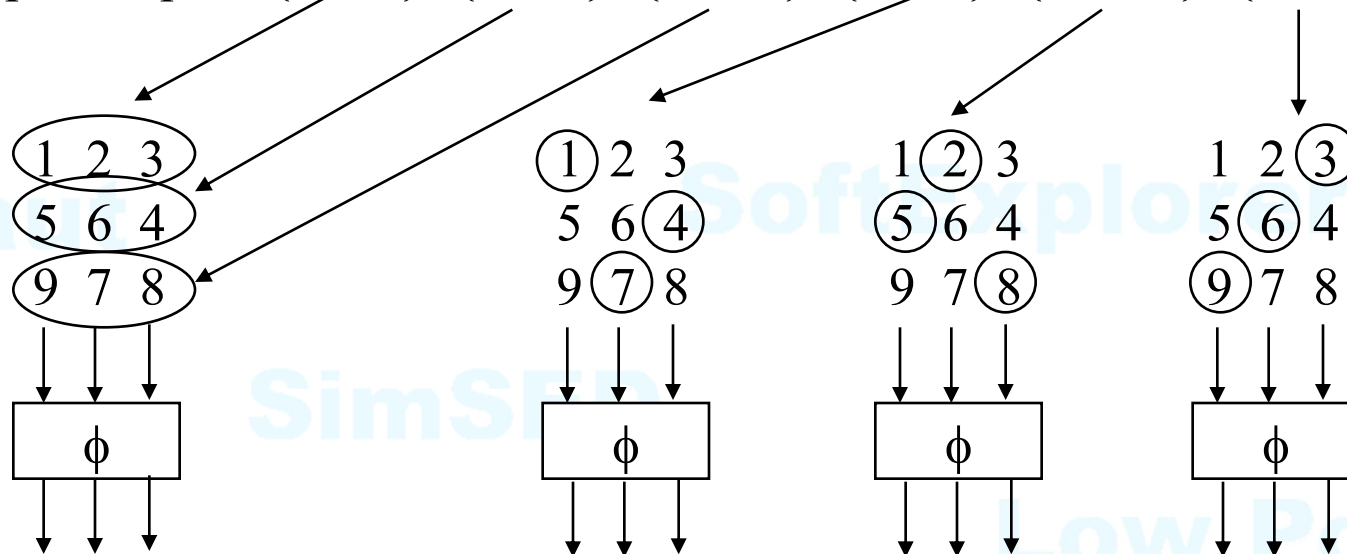
Avantage : Solution entièrement générique.

Inconvénient : Coût matériel important, consommation, perte de cycle.

Solution à la compilation

Trouver a priori un rangement mémoire « non trivial » permettant de résoudre les conflits : la donnée i se trouve à l'adresse $\rho(i)$.

Exemple simple : $\{1,2,3\}$, $\{4,5,6\}$, $\{7,8,9\}$, $\{1,4,7\}$, $\{2,5,8\}$, $\{3,6,9\}$



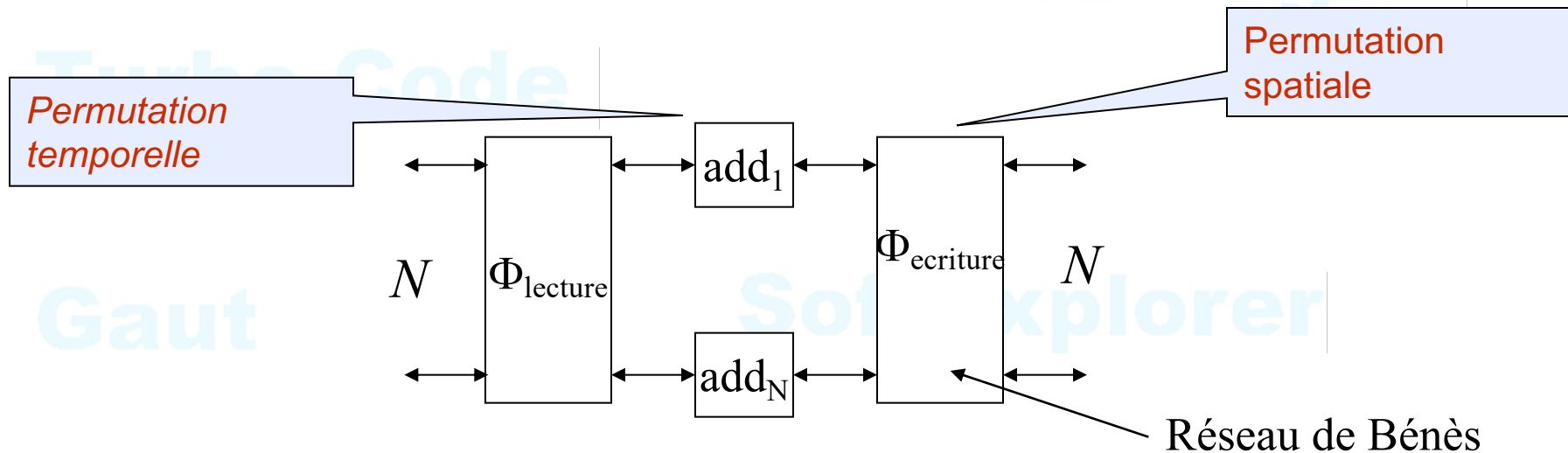
Adresse
mémoire:

1	4	7
2	5	8
3	6	9

$\rho(i) = \{1, 4, 7, 8, 2, 5, 6, 9, 3\}$

Solution à la compilation

Il existe toujours une solution pour un turbo-décodeur 2D.



AVANTAGE : Solution générique

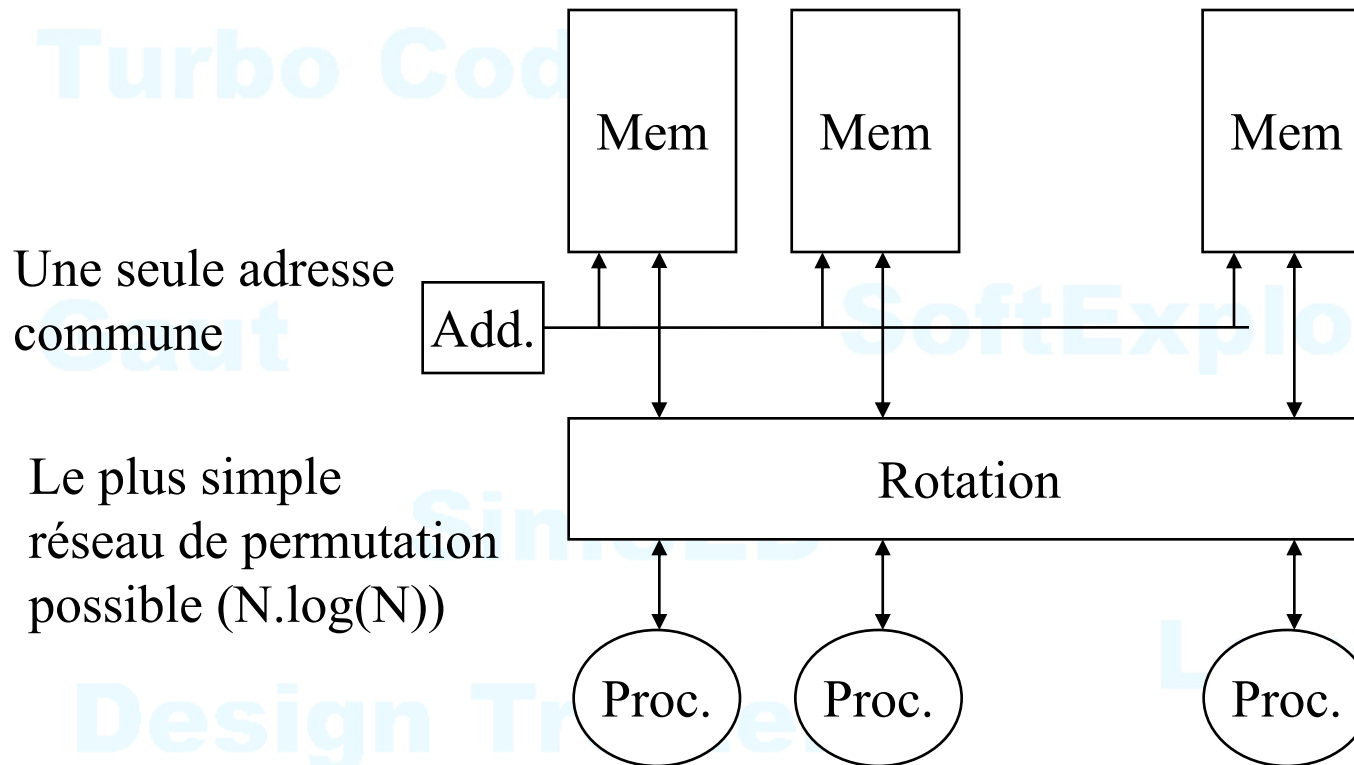
INCONVENIENTS : Besoin de mémoriser :

- la séquence d'adresse lecture/écriture de chaque mémoire.
- les permutations spatiales à effectuer

Réseau de Benes complexe

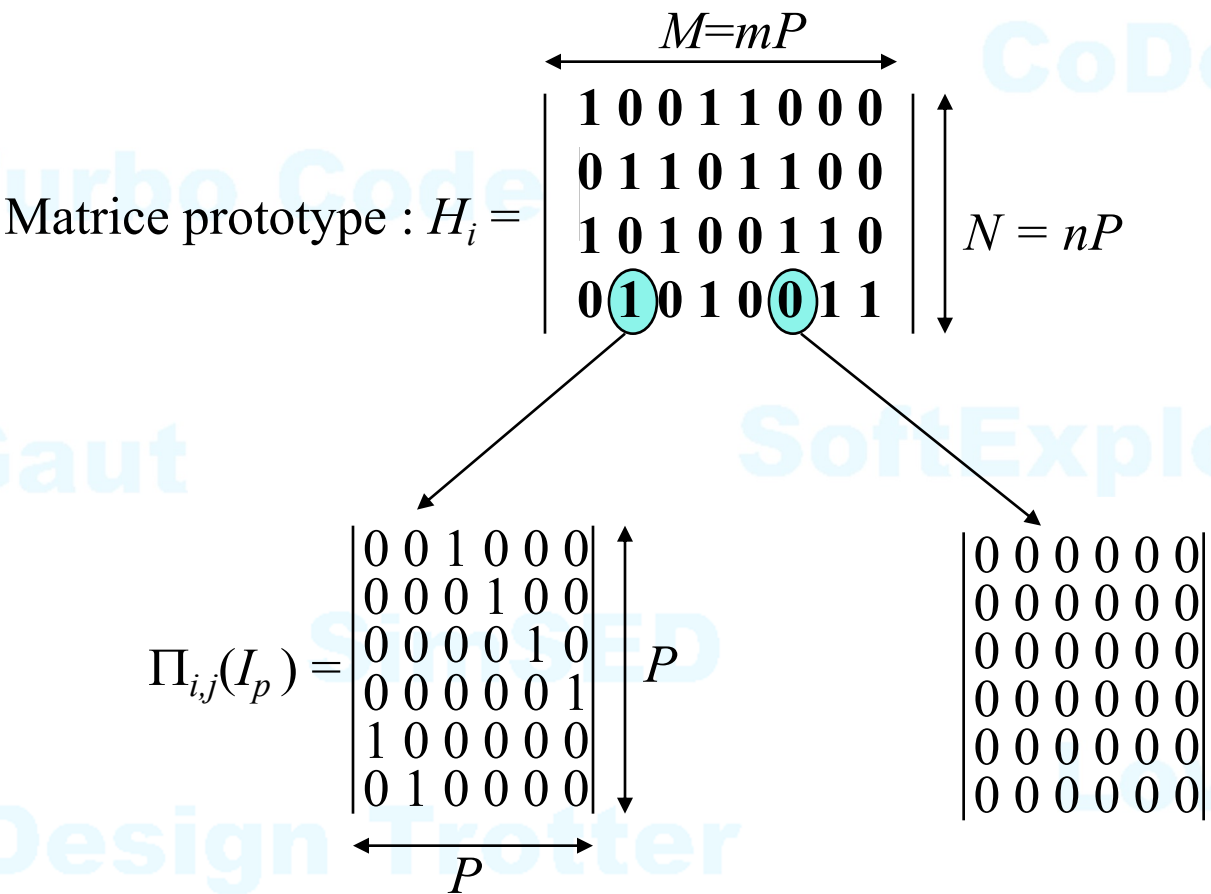
Solution à la conception

Principe : concevoir l'architecture parallèle la plus « simple possible »



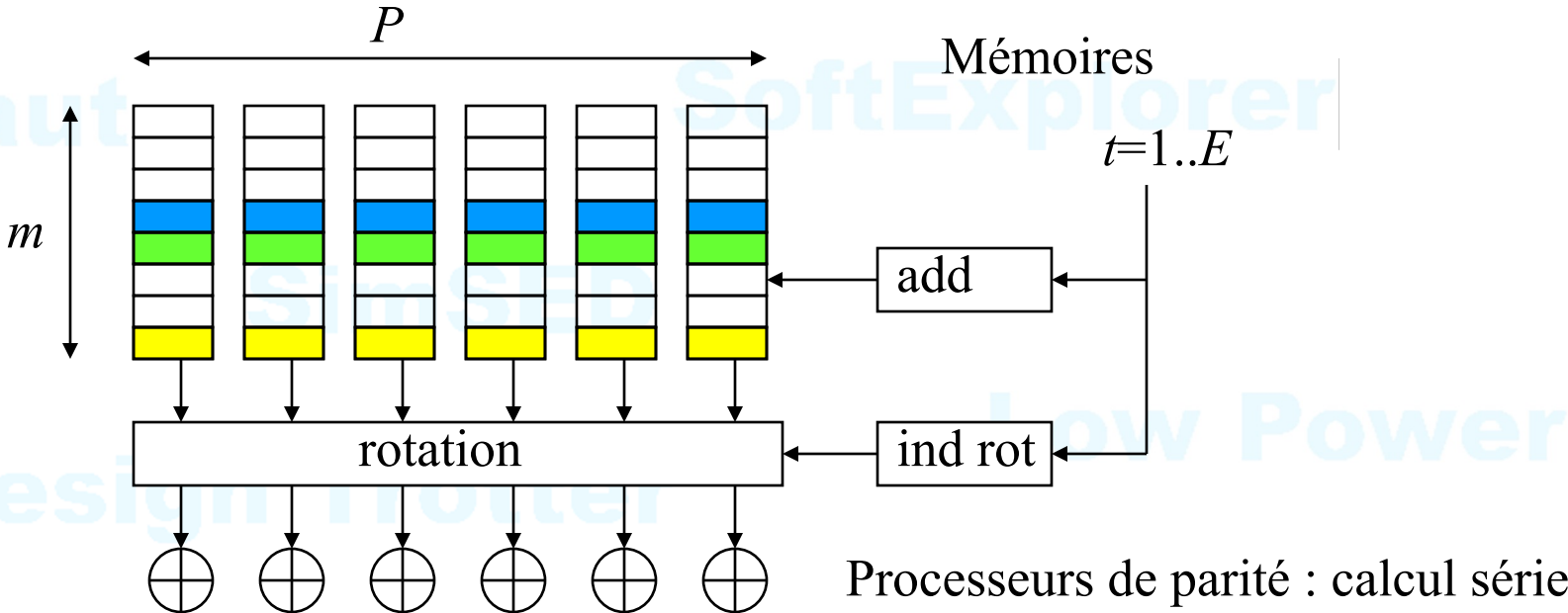
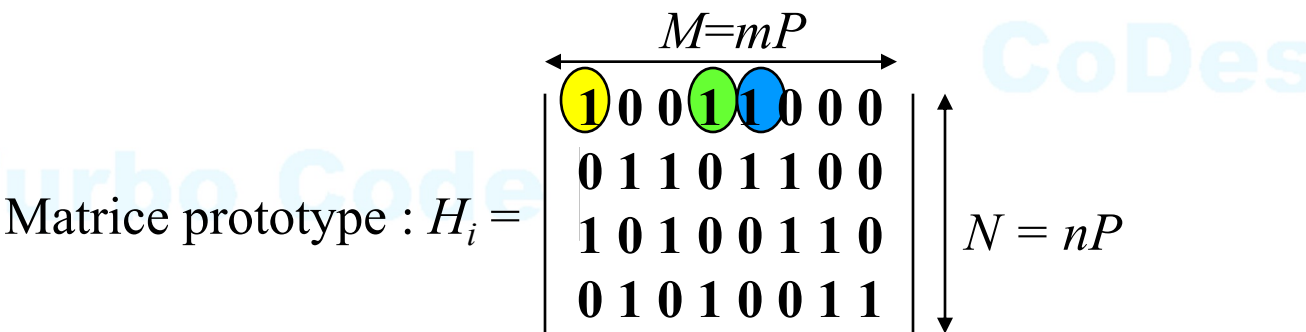
Construire le code autour de cette architecture...

LDPC (ex : DVB-S2)



Très peu d'information pour mémoriser la matrice

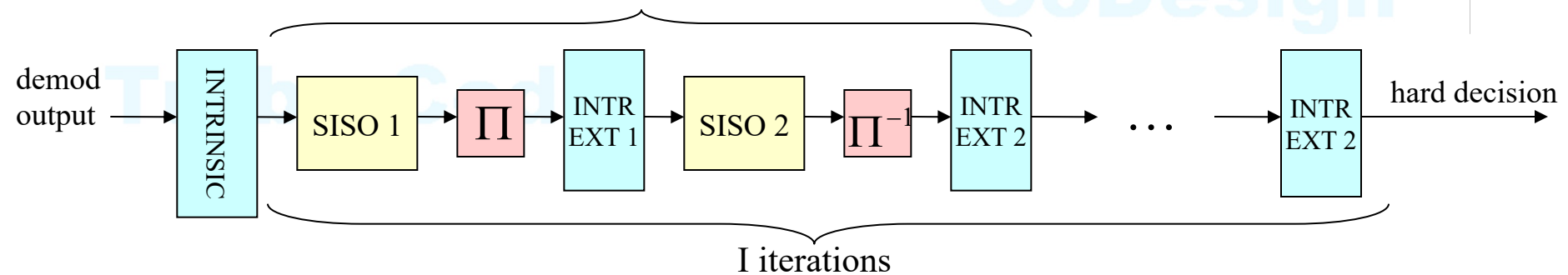
Cas LDPC



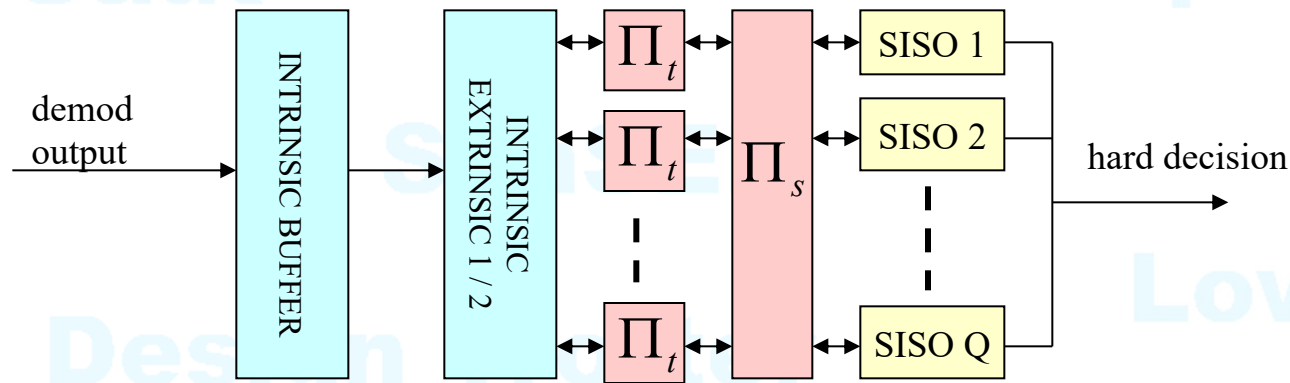
Turbo Décodeur haut débit

o Implémentation « pipe-line » :

Iteration #1



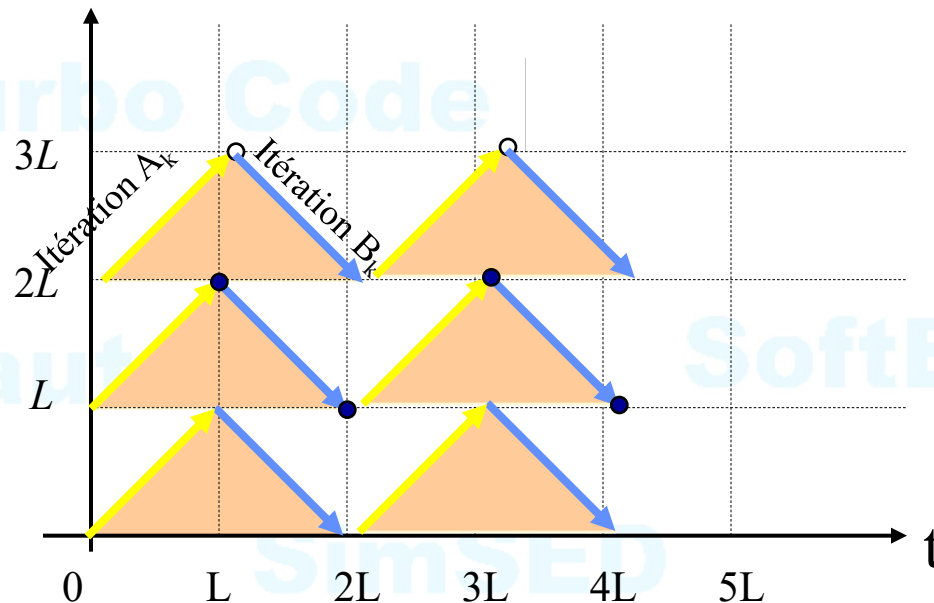
❖ Implémentation parallèle :



Pas de duplication
Mémoire.

Latence + faible.

Problème de placement mémoire...



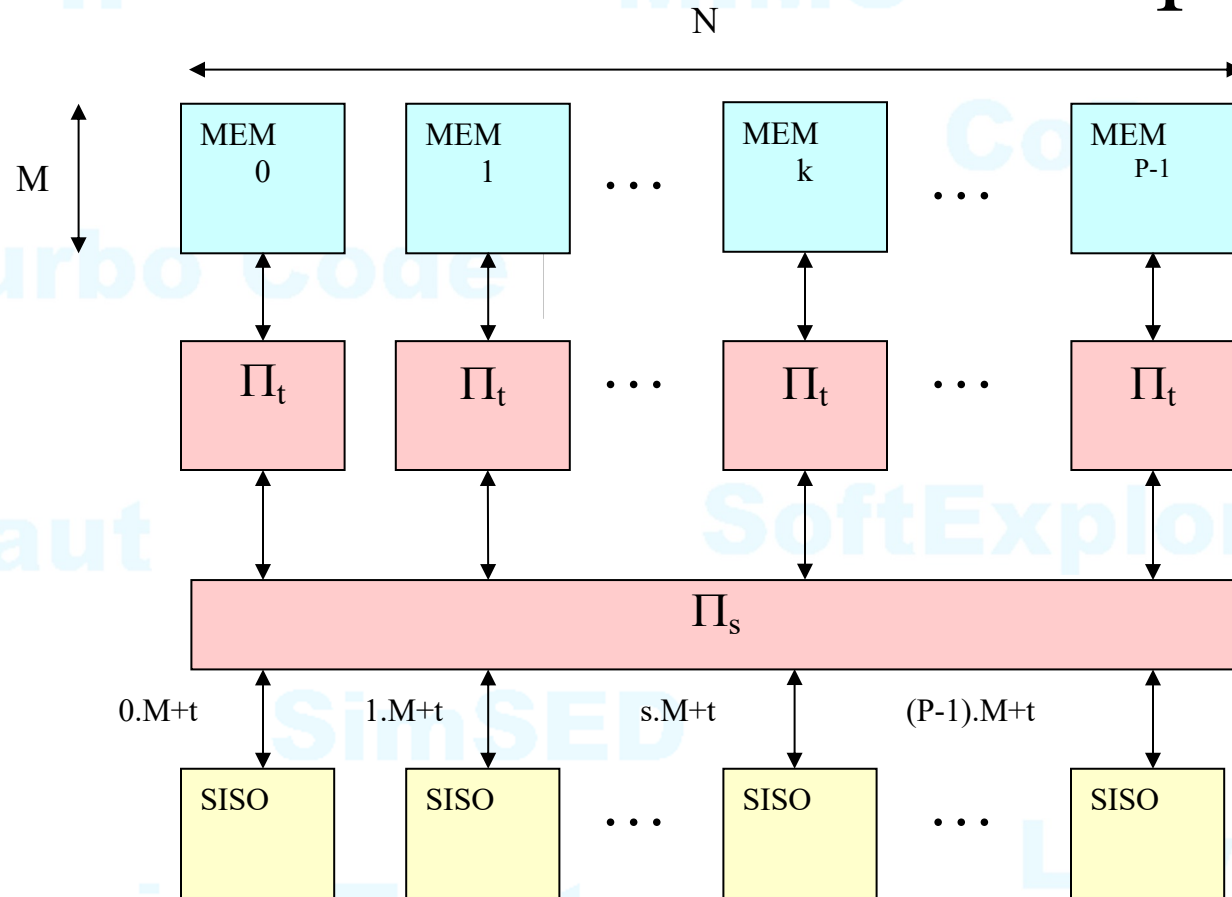
=> idée : découper la trame en P blocs décodés en // par P décodeurs MAP

=> contraintes sur l'architecture

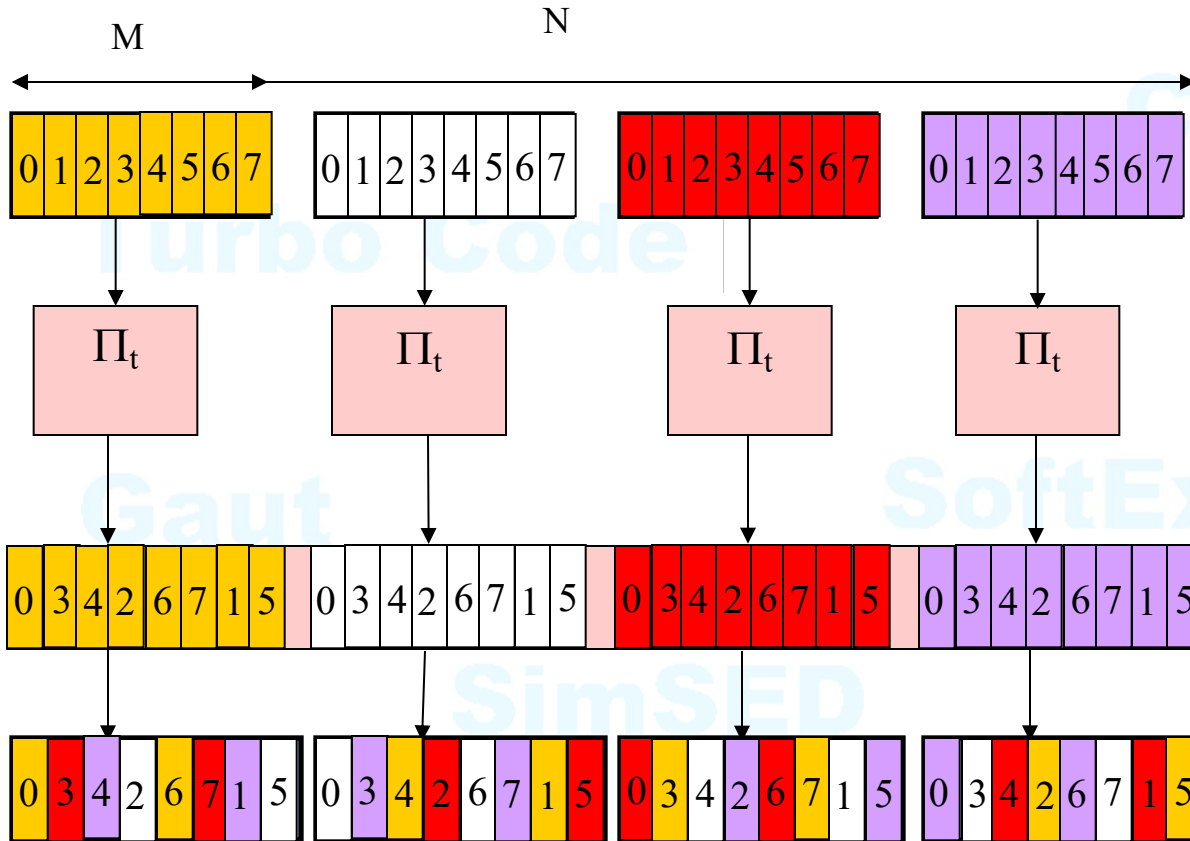
Permet de réaliser les n itérations de décodage en une durée mot de code

=> augmentation débit et diminution mémoire

Cas turbo-code : entrelaceur parallèle



Cas Turbo-Code (ex: DVB-RCS)

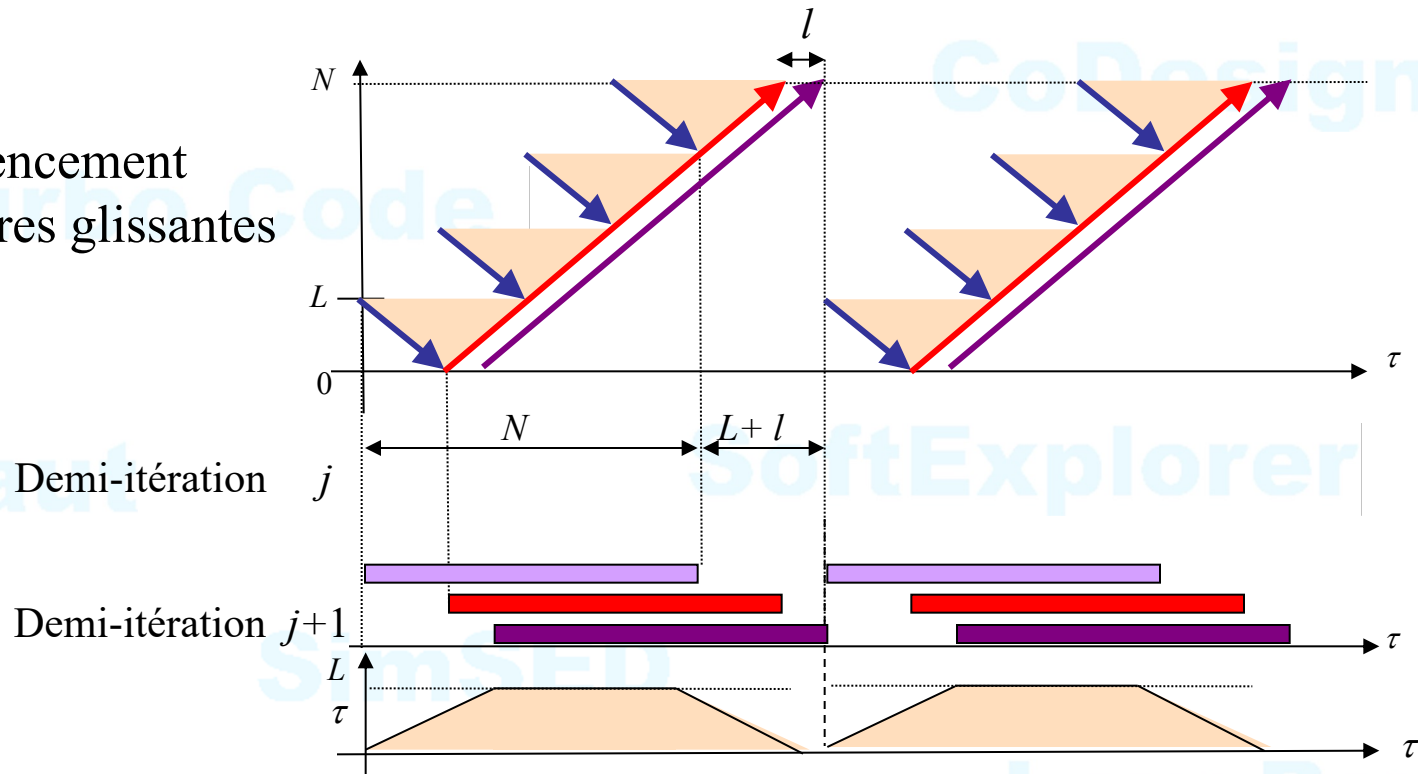


$$\Pi_T(t) = \{0, 3, 4, 2, 6, 7, 1, 5\}$$

$$\Pi_S = \text{rot}\{0, 2, 3, 1\}$$

Séquencement entre demi-itération

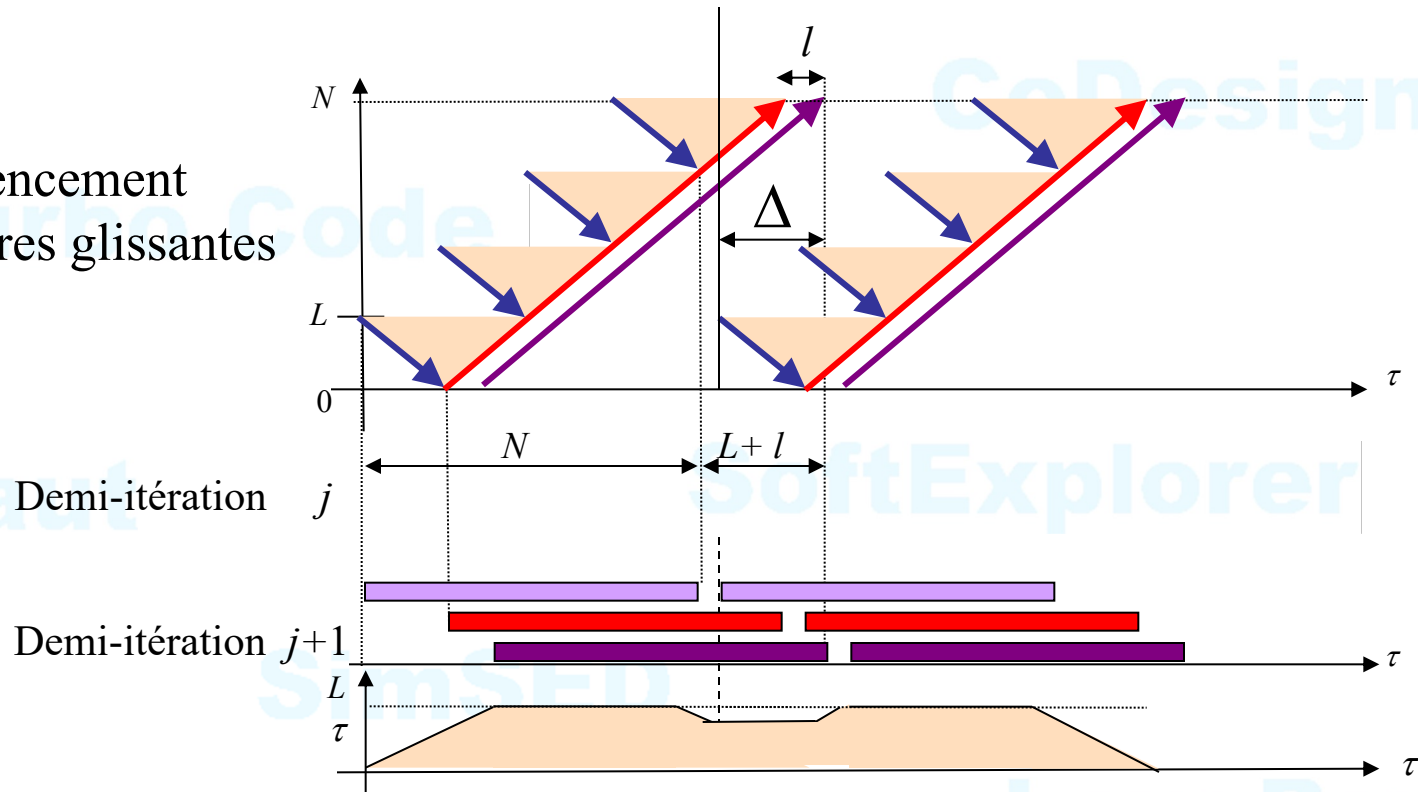
Séquencement
Fenêtres glissantes



Perte moyenne de 25 % des capacités de calcul.

Idée : avancer le début de la 2^{ème} demi-itération

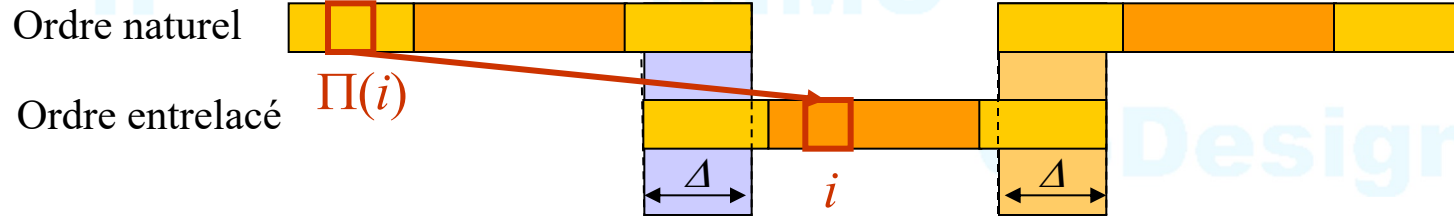
Séquencement
Fenêtres glissantes



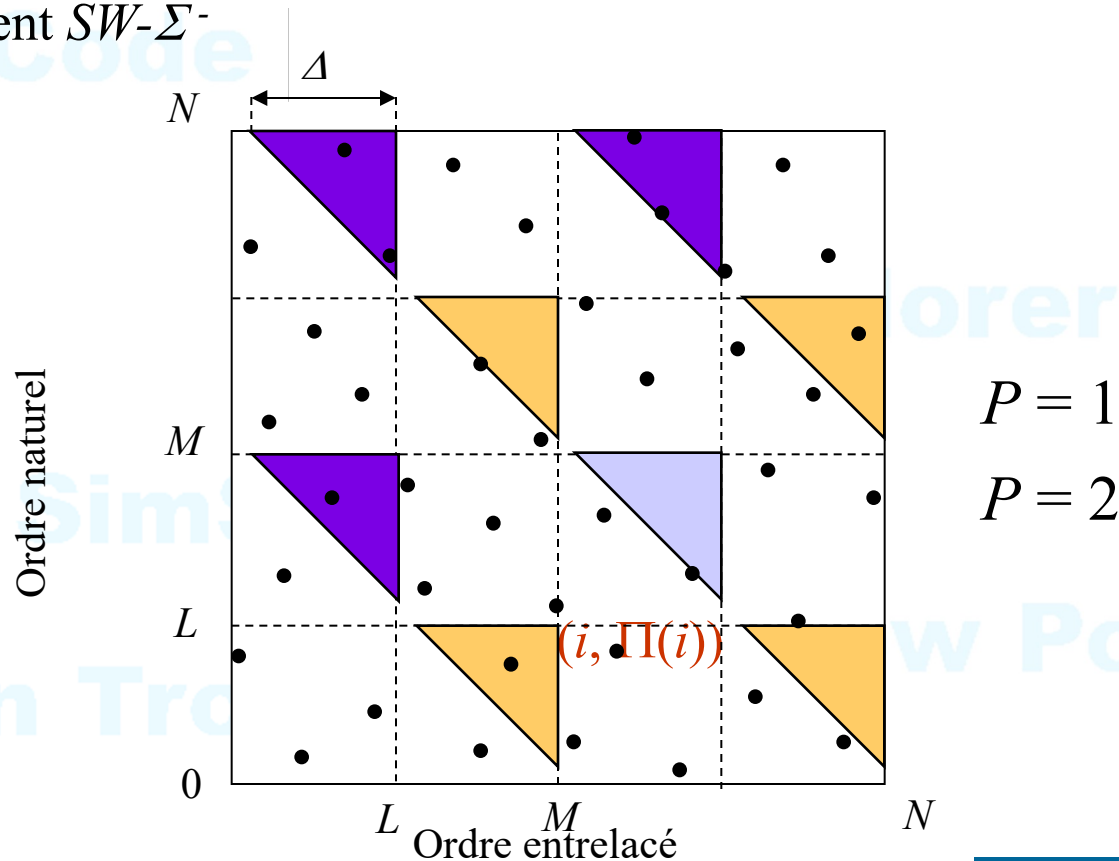
⇒ Meilleure utilisation de l'architecture.



Représentation graphique



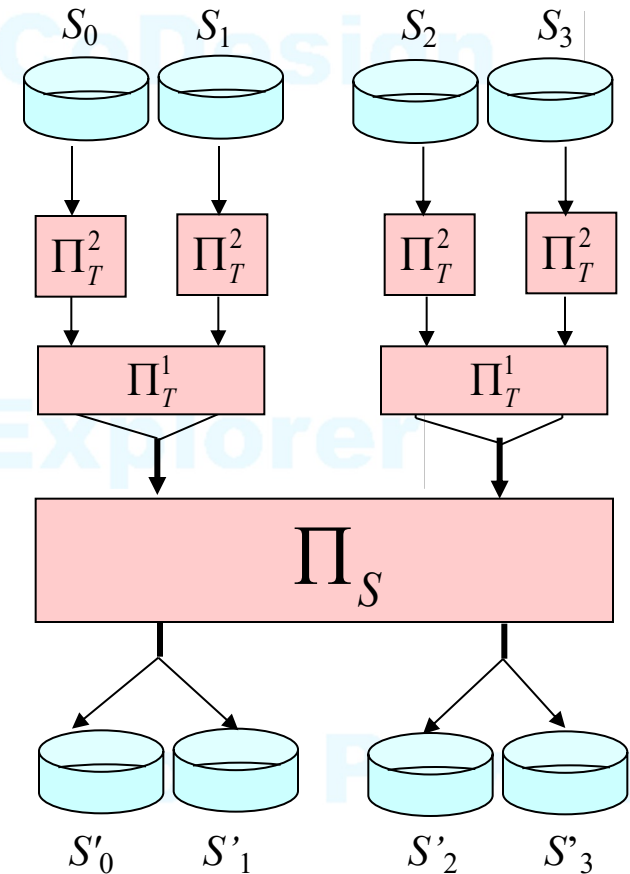
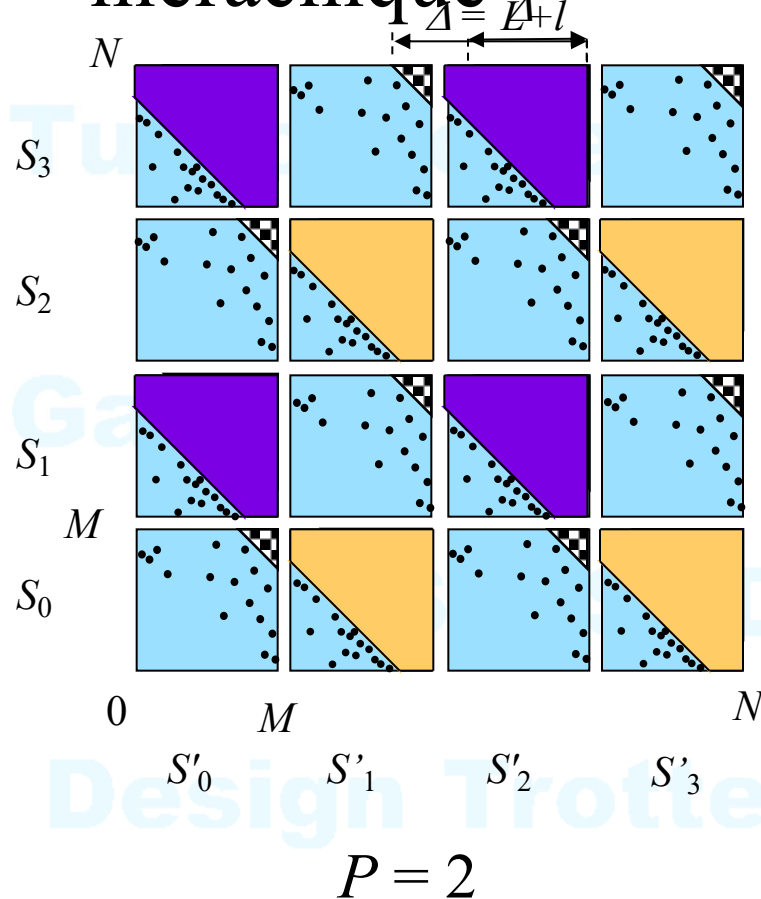
Séquencement $SW-\Sigma^-$



Conflits de
consistance

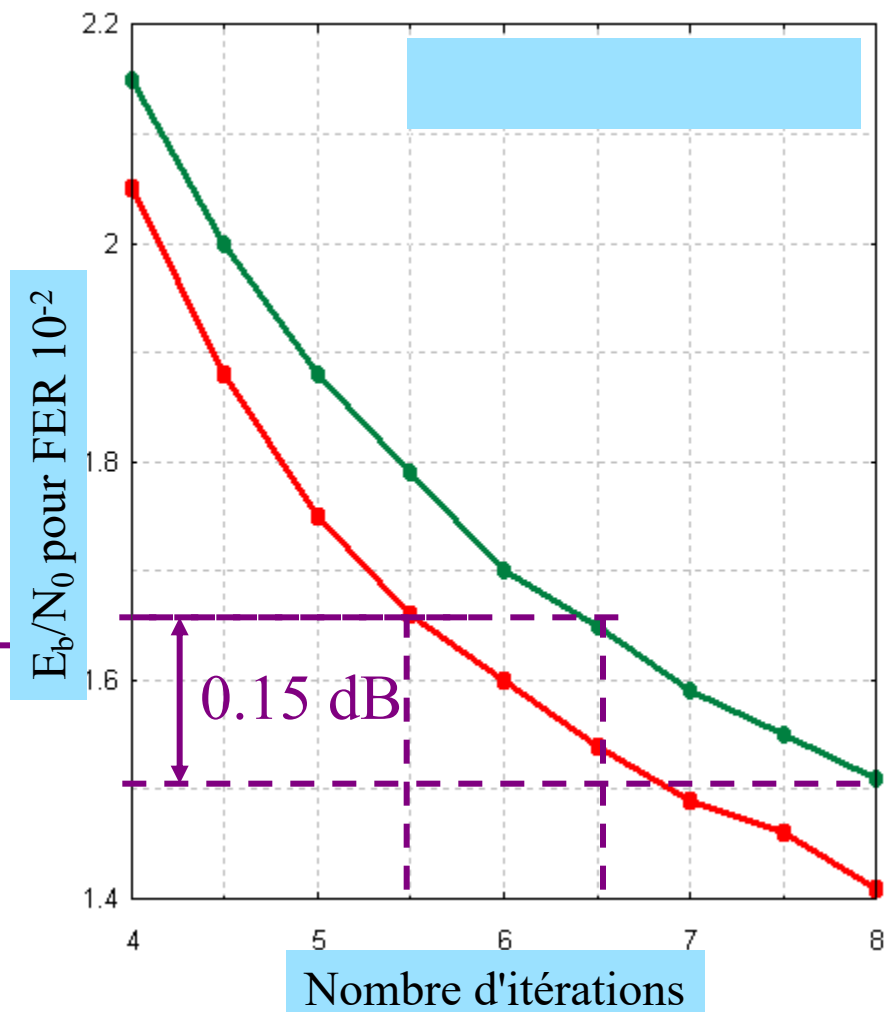
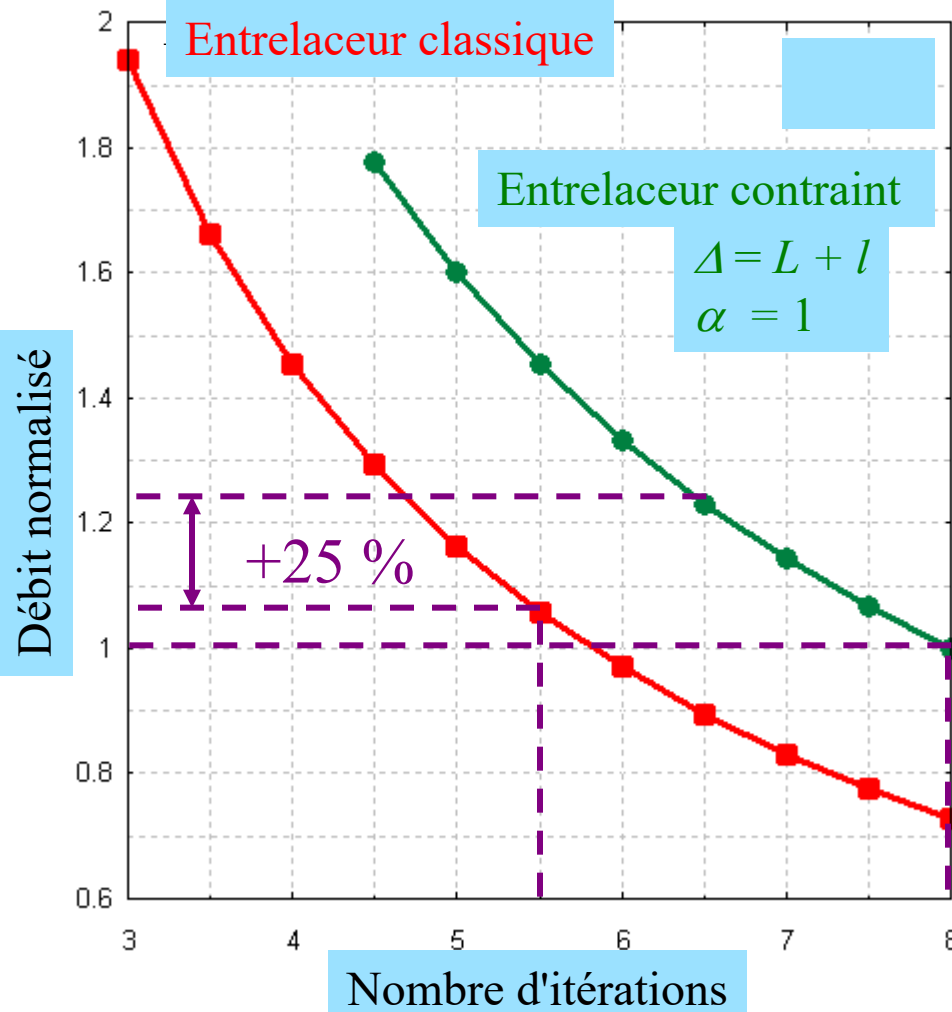
Exemple d'entrelaceur contraint

- o Masque d'entrelaceur => entrelaceur hiérarchique



Performances de l'entrelaceur contraint

- o $N = 1024$ symboles, $P = 16$ processeurs (débit de 100



PLAN

❶ Code convolutif et son décodage

❷ Turbo-code

❸ Code LDPC

❹ Décodeurs haut débits

❹ Parallélisme

❹ Fréquence d'horloge

❹ Critère d'arrêt et bufferisation

❹ Conclusions

Augmenter la fréquence d'horloge...

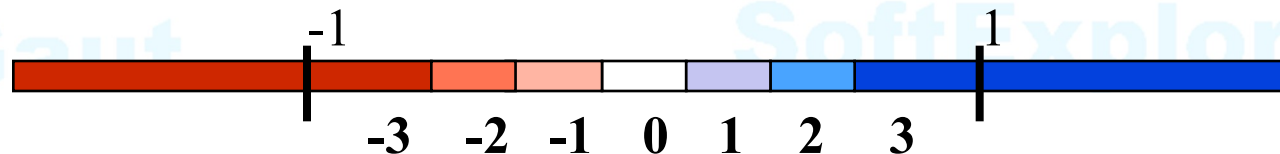
- o *Changer de technologie*
- o Diminuer le chemin critique
 - Diminuer le nombre de bit en interne
 - Utilisation d'algorithme sous-optimal
 - Optimisation architecturale
 - Pipe-line

Diminuer le nombre de bit en interne

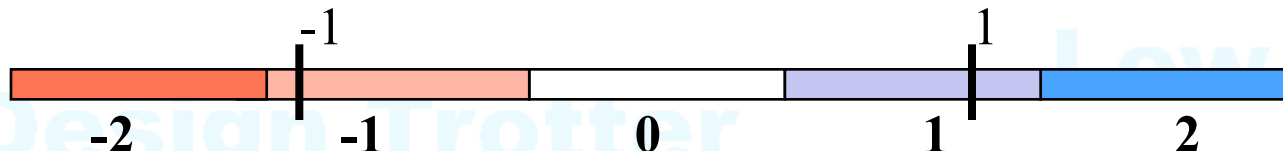
- Quantification de l'entrée sur b_{LLR} bits

$$y_Q = \text{sat}\left(\left\lfloor \frac{y}{M} 2^{b_{LLR}-1} + 0,5 \right\rfloor, 2^{b_{LLR}-1}\right)$$

Si $M \ll 1 \Rightarrow y_Q$ prend majoritairement les valeurs $\pm 2^{b_{LLR}-1}$



Si $M \gg 1 \Rightarrow y_Q$ prend majoritairement la valeur $\{-1, 0, 1\}$



Comment choisir b_{LLR} et M ?

Compromis performances complexité

- o En fonction des performances du code (taille, type, rendement) il existe un SNR de fonctionnement. Pour ce SNR et un b_{LLR} , il existe une valeur optimal de M .
- o Après, le choix de b_{LLR} est une question de compromis...
- o Une fois b_{LLR} choisi, on en déduit la précision des variables internes. Il existe des possibilité de réduction des dynamiques par saturation, arrondi ou troncature (c'est de la cuisine interne).
- o En général, $b_{LLR} = 3$ ou 4 est un bon compromis. A partir de $b_{LLR} = 6$, il n'y a plus rien à gagner.

Utilisation d'algorithmes sous-optimaux

Les algorithmes itératifs sont eux même sous-optimaux par rapport au MAP
 Calcul dans le domaine logarithmique
 Il est possible de simplifier la fonction « sum ».

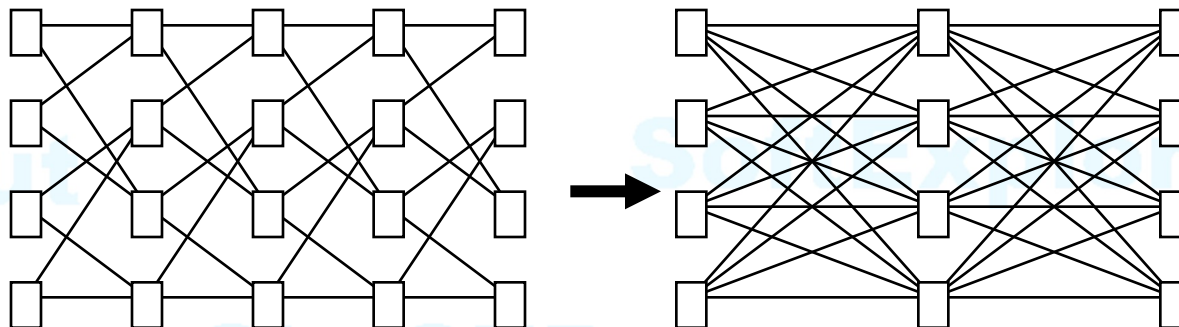
Turbo-code	LDPC
Log-Map Max-Log-Map Toutes les versions intermédiaires	Min-Sum avec offset λ -min A-sum

Gain de surface, de vitesse et de consommations...
 Perte de performances

Optimisation architecturale : Turbo-code

ACSO $\rightarrow$ OACS

Treillis compaction (2 bits/cycle d'horloge) :

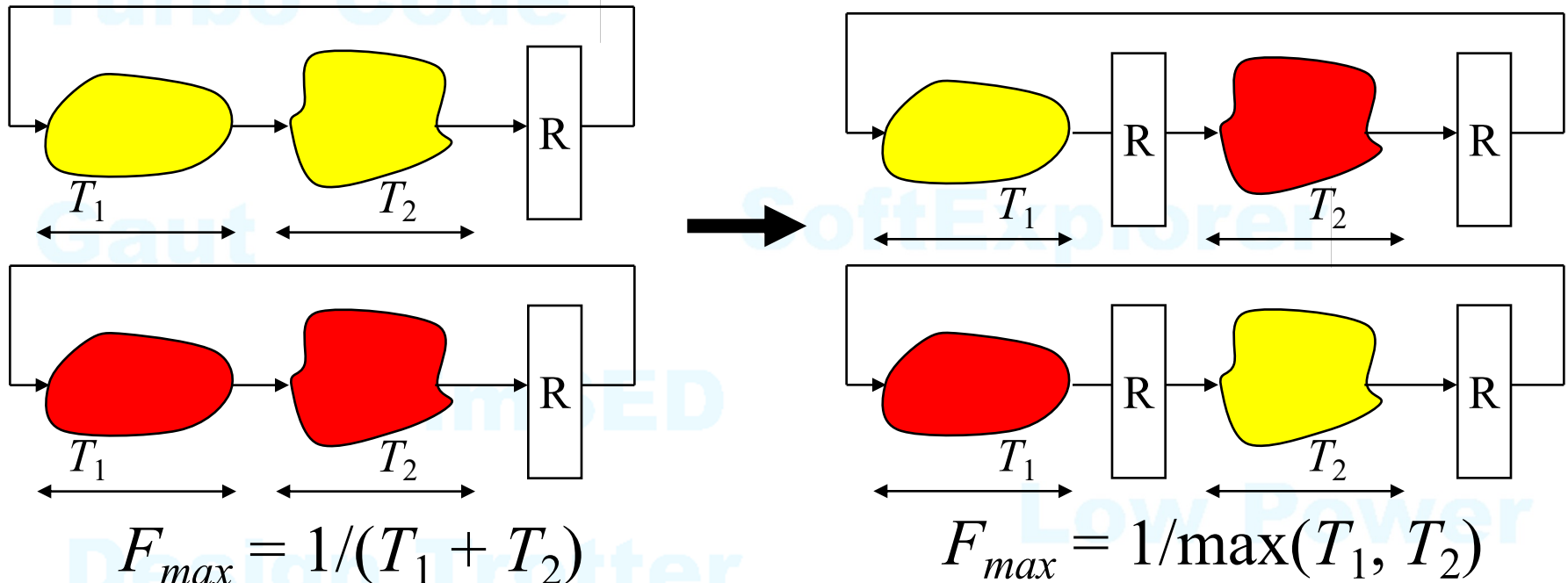


Logique redondante + comparaison MSB first :

$\Rightarrow$ temps critique turbo-code = 1 Full Adder + 1 registre

Technique Pipe-line

Principe : couper le chemin critique en « entrelaçant » deux mots de code (ou blocs de données) différents :



Technique très efficace

PLAN

❶ Code convolutif et son décodage

❷ Turbo-code

❸ Code LDPC

❹ Décodeurs haut débits

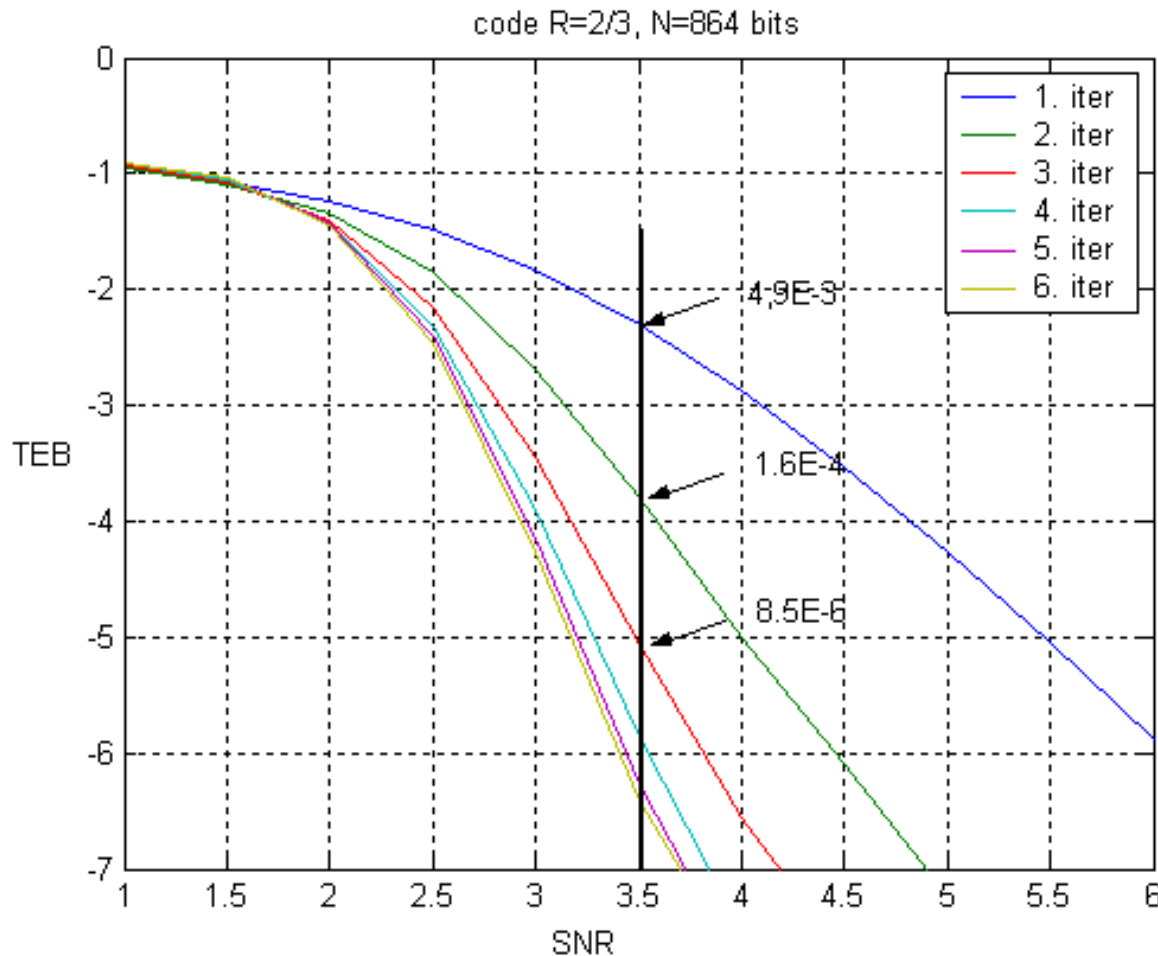
❹ Parallélisme

❹ Fréquence d'horloge

❹ Critère d'arrêt et bufferisation

❹ Quelques résultats théoriques

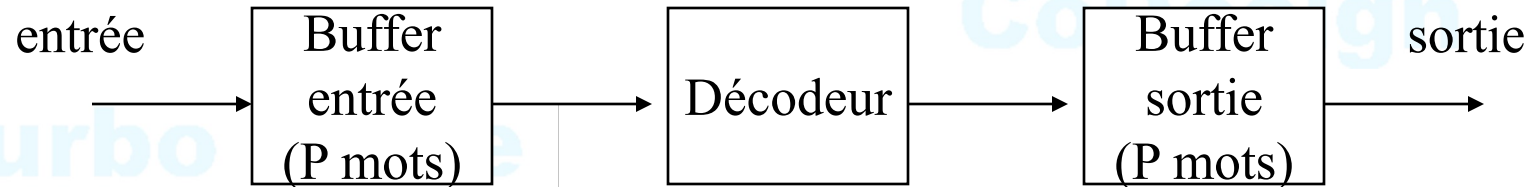
Principe de la bufferisation



A 3,5 dB, dans la majorité des cas, le mots est décodé avant 6 itérations.

Est-il utile de faire un décodeur pouvant TOUJOURS faire 6 itérations ?

Principe de la bufferisation



Tant que le buffer d'entrée n'est pas rempli et que le mot courant est non décodé, refaire une itération (ou demi-itération) de décodage.

La stratégie optimale s'obtient par traitement sur les chaînes de Markov

Méthode très efficace (réduction de la complexité du décodeur sans perte de performances) mais :

- nécessité d'avoir un critère d'arrêt
- augmentation de la latence de décodage
- ajout de (petite) mémoire en entrée et en sortie

Critères d'arrêts

Cas LDPC : Il suffit de vérifier que toutes les contraintes de parités sont vérifiées.

Cas Turbo-Code : Il existe différents critères d'arrêt

- arrêt si le mot décodé reste identique entre deux itérations.
- arrêt si le minimum des extrinsèques dépasse un seuil

Ces critères sont efficaces et ne dégradent pas de façon sensible les performances.

PLAN

❶ Code convolutif et son décodage

❷ Turbo-code

❸ Code LDPC

❹ Décodeurs haut débits

❹ Parallélisme

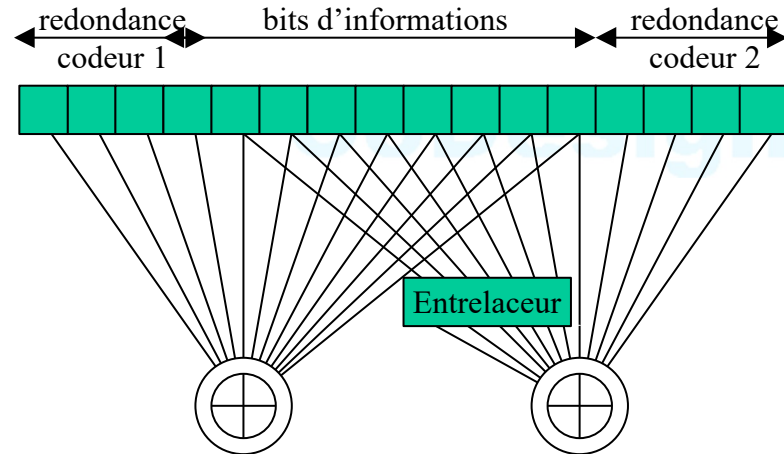
❹ Fréquence d'horloge

❹ Critère d'arrêt et bufferisation

❹ Conclusion

LDPC vs Turbo-Code

Une autre vision d'un turbo-code



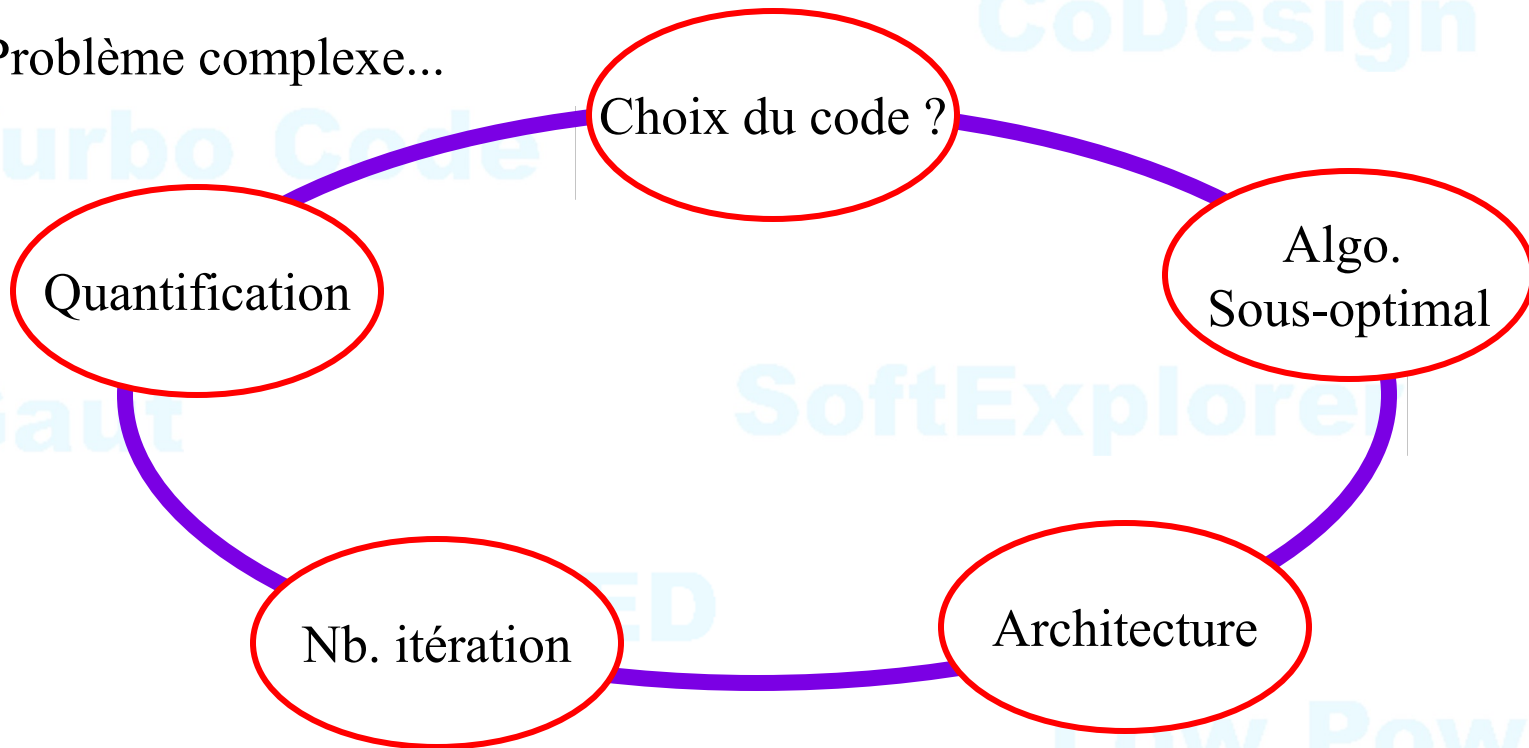
LDPC et Turbo-code sont aux deux extrémités du spectre des graphes de Tanner

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

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

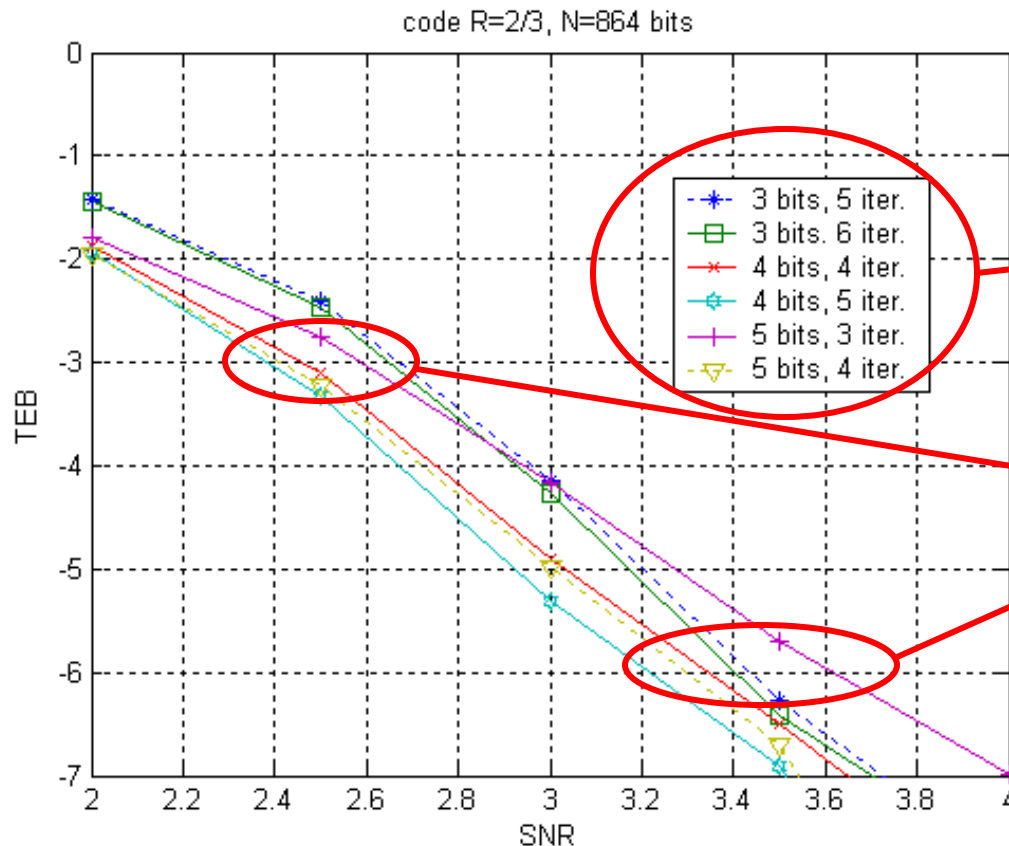
Décodeur haut débit : conclusion

Problème complexe...



Et de plus, tester une solution pour un FER faible prend (beaucoup) de temps...

Exemple compromis perf./complexité



Complexité

La hiérarchie
change en
fonction du SNR

Problème complexe

Solution pour faciliter la recherche

Utiliser au maximum les outils théoriques :

- EXIT chart

- Méthode impulsionnelle

- ...

Technique logicielle en créant une base de test

- Créer une base de données de mot non décodé avec les paramètres

- « pire cas »

- Créer un modèle de fonction de coût

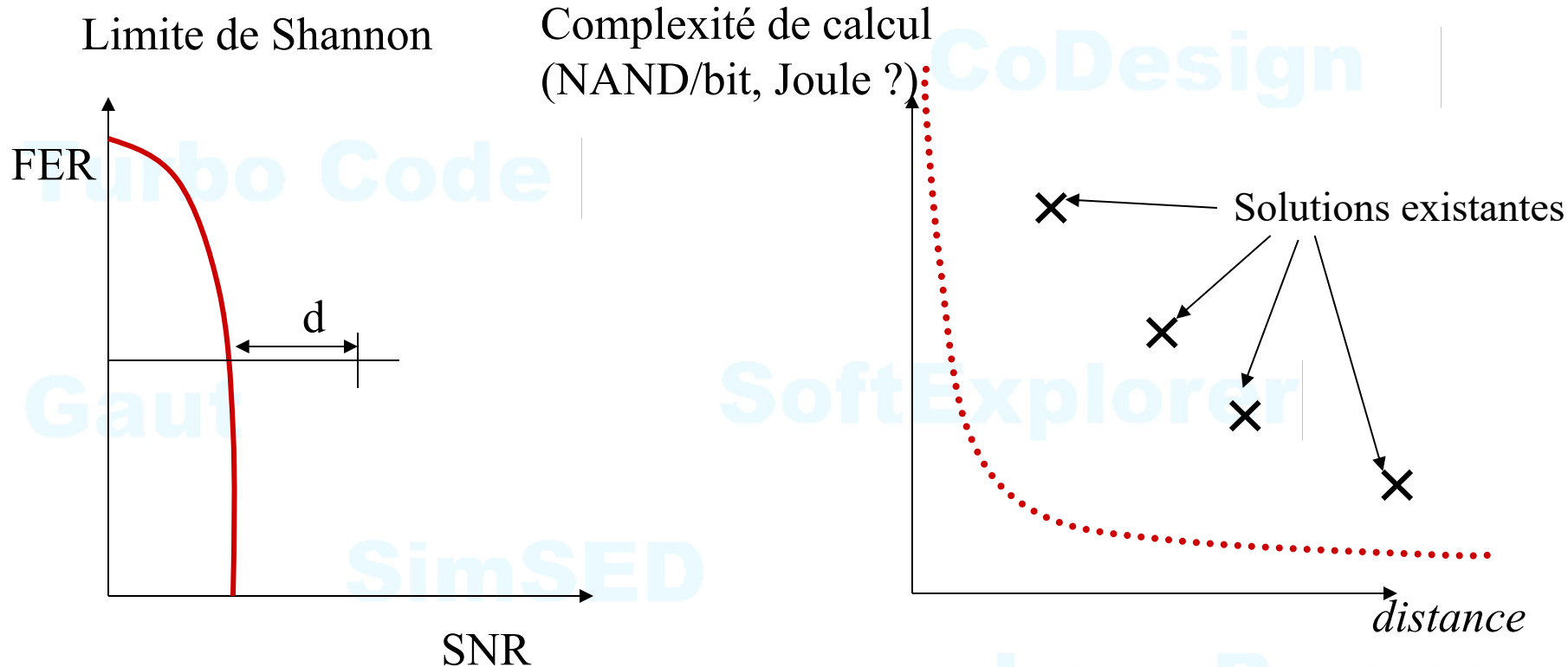
- Optimiser vos paramètres sur votre base de donnée

Remplacer la simulation logicielle par l'émulation sur FPGA

- Codeur/décodeur VHDL paramétrable + Générateur de bruit

- Test/validation des configurations à très haut débit

Question ouverte ?



Existe t'il une borne inférieure de complexité ?

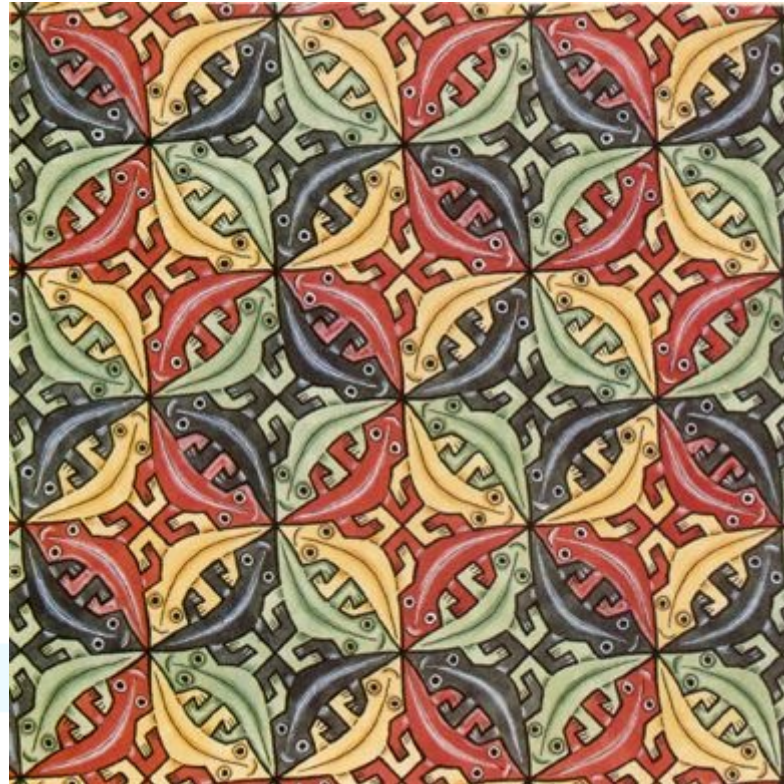
Si oui, est-on loin... ?

Quelques défis dans le domaine du codage...

- o Les codes à 10 Giga bits/seconde et plus...
- o Implémentation de codes LDPC sur $GF(2^q)$
- o Après le débit et la surface, la consommation...
- o Les décodeurs analogiques

Conclusion

Code, codeur et
décodeur sont
étroitement liés.



Escher (1898 - 1972)

Ils doivent être conçu conjointement pour
couvrir l'espace du possible