Proposition d'une méthodologie adéquation algorithme architecture

Emmanuel Boutillon

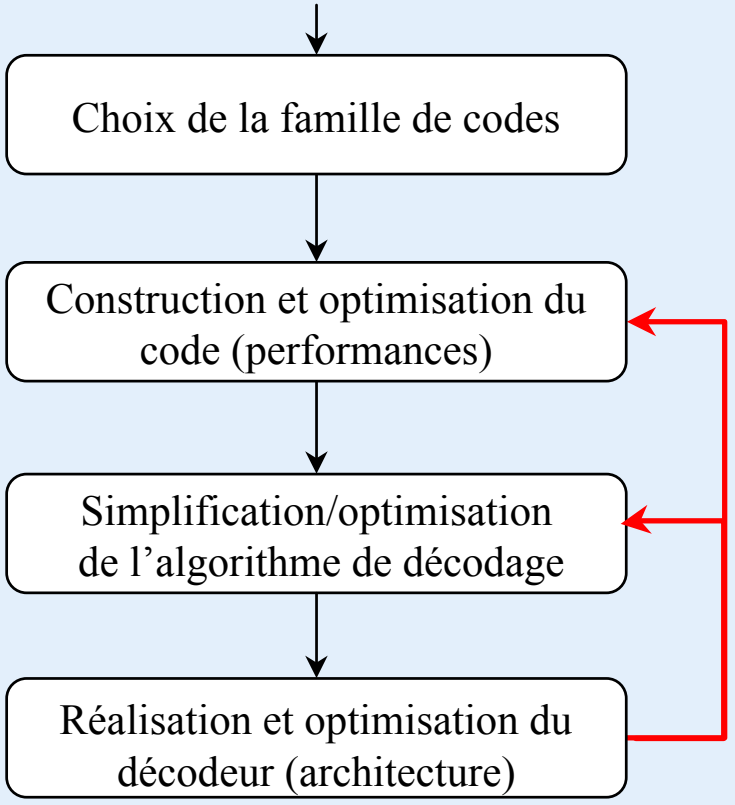
LESTER
Université de Bretagne Sud

David Gnaedig (Turbo-Concept)
Frédéric Guilloud (ENST-Bretagne)

L.E.S.T.E.R
Laboratoire d'Electronique des Systèmes TEmps Réel

Démarche de conception d'un système de codage

❖ Adéquation Algorithme Architecture:



➢ Comment mettre en place la rétro-action… ?

1) Oublier l'algorithme…

et optimiser/modifier sans a priori

2) …se rappeler de l'algorithme

et voir ce que l'on peut faire…

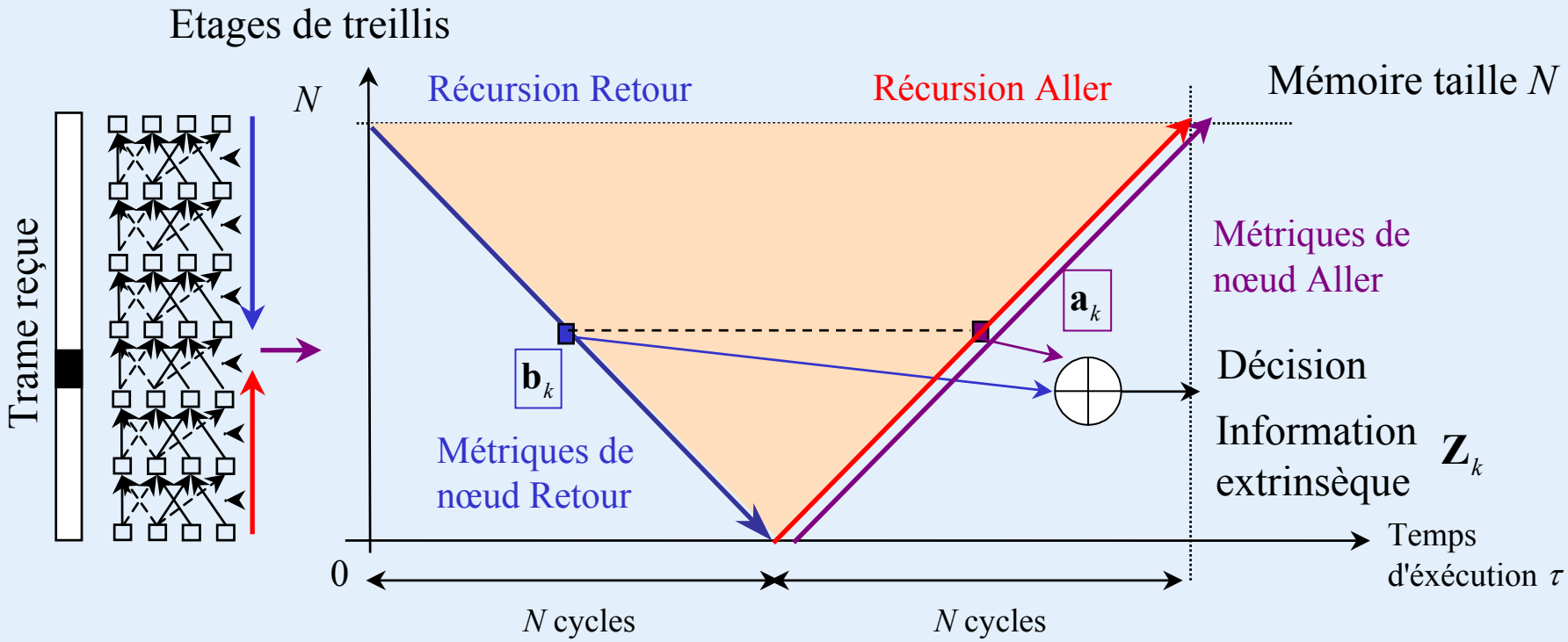
Plan

Turbo-code : Optimisation d'un entrelaceur/architecture

LDPC : Architecture pour le "shuffled-scheduling"

Décodage à entrées et sorties pondérées

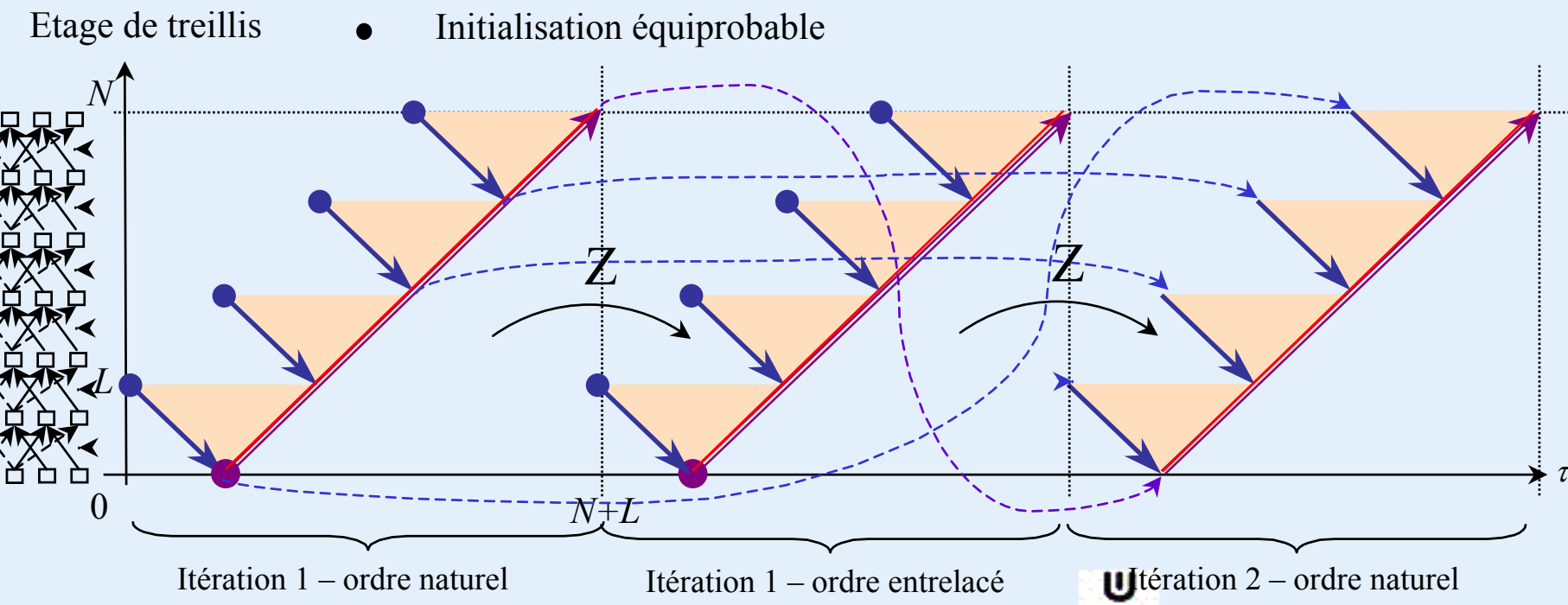
- Approche Adéquation Algorithme Architecturelgorithme (sous-optimal) Max-Log-MAP [Robertson *et al.*, 1995]:



Séquencement *"triangle"*

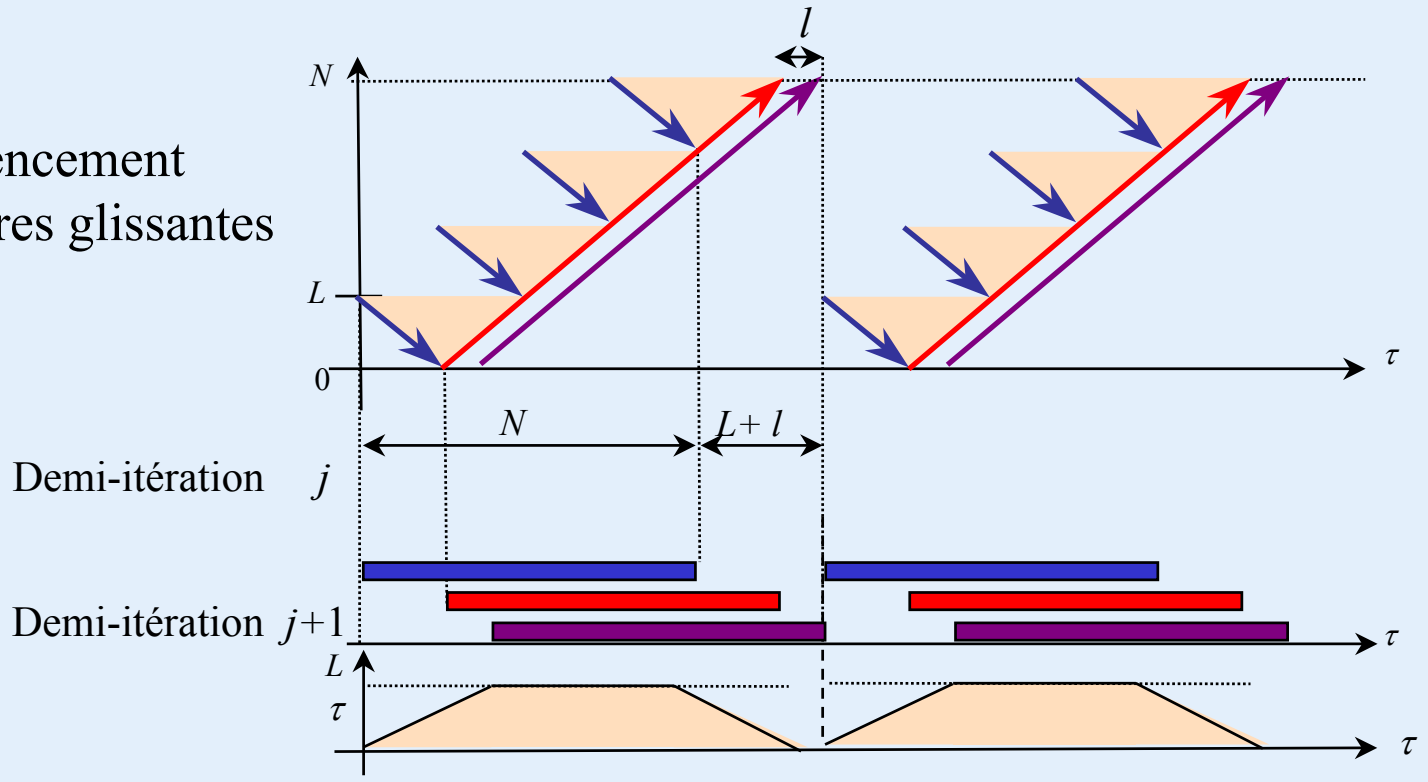
Séquencement fenêtre glissante

- Algorithme à fenêtre glissante [Benedetto *et al.*, 1996]
  - Memoire taille L
- Initialisation des récursions par valeurs finales de l'itération précédente [Dingninou *et al.*, 1999]



Séquencement entre demi-itération

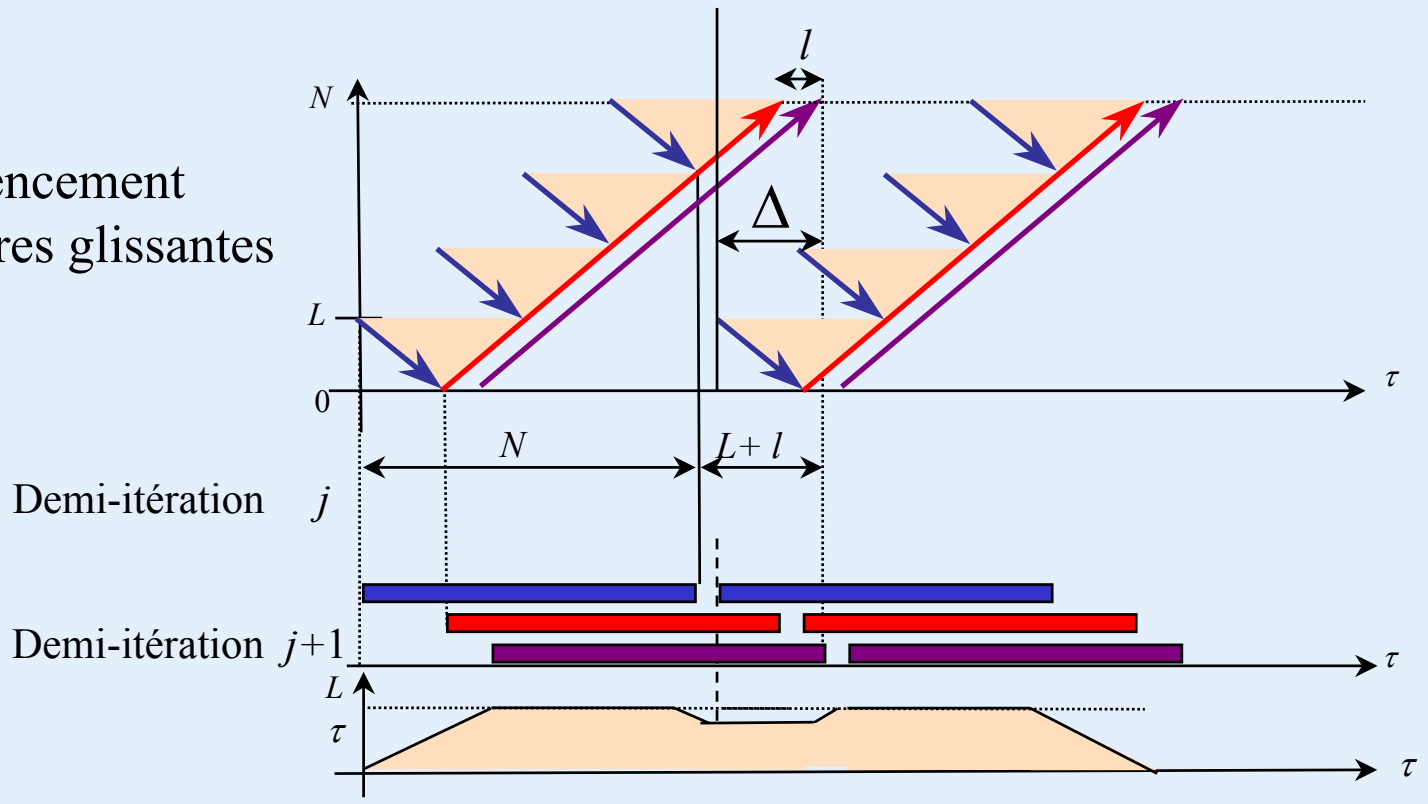
Séquencement
Fenètres glissantes



Perte moyenne de 25 % des capacités de calcul.

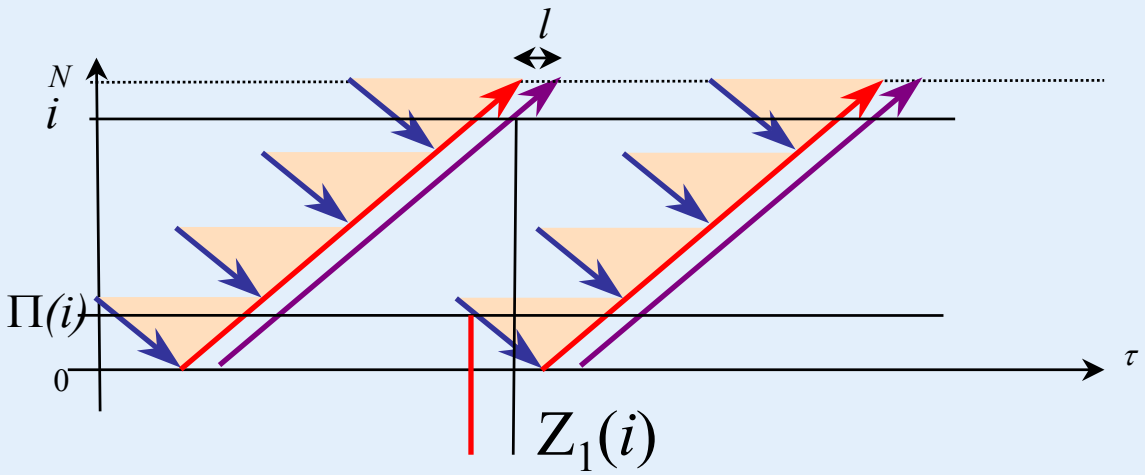
Idée : avancer le début de la 2ième demi-itération

Séquencement
Fenètres glissantes

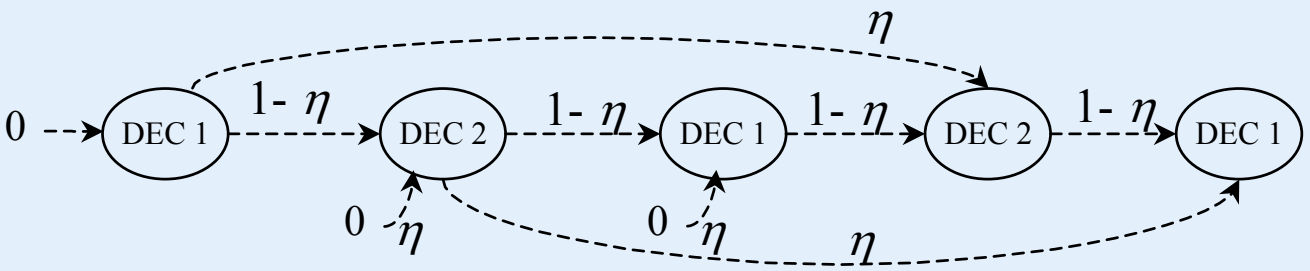


⇒ Meilleure utilisation de l'architecture.

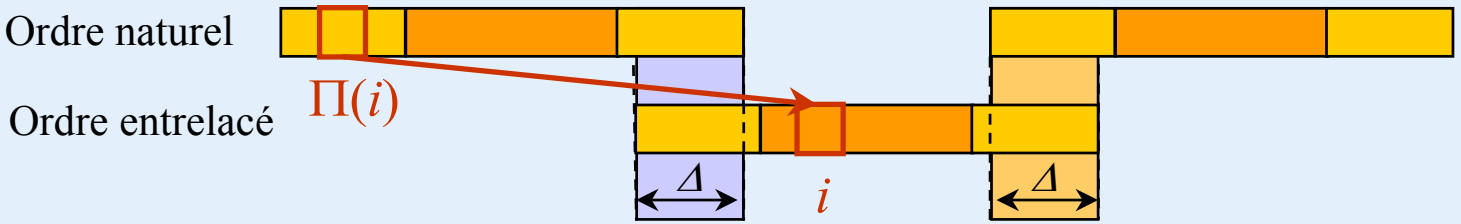
Problème de consistance…



$Z_1(i)$ de l'itération courante n'existe pas encore

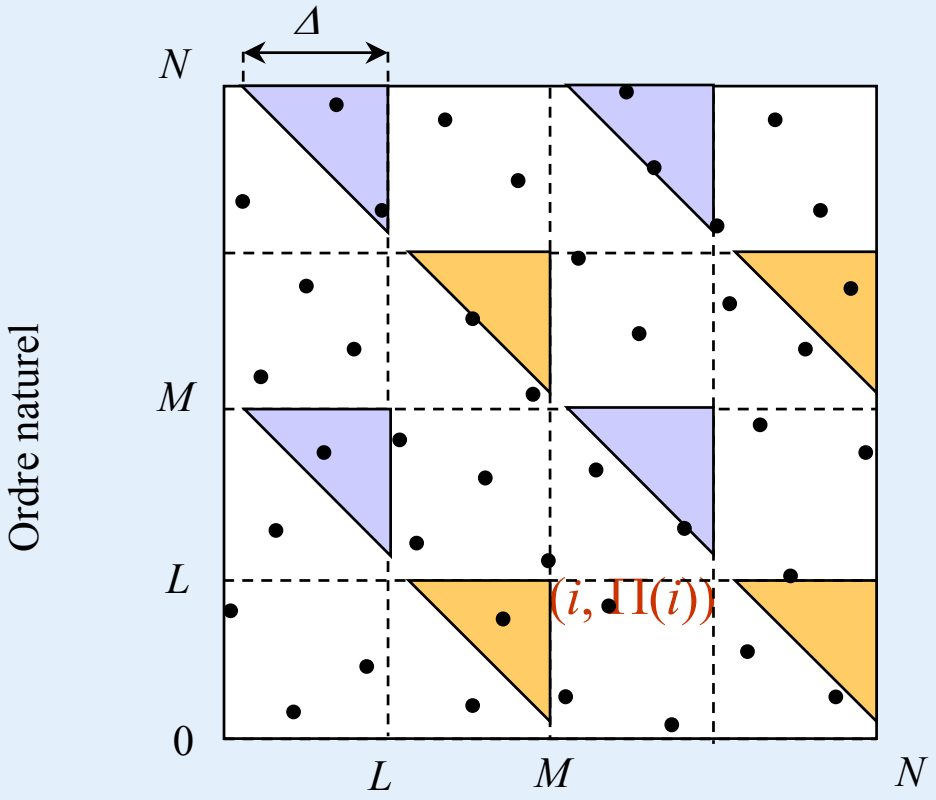


Représentation graphique des conflits de consistence



Séquencement *SW-Σ⁻*

Conflits de consistance

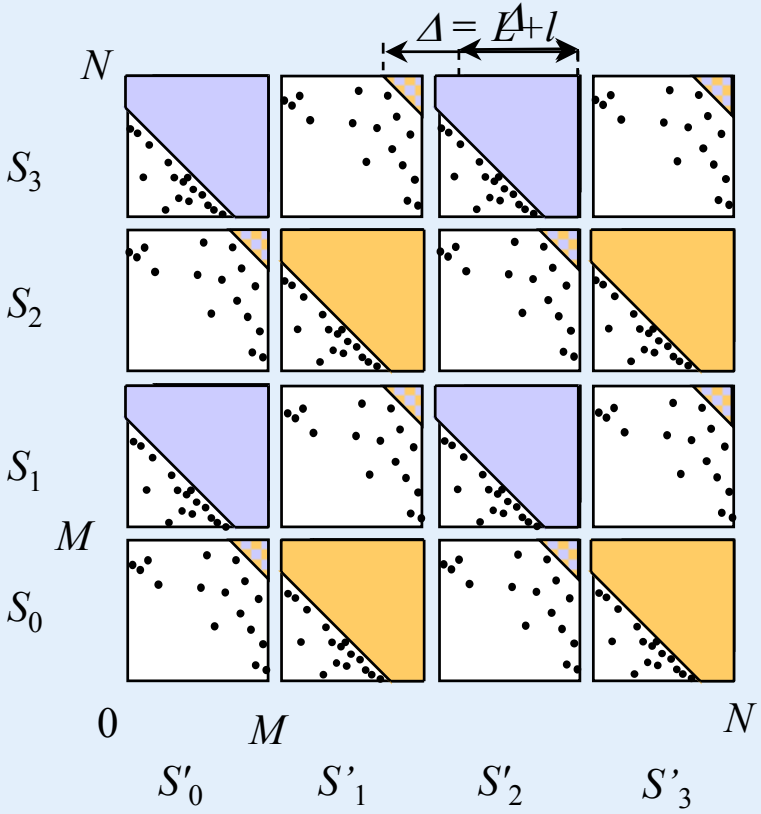


$P = 1$

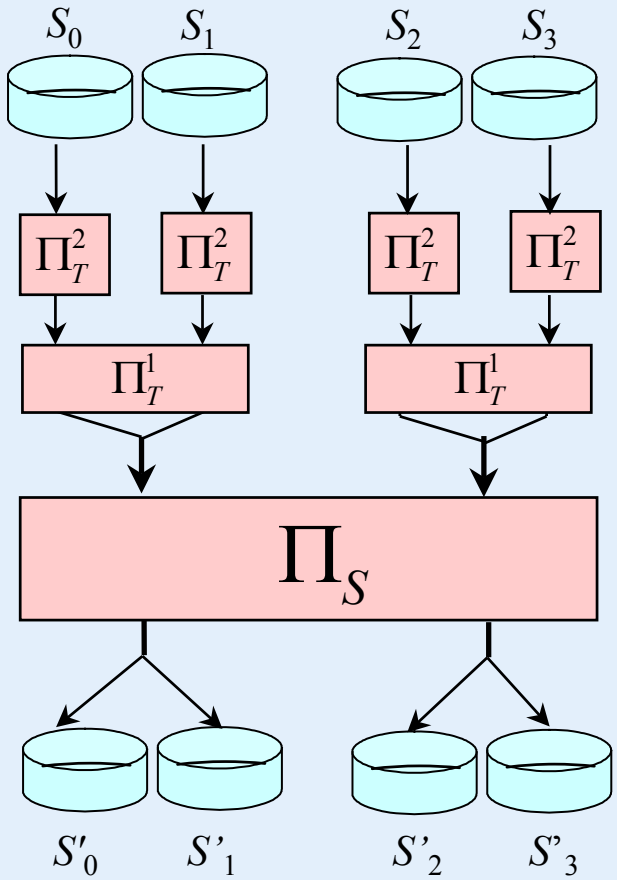
$P = 2$

Exemple d'entrelaceur contraint

- Masque d'entrelaceur => entrelaceur hiérachique

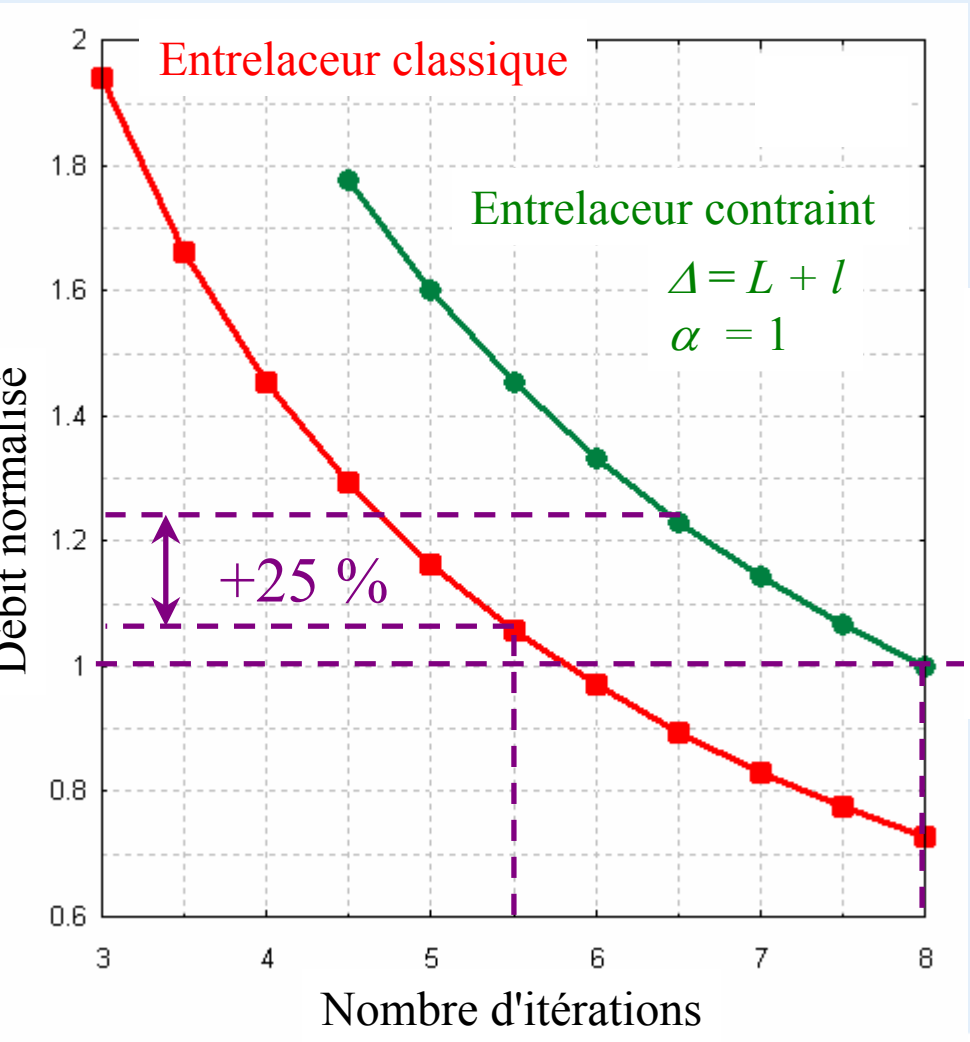


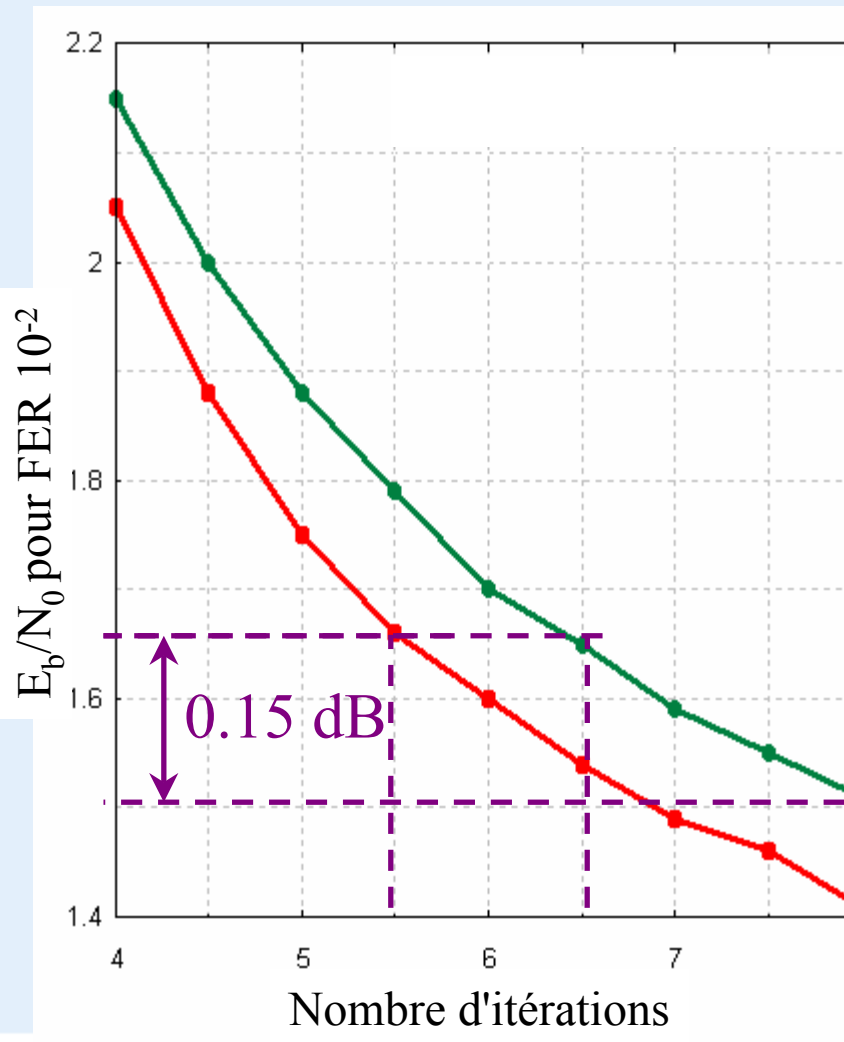
$P = 2$



Performances de l'entrelaceur contraint

- $N = 1024$ symboles, $P = 16$ processeurs (débit de 100 Mbit/s)





Conclusion exemple 1 : turbo-code

- Algorithme Turbo-code
- Architecture
- Constat de sous-utilisation de ressources matérielles
- Modification de l'exécution du matériel
- Retour vers la construction du code pour rendre cette modification acceptable
- Mesure des gains de l'approche

Plan

Turbo-code : Optimisation d'un entrelaceur/architecture

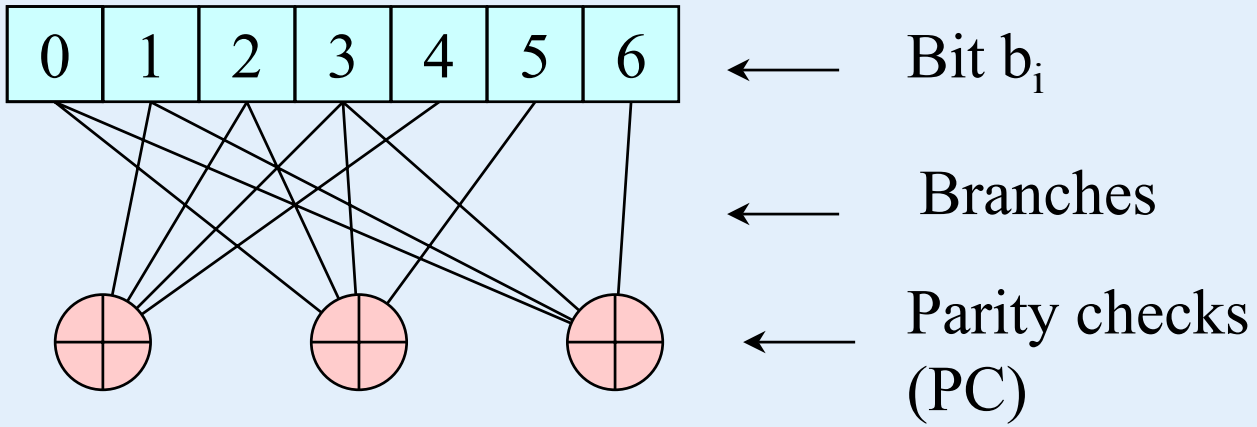
LDPC : Architecture pour "shuffled-scheduling"

- Rappel code LDPC
- Séquencement des calculs
- Changement de point de vue
- Architecture pour shuffle-BP

Code LDPC

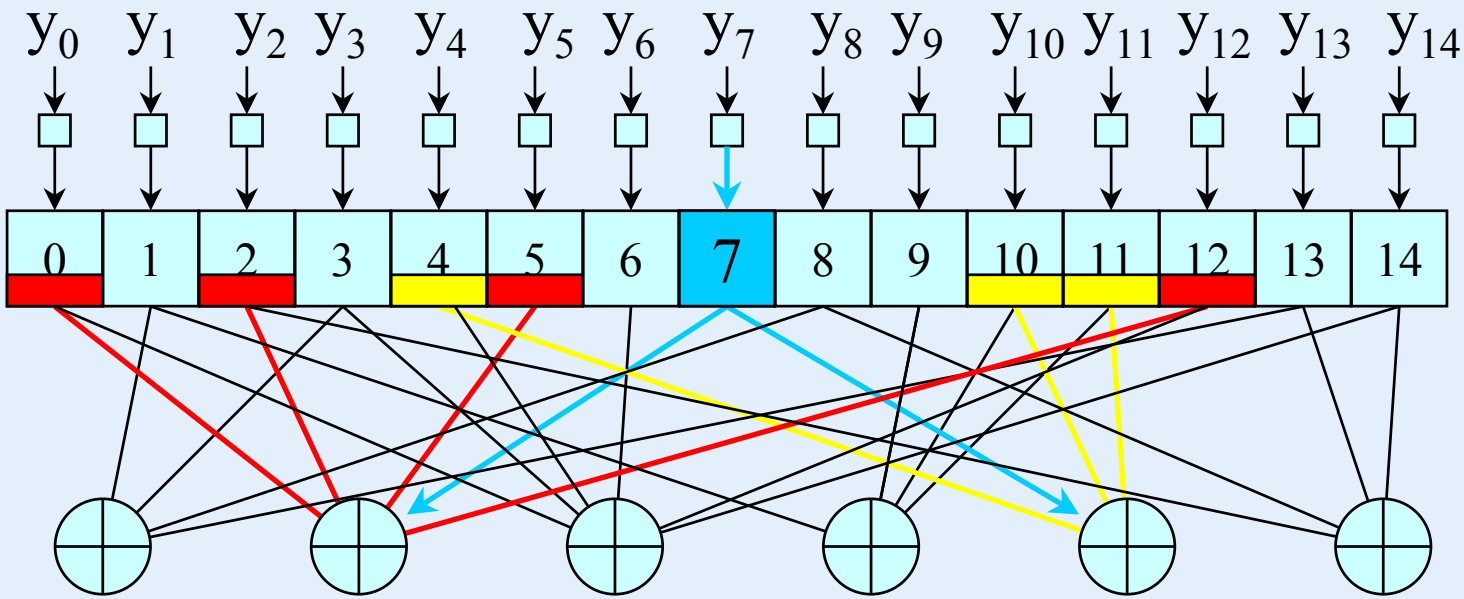
$$H = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Peut être défini par un graph bi-partite



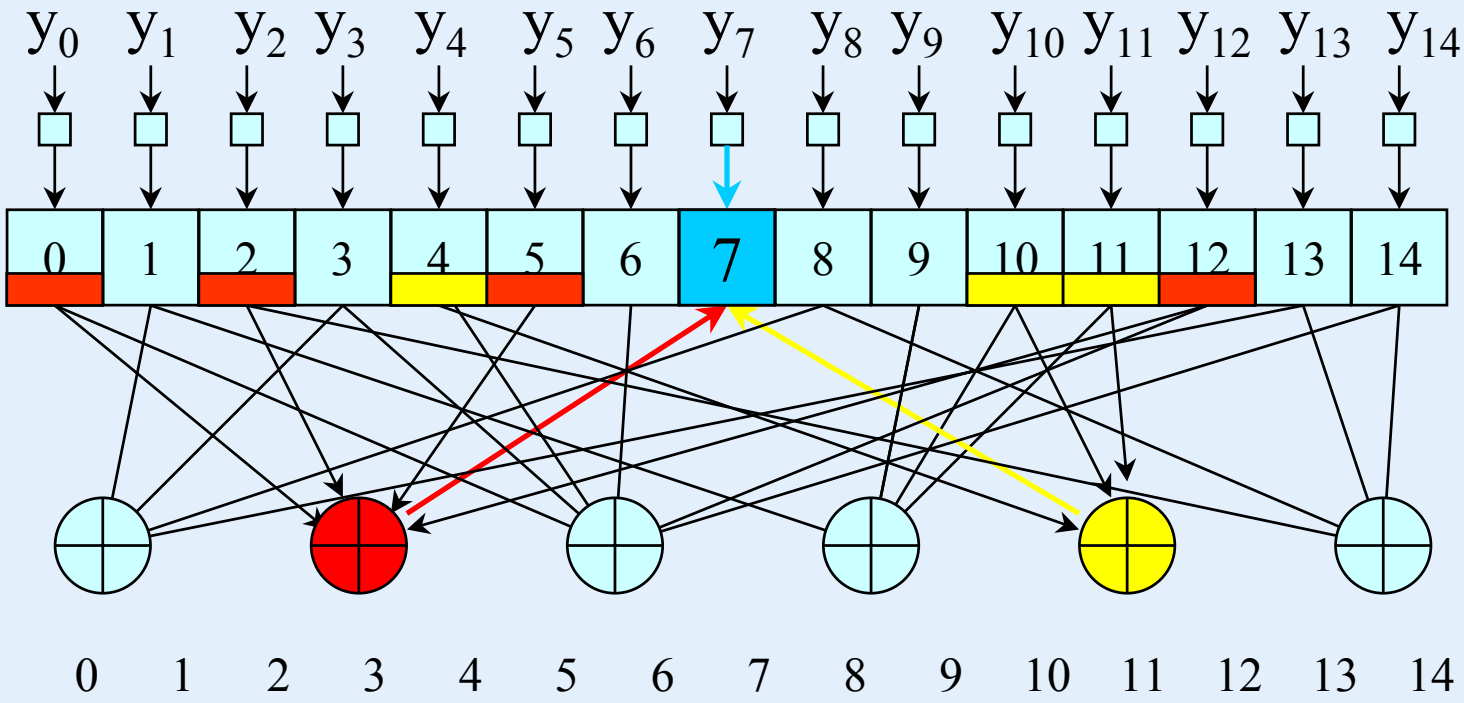
$(b_0, b_1, \ldots, b_6)$ mot de code <=> toutes les PC sont respectées

Décodage itératif



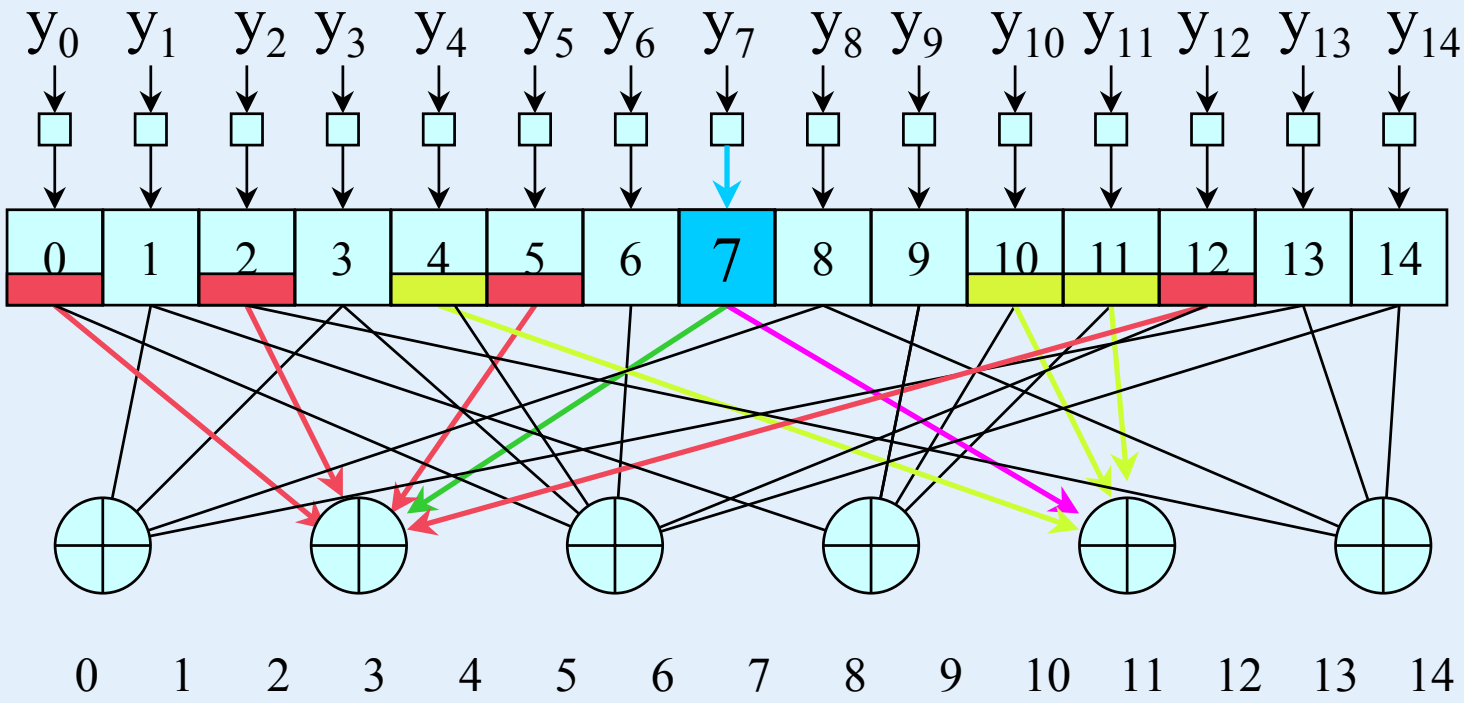
Première itération: message bit-> parité

Décodage itératif



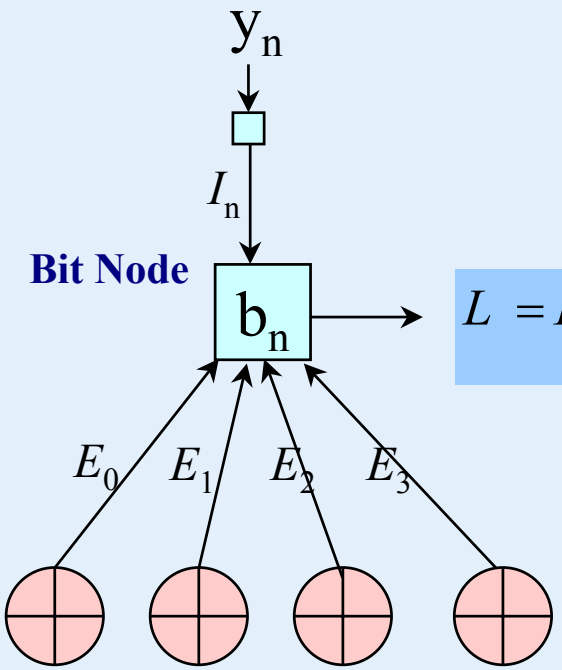
Première itération: message parité ⇨ bit

Décodage itératif

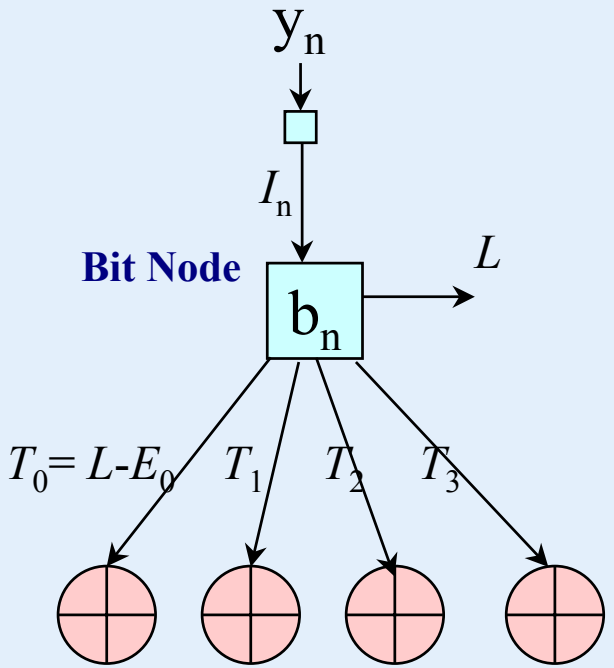


Deuxième itération: message parité -> bit

Calcul d'un nœud de variable (LLR)



$$L = I_n + \sum_m E_m$$



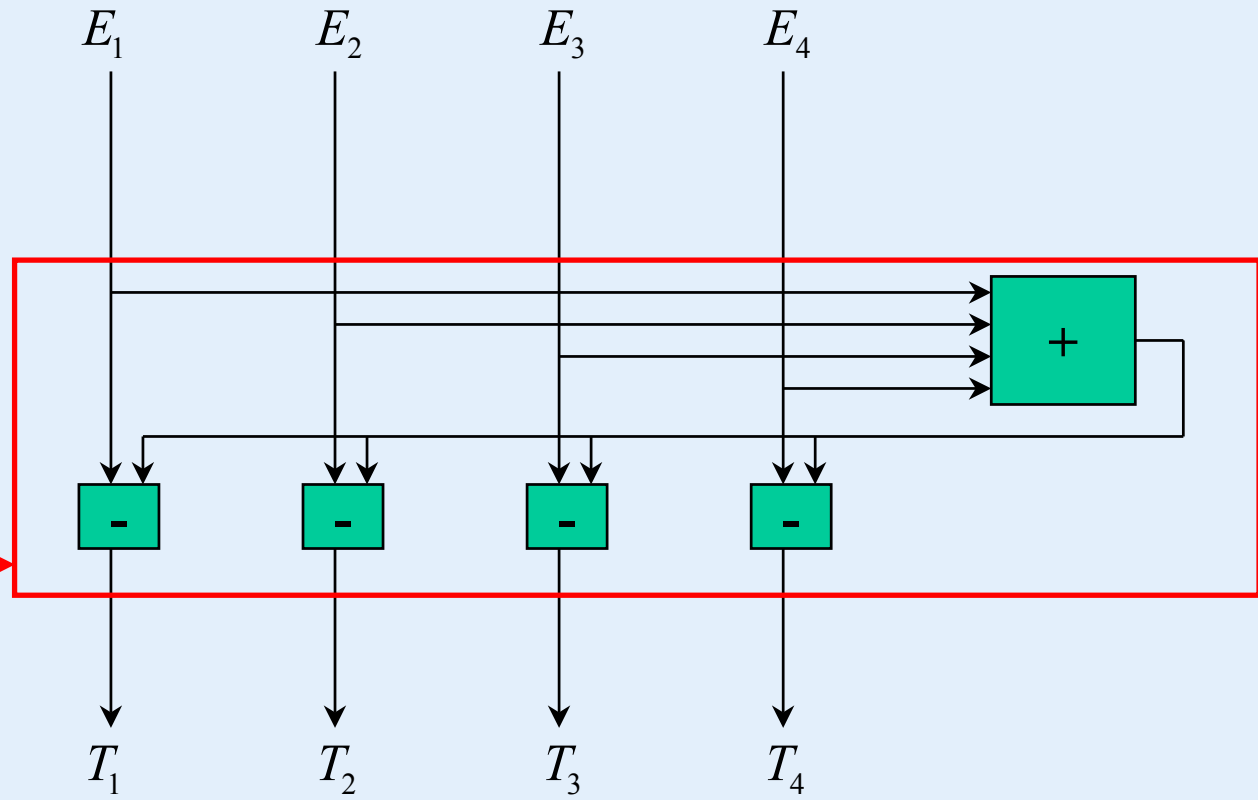
$$T_i = I_n + \sum_{m,m\neq i} E_i = L - E_i$$

Calcul d'un noeud

$$T_i = \sum_{k \neq i} E_k$$

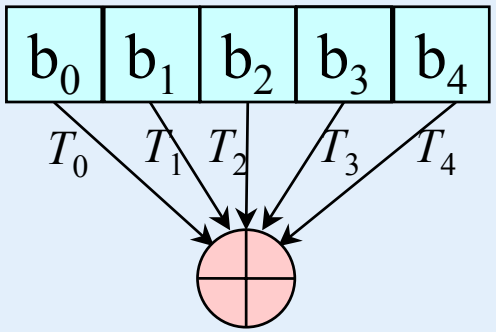
avec

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

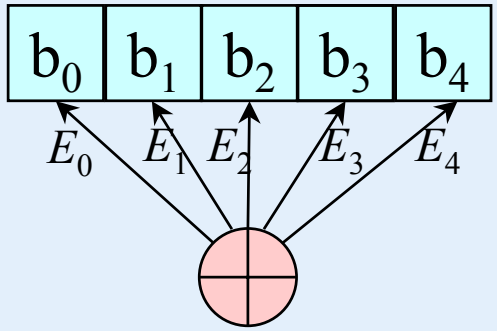


Generic Node Unit →

Calcul d'un nœud de parité



Check Node



Check Node

$$E_i = \bigoplus_{j \neq i} T_i = T_1 \oplus T_2 \ldots \oplus T_{i-1} \oplus T_{i+1} \ldots \oplus T_{dc}$$

Deux méthodes de calcul : directe ou fréquentielle

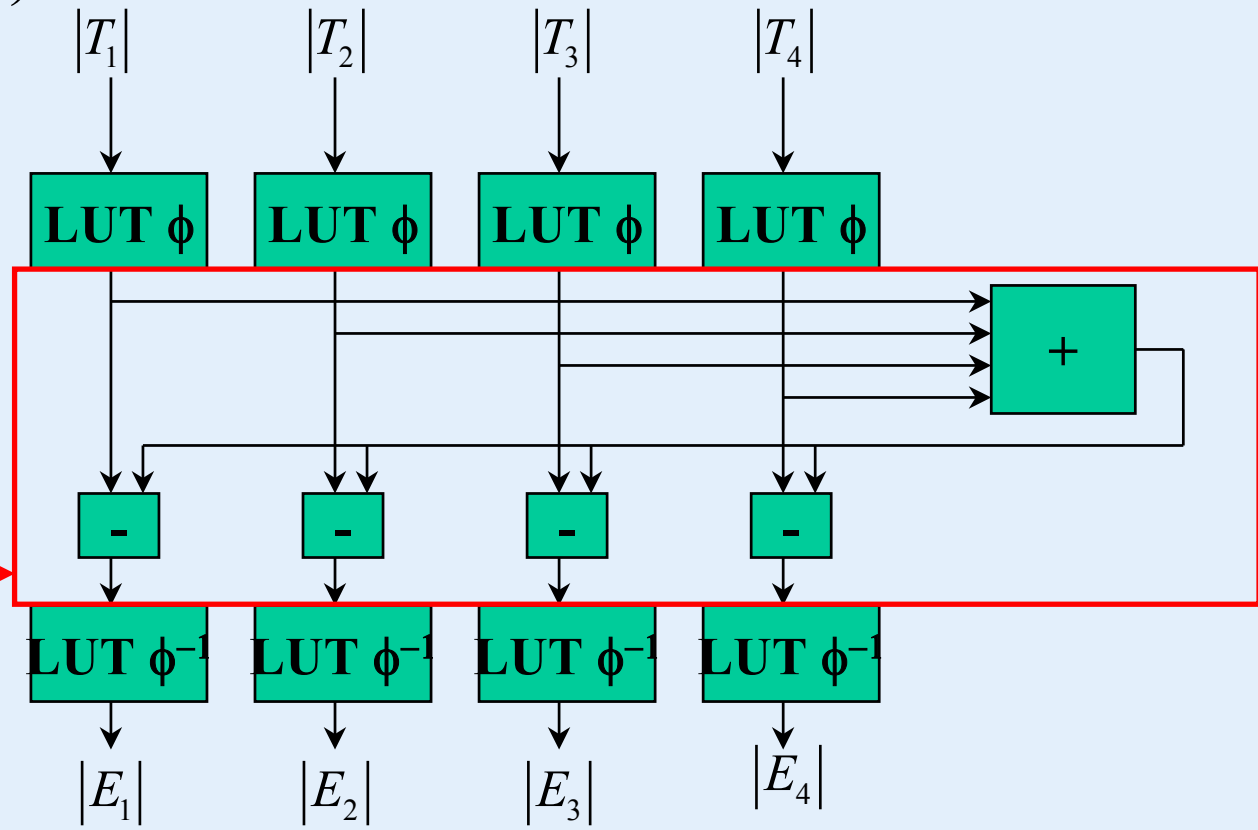
Calcul parité: méthode fréquentielle
b) calcul du module

$$|E_k| = \Phi^{-1}\left(\sum_{i \neq k} \Phi(|T_i|)\right)$$

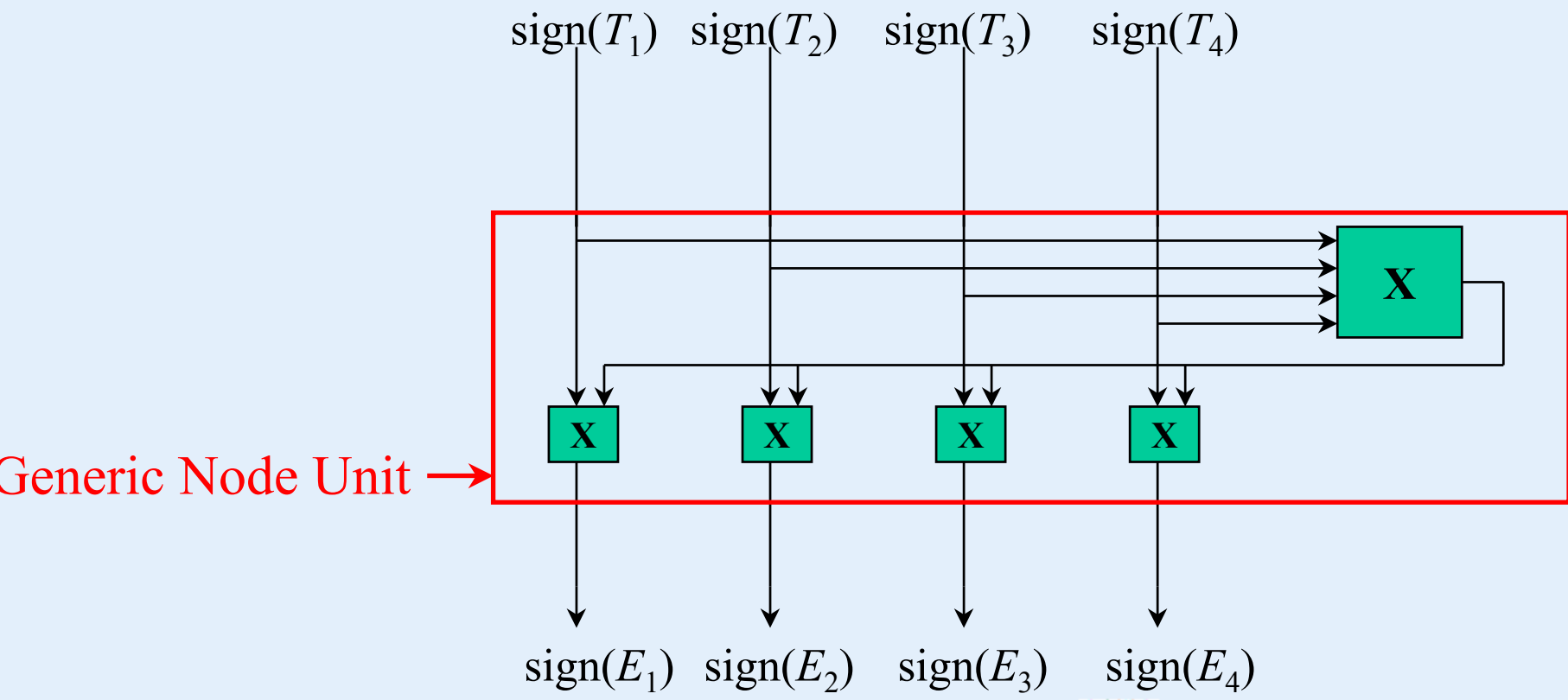
avec

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

Generic Node Unit →



Calcul parité: méthode fréquentielle
a) calcul du signe

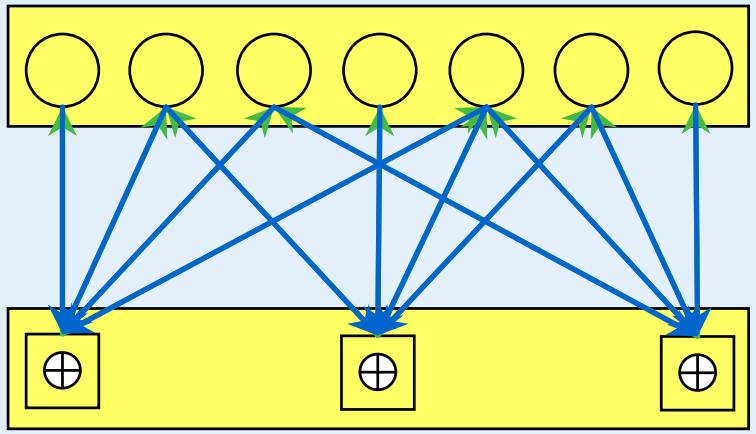


Algorithme BP

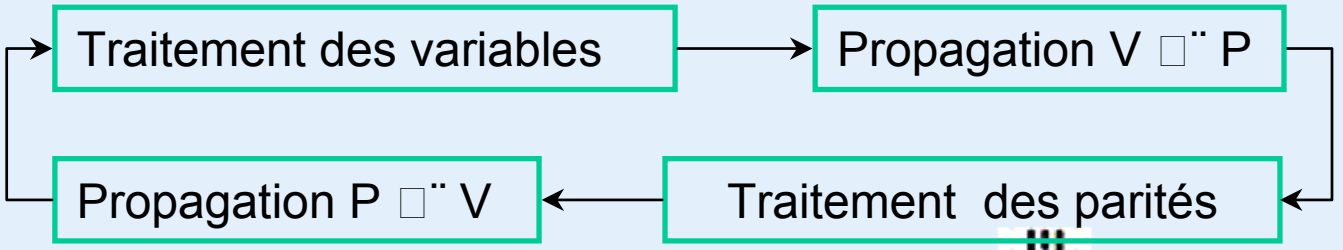
- Propagation de croyance (Belief Propagation)
- Initialisation :
  - $I_n = 2y_n/\acute{o}^2$
  - Extrinsèque branche : $E_{m,n}=0$
  - Fiabilités : $T_{m,n}=0$
- Itérations : tous les messages sont traités

$$T_{m,n} = L_n - E_{m,n}$$

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

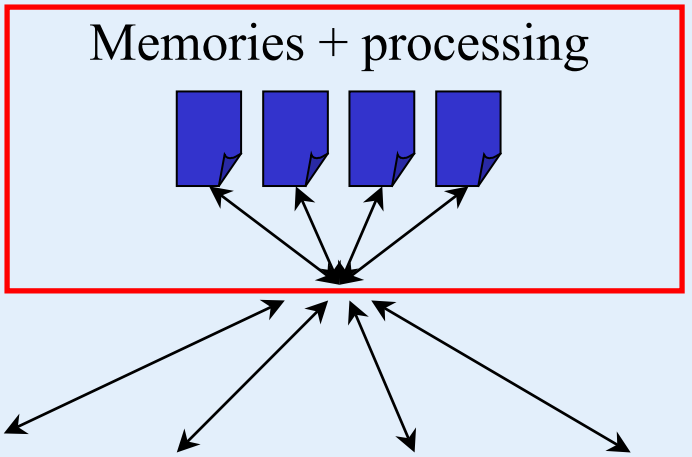


$$E_{m,n} = \Phi^{-1}\left(\sum_{n' \in N(m) \backslash n} \Phi(T_{m,n'}) \right)$$



Generic Node Units (exclude the Φ function)

Generic Node unit



- d entrée $\boldsymbol{e_m}$ et d sortie $\boldsymbol{s_n}$ (d=j or k)
- Relation entrées/sorties:

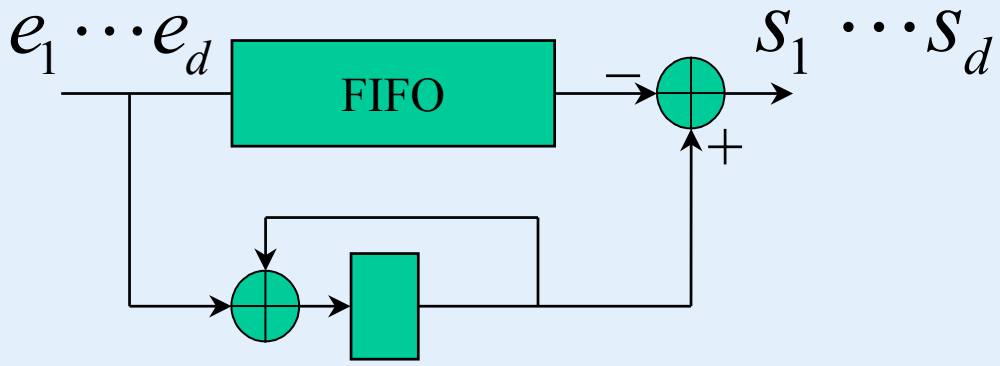
$$s_n = \sum_{m \neq n} e_m$$

- NB: remplacer Σ by × pour le traitement du signe

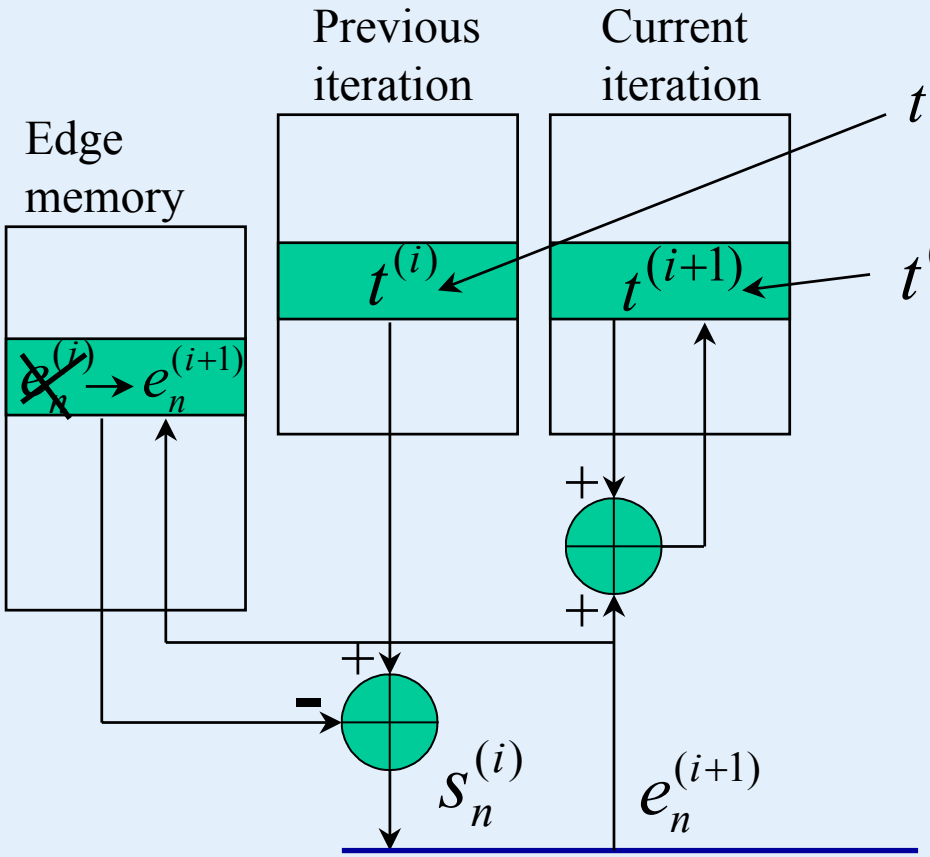
- Mode de réalisation possible du GNU ?

Implémentation du Generic Node Unit : a) Mode "compact" («traitement immédiat»)

- Exemple: implémentation série



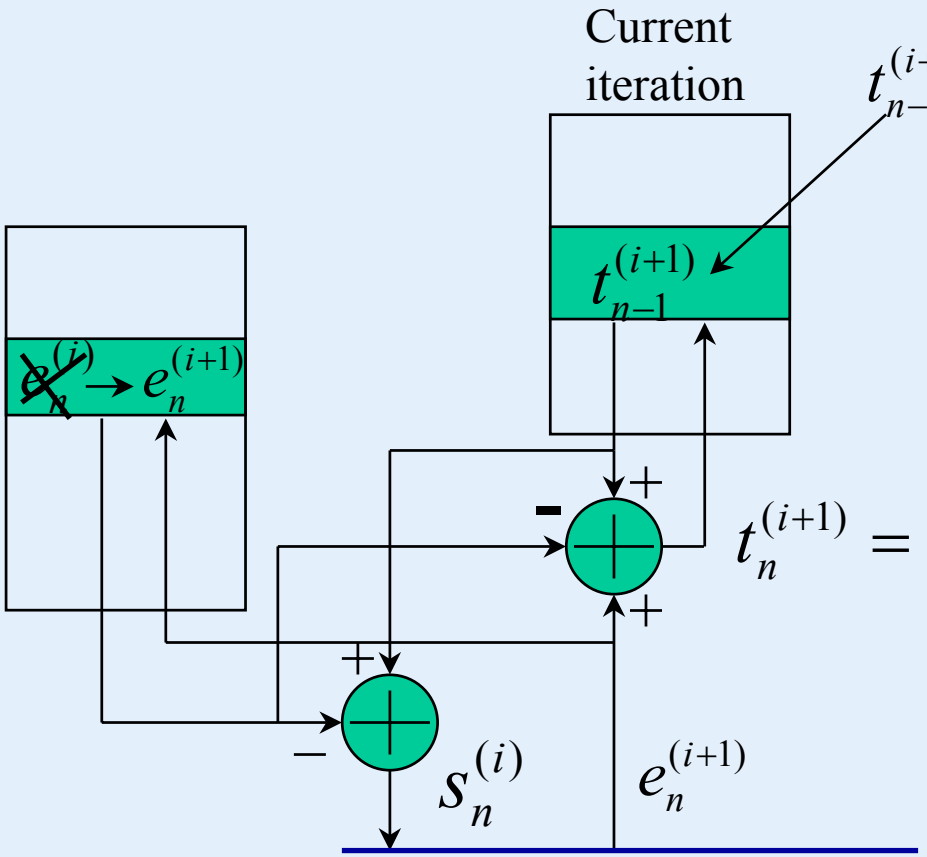
Implémentation du Generic Node Unit : b) Mode distribué, mis à jour différée



$$t^{(i)} = e_1^{(i)} + e_2^{(i)} + \ldots + e_d^{(i)}$$

$$t^{(i+1)} = e_1^{(i+1)} + \ldots + e_n^{(i+1)}$$

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



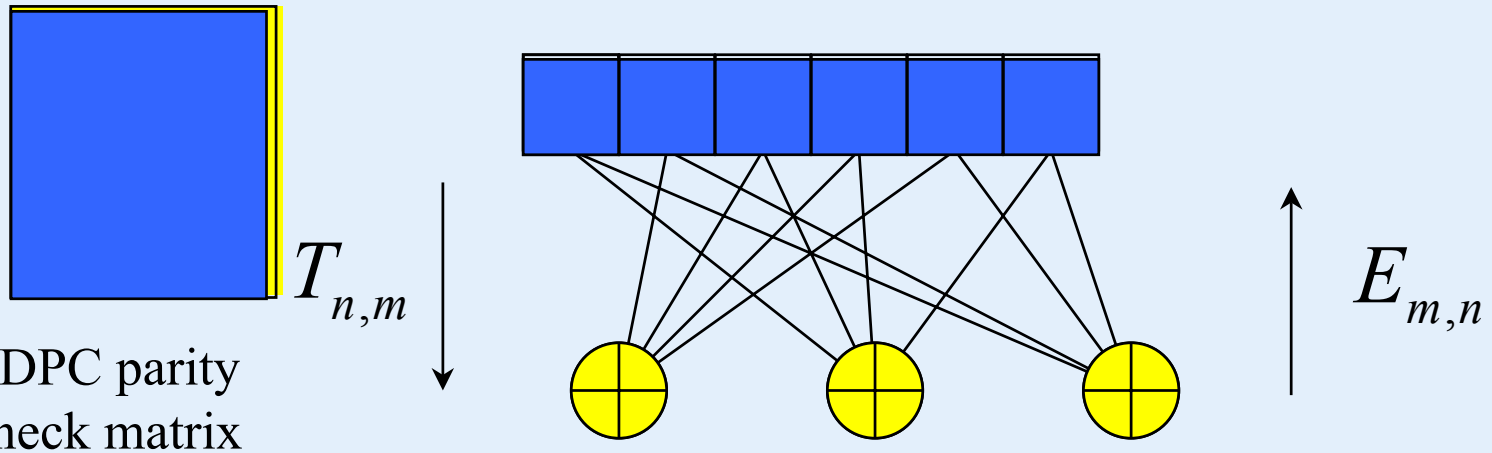
Plan

Turbo-code : Optimisation d'un entrelaceur/architecture

LDPC : Architecture pour "shuffled-scheduling"

- Rappel code LDPC
- Séquencement des calculs
- Changement de point de vue
- Architecture pour shuffle-BP

Flooding scheduling (classical one)

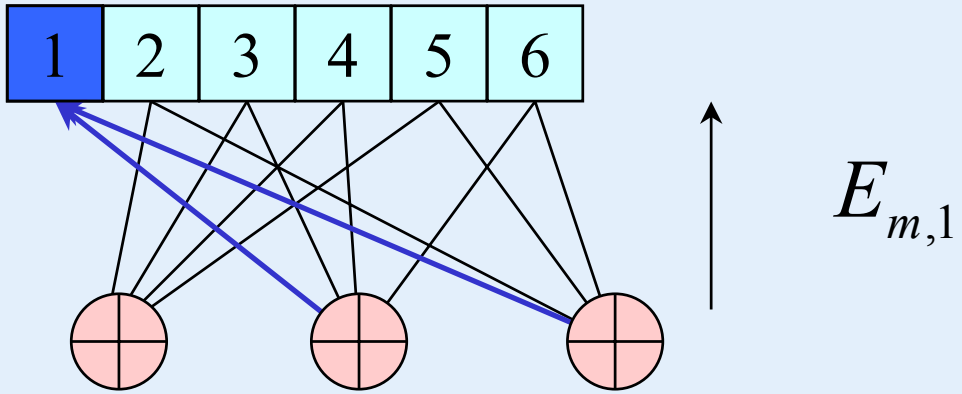


LDPC parity check matrix

Vertical Shuffle Scheduling (Shuffle BP's Zhang-Fossorier02)

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

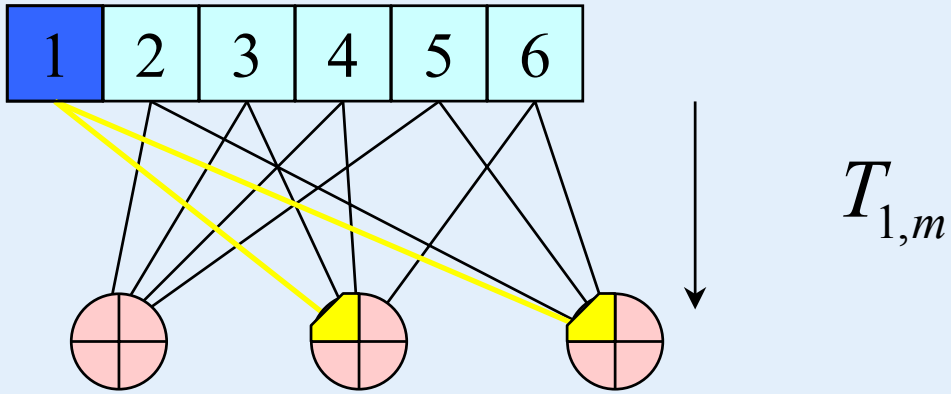
LDPC parity check matrix



Vertical Shuffle Scheduling (Shuffle BP's Zhang-Fossorier02)

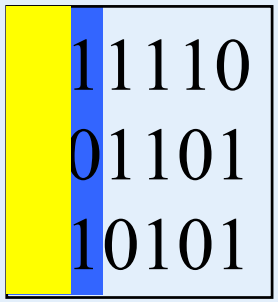
11110
01101
10101

LDPC parity check matrix

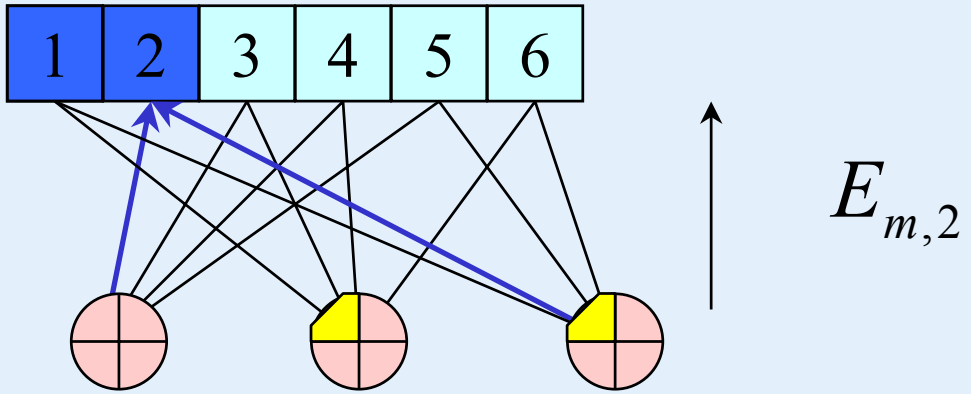


$T_{1,m}$

Vertical Shuffle Scheduling (Shuffle BP's Zhang-Fossorier02)



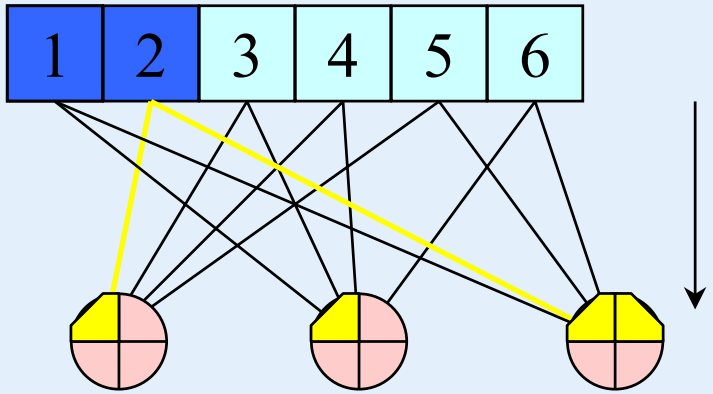
LDPC parity check matrix



Vertical Shuffle Scheduling (Shuffle BP‘s Zhang-Fossorier02)

1110
1101
0101

LDPC parity check matrix



et ainsi de suite…

$T_{2,m}$

L'algorithme converge 2 fois plus rapidement que le séquencement par innondation classique

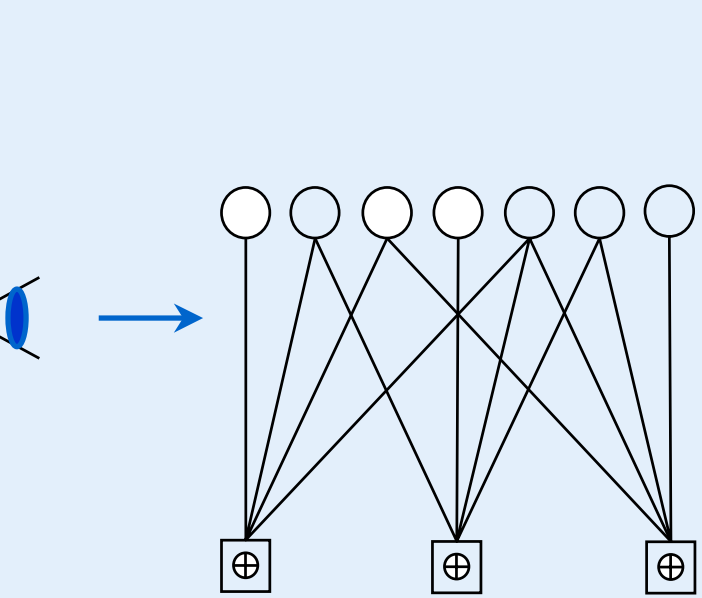
Plan

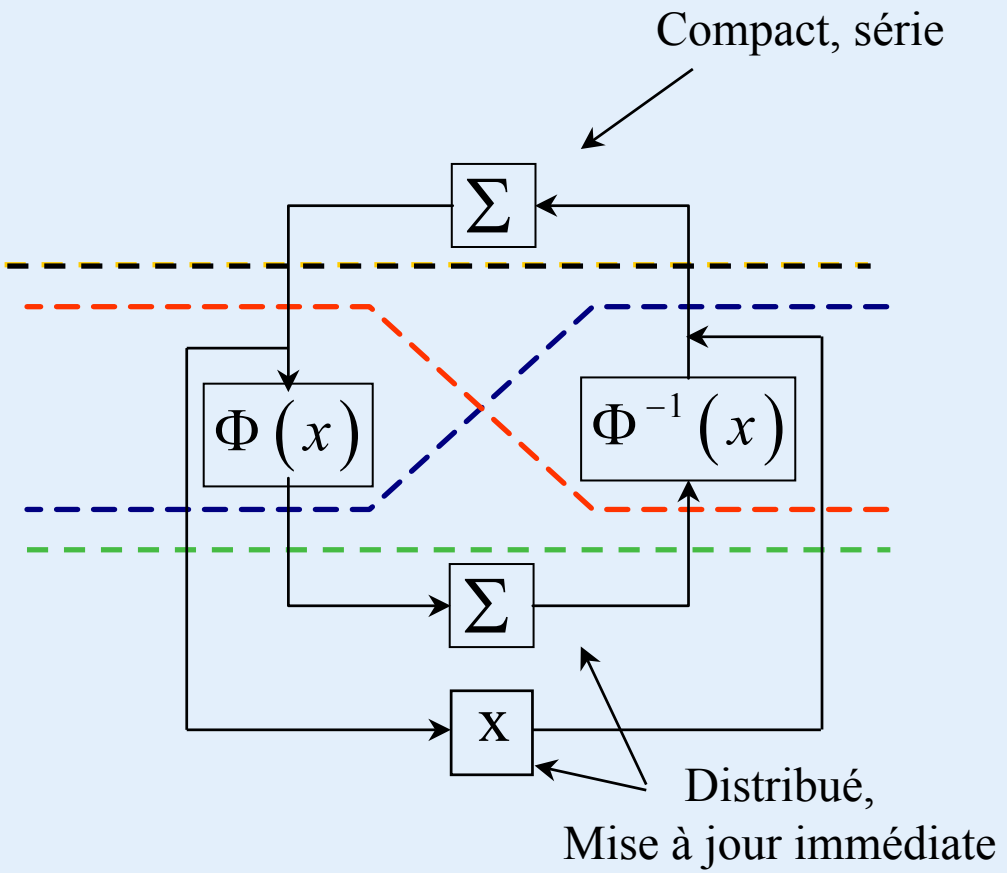
Turbo-code : Optimisation d'un entrelaceur/architecture

LDPC : Architecture pour "shuffled-scheduling"

- Rappel code LDPC
- Séquencement des calculs
- Changement de point de vue
- Architecture pour shuffle-BP

Implémentation/séquencement





Permet une architecture très efficace pour le shuffled BP

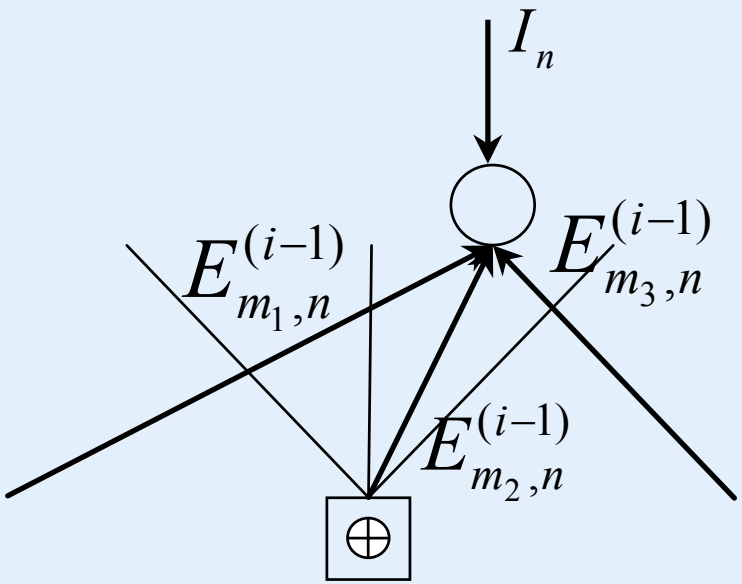
Plan

Turbo-code : Optimisation d'un entrelaceur/architecture

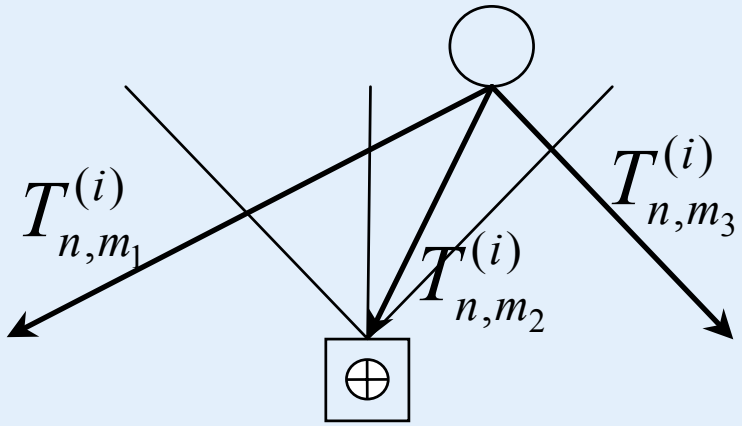
LDPC : Architecture pour "shuffled-scheduling"

- Rappel code LDPC
- Séquencement des calculs
- Changement de point de vue
- Architecture pour shuffle-BP

VNU : mode compact, série



3 cycles

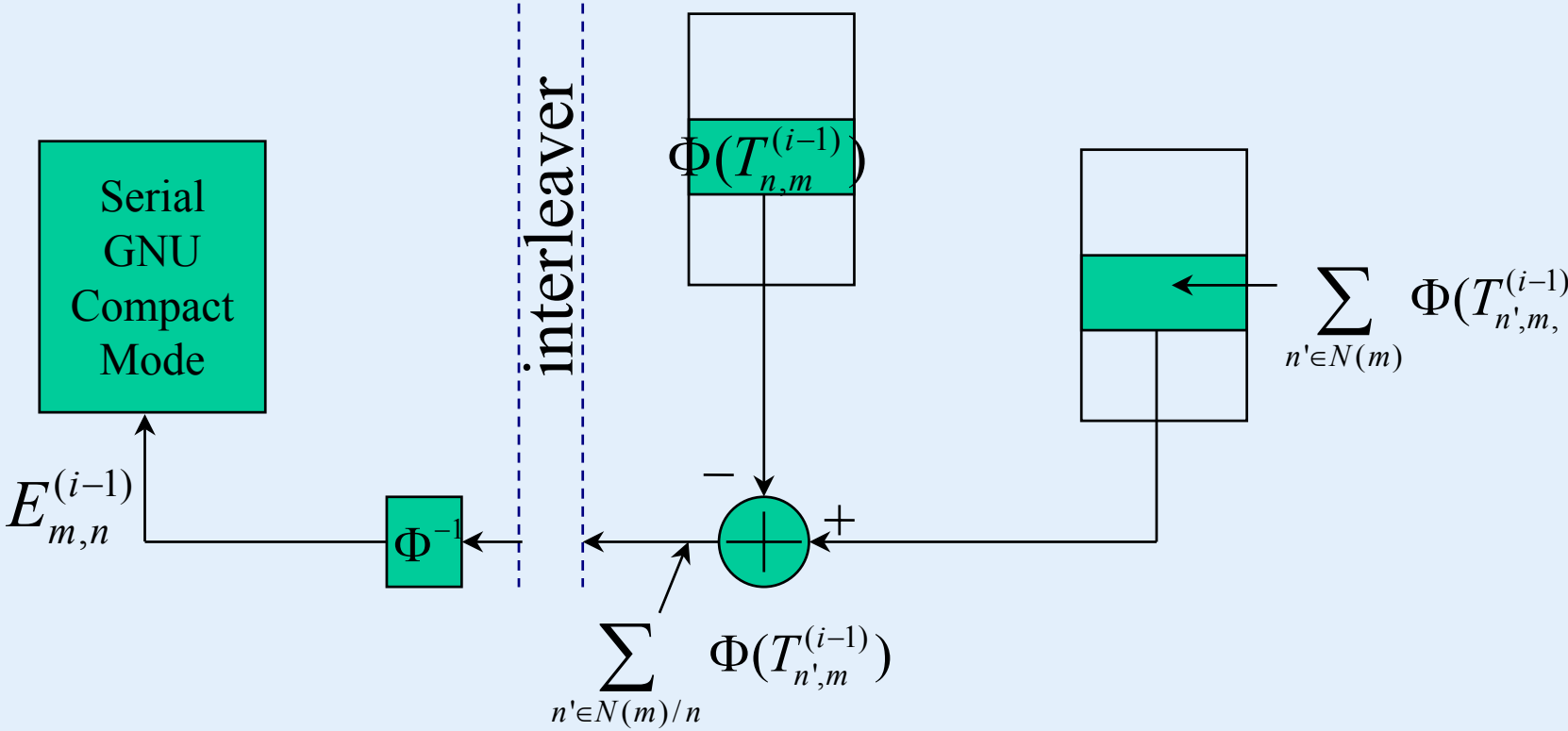


3 cycles

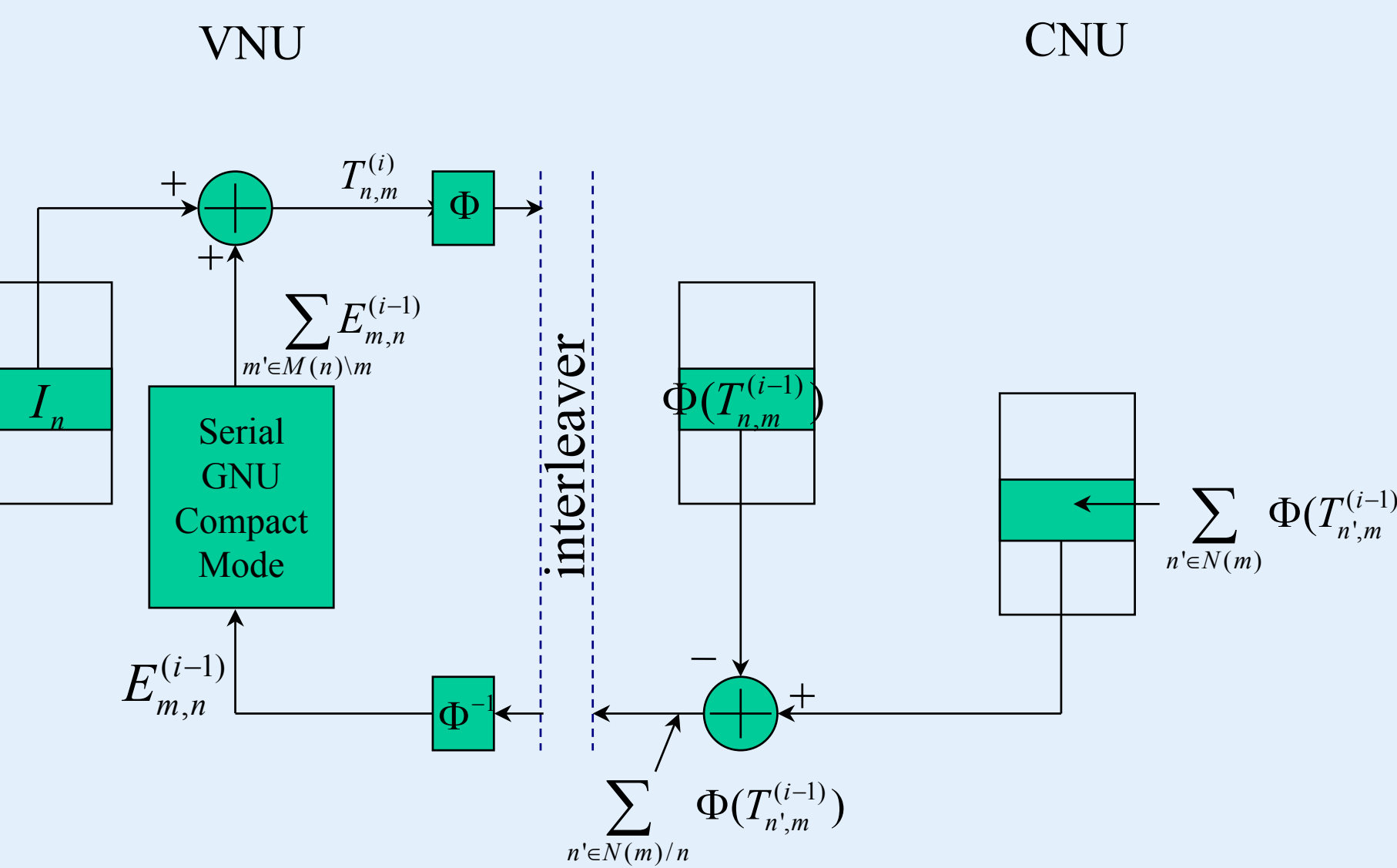
VNU

CNU

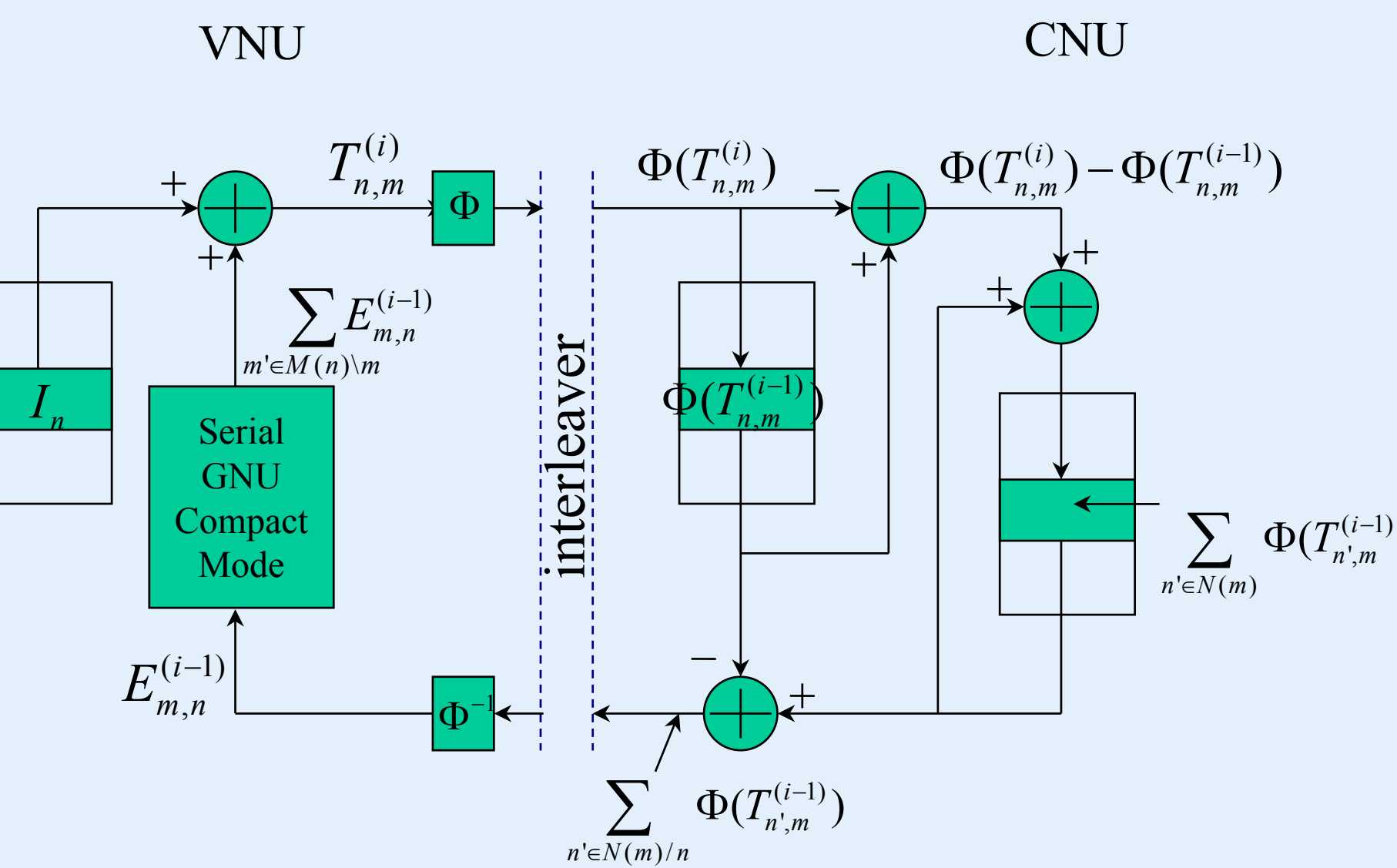
$$E_{m,n} = \Phi^{-1}\left(\sum_{n' \in N(m) \setminus n} \Phi(T_{n',m}) \right)$$



Tous les CNU reliés au VNU envoient leur message



Le VNU calcule ses sorties et les envoies aux CNU



$\sum_{n' \in N(m)} \Phi(T_{m,n'}^{(i-1)})$ est partiellement mis à jour.

Conclusion exemple 2 : LDPC

- Analyse du décodeur itératif
- Modes de réalisation des "processeurs génériques"
- Cycle de calcul dans le graphe du code
  - Changement du point de vue : nouvelle reformulation de l'algorithme
- Architecture "Vertical Shuffle BP"
  - Architecture simple
  - Convergence deux fois plus rapide que séquencement classique
- Retrouve tous les séquencements publiés

Conclusion générale

- Esquisse d'une "méthode" d'optimisation AAA
  - Partir de l'architecture en oubliant les a priori
  - Plus de liberté de penser et d'innover
  - Sélectionner/valider/modifier

- Présentation de deux exemples :
  - Entrelaceur Turbo-Code
  - Séquencement LDPC

- …arriver à sortir de nos a priori…