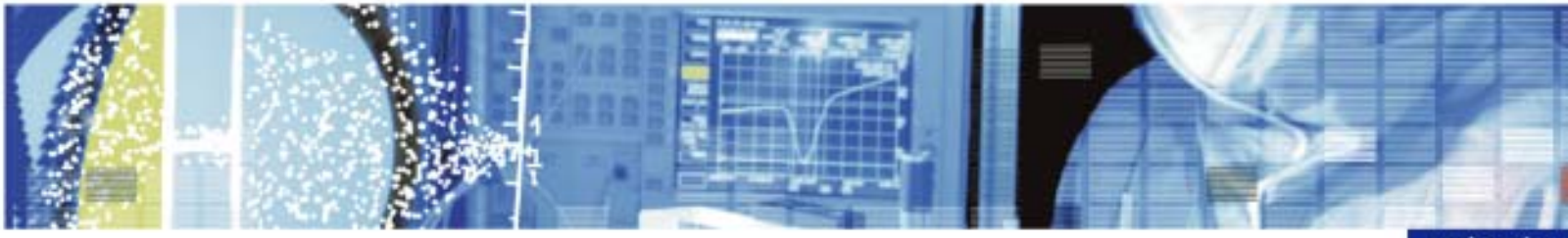
École Nationale Supérieure des Télécommunications de Bretagne

ENST Bretagne



Architectures for LDPC Decoders

Frédéric Guilloud, ENST Bretagne

Emmanuel Boutillon, University of South Britany

June 12 - 15

2005 IEEE Communication Theory Workshop

UBS
UNIVERSITÉ DE BRETAGNE-SUD

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

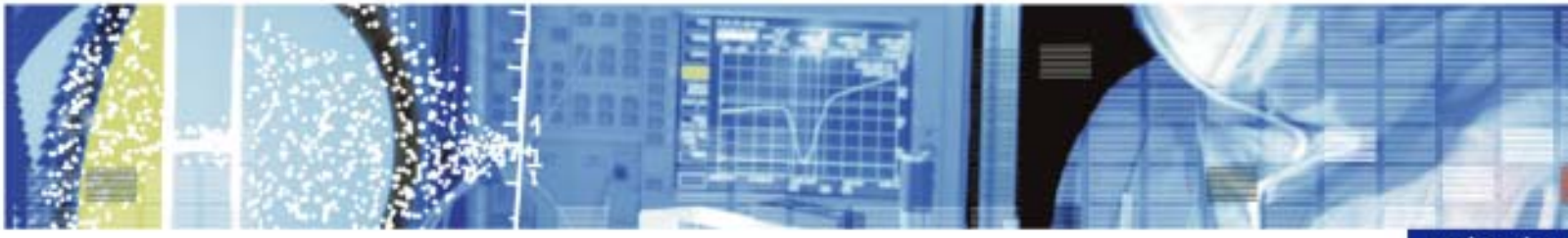
get

Contents

- Part 1 : Main Bottlenecks in LDPC Decoder Design
  - Design flow
  - Implementation costs

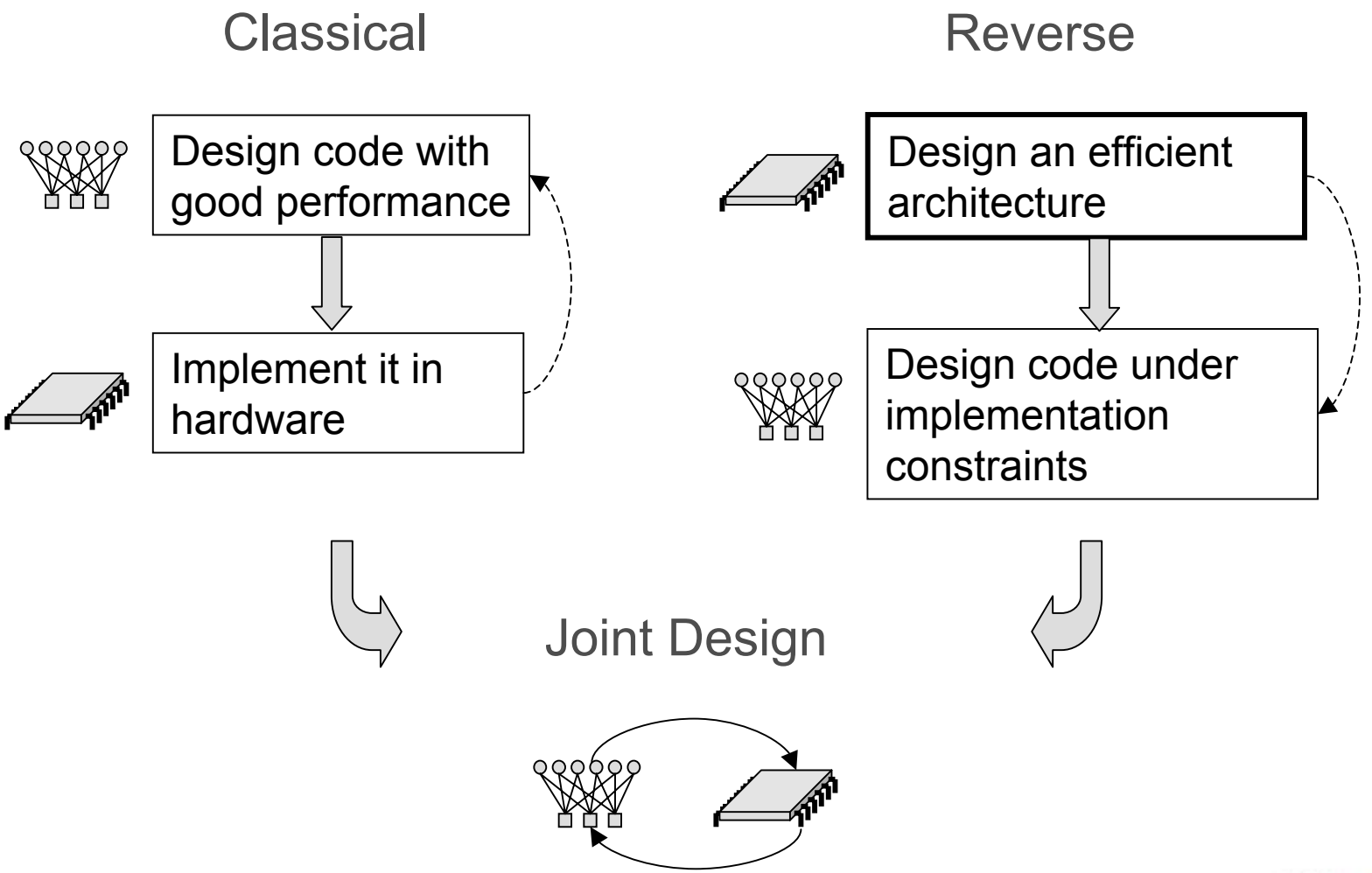
- Part 2 : Case study

  Ad hoc implementation for shuffle-BP scheduling
  - Decoding Schedules
  - Generic Node Units
  - Architectural study



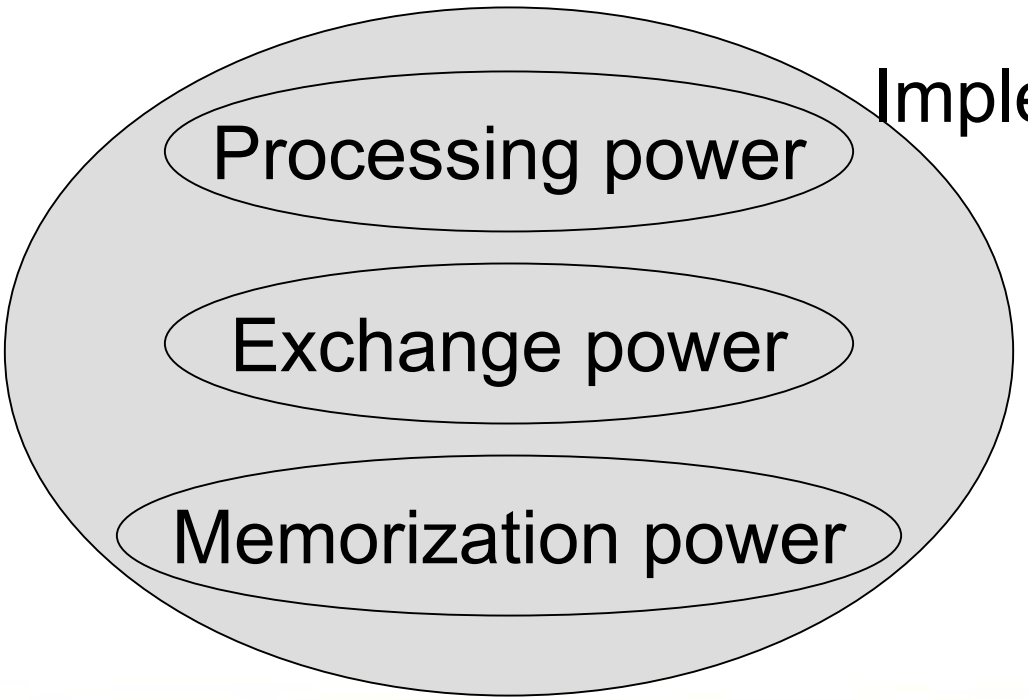
Part 1 : Main bottlenecks in LDPC decoders designs

Design Flows



Implementation costs

- An efficient LDPC code is a good thing …
- The opportunity to implement it in a **small area** chip is even better !



Implementation costs

Processing power (1/2)

- Check update:

  For all edges (m,n) connecting the m^{th} check to the n^{th} variable: 2 additions and 2 LUT (f) per edges.

$$f(x) = -\ln\left(\tanh\left(x/2\right)\right) \qquad E_{m,n} = 2\tanh^{-1}\prod_{n'\in N(m)\setminus n}\tanh\left(\frac{1}{2}T_{n',m}\right) = \mathrm{f}^{-1}\left(\sum_{n'\in N(m)\setminus n}\mathrm{f}\left(T_{n',m}\right)\right)$$

- Variable update:

  For all edges (n,m) connecting the n^{th} variable to the m^{th} check : 2 additions per edges.

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

- NB : We define also $f^{-1}(R_{m,n}) = E_{m,n}$ and $Q_{n,m} = f(T_{n,m})$

- Complexity proportionnal to the number of edges

Processing power (2/2)

- Specifications:
  - Information size: K bits
  - Information throughput: D_b bit/s
  - The maximum number of iterations: $i_{\max}$
  - The clock frequency: f_{clk}

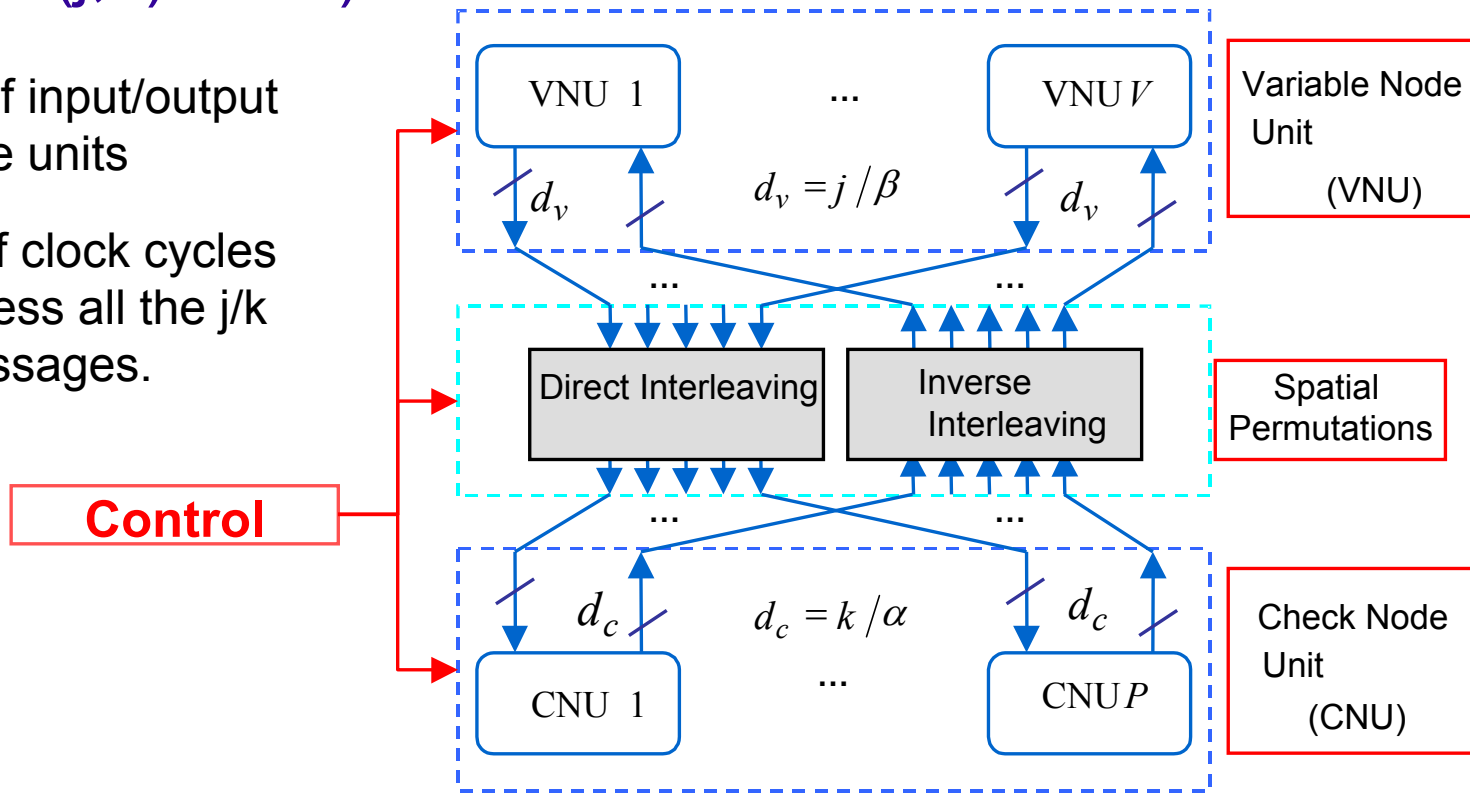
- Number of edges to be processed:

$$R_e = \frac{E i_{\max} D_b}{K f_{\text{clk}}} \text{ edges/clock cycle}$$

E being the number of non zero entries in the matrix

Generic Message Passing Architecture (Regular (j,k) code)

- d_v, d_c: number of input/output ports of the node units
- α, β: number of clock cycles required to process all the j/k node output messages.



Number of edges to be processed: $R_e = P\frac{k}{\alpha}$ edges/clock cycle

so $P=Re/k$ check nodes per clock cycle

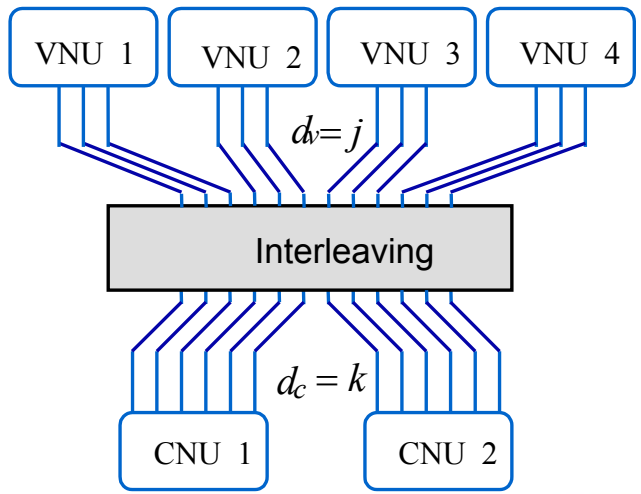
(if α=1)

Design example

- Assume these specifications:
  - Information size: 100 bits
  - Information throughput: 100Mbit/s
  - The maximum number of iterations: $i_{\max}=20$
  - The clock frequency: $f_{\text{clk}}=$ 1GHz
  - Regular (3,6) LDPC code with rate R=0.5

$$R_e = \frac{Ei_{\max}D_b}{Kf_{\text{clk}}} = \frac{200\times 3\times 20\times 10}{100\times 10^9}$$

$$R_e = 12 \text{ edges/clock cycle}$$

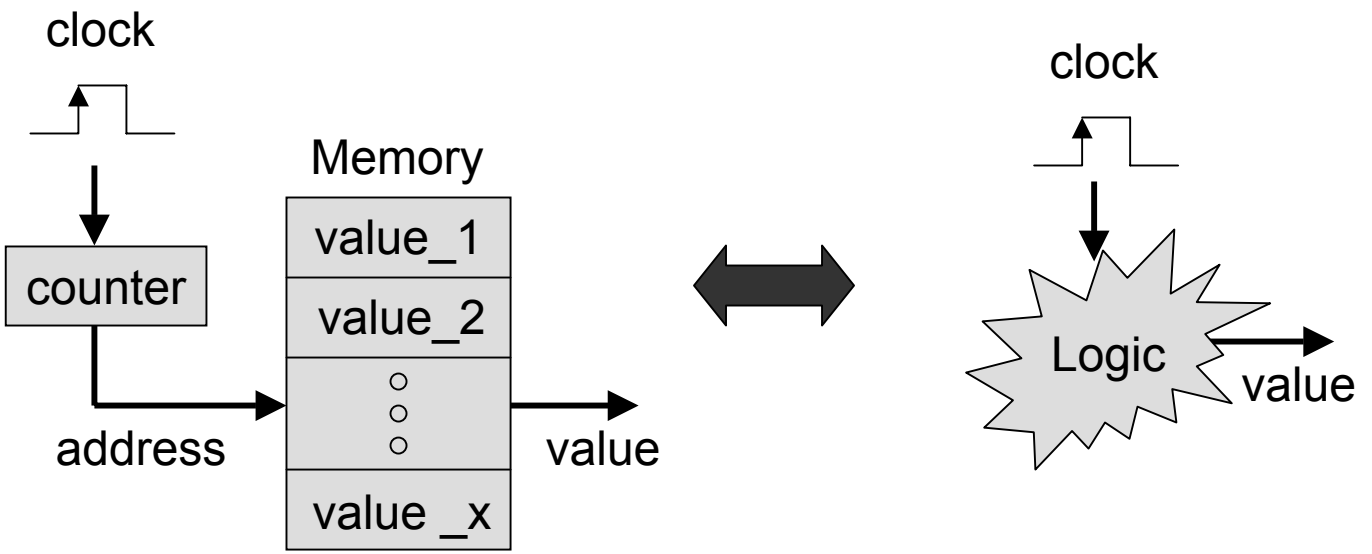


- Number of edges to be processed (if $\alpha=1$):

$P=Re/k=12/6=2$ check nodes per clock cycle

Memorization power

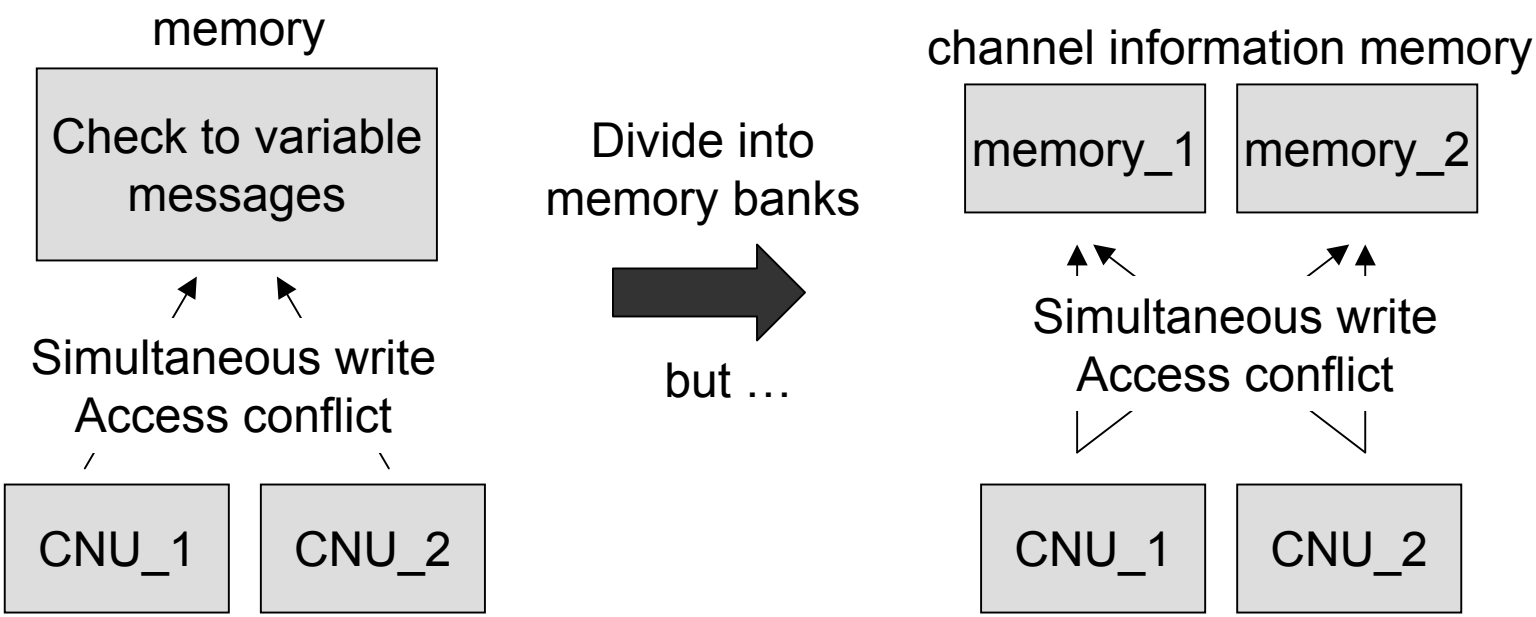
- The parity check matrix has to be saved inside the chip.
  - Exchange memory against processing (addresses processing)



(CNU = Check Node Unit)

Exchange power (1/2)

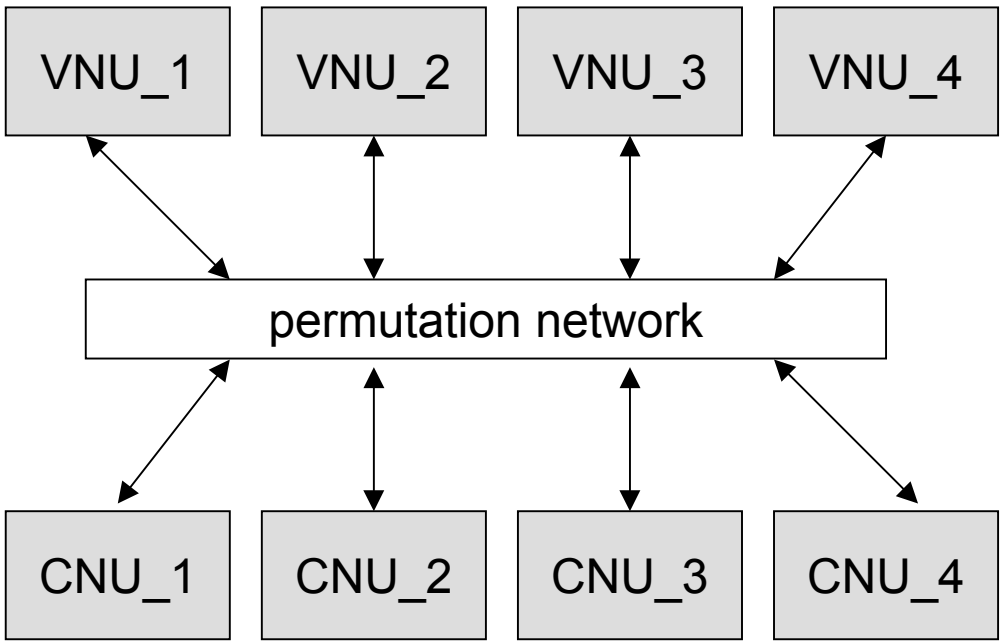
- Memory conflicts



(VNU = Variable Node Unit)
(CNU = Check Node Unit)

Exchange power (2/2)

- Routing conflicts



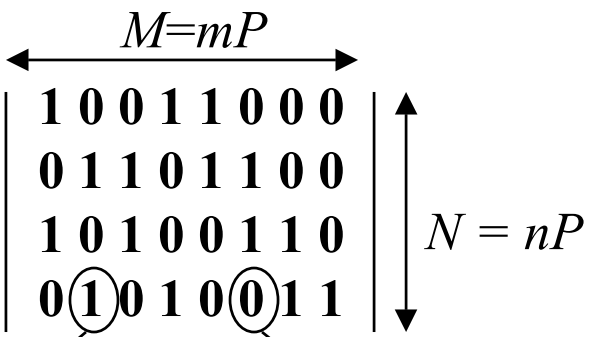
Simple permutation networks :

Rotations,
Protographs, …

Interleaver constraint: rotations (1/2)

[Sridhara, Fuja, Tanner 2001]
[Hocevar 2003]

Prototype Matrix: H =



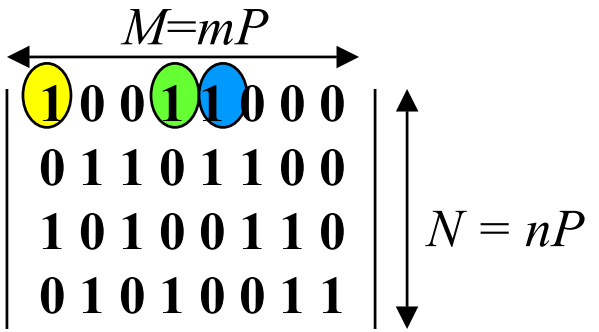
$$\Pi_{i,j}(I_p) = \begin{vmatrix} 0&0&1&0&0&0 \\ 0&0&0&1&0&0 \\ 0&0&0&0&1&0 \\ 0&0&0&0&0&1 \\ 1&0&0&0&0&0 \\ 0&1&0&0&0&0 \end{vmatrix}_{P \times P}$$

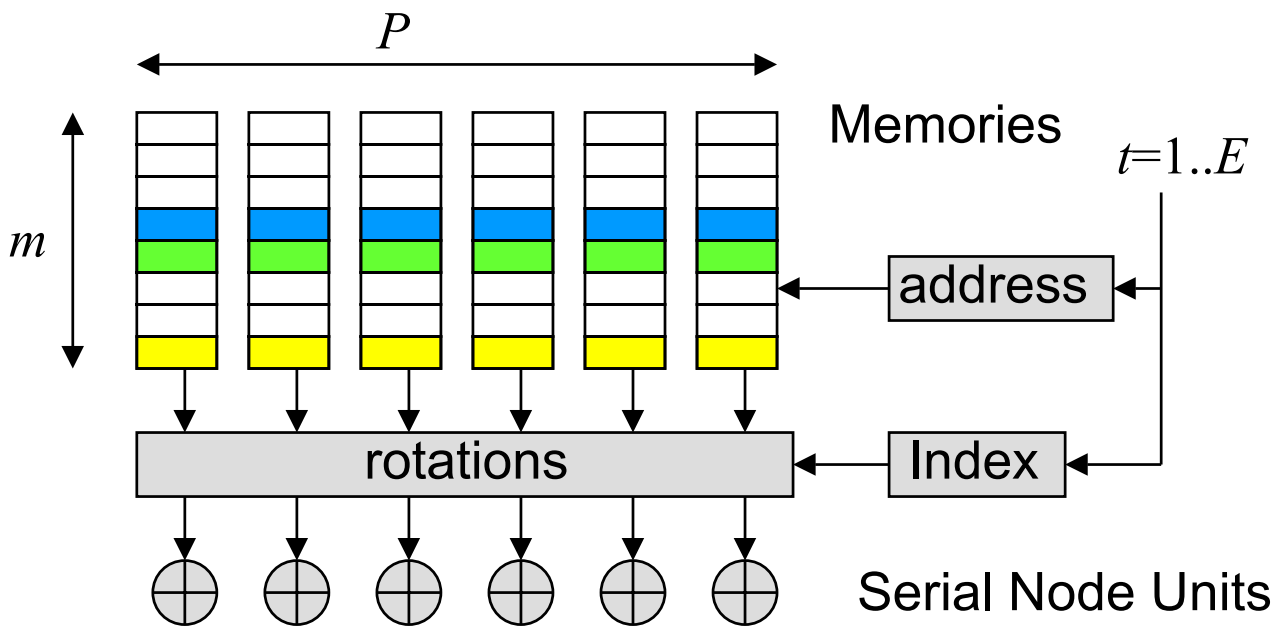
$$\begin{vmatrix} 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \end{vmatrix}$$

Moreover : Small memory requirements for parity check matrix

Interleaver constraint: rotations (2/2)

Prototype Matrix: H =





Conclusion : LDPC decoder design from the architecture point of view

Code performance

- Code itself ☺
- Decoding algorithm
  - CNU
    BP,BP-Based, λ-min, A-min*
  - VNU
  - Precision
  - Scheduling
    Flooding, shuffled

High throughput

- Improve Hardware Utilization efficiency:

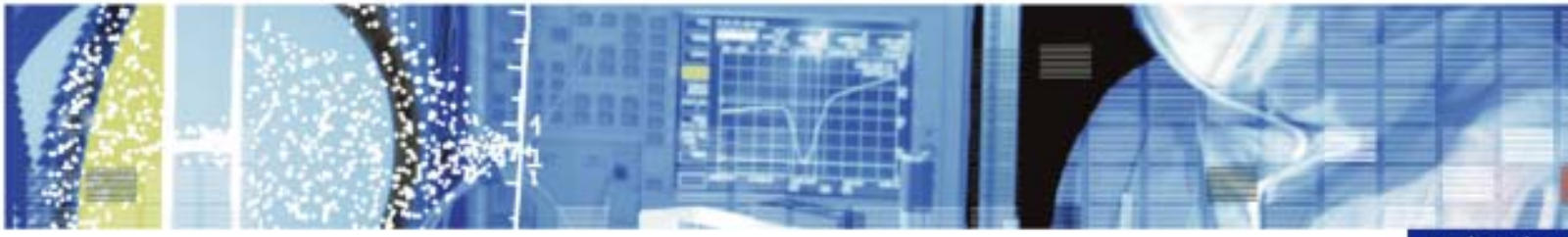
Q_{hard} x activity rate / processing x clock frequency

Low cost

- Node unit complexity
- Structured parity check matrix

Versatility

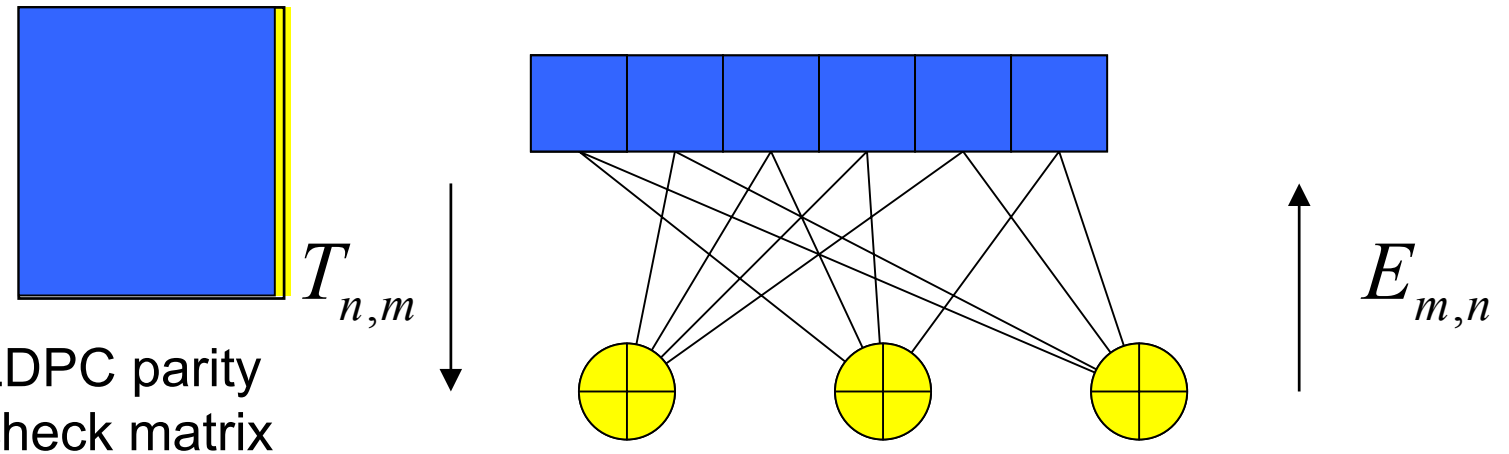
- Size
- Rate



Part 2 : Case study

Scheduling implementation

Flooding scheduling (classical one)

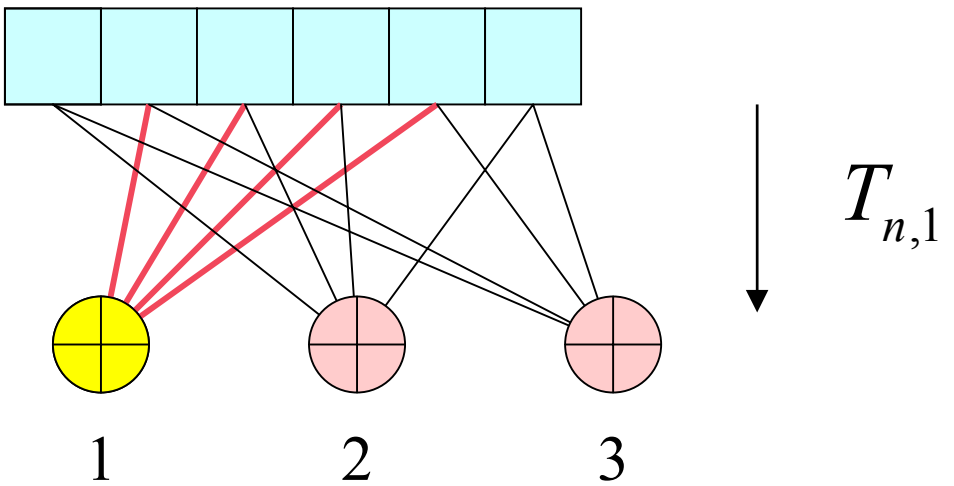


LDPC parity check matrix

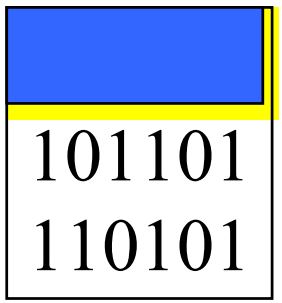
Horizontal Shuffle Scheduling (Turbo Decoding ‘s Mansour02)

011110
101101
110101

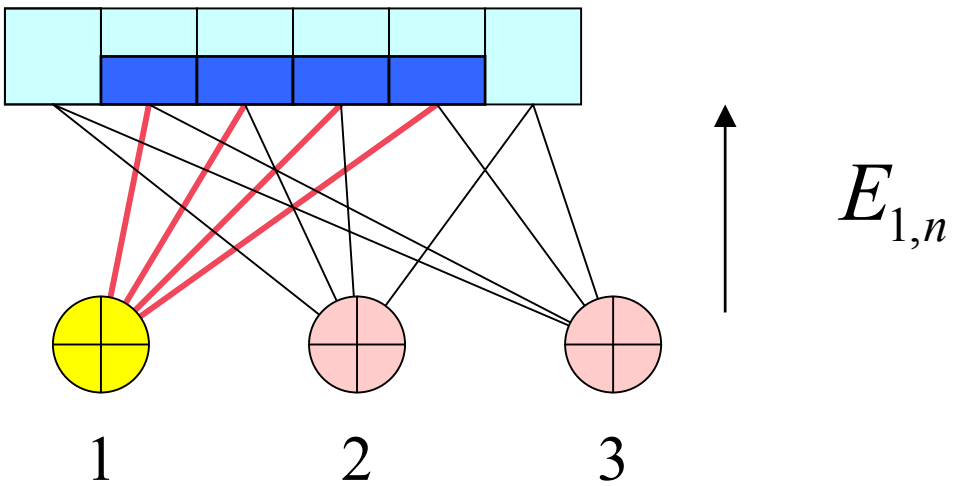
LDPC parity check matrix



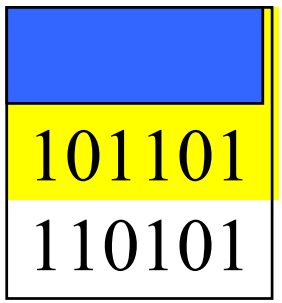
Horizontal Shuffle Scheduling (Turbo Decoding ‘s Mansour02)



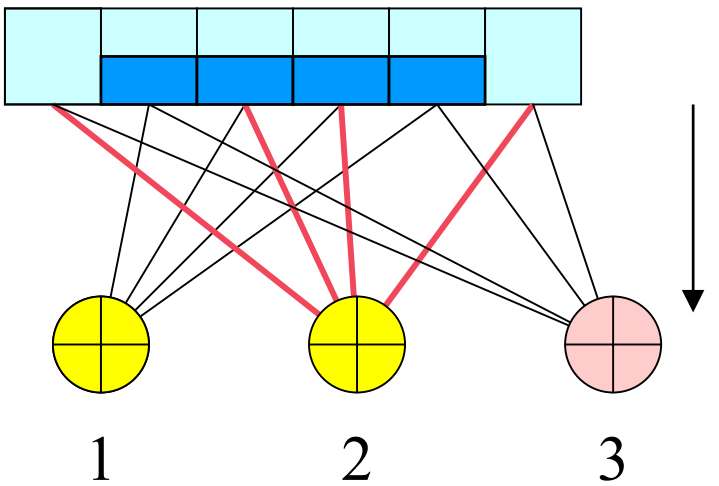
LDPC parity check matrix



Horizontal Shuffle Scheduling (Turbo Decoding 's Mansour02)

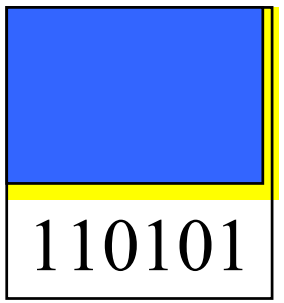


LDPC parity check matrix

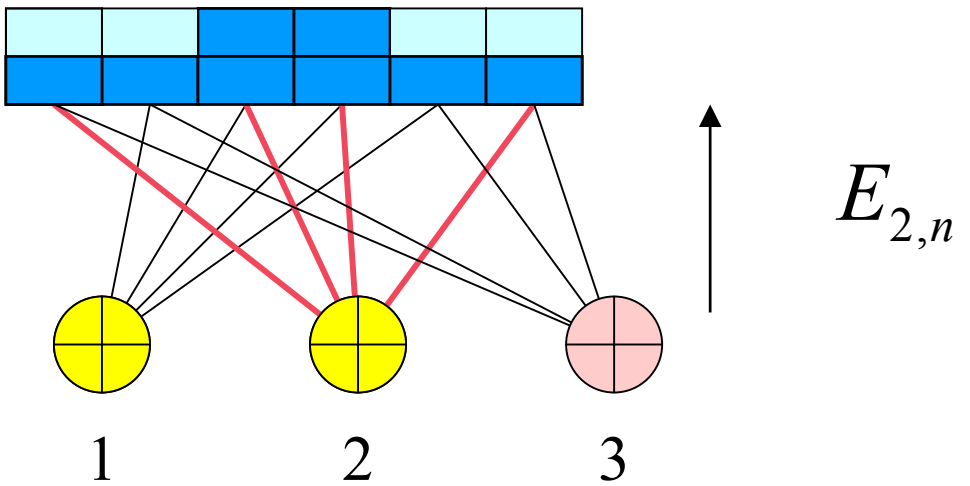


$T_{n,2}$

Horizontal Shuffle Scheduling (Turbo Decoding ‘s Mansour02)



LDPC parity check matrix

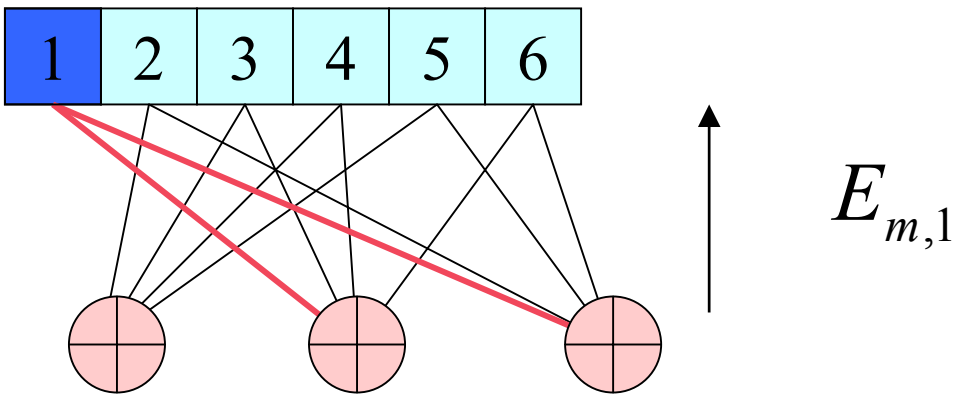


→ Fewer iterations required to converge

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

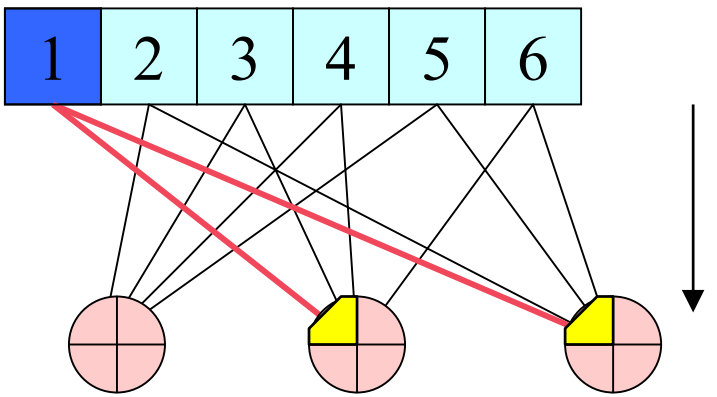
LDPC parity check matrix



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

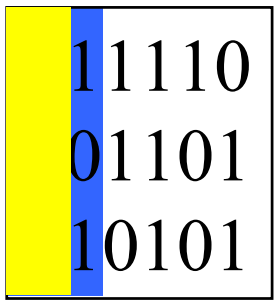
11110
01101
10101

LDPC parity check matrix

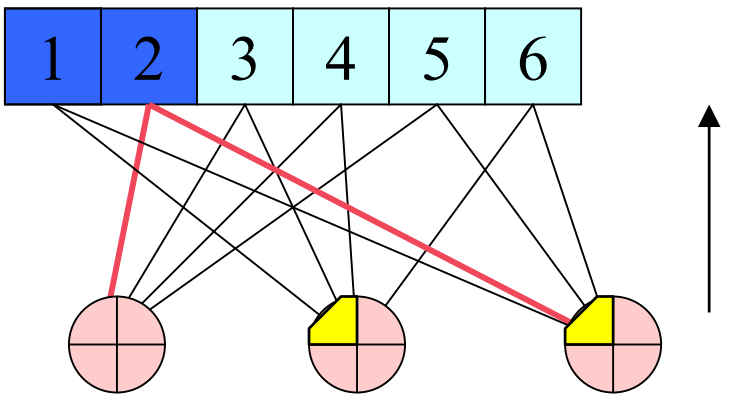


$T_{1,m}$

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

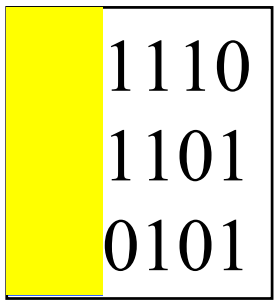


LDPC parity check matrix

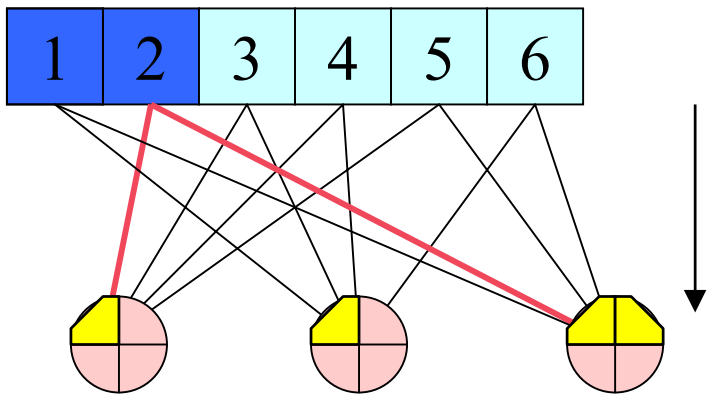


$E_{m,2}$

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



LDPC parity check matrix

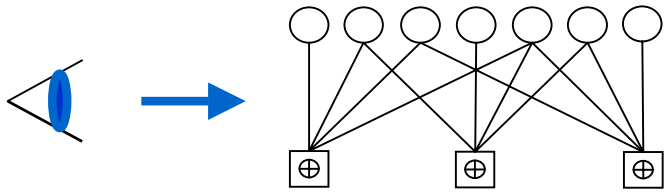


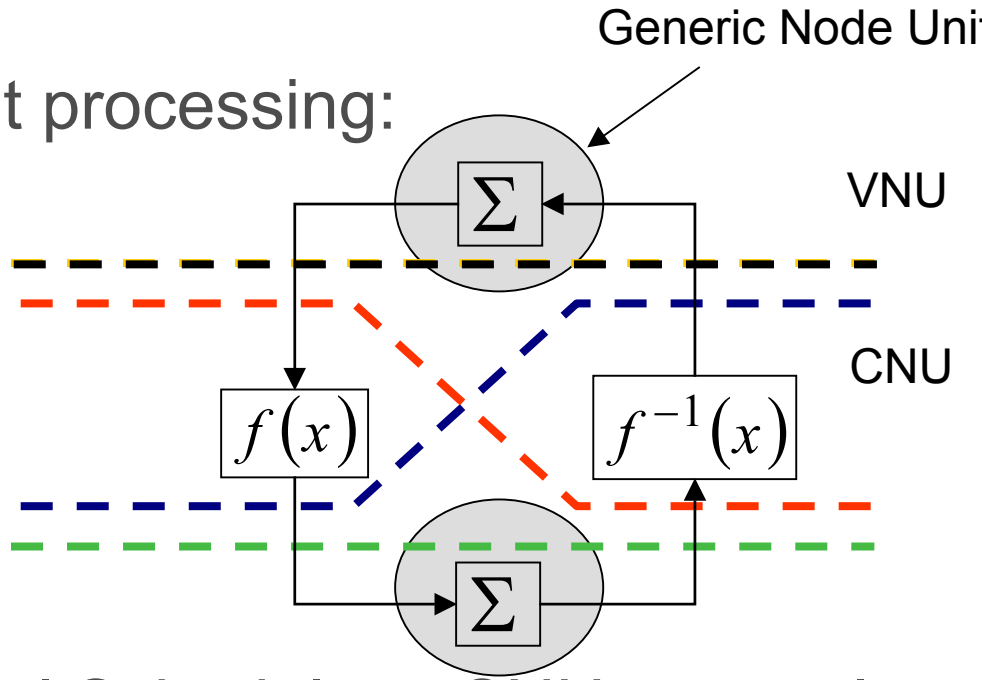
$T_{2,m}$

$\rightarrow$ and so on and so forth : faster convergence also

Implementation Issue for schedulings

- Classical Node Unit processing:

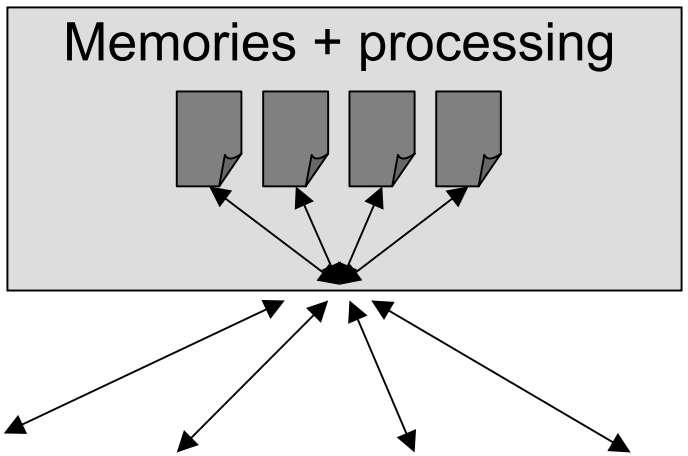




- Flooding / Horizontal Schedules : CNUs control and VNUs save information
- Vertical Schedule : Reverse !
- **Solution : use the symmetry**

Generic Node Units (exclude the f function)

Generic Node unit



- d inputs e_m and d outputs s_n ($d=j$ or k)
- Input / output law:

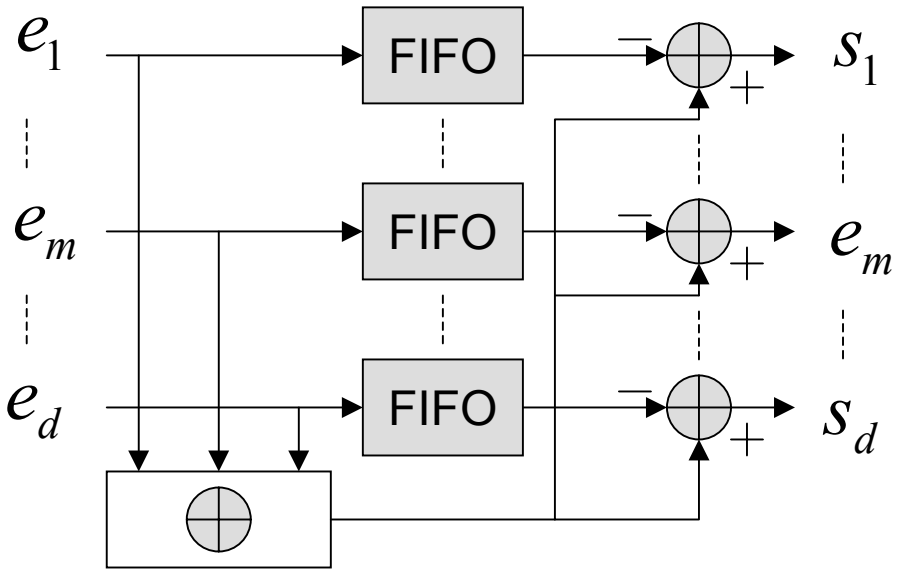
$$s_n = \sum_{m \neq n} e_m$$

- NB: replace Σ by $\times$ for sign processing

- How to manage inputs & outputs for node units?

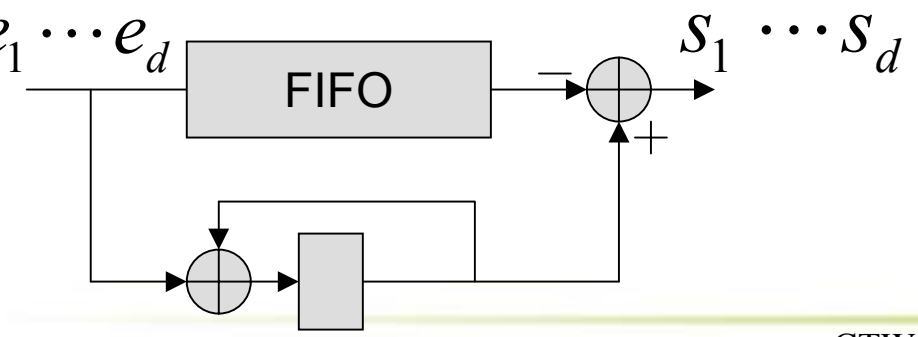
b)

Generic Node Unit implementations: a) Compact mode («no memories »)



- compact mode and parallel implementation

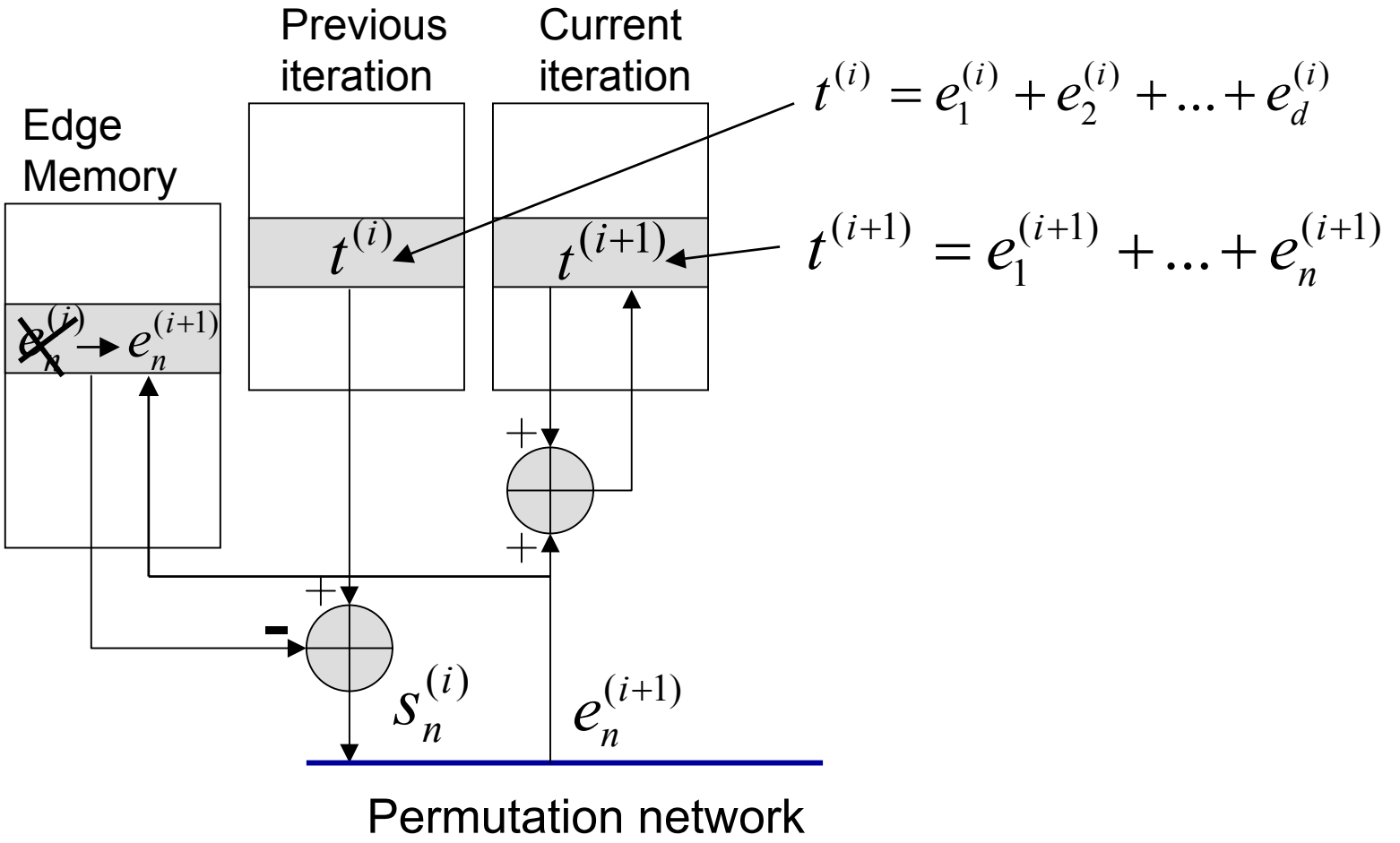
$$(\alpha \text{ or } \beta = 1 \rightarrow d_c = k \text{ or } d_v = j)$$



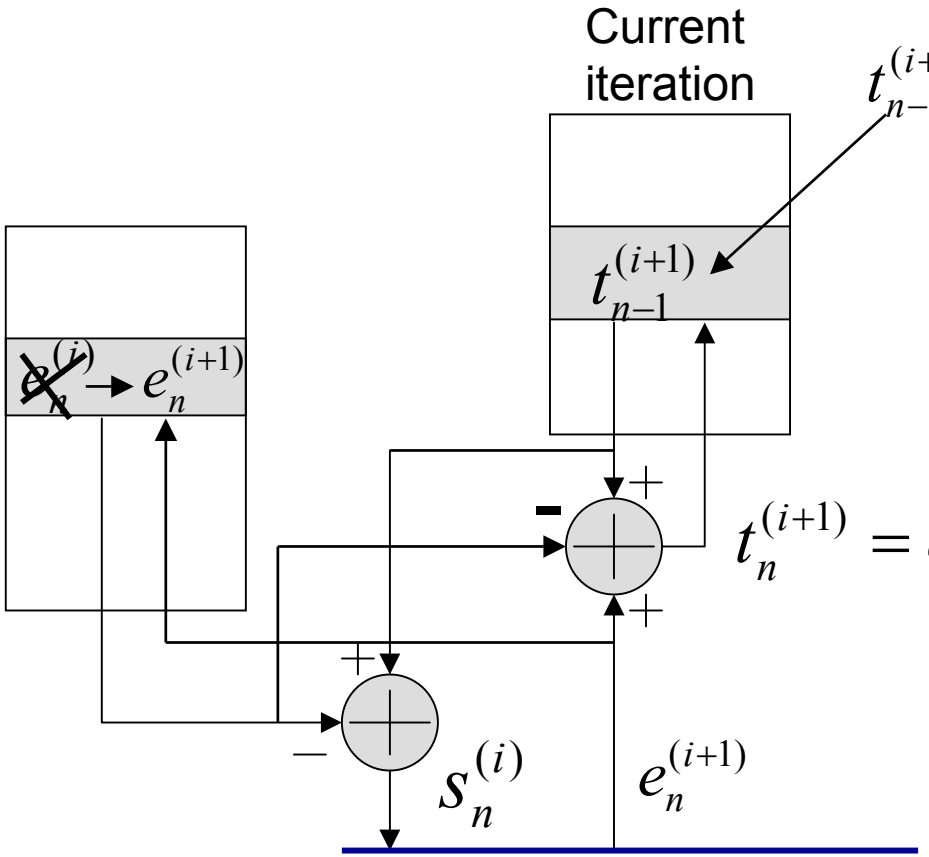
- compact mode and serial implementation

$$(\alpha = k \text{ or } \beta = j \rightarrow d_v = d_c = 1)$$

Generic Node Unit implementations:
b) Distributed Mode with slow update (Serial)



Generic operator implementation: c) Distributed Mode with fast update



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

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

$t^{(i+1)}$ is changing all along the iteration

Control mode combinations: span over the schedulings:

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

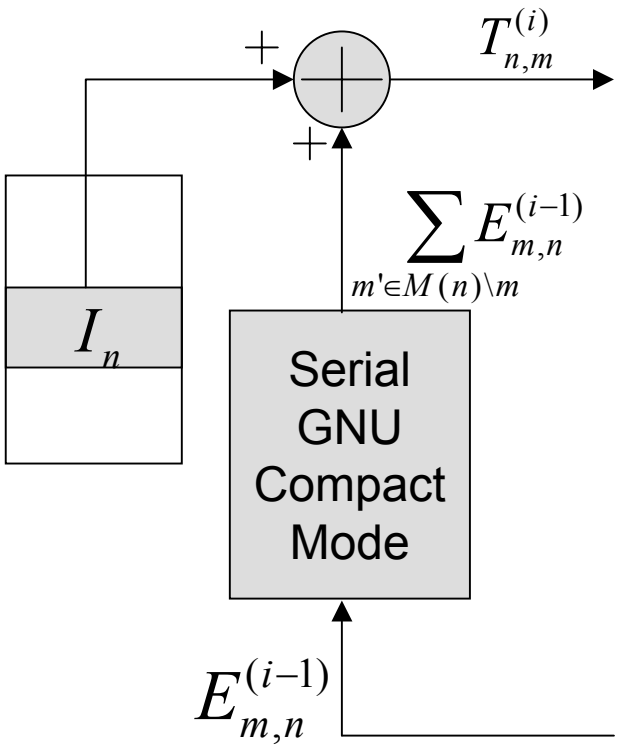
Example: Vertical Shuffle ad-hoc architecture (1)

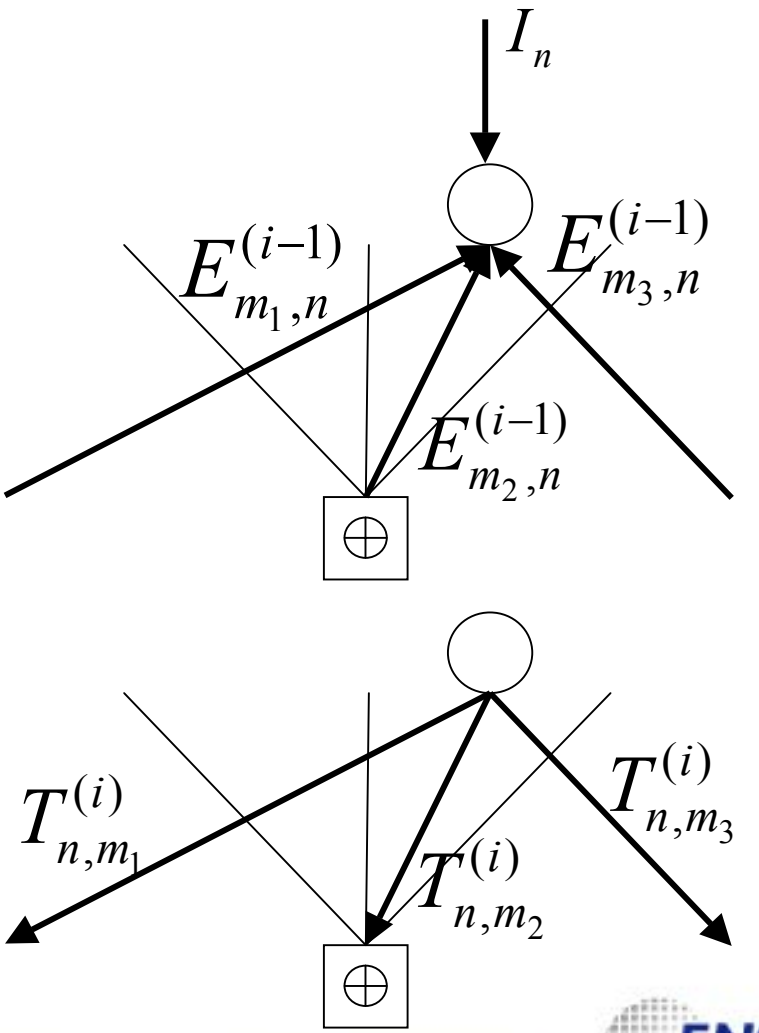
- Originally proposed as Shuffle-BP (Zhang & Fossorier, 2002)

- Combination of:
  - Compact Mode for VNU
  - Distributed Mode with fast update for CNU

Example: Vertical Shuffle ad-hoc architecture (2)

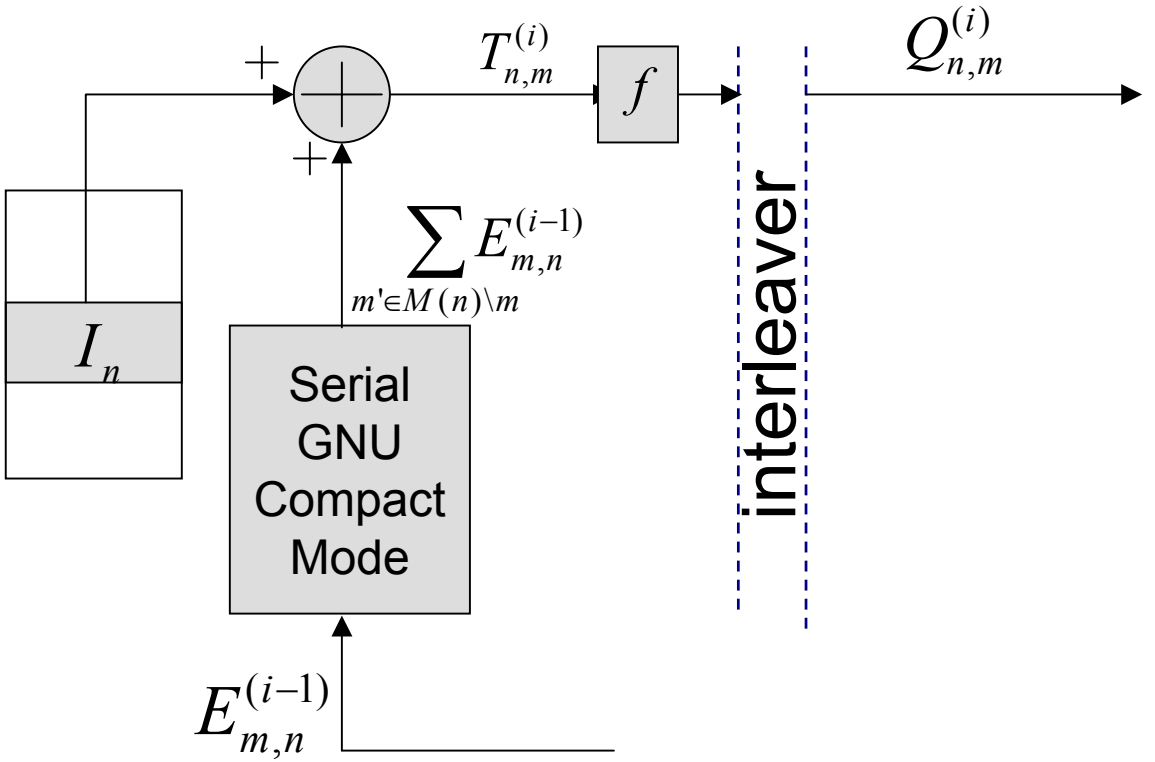
- Compact VNU (serial)





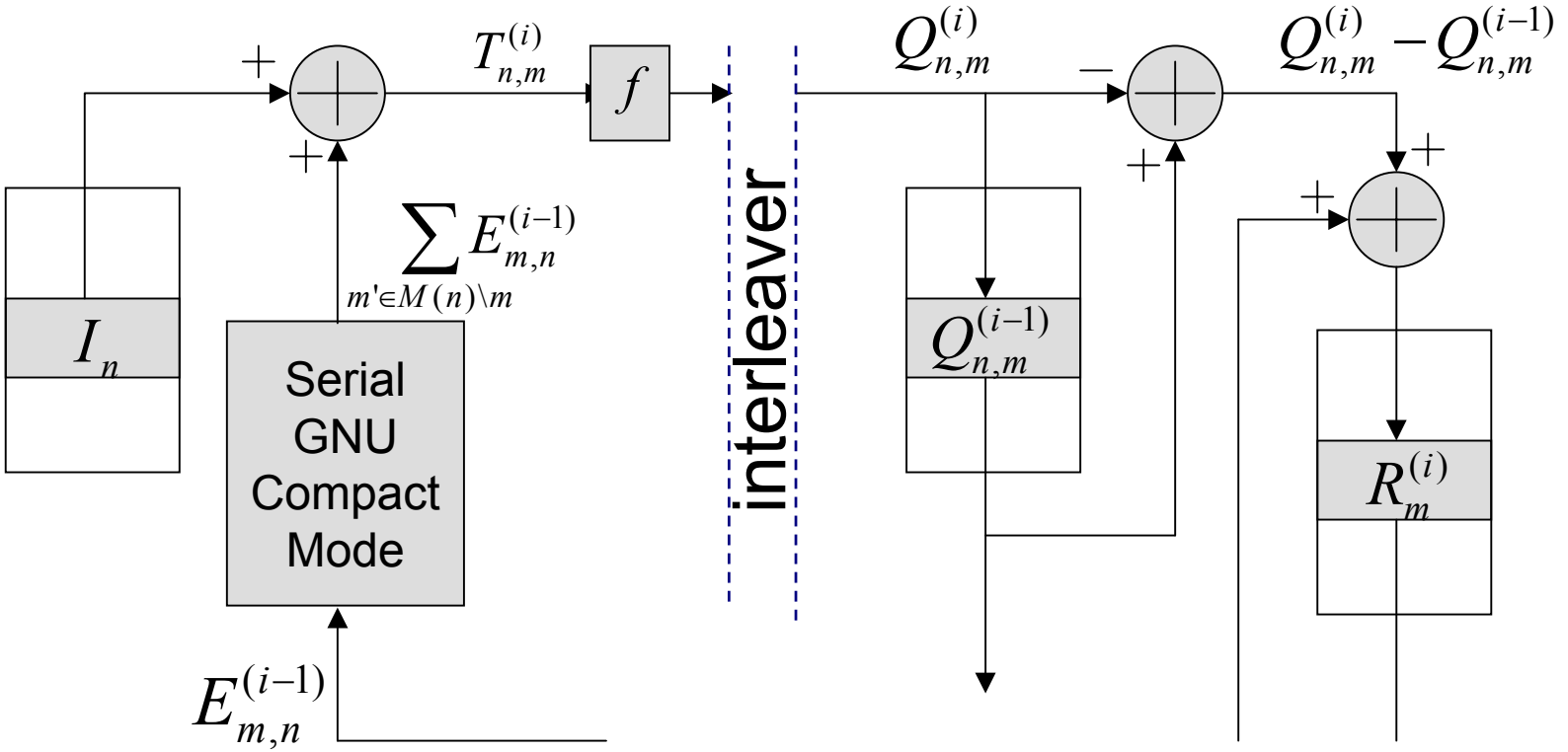
Example: Vertical Shuffle ad-hoc architecture (3)

- Go through the f functions and the interleaver



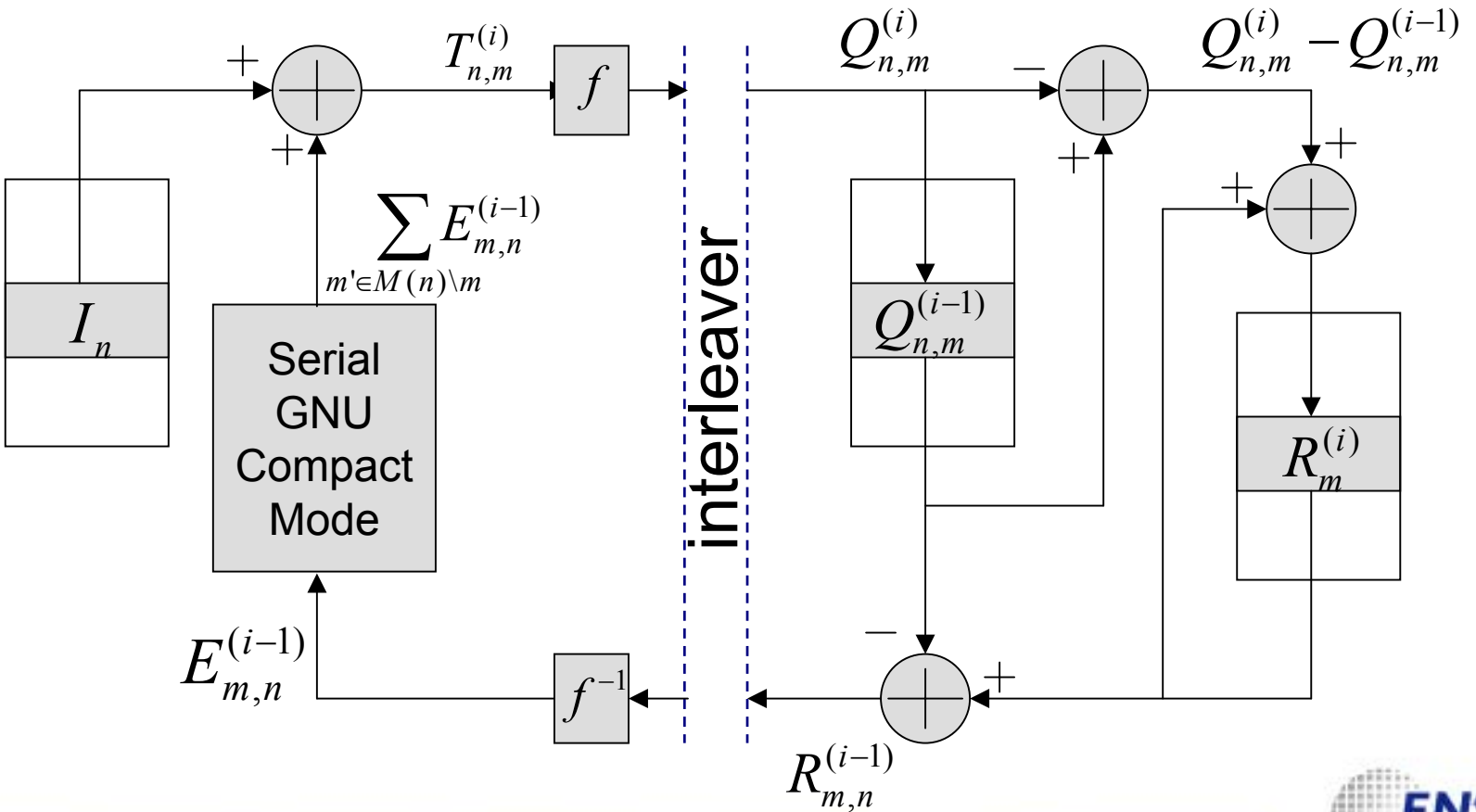
Example: Vertical Shuffle ad-hoc architecture (4)

- Save new message $Q_{n,m}$ and update the accumulation R_m (fast update)



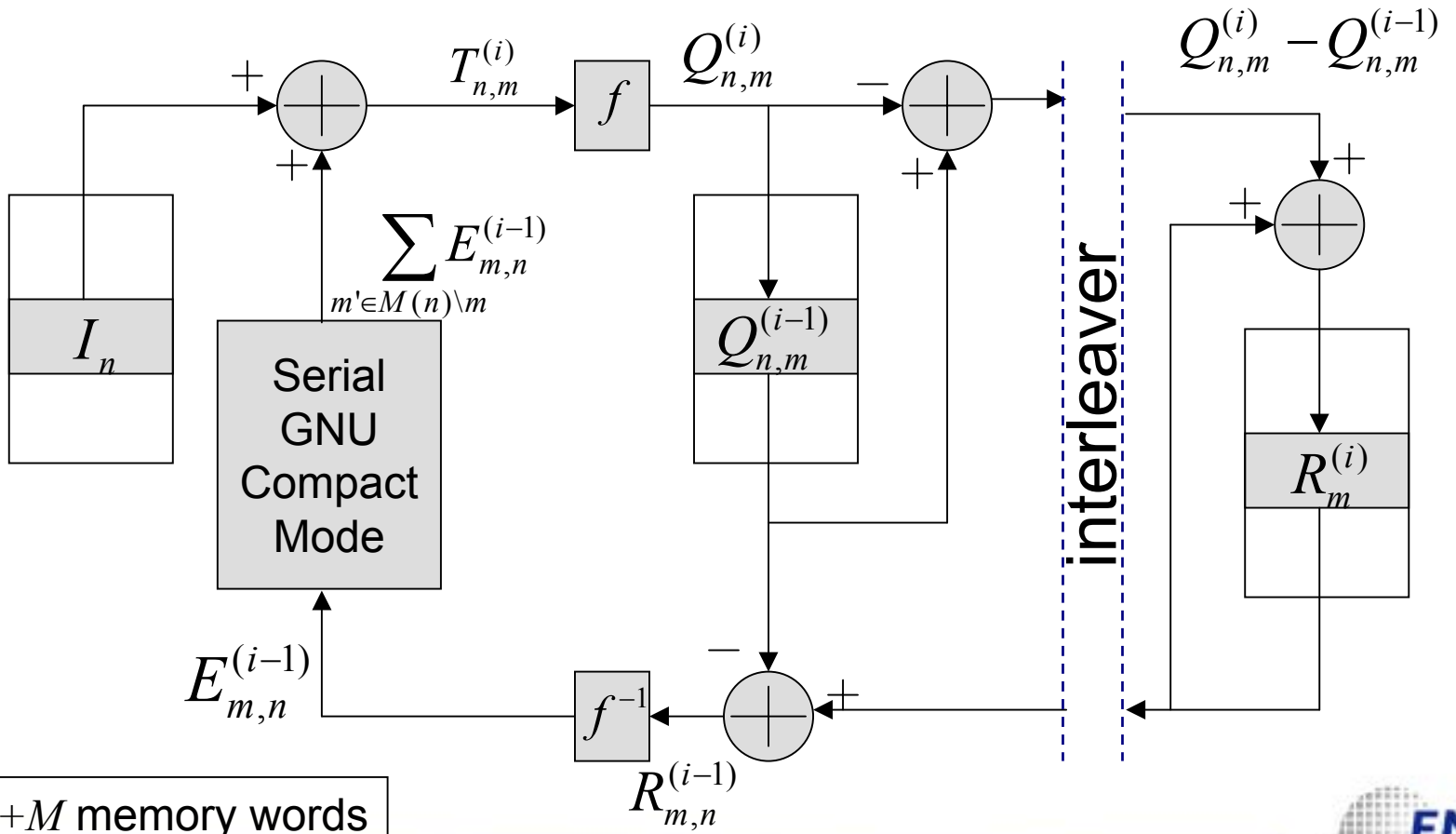
Example: Vertical Shuffle ad-hoc architecture (5)

- How did we get the Extrinsic information ?



Example: Vertical Shuffle ad-hoc architecture (6)

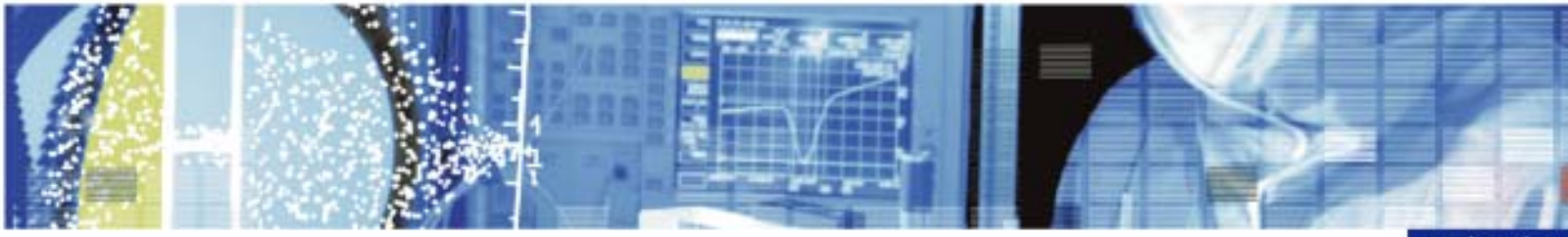
- NB : $Q_{n,m}$ messages can be moved to VNU:



$N+E+M$ memory words

Conclusion

- Investigations on high level design allowed us to propose a new and memory efficient architecture for the shuffled-BP schedule
- Channel information still available (turbo equalization for example)
- Characterization of LDPC decoders by some generic parameters (interconnection network position, node unit architecture, node unit control mode, parallelism parameters)



Thank you for your attention !